Ingénierie des modèles pour les applications environnementales

André Miralles

ACADEMIE DE MONTPELLIER

UNIVERSITE DE MONTPELLIER II

- SCIENCES ET TECHNIQUES DU LANGUEDOC -

THESE

Présentée à l'Université des Sciences et Techniques du Languedoc
Pour obtenir le diplôme de DOCTORAT

Spécialité :	Informatique
Formation Doctorale :	Informatique
École Doctorale :	Information, Structures et Systèmes

INGENIERIE DES MODELES POUR LES APPLICATIONS ENVIRONNEMENTALES

par

André MIRALLES

Soutenue le 11 Décembre 2006

Devant le jury composé de :

Dony Christophe	Professeur Université Montpellier II - LIRMM	Président
Bédard Yvan	Professeur Université Laval	Rapporteur
Papajorgji Petraq	Université de Floride Gainesville	Rapporteur
Schneider Michel	Professeur Université Blaise Pascal Clermont II	Rapporteur
Desfray Philippe	Vice Président Recherche et Développement - Société Softeam	Examinateur
Givone Pierrick	Directeur scientifique adjoint - Cemagref	Examinateur
Libourel Thérèse	Professeur Université Montpellier II - LIRMM	Directeur de thèse

À ma femme Chantal

À mes enfants Claire, Alicia et Bastien

À mi madre Gloria

A mi padre Francisco que
me ha enseñado su oficio de Ebanista

Realizado por Francisco Miralles y Andrés Miralles (1973)

Altar de bautizo

Remerciements

Je remercie Yvan Bédard, Petraq Papajorgji et Michel Schneider d'avoir assumé la lourde tâche d'être rapporteurs de ma thèse ainsi que Philippe Desfray, Christophe Dony, Pierrick Givone et Thérèse Libourel d'avoir accepté de faire partie de mon jury.

Je remercie l'ensemble des collègues de l'UMR 3S qui m'ont accueilli dans cette unité et qui m'ont permis de faire cette thèse dans des conditions privilégiées.

Je n'oublie pas Gilles Lechapt, Patrick Demangeot, Pierre Martinand et Pierre Maurel pour l'aide qu'ils m'ont apportée lors de mon arrivée à L'UMR 3S.

J'ai une pensée toute particulière pour tous les collègues de l'équipe COPAIN qui m'ont toujours apporté un soutien sans faille, sans oublier Frédéric Vigier avec qui j'ai eu des rapports très amicaux.

Je remercie Sylvie Blin de m'avoir « nourri » en bibliographie pendant ces années de recherche et François Pinet de m'avoir donné toute sa bibliographie géomatique.

Pacsé dans le même bureau depuis mon arrivée à l'UMR 3S, je n'oublierai pas les nombreuses discussions très enrichissantes que j'ai eues avec Jean-Stéphane Bailly aussi bien de nature personnelle, technique que scientifique.

Je remercie Philippe Monnier pour le regard de mathématicien qu'il a porté sur l'étude terminologique, pour les corrections apportées à mon approche et ses conseils éclairés.

Merci à Chantal Miralles, à Thérèse Libourel, à Ariane Vallet, à Jean-Pierre Baume, à François Pinet et à Alicia Miralles pour la fastidieuse tâche de relecture.

Toute ma reconnaissance à ma famille Chantal, Claire, Alicia et Bastien pour m'avoir soutenu et pour avoir eu beaucoup de patience pendant ces longues années de thèse.

Je remercie la Direction Scientifique du Cemagref d'avoir fait le choix de l'atelier de génie logiciel Objecteering sans lequel cette recherche n'aurait pas pu avoir lieu dans le délai où elle a été réalisée.

Enfin, je remercie Christophe Caradec et Olivier Saint-Léger du support technique de la société Softeam qui m'ont toujours aidé à résoudre les problèmes techniques que j'ai rencontrés au cours de ces années.

Sommaire Général

Liste des Figures

Liste des Tableaux

Introduction

L’introduction présente au paragraphe I le contexte de la recherche, au paragraphe II les problématiques pour lesquelles nous proposons une solution (cf. III).

Au paragraphe IV, nous fixons les objectifs de la recherche, au paragraphe V le cadre informatique dans lequel sera menée la recherche est précisé, au paragraphe VI le processus de recherche mis en œuvre est explicité et au paragraphe VII nous présentons la structure du mémoire.

Sommaire détaillé

I CONTEXTE

La présente recherche a été initiée au sein de l'Unité Mixte de Recherche *Structures et Systèmes Spatiaux* (UMR 3S) et s'est poursuivie au sein de l'Unité Mixte de Recherche *Territoires, Environnement, Télédétection et Information Spatiale* (UMR TETIS). La première unité regroupait des moyens techniques et humains du Cemagref et de l'ENGREF lesquels ont été fusionnés, en 2005, avec ceux du Cirad donnant ainsi naissance à l'UMR TETIS.

Les travaux de recherche effectués au sein de ces deux unités s'inscrivent dans l'Axe Thématique de Recherche *Méthodes pour la recherche sur les systèmes environnementaux* du Cemagref (ATR METHODO), axe rattaché à la Direction Scientifique du Cemagref. Comme son acronyme l'indique, cet ATR est fondamentalement méthodologique. Il est positionné transversalement par rapport aux autres ATR du Cemagref.

L'objectif principal de l'UMR TETIS est de produire des connaissances, des théories et des méthodologies permettant :

- *De maîtriser l'information spatiale sur les **systèmes environnementaux et territoriaux**.*
- *De mobiliser cette information pour caractériser et comprendre les dynamiques de ces systèmes, au service des démarches de connaissance, de développement, de gestion et de gouvernance (Kosuth, 2005).*

Outre la recherche, l'UMR TETIS a aussi une activité de formation et une autre d'appui technique aux ministères de la Recherche, de l'Agriculture et de l'Environnement. *Les activités de cette UMR sont organisées autour de 4 axes :*

- *Analyse spatiale des Espaces et territoires : Développer les méthodes d'analyse et de modélisation des structures spatiales et de la dynamique temporelle des systèmes agri-environnementaux et territoriaux.*
- *Acquisition de données et Télédétection : Développer les méthodes d'acquisition de données spatialisées sur ces systèmes par télédétection satellitale et capteurs embarqués, et les méthodes de construction d'informations élaborées à partir de ces données.*
- ***Ingénierie des Systèmes d'Information** : Développer les méthodes de conception et de développement de systèmes d'information sur des problématiques de gestion environnementale et territoriale, sur la base d'une identification et analyse des besoins des acteurs. Développer les méthodes de gestion de ces Systèmes d'Information*
- *Place du partage de l'information dans le Développement territorial : Développer les concepts et méthodes pour conduire et formaliser des diagnostics de développement territorial ou de gestion environnementale, définir les modes d'usage concerté de ces informations, accompagner et évaluer l'usage de ces informations et son impact sur les acteurs et les modes de gestion et de gouvernance (Kosuth, 2005).*

Le travail de recherche réalisé au cours de la présente thèse s'inscrit principalement dans l'axe *Ingénierie des Systèmes d'Information*. Cependant au vu des résultats obtenus, ce travail s'inscrit pour partie dans l'axe de recherche *Analyse spatiale des Espaces et territoires*.

Les activités de l'UMR TETIS portant essentiellement sur les systèmes environnementaux et territoriaux, l'*acquisition*, la ***modélisation*** et le *traitement* de l'*Information Géographique* sont au cœur des travaux de recherche de cette UMR et donc de la présente thèse.

Dans les deux dernières décennies et dans les années à venir, les systèmes d'acquisition (satellite, lidar, etc.) et de positionnement (GPS, GALILEO) ont ou vont bénéficier d'avancées technologiques qui permettront de disposer d'informations géographiques plus abondantes et plus précises à un moindre coût. Or, l'acquisition et le traitement d'une information abondante ne peut s'envisager qu'en mettant en œuvre des moyens informatiques puissants.

La présente recherche est à la conjonction de l'*Information Géographie* et de l'*Informatique*, conjonction constituant le domaine ***Géomatique*** où sont mis en œuvre les ***Systèmes d'Information Géographique***.

Le volume d'information disponible ayant fortement progressé, l'UMR TETIS a acquis depuis de nombreuses années la maîtrise du traitement de l'information produite par les systèmes d'acquisition. Souvent, le traitement de l'information est effectué à l'aide de logiciels de traitement d'images (Imagine, eCognition, etc.)

ou de logiciels SIG (ArcInfo, SPRING, etc.). Ces deniers gèrent des bases de données spatiales ou spatio-temporelles qui sont de plus en plus volumineuses.

Malheureusement, ces outils commerciaux ne répondent pas toujours aux besoins des scientifiques de l'UMR TETIS qui, dans le cadre de leurs recherches ont parfois à effectuer des traitements spécifiques. Afin de réaliser ces derniers, les scientifiques sont conduits à développer ou à faire développer des logiciels qui répondent à leurs besoins immédiats. Ils privilégient alors souvent l'efficacité et la rapidité du développement au détriment de la ***modélisation*** de l'application et du ***capital de travail et de connaissances*** que représente le code informatique produit.

Par ailleurs, dans sa mission d'appui technique, l'UMR TETIS est amenée à intervenir dans l'analyse et parfois dans la maîtrise d'œuvre et la maîtrise d'ouvrage de *Systèmes d'Information Géographique* développés par les ministères de tutelle ou les collectivités territoriales. Les projets de *Système d'Information à Référence Spatiale sur les ouvrages de protection contre les crues* (SIRS Digues) et de *Système Informatisé pour la Gestion des Épandages de Matières Organiques* (SIGEMO) s'inscrivent dans cette catégorie.

La modélisation des applications informatiques et des *Systèmes d'Information Géographique* étant une faiblesse de l'unité, il était nécessaire d'acquérir une compétence sur ces deux points.

La présente recherche porte sur la ***conception*** et le ***développement*** des applications informatiques et des ***Systèmes d'Information Géographique*** et en particulier sur la ***modélisation*** de ces applications.

La plupart des travaux de recherche de l'UMR 3S et ensuite ceux de l'UMR TETIS s'inscrivent dans des collaborations ou des projets avec des partenaires universitaires (LIRMM, LIMOS, Université de Liège, Université Laval, etc.), des partenaires industriels (CNES, Spot Image, GEOSYS, SWORD Group, etc.) et/ou des partenaires institutionnels (ministères, collectivités locales, CNIG, IGN, Chambres d'Agriculture, instituts techniques : ONF, ONIC, etc.). Beaucoup de ces partenaires ayant une activité relevant du domaine géomatique, les travaux de cette recherche les intéressent à différents degrés : recherches, enseignements et/ou appuis techniques.

Des relations privilégiées ont été initiées par Thérèse Libourel avec le LIRMM et l'école doctorale *Information, Structures, Systèmes* de l'Université Montpellier II. La recherche présentée ici s'inscrit dans le cadre de cette collaboration laquelle va se poursuivre par une nouvelle recherche menée en commun avec l'Université Laval. La problématique de cette nouvelle recherche est présentée dans les perspectives (cf. Perspectives-IV).

En outre, la collaboration avec ces deux instances montpelliéraines devrait s'intensifier dans les années à venir dans la mesure où l'UMR TETIS a effectué une demande de reconnaissance auprès du Ministère de la Recherche (quadriennal 2007-2010) et de rattachement aux instances de l'UM II et plus particulièrement à l'école doctorale *Information, Structures, Systèmes*.

II PROBLEMATIQUE

II.1 Améliorer la capitalisation des connaissances

En termes de développement d'applications et de *Systèmes d'Information Géographique*, le Cemagref est confronté à deux grandes catégories de développements :

- ⇨ La première est liée à sa mission d'appui technique aux ministères et aux collectivités locales. Il est souvent amené à intervenir dans des projets de *Systèmes d'Information Géographique* pour le compte de certains ministères et des collectivités territoriales.
- ⇨ La seconde comprend les développements internes effectués à des fins de recherche. Alors, le développement repose souvent sur une seule et même personne.

Pour les applications relevant de la première catégorie, la demande émane d'acteurs qui peuvent être des élus, des présidents d'association, des hauts fonctionnaires, etc. Cette catégorie d'acteurs se caractérise par des emplois du temps très chargés. Il n'est pas rare que les acteurs de cette catégorie interagissent directement avec le système à modéliser et qu'ils aient une très bonne connaissance du système. Ce sont des experts incontournables dont il faut recueillir leurs besoins mais aussi la connaissance qu'ils ont du système.

Le problème est que leur planning chargé les rend difficilement ***mobilisables simultanément de façon répétitive et soutenue*** pour réaliser le modèle de l'application comme le recommande les méthodes récentes de conduite de projet (Processus Unifié, eXtreme Programming entre autres).

Pour les applications de la seconde catégorie, la méthode de conduite de projet appliquée est la plupart du temps minimaliste et elle est rarement partagée avec les autres membres de l'équipe. Il est donc difficile dans ce contexte de garantir la reproductibilité du développement ce qui pose encore des problèmes de capitalisation. Cette dernière devient encore plus difficile à assurer lorsque le développement est réalisé par du personnel non permanent (vacataires, stagiaires, thésards, etc.).

Ce personnel non permanent intervient souvent pour réaliser rapidement (avec des contraintes de délais fortes) une application afin de tester et de valider les concepts d'une recherche sans trop se soucier ni de la qualité du développement ni de la pérennité de l'application.

De plus, il n'est pas rare d'une part, que le chercheur demandeur de l'application décrive les concepts et les fonctionnalités soit verbalement soit à l'aide de schémas « griffonnés » et, d'autre part, que le chercheur ne soit pas en mesure de suivre le code produit.

Après le départ du personnel non permanent, l'application devient rapidement une boîte noire car rarement une documentation soignée est réalisée. Dans ce contexte, l'évolution de l'application est difficile à assurer et cette difficulté s'accroît au fil du temps car les directives données par le chercheur s'estompent progressivement de sa mémoire. Dans ce contexte, le code est de fait le principal produit du développement.

La capitalisation des connaissances n'est pas une problématique spécifique au Cemagref, de nombreux organismes de recherche (Cirad, LIRMM…), d'entreprises etc. sont confrontés aux mêmes difficultés.

Le problème auquel sont confrontés les concepteurs est d'***améliorer la capitalisation des connaissances*** mobilisées pour réaliser une application.

II.2 Augmenter l'efficacité de capture des connaissances

Ivar Jacobson définit le ***processus de développement d'une application informatique*** comme étant l'ensemble des activités nécessaires pour transformer les spécifications et les exigences des acteurs du domaine étudié en une application informatique (Jacobson *et al.*, 1999). Cette définition est schématisée par la figure 1 :

Figure 1 Un processus de développement d'une application informatique (Jacobson *et al.*, 1999).

Ces mêmes auteurs ont formalisé les différents « ingrédients » intervenant dans le processus de développement d'une application informatique. Ce modèle est connu sous le nom de *4 Ps* : le *Personnel*[1], le *Projet*, le *Produit* et le *Processus* de développement d'une application (cf. figure 2).

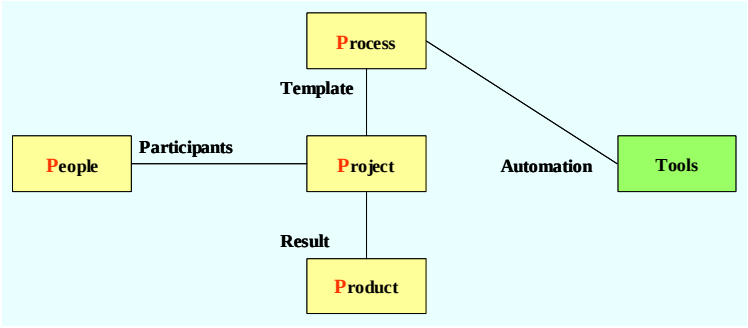


Figure 2 Les 4 Ps du processus de développement d'une application (Jacobson *et al.*, 1999).

[1] Acteur dans notre contexte.

Ce modèle exprime que le ***résultat*** d'un ***projet*** est un ***produit*** qui nécessite des ***participants***[1] pour être mené à bien. La réalisation du projet s'effectue suivant un ***guide***[2] qui définit, organise et explique les étapes successives du ***processus*** de développement. Pour mettre en œuvre le processus de développement, il faut des ***outils*** de planification du projet, d'expression des besoins, de modélisation, etc. qui automatisent certaines activités.

Cette description rapide du processus de développement brosse rapidement la diversité des sujets et des problèmes que doit aborder et traiter un chef de projet en charge du développement d'une application. Afin d'illustrer la complexité inhérente d'un développement, (Muller *et al.*, 2000) utilisent deux métaphores reportées ici in extenso :

- La première est relative à la gestion du projet : *le développement d'un logiciel peut être vu comme la traversée d'un océan en bateau. Le jour du départ est connu, le jour d'arrivée l'est moins ; en cours de route, il faudra affronter des tempêtes et réparer des avaries*.
- La seconde concerne la pluridisciplinarité des développeurs. Il écrit : *si l'informaticien produisait des meubles, il commencerait par planter des glands, débiterait les arbres en planches, creuserait le sol à la recherche de minerai de fer, fabriquerait une forge pour faire ses clous et finirait par assembler le tout pour obtenir un meuble. […] Un mode de développement qui repose sur les épaules de quelques programmeurs héroïques, de gourous et autres magiciens du logiciel, ne constitue pas une pratique industrielle pérenne et reproductible*.

(Muller *et al.*, 2000) n'utilisent pas ces deux métaphores pour le plaisir. Elles illustrent bien le challenge auquel sont confrontés le chef de projet et les développeurs qui doivent mener à terme le développement d'une application. Ce challenge est confirmé par les statistiques établies par (The Standish Group, 2001) qui montrent que le développement d'une application est une véritable gageure puisque :

- 23%[3] ont un risque d'échec.
- 49% ont un risque de dérapage.
- 28% seulement ont une chance de succès.

Ces chiffres sont issus d'une étude portant sur les développements réalisés aux États-Unis en 2000. Il est à noter que les auteurs de cette étude ne distinguent pas, dans les 28 %, les développements visant à créer une nouvelle application de ceux destinés à faire évoluer une application existante. La chance de succès de la seconde typologie de développement est bien plus élevée que celle d'une nouvelle application.

Pour mieux maîtriser cette complexité, le chef de projet a recours à une méthode de conduite de projet qui permet de rendre le développement d'une application informatique plus rationnel et plus reproductible. De plus, l'adoption d'une méthode de conduite de projet contribue à *augmenter le niveau de satisfaction des clients* et des acteurs *tout en rendant le travail de développement plus facile* (Bénard, 2001).

Le but d'une méthode de conduite de projet est de rationnaliser le développement d'une application et surtout de le rendre reproductible. Il résulte de ce constat que toute amélioration de la méthode de conduite de projet va favoriser les chances de succès du développement d'une application.

Or, il apparaît que les méthodes de conduite de projet récentes ont toutes pour objet principal d'augmenter la participation des acteurs du domaine. Elles s'appuient sur le constat que ces derniers sont, chacun à leur niveau, détenteurs d'une brique de connaissance sur le système (cf. Chapitre 2-II) mais aussi qu'ils seront les bénéficiaires de l'application finale et qu'à ce titre l'application doit répondre à leurs besoins et à leurs exigences.

La capture des concepts et des spécifications du système et l'expression des exigences et des besoins par les acteurs sont donc des activités incontournables agissant directement sur la qualité finale de l'application et sur la satisfaction des acteurs.

Le problème auquel sont confrontés les concepteurs est d'***augmenter l'efficacité de la capture*** des concepts et des spécifications ainsi que celles des exigences et des besoins.

[2] Méthode dans notre contexte.
[3] Chiffres 2000.

III SOLUTION PROPOSEE

III.1 Améliorer la capitalisation des connaissances

Améliorer la capitalisation des connaissances peut prendre plusieurs formes. L'une d'elles et non la moindre consiste à assurer une meilleure pérennité du personnel impliqué dans le développement des applications. Cette forme de capitalisation n'est pas l'objet de la recherche car elle ne présente aucun aspect « technique ».

Nous nous intéresserons ici à la capitalisation de connaissances contenues dans les modèles ou les codes. Cette typologie de capitalisation est une préoccupation qui date depuis fort long mais, ces dernières années, elle a pris de l'ampleur avec l'idée préconisée par l'OMG de séparer des spécifications thématiques et d'implémentation au sein de deux types de modèles. Cette idée prône de faire un premier modèle, que nous qualifierons pour l'instant de thématique, contenant tous les concepts thématiques et de réaliser un second modèle, que nous qualifierons pour l'instant d'implémentation, où les concepts thématiques sont complétés avec les concepts et les spécifications d'implémentation liées à un langage de programmation ou à une plate-forme matérielle. C'est ce second modèle qui va permettre de générer le binaire de l'application.

La vision de l'OMG est que le modèle thématique et le modèle d'implémentation ne sont pas indépendants et qu'il est possible d'établir un couplage entre eux au moyens de transformations de modèles. Selon l'OMG, ces dernières doivent permettre de transférer les concepts contenus dans un modèle thématique vers le modèle d'implémentation. Cette approche est connue sous le nom de *Model Driven Architecture*[4] (Miller *et al.*, 2001, 2003). Actuellement, une part importante des moyens de recherche en informatique est consacrée à l'étude de ces transformations de modèles.

Un des avantages majeurs de l'approche *Model Driven Architecture* (MDA) est qu'un modèle ne contenant que des concepts thématiques peut être remobilisé à tout moment au cours du développement d'une application mais aussi au cours de cycle de vie de l'application.

Au cours du développement, les concepts thématiques étant indépendants du type d'application visé (application monoposte, application répartie avec son système d'information, etc.), ils peuvent être mobilisés dans plusieurs types d'implémentation.

Au cours du cycle de vie de l'application, ces concepts thématiques constituent le capital de base résultant de l'analyse du système qui va permettre de réaliser une nouvelle version de l'application. À cette occasion, de nouvelles technologies informatiques peuvent être mises en œuvre sans problème majeur car elles n'affectent pas le modèle thématique.

Nous envisageons dans la présente recherche de multiplier le nombre de modèles afin d'avoir une capitalisation plus fine des connaissances qui ne soit pas liée qu'aux seuls aspects techniques comme le préconise les informaticiens dans l'utilisation de l'approche *Model Driven Architecture* et les exemples qu'ils utilisent pour présenter ou enseigner cette approche.

Actuellement, il n'existe pas d'approche fondée sur ce principe car la principale difficulté est de maintenir en cohérence plusieurs modèles. En effet, s'ils sont séparés, le coût pour maintenir en cohérence plusieurs modèles séparés est exorbitant et irréalisable dans un secteur économique hautement concurrentiel comme celui de l'informatique. Cela n'est possible que si le maintien en cohérence est totalement automatisé. C'est l'objet de la présente recherche.

III.2 Augmenter l'efficacité de capture des connaissances[5]

Il est réputé vrai que le prototypage d'application est une pratique efficace pour valider les besoins des utilisateurs lors du développement de systèmes (Champoux, 1997 ; Connell et al., 1989 ; McConnell, 1998 ; Paquette, 1990 ; Turban et al., 2001 ; Voyer, 2000). De plus, des expériences passées dans le cadre de projets […] démontrent que le prototypage […] est une approche relativement bien adaptée à ce contexte. […] Pour cela, nous proposons de concevoir et d'exploiter un outil supportant le maquettage d'application […]. Cet outil vise à structurer le […] prototypage *d'application dès le début d'un projet de développement.*

[4] MDA

[5] La proximité intellectuelle et les travaux menés par Louis-Etienne Guimond (Guimond, 2005) est telle que nous avons reporté in extenso la solution qu'il préconise car elle correspond exactement à notre vision. Le texte en italique est écrit par Louis-Etienne Guimond, les quelques modifications effectuées pour adapter ce texte à notre contexte sont en caractères droits. Plusieurs suppressions permettent de généraliser le texte de Louis-Etienne Guimond écrit pour la technologie SOLAP.

III.2.1 *Premier apport*

Premièrement, […] le maquettage peut agir en complément du prototypage en début de projet. Le maquettage s'apparente au prototypage jetable (de l'anglais throwaway prototyping) ; la différence étant que le prototypage jetable utilise des données réelles alors que le maquettage n'y fait pas appel (Connell et al., 1989).

Élaborer une maquette consiste à produire une représentation sommaire, simulée et statique de ce que pourrait être un système final à l'aide d'exemples visuels des concepts et des fonctionnalités possibles dans le système (Connell et al., 1989).

Sans faire appel à des données réelles, ces exemples peuvent tout de même refléter la réalité et les données des utilisateurs. De plus, le maquettage présente les avantages d'être plus rapide et moins dispendieux à réaliser que le prototypage. Il est toutefois moins exhaustif que peut l'être un prototype dans sa représentation de la solution finale.

En Annexe VI *de ce mémoire se trouve une discussion plus approfondie sur les fondements théoriques du maquettage et des différentes formes de prototypage […].*

III.2.2 *Deuxième apport*

Deuxièmement, nous croyons que le […] prototypage *devrait être appliqué dès la première discussion avec les utilisateurs au début d'un projet. En leur présentant […]* un prototype *d'application […], même générale, leur apprentissage des concepts et des possibilités […] se trouverait accéléré par rapport à l'application d'une méthode plus traditionnelle (cf. Figure 3). Cela permettrait aussi de recueillir un feed-back des utilisateurs plus tôt dans l'analyse de leurs besoins.*

Ces deux éléments éliminent dès le départ le risque qu'il y ait confusion chez les utilisateurs concernant le type de solution qui leur est proposée. Le […] prototypage *est tout indiqué pour explorer les attentes préliminaires des utilisateurs (Connell et al., 1989). De plus, le fait de l'appliquer tôt dans le projet est en accord avec le principe fondamental des méthodes AGILE concernant le changement.*

Dans le manifesto des méthodes AGILE, il est précisé qu'il est plus efficace d'accueillir le changement que de tenter de le prévenir (Fowler et al., 2001). Cependant, les méthodes AGILE proposent des mécanismes pour « accueillir » le changement en cours de développement.

Nous proposons plutôt de « provoquer » le changement le plus tôt possible, alors que ses répercussions sont beaucoup plus faciles à prendre en compte durant un effort de […] prototypage*. Cela améliore le rendement de l'équipe de développement qui démarre dès le début avec une meilleure connaissance des besoins.*

Nous visons ainsi à rendre les méthodes de développement plus efficaces en diminuant l'écart initial entre la connaissance des besoins et les besoins totaux qui devront être comblés.

III.2.3 *Troisième apport*

Troisièmement, nous proposons un outil de […] prototypage *spécialement adapté […]* à la capture des concepts*. Il existe déjà diverses technologies pour réaliser un prototype lors du développement de système d'aide à la décision, mais aucune ne nous semble appropriée […]* à notre besoin de capitalisation*. Nous croyons donc qu'il y a un réel besoin d'un outil conçu pour cette tâche.*

[…].

III.2.4 *Discussion*

À la différence avec Louis-Étienne Guimond, nous pensons que le prototypage est préférable au maquettage dans la mesure où il est organisé en complément du développement afin de constituer un ***continuum de développement***. Il permet alors non plus de « provoquer » le changement mais d'instrumenter ce changement.

Dans cette perspective, il faut coordonner la complémentarité du prototype et de l'application et définir clairement les objectifs et les conditions d'utilisation de chacun d'eux.

III.3 Postulats[5]

La solution proposée est fondée sur deux postulats : le premier est relatif à l'amélioration de la capitalisation des connaissances et le second concerne l'augmentation de l'efficacité de capture des connaissances.

Le premier postulat que nous posons est qu'il existe une transformation générique qui permet de coupler deux ou plusieurs modèles entre eux et ce quelque soit le type de modèle. Ce postulat repose sur le constat qu'un

concept thématique saisi lors de l'analyse dans un modèle thématique doit aussi exister dans le modèle d'implémentation.

Le […] second *postulat veut que l'application du […]* prototypage *accélère l'apprentissage […] par les utilisateurs, ce qui améliore leur compréhension et facilite la découverte des besoins latents et inconnus soulevés par l'apparition de nouvelles possibilités.*

D'après Louis-Étienne Guimond, ce second postulat repose sur un postulat plus général qui est *que la zone de connaissances communes entre les utilisateurs et l'équipe de développement s'accroisse grâce au […]* prototypage. *Il s'agit d'une application du modèle de communication de (Schramm, 1954) selon lequel il doit y avoir une compréhension adéquate entre la source et la destination pour qu'une communication puisse réellement s'établir. Cette compréhension s'illustre par une zone de connaissances communes, appelée commonness. […]*.

IV OBJECTIF DE LA RECHERCHE

Le domaine d'application de la présente recherche est induit par les activités de recherche, de formation et d'appui technique de l'UMR TETIS sur les systèmes environnementaux et territoriaux. Il s'agit du domaine géomatique et plus particulièrement des *Systèmes d'Information Géographique*.

Le premier objectif est issu du domaine d'application de l'UMR TETIS et du constat que les scientifiques de cette UMR ne disposaient pas de moyen de modélisation approprié au développement d'applications géomatiques.

Le premier objectif visé est de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique*.

Le but du deuxième objectif est de résoudre la problématique posée par la capitalisation de connaissances, besoin inhérent au développement d'une application.

Le deuxième objectif visé est d'assurer la *capitalisation des connaissances* mobilisées au cours du développement d'une application et contenues dans les modèles et les codes informatiques.

Le troisième objectif est lié à l'augmentation de l'efficacité de capture des connaissances. Étant donné que, dans un processus de développement, la phase d'analyse est consacrée à la capture de concepts et des spécifications, nous avons donc recherché à accroître l'efficacité de capture au cours de cette phase. Pour ce faire, nous avons adopté la technique du prototypage.

Le troisième objectif visé est de définir et de concevoir un *processus de développement permettant le prototypage rapide en séance d'analyse*.

Ce troisième objectif répond au problème posé par les acteurs auxquels nous sommes confrontés qui sont difficilement mobilisables simultanément de façon répétitive et soutenue pour réaliser le modèle de l'application.

L'objectif général de la recherche peut se résumer ainsi :

Réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique* adapté à un *processus de développement permettant le prototypage rapide en séance d'analyse* et assurant la *capitalisation des connaissances*.

V CADRE INFORMATIQUE DE LA RECHERCHE

Dans la perspective de modélisation des *Systèmes d'Information Géographique*, un premier choix a été de préférer le ***formalisme objet*** au formalisme Entité/Relation car chacun s'accord à trouver ce formalisme plus naturel pour décrire un domaine.

Un second choix a consisté à adopter le langage ***Unified Modeling Language***[6] (cf. Annexe I) qui permet de *spécifier*, de *visualiser*, de *construire* et de *documenter* les concepts d'un système avec pour objectif final de produire une application informatique (OMG, 2002, 2003). Ce langage a été standardisé en 1997 par l'*Object Management Group* (OMG), consortium international réunissant des utilisateurs, des chercheurs, des développeurs et des industriels. Le langage UML a été conçu pour ***faciliter la communication entre les acteurs du domaine et les concepteurs*** et pour ***satisfaire aux besoins de modélisation depuis l'analyse jusqu'à l'implémentation***.

La présente recherche s'appuie sur le ***« pouvoir » de communication*** du langage UML et le ***continuum de modélisation*** de ce langage qui permet de décrire un système depuis l'analyse jusqu'à l'implémentation.

De par ses nombreuses qualités, le langage UML est devenu le langage de modélisation non seulement de la communauté informatique mais aussi de nombreuses communautés exogènes.

Ce constat et le manque de capitalisation des modèles et des codes informatiques développés au Cemagref ont conduit la Direction Scientifique à doter l'établissement d'un atelier de génie logiciel afin que ses scientifiques disposent d'un outil de modélisation commun. Le but recherché était de faciliter la communication entre chercheurs de communautés différentes mais aussi de capitaliser les modèles et les codes produits. En effet, l'utilisation d'un atelier de génie logiciel permet de documenter un système et tous ses composants élémentaires. C'est un outil de capitalisation de connaissances.

Un atelier de génie logiciel est un moyen de capitalisation des connaissances d'un domaine.

La Direction Scientifique du Cemagref a choisi Objecteering comme atelier de génie logiciel. Cet atelier est doté d'une part, d'un mécanisme d'***extension du métamodèle d'UML*** permettant d'ajouter des concepts spécifiques à une communauté (cf. Annexe I-III.7) et, d'autre part, d'un ***langage de programmation interne***, appelé J, permettant d'introduire, de modifier et de supprimer des concepts dans un modèle. Le mécanisme extension, appelé ***Profil***, a été standardisé par l'OMG (OMG, 1999, 2001, 2003).

Le démarrage de la présente recherche (octobre 2000) coïncidant à quelques mois près avec la décision de la Direction Scientifique de doter le Cemagref d'un atelier de génie logiciel (fin 99), nous avons ***adopté l'atelier de génie logiciel Objecteering*** pour son mécanisme d'extension et son langage de programmation interne.

VI PROCESSUS DE RECHERCHE MIS EN ŒUVRE

Le processus mis en œuvre au cours des travaux de recherche décrits ici est un processus itératif et incrémental. Son cycle, représenté en figure 3, est semblable à celui de la méthode de conduite de projet Processus Unifié.

Après la définition de l'objectif général, la recherche est effectuée suivant des itérations où se succèdent une phase au cours de laquelle l'objectif élémentaire à atteindre est défini, une phase de recherche bibliographique, une phase de conceptualisation et de formalisation d'une solution et enfin une phase d'implémentation de la solution retenue. Tous les objectifs élémentaires étant traités, la rédaction du mémoire peut être entreprise et finalisée.

[6] UML

Le processus mis en œuvre est de nature incrémentale puisque, à chaque itération, un l'objectif élémentaire est abordée, une solution est trouvée et implémentée après une étude bibliographique.

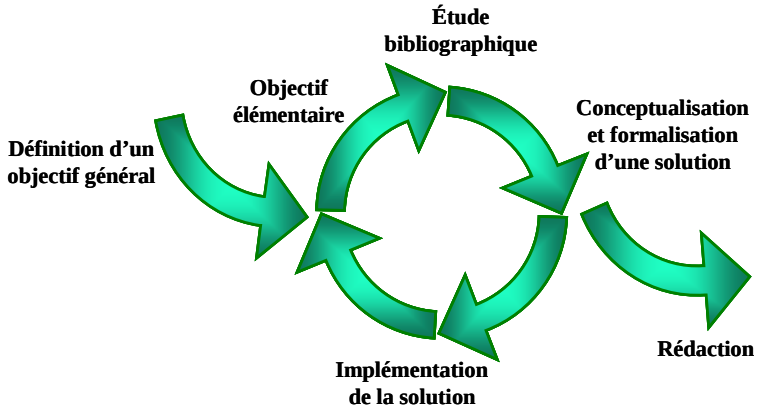


Figure 3 Cycle itératif et incrémental du processus de recherche mis en œuvre.

La représentation du processus de recherche de la figure 3 a l'inconvénient d'être très général et de manquer de précision quant au contenu et les articulations des itérations. Ce manque de précision a conduit à déployer linéairement les itérations afin de montrer et de mieux appréhender les différentes étapes du processus réellement effectué (cf. figure 4).

La première itération concerne l'objectif élémentaire de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique*. La bibliographie des formalismes et des méthodes de conception des *Systèmes d'Information Géographique* a été effectuée. En parallèle à cette étude bibliographique, il était nécessaire de bien maîtriser les fondamentaux pour mener à bien la recherche. Aussi, le formalisme objet et le langage UML ont été appris et mis en œuvre dans le cadre du projet SIRS Digues. Cela a permis d'être confronter aux problématiques réelles de conception d'un *Systèmes d'Information Géographique*. Au cours de cette période, le mécanisme d'extension du métamodèle d'UML et le langage J ont été maîtrisés et testés dans le cadre du projet SIRS Digues.

Fort de ces acquis, le langage pictogrammique de l'atelier de génie logiciel Perceptory a été adopté et les patrons de conception propre au domaine de l'*Information Géographique* ont été conceptualisés et formalisés. Une version simplifiée du langage pictogrammique a été implémentée dans un artefact de modélisation qui a été appelé ***Profil UML-SIG***, artefact qui sera détaillé ultérieurement.

La deuxième itération traite l'objectif élémentaire de capitalisation des connaissances. Comme pour l'itération précédente, la phase d'appropriation a été l'occasion de s'imprégner des concepts et de la philosophie sous-tendant l'approche *Model Driven Architecture* en faisant d'une part, une étude bibliographique du domaine et, d'autre part, des recherches exploratoires sur la diffusion automatique de concepts entre deux modèles et le maintien en cohérence des deux modèles.

Ces recherches exploratoires ont permis de formaliser un objet technique multimodèle qui a été nommé ***Software Development Process Model*** et d'imaginer son fonctionnement.

Cet objet technique doté de son fonctionnement a ensuite été implémenté dans un artefact de modélisation qui a été appelé ***Profil MDA***. Ce dernier sera décrit en détail plus avant.

La troisième itération s'intéresse à l'objectif élémentaire relatif au *processus de développement permettant le prototypage rapide en séance d'analyse*. Cela a conduit à s'intéresser aux méthodes de conduite de projet. Une bibliographie approfondie a donc été effectuée sur ce sujet.

L'analyse des différentes méthodes de conduite de projet a montré qu'il n'existait aucune méthode répondant à notre objectif élémentaire. La méthode ***Continuous Integration Unified Process*** a donc été imaginée comme soution. Cette étude bibliographique a été très bénéfique car elle a permis de proposer une généralisation de l'approche *Model Driven Architecture*, désignée ***Software Development Process Approach***. Cette généralisation a eu pour conséquence une meilleure formalisation et une meilleure intégration de l'objet technique *Software Development Process Model*.

Bien qu'étant représentée comme la dernière étape, la rédaction de certaines parties du mémoire a été effectuée au cours des phases d'approfondissement de la bibliographie.

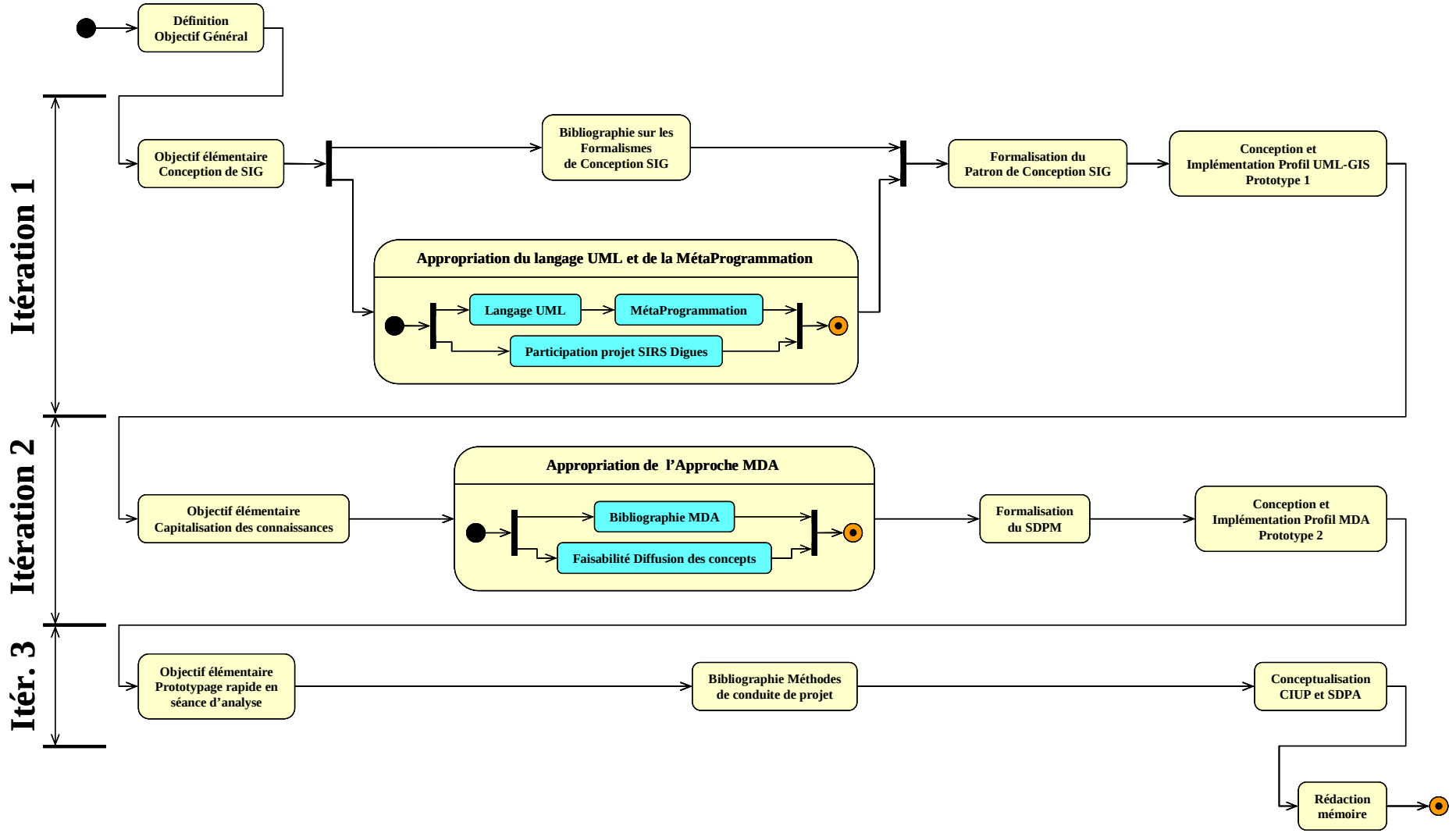


Figure 4 Les différentes étapes du processus de recherche.

En résumé, le processus itératif et incrémental expérimenté au cours de cette recherche alterne les aspects théoriques et les aspects techniques. Il a permis un pilotage plus aisé et plus fin de la recherche puisque, à chaque itération, l'objectif élémentaire à traiter pouvait être adapté en fonction des résultats de l'itération précédente.

VII DEMARCHE ET STRUCTURE DE LA THESE

La présente recherche étant à la conjonction des sciences de l'*Information Géographie* et de l'*Informatique*, cette double appartenance structure le plan du mémoire :

- ⇨ Une ***Partie A*** est dédiée à l'étude bibliographique.
- ⇨ Une ***Partie B*** est consacrée à la généralisation de l'approche Model-Driven Architecture.
- ⇨ Une ***Partie C*** présente la contribution à la modélisation des *Systèmes d'Information Géographique*.

Ces trois parties sont complétées par une ***Conclusion générale*** résumant les avancées de la recherche, des ***Références bibliographiques***, d'un ***Glossaire*** et d'une série d'***Annexes***. Le contenu des parties A, B et C est détaillé dans les paragraphes suivants et permet de décrire la démarche adoptée.

VII.1 Partie A – Étude Bibliographique

L'objectif premier de la recherche étant de créer un *outil d'aide à la conception de Systèmes d'Information Géographique*, nous nous sommes imprégnés d'une part, des concepts spécifiques au domaine de l'*Information Géographique* et, d'autre part, des méthodes et formalismes de cette communauté.

Un balayage rapide de ces méthodes et formalismes a permis de s'apercevoir qu'une même propriété spatiale ou temporelle pouvait porter plusieurs noms suivant la méthode ou le formalisme le mettant en œuvre, avec parfois des différences sémantiques. Par ailleurs, la capacité d'expression des propriétés spatiales et temporelles des entités manipulées en *Information Géographique* variait beaucoup d'une méthode à l'autre ou d'un formalisme à l'autre. Ce constat a conduit à effectuer une étude terminologique des propriétés spatiales et temporelles du langage pictogrammique de l'atelier de génie logiciel Perceptory car il semblait être le plus complet en termes de capacité d'expression. Cette étude terminologique a fait l'objet de la Partie C.

Fort du principal résultat de cette étude terminologique qui a été la dérivation d'un *Métamodèle SIG*, les méthodes et formalismes balayés précédemment ont été ré-analysés plus finement suivant une grille de lecture détaillée au Chapitre 1. Bien évidemment, la capacité d'expression était l'une caractéristiques de cette grille de lecture.

Au vu du foisonnement de méthodes et de formalismes développés, nous avons porté notre effort de comparaison sur ceux nous paraissant les plus importants ou ayant des caractéristiques particulières pouvant alimenter notre réflexion ou orienter nos recherches. Cette comparaison a confirmé la richesse du langage pictogrammique de l'atelier de génie logiciel Perceptory qui a été adopté et implémenté dans l'*outil d'aide à la conception de Systèmes d'Information Géographique*.

L'objectif élémentaire de *capitalisation des connaissances* du domaine étudié a amené à s'intéresser aux idées maîtresses de l'approche *Model Driven Architecture* d'où l'étude bibliographique du Chapitre 2 autour de cette approche.

Cette approche présentant quelques insuffisances à nos yeux, nous avons examiné les caractéristiques d'un certain nombre de méthodes de conduite de projet (cf. Chapitre 2). Deux d'entre elles ayant retenues plus particulièrement notre attention, nous avons approfondi leurs études bibliographiques. Il s'agit des méthodes Processus Unifié et eXtreme Programming.

VII.2 Partie B - Contribution à la généralisation de l'approche Model-Driven Architecture : L'approche Software Development Process

Les concepts informatiques du domaine informatique permettant la mise en œuvre d'un *processus de développement permettant le prototypage rapide en séance d'analyse* et la *capitalisation des connaissances* ont été définis.

Il s'agit d'une part, d'une nouvelle méthode de conduite de projet, appelée *Continuous Integration Unified Process* et, d'autre part, d'une proposition de généralisation de l'approche *Model Driven Architecture*, désignée *Software Development Process Approach*. Afin de satisfaire l'objectif élémentaire de réaliser un *outil d'aide à la*

conception de Systèmes d'Information Géographique, l'approche *Software Development Process Approach* a été réifiée en un artefact de modélisation nommé *Software Development Process Model*.

Appliquer un *processus de développement permettant le prototypage rapide en séance d'analyse* impose comme contrainte que la génération des prototypes soit effectuée « à la volée ». Si cette condition n'est pas réalisée, le processus de développement peut être fortement ralenti et les acteurs présents auront tendance à se désintéresser de l'analyse en cours alors que le but du prototypage rapide est justement de rendre plus efficace leur participation.

L'avantage de soumettre un prototype à la critique immédiate des acteurs est que ces derniers seront plus réactifs face à un prototype que face à un modèle dont ils ne connaissent pas ou ne maîtrisent pas toutes les finesses du langage de modélisation. Dans ce contexte de prototype rapide, les acteurs vont valider, modifier et supprimer des concepts plus efficacement que dans un contexte d'analyse classique.

Même si le développement d'un prototype est sommaire, c'est un développement au même titre que celui d'une application classique.

Ce constat a conduit à superposer un cycle de prototypage rapide sur la phase d'analyse du cycle de développement principal et à proposer la méthode Continuous Integration Unified Process. L'objectif avec cette méthode est de rechercher l'excellence sémantique du modèle au cours du cycle de prototypage rapide et l'excellence technique dans les phases dédiées du cycle de développement principal.

Cette nouvelle méthode est basée sur la méthode Processus Unifié et applique, en séance d'analyse, des pratiques issues de la méthode eXtreme Programming.

Le but final étant de générer des prototypes « à la volée » en séance d'analyse, cela suppose d'automatiser complètement l'évolution du modèle au cours du cycle de prototypage rapide depuis l'analyse jusqu'à l'implémentation. Ce type de processus automatique est dit ***full MDA (Kleppe, 2004 ; Softeam, 2005)***.

Le challenge technique a consisté à concevoir une panoplie de transformations simples permettant de pratiquer un processus full MDA lors de la conception et le développement d'applications SIG.

Parallèlement à cette préoccupation de recherche de qualité sémantique en séance d'analyse, les idées de séparation des spécifications thématiques et techniques préconisées par l'approche *Model Driven Architecture* paraissant intéressantes, une analyse approfondie de cette approche a conduit au constat que la préoccupation sous-jacente à l'approche *Model Driven Architecture* était la *capitalisation des connaissances*, autre objectif élémentaire de la recherche en cours.

Or, la *capitalisation des connaissances* est un souci et une problématique constante tout au long du cycle de développement d'une application informatique. En outre c'est une préoccupation indépendante de la nature de l'application et la méthode de conduite de projet adoptée.

Fort de ce constat, l'approche Software Development Process Approach a été proposée en considérant le cycle complet du développement d'une application.

Cette nouvelle approche s'appuie sur l'idée maîtresse qu'à chaque phase du développement un certain type de connaissances est mobilisé et introduit dans le modèle. Aussi, elle propose d'associer un modèle à chaque phase du développement, chacun d'eux constituant un niveau de *capitalisation des connaissances*.

La Software Development Process Approach généralise donc l'approche Model Driven Architecture et affine la capitalisation des connaissances en mobilisant un nombre de modèles plus important. Cette nouvelle approche présente l'avantage de pouvoir être étendue en ajoutant des modèles supplémentaires correspondant à des phases des tests, à des préoccupations particulières au sein d'une phase, etc.

Le principe de capitalisation des connaissances étant posé, un artefact multimodèle nommé Software Development Process Model a été défini et implémenté au sein d'un atelier de génie logiciel.

Les réflexions ayant conduit à proposer la méthode *Continuous Integration Unified Process*, l'approche *Software Development Process Approach* et l'artefact multimodèle *Software Development Process Model* résultent de l'étude bibliographique du Chapitre 2. Les concepts présentés ici constituent la principale contribution de la présente recherche. Ils sont détaillés au Chapitre 3.

Le Chapitre 4 traite des contraintes techniques imposées par l'atelier de génie logiciel qui ont conduit à limiter la capacité d'expression du langage pictogrammique dans l'attente d'une nouvelle version de cet atelier qui devrait lever le verrou technologique.

Le Chapitre 5 décrit la conception des transformations de modèle qui assurent l'évolution du modèle dans un processus full MDA satisfaisant ainsi l'objectif élémentaire de prototypage rapide. Les transformations spécifiques aux *Systèmes d'Information Géographique* reposent sur l'étude terminologique du Chapitre 7. Les relations et les lois de composition identifiées dans ce chapitre ont permis de définir certains mécanismes de ces transformations.

Enfin, le Chapitre 6 présente l'implémentation de l'artefact multimodèle *Software Development Process Model* et des transformations qui assurent l'évolution automatique du modèle. Ce chapitre présente aussi les interfaces Homme/Machine qui permettent d'exécuter ces transformations.

VII.3 Partie C – Contribution à la Modélisation des Systèmes d'Information Géographique

Suite au constat qu'une propriété spatiale ou temporelle pouvait être appelée différemment selon la méthode ou le formalisme de conception mis en œuvre pour la conception d'un *Systèmes d'Information Géographique*, il nous a semblé important de bien stabiliser la terminologie utilisée et de comparer la capacité d'expression des méthodes et des formalismes du domaine.

Pour ce faire, nous avons au Chapitre 7 mobilisé la *Théorie naïve des ensembles* que nous avons appliquée au langage pictogrammique de l'atelier de génie logiciel Perceptory, langage qui nous paraissait le plus complet au niveau de sa capacité d'expression. L'approche ensembliste a permis de construire trois taxinomies terminologiques et d'identifier la nature des relations entre ces taxinomies.

La similitude de ces taxinomies avec un modèle UML a conduit à définir un mécanisme de dérivation pour produire un modèle UML à partir des taxinomies. Le modèle UML obtenu représente en fait un métamodèle du langage pictogrammique, métamodèle que nous avons appelé ***Métamodèle SIG***. Au Chapitre 1, nous avons ensuite utilisé ce *Métamodèle SIG* comme outil de comparaison.

Le domaine d'intervention privilégié du Cemagref étant l'environnement, l'*outil d'aide à la conception de Systèmes d'Information Géographique* qui doit être réalisé dans le cadre de cette recherche sera utilisé pour modéliser des territoires (observatoires de pratiques territorialisées, …), des problématiques environnementales (suivi du patrimoine des digues, pollution…), etc. Or, de nombreuses entités manipulées dans ces problématiques ont une nature floue ou suscite une perception de flou (zone marécageuse).

Ce constat nous a conduits à étendre le langage pictogrammique de l'atelier de génie logiciel Perceptory pour exprimer des propriétés spatiales ou temporelles floues et à définir les concepts associés. Nous avons ensuite étendu le *Métamodèle SIG* du Chapitre 7 pour prendre en compte les propriétés spatiales ou temporelles floues. Le Chapitre 8 traite de l'extension du langage pictogrammique et de celle du *Métamodèle SIG*

Avertissement : La génération de prototypes « à la volée » suppose que les interfaces Homme/Machine soient elles-aussi produites automatiquement. Dans la phase de faisabilité actuelle, seule la génération de code SQL est entièrement automatisée, celle des interfaces Homme/Machine n'est pas réalisée. Au terme de cette recherche, il n'est donc pas possible d'appliquer pleinement la méthode *Continuous Integration Unified Process*. Cependant, les recherches qui vont permettre de générer automatiquement des interfaces Homme/Machine simples sont actuellement en cours dans le cadre du projet de *Conception d'Observatoires de Pratiques Territorialisées* (COPT) financé par *Agence Nationale de la Recherche*. Ces recherches doivent être finalisées dans les prochains mois, la méthode *Continuous Integration Unified Process* sera alors pleinement utilisable.

Avertissements

Nom Sémantiquement Riche vs Nom d'Implémentation

Contrairement à la pratique du domaine informatique, les concepts des modèles UML de ce document portent le nom utilisé dans le domaine avec tous les inconvénients qu'ils présentent du point de vue informatique : ils contiennent des espaces, des apostrophes, des caractères accentués, etc., caractères qui sont source d'erreurs à la compilation.

Un ***Profil de Nommage*** a été conçu et implémenté dans l'atelier de génie logiciel utilisé pour convertir les noms du domaine, *nom sémantiquement riche*, en noms interprétables par un compilateur, *nom d'implémentation*.

Le *Profil de Nommage* a pour première fonction de renommer les éléments de modélisation avec des noms d'implémentation épurés de tout caractère non compilable. Il a été doté d'une deuxième fonction : celle de préfixer systématiquement le nom d'implémentation des classes par **tb** ou **cl** selon que le langage cible est SQL ou un langage de programmation (Java, C++, etc.). Si cela est nécessaire, sa troisième fonction consiste à réduire la longueur des noms sémantiquement riche en fixant un nombre maximum de caractères par mot. Ce nombre est paramétrable via le panneau de configuration de l'atelier de génie logiciel.

Sans le *Profil de Nommage*, un concepteur soucieux de la génération de code ferait lui-même le renommage des concepts en nom d'implémentation. En opérant ainsi, le modèle serait compilable mais serait sémantiquement dégradé car le concepteur peut introduire involontairement des ambiguïtés. Par exemple[7], hors de son contexte, une classe *Foret* sera sûrement perçue par un forestier comme étant la zone couverte d'arbres (concept *Forêt*) alors qu'il pourrait très bien s'agir du concept *Foret* au sens de la mèche qui va permettre de percer un trou dans un arbre.

Le *Profil de Nommage* permet au concepteur de travailler et de conserver tout long de la modélisation les noms du domaine qui sont sémantiquement plus riche que ceux d'implémentation. La communication entre les acteurs du domaine et le concepteur est améliorée et, par voie de conséquence, la qualité du modèle. Le renommage des concepts avec des noms d'implémentation est effectué juste avant la génération de code.

Le Profil de Nommage permet donc de réaliser une modélisation sémantiquement plus riche puisque les noms des concepts du système ne sont pas contraints par des exigences d'implémentation. Par conséquence, tous les modèles manipulés au cours du développement sont des modèles sémantiques.

Nom de Classe et Nom de Rôle identiques

Dans de nombreux modèles UML de ce document, le nom du rôle d'une association est le même que celui de la classe contiguë au rôle. C'est un choix de modélisation que nous utilisation fréquemment car il est programmable et permet de générer automatiquement des patrons de conception ou des modèles avec une sémantique satisfaisante (cf. Annexe I-III.4.1). Si le besoin s'en fait sentir, le concepteur a toujours la possibilité intervenir manuellement après génération pour améliorer la sémantique du modèle.

Du point de vue langage UML, rien n'interdit d'adopter cette règle. Par contre, le code généré peut être à l'origine d'erreurs car, pour certains compilateurs, ces deux noms doivent être différents. Dans notre atelier de génie logiciel, ce problème est éludé puisque le ***Profil de Nommage*** préfixe automatiquement le nom des classes par **cl** ou **tb** suivant le langage cible (langage de programmation ou SQL).

[7] Souvent, le contexte permet de lever l'ambiguïté.

Concept Volume

Même s'il doit être affiné, le concept **Volume** a été introduit aux Chapitres 4 et 5 car il préfigure l'évolution de la recherche vers les langages de description des concepts spatiaux d'un espace 3D. Le langage pictogrammique adopté ne permet de décrire pour l'instant que les concepts spatiaux d'un espace 2D mais devrait intégrer prochainement les concepts spatiaux d'un espace 3D.

Deux autres raisons ont aussi contribué à introduire le concept *Volume* dans ces chapitres. La première concerne les patrons de conceptions SIG qui perdraient de leur intérêt et seraient en plus difficiles à appréhender sans ce concept. La seconde est consécutive à un problème technique. Il a été nécessaire d'ajouter un concept 3D afin de finaliser le codage de certaines fonctionnalités et en particulier celles permettant la gestion des pictogrammes. De ce fait, le concept *Volume* est présent dans un certain nombre de captures d'écran.

Par ailleurs, le concept de *Volume* tel qu'il a été modélisé dans le patron de conceptions SIG est dans l'esprit du concept *GM_Solid* de la norme ISO 19107 (ISO, 2003a) qui définit un volume par les surfaces qui le délimitent.

Partie A – Étude Bibliographique

Chapitre 1. Étude bibliographie autour des méthodes et formalismes pour la conception des systèmes d'information géographique : Comparaison avec le Métamodèle SIG

Avertissement : Les entités référencées et leurs propriétés de spatialité et de temporalité sont au cœur de l'étude bibliographique du présent chapitre. Pour ne pas alourdir ce dernier, nous avons volontairement consacré la Partie C à une formalisation des concepts liés aux langages SIG et pictogrammiques. Il est donc conseillé de s'approprier cette formalisation au préalable car la comparaison des méthodes et des formalismes présentée dans ce chapitre repose essentiellement sur cette formalisation.

Afin d'atteindre notre objectif élémentaire de réaliser d'un *outil d'aide à la conception de Systèmes d'Information Géographique*, une première étape a consisté à s'approprier les méthodes et les formalismes déjà existants qui permettent de concevoir et de réaliser des *Systèmes d'Information Géographique*.

Le but final de cette appropriation était d'adopter une méthode ou un formalisme afin de l'intégrer dans notre* outil d'aide à la conception de *Systèmes d'Information Géographique.

Ainsi, le paragraphe I reprend un extrait du texte de présentation du *Système européen de navigation par satellite* GALILEO pour montrer l'importance stratégique prise par l'*Information Géographique*. Il présente aussi les spécificités propres à cette typologie d'information et donne un aperçu du nombre de méthodes et de formalismes créés au cours de la dernière décennie.

Le nom d'un même concept étant différent d'une méthode ou formalisme à l'autre, nous avons ressenti le besoin d'effectuer une étude autour de la terminologie utilisée pour désigner les propriétés spatiales et temporelles des concepts étudiés.

Le langage pictogrammique de l'atelier de génie logiciel Perceptory nous paraissant au premier abord le plus « riche » en termes de capacité d'expression des propriétés spatiales et temporelles, l'étude terminologique a été mené sur ce langage. Elle a permis de construire trois taxinomies terminologiques (cf. Chapitre 7) desquelles un métamodèle du langage pictogrammique a été dérivé.

Ce métamodèle a été ensuite utilisé comme instrument de comparaison de la capacité d'expression des méthodes ou des formalismes.

Au paragraphe II, nous proposons des définitions du vocabulaire utilisé pour désigner certains concepts clés évoqués et manipulés tout au long du présent document.

Le paragraphe III replace les langages SIG et pictogrammique issus de l'étude terminologique du Chapitre 7 dans le contexte de l'atelier de génie logiciel Perceptory.

Au paragraphe IV, nous avons comparé la capacité d'expression d'un certain nombre de méthodes et formalismes soit parce qu'ils sont reconnus internationalement soit parce qu'ils présentent des caractéristiques techniques ou scientifiques intéressantes. La nature internationale et le caractère consensuel des normes du domaine de l'*Information Géographique* ont conduit à les inclure dans cette étude comparative.

Outre la capacité d'expression, ces méthodes, formalismes et normes ont été analysés en portant une attention toute particulière sur les points suivants : existence ou non de pictogrammes associés, prise en compte ou non de la dimension du système de coordonnées, génération d'un *Patron de conception SIG* au sens de la définition 81 et enfin l'implémentation ou non de la méthode ou du formalisme dans un outil informatique.

Le paragraphe V conclut l'étude bibliographique.

Sommaire détaillé

I GENERALITES ET SPECIFICITES DU DOMAINE L'INFORMATION GEOGRAPHIQUE

Ces dernières années, l'*Information Géographique* dans sa forme numérique est devenue une information stratégique aussi bien pour les organismes ou les sociétés qui produisent cette information ou la commercialisent que pour les états qui la destinent à une multitude d'applications *tels que le transport (localisation de véhicules, recherche d'itinéraire, contrôle de la vitesse, systèmes de guidage, etc.), les services sociaux (par exemple aide aux handicapés ou aux personnes âgées), la justice et les douanes (contrôles frontaliers), les travaux publics (Systèmes d'Information Géographique), le sauvetage de personnes en détresse ou les loisirs (orientation en mer et en montagne, etc.) (Commission Européenne, Site Web non daté).*

De fait, l'*Information Géographique* manipule des concepts du domaine étudié qui ont la particularité d'*être décrit de manière unique et générique* (Claramunt *et al.*, 1997 ; Gayte *et al.*, 1997) par :

- Des propriétés[8] ***thématiques*** qui décrivent le domaine modélisé, par exemple l'occupation du sol, le réseau hydrologique d'un territoire, etc.
- Des propriétés ***spatiales*** qui précisent la nature géométrique du concept thématique étudié. Par exemple, Polygone.
- Des propriétés ***temporelles*** qui rendent compte de l'existence et/ou de l'évolution de l'entité modélisée. Par exemple, *Instant*.

Les propriétés temporelles ont un statut particulier dans la mesure où elles peuvent s'appliquer, indépendamment ou simultanément, sur le concept lui-même ou sur la propriété spatiale. Lorsqu'elles s'appliquent sur la propriété thématique, elles rendent compte de l'évolution du concept thématique alors que lorsqu'elles s'appliquent sur la propriété spatiale, elles renseignent l'évolution de la propriété spatiale dans le temps.

Cette dernière typologie concerne un très grand nombre d'objets : ceux qui sont en mouvement (*Véhicule* en déplacement par exemple), ceux dont la géométrie change au cours de leur vie (évolution de la surface d'un *Lac*) et bien d'autres objets dont l'énumération exhaustive serait impossible.

La typologie d'objets possédant les propriétés listées ci-dessus sont dits *Entités géographiques* (Claramunt *et al.*, 1997 ; Gayte *et al.*, 1997).

L'importance stratégique et l'usage généralisé de l'*Information Géographique* ont donné lieu d'une part, à de nombreux travaux de recherche afin de faciliter la modélisation et la conception des *Systèmes d'Information Géographique* et, d'autre part, à des travaux de normalisation.

Un nombre important de méthodes et de formalismes ont été conçus depuis une vingtaine d'années (Bédard *et al.*, 1989) mais, au cours de la dernière décennie, le nombre de méthodes et de formalismes a littéralement explosé :

- Aigle et T-Aigle (Lbath, 1997 ; Lbath *et al.*, 2000),
- CONGOO (Pantazis *et al.*, 1996),
- Extent UML (Price *et al.*, 1999),
- ERT (Renolen, 1997),
- GeoFrame (Filho *et al.*, 1999a ; Rocha *et al.*, 2001),
- GeoOOA (Kösters *et al.*, 1997), Geo-OM (Tryfona *et al.*, 1997),
- GISER (Shekhar *et al.*, 1997),
- MADS (Parent *et al.*, 1997b ; Vangenot, 2001),
- OMT-G (Borges *et al.*, 2001),
- Perceptory (Bédard, 1999a ; Larrivée *et al.*, 2005 ; Proulx *et al.*, 2002),
- POLLEN (Gayte *et al.*, 1997),
- STER (Tryfona *et al.*, 1998a, 1998b ; Tryfona *et al.*, 1999).

[8] Certains auteurs parlent de composantes.

La liste présentée par ordre alphabétique est loin d'être exhaustive d'autant plus que de nombreux travaux de recherche ne donnent pas de nom au langage issu de ces travaux, au framework, à la méthode, etc. La plupart de ces méthodes et formalismes étendent soit le formalisme entité/relation soit le formalisme objet.

Aux méthodes et formalismes listés ci-dessus, il faut ajouter les normes produites par l'*Organisation Internationale de Standardisation* (ISO) et les contributions de l'*Open Geospatial Consortium* (OGC). Les textes publiés par ces deux instances internationales ne peuvent pas être ignorés car ils résultent d'un consensus entre experts du domaine de l'*Information Géographique* au niveau international.

Actuellement, les travaux de ces deux instances sont essentiellement axés vers la définition des spécifications permettant d'échanger de l'information géographique. À ce titre, ces travaux contribuent à l'interopérabilité des systèmes.

II DEFINITIONS GENERALES

Le balayage rapide des méthodes, formalismes, normes, etc. du paragraphe précédent a permis de constater la diversité des noms utilisés pour désigner les propriétés spatiales et temporelles d'une entité géographique. Ces propriétés peuvent être simples ou issues d'une combinaison d'entre-elles. Face à ce constat, nous avons posé les définitions suivantes :

Définition 1 La ***spatialité*** désigne, si elle existe, la propriété géométrique d'un concept.

Par exemple, la spatialité du concept thématique *Bâtiment* peut être le polygone de son emprise au sol, le volume du bâtiment, etc. On dit que le concept *Bâtiment* a une ***spatialité polygonale*** ou une ***spatialité volumique***.

Définition 2 La ***temporalité*** désigne, si elle existe, la propriété temporelle d'un concept.

Par exemple, la *date* à laquelle a eu lieu une avalanche est une information indispensable pour la gestion des risques. On dit que le concept thématique *Avalanche* a une temporalité *Instant*.

Dans notre contexte de travail, le concept d'entité géographique proposé par (Claramunt *et al.*, 1997) et (Gayte *et al.*, 1997) ne correspondant pas exactement à notre perception, nous avons défini le concept d'entité référencée :

Définition 3 Une ***entité référencée*** est un concept qui a *au moins une spatialité* ***ou*** *au moins une temporalité*. La propriété de temporalité peut renseigner l'entité référencée ou la spatialité.

Le modèle ci-dessous découle de la définition 3 :

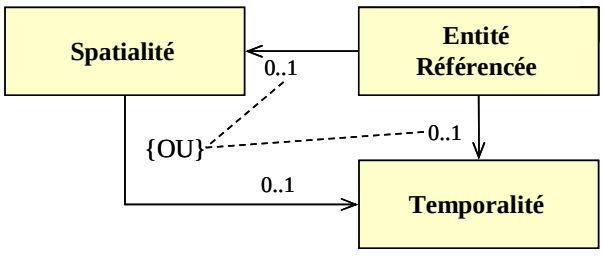


Figure 5 Modèle d'une entité référencée[9].

[9] L'opérateur OU impliqué dans la contrainte est la loi, dite parfois de disjonction inclusive, de l'Algèbre de Boole dont la table de vérité est donnée sur le site http://fr.wikipedia.org/wiki/Algèbre_de_Boole_(logique).

Il faut signaler qu'***un concept n'acquiert le statut d'entité référencée que s'il a une propriété spatiale ou une propriété temporelle.*** Cette dernière remarque est importante car, dans le processus de modélisation mis en œuvre dans le cadre de cette recherche, la saisie d'une entité référencée s'effectue en deux temps : d'abord le concept thématique étudié est saisi en utilisant le langage UML et ensuite la spatialité et/ou la temporalité du concept sont ajoutées via le langage pictogrammique.

III LE LANGAGE PICTOGRAMMIQUE DE L'ATELIER DE GENIE LOGICIEL PERCEPTORY

Nous ne nous attarderons pas ici sur le langage pictogrammique déployé par l'équipe d'Yvan Bédard dans l'atelier de génie logiciel Perceptory puisque les langages SIG et pictogrammique du Chapitre 7 en sont issus. Ces langages résultent d'une accumulation d'expérience longue de plus de seize ans (Bédard *et al.*, 2004) puisque les travaux de cette équipe ont été les premiers du genre et ont fortement influencé la plupart des méthodes et des formalismes créés par la suite. Cette influence a porté aussi bien sur la sémantique des concepts de spatialité et de temporalité que sur les pictogrammes utilisés pour annoter les concepts thématiques.

Ils font suite aux travaux de recherche en modélisation de bases de données spatio-temporelles (Bédard *et al.*, 1989) qui ont permis de concevoir le formalisme MODUL-R (Bédard *et al.*, 1992 ; Bédard *et al.*, 1996 ; Caron, 1991) comme une extension au formalisme Entité/Relation (Chen, 1976) pour renseigner la spatialité et/ou la temporalité des entités référencées.

Le formalisme MODUL-R a été implémenté dans Orion, *premier atelier de génie logiciel incluant les pictogrammes spatiaux et la génération de code automatique* (Bédard *et al.*, 2004). MODUL-R et Orion préfiguraient le langage pictogrammique et l'atelier de génie logiciel Perceptory.

Après quelques années d'utilisation de MODUL-R et d'Orion dans différents projets de conception et de développement de bases de données spatio-temporelles, cette équipe a ressentie le besoin d'améliorer le langage pictogrammique. Aussi, en 1996-1997 un langage pictogrammique étendu a été implémenté dans la première version de l'atelier de génie logiciel Perceptory (Bédard *et al.*, 2004). À cette occasion le formalisme Entité/Relation a été abandonné au profit du formalisme objet et du langage UML.

Dans sa version actuelle, Perceptory est un plug-in implémenté dans la plate-forme Visio[10]. Toutefois, il est en cours de redéveloppement en Java[11] et sera intégré dans l'atelier de génie logiciel ArgoUML (ArgoUML, Site Web non daté) qui est un logiciel sous licence libre. L'utilisation de Perceptory sera alors totalement gratuite[12].

Dans la version Java, le langage pictogrammique devrait être étendu à la représentation des spatialités dans un espace 3D puisque plusieurs publications (Bédard *et al.*, 2004 ; Larrivée *et al.*, 2005, 2006)) sont issues de cette équipe.

Perceptory a été conçu comme une alternative aux ateliers de génie logiciels professionnels car de nombreuses études montrent que ces derniers sont souvent utilisés comme outil de dessin et que la génération de code est peu exploitée. C'est ce constat qui a amené à n'implémenter dans Perceptory que les principaux concepts UML du diagramme de classes (paquetage, classe, attribut, association, généralisation/spécialisation et notes). Perceptory permet de générer le code SQL pour le SGBDR Oracle.

Dans son processus interne, Perceptory interprète l'information représentée par les pictogrammes pour générer le code SQL correspondant. De ce fait, la gestion des relations entre concepts SIG est déléguée au SGBDR cible, Oracle dans le cas présent. Perceptory utilise donc virtuellement le schéma d'Oracle.

Cet outil simple mais opérationnel donne entière satisfaction pour la modélisation des bases de données. Sur ce créneau, il a les qualités d'un produit industriel. Perceptory peut être mis en œuvre depuis la phase d'analyse jusqu'à la génération de code SQL permettant ainsi de créer des bases de données spatiales et spatio-temporelles.

L'atelier de génie logiciel Perceptory est téléchargeable gratuitement sur le site (Bédard, 1998). Bien que téléchargé par les utilisateurs de plus de quarante pays (Bédard *et al.*, 2004), la majorité d'entre eux réside dans les pays du G7. Une documentation et des tutoriaux facilitent la prise en main de cet atelier.

Le cadrage stratégique de Perceptory en fait un outil de modélisation particulièrement adapté à un public peu enclin à investir lourdement dans l'appropriation de langage UML et/ou des utilisateurs disposant de moyens financiers limités.

---

[10] Visio est un logiciel qu'il est possible d'étendre avec de nouveaux formalismes.
[11] La nouvelle version devrait être disponible fin 2006 (Bédard, 1998).
[12] Actuellement, l'utilisateur de Perceptory doit acquérir une licence Visio minimale.

IV CONCEPTS SIG MIS EN ŒUVRE PAR DIFFERENTS FORMALISMES ET METHODES DE MODELISATION DE SYSTEME D'INFORMATION GEOGRAPHIQUE

Les méthodes, formalismes et normes analysés dans ce paragraphe sont présentés suivant un ordre alphabétique. Le lecteur intéressé par une vision historique des méthodes et formalismes dédiés à la conception de *Systèmes d'Information Géographique* pourra se reporter à la communication de (Filho *et al.*, 1999b) et à celle de (Bédard *et al.*, 2004). La comparaison de la capacité d'expression des méthodes, des formalismes et des normes a été effectuée en s'appuyant sur le *Métamodèle SIG* (cf. Chapitre 7). Pour ce faire, lorsqu'un concept de spatialité ou de temporalité de la méthode ou du formalisme analysé est sémantiquement identique à un concept du *Métamodèle SIG*, il est représenté coloré en jaune paille. Dans le cas contraire, il est représenté estompé en gris.

IV.1 CONGOO (CONception Géographique Orientée Objet)

Le formalisme CONGOO est issu des travaux de (Pantazis *et al.*, 1996 ; Pantazis *et al.*, 1997). C'est un formalisme mis en œuvre dans le cadre de la méthode de conduite de projet MECOSIG[13] (Pantazis *et al.*, 1996) qui propose une démarche au concepteur d'un *Système d'Information Géographique* pour réaliser son application. Ce formalisme distingue trois catégories d'objets géographiques : les objets géographiques simples (OGS), composés (OGC) et complexes (OGX).

La première catégorie est composée des entités référencées annotées des concepts de spatialité primitive de nature discrète Point, Ligne et Polygone auxquels il faut ajouter le concept de Surface dont la nature est continue. Le langage SIG défini au Chapitre 7 ne permet pas de modéliser les concepts de spatialité ou de la temporalité de nature continue.

La deuxième catégorie inclut d'une part, les objets géographiques homogènes[14] (OGCH) qui correspondent aux entités référencées à multi-spatialité (cf. Chapitre 7-IV.4) et, d'autre part, les objets géographiques composés non homogènes (OGCNH) qui sont les macro-entités référencées d'agrégation (cf. Chapitre 7-IV.3).

La troisième catégorie inclut les objets géographiques complexes qui, comme pour la deuxième catégorie, distingue ceux qui sont homogènes (OGXH) de ceux qui sont non homogènes (OGXNH). Les premiers sont les entités référencées à spatialité multiple (cf. Chapitre 7-IV.2.1.1) et les seconds sont des entités macro-entités référencées d'agrégation composées d'entités référencées à spatialité multiple. Le formalisme CONGOO ne permet pas d'exprimer la spatialité alternative.

Une limite forte du formalisme CONGOO est qu'il ne s'intéresse qu'aux propriétés spatiales des entités référencées. De ce fait, la temporalité et les combinaisons de spatialité et de temporalité ne peuvent pas être exprimées avec ce formalisme. Les spatialités dérivées et temporalités dérivées n'existent pas dans ce formalisme. La figure 6 résume les concepts SIG couverts par le formalisme CONGOO.

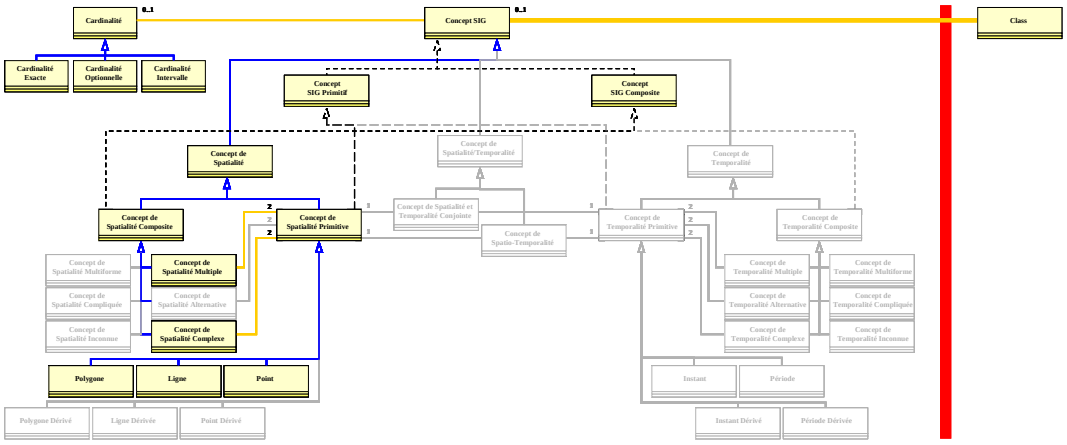


Figure 6 Concepts SIG couverts par le formalisme CONGOO.

[13] Méthode de COnception de Systèmes d'Information Géographique.

[14] Exemple extrait de (Pantazis *et al.*, 1996) : un concept État composé uniquement de Communes sera considéré comme un objet géographique homogène.

Le formalisme CONGOO permet d'exprimer aussi la topologie entre objets géographiques composés ou complexes. Ce formalisme a une particularité intéressante qui est de pouvoir exprimer les topologies interdites entre objets géographiques. Cette faculté est particulièrement intéressante car, lors de l'analyse de certains systèmes, il est tout aussi important et parfois plus facile de décrire ce qui ne doit pas être que ce qui doit être[15].

Bien que les objets géographiques de certains exemples soient annotés simultanément du concept de spatialité et d'un pictogramme (cf. figure 7), dès que le modèle devient conséquent, la représentation pictogrammique est abandonnée.

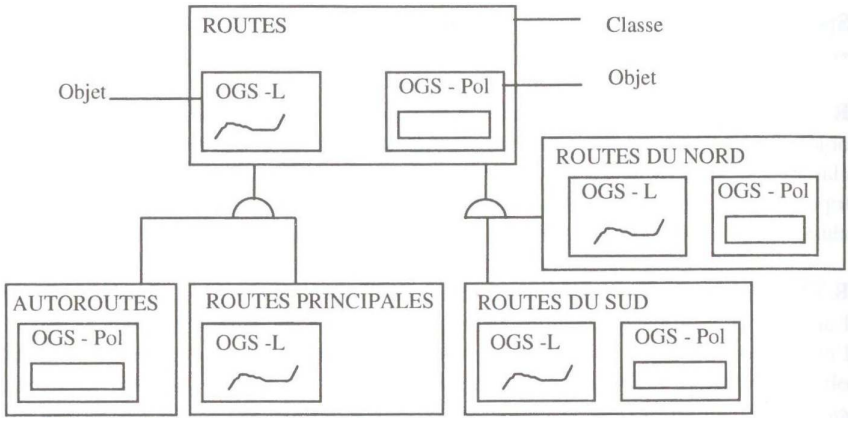


Figure 7 Exemple d'un réseau routier décrit avec le formalisme CONGOO (extrait (Pantazis *et al.*, 1996)).

Enfin, CONGOO est un formalisme de modélisation des *Systèmes d'Information Géographique* qui a la capacité de représenter les entités référencées dont les limites sont incertaines (Pantazis *et al.*, 1997).

C'est un formalisme qui a été pensé essentiellement pour les objets géographiques représenté dans un système de coordonnées 2D.

Par ailleurs, comme ce formalisme est principalement dédié à l'analyse d'un système, les relations entre concepts de spatialité sont considérées comme acquises et ne sont jamais représentées dans les modèles.

Bien que l'annonce ait été faite (Pantazis *et al.*, 1997), le formalisme CONGOO n'a jamais été implémenté au sein d'une atelier de génie logiciel dédié (Billen *et al.*, 2004). C'est probablement la raison principale de sa diffusion limitée.

IV.2 GeoFrame

GeoFrame est un framework de conception de bases de données géographiques fondé sur le langage UML (Filho *et al.*, 1999a). Il permet d'exprimer les propriétés spatiales des entités référencées via le formalisme UML-GeoFrame. Ce formalisme initialement limité à l'expression des seules propriétés spatiales a été étendu aux concepts temporels par Luciana Vargas da Rocha en 2001 en définissant le framework UML-GeoFrame-T (Rocha, 2001 ; Rocha *et al.*, 2001).

Ce formalisme a une particularité intéressante car il distingue les concepts non géographiques (NonGeographicObject) des concepts géographiques qui ont des propriétés spatiales. Dans ce dernier cas, il offre au concepteur la possibilité de faire la différence entre les entités référencées discrètes (GeographicObject) et celles qui ont une nature continue (GeographicField). Le pictogramme associé à un NonGeographicObject est △, celui à GeographicObject est ◬ et celui à GeographicField est Ⓕ (Filho *et al.*, 1999a ; Sodré *et al.*, 2005).

La dichotomie entre les concepts géographiques et non géographiques de GeoFrame est implicite lors de la mise en œuvre des langages SIG ou pictogrammique. Avant annotation, un concept thématique est non géographique et devient entité référencée après annotation. Il possède alors une spatialité et/ou une temporalité. Si la spatialité ou la temporalité d'un concept n'est pas disponible au cours de la modélisation, la spatialité inconnue et/ou la temporalité inconnue du langage SIG (cf. Chapitre 7-VI) permettent de faire la distinction entre concept et entité référencée. La dichotomie introduite par GeoFrame est bien couverte par les langages SIG et pictogrammique.

[15] Par exemple, le croisement d'une route et d'un cours d'eau est une situation à proscrire.

Dans le contexte d'objets géographiques (△), UML-GeoFrame a adopté les concepts de spatialité primitive (Point, Ligne et Polygone). Concernant la pictogrammie spatiale, le formalisme UML-GeoFrame a adopté la même que celle du langage pictogrammique. Les travaux de Luciana Vargas da Rocha (Rocha, 2001 ; Rocha *et al.*, 2001) reprennent ceux de Nectaria Tryfona et de Christian S. Jensen. Ils affinent les propriétés temporelles des entités référencées comme le montre le modèle temporel de GeoFrame-T (cf. figure 8). Les concepts d'Instant (instanteTemporal), de Période (intervaloTemporal) et l'union d'Instant et de Période (elementoTemporal) possèdent une information complémentaire qui distingue différentes notions temporelles. La première notion concerne la temporalité de l'objet modélisé (TempoValidade), la seconde la date de la transaction (TempoTransação) et la troisième combine les deux précédentes (Bitemporal). De plus, la temporalité de l'objet modélisé peut être linaire ou branché. Un pictogramme a été associé à chacune de ces douze notions temporelles (Rocha, 2001 ; Rocha *et al.*, 2001).

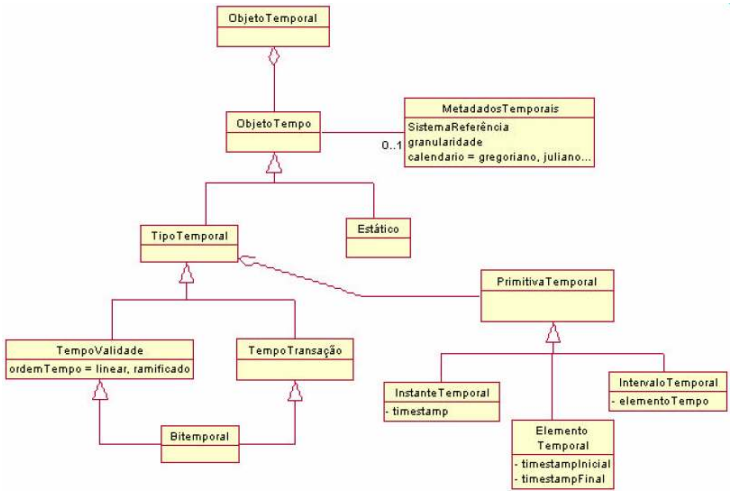


Figure 8 Modèle temporel de GeoFrame-T (extrait (Rocha, 2001)).

Le formalisme UML-GeoFrame a aussi adopté les concepts de spatialité multiple et de temporalité multiple, celui de spatialité multiforme ainsi que celui de spatialité et temporalité conjointe. Par contre, les concepts de multi-spatialité et de multi-temporalité ne sont pas couverts par ce formalisme.

L'application d'affection **<<** entre les concepts SIG et les entités référencées est implémentée par une relation de généralisation/spécialisation ce qui distingue ce formalisme des autres qui généralement l'implémente par une association. De ce fait, la spatio-temporalité au sens de la définition 38 n'est pas possible. La figure 9 résume les concepts SIG couverts par le framework GeoFrame.

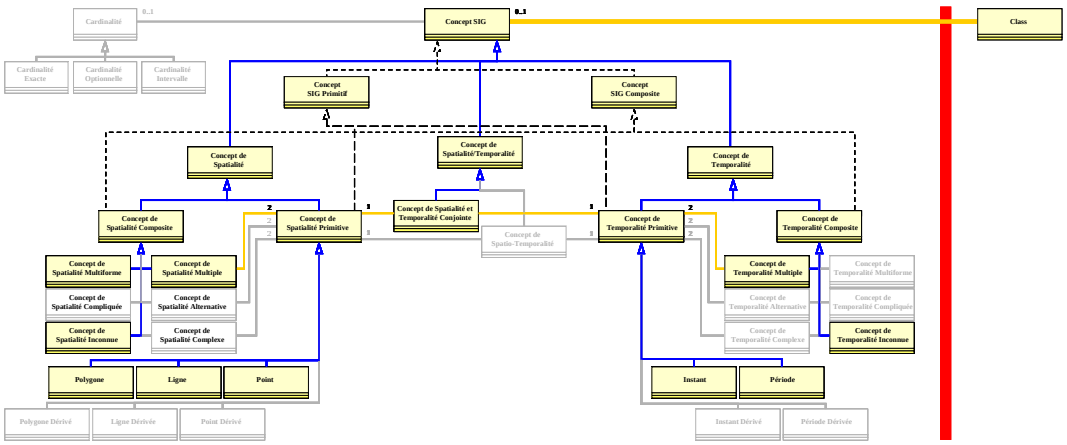


Figure 9 Concepts SIG couverts par UML-GeoFrame.

Dans le contexte de champs géographiques (F), Jugurta Lisboa Filho propose d'annoter les entités géographiques avec des concepts adaptés à la vision continue du concept thématique (Filho *et al.*, 1999a). Pour ce faire, il a introduit les concepts d'échantillonnage irréguliers de points, de grille de points, de polygones adjacents, d'iso-lignes, de grille de cellules et de réseaux de triangles irréguliers (TIN). Bien évidemment, un pictogramme a été associé à chacun de ces concepts.

La dimension du système de coordonnées n'est pas explicitement indiquée mais la nature 2D des concepts relatifs aux champs géographiques laisse à penser que c'est la seule dimension exprimable avec le framework UML-GeoFrame.

Initialement implémenté dans un outil informatique appelé SIGPROGB (Filho *et al.*, 1999a), le framework GeoFrame a été ensuite implémenté dans l'atelier de génie logiciel ArgoUML (ArgoUML, Site Web non daté). Il a pour nom ArgoCASEGEO (Filho *et al.*, 2004).

IV.3 MADS (Modeling of Application Data with Spatio-temporal features)

Le formalisme MADS est issu des recherches effectuées principalement au sein de l'École Polytechnique de Lausanne par l'équipe de Stefano Spaccapietra. Ce formalisme met à disposition du concepteur des concepts de spatialité primitive. Le tableau 1 liste les sept concepts primitifs de MADS et les pictogrammes associés. Il montre d'une part, la correspondance entre les concepts du formalisme MADS et ceux du langage SIG et, d'autre part, la correspondance entre les pictogrammes des deux formalismes.

Concept MADS	Concept SIG
Point	Point
Line	Ligne
OrientedLine	
SimpleSurface	Polygone
Surface	
Instant	Instant
Interval	Période

Pictogrammes MADS	Langage pictogramme
•	⊡
∫	⊠
∫	
◥	◙
◥	
⏲	⏲
◔	◔

Tableau 1 Correspondance entre concepts de SIG à droite et entre pictogrammes à gauche.

Les concepteurs de MADS ont aussi défini des concepts de spatialité complexe et de temporalité complexe mais selon (Brodeur *et al.*, 2002) ils sont couverts par les concepts primitifs du langage SIG.

MADS n'a pas de concepts dédiés pour exprimer la spatialité alternative, la spatialité multiple ou leurs homologues temporels. Toutefois, pour exprimer la spatialité multiple ou la temporalité multiple, les auteurs suggèrent d'utiliser le mécanisme du multihéritage comme le montre la figure 10.

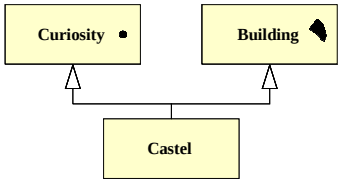


Figure 10 Multihéritage induisant plusieurs géométries pour un objet (Parent *et al.*, 1998).

Ce mécanisme n'est pas totalement équivalent à la spatialité multiple du langage SIG car, dans l'exemple de la figure 10, il a été nécessaire d'introduire deux concepts supplémentaires (Curiosity, Building) qui ne seront peut être utilisés que pour exprimer la spatialité multiple du concept *Castel*. La multi-représentation introduite plus récemment permet de pallier ce problème (Vangenot, 2001 ; Vangenot *et al.*, 2002).

Concernant les concepts combinant spatialité et temporalité, MADS propose la spatialité et temporalité conjointe mais par contre il n'offre pas l'équivalent des concepts de spatio-temporalité (cf. définition 38).

Il est à signaler que la convention pictogrammique de la spatialité et temporalité conjointe est inversée c'est-à-dire que la spatialité est placée à droite et la temporalité à gauche. La multi-spatialité et multi-temporalité ne sont pas prises en compte par le formalisme MADS.

Les spatialités inconnue, compliquée et multiforme et leurs homologues temporels n'ont pas d'équivalent formels dans MADS. Comme dans Perceptory, le formalisme MADS permet au concepteur d'annoter les propriétés spatiales et temporelles des attributs. La figure 11 résume les concepts SIG couverts par MADS.

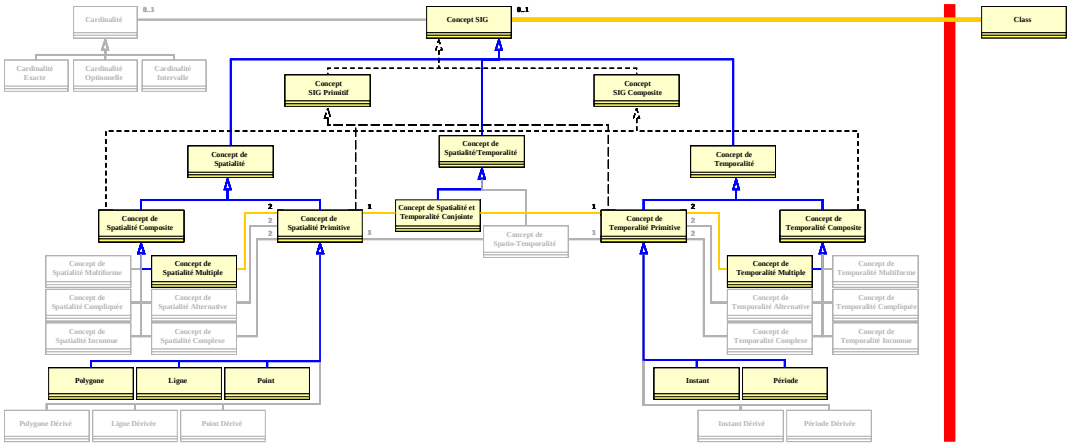


Figure 11 Concepts SIG couverts par le formalisme MADS.

Les concepteurs du formalisme MADS se sont intéressés à l'expression des concepts de spatialité floue et de la temporalité floue (Shu *et al.*, 2003) mais ces concepts semblent ne pas être une priorité de l'équipe puisqu'ils ne sont plus évoqués dans le dernier ouvrage consacrés à ce formalisme (Parent *et al.*, 2006).

Le formalisme MADS propose aux concepteurs de *Systèmes d'Information Géographique* la possibilité de décrire la topologie entre entités référencées. Il permet aussi d'annoter directement, lorsque cela et nécessaire, la nature spatiale, temporelle et/ou topologique des relations d'associations.

L'application d'affection << entre les concepts SIG et les entités référencées est implémentée de façon implicite par une association.

Dans MADS, les entités référencées sont modélisées dans un système de coordonnées unique de dimension 2 puisque le concept Point a une abscisse ***X*** et une ordonnée ***Y*** (Parent *et al.*, 2006).

La figure 12 illustre l'architecture des concepts de MADS. Cette figure montre aussi que les relations entre concepts primitifs ne sont pas explicitées comme dans la plupart des formalismes. La raison est que ces concepts sont au final transformés en concepts spécifiques au SGBDR cible.

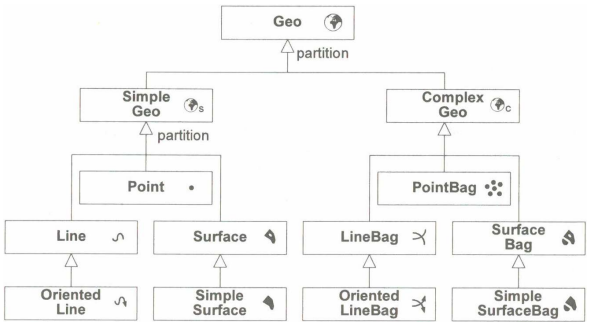


Figure 12 Les types de données du formalisme MADS (extrait (Parent *et al.*, 2006)).

Le formalisme MADS a été implémenté dans une application informatique dédiée. Malgré de nombreux développements depuis une dizaine d'années et en particulier dans le cadre du projet européen MurMur (MurMur, 2000) au cours duquel 219 mois/homme ont été consacrés au développement (Zimányi *et al.*, 2002), seul un prototype de recherche est disponible à ce jour.

IV.4 La Normalisation relative aux concepts de spatialité et de temporalité

IV.4.1 *Normes ISO 19107, ISO 19108 et ISO 19109*

La norme ISO 19107 (ISO, 2003a), intitulée Information Géographique - Schéma Spatial, est dédiée aux seuls concepts de spatialité. La norme ISO 19108 (ISO, 2002), intitulée Information Géographique - Schéma Temporel, est quant à elle consacrée aux seuls concepts de temporalité.

Ces deux normes sont très complètes puisque les membres du *Comité Technique 211* (TC 211) ont même étudié et défini les concepts permettent de décrire la topologie soit entre concepts de spatialité soit entre concepts de temporalité.

Bien évidemment, ces normes définissent les spécifications des concepts de spatialité primitive Point (GM_Point), Ligne (GM_LineString) et Polygone (GM_Polygon) ainsi que ceux de temporalité primitive Instant (GM_Instant) et Période (GM_Period) (cf. figure 13). Elles définissent aussi les concepts de multi-spatialité mais pas ceux de multi-temporalité. La spatialité complexe est couverte par les concepts ISO de la famille GM_Aggregate mais l'ISO ne propose pas d'équivalent temporel.

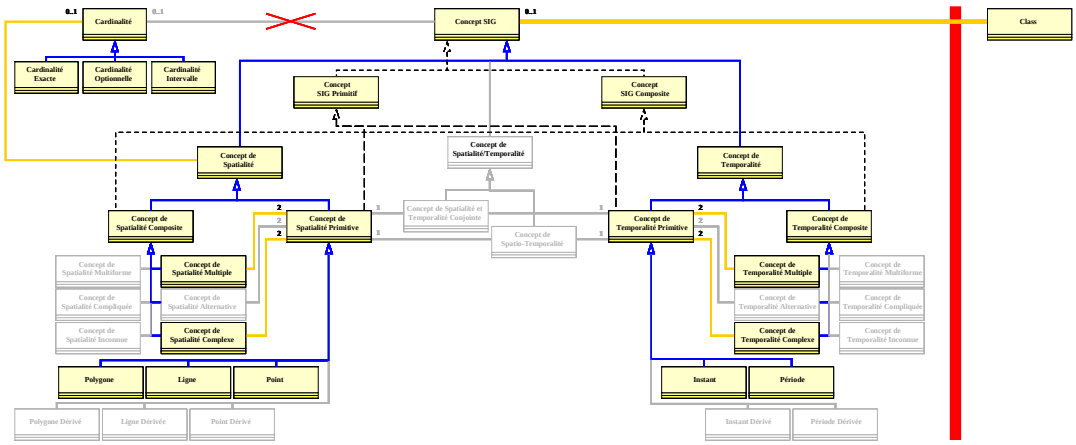


Figure 13 Concepts SIG couverts par les normes ISO 19107, ISO 19108 et ISO 19109.

Selon (Brodeur *et al.*, 2002), les spatialités multiple et alternative sont couvertes par la norme ISO 19109 (ISO, 2005) qui précise les règles d'application des schémas. La norme ISO 19109 s'appliquant aux schémas spatial et temporel, les mécanismes décrits par Jean Brodeur pour modéliser les spatialités multiple et alternative avec les concepts ISO peuvent aussi être appliqués pour représenter les temporalités multiple et alternative.

La variété des concepts de spatialité et temporalité définis souffre de l'absence des concepts combinant spatialité et temporalité (Brodeur *et al.*, 2002). Toutefois, c'est un thème qui est au programme de travail du TC 211 et qui donnera lieu à la norme ISO 19141 dont le titre est Geographic information - Schema for moving features (ISO, Target year: 2007). Le document de travail est actuellement au stade de *Committee draft*. La norme devrait être publiée en octobre 2007.

Par ailleurs, ces normes ne définissent pas les spatialités et les temporalités de nature informative qui facilitent la modélisation des systèmes (cf. Chapitre 7) : spatialité dérivée, spatialité inconnue, spatialité compliquée, spatialité multiforme et les homologues temporels.

Par contre, ces normes définissent les *concepts de nature volumique* (GM_Solid, GM_Cone, GM_Cylinder, etc.) sur lesquels l'équipe d'Yvan Bédard a également travaillé et défini une série de pictogrammes.

Une autre famille de concepts définis par ces normes et absents du langage SIG sont les concepts permettant l'interpolation (GM_BSplineCurve, GM_BBezier, etc.). L'absence de ces concepts du langage SIG est un choix qui résulte de la spécificité *bases de données* de l'équipe d'Yvan Bédard.

La dimension du système de coordonnées est prise en compte par l'ISO. Toutefois, cette caractéristique est redondante car elle est définie au sein de deux concepts :

- ⇨ Dans le concept SC_CRS[16] de la norme intitulée Information Géographique - Système de références spatiales par coordonnées ISO 19111 (ISO, 2003b). Celui-ci a une relation d'association avec le concept racine GM_Object dont héritent tous les concepts de spatialité primitive évoqués ci-dessus et en particulier le concept GM_Point.
- ⇨ Dans le concept GM_Point qui intervient dans toutes les autres spatialités primitives (GM_LineString, etc.). Or, GM_Point fait appel au concept *DirectPosition* qui possède un attribut *coordinate* défini comme une liste de valeurs et un attribut *dimension* égal au nombre de valeurs de cette liste.

Le choix de séparer au sein de normes différentes les concepts de spatialité et ceux relatifs au système de coordonnées a conduit à définir deux fois la dimension du système de coordonnées. La gestion de cette caractéristique en est alourdie puisqu'il faut constamment vérifier l'égalité des deux valeurs. Nous verrons au paragraphe III.5 du Chapitre 5 comment nous avons traité ce problème.

Les relations entre concepts de spatialité primitive ne sont pas explicitées dans le schéma spatial. Par contre, les constructeurs des concepts de spatialité primitive permettent de reconstituer ces relations. Les relations entre concepts de temporalité primitive sont quant à elles parfaitement précisées.

Les travaux concernant ces normes sont essentiellement théoriques et définissent, au travers de la spécification des interfaces, l'interopérabilité entre technologies (COM, CORBA, etc.), entre bases de données ou en vue de l'échange de données.

Enfin, les deux normes ISO 19107 et ISO 19108 ne s'intéressent pas à la représentation pictogrammique des concepts SIG.

IV.4.2 *Spécifications OGC*

Les travaux de OGC sont alignés sur ceux de l'ISO et *implémentent un profil du schéma spatial décrit dans la norme ISO 19107* (OGC, 2005). Aussi, les concepts de spatialité primitive sont, au préfixe GM_ en moins, les mêmes que ceux de la norme décrite dans le paragraphe précédent. Par ailleurs, le schéma est fortement simplifié puisque seuls les concepts de multi-spatialité ont été retenus. La figure 14 illustre les concepts SIG couverts par les spécifications de l'OGC.

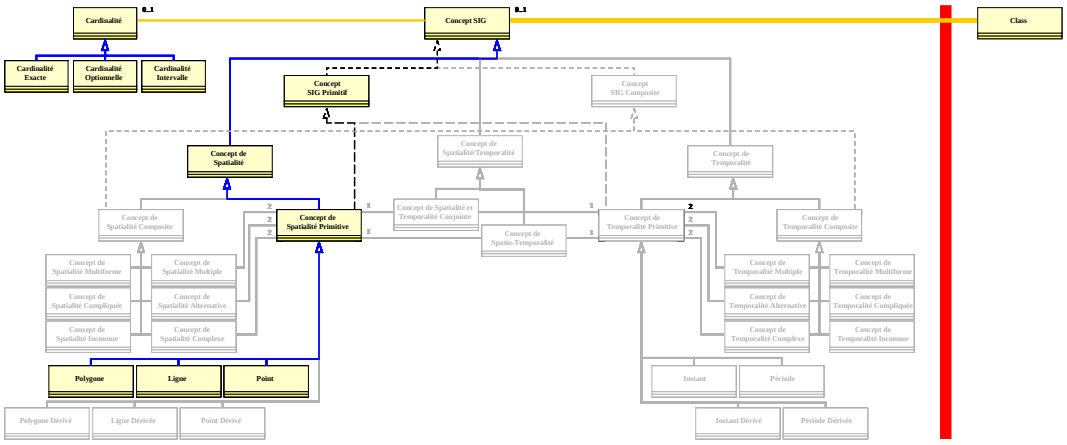


Figure 14 Concepts SIG couverts par les spécifications de l'OGC.

Tout comme les travaux de l'ISO, ceux de l'OGC n'ont abordé ni les concepts combinant spatialité et temporalité ni l'utilisation d'un langage pictogrammique facilitant la communication entre concepteur et acteurs du domaine modélisé.

Les réflexions de l'OGC étant alignées sur celles de l'ISO, les concepts OGC sont associés à un système de coordonnées (Spatial Reference System) via le concept racine *Geometry*.

[16] CRS : Coordinate Reference System.

Contrairement au schéma spatial de l'ISO, celui de l'OGC indique les relations existant entre concepts de spatialité primitive définissant complètement un *Patron de conception SIG* au sens de la définition 81.

De par sa simplicité, le schéma de l'OGC a été adopté par différents SGBD et en particulier par MySQL[17] et par PostGis[18]. Ces SGBD offrent à leurs utilisateurs des fonctionnalités permettant la saisie, le stockage et l'interrogation des bases de données mais aussi la possibilité de réaliser des requêtes de nature spatiale ou topologique (Clais, 2003).

IV.5 POLLEN (Procédure d'Observation et de Lecture de L'ENvironnement)

La méthode POLLEN est une méthode qui a été développée dans la deuxième partie des années 90 pour aider les concepteurs de Système d'Information sur l'Environnement à modéliser, concevoir et implémenter leurs applications (Claramunt *et al.*, 1997 ; Gayte *et al.*, 1997). Elle *s'appuie d'une part sur les concepts théoriques de représentation du temps et de l'espace (Chen, 1976) et d'autre part sur le formalisme de la méthode OMT (Rumbaugh et al., 1995).*

Les concepts fondamentaux de cette méthode sont les concepts de spatialité *Point*, *Ligne* et *Aire* et les concepts de temporalité *Instant* et *Intervalle*. La terminologie est légèrement différente de celle de la majorité des autres méthodes mais la sémantique correspond à la celle de concepts du *Métamodèle SIG* : *Aire* correspond à *Polygone* et *Intervalle* à *Période*.

Les classes spatio-temporelles de la méthode POLLEN permettent l'annotation simultanée des entités référencées avec des concepts de spatialité et de temporalité. Ces classes spatio-temporelles héritent d'une classe spatiale et d'une classe temporelle (cf. figure 15). De ce fait, la spatialité et la temporalité s'appliquent toute deux aux concepts thématiques. La sémantique de ces classes spatio-temporelles ne correspond pas à la spatio-temporalité du langage SIG mais à celles des concepts de spatialité et temporalité conjointe (cf. Chapitre 7-IV.2.2.1).

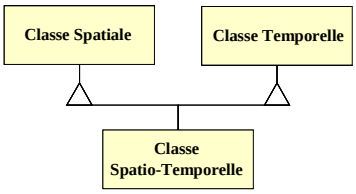


Figure 15 Principe de modélisation des entités spatio-temporelles.

La figure 16 montre de façon synthétique les concepts SIG primitifs et ceux combinant la spatialité et la temporalité couverts par la méthode POLLEN.

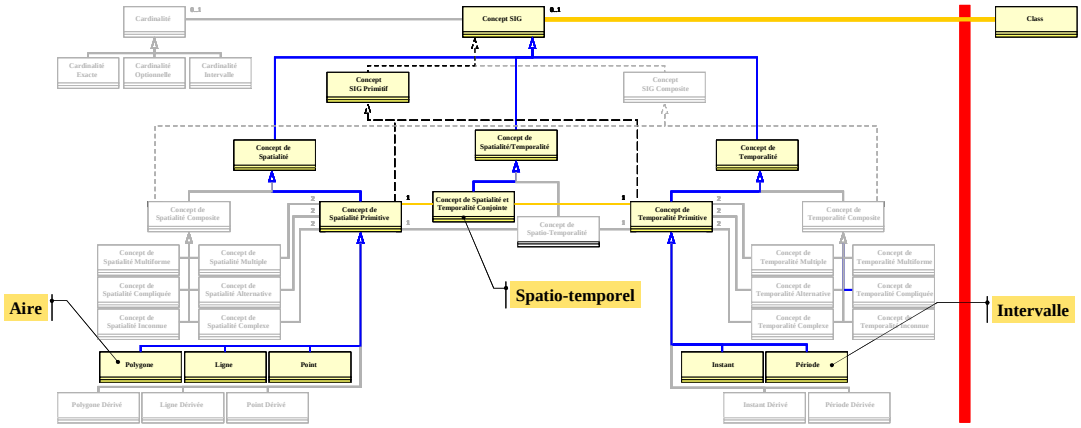


Figure 16 Concepts SIG couverts par la méthode POLLEN.

[17] Extrait : MySQL implémente l'extension spatiale en suivant les spécifications du Open GIS Consortium (OGC) (MySQL, 2006).
[18] Extrait : PostGIS follows the OpenGIS "Simple Features Specification for SQL" and has been certified as compliant with the "Types and Functions" profile (PostGIS, 2005).

Les auteurs de la méthode POLLEN et du formalisme associé ont opté pour *manipuler les classes spatiales comme des* ***classes opaques****, c'est-à-dire en ignorant leur structure, et en ne s'intéressant uniquement qu'aux services qu'elles rendent*. De cette approche découle deux choix décisifs :

- ⇨ Le premier est relatif aux patrons de conceptions SIG. L'objectif de la modélisation étant l'implémentation d'un *Système d'Information Géographique* et de sa base de données, les auteurs de POLLEN n'ont pas éprouvé le besoin de définir le *Patron de conception SIG* décrivant les relations entre concepts SIG. De fait, le patron de conception spatial est celui de la plate-forme cible (SGBDR ou l'outil SIG). À la variante terminologique près signalée ci-dessus, le patron de conception temporel est celui rencontré dans toutes les méthodes ou formalismes (cf. figure 17).

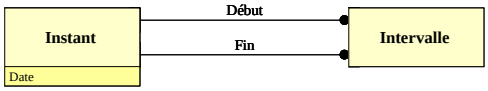


Figure 17 Patron temporel de la méthode POLLEN[19].

- ⇨ Le second choix concerne le type de l'application d'affection **<<** entre les concepts SIG et les entités référencées. Le choix des auteurs de ne s'intéresser qu'aux services des classes opaques et d'ignorer la structure a conduit tout naturellement à relier les entités référencées aux classes opaques par des relations de généralisation.

La figure 18 montre le modèle d'analyse des entités référencées *Cours d'eau* et *Route* suivant le formalisme POLLEN.

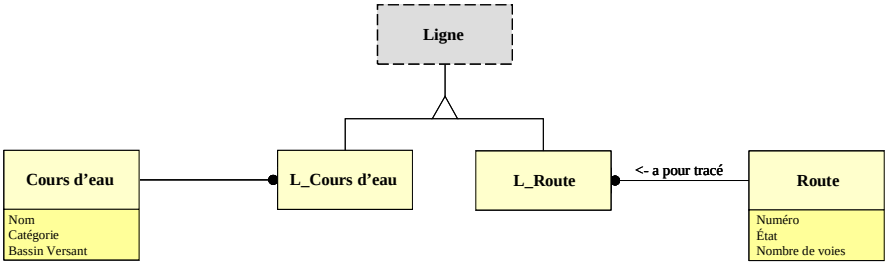


Figure 18 Exemple de modèle en phase d'analyse (Gayte *et al.*, 1997).

Cette particularité de la méthode POLLEN résulte d'une préoccupation d'implémentation comme le montre l'analyse des modèles de conception détaillée des exemples dans (Gayte *et al.,* 1997). Tout modèle de conception détaillée précise la plate-forme logicielle cible. La figure 19 est le modèle de conception détaillée dérivé du modèle d'analyse de la figure 18 et destiné à ArcInfo. Dans la phase de conception, la classe opaque *Ligne* est remplacée par la classe abstraite *Arc* d'ArcInfo. Pour une cible MapInfo, la classe opaque *Ligne* aurait été substituée par la classe abstraite *Obj* (Gayte *et al.*, 1997).

Sans l'explicité formellement, les auteurs de la méthode POLLEN opèrent par transformation de modèle entre la phase d'analyse et la phase de conception détaillée puisque les classes opaques sont remplacées par les classes correspondantes de la plate-forme logicielle cible.

Il est à signaler aussi qu'un modèle d'analyse peut donner lieu à plusieurs modèles de conception détaillée préfigurant les travaux autour de l'approche MDA et plus particulièrement la dichotomie des modèles suivant qu'ils sont indépendants ou spécifiques à une plate-forme.

L'absence de pictogrammie est une faiblesse du formalisme de la méthode POLLEN car la mise en œuvre des classes opaques n'est pas aisée. En effet, soit la représentation des classes opaques est unique dans le modèle, soit elle est multiple. Dans le premier cas, le modèle peut devenir rapidement illisible à cause des nombreuses relations de généralisation qui relient les entités référencées aux classes opaques. Dans le second cas, le principe d'unicité des concepts préconisé par le formalisme objet n'est plus respecté.

[19] Formalisme OMT de la méthode Pollen.

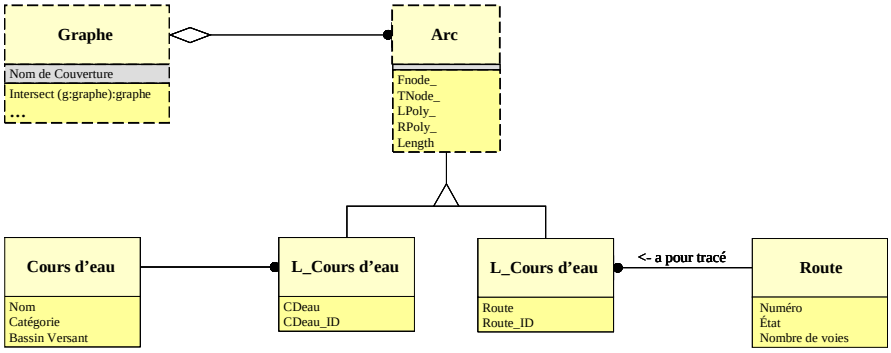


Figure 19 Modèle de conception détaillé pour l'outil SIG ArcInfo du modèle de la figure 18 (Gayte *et al.*, 1997).

Comme dans beaucoup de formalismes, la dimension du système de coordonnées n'est pas prise en compte car elle est déléguée au SGBD. La méthode de modélisation de Système d'Information sur l'Environnement POLLEN n'a jamais été implémentée dans un atelier de génie logiciel.

IV.6 STER (Spatio-Temporal Entity/Relation)

Comme l'indique son nom complet, STER est un formalisme fondé sur celui d'Entité/Relation conçu par Peter Pin-Shan Chen (Chen, 1976). Il est dédié à la conception de bases de données spatio-temporelles. Nectaria Tryfona, un des concepteurs majeurs de ce formalisme, a une longue expérience en modélisation de bases de données spatio-temporelles puisqu'il a participé d'une part, à la conception du formalisme Geo-OM (Tryfona *et al.*, 1997) *qui dérive directement de la méthode OMT* (Claramunt *et al.*, 1997) et, d'autre part, à l'intégration des concepts de spatialité et de temporalité au formalisme UML (Price *et al.*, 1999).

Point, Line et Region sont les concepts de spatialité primitive du formalisme STER qui correspondent respectivement à Point, Ligne et Polygone du langage SIG. Ce formalisme qui a inspiré les travaux de Luciana Vargas da Rocha (Rocha, 2001 ; Rocha *et al.*, 2001) distingue la temporalité de l'objet modélisé (*existence time*) de celle de l'objet informatique (*transaction time*). Par contre, comme le montre la figure 20, STER ne différencie pas les concepts *Instant* et *Période*. En effet, les concepts *existence time* et *transaction time* sont traités comme des périodes puisqu'ils ont un début et une fin.

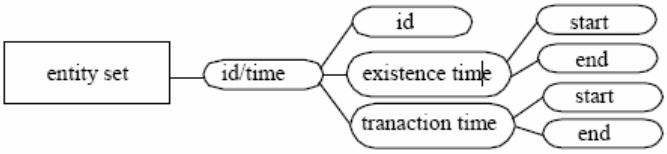


Figure 20 Patron de conception des concepts existence time et transaction time (extrait (Tryfona *et al.*, 1998b)).

Au niveau pictogrammie, le formalisme STER affecte un pictogramme à chacun des concepts de spatialité et de temporalité qu'il met en œuvre : (P) est associé à *Point*, (L) à *Line*, (R) à *Region*, (et) à *existence time* et (tt) à *transaction time*.

La multi-spatialité et la multi-temporalité auraient put être envisagées dans STER via la cardinalité des associations mais les différentes communications autour de STER ne mentionnent pas ce type de spatialité ou de temporalité. Par ailleurs, les patrons représentant l'implémentation finale des spatialités affectent une cardinalité 1 à la spatialité.

À la différence d'autres formalismes, STER différencie la spatialité et temporalité conjointe (cf. figure 21-modèle A) de la spatio-temporalité (cf. figure 21-modèle B). Dans le premier cas, les propriétés spatiales et temporelles portent toute deux sur le concept thématique et les pictogrammes spatiaux et temporels sont séparés alors que, dans le second cas, la propriété temporelle porte sur la propriété spatiale et les deux propriétés sont incluses dans le même pictogramme. Le formalisme STER permet aussi la modélisation de la spatio-temporalité et temporalité conjointe ce qui n'a pas été abordé dans le langage SIG mais qui est implicitement modélisable.

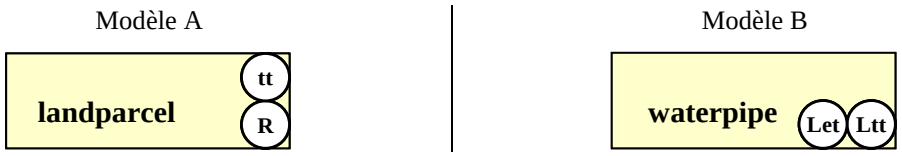


Figure 21 Modélisation de l'entité polygonale landparcel et son temps de transaction (à gauche) et celle de la position dans le temps de l'entité linéaire waterpipe (à droite) (extrait (Tryfona *et al.*, 2000)).

Comme Perceptory, une caractéristique intéressante du formalisme STER est que les propriétés spatiales et temporelles peuvent renseigner les entités mais aussi les attributs de ces dernières. Les propriétés spatiales et temporelles peuvent en outre annoter les relations entre entités (Tryfona *et al.*, 1999). La figure 22 illustre l'ensemble des concepts SIG couverts par le formalisme STER.

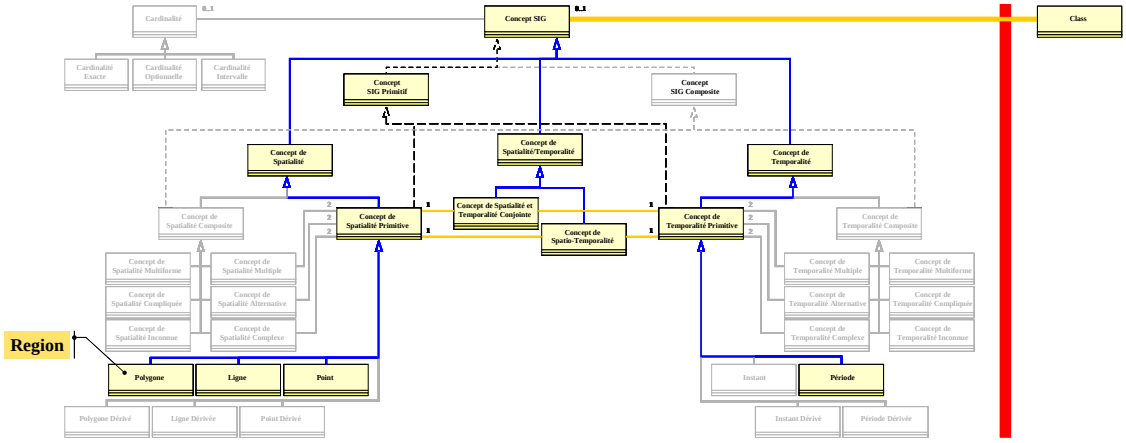


Figure 22 Concepts SIG couverts par formalisme STER.

Dans le cadre du projet européen Chorochronos, un prototype implémentant le formalisme STER a été réalisé (Tryfona *et al.*, 1998b ; Tryfona *et al.*, 1999). Il est structuré en deux composants (cf. figure 23) :

- Un premier CASTER (Computer-Aided SpatioTemporal Entity Relationship) permet de réaliser des modèles Entité/Relation enrichis du formalisme STER.
- Un second CASTR (Computer-Aided SpatioTemporal Relational) qui transforme un modèle réalisé avec le formalisme STER en modèle Entité/Relation dédié au SGBDR Oracle. Cette transformation s'effectue suivant les règles du modèle STR (Spatio-Temporal Relational).

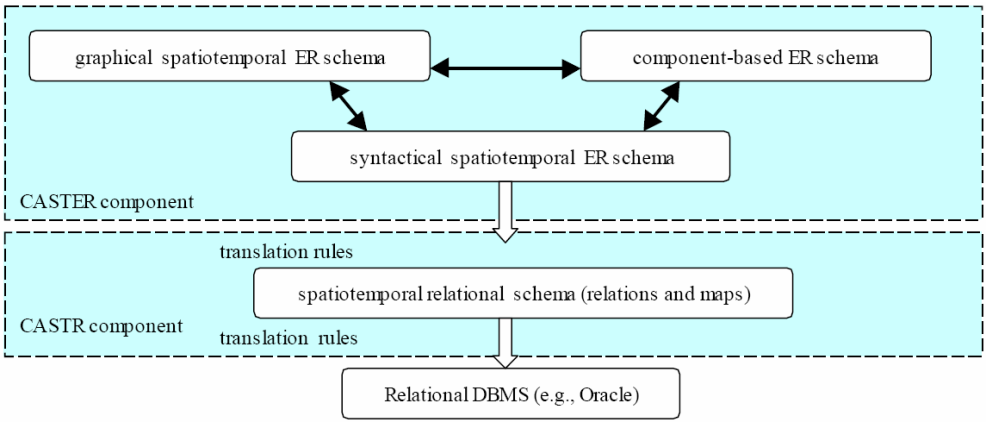


Figure 23 Architecture du prototype implémentant le formalisme STER (extrait (Tryfona *et al.*, 1999)).

Les deux composants CASTER et CASTR constituant le prototype ont été conçus en amont du SGBDR Oracle (Tryfona *et al.*, 1999). La figure 23 montre, outre l'architecture des deux composants, la séquence des transformations permettant de convertir un modèle réalisé avec le formalisme STER et de l'instancier dans le SGBDR Oracle.

Dans la mesure où le SGBDR cible est Oracle, le formalisme STER ne s'intéresse pas à la dimension du système de coordonnées puisque c'est au cours du développement que ce problème est abordé en fonction des données présentes ou à saisir dans la base. Pour la même raison, les relations entre concepts de spatialité ne sont pas décrites.

V CONCLUSION

Un constat important dans le cadre de cette recherche est que l'application d'affectation << est généralement instanciée (matérialisé) par une relation d'association mais deux formalismes POLLEN et UML-GeoFrame ont opté pour une instanciation en relation de généralisation/spécialisation.

Bien que les noms soient parfois différents, les concepts primitifs Point, Ligne et Polygone sont toujours présents dans les méthodes, formalismes et normes analysés. Ces méthodes, formalismes et normes définissent toutes des combinaisons de spatialités primitives et de temporalités primitives dont la sémantique est souvent proche. Lorsqu'il y a des différences sémantiques, elles sont principalement dues au type d'instanciation de l'application d'affectation << (cf. Chapitre 7).

La plupart des formalismes et méthodes récentes ont adoptés un langage pictogrammique pour exprimer les concepts de spatialité et de temporalité car cela améliore la lisibilité et l'appropriation des modèles par les acteurs. Le langage pictogrammique utilisé par Yvan Bédard et son équipe pour exprimer la spatialité et la temporalité des entités référencées offrent une capacité d'expression quasiment sans limites et se révèle être un atout formidable pour décrire finement la géométrie et l'évolution d'une entité référencée et ce même lorsque la spatialité et/ou la temporalité à décrire sont riches voire complexes.

Toutefois, le langage SIG souffre de l'absence de concepts topologiques qui permettent d'exprimer ce type de relation entre entités référencées. L'équipe d'Yvan Bédard a travaillé sur le sujet mais ces travaux n'ont pas été volontairement intégrés dans l'atelier de génie logiciel Perceptory (Bédard *et al.*, 1998 ; Bédard *et al.*, 1999 ; Normand, 1999). Ce choix délibéré est le résultat de leur expérience. En effet, le traitement des relations topologiques, souvent appelées contraintes d'intégrité spatiales, n'est pas effectué en même temps ni de la même façon que la modélisation de la base de données. C'est la raison pour laquelle ces relations sont modélisées avec un autre outil appelé CSory compatible avec le référentiel de Perceptory.

L'impression initiale de « richesse » en termes de capacité d'expression étant confirmée par l'étude comparative, nous avons adopté les langages SIG et pictogrammique développés par l'équipe d'Yvan Bédard et implémentés dans Perceptory.

Malgré son importance, peu de méthodes, formalismes et normes s'intéressent à la représentation de la dimension du système de coordonnées. La principale raison vient du fait que ces méthodes, formalismes et normes ont été conçus pour modéliser ou implémenter des *Systèmes d'Information Géographique* dans des SGBDR (Oracle, MapInfo, PostGis, MySQL, etc.) qui assurent directement la gestion de la dimension du système de coordonnées.

Ayant été pensés pour la conception de *Systèmes d'Information Géographique* dans des SGBDR, ces méthodes et ces formalismes adoptent le *Patron de conception SIG* du SGBDR cible et elles ne se préoccupent pas des relations entre concepts SIG primitifs.

Enfin, le Métamodèle SIG dérivé de l'étude terminologique du Chapitre 7 s'est révélé être un excellent outil de comparaison entre formalismes car il permet de visualiser d'un simple coup d'œil les concepts couverts.

Chapitre 2. Étude bibliographique autour de l'approche Model-Driven Architecture et des méthodes de conduite de projet

Martin Fowler[20] :
You should use iterative development only on projects that you want to succeed (Larman, 2002b).

Le chapitre précédent présentait les méthodes et formalismes pour la conception des *Systèmes d'Information Géographique* et comparait la capacité d'expression de ces méthodes et formalismes. Cette étude bibliographique s'intégrait dans objectif élémentaire de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique*.

La *capitalisation des connaissances* étant un autre objectif élémentaire de la recherche, nous nous sommes intéressés aux idées de séparation de spécifications préconisées par l'approche *Model Driven Architecture*. L'approche MDA présentant certaines insuffisances, nous avons cherché une solution dans les méthodes de conduite de projet qui organise le développement d'une application.

Le présent chapitre est donc consacré à l'étude bibliographique de l'approche MDA et d'un certain nombre de méthodes de conduite de projet.

Le paragraphe I décrit les concepts majeurs de l'approche MDA, présente un panorama succinct des recherches en cours et une étude comparative du développement d'une application suivant un processus traditionnel et suivant l'approche MDA.

Le paragraphe II présente les premières méthodes de conduite de projet mise en œuvre pour le développement d'applications informatiques et quelques unes des méthodes récentes qui ont alimentées notre réflexion.

Le paragraphe III conclut ce chapitre en présentant les points que nous avons cherchés à résoudre dans ce travail de recherche.

[20] Traduction (Larman, 2002a) : Ne faites appel au développement itératif que pour les projets que vous voulez voir aboutir.

Sommaire détaillé

I APPROCHE MODEL DRIVEN ARCHITECTURE (MDA™)

I.1 Généralités

En 2001, l'Object Management Group (OMG) a formalisé un ensemble de préconisations dans l'objectif de résoudre les problèmes de productivité, de portabilité, d'intégration et d'interopérabilité rencontrés au cours du développement des applications informatiques (Kleppe *et al.*, 2003). Ces préconisations doivent aussi faciliter le développement d'applications en remobilisant les connaissances métiers et/ou techniques existantes.

De ces réflexions est issue l'approche *Model Driven Architecture* qui fixe le cadre d'utilisation des modèles lors du développement d'une application (Miller *et al.*, 2001). Le respect des préconisations de cette approche doit faciliter le développement, la maintenance et l'évolution de toute nouvelle application mais aussi minimiser l'impact de l'évolution des plates-formes matérielles et/ou logicielles.

En 2003, un guide de la démarche MDA est édité par l'OMG (Miller *et al.*, 2003). Ce document décrit comment mettre en œuvre l'approche MDA et introduit quelques nouveaux concepts.

Afin de promouvoir l'approche MDA, l'OMG prône qu'un développement effectué suivant cette approche permet de réaliser des gains de productivité.

I.2 Description

L'approche MDA préconise la séparation des *spécifications métiers* propre à un domaine de celles d'implémentation. Pour réaliser cette séparation, l'approche MDA propose de mettre en œuvre deux types de modèles (cf. figure 24) :

⇨ Les modèles indépendants des plates-formes (modèle PIM[21]).

[21] PIM : Platform Independent Model

⇨ Les modèles spécifiques à une plate-forme (modèle PSM[22]).

Ces modèles constituent une architecture. D'après la figure 24, les modèles PIM et PSM peuvent être décrits avec un ou plusieurs métamodèles.

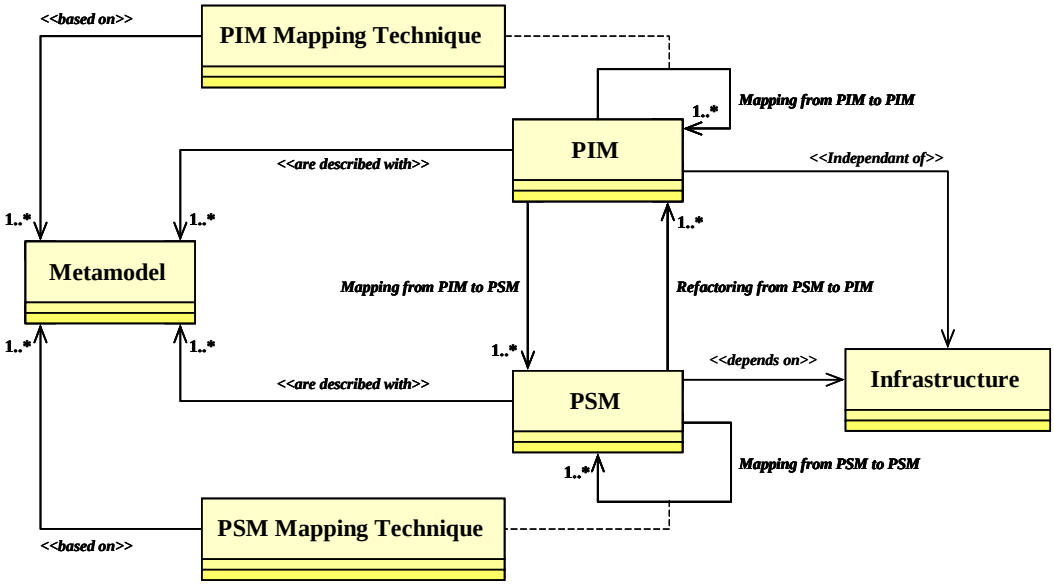


Figure 24 Extrait du métamodèle de l'approche MDA (Miller *et al.*, 2001).

En 2003, un nouveau type de modèle a été défini et ajouté à l'architecture MDA afin de *combler le fossé existant entre les experts du domaine thématique et les experts de la conception et de la construction de l'application* (Miller *et al.*, 2003). Cette nouvelle typologie de modèles est appelée *modèle indépendant de la programmation* (Blanc, 2005) plus couramment qualifiée CIM[23]. Selon Xavier blanc, les modèles CIM sont destinés à décrire les exigences des acteurs du domaine.

Une caractéristique importante du modèle CIM est que le *vocabulaire familier des acteurs du domaine est utilisé pour sa spécification* (Miller *et al.*, 2003). En outre, il est explicitement mentionné dans ce guide que ce modèle est *utile, non seulement comme aide à la compréhension du problème, mais aussi comme source de vocabulaire partagé à utiliser dans d'autres modèles*. Dans la communauté base de données, le dictionnaire de données joue le rôle de source de vocabulaire du modèle CIM.

L'approche MDA introduit aussi la notion de ***transformation de modèles*** qui permet de faire évoluer les modèles au cours du développement. Formellement les transformations peuvent être manuelles ou automatiques (Miller *et al.*, 2001, 2003). Cependant, les recherches actuelles sont principalement axées sur l'automatisation des transformations afin d'augmenter la productivité du développement.

La figure 24 met aussi en exergue l'existence de transformations dont le modèle source est du même type[24] que le modèle issu de la transformation. Ces transformations sont appelées transformations PIM/PIM ou PSM/PSM. Cette même figure montre qu'il y a aussi des transformations dont les modèles avant et après transformation sont de types différents. Ce sont les transformations qualifiées de PIM/PSM ou PSM/PIM.

Il faut signaler que le métamodèle de la figure 24 fait apparaître un autre intérêt important de l'approche MDA qui est de pouvoir dériver plusieurs modèles PSM d'un même modèle PIM grâce à plusieurs transformations PIM/PSM.

I.3 Panorama succinct des recherches en cours

Dans l'approche MDA, les transformations étant l'objet technique permettant l'évolution automatique des modèles, l'effort de recherche depuis la formalisation de l'approche MDA a été concentré sur cet objet technique. Les recherches portent plus spécialement sur la conception de langages de transformation. Les

[22] PSM : Platform Specific Model
[23] Computation Independent Model
[24] *Mapping from PIM to PIM* & *Mapping from PSM to PSM*

tableaux 2 et 3 donnent un aperçu du nombre de langages actuellement étudiés et dont certains ont été implémentés.

Requirements	UMT w/XSLT	ATL	YATL	BOTL	MOLA	UMLX	VMT
Commonly-used language	yes	no	no	no	no	no	partly
Inheritance	no	yes	no	no	no	no	yes
Graphical notation	no	no	no	yes	yes	yes	yes
Lexical notation	yes	yes	yes	no	no	no	yes
Declarative	yes	no	no	yes	no	yes	no
Bidirectional	no	no	no	yes	no	no	no
XML support	yes	no	no	no	no	no	no
Text-to-UML	yes	no	no	no	yes	no	no
UML-to-UML	yes	yes	yes	yes	yes	yes	yes
UML-to-text	yes	yes	no	no	no	no	no
UML tool independence	yes	yes	yes	yes	no	yes	no
No proprietary intermediate structures	no	yes	yes	yes	yes	yes	yes
Traceability	no	no	yes	no	no	yes	yes
Metamodel/MOF-based	no	yes	yes	yes	yes	yes	no

Tableau 2 Langages et outils de transformation (extrait (Grønmo *et al.*, 2005)).

Requirements	UMT w/XSLT	QVTMerge	QVT-Compuware/ Sun	MOFScript
Commonly-used language	yes	no	no	no
Inheritance	no	yes	yes	yes
Graphical notation	no	yes	yes	no
Lexical notation	yes	yes	yes	yes
Declarative	yes	yes/no	no	no
Bidirectional	no	yes	yes	no
XML support	yes	no	no	yes
Text-to-UML	yes	no	no	no
UML-to-UML	yes	yes	yes	no
UML-to-text	yes	no	no	yes
UML tool independence	yes	yes	yes	yes
No proprietary intermediate structures	no	yes	yes	yes
Traceability	no	yes	no	no
Metamodel/MOF-based	no	yes	yes	yes

Tableau 3 Propositions OMG (extrait (Grønmo *et al.*, 2005)).

Percevant l'importance stratégique, l'OMG a cherché à « unifier » les langages de transformation en lançant un l'appel à proposition en 2002 (OMG, 2002b). L'objectif suivi par l'OMG est, comme pour le langage UML, de définir un langage de transformation de modèles adopté par la majorité de la communauté informatique. *Cet appel d'offre a suscité un grand intérêt* puisque sept propositions ont été soumises (Belaunde, 2004 ; Belaunde *et*

al., 2004). L'une d'entre-elles est *issue d'un consortium*[25]*, d'industriels et universitaires français, appelé OpenQVT*.

L'union européenne a pris toute la mesure de l'importance stratégique des recherches autour de l'approche MDA. Aussi, afin de soutenir le secteur informatique considéré comme vital pour l'économie européenne, la Commission Européenne a fiancé le projet intégré ModeLWare (European Commission, Non daté). Le but était, pour la Commission Européenne, de soutenir la compétitivité et l'innovation du secteur informatique.

L'objectif principal du projet ModeLWare était de développer une solution basée sur l'Ingénierie Dirigé par les Modèles permettant d'augmenter de 15 à 20 % la productivité du développement logiciel. Ce projet avait aussi comme objectifs secondaires d'une part, d'aboutir à une solution industrielle et, dautre part, de garantir l'adoption de cette solution par l'industrie. Le consortium était constitué de 19 partenaires industriels et universitaires.

L'un des principaux résultats de ce projet est la conception et l'implémentation de la plate-forme MODELWARE Open MDD. Elle est destinée à assurer un service d'interopérabilité entre ateliers de génie logiciel, outils de langages de transformations, etc. Le schéma de la figure 25 présente une vue d'ensemble de cette plate-forme.

Le service d'interopérabilité s'appuie sur un modèle pivot appelé ModelBus qui assure l'interconnexion des plates-formes informatiques. Pour ce faire, des adaptateurs entre le langage de la plate-forme et le langage pivot ModelBus ont été conçus et développés.

Enfin au niveau français, les recherches autour de l'approche MDA sont soutenues par les pouvoirs publics au travers de :

- ⇨ Soit des actions spécifiques comme l'AS MDA, Action Spécifique du CNRS sur l'Ingénierie Dirigée par les Modèles.
- ⇨ Soit au travers de projets RNTL : ACCORD (Carrez, 2003), MoDAthéque (2004-2005), ModeFact, etc.

L'AS MDA a permis de mettre en place un dispositif de *veille technologique et de travaux de recherche* (Bézivin *et al.*, 2005) regroupant sept organismes de recherche français. L'un des objectifs des coordinateurs de cette AS étaient de ne pas *se limiter à des avancées technologiques guidées par l'OMG, mais au contraire s'élargir pour tenter d'établir une synergie entre les travaux de recherche présents et passés manipulant eux aussi des modèles* (Bézivin *et al.*, 2005).

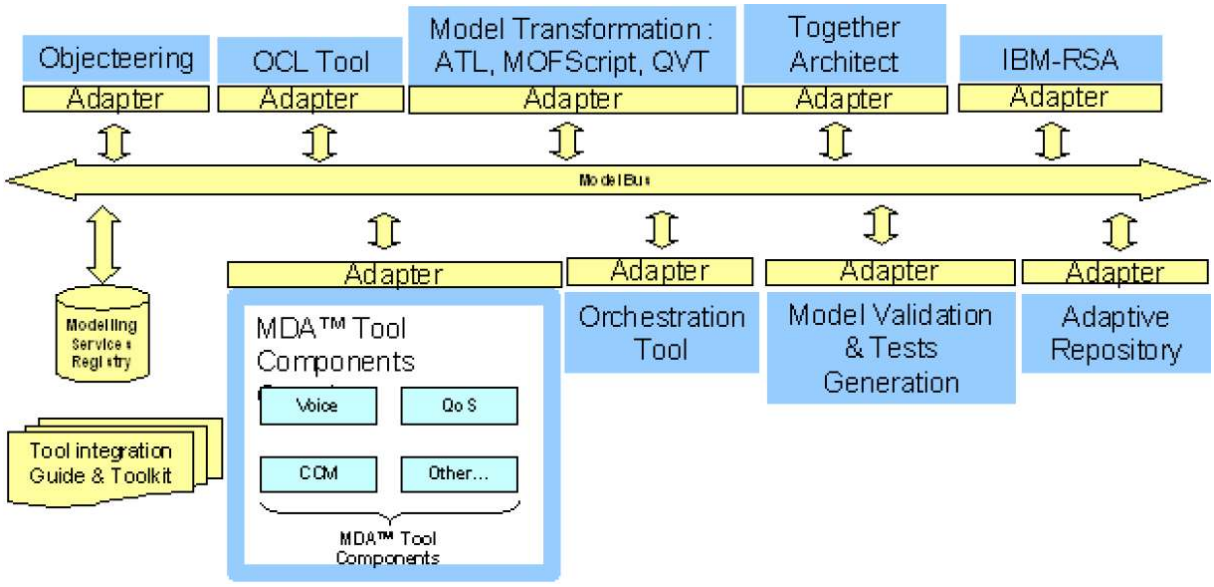


Figure 25 Vue générale de la plate-forme MODELWARE Open MDD (extrait (European Commission, Non daté)).

[25] Partenaires industriels : France Télécom, Thalès, Softeam et TNI-Valiosys – Partenaires universitaires : INRIA, LIP6 et LIFL (Belaunde *et al.*, 2004).

Ce groupe a rédigé un rapport faisant la synthèse des réflexions suscitées par l'approche MDA, rapport structuré en plusieurs chapitres. Chacun des chapitres traite d'une thématique particulière de l'Ingénierie Dirigée par les Modèles (Bézivin *et al.*, 2005 ; Blay-Fornarino *et al.*, 2005 ; Bouzeghoub, 2005 ; Favre *et al.*, 2005 ; Gérard *et al.*, 2005 ; Jézéquel *et al.*, 2005 ; Marvie *et al.*, 2005).

Les principaux langages de transformation de modèles actuellement développés par la recherche française sont : ATL[26], MTL[27] et TRL[28]. Le premier est un langage développé par l'équipe de recherche ATLAS qui implique l'INRIA, l'université de Nantes et TNI-Software. Ce langage permet la « manipulation » de métadonnées et la transformation de modèles en s'appuyant sur une technique de tissage (Bézivin *et al.*, 2003 ; Jouault *et al.*, 2006). Le deuxième est un langage développé par l'équipe TRISKELL regroupant le CNRS, l'Université Rennes I, l'INRIA et l'INSA de Rennes. C'est un langage permettant *l'écriture de programmes de transformations de modèles*. *Grâce à ce langage, les transformations peuvent être décrites dans un métamodèle pivot* (Macedo de Amorim, 2004 ; Silaghi *et al.*, 2004). Le troisième est une contribution à l'effort de standardisation des langages de transformations (QVT Project, Non Daté ; Sriplakich, 2003).

Ce rapide panorama des travaux internationaux de standardisation, des projets de recherche européens et français autour de l'approche MDA montre d'une part, l'intérêt de la communauté scientifique française pour ces thèmes et, d'autre part, l'importance stratégique et économique des transformations de modèle et des langages permettant de les écrire.

I.4 Comparaison d'un processus de développement traditionnel et d'un processus de développement appliquant l'approche MDA

Afin de confirmer ou démentir les allégations de l'OMG concernant le gain de productivité attendu avec l'approche MDA (cf. I.1), la société américaine *Compuware Corporation* a demandé au cabinet de conseil *The Middleware Company* de faire une étude comparative pour vérifier ces affirmations (The Middleware Company, 2003).

Pour ce faire, le cabinet conseil a demandé à deux équipes de niveau de compétences équivalent de développer une application de commerce électronique d'animaux domestiques *PetStore* : la première en mettant en œuvre un processus de développement traditionnel et la seconde en utilisant l'approche MDA. Le cahier des charges transmis aux deux équipes était le même. Le tableau 4 montre les durées planifiées et réalisées par ces équipes.

Durée / Équipe	Planifiée		Réalisée	
Traditionnelle	499 h		507 h	
MDA	442 h	-11%	330 h	-35%

Tableau 4 Comparaison du processus de développement traditionnel / approche MDA.

Les résultats parlent d'eux mêmes. ***Initialement évalué à 11%, le gain de productivité de l'équipe MDA a été triplé***. Selon l'un des programmeurs de l'équipe MDA, ce gain aurait pu être plus élevé car le temps de la réalisation aurait pu être réduit de 10 à 20 %. En effet, au démarrage du projet l'équipe MDA n'avait aucune expérience de cette approche et a dû, dans un premier temps, se former aux principes et acquérir la maîtrise des outils MDA.

Un autre résultat ressort de cette étude : ***l'équipe traditionnelle, tout au long du développement, a été amenée à corriger des bogues alors que l'équipe MDA n'a pas eu ce problème***. Les analystes du cabinet conseil encadrant l'étude attribuent cette absence de bogue à ***l'utilisation intensive de la génération automatique de code***.

[26] ATLAS Transformation Language
[27] Model transformation language
[28] Transformation Rules Language

II METHODES DE CONDUITE DE PROJET : PANORAMA SUCCINCT

Lorsque les trois *amigos*, Grady Booch, James Rumbaugh et Ivar Jacobson, ont conçu le langage UML, ils ont délibérément distingué le langage de modélisation de la méthode permettant de mener à bien le projet de développement d'une application (Fayet, 2002) afin de laisser libre chaque concepteur d'adopter la méthode la plus adaptée à son environnement professionnel.

Actuellement, il existe de nombreuses méthodes de conduite de projet qui sont classées en deux grandes familles : les méthodes dites ***traditionnelles*** et les méthodes ***agiles***.

II.1 Les méthodes traditionnelles

Les méthodes ***traditionnelles*** dérivent le plus souvent des méthodes de gestion de projet éprouvées en ingénierie industrielle ou dans le secteur du Bâtiment et des Travaux Public (Larman, 2002a). Il existe une multitude de méthodes traditionnelles pour développer une application informatique mais la plupart reposent sur quelques méthodes de référence :

- ⇨ Cycle en cascade et cycle en V.
- ⇨ Cycle en spirale.
- ⇨ RAD (Rapid Application Development).
- ⇨ UP (Unified Process).

Dans les paragraphes suivants, nous allons décrire quelques caractéristiques intéressantes de ces méthodes dont nous avons besoin pour la compréhension de notre contribution. Nous avons aussi décrit la méthode 2TUP qui est une méthode dérivée de la méthode Processus Unifié dont certains aspects rappellent l'approche MDA.

II.1.1 *Cycle en cascade - Cycle en V*

Ce ne sont pas des méthodes de conduite de projet au sens défini par (Jacobson *et al.*, 1999) car seul le cycle du processus de développement est décrit. La gestion du projet, la composition de l'équipe, le rôle des membres au sein de l'équipe, les relations entre l'équipe et le client, etc. ne sont pas pris en compte. Bien que très anciens, ces cycles de développement sont toujours utilisés car ils sont simples à mettre en œuvre dans des projets de taille modeste.

Le cycle de développement en *cascade* a été formalisé par Winston Royce en 1970 (Royce, 1970). Le processus de développement est linéaire au cours du temps comme peut l'être la construction du gros œuvre d'une maison où les fondations sont coulées avant de monter les murs et de poser la toiture. Selon Winston Royce, l'analyse et le codage sont deux phases incontournables qu'elle que soit la taille de l'application à développer. Elles constituent le cœur du processus de développement. Les autres phases deviennent indispensables lorsque la taille de l'application devient conséquente.

Selon ce cycle, les phases de développement se succèdent l'une après l'autre (cf. figure 26) sans *aucun mécanisme formel de rétroaction pour prendre en compte l'évolution des besoins des utilisateurs* (Guimond, 2005).

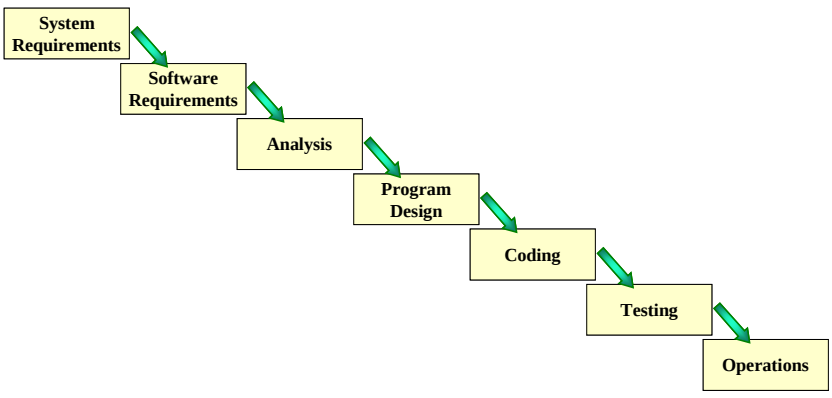


Figure 26 Cycle de développement en cascade (Royce, 1970).

L'inconvénient majeur lié à ce cycle est que la totalité de l'analyse est effectuée au début du projet et qu'ensuite le développement est réalisé étape après étape jusqu'à la livraison de l'application et ce sans aucune validation intermédiaire de la part des acteurs. Le risque sous-jacent est que l'analyse peut être accomplie de façon approximative ou imparfaite. Les développeurs sont alors en possession d'informations ou de consignes incomplètes ou erronées qui vont êtres utilisées au cours du développement. L'application a alors de fortes chances de dériver et de ne plus correspondre aux attentes des acteurs.

Cette situation n'est pas un cas d'école puisque, dans 49% des développements, l'application finale ne correspond pas aux attentes les acteurs ou ne répond pas aux besoins exprimés par ces derniers (cf. Introduction-II).

Selon (Muller *et al.*, 2000), le cycle en V est une représentation différente du cycle en cascade qui permet d'exprimer *le fait que le développement des tests et le développement du logiciel sont effectués de manière synchrone*.

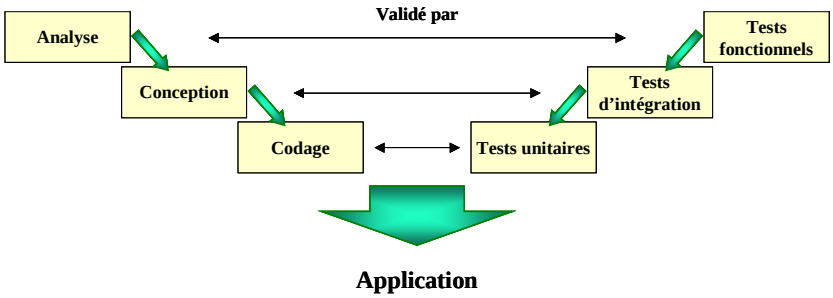


Figure 27 Cycle de développement en V (Muller *et al.*, 2000).

Ces deux cycles de développement sont les plus connus car historiquement ceux sont les premiers à avoir été utilisés pour le développement d'applications conséquentes. Ils sont également connus car les méthodes traditionnelles conçues à postériori avaient pour but essentiel de résoudre les inconvénients inhérents à ces cycles.

II.1.2 *Cycle en spirale*

Le cycle de développement en *spirale* a été introduite par Barry Boehm en 1988 (Boehm, 1988). L'objectif visé en adoptant le cycle en spirale (cf. figure 28) était de diriger le développement par les risques et non plus par la documentation ou le code comme c'était le cas jusqu'alors.Avec ce cycle, Barry Boehm adoptait le concept de *modèle évolutif* de (McCracken *et al.*, 1982) et proposait un processus en quatre itérations nommées *Round*. Ce cycle se caractérise par une succession de quatre phases :

- La définition des objectifs ainsi que l'identification des alternatives d'implémentation possibles et des contraintes.
- L'évaluation des risques induits par l'alternative d'implémentation. Le prototypage est un des moyens d'évaluation possible.
- La conception et le développement incluant au dernier Round les tests unitaires, d'intégration et fonctionnels.
- La validation du produit de la phase par les acteurs du domaine et la planification du Round suivant.

L'année suivante, (Boehm *et al.*, 1989) proposaient une théorie de gestion du développement des applications informatiques. Cette théorie, appelée Théorie-W, avait pour objectif que tous les partenaires d'un projet soient gagnants.

La fusion du cycle en spirale et de la Théorie-W ne sera effective qu'en 1994 (B. W. Boehm *et al.*, 1994). La méthode *Next Generation Process Model* (NGPR) issue de cette fusion est présentée comme une extension au cycle de développement en spirale. La méthode NGPR sera affinée en 1998 et renommée *WinWin Spiral Model* (Boehm *et al.*, 1998).

Le but poursuivit par Barry Boehm avec ces travaux de recherche était de maximiser le degré de satisfaction de l'ensemble des acteurs impliqués dans un projet de développement et en particulier de prendre en compte les exigences des acteurs.

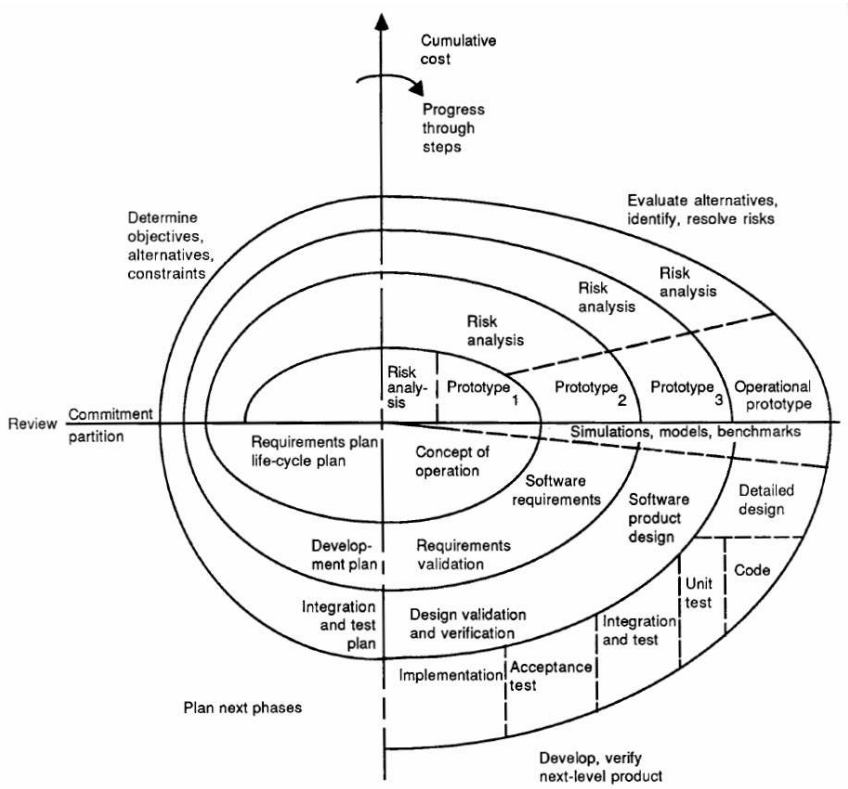


Figure 28 Cycle de développement en spirale (extrait (Boehm, 1988)).

II.1.3 *Méthode Rapid Application Development (RAD) (Extrait de l'Annexe II-I)*

Le fondateur de cette méthode est James Martin auteur de nombreux ouvrages dont celui portant le même nom que la méthode : *Rapid Application Development* (Martin, 1991). Cette méthode a été adaptée au contexte français et promue en France entre autres par Jean-Pierre Vickoff. Il est auteur de l'ouvrage *le Développement d'application client-serveur* (Vickoff, 1995) et du site Internet http://www.rad.fr/ (Vickoff, Site Web non daté).

L'objectif de cette méthode de conduite de projet est de changer radicalement le processus de développement afin de réaliser *très rapidement* des applications *de qualité*. Une particularité de cette méthode est de mettre au même niveau les quatre composantes nécessaires au développement d'une application : l'*équipe*, la *méthodologie*, la *gestion* et les *outils* (CASEMaker Inc., 2000 ; University of California, 1997b).

Selon les partisans de cette méthode, si l'une des composantes est défaillante, la vitesse de développement ne sera pas maximale. Cela sous-entend qu'il est primordial d'arriver à trouver un équilibre et à mettre en place une coordination parfaite des quatre composantes. C'est la principale tâche du chef de projet. Cet équilibre est symbolisé par un parallélépipède posé sur quatre piliers identiques (cf. figure 29).

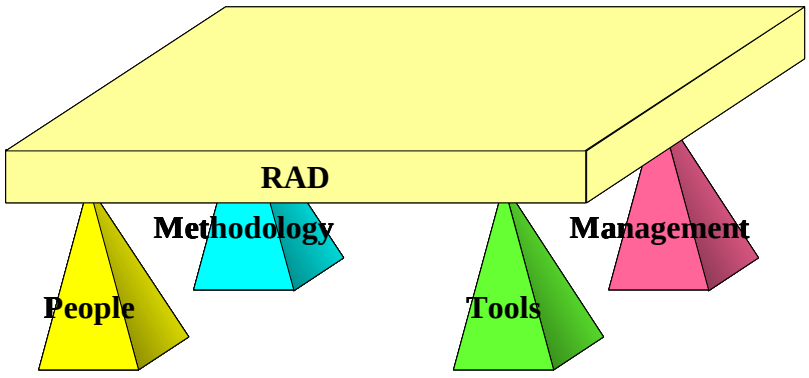


Figure 29 Les quatre composantes de la méthode RAD (University of California, 1997b).

Selon son auteur, la durée optimale d'un développement s'appuyant sur la méthode RAD est de 90 jours (Martin, 1991). Quelquefois, cette durée peut être portée à 120 jours. De ce fait, la méthode RAD est bien adaptée au développement d'applications de petite taille.

Le cycle de développement de cette méthode se compose de cinq phases : *Initialisation*, *Cadrage*, *Design*, *Construction* et *Finalisation*. La figure 30 illustre le cycle de développement de la méthode RAD et indique quelques tâches essentielles à accomplir au cours de chacune des phases. Le paragraphe I.2 de l'Annexe II fournit une description plus complète des tâches à réaliser.

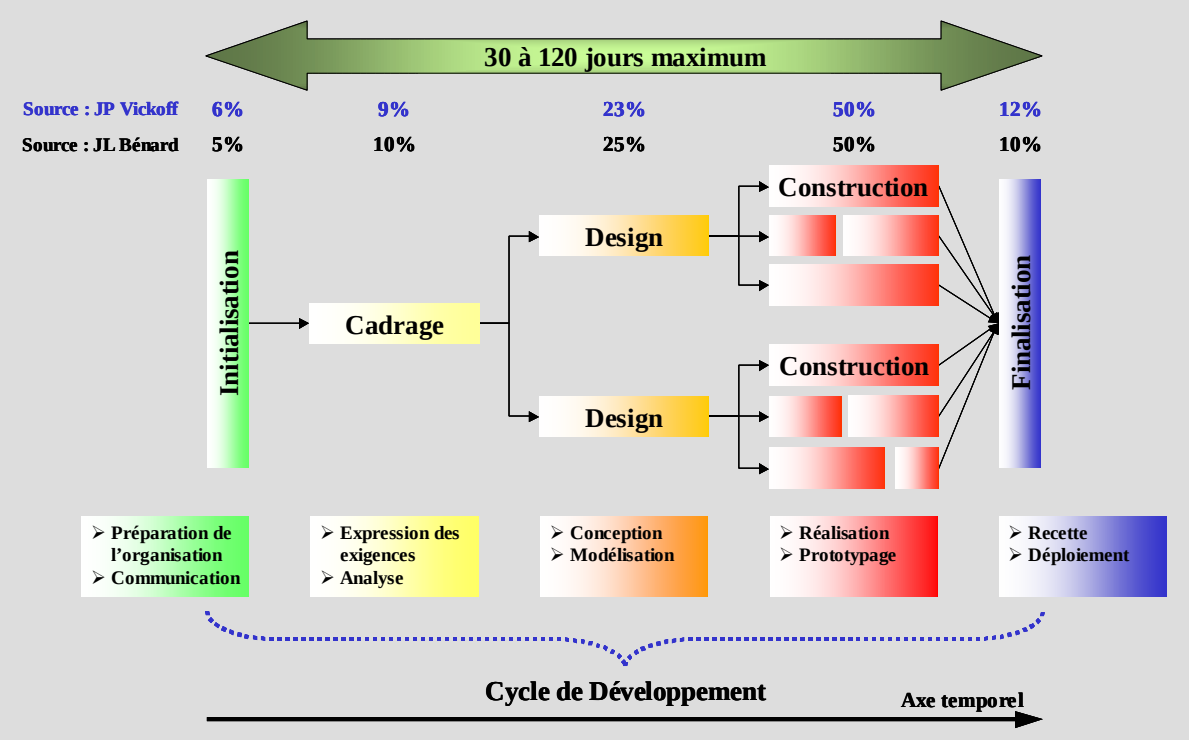


Figure 30 Cycle de développement d'un projet RAD (Bénard, 2002a ; Vickoff, 2000).

Bien évidemment, les acteurs participent à la phase de Cadrage pendant laquelle ils décrivent les grandes lignes du système à modéliser et où ils expriment leurs exigences. Le principal objectif de la phase de Design reste l'approfondissement des spécifications. En phase de *Construction*, les acteurs du domaine participent à des réunions plénières, appelées *Focus*, qui sont l'occasion de présenter l'avancement du projet et de le valider dans toutes ses dimensions. La participation des acteurs tout au long du cycle de développement simplifie la validation finale de l'application.

II.1.4 *Méthode Processus Unifié (UP) (Extrait de l'Annexe II-II)*

Les idées qui donneront naissance à la méthode de conduite Processus Unifié datent de la fin des années soixante alors qu'Ivar Jacobson travaillé chez Ericsson (Jacobson, 2003). La méthode Processus Unifié est la *synthèse des meilleures pratiques de développement logiciel sur trois décennies dans des domaines variés*[29] (Jacobson *et al.*, 1999). Bien qu'elle soit totalement dissociée du langage UML (Fowler, 2004b ; Millan *et al.*, 2004), elle est adossée à ce langage de modélisation car les auteurs d'UML, Grady Booch, Ivar Jacobson et James Rumbaugh, ont participé à sa formalisation.

La méthode Processus Unifié repose sur quatre principes suivants : le processus de développement doit être ***conduit par les cas d'utilisation***, ***itératif et incrémental***, ***centré sur l'architecture*** et ***piloté par les risques*** (Larman, 2002b ; Muller *et al.*, 2000 ; Roques *et al.*, 2002).

Le principe du ***développement conduit par les cas d'utilisation*** va permettre de recentrer à tout moment le développement d'une application sur les besoins des acteurs.

[29] Télécommunications, aéronautique, défense, transport, etc.

Le but de tout projet informatique est de reproduire le comportement d'un système afin de satisfaire un besoin. Pour ce faire, il faut d'une part, capturer la connaissance qu'ont les acteurs du système et, d'autre part, identifier les besoins et les exigences des acteurs.

Les cas d'utilisation sont l'objet technique qui permet de représenter le comportement du futur système, ce que doit faire le système mais aussi qui et quoi interfèrent avec le système. De ce fait, une fois implémenté, un cas d'utilisation répond aux besoins des acteurs. Si ce n'est pas le cas, c'est qu'il a mal été défini.

Les cas d'utilisation doivent donc être décrits au début du projet et tout développement doit être « au service » du fonctionnement et/ou des exigences exprimés dans le cas d'utilisation. Il devient alors un outil de planification.

Le principe du ***développement itératif et incrémental*** est sans aucun doute le plus important (Larman, 2002a ; Roques, 2002). Le qualificatif d'itératif s'applique à *la gestion d'une succession de versions exécutables* (Booch *et al.*, 2000 ; Muller *et al.*, 2000) alors que celui d'incrémental porte sur l'évolution du modèle. Les nouveaux concepts et les nouvelles spécifications ajoutées dans le modèle au cours de l'itération constituent l'incrément. Un incrément est donc la différence entre l'état du modèle en début d'itération et en fin d'itération (Roques *et al.*, 2002). Selon (Jacobson *et al.*, 1999), *le résultat d'une itération est un incrément.*

La méthode de conduite de projet Processus Unifié prévoit que le développement d'une application soit découpé en itérations de courte durée, 2 à 4 semaines.

A chaque itération, les spécifications sont validées par les acteurs (Roques *et al.*, 2002) et un sous-système exécutable est produit. De ce fait, le processus itératif est le meilleur garant d'une part, pour produire un exécutable conforme aux spécifications et aux exigences des acteurs améliorant de fait la qualité du produit final et, d'autre part, pour permettre de suivre *l'avancement du projet* (Bénard, 2002b), le *contrôle des coûts et des délais* (Roques *et al.*, 2002).

Le processus de développement itératif et incrémentale facilite la capitalisation des connaissances du domaine parce qu'il est toujours possible d'ajouter des concepts nouveaux ou bien oubliés par un acteur lors d'une itération précédente. D'autre part, ce type de processus permet de "mixer" les connaissances d'acteurs ayant des points de vue différents sur le même concept (Booch *et al.*, 2000). Il permet aussi la capitalisation *des enseignements des cycles précédents* (Bénard, 2002b).

Le principe du ***développement centré sur l'architecture*** a été introduit par les auteurs de la méthode afin de souligner l'importance du travail de structuration de l'application en systèmes et sous-systèmes que doit impérativement faire le concepteur. La structuration d'une application est nécessaire pour les quatre raisons suivantes (Jacobson *et al.*, 1999) :

- ⇨ Compréhension du système : souvent, un acteur n'a qu'une connaissance partielle du système et c'est l'assemblage de ces connaissances partielles qui permet de reconstituer la totalité du système.
- ⇨ Organisation du développement : pour les projets d'une certaine taille, l'application est décomposée afin de pouvoir répartir la charge de développement.
- ⇨ Réutilisation de composants existants : pour une entreprise, le développement doit être réalisé dans les meilleurs délais et à moindre coût. Cette contrainte conduit tout naturellement à réutiliser les composants logiciels réalisés au cours de projets antérieurs.
- ⇨ Évolution du système : dans une application bien structurée, la correction de bogues ou la réalisation d'une nouvelle version sera plus facile que dans une application non structurée.

Selon (Roques, 2002 ; Roques *et al.*, 2002), *la structuration en sous-systèmes ne doit pas être une simple description du système sous forme graphique ou textuelle mais elle doit être matérialisée*. Cette exigence introduit par Pascal Roques impose l'utilisation un atelier de génie logiciel lors de la modélisation.

Le principe du ***développement piloté par les risques*** a été adopté car ces derniers peuvent mettre en difficulté un projet voire le faire échouer (Jacobson *et al.*, 1999). Le pilotage par les risques du développement d'une application permet de prendre en compte des problèmes très tôt voire à leur origine et de les traiter par anticipation.

Il existe plusieurs catégories de risques mais les risques les plus importants sont d'une part, les *risques financiers* et les *risques commerciaux* (Muller *et al.*, 2000) et, d'autre part, les *risques non techniques* et des *risques techniques* (Jacobson *et al.*, 1999).

Concernant les risques financiers, la question fondamentale est de s'assurer que le commanditaire de l'application a les moyens financiers pour mener à terme la réalisation de l'application.

Les risques commerciaux sont liés au positionnement commercial de l'application, au niveau de concurrence du marché, à la stratégie commerciale du commanditaire, etc.

Les risques non techniques sont essentiellement liés à la gestion du projet : absence de compétence dans des domaines ou des technologies clés du projet, accepter un projet avec des délais souhaités/imposés par le client difficiles à tenir, etc.

Les risques techniques sont quant à eux induits par l'introduction de nouvelles technologies dans le développement de l'application, par une bonne évaluation des performances requises (les contraintes fonctionnelles d'un serveur soumis à quelques connexions par heure ou à plusieurs milliers de connexions ne sont pas les mêmes), etc.

Dans la méthode Processus Unifié, le cycle de développement d'une application (cf. figure 31) est structuré en quatre phases : l'***inception***, l'***élaboration***, la ***construction*** et la ***transition*** (Bénard, 2002b ; Jacobson *et al.*, 1999 ; Kruchten, 1999 ; Larman, 2002a).

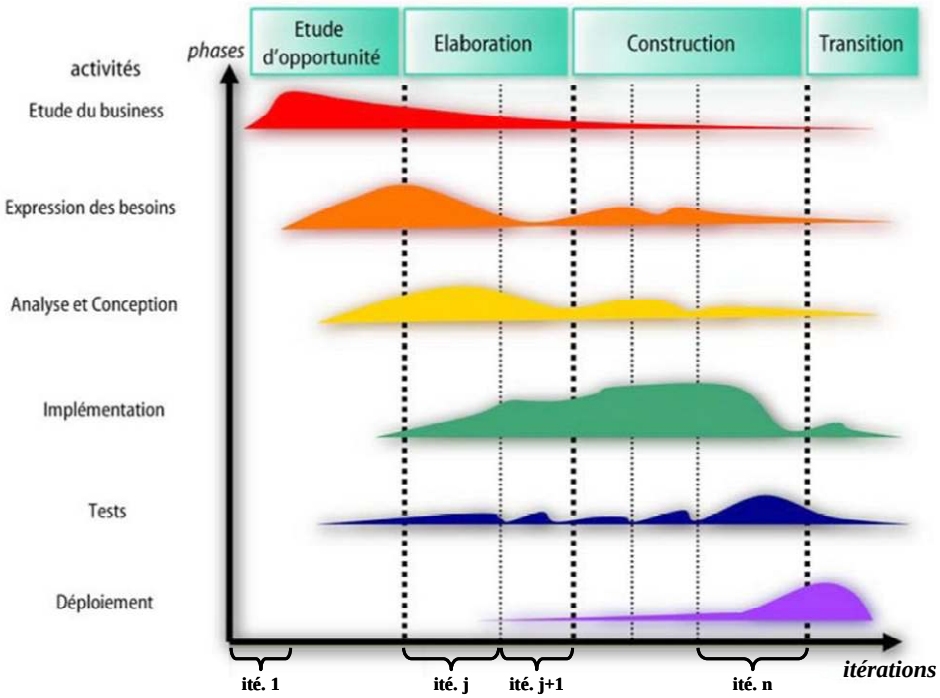


Figure 31 Activités et intensité de production en fonction de l'avancement du projet (Booch *et al.*, 2000 ; Jacobson *et al.*, 1999 ; Kruchten, 1999).

L'***inception*** est la phase d'ébauche du projet (Booch *et al.*, 2000). Dans la phase d'***élaboration***, l'identification et la description des besoins et l'évaluation des risques sont effectuées. La phase de ***construction*** consiste à concevoir et à implémenter les cas d'utilisation. Craig Larman résume cette phase par la métaphore : *on met de la chair au squelette créé lors de la phase d'élaboration* (Larman, 2002a). La ***transition*** est la phase de transfert de l'application vers les utilisateurs (Booch *et al.*, 2000 ; Roques, 2002) et de son déploiement dans un environnement opérationnel (Jacobson *et al.*, 1999).

Au cours de ces quatre phases, le concepteur d'une application met en œuvre des *activités*[30] (Bénard, 2002b ; Muller *et al.*, 2000) très diverses comme l'*étude d'opportunité*, l'*expression des besoins*, l'*analyse*, la *conception*, l'*implémentation*, les *tests* et le *déploiement* (cf. figure 31-Axe vertical).

La figure 31 montre aussi que les activités peuvent être itérées plusieurs fois au sein d'une phase. Afin de rappeler le caractère itératif du processus de développement, les activités sont le plus souvent représentées en cercle (cf. figure 32).

Au cours d'un processus de développement itératif, la participation des acteurs est concentrée au début de l'itération pour exprimer les exigences et réaliser l'analyse et en fin d'itération pour valider le produit du cycle. Selon l'avancement du projet, le produit peut être des spécifications ou une version de l'application.

[30] Certains auteurs (Larman, 2002a ; Roques, 2002) préfèrent utiliser le terme de discipline. Auparavant, le terme *Workflow* était utilisé (Booch *et al.*, 2000) mais il a été abandonné en 2000 et remplacé par le terme *discipline* suite aux travaux de standardisation de l'OMG (Larman, 2002a)

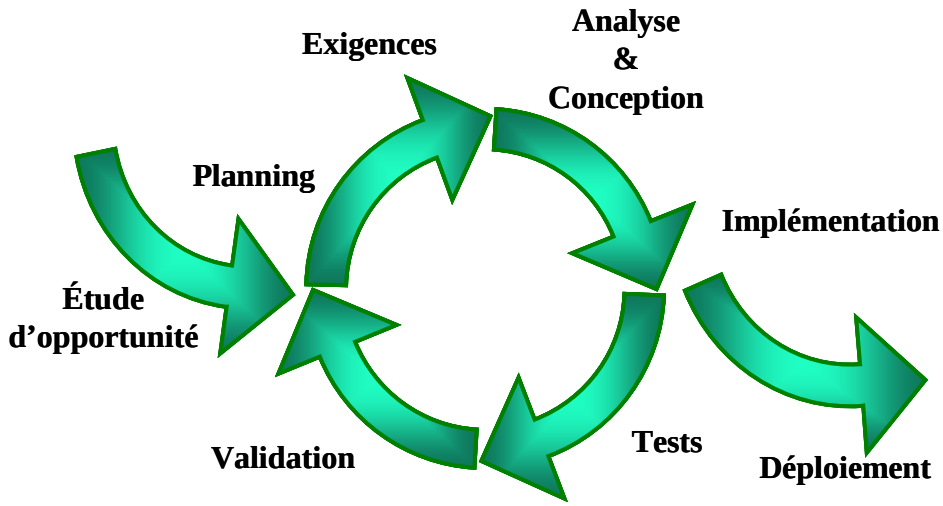


Figure 32 Cycle itérative de la méthode Processus Unifié (Kruchten, 1999).

II.1.5 *Méthode 2 Track Unified Process (2TUP)*

La méthode 2TUP est une méthode Processus Unifié qui a été conçue par (Roques *et al.*, 2002) pour *apporter une réponse aux contraintes de changement continuel imposées aux systèmes d'information de l'entreprise* (Roques *et al.*, 2002).

Cette méthode est fondée sur le constat que les corrections et les évolutions d'un système peuvent être de nature fonctionnelle ou technique. Ces deux natures d'évolution constituent deux axes d'évolution possible d'une application. (Roques *et al.*, 2002) ont introduit cette dichotomie sur la base de leurs propres expériences suite au constat que les évolutions sollicitées par un commanditaire sont de trois types :

- ⇨ Des évolutions liées au métier de l'entreprise ou à la thématique ; dans ce cas les aspects fonctionnels (branche de gauche) sont les seuls concernés. Par exemple, un changement de réglementation administrative du domaine peut entraîner l'ajout de fonctionnalités au système, la suppression de certaines d'entre elles et/ou leurs modifications.
- ⇨ Des évolutions concernant l'architecture du système alors les aspects techniques (branche de droite) sont les seuls impliqués. La distribution sur des serveurs distants d'une base de données centralisée n'affectera en aucun cas les aspects fonctionnels mais seulement l'architecture du système.
- ⇨ Le troisième type résulte de la combinaison des deux premiers. Une entreprise ou une organisation peut souhaiter procéder à une remise à niveau de son système en intégrant les changements de réglementation et en offrant un service sur Internet inexistant auparavant. Dans ce cas, les deux branches du Y vont concourir à l'évolution globale du système indépendamment l'une de l'autre, indépendance qui permet de les traiter en parallèle ou l'une après l'autre selon l'urgence des besoins.

Selon les auteurs de la méthode 2TUP, l'application est le résultat de la fusion des produits issus des deux axes. Cette idée de fusion est à l'origine de la représentation du processus de développement par la lettre Y. Les deux axes d'évolution sont représentés par les branches supérieures de la lettre Y et le résultat de la fusion par la branche verticale (cf. figure 33).

La branche de gauche porte sur les aspects fonctionnels liés à la spécification et à la capture des besoins des acteurs du domaine alors que la branche de droite concerne la capture des besoins techniques du système.

La première *capitalise la connaissance métier* ou thématique du domaine alors que la seconde *capitalise le savoir-faire technique* du concepteur ou de l'entreprise. Ces deux types de connaissances se rejoignent dans le modèle de conception préliminaire pour constituer la branche verticale de l'Y.

Les différentes alternatives pour réaliser chacun des composants sont ensuite étudiées. Le résultat est le modèle de conception détaillée de l'application.

Les composants sont ensuite codés et testés unitairement et intégrés dans l'application. En recette, l'application produite est validée par les acteurs.

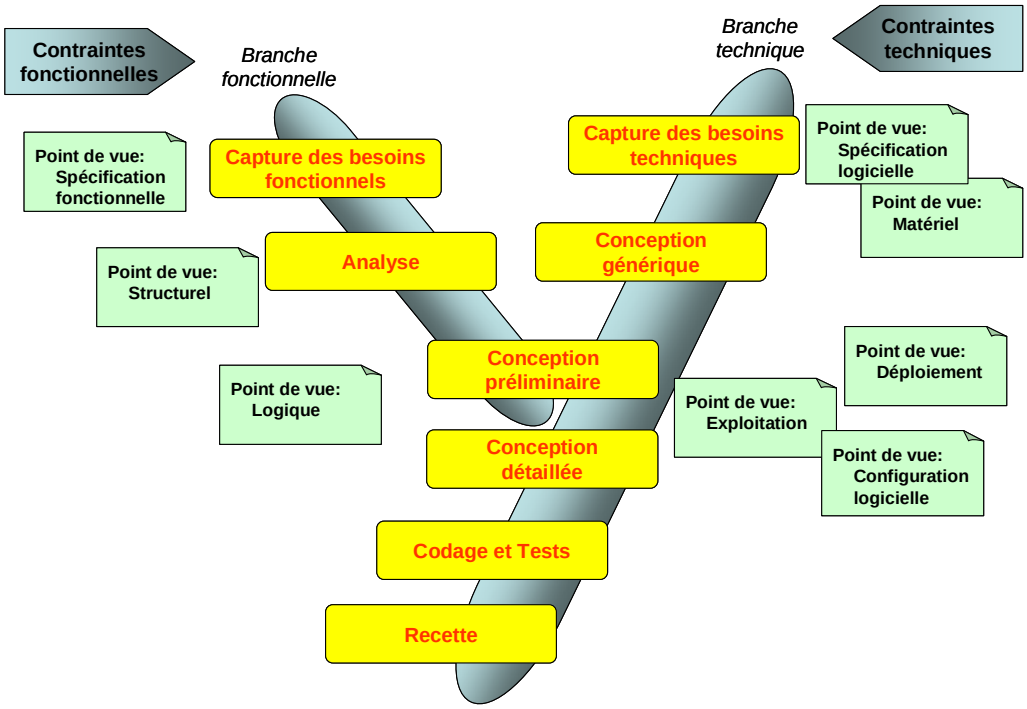


Figure 33 Le processus de développement en Y de la méthode 2TUP.

II.2 Les méthodes agiles

II.2.1 *Généralités*

Les méthodes agiles ont été développées *en réaction aux méthodes dites "merisiennes" (cycle en V, etc.) qui préconisent des cycles de développement long* (Bénard, 2001).

Les idées, les principes et les pratiques de la plupart de ces méthodes étant très proches, leurs auteurs se sont réunis en février 2001 afin d'identifier les points de convergence. Ces derniers étant très nombreux, les auteurs de ces méthodes ont convenu d'une part, de structurer ce courant de pensée en créant un consortium baptisé *Alliance Agile* et, d'autre part, de rédiger le *Manifeste Agile* qui constitue l'acte fondateur de ce courant de pensée.

Cet acte définit une nouvelle philosophie de développement des logiciels qui est fondée sur quatre valeurs fondamentales :

- ⇨ Les ***individus et leurs interactions*** sont privilégiés aux processus et aux outils.
- ⇨ Les ***logiciels immédiatement disponibles*** sont préférés à une documentation exhaustive.
- ⇨ La ***collaboration avec le client*** est encouragée à tout acte contractuel.
- ⇨ La ***réactivité face aux changements*** est favorisée à un plan préétabli.

Ces valeurs sont déclinées en une douzaine de principes eux même déclinés en pratiques. (Boehm *et al.*, 2004) comparent plusieurs méthodes agiles et la méthode Processus Unifié suivant une grille de lecture commune constituée d'une quinzaine de critères portant sur le champ d'application de la méthode, l'étendue du cycle de développement et les contraintes liées à chacune d'elles.

Une contrainte commune à toutes ces méthodes est l'exigence d'une participation forte des acteurs du domaine à modéliser, participation pouvant aller jusqu'à la présence en permanence d'un acteur au sein de l'équipe de développement.

Face à la multitude de méthodes agiles, nous avons approfondi la méthode eXtreme Programming qui est la méthode agile probablement la plus répandue et dont certains concepts ont été mobilisés dans le présent travail recherche.

II.2.2 *Méthode eXtreme Programming (XP) (Extrait de l'Annexe II-III)*

La méthode eXtreme Programming a été conçue par Kent Beck, Ward Cunningham et Ron Jeffries, tous trois experts en développement logiciel et souvent surnommés *les trois extremos*. L'acte de fondateur de cette méthode est la publication de l'ouvrage de Kent Beck *Extreme Programming Explained* (Beck, 2000). Elle a été formalisée et mise en œuvre pour la première fois chez Chrysler lors du développement du logiciel C3 Payroll[31].

Son originalité par rapport aux méthodes traditionnelles réside dans l'organisation légère des activités connexes au développement a contrario des méthodes de développement traditionnelles qui privilégient une planification forte.

La méthode eXtreme Programming rassemble sa communauté autour de quatre valeurs *prenant en considération à la fois les enjeux commerciaux et les aspects humains du projet* (Bénard, 2002c) :

- La ***communication*** : les concepteurs de cette méthode ont érigé la communication en valeur fondamentale car de nombreux projets de développement ont échoué à cause d'une mauvaise communication entre les différents acteurs du projet : commanditaire, utilisateurs, chef de projet, équipe de développement, etc.
- La ***simplicité*** : l'objectif premier de tout développement étant la satisfaction du commanditaire, la simplicité du développement est une valeur vitale pour obtenir un code compréhensible et ne pas développer des gadgets inutiles au commanditaire.
- Le ***feedback*** : c'est une valeur essentielle aux développeurs qui doivent avoir à tout moment un retour d'information rapide et de qualité de la part du commanditaire et de tous les autres membres de l'équipe.
- Le ***courage*** : cette valeur est une qualité humaine incontournable d'après les auteurs de la méthode car ils estiment que les acteurs utilisant cette méthode doivent avoir la détermination et la bravoure de dire les choses telles qu'elles sont et de prendre des décisions difficiles comme *changer une structure de code ou de le jeter* (Cros, 2001).

Les quatre valeurs eXtreme Programming sont déclinées en principes dont les principaux sont d'encourager les processus de *feed-back rapide*, de favoriser *l'évolution incrémentale* du code, de rechercher la *simplicité* dans le code produit et de viser la *qualité* du code.

Pour mettre en œuvre cette méthode, les concepteurs d'eXtreme Programming assimilent le fonctionnement et les relations de l'équipe à un *jeu de rôle* (Beck, 2000 ; Bénard *et al.*, 2002). Au sein de l'équipe, les rôles suivants doivent être impérativement assurés : le rôle de Programmeur, celui de Client, du Testeur, du Tracker, du Coach, du Consultant et du Manager. Les auteurs ont décrit et formalisé les responsabilités de chacun de ces rôles et leurs relations (cf. Annexe II-III.2).

Afin de les respecter, les auteurs ont déclinés les principes de la méthode en douze pratiques que l'équipe de développement doit appliquer. Dix de ces pratiques sont décrites succinctement ci-dessous suivies des deux pratiques dont la description a été détaillée car elles interviennent dans notre contribution :

- Planification : identification et assimilation des concepts à implémenter pour réaliser une fonctionnalité, évaluation des coûts en termes de durée, décision collective des concepts à implémenter au cours de l'itération et réalisation de la fonctionnalité.
- Livraisons fréquentes : la durée d'une itération varie de 1 à 3 semaines selon le type de projet et le fonctionnement de l'équipe ; la date de livraison est quant à elle décidée par le client.
- Métaphore : adopter une métaphore pour désigner le système à développer et, si besoin est, les sous-systèmes.
- Conception simple : ne pas implémenter des fonctionnalités non demandées par le client, préférer des technologies simples et éprouvées à des technologies nouvelles et non maîtrisées.
- Généralisation des tests : conception de tests pour chacune des fonctionnalités à implémenter mais aussi pour chacun des composants. Le code d'un test doit être impérativement écrit avant le codage de la fonctionnalité ou du composant.
- Remaniement : pour un équipe eXtreme Programming, le code représente le livrable le plus fiable aussi il doit être de qualité. Pour ce faire, un code doit être parfaitement compréhensible et bannir la duplication de code. Le remaniement fréquent et la factorisation du code permettent d'y parvenir.

[31] C3 : Chrysler Comprehensive Compensation, projet destiné à assurer le payement sans faille des employés de chez Chrysler au passage de l'an 2000 (Jeffries, 2004).

- ⇨ Programmation en binôme : accroître la communication au sein de l'équipe, maintenir un niveau de production plus soutenu et une meilleure appropriation du code de l'application.
- ⇨ Responsabilité collective du code : le code n'appartient pas un membre en particulier mais à l'équipe ; de ce fait, les membres de l'équipe ont droit de regard sur un code et peuvent l'améliorer s'ils ont une meilleure solution.
- ⇨ Rythme durable : le rythme de développement doit être soutenu dans la durée. Pour ce faire, la durée de travail doit être raisonnable (40 h par semaine) et le recours à des heures supplémentaires doit être évité. Si ce n'est pas le cas, la composition de l'équipe ou l'envergure du projet doivent être réexaminées.
- ⇨ Règles de codage : adopter des règles communes pour présenter le code, pour nommer des concepts, pour décrire une fonctionnalité ou un composant, etc.

La pratique de l'***Intégration continue*** consiste à incorporer ***« en continu »*** le code développé dans la version commune de l'application. Cette pratique permet à l'équipe de disposer d'un livrable de l'application en permanence. Cette pratique limite fortement la phase d'intégration des méthodes traditionnelles, phase au cours de laquelle les anomalies ou les dysfonctionnements dus à des lacunes de spécification apparaissent .

Dans un projet eXtreme Programming, la fréquence d'intégration est au maximum journalière mais peut être horaire, cela dépend uniquement de la complexité de la fonctionnalité ou du composant. Si une fonctionnalité ou un composant n'est pas codé dans la journée, le lendemain il est scindé afin de faciliter sa réalisation.

En pratiquant l'intégration continue, les binômes mettent en continu leurs développements à la disposition des autres membres de l'équipe. Ces derniers peuvent alors interagir avec le code en effectuant des tests, en le remaniant, etc. Cette pratique contribue donc à améliorer la qualité globale de l'application.

Lors de l'intégration, les tests unitaires et de recette jouent un rôle très important car ils vont permettre de s'assurer que la cohérence du code produit sur une machine locale reste intacte dans la version commune. Dans le cas contraire, le binôme, avec l'aide des autres membres de l'équipe, doit modifier et remanier le code tant que tous les tests ne sont pas positifs.

La pratique du ***Client sur site*** consiste à favoriser la communication entre le client et les programmeurs en accueillant en permanence le client ou son représentant au sein de l'équipe. Cette pratique garantit une forte réactivité et un feedback élevé. Le but principal de cette pratique est d'apporter une solution à l'absence de spécifications détaillées.

La principale tâche du client est la rédaction des scénarios qui vont permettre de coder les fonctionnalités de l'application. Une seconde tâche tout aussi importante est la détermination des tests de recette que le testeur doit implémenter pour valider une fonctionnalité. Le client intervient en fixant les priorités entre fonctionnalités, en précisant les spécifications qui n'ont pas été définies auparavant ou qui sont restées floues lors des précédentes discussions, etc.

Sa présence dans l'équipe lui permet de voir le résultat immédiat de son travail de spécifications et d'apprécier l'évolution de l'application. Cette proximité lui permet aussi d'évaluer rapidement la pertinence de ses spécifications. Si le projet diverge ou prend du retard, il s'en rendra compte immédiatement.

III CONCLUSION

III.1 Approche MDA

Les deux idées maîtresses de l'approche MDA sont : la séparation des spécifications au sein de deux voire trois modèles et la mise en œuvre de transformations de modèle afin d'automatiser les évolutions. L'objectif final étant d'automatiser complètement le processus de développement, parfois appelé ***Full MDA*** (Kleppe, 2004 ; Softeam, 2005). Ces deux idées nous ont séduits car elles répondent à nos besoins. La séparation des spécifications au sein de deux voire trois modèles est une réponse à l'objectif élémentaire de *capitalisation des connaissances* que nous nous sommes fixé (cf. Introduction-IV).

Nous avons adopté cette approche malgré les réticences exprimées par certains milieux industriels. En effet, les dirigeants des entreprises du secteur informatique sont souvent séduits par l'approche MDA mais beaucoup restent sur l'expectative comme le montre l'entrefilet de Pierre Tran dans le Monde Informatique. Le titre résume bien la situation : *Les patterns, oui. MDA, non* (Tran, 2005).

Dans cet article, Pierre Tran rapporte que *lorsque les entreprises entendent parler de MDA, elles sont intéressées. Mais lorsqu'elles en réalisent toutes les implications, elles s'enfuient. Pour elles, MDA est rigide, lent et difficile à implémenter.* Cette perception est issue de deux études[32] publiées par la société Forrester Research et *menées sur 389 organisations en collaboration avec le magazine américain Application Development Trends* (Tran, 2005).

L'une des implications majeures de l'approche MDA est la gestion et le maintien en cohérence de plusieurs modèles. C'est un point que nous avons résolu dans le cadre de cette recherche.

Les idées de transformation de modèle de la communauté MDA permettent d'envisager d'une part, de gérer et maintenir en cohérence plusieurs modèles automatiquement et, d'autre part, d'implémenter les transformations faisant évoluer un modèle depuis l'analyse jusqu'à l'implémentation.

Pour atteindre l'objectif élémentaire de mettre en œuvre un *processus de développement permettant le prototypage rapide en séance d'analyse* avec l'*outil d'aide à la conception de Systèmes d'Information Géographique*, nous avons été amené à définir une panoplie de transformations automatisant l'évolution du modèle depuis l'analyse jusqu'à l'implémentation[33].

Au démarrage de la présente recherche en octobre 2000, tous les travaux de recherche autour des langages de transformation de modèle venaient de commencer ou n'avaient pas encore débuté (QVT). Aussi, la plupart de ces travaux de recherche présentaient l'inconvénient de ne proposer que des plates-formes de transformation de modèle incomplètes.

Nous avons exploré d'autres voies car l'utilisation de plates-formes incomplètes n'était ni compatible avec l'objectif élémentaire de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique* ni acceptable avec l'objectif élémentaire d'utiliser un *processus de développement permettant le prototypage rapide en séance d'analyse*, donc en présence d'acteurs.

Ces langages présentaient souvent un second inconvénient qui était que la plate-forme d'exécution des transformations de modèle était indépendante de l'atelier de génie logiciel dans lequel le modèle était saisi par le concepteur. Il était alors nécessaire d'exporter le modèle via un langage pivot, XMI le plus souvent, de réaliser la transformation dans la plate-forme externe et d'importer ensuite le résultat dans l'atelier de génie logiciel.

Le mécanisme d'import/export d'un modèle vers une plate-forme externe de transformation de modèle n'était pas compatible avec l'objectif élémentaire d'utiliser un *processus de développement permettant le prototypage rapide en séance d'analyse*. C'est la raison pour laquelle nous avons implémenté l'ensemble des transformations au sein d'un atelier de génie logiciel doté d'un langage de métaprogrammation.

III.2 Méthode de conduite de projet

Dans tout projet, la méthode de conduite est fondamentale et souvent préfigure les chances de succès du projet. La méthode définit en particulier la participation des acteurs (commanditaires, experts, utilisateurs, etc.), gage de réussite du projet. En effet, il est impensable de faire l'analyse d'un système et de développer une application sans leurs expertises et sans l'expression de leurs besoins.

L'analyse des méthodes du paragraphe II montrent que la participation des acteurs peut être décrite par deux autres variables : l'***intensité*** de participation et la ***fréquence*** de participation.

Dans les processus basés sur un cycle en cascade, les acteurs font l'analyse en début de projet et la conception et l'implémentation se poursuivent au sein de l'équipe sans leur participation jusqu'à la présentation aux acteurs du produit final en vue de sa validation. ***La participation des acteurs est concentrée au début du cycle et son intensité peut être évaluée à plusieurs jours. Une seconde participation intervient à la réception de l'application. La fréquence est égale à 2.*** Avec ce type de processus, il n'est pas rare que l'application dérive ou soit un échec.

Les processus de développement itératifs (Processus Unifié par exemple) impliquent les acteurs au démarrage du projet pour fixer les grandes lignes du projet. Ensuite, ils interviennent au début de chaque

[32] Model-Driven Matters More Than MDA - Pattern-Based Development Appeals.

[33] Dans la phase de faisabilité actuelle, seule la génération de code SQL a été effectuée mais elle sera complétée dans les mois à venir par la génération d'interfaces Homme/Machine.

itération pour fournir les concepts et les spécifications à implémenter au cours de l'itération et en fin d'itération pour valider l'incrément produit. L'intensité de participation est de quelques jours par itération avec une fréquence fixée par la durée de l'itération (2 à 4 semaines). ***La participation est répartie tout au long du développement et son intensité globale peut être comparable à la précédente. La fréquence est supérieure à la précédente.*** Le processus itératif a été introduit pour réduire les risques de dérive et d'échecs, en faisant contrôler l'évolution du développement par les acteurs.

Les méthodes agiles préconisent une participation accrue des acteurs du domaine. Pour eXtreme Programming, cette pratique se traduit par la présence en permanence du client ou de son représentant dans l'équipe de développement. Le client devient donc un membre à part entière de l'équipe. Il a pour rôle de décrire les scénarios qui vont permettre de réaliser les fonctionnalités, de préciser les spécifications, d'élaborer les tests de validation du bon fonctionnement de l'application, … et de valider le développement en continu. ***La participation est alors continue et son intensité coïncide avec la durée du projet. La fréquence est infinie.*** La présence du client dans l'équipe accroît sa satisfaction et la qualité du développement (réduction des bogues d'un facteur 5 selon certaines expériences). Le développement est alors quasiment sous contrôle du client.

Il est intéressant de constater que l'intensité de participation n'est pas le seul facteur améliorant la satisfaction des acteurs et la qualité du produit final. La fréquence de participation est probablement plus importante.

Fort de ce constat, nous avons cherché à accroître le niveau de performance en adoptant un *processus de développement permettant le prototypage rapide en séance d'analyse* ce qui a pour effet de rendre la participation des acteurs plus active. Une nouvelle méthode de conduite de projet est proposée. Elle a pour seul objectif de rendre plus active la participation des acteurs.

La dichotomie entre branche fonctionnelle et branche technique introduite par (Roques *et al.*, 2002) dans le processus de développement en Y n'est pas sans rappeler la séparation des spécifications métiers de celles d'implémentation que préconise l'approche MDA (cf. I). En effet, l'indépendance vis-à-vis de la plate-forme est assumée dans la branche fonctionnelle alors que la branche technique est spécifique à une plate-forme.

Dans les faits, il y a une légère différence entre la dichotomie de la démarche MDA et celle du processus de développement en Y. Cette différence est perceptible lors de la modélisation de concepts matériels ou techniques qui sont fondamentalement des concepts liés à une plate-forme mais qui, dans les phases d'analyse et de conception, sont indépendants de la plate-forme et du langage de programmation.

Par exemple, Le concept de fichier est indépendant du langage de programmation permettant sa création, son ouverture, sa consultation et sa fermeture. À ce titre, c'est un concept PIM dans le contexte de la démarche MDA mais un concept technique dans le processus en Y. Il ne relève pas de la branche fonctionnelle comme les autres concepts métiers ou thématiques mais de la branche technique. L'exemple du concept de fichier n'est pas le seul dans ce cas, bien d'autres concepts matériels ou techniques posent le même problème de classification au cours du développement.

C'est cette difficulté de classification des concepts matériels ou techniques au cours du développement qui a conduit la société Thales à étudier l'approche en double Y (Farcet, 2003).

Fort de cette analyse et soucieux de satisfaire l'objectif élémentaire de *capitalisation des connaissances*, nous proposons une généralisation de l'approche MDA. Un artefact multimodèle a été conçu et implémenté pour atteindre l'objectif élémentaire de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique*.

En 2001, (Miller *et al.*, 2001) avaient décrit de façon détaillée les mécanismes existants entre les modèles PIM et PSM en explicitant les transformations relatives à ces deux modèles et avaient formalisé cette approche par un métamodèle (cf. figure 24). En 2003, les auteurs du guide de l'approche MDA (Miller *et al.*, 2003) ont introduit le modèle CIM sans préciser des mécanismes pouvant exister entre le modèle CIM et les modèles PIM et PSM. Dans ce guide, aucune des transformations décrites ne fait intervenir le modèle CIM.

La généralisation de l'approche MDA que nous proposons précise les mécanismes entre le modèle CIM et les modèles PIM et PSM.

Partie B - Contribution à la généralisation de l’approche Model-Driven Architecture : Software Development Process Approach

Chapitre 3. Définitions et descriptions de la méthode Continuous Integration Unified Process, de l'approche Software Development Process Approach et de l'artefact Software Development Process Model

Ce chapitre présente la démarche intellectuelle d'élaboration des nouveaux concepts qui a permis d'atteindre l'objectif général de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique* adapté à un *processus de développement permettant le prototypage rapide en séance d'analyse* assurant la *capitalisation des connaissances* (cf. Introduction-IV).

Ces nouveaux concepts sont une méthode de conduite de projet dérivée de la méthode Processus Unifié, désignée *Continuous Integration Unified Process*, une généralisation de l'approche MDA que nous avons appelée *Software Development Process Approach* et un artefact[34] multimodèle que nous avons nommé *Software Development Process Model*

Avant d'introduire ces trois concepts, nous précisons, dans le paragraphe I, notre vision de deux « ingrédients » importants du développement d'une application : le processus et les modèles.

Le paragraphe II introduit les raisons qui ont conduit à proposer et à définir la nouvelle méthode de développement : *Continuous Integration Unified Process*.

Le paragraphe III identifie la *capitalisation des connaissances* qui sous-tend l'approche MDA comme étant la principale préoccupation. Or c'est une préoccupation qui se pose tout au long du cycle de développement ce qui a amené à proposer et à définir l'approche *Software Development Process Approach*, nouvelle approche qui généralise l'approche MDA.

Le paragraphe IV détaille et définit l'artefact multimodèle, nommé *Software Development Process Model*, qui réifie la nouvelle approche *Software Development Process Approach* adaptée à la méthode *Continuous Integration Unified Process*.

Pour atteindre l'objectif élémentaire de réaliser un *outil d'aide à la conception*, nous avons été conduits à concevoir d'une part, des mécanismes de transferts des concepts entre les modèles constituant le *Software*

[34] Cf. définition 79

Development Process Model et, d'autre part, une architecture assurant le maintien en cohérence de cet artefact multimodèle.

Le paragraphe V décrit les notions qu'il a été nécessaire de mobiliser pour cela. Il présente aussi le principe général de la diffusion des concepts entre modèles.

Enfin, le paragraphe VI conclut notre propos.

Sommaire détaillé

I VISIONS SUR LE DEVELOPPEMENT D'UNE APPLICATION

I.1 Vision machine à états finis[35] du processus de développement

Qu'ils soient linéaires ou itératifs, tous les processus de développement possèdent une phase d'analyse, une phase de conception et une phase d'implémentation.

Au cours du développement, c'est le chef de projet ou la personne faisant[36] office qui prend la décision de changer de phase. Lors de ces transitions, le modèle d'analyse devient modèle de conception et ensuite modèle d'implémentation. Ces changements s'effectuent toujours suivant la même séquence.

Le développement de l'application peut être vu comme une machine à états finis. Il peut donc être modélisé par un diagramme d'états (figures 34 et 35).

La transition d'un état à l'autre est le résultat de la décision du chef de projet de changer de phase de développement. En phases d'analyse et de conception, les activités au sein des états sont l'ajout, la modification et la suppression de concepts thématiques, de spécifications propres au domaine, etc. Le développement progressant, en phase d'implémentation mais aussi en phase de conception, les concepts et les spécifications introduits dans le modèle sont destinés à une plate-forme logicielle et/ou matérielle.

Dans un processus de développement linéaire, toutes les exigences des commanditaires étant traitées, le développement de l'application prend fin.

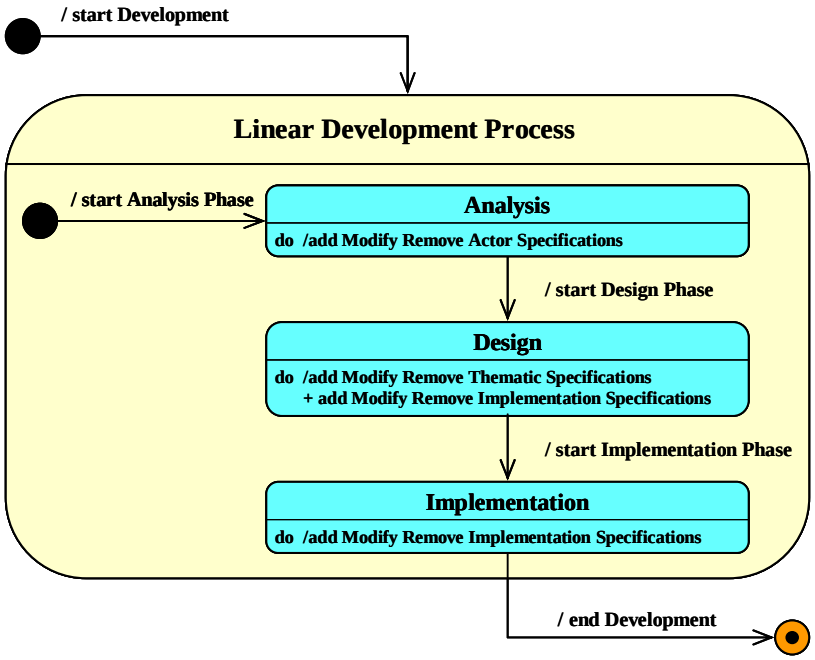


Figure 34 Processus de développement linéaire.

Dans un processus de développement itératif, le chef de projet vérifie, en fin d'itération, s'il reste des sous-systèmes à implémenter et si toutes les exigences des commanditaires sont satisfaites. Dans l'affirmative, le développement s'arrête sinon une nouvelle itération est effectuée et un nouvel incrément de l'application est réalisé. Ce nouvel incrément peut être soit un sous-système qui est analysé, conçu et ensuite implémenté, soit des améliorations qui ont été différées lorsque le développement d'une application est en cours de finalisation.

En effet, le pilotage par les risques du processus de développement amène à traiter en priorité les risques majeurs, c'est-à-dire à approfondir la conception des sous-systèmes dont l'analyse a été la moins élaborée. L'adoption de ce principe amène tout naturellement à différer les améliorations d'une application en fin de développement.

---

[35] Bien qu'il existe d'autres activités (planification de projet, gestion de personnel, etc.) associées à chacune des phases, dans ce chapitre nous nous intéressons uniquement à celles relatives à l'action de modélisation.

[36] Cas d'un programmeur « isolé ».

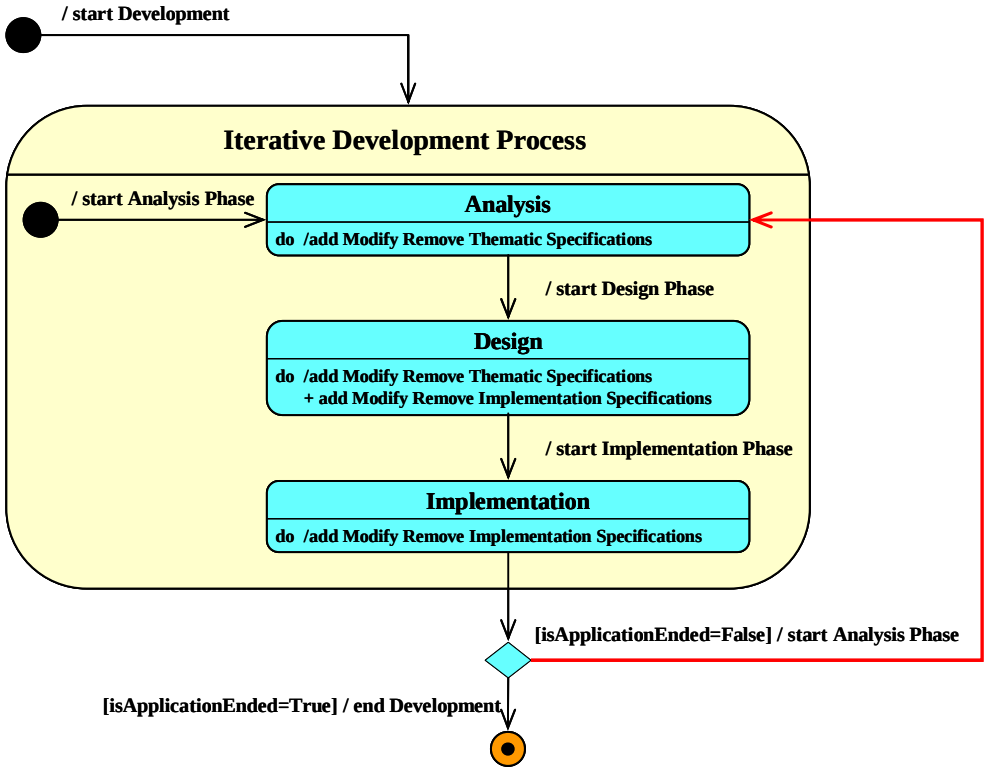


Figure 35 Processus de développement itératif.

Quel que soit le processus de développement, les activités de modélisation sont exercées préférentiellement au cours d'une des phases du processus de développement. Pour ces processus simplifiés, la phase de conception déroge à la règle puisque deux types d'activités de natures différentes sont réalisés.

I.2 Vision ensembliste des modèles impliqués dans un processus de développement

En 1994, Philippe Desfray (Desfray, 1994) écrivait : *Modelling is gradually decomposed from phase to phase by bringing in additional information*. La figure 36 accompagnait ce texte pour illustrer l'augmentation du volume d'information contenu dans un modèle depuis l'analyse jusqu'à l'implémentation. Cette figure a un second intérêt, c'est l'organisation des ellipses.

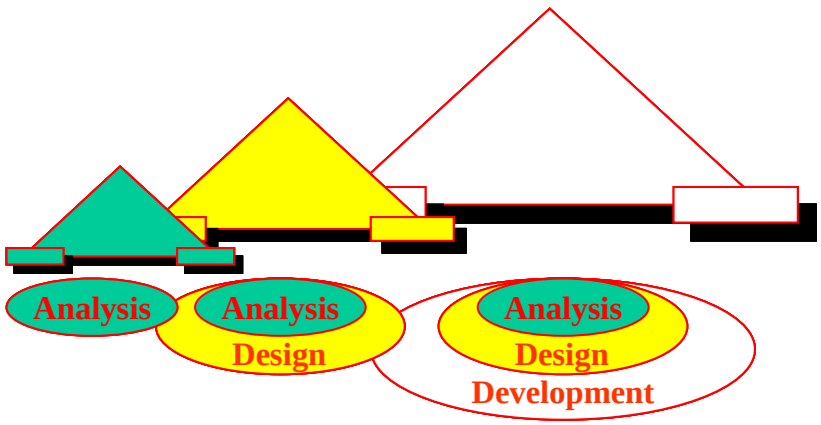


Figure 36 Évolution de la taille du modèle au cours du développement (Desfray, 1994).

L'inclusion de l'ellipse d'analyse dans celle de conception, elle-même contenue dans celle de développement montre parfaitement qu'il existe une relation ensembliste :

- ⇨ Entre les états d'un mono-modèle[37] au cours du cycle de développement.
- ⇨ Entre les modèles successifs des phases d'un cycle de développement qui sont parfois archivés (vision multimodèle).

Les ellipses de la figure 36 expriment que le modèle d'analyse est un sous-ensemble du modèle de conception, ce dernier étant lui-même un sous-ensemble du modèle d'implémentation ce qui s'écrit :

$$\boldsymbol{M_A \subset M_D \subset M_I}$$

$\boldsymbol{M_A}$: Modèle d'analyse

$\boldsymbol{M_D}$: Modèle de conception

$\boldsymbol{M_I}$: Modèle d'implémentation (Development)

Quelque soit type de l'application, le développement suit toujours cette vision ensembliste puisque les concepts manipulés par les acteurs sont capturés lors de l'analyse et restent, exceptés ceux qui sont jugés inutiles par les acteurs, présents dans le modèle jusqu'à l'implémentation.

II METHODE DE CONDUITE DE PROJET CONTINUOUS INTEGRATION UNIFIED PROCESS : IDEES FONDATRICES ET DEFINITION

La méthode de conduite de projet Processus Unifié est un recueil des meilleures pratiques récentes de développement logiciel dans des domaines très variés (cf. Annexe II-II.1). Bien qu'indépendante du langage UML, cette méthode a été conçue par les pères d'UML et articulée avec ce langage.

Par ailleurs, de par la complexité de leur domaine de recherche lié à l'environnement, les scientifiques du Cemagref sont amenés à décrire et à modéliser des systèmes qui atteignent rapidement quelques dizaines voire quelques centaines de concepts.

Par exemple, le modèle du Système d'Information à Référence Spatiale des Digues permettant d'assurer la gestion patrimoniale de l'état des digues et historiques des travaux effectués comprend plus de cent cinquante classes. Dans ce contexte, le principe du *développement centré sur l'architecture* de la méthode Processus Unifié (cf. Chapitre 2-II.1.4), principe dont la méthode eXtreme Programming se désintéresse totalement, est très séduisant car il permet de structurer les concepts d'une application au sein de paquetages thématiques.

Il n'est pas rare aussi que dans des disciplines proches, deux concepts voisins mais ayant des différences portent le même nom. Par exemple, le concept de *Bief* d'un cours d'eau n'est pas du tout le même que le *Bief* d'une raie pour l'irrigation par gravité. Pourtant, ces deux concepts sont très voisins. Tous deux vont être utilisés par les hydrauliciens/numériciens pour décrire la géométrie et faire tourner des modèles de calculs hydrauliques à surface libre. Dans ce contexte, la structuration des concepts par discipline s'avère aussi être un plus.

L'articulation de la méthode Processus Unifié avec le langage UML et les problèmes de structuration rencontrés par les scientifiques nous ont fait adopter la méthode de conduite de projet Processus Unifié.

De plus, le principe du développement centré sur l'architecture est très intéressant dans le cadre des travaux menés car il est alors possible de structurer les concepts du domaine SIG qui seront générés automatiquement.

L'objectif élémentaire de mettre en œuvre un *outil d'aide à la conception de Systèmes d'Information Géographique* dans un *processus de développement permettant le prototypage rapide en séance d'analyse* ne peut pas être atteint avec la méthode Processus Unifié car la durée d'une itération est de 2 à 4 semaines (cf. Chapitre 2-II.1.4) et qu'en outre les activités de modélisation à réaliser aux différentes phases sont souvent manuelles.

Fort de ce constat et dans l'objectif d'œuvrer suivant un *processus de développement permettant le prototypage rapide en séance d'analyse*, nous avons mobilisé des concepts de la méthode eXtreme Programming pour concevoir et proposer une variante de la méthode Processus Unifié.

[37] Un mono-modèle est un modèle unique utilisé au cours de tout le cycle de développement depuis l'analyse jusqu'à l'implémentation. Le code produit avec des méthodes de développement centré sur le code, eXtreme Programming par exemple, est un mono-modèle.

Le choix de réaliser du prototypage en séance d'analyse a été effectué afin de placer le concepteur d'une application dans un contexte simailaire à celui d'un développeur d'une équipe eXtreme Programming.

Au cours de l'analyse, les acteurs et le concepteur sont réunis en un même lieu et font ensemble l'analyse du système. Ils sont alors dans les conditions de la pratique du *Client sur Site* prônée par la méthode eXtreme Programming (cf. Chapitre 2-II.2.2). Or, cette pratique est destinée à améliorer la communication entre le client et les développeurs et ce à deux niveaux :

- ⇨ L'activité première du client est d'« alimenter » l'équipe en concepts, en spécifications, etc. qui vont permettre aux développeurs de réaliser l'application. C'est l'activité classique d'analyse.
- ⇨ La seconde activité du client est d'évaluer la pertinence des spécifications qu'il a fournies en validant le produit obtenu. Cette activité correspond à la validation de l'incrément produit. Elle n'est possible que parce que la mèthode eXtreme Programming a adopté la pratique d'*Intégration continue.*

En eXtreme Programming, l'analyse, la conception, l'implémentation et la réalisation des tests, activités d'un cycle Processus Unifié, sont donc effectuées en continu tout au long du projet. L'intégration d'un incrément dans le code général finalise la validation.

L'objectif élémentaire d'œuvrer suivant un **processus de développement permettant le prototypage rapide en séance d'analyse** ***relève de la conjonction des deux pratiques :*** **Client sur site** ***assurant les deux fonctions d'analyse et de validation et l'*****Intégration continue.**

En séance d'analyse, la pratique du Client sur site est satisfaite de fait, reste à relever le challenge lié à la pratique de l'*Intégration continue*.

Effectuer une intégration suppose qu'un incrément a été produit et que toutes les activités des phases de conception, d'implémentation voire de tests ont été réalisées.

Si ces activités sont effectuées manuellement, la séance d'analyse va être entrecoupée de temps-morts improductifs donc onéreux. Le travail d'analyse qui est fondamental va rapidement devenir fastidieux pour les acteurs. Ils vont alors se démobiliser et se désintéresser du problème. L'analyse devient donc moins pertinente et la qualité de l'application se dégrade.

Par contre, si ces mêmes activités sont automatisées alors les temps-morts n'existent plus et, face à un prototype, la réactivité ainsi que le feedback des acteurs sont bien meilleures que face à un modèle dont ils ne connaissent pas toutes les finesses du langage de modélisation. Ils peuvent alors valider, modifier ou supprimer les concepts capturés et régénérer un nouveau prototype immédiatement.

De fait, l'objectif de modéliser un système suivant un processus de développement permettant le prototypage rapide en séance d'analyse accroît la communication entre les acteurs et le concepteur. La communication est la valeur érigée en valeur fondamentale par Kent Beck concepteur de la méthode eXtreme Programming (cf. Chapitre 2-II.2.2).

Fort de ces réflexions, nous proposons de définir une variante de la méthode Processus Unifié qui permette de satisfaire l'objectif élémentaire d'opérer suivant un *processus de développement permettant le prototypage rapide en séance d'analyse* :

Définition 4 La méthode ***Continuous Integration Unified Process*** superpose, au cycle principal de la méthode Processus Unifié, un cycle de prototypage rapide (figure 37) doté d'un processus automatisant l'évolution des modèles depuis l'analyse jusqu'à l'implémentation.

Nota En figure 37, seules les phases d'analyse, de conception, d'implémentation et de validation du cycle de prototypage rapide ont été représentées car ce sont les phases qui ont été automatisées dans le cadre de cette recherche. Toutefois, la définition 4 ne restreint pas le principe de la méthode à ces seules phases et d'autres phases peuvent être automatisées.

L'idée d'évolution automatique des modèles depuis l'analyse jusqu'à l'implémentation est une préoccupation de la communauté MDA. Cette idée n'est pas affichée comme un objectif dans les textes fondateurs de l'approche MDA (Miller *et al.*, 2001, 2003) mais il est évident à la lecture de ces textes que c'est l'un des objectifs recherchés et atteint dans certains environnements contraints. Par exemple, le standard CWM[38] (OMG, 2001b) permet de couvrir le cycle complet de conception, réalisation et gestion des entrepôts de données (Miller *et al.*, 2001). Certains auteurs qualifient de ***full MDA*** (Kleppe, 2004 ; Softeam, 2005) le processus d'évolution automatique des modèles depuis l'analyse jusqu'à l'implémentation. Pour faire évoluer automatiquement un

[38] Common Warehouse Metamodel

modèle depuis l'analyse jusqu'à l'implémentation, la communauté MDA suggère de faire appel aux ***transformations de modèle*** (cf. Chapitre 2-I.2).

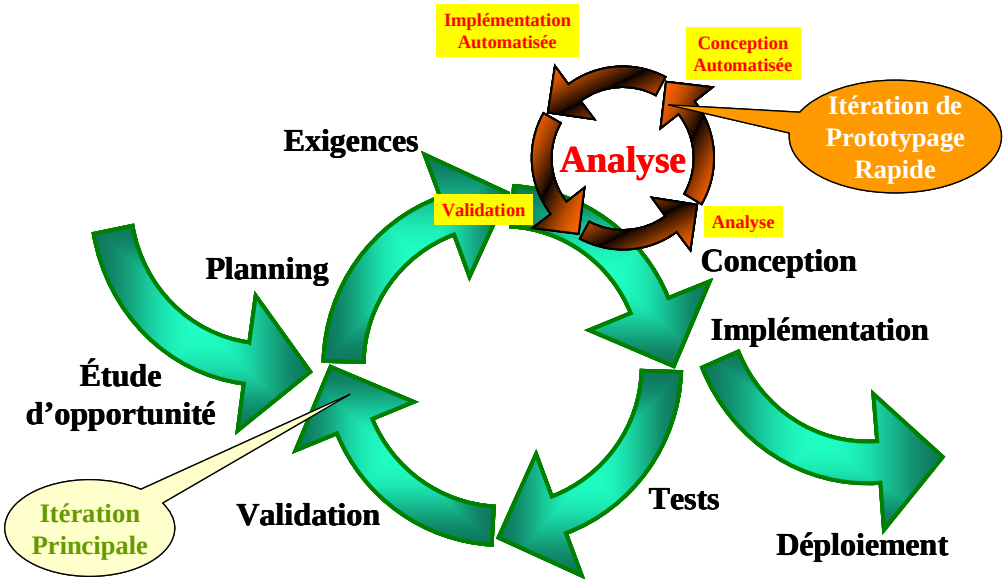


Figure 37 Cycle de prototypage rapide de la méthode *Continuous Integration Unified Process*.

Nous avons adopté ces idées novatrices dans le contexte de la méthode *Continuous Integration Unified Process*. Aussi, l'automatisation complète des phases de conception et d'implémentation suppose la mise en œuvre d'une *panoplie de transformations de modèle* que nous détaillerons au Chapitre 5.

Doté de cette panoplie de transformations, la durée des phases de conception et d'implémentation du cycle de prototypage rapide est réduite au temps d'exécution de ces transformations lui-même lié au volume de concepts réifié dans le modèle.

III SOFTWARE DEVELOPMENT PROCESS APPROACH : PRINCIPE FONDATEUR ET DEFINITION GENERALE

Le guide de l'approche MDA (Miller *et al.*, 2003) stipule que : *« The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns »*. Afin de satisfaire ces objectifs, les concepteurs de l'approche MDA ont introduit en 2001 les concepts de modèle indépendant des plates-formes (modèle PIM) et de modèle spécifique à une plate-forme (modèle PSM) (Miller *et al.*, 2001).

La séparation des préoccupations via une architecture à plusieurs modèles relève en fait d'une préoccupation plus fondamentale qui est la* capitalisation *des concepts, des spécifications, etc.

Le guide d'utilisation de l'approche MDA publié en 2003 vient confirmer ce constat. Le modèle CIM a été introduit dans ce guide pour *combler le fossé existant entre les experts du domaine thématique et les experts de la conception et de la construction de l'application* (Miller *et al.*, 2003) mais ce n'est pas sa seule fonction puisque il est *utile, non seulement comme aide à la compréhension du problème, mais aussi comme source de vocabulaire partagé à utiliser dans d'autres modèles*. De fait dans cette seconde fonction, il capitalise une partie de la connaissance des acteurs du domaine.

La dichotomie entre branches fonctionnelle et technique introduite dans le processus de développement en Y (cf. Chapitre 2-II.1.5) n'est pas sans rappeler la séparation des spécifications métiers de celles d'implémentation préconisée par l'approche MDA. La préoccupation sous-jacente en adoptant le processus de développement en Y est aussi une préoccupation de capitalisation, plus des acquis peut-être que des connaissances.

La capitalisation des connaissances est donc une préoccupation constante qui se pose tout a long du cycle de développement d'une application ou d'un système d'information. L'objectif élémentaire de* capitalisation des connaissances *que nous avons adopté est issu de ce constat et généralise les idées de l'approche MDA.

Pour atteindre l'objectif élémentaire de *capitalisation des connaissances* tout au long du cycle de développement, nous avons remplacé la logique de séparation technique par une logique processus de développement.

L'adoption de cette nouvelle logique permet d'énoncer la définition de la nouvelle approche que nous proposons :

> **Définition 5** La ***Software Development Process Approach*** est une approche de développement au cours de laquelle un ou plusieurs modèles sont mis en œuvre à chacune des phases du cycle de développement d'une méthode de conduite de projet.

Cette définition générale n'est rien d'autre que la formalisation de la pratique d'archivage d'étapes intermédiaires, donc de capitalisation, d'un modèle ou d'un code au cours du développement d'une application informatique. De plus, il n'est pas rare que cet archivage soit réalisé lorsqu'une phase du cycle de développement est achevée. Admettre l'existence de plusieurs modèles au cours d'une phase de développement correspond au souci de l'approche MDA de pouvoir dériver plusieurs modèles PSM d'un même modèle PIM ce qui est aussi une pratique courante dans un processus de développement.

Une analyse approfondie des approches *Software Development Process Approach* et MDA montre que la première englobe la seconde. En effet, le modèle CIM qui est le modèle du domaine correspond au modèle de la phase d'analyse, alors que le ou les modèles PSM correspondent aux modèles de la phase d'implémentation et enfin que le modèle PIM correspond, suivant la méthode, aux modèles d'une ou de plusieurs phases intermédiaires. Pour un cycle en cascade, le modèle PIM est le modèle de la phase de conception alors que pour un cycle en Y (cf. Chapitre 2-II.1.5) le modèle PIM correspond aux modèles des phases de conception préliminaire et de conception détaillée. La *Software Development Process Approach* fournit en fait le fil logique manquant entre le modèle CIM et les modèles PIM et PSM du texte fondateur de l'approche MDA.

Définir une façon de travailler ne suffit pas pour garantir la capitalisation des connaissances. Le problème majeur rencontré lorsque l'archivage de modèles est pratiqué est qu'il est difficile de faire « vivre » en cohérence plusieurs modèles. Rapidement, les différents modèles dérivent car maintenir plusieurs modèles en cohérence manuellement relève de l'« héroïsme » et s'avère très coûteux. C'est la raison pour laquelle il faut d'une part, réifier l'approche de travail en un artefact qui opérationnalise les idées et, d'autre part, mettre en place les mécanismes de diffusion entre les différents modèles évitant ces problèmes d'incohérence.

Cet artefact est le concept *Software Development Process Model* dont la définition est :

> **Définition 6** Le ***Software Development Process Model*** est un artefact de modélisation multimodèle composé d'au moins un modèle par phase du cycle de développement de la méthode de conduite de projet mise en oeuvre. Lorsque plusieurs modèles sont utilisés pour une même phase, ils sont regroupés au sein d'un artefact unique faisant office de container et seul représentant de la phase. Le nom de la phase de développement est mentionné pour chacun des modèles et des containers.

Il faut souligner que la définition 5 ne limite pas la *Software Development Process Approach* au seul cycle de développement de la méthode *Continuous Integration Unified Process* pour laquelle elle a été pensée. En effet, cette nouvelle approche peut aussi bien être appliquée au cycle de développement en cascade, qu'au cycle de la méthode de conduite de projet RAD (Annexe II-I), qu'au cycle en Y de la méthode 2TUP (cf. Chapitre 2-II.1.5), etc.

Ainsi, pour un cycle de développement en cascade, les modèles constituant l'artefact *Software Development Process Model* pourraient être les modèles d'*Analyse*, de *Conception*, d'*Implémentation*, de *Tests* et de *Déploiement* (Royce, 1970).

Les modèles du *Software Development Process Model* destinés à la méthode de conduite de projet RAD pourraient être nommés *Initialisation*, *Cadrage*, *Conception*, *Construction* et *Finalisation* (Annexe II-I.2).

Il faut aussi souligner que la définition 5 ne restreint pas l'utilisation de cette approche au seul formalisme objet et au seul langage UML. En effet, rien n'interdit du point de vu théorique d'adopter des formalismes différents selon le besoin, le type d'acteurs ou la phase de développement. Par exemple, la modélisation de l'organisation de la production d'une entreprise pourrait :

- ⇨ Débuter en phase de préétude avec des formalismes comme IDEF0/SADT (ref) ou AMS (ref) car les interlocuteurs industriels sont familiers à ces formalismes.
- ⇨ Se poursuivre avec le langage UML.

⇨ Être finalisée avec le formalisme Entité/Relation car les données de production sont stockées dans une base de données.

Bien évidemment, la difficulté majeure sera de faire cohabiter ces différents formalismes. C'est la raison pour laquelle nous n'avons utilisé qu'un seul formalisme, le formalisme objet, et un seul langage, le langage UML. L'avantage du langage UML est qu'il peut être mis en œuvre au cours de quasiment toutes les phases du cycle de développement.

La Software Development Process Approach est une approche de modélisation ouverte, souple et adaptable.

Ayant adopté la méthode *Continuous Integration Unified Process*, nous avons défini un *Software Development Process Model* spécifique à cette méthode. C'est l'objet du paragraphe suivant.

IV SOFTWARE DEVELOPMENT PROCESS MODEL DESTINÉ À LA MÉTHODE CONTINUOUS INTEGRATION UNIFIED PROCESS

IV.1 Définition d'un Software development process model spécifique à la méthode Continuous integration unified process

La méthode Processus Unifié étant adoptée, nous avons désigné les modèles du *Software Development Process Model* du nom des phases de développement de cette méthode de conduite de projet. Cependant, la remarque finale du paragraphe I relative à l'indifférenciation d'activités de nature distinctes en phase de conception a incité à adopter la dichotomie proposée par le cycle en Y : phase de conception préliminaire suivie d'une phase de conception détaillée.

Par ailleurs, notre vision du *Software Development Process Model* est monolithique c'est-à-dire que cet artefact est LE MODÈLE de l'application car il contient tous les concepts, spécifications, etc. participant à la modélisation de l'application. Les modèles associés à chacune des phases ne sont que des artefacts de capitalisation des connaissances tout au long du cycle de développement. Selon cette vision monolithique, le *Software Development Process Model* est un container regroupant les autres modèles. C'est la raison qui a conduit à qualifier de sous-modèles les modèles associés aux phases de développement.

La définition suivante est le fruit de cette vision monolithique :

Définition 7 Le ***Software Development Process Model (dedicate to the Continuous Integration Unified Process)*** est un container constitué de sous-modèles chacun étant associé à une phase du cycle de développement de la méthode *Continuous Integration Unified Process*. À la phase d'analyse est associé un *sous-modèle d'analyse*, à la phase de conception préliminaire correspond un *sous-modèle de conception préliminaire*, la phase de conception avancée s'appuie sur un *sous-modèle de conception avancée* et enfin la phase d'implémentation est représentée par un container regroupant les différents *sous-modèles d'implémentation*.

Nota 1 Sauf ambiguïté, le texte entre parenthèse sera omis dans le reste du document.

Nota 2 La structuration en container est dictée par le principe du développement centré sur l'architecture de la méthode Processus Unifié.

Nota 3 La phase d'implémentation s'appuie sur plusieurs sous-modèles afin de prendre en compte la préoccupation de la communauté MDA de pouvoir dériver plusieurs modèles PSM d'un même modèle PIM.

IV.2 Définition et activités de chacun des sous-modèles

Comme pour les cycles de développement en cascade ou itératif décrits au paragraphe I, le processus de développement avec le *Software Development Process Model* peut être décrit par un diagramme d'états. La figure 38 représente le processus de développement avec cet artefact.

Immédiatement après la création du *Software Development Process Model*, le sous-modèle d'analyse va être créé avec le démarrage de la phase d'analyse. En séance d'analyse, des concepts et des spécifications issus du domaine étudié vont être ajoutés dans le sous-modèle d'analyse. Naturellement, ce modèle est un modèle PIM.

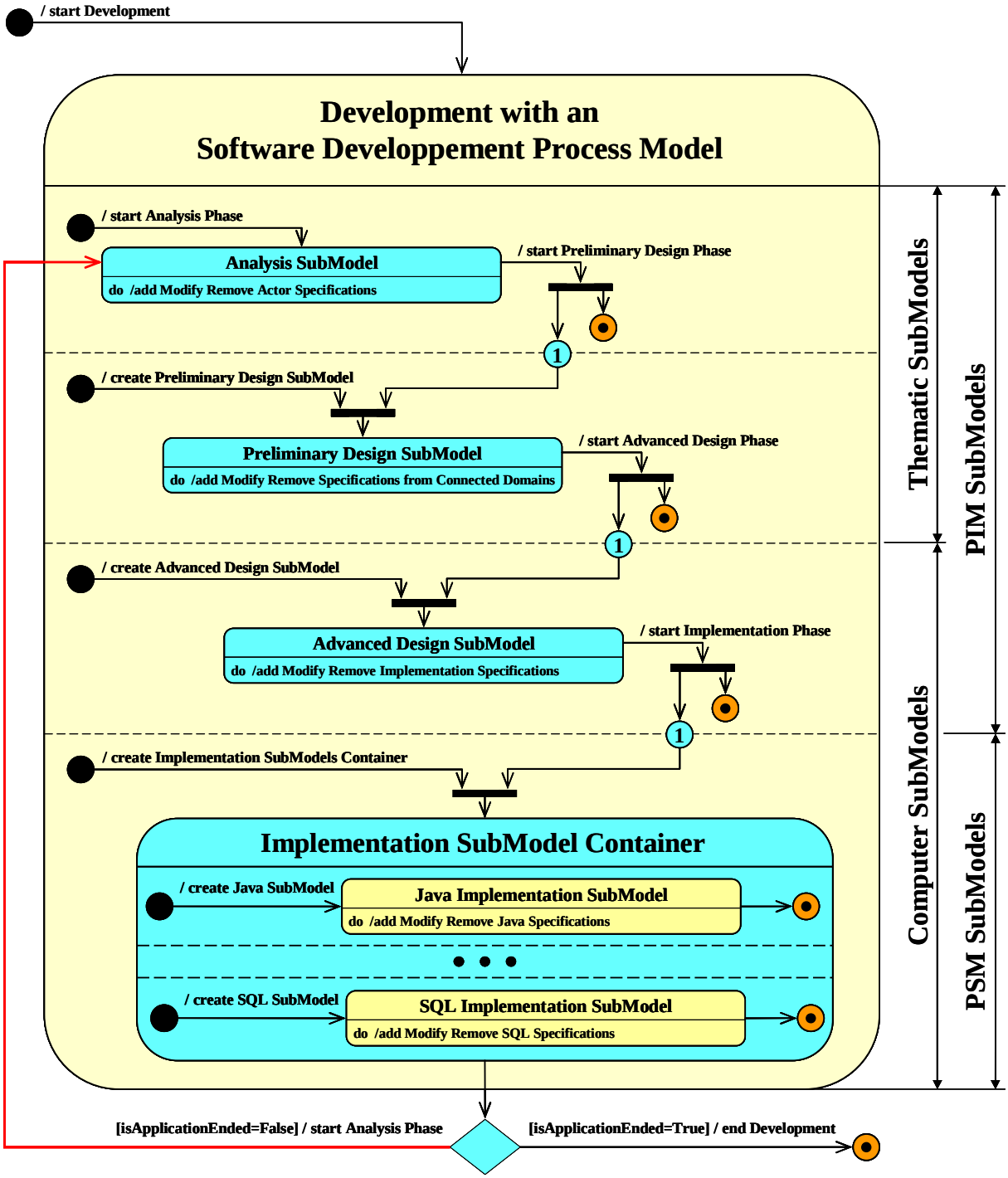


Figure 38 Processus de développement avec un *Software Development Process Model*.

Nous proposons la définition suivante :

Définition 8 Le ***sous-modèle d'analyse*** (SM_A) est un modèle PIM ne contenant que des concepts et des spécifications thématiques du domaine étudié. Les seules activités autorisées sur ce sous-modèle sont l'ajout, la modification et la suppression de concepts et de spécifications thématiques du domaine des acteurs.

Par exemple, au cours de la conception et du développement d'un *Système d'Information Géographique*, les entités référencées seront saisies au sein du sous-modèle d'analyse et les propriétés spatiales et temporelles y seront exprimées via le langage pictogrammique du Chapitre 7.

Lorsque le chef de projet le décidera, la phase de conception préliminaire débutera. Le sous-modèle de conception sera alors créé. Selon la vision ensembliste des modèles (cf. I.2), les concepts du sous-modèle d'analyse doivent être incorporés dans le sous-modèle de conception préliminaire. Ensuite, de nouveaux concepts et spécifications thématiques seront ajoutés. À la différence des concepts et spécifications du sous-modèle d'analyse, ils sont issus de domaines connexes à celui en cours de modélisation. Comme le précédent, ce modèle est un modèle PIM, d'où la définition :

Définition 9 Le ***sous-modèle de conception préliminaire*** ($\boldsymbol{SM_{PD}}$) est un modèle PIM contenant, outre les concepts et les spécifications du sous-modèle d'analyse, les concepts et les spécifications thématiques de domaines connexes indispensables à la réalisation de l'application.

$$\boldsymbol{SM_A \subset SM_{PD}}$$

Par exemple, le développement d'une application de prévision des crues d'un cours d'eau fera sûrement appel à des entités référencées. Elles seront saisies dans le sous-modèle d'analyse. Il est probable que ce type d'application mobilise des concepts hydrauliques, équations de Saint Venant par exemple, ou des concepts mathématiques, différences finies, éléments finis, modèles mathématiques qui permettent de calculer une ligne d'eau ou une zone inondée. Les concepts hydrauliques et mathématiques utilisés alors sont le type même de concepts thématiques indépendants des modèles qui n'appartiennent pas au domaine des acteurs. Par contre, ils sont indispensables au fonctionnement de l'application finale.

Les concepts thématiques du sous-modèle de conception préliminaire étant incorporés, le chef de projet demandera à son équipe de s'attaquer à la conception avancée. Le sous-modèle de conception avancée sera créé et l'ensemble des concepts du sous-modèle de conception préliminaire sera incorporé comme l'exige la vision ensembliste (cf. I.2). Dans le sous-modèle de conception avancée, les concepts et les spécifications introduits seront des concepts décrivant le matériel informatique et les technologiques informatiques déployées dans l'application mais indépendants de toute plate-forme et tout particulièrement indépendants du langage de programmation. La définition sous-modèle de conception avancée est :

Définition 10 Le ***sous-modèle de conception avancée*** ($\boldsymbol{SM_{AD}}$) est un modèle PIM qui intègre d'une part, tous les concepts et les spécifications du sous-modèle de conception préliminaire et, d'autre part, les concepts et les spécifications décrivant le matériel et/ou les technologiques informatiques utilisés par l'application.

$$\boldsymbol{SM_A \subset SM_{PD} \subset SM_{AD}}$$

La mise en œuvre d'une application informatique nécessite des composants matériels et/ou logiciels mais aussi des technologies informatiques comme les technologies de la communication. La description et/ou l'accès des composants matériels ou celle des technologies informatiques peut être effectuée jusqu'à un certain niveau indépendamment de toute plate-forme. Par exemple, le concept Client-serveur peut être décrit sans préciser le langage de programmation qui va permettre de l'implémenter. Il en est de même des interfaces Homme/Machine des applications (fenêtre, bouton, champ de saisie, cases à cocher, etc.), des disques durs (fichier texte, fichier binaire, répertoire, etc.), etc. Tous ces concepts doivent, dans notre vision du *Software Development Process Model*, être modélisés au sein du sous-modèle de conception avancée.

La phase de conception avancée terminée, le chef de projet demandera à son équipe de finaliser l'application. Pour ce faire, au moins un sous-modèle d'implémentation devra être réalisé. Le concepteur devra en premier lieu créer un container qui regroupera les sous-modèles d'implémentation propres aux différentes plates-formes matérielles ou logicielles cibles et ensuite il devra créer au moins un sous-modèle, plus s'il le souhaite. Tous les concepts du sous-modèle de conception avancée seront incorporés conformément à la vision ensembliste (cf. I.2). Alors seulement, les spécifications propres à une plate-forme ou à un langage de programmation seront précisées. Les définitions du container et d'un sous-modèle d'implémentation sont :

Définition 11 Le ***container de sous-modèles d'implémentation*** a pour seule fonction de regrouper les sous-modèles d'implémentation nécessaires en phase d'implémentation.

Définition 12 Un ***sous-modèle d'implémentation*** (SM_I) est un modèle PSM dérivé du sous-modèle de conception avancé incluant en plus les concepts et les spécifications propres à la plate-forme cible (langage de programmation cible, le code des opérations, etc.).

$$SM_A \subset SM_{PD} \subset SM_{AD} \subset SM_I$$

La plupart des systèmes d'information récents, qu'ils soient géographiques ou non, mobilisent deux langages :

- ⇨ Le langage SQL qui réifie la base de données, l'alimente en information, réalise parfois certains traitements, etc. Ce langage de requête dédié aux bases de données a été conçu sans composants informatiques permettant d'implémenter des interfaces Homme/Machine.
- ⇨ Un langage de programmation pour pallier la déficience du langage SQL en termes d'interfaces Homme/Machine.

Au terme de la première itération, le chef de projet doit se poser la question si le développement est terminé ou pas. Si c'est le cas, l'application est alors déployée sinon un nouveau sous-système est analysé, conçu et implémenté. Le processus[39] est le même que précédemment.

Au vu des définitions 7 à 12, les sous-modèles composant le *Software Development Process Model* destinés à la méthode *Continuous Integration Unified Process* peuvent être analysés suivant une double grille de lecture (cf. figure 38) :

- ⇨ La première grille de lecture est conforme à l'approche MDA :
  - Les sous-modèles d'analyse, de conception préliminaire et de conception avancée sont alors vus comme un raffinement du modèle PIM[40].
  - Les différents sous-modèles d'implémentation correspondent aux modèles PSM de l'approche MDA.
- ⇨ La grille seconde grille correspond à la vision des acteurs impliqués dans le projet :
  - Les sous-modèles d'analyse et de conception préliminaire peuvent être appréhendés par les acteurs non informaticiens.
  - Par contre, les sous-modèles de conception avancée et d'implémentation sont plutôt destinés à la communauté informatique car ces modèles sont souvent trop complexes pour des acteurs néophytes et ces derniers ne peuvent pas se les approprier facilement.

L'architecture multimodèle du *Software Development Process Model* a été conçue pour satisfaire l'objectif élémentaire de *capitalisation de connaissances* mais elle pose deux écueils majeurs :

- ⇨ Le premier est la diffusion des concepts capturés depuis le sous-modèle analyse jusqu'au(x) sous-modèle(s) d'implémentation.
- ⇨ Le second est le maintien en cohérence de cette architecture multimodèle.

Ce sont les deux mêmes écueils qui sont rencontrés lorsque le modèle d'une application est archivé aux différentes phases du développement et qu'il faut, pour des raisons diverses, revenir sur des concepts ou sur le modèle de la phase précédente. Le paragraphe suivant définit les principes qui ont permis de remédier à ces deux écueils.

V GESTION DE L'ARCHITECTURE MULTIMODELE DU SOFTWARE DEVELOPMENT PROCESS MODEL : PRINCIPE DE LA TRANSFORMATION DE DIFFUSION

La ***Transformation de diffusion*** que nous allons définir ici a nécessité deux notions importantes que sont le clonage et la traçabilité.

[39] Après la première itération, il n'est plus nécessaire de créer les sous-modèles puisqu'ils existent déjà sauf, bien entendu, si un nouveau sous-modèle implémentation est nécessaire,

[40] Ici, nous ne faisons pas la distinction entre les modèles CIM et PIM.

V.1 Introduction de la notion de clonage

La vision ensembliste des modèles associés aux phases de développement (cf. I.2) fournit nous semble-t-il la solution à l'écueil de diffusion des concepts depuis le sous-modèle d'analyse jusqu'au sous-modèle d'implémentation.

Dans l'architecture multimodèle du *Software Development Process Model*, un concept capturé en phase d'analyse est saisi dans le sous-modèle d'analyse ainsi que la description de ses propriétés, de son comportement etc. En phase de conception préliminaire, la vision ensembliste impose que le concept avec toutes ses propriétés et ses comportements soit présent dans le sous-modèle de conception préliminaire. ***C'est l'esprit même du concept de clonage au sens biologique du terme***[41].

Le cœur de la transformation diffusion qui va permettre la diffusion des concepts d'un sous-modèle à l'autre est une *transformation de clonage à l'identique*[42] comme défini par le dictionnaire en ligne allwords.com. Elle apporte une solution opérationnelle à l'écueil évoqué précédemment.

V.2 Introduction de la notion de traçabilité : L'architecture de traçabilité

Un premier type d'incohérence est induit par la nature multimodèle du *Software Development Process Model*. Un concept ***c*** saisi dans le sous-modèle d'analyse et diffusé jusqu'aux sous-modèles d'implémentation va exister en autant d'exemplaires que de sous-modèles.

Or pour que le *Software Development Process Model* reste en cohérence, il faut que chaque concept source et ses clones soient eux mêmes en cohérence entre sous-modèles mais aussi tout au long du processus de développement. Cette exigence doit être respectée en particulier à l'occasion des activités de modifications (correction d'une faute d'orthographe par exemple) qui sont naturellement effectuées sur n'importe quel sous-modèles.

Pour résoudre ce problème, nous avons mobilisé la notion de traçabilité. De façon générale, la traçabilité est une préoccupation récurrente en processus de développement et en modélisation (Desfray, 2001 ; D'Souza *et al.*, 1998 ; Jacobson *et al.*, 1999).

Dans un processus de développement sous-tendu par l'approche MDA, Philippe Desfray estime que d'une part, *la gestion de la traçabilité est absolument essentielle dans la démarche MDA* pour assurer la cohérence entre les modèles et, d'autre part, que si cette gestion *n'est pas effectuée automatiquement, les modèles deviennent inévitablement incohérents*. Fort de ces recommandations, nous avons généralisé l'utilisation de la *traçabilité* et nous l'avons *automatisée*.

Nous avons décidé de relier un concept source à son clone par un lien physique, désigné Lien de Traçabilité de Clonage, et ce d'une part, pour tous les concepts et, d'autre part, depuis le sous-modèle d'analyse jusqu'au(x) sous-modèle(s) d'implémentation.

L'ensemble de ces liens constitue l'***Architecture de Traçabilité de Clonage***, architecture qui est indépendante et orthogonale à celle du *Software Development Process Model*.

La figure 39 montre un exemple. Un lien de traçabilité relie le concept *Parcel* du sous-modèle d'analyse à son clone du sous-modèle de conception préliminaire, un autre est établi entre la classe *Parcel* du sous-modèle de conception préliminaire et son clone du sous-modèle de conception avancée, etc. Il en est de même pour le concept *Farm* et l'association *Use*.

L'architecture de traçabilité de clonage peut être assimilée à un *écheveau* dont chaque *fil d'Ariane* constitue une ***lignée de clonage*** entre le concept source et ses clones successifs permettant ainsi d'identifier sans ambiguïté une lignée de clonage dans *le dédale* des concepts présents dans le *Software Development Process Model*. Les liens de traçabilité assurent ainsi un accès direct aux clones et une navigation rapide entre sous-modèles.

Le principe de créer des lignées de clonage étant acquis et défini, restait à automatiser sa gestion conformément aux recommandations de Philippe Desfray. Devant être appliquée à tout concept d'un sous-modèle source pour créer leurs clones dans le sous-modèle destination, la *Transformation de diffusion* était toute indiquée pour assurer la gestion du lien de traçabilité de clonage entre un concept source et son clone.

---

[41] **Biology science**: Any of a group of genetically ***identical*** cells or organisms derived from a single parent cell or organism by asexual reproduction (http://www.allwords.com).
Computer science: An ***imitation*** of an existing computer or software product, usually cheaper, and produced by a different manufacturer (http://www.allwords.com).

[42] Par exemple, l'analyse de la transformation de clonage réalisant la séparation des préoccupations thématiques et techniques dans le projet RNTL ACCORD (Carrez, 2003) montre que le clonage est effectué de façon incomplète.

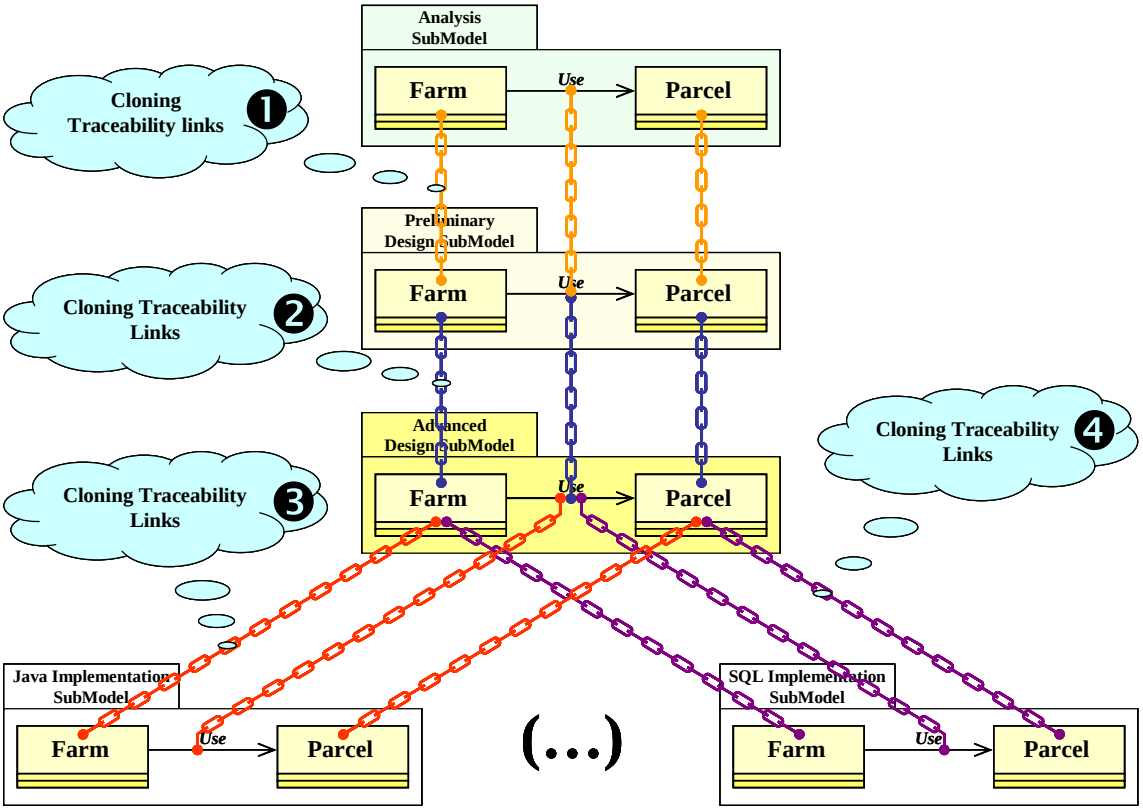


Figure 39 L'architecture de traçabilité de clonage.

La Transformation de diffusion a donc la responsabilité de créer, une fois le clonage terminé, le lien de traçabilité de clonage entre un concept et son clone.

V.3 Transformation de diffusion : Principe général

Dans un processus de développement linéaire (cf. I.1), l'opération de clonage et l'architecture de traçabilité résolvent les problèmes de diffusion et de maintien en cohérence du *Software Development Process Model*. Malheureusement, cela ne suffit pas dans un processus de développement itératif comme celui qui a été adopté dans la méthode *Continuous Integration Unified Process*.

En effet, lorsque la *Transformation de diffusion* est appliquée N fois entre deux sous-modèles, analyse et conception préliminaire par exemple, un concept ***c*** du sous-modèle d'analyse saisi lors de la première itération sera dupliqué N fois dans le sous-modèle de conception préliminaire. Ce fonctionnement est inacceptable pour la cohérence globale du *Software Development Process Model*.

Pour résoudre cet inconvénient, nous avons exploité l'information que constitue l'existence ou non du lien de traçabilité entre un concept et son clone.

Étant donné que le lien de traçabilité de clonage est créé en fin de *Transformation de diffusion*, lorsqu'un concept n'a pas de lien de traçabilité, cela signifie qu'il n'a pas de clone. Alors, la *Transformation de diffusion* doit effectuer une opération de clonage à l'identique.

Par contre, si le lien de traçabilité de clonage existe, alors la *Transformation de diffusion* ne doit pas créer de clone mais simplement effectuer une mise à jour de celui-ci. Ainsi, la duplication des clones liés au processus itératif est évitée et les modifications de noms en particulier sont transmises.

Afin que l'architecture multimodèle du *Software Development Process Model* soit opérationnelle, la *Transformation de diffusion* qui permet de gérer tout artefact multimodèle dans un processus itératif est une transformation complexe. Le principe général de cette transformation a été décrit ici mais nous verrons qu'elle a été dotée de responsabilité supplémentaire pour résoudre d'autres problèmes techniques (cf. Chapitre 5-II).

VI CONCLUSION

VI.1 Méthode Continuous Integration Unified Process

La méthode *Continuous Integration Unified Process* a été conçue pour satisfaire l'objectif élémentaire de modéliser une application suivant un *processus de développement permettant le prototypage rapide en séance d'analyse*. Elle offre un cadre de travail au concepteur qui présente l'avantage de concentrer la participation des acteurs aux deux principales activités qu'ils ont à assurer au cours d'un développement : l'analyse et la validation des produits.

Afin d'atteindre cet objectif élémentaire, il a été nécessaire d'intégrer dans notre méthode deux pratiques de la méthode eXtreme Programming : la pratique du *Client sur site* et celle de l'*Intégration Continue*.

Enfin pour atteindre l'objectif de prototypage rapide, il sera nécessaire de concevoir et d'implémenter des transformations de modèles permettant de mettre en œuvre un processus full MDA qui automatise l'évolution d'un modèle depuis l'analyse jusqu'à l'implémentation. C'est l'objet des chapitres suivants.

VI.2 Software Development Process Approach et Software Development Process Model

L'objectif de capitalisation qui sous-tend ce travail a conduit à ré-analyser l'approche MDA et à montrer que l'objectif de l'approche MDA n'est qu'une des facettes d'un problème de capitalisation plus général que tout chef de projet, concepteur ou programmeur rencontre au cours du développement d'une application. C'est ce constat qui a conduit à proposer et à définir une nouvelle approche, appelée *Software Development Process Approach*, qui généralise l'approche MDA.

Tout comme l'approche MDA, la *Software Development Process Approach* est une nouvelle façon de piloter le développement d'une application mais la gestion de plusieurs modèles est une tache difficile qui rebutera tout concepteur et les entreprises si leur gestion n'est pas automatisée.

Afin que la *Software Development Process Approach* proposée soit réellement mise en œuvre, il a été nécessaire de concevoir et de réifier un artefact de développement permettant d'opérationnaliser cette approche. Cet artefact est le *Software Development Process Model* qui est un artefact multimodèle.

Cette approche et cet artefact présentent la particularité de pouvoir être adaptés à n'importe quelle méthode de conduite de projet et de pouvoir être étendus à volonté selon le besoin du concepteur.

Pour satisfaire l'objectif général, nous avons défini un *Software Development Process Model* spécifique à la méthode *Continuous Integration Unified Process*. Pour ce faire, nous avons précisé et défini tous les « ingrédients » de cet artefact multimodèle. Selon notre vision, cet artefact est monolithique et constitue LE MODÈLE de l'application.

La gestion de cet artefact multimodèle a nécessité de concevoir une *Transformation de diffusion* de concepts entre les différents modèles qui, selon le contexte de chaque concept, réalise soit une opération de clonage, soit une opération de mise à jour et ce de façon transparente pour le concepteur.

Le maintien en cohérence de l'artefact multimodèle *Software Development Process Model* dans le processus itératif de la méthode *Continuous Integration Unified Process* a quant à lui imposé la mise en œuvre d'une architecture de traçabilité de clonage orthogonale à celle multimodèle de cet artefact.

Grâce au *Software Development Process Model*, l'objectif élémentaire de *capitalisation des connaissances* pourra être atteint puisque chaque sous-modèle de cet artefact incorpore les concepts indispensables à la réalisation de la phase de développement associée. De plus, comme depuis l'analyse jusqu'à l'implémentation chaque sous-modèle est un sous-ensemble du sous-modèle suivant et que chaque concept a un lien de traçabilité unique avec son clone, l'ensemble des concepts introduits au cours d'une phase de développement est constitué des concepts exempts de lien de traçabilité avec un concept source.

Par sa définition (cf. définition 8), le sous-modèle d'analyse est le modèle qui correspond le mieux au modèle CIM (cf. Chapitre 2-I.2) de l'approche MDA. Le sous-modèle d'analyse est l'un des ingrédients de l'artefact *Software Development Process Model* et son articulation avec les autres sous-modèles est parfaitement définie.

Nous avons aussi vu que la description d'un processus de développement à l'aide d'un diagramme d'état s'avère être un excellent outil de description des activités de modélisation à réaliser sur chacun des sous-modèles.

Enfin, la vision ensembliste des modèles impliqués dans un processus de développement (cf. I.2) montre que la différence entre un modèle CIM et un modèle PIM est très ténue. La conséquence de cette vision est que la principale différence réside dans le vocabulaire employé.

Le guide de l'approche MDA (Miller *et al.*, 2003) conseille d'utiliser le vocabulaire des acteurs dans le modèle CIM. Les modèles PIM sont la plupart du temps réalisés avec des noms d'implémentation.

Nous avons signalé dans la rubrique Avertissements qu'une fonctionnalité de nommage a été conçue et implémentée au sein d'un *Profil de Nommage* permettant de convertir les noms des concepts issus du domaine des acteurs, de domaines connexes ou du domaine informatique, *noms sémantiquement riches*, en noms interprétables par un compilateur ou un interpréteur, *noms d'implémentation*.

Disposant d'une telle fonctionnalité, la modélisation d'une application peut être effectuée avec des noms sémantiquement riches depuis l'analyse jusqu'à l'implémentation, les noms sémantiquement riches noms étant convertis en noms d'implémentation juste avant la génération de code.

La diffusion des concepts et les transformations de modèles que nous verrons au Chapitre 5 s'exécutent de la même façon que ce soit des noms sémantiquement riche ou d'implémentation.

La vision ensembliste des modèles impliqués dans un processus de développement et la fonctionnalité de nommage dont nous disposons nous ont amené à ne plus différencier les modèles CIM et PIM et à fondre les premiers dans les seconds.

Étant donné que toutes les activités de modélisation décrites en figure 38 sont des activités manuelles, le but des chapitres suivants est de concevoir les transformations qui vont permettre d'obtenir un processus full MDA pour le développement de Systèmes d'Information Géographique.

Chapitre 4. Concepts UML supports des concepts SIG et des entités référencées - Concepts SIG implémentés dans le Profil UML-SIG - Incidence sur les langages SIG et pictogrammique

Avertissement : Les concepts UML adoptés pour représenter les entités référencées et leurs propriétés de spatialité et de temporalité ont une incidence sur les langages SIG et pictogrammique formalisés au Chapitre 7, incidence qui est représentée sur le *Métamodèle SIG* décrit lui aussi au Chapitre 7. Il est donc recommandé de s'imprégner des concepts majeurs du Chapitre 7.

Le précédent paragraphe a présenté les idées novatrices qui vont permettre d'atteindre les trois objectifs élémentaire visés que sont la réalisation d'un *outil d'aide à la conception de Systèmes d'Information Géographique*, la mise en œuvre de cet outil dans un *processus de développement permettant le prototypage rapide en séance d'analyse* et la *capitalisation de connaissances* au sein de cet outil (cf. Introduction-IV).

Ces idées ont conduit à définir la méthode *Continuous Integration Unified Process* et à celle de l'artefact *Software Development Process Model*. Or pour appliquer la méthode *Continuous Integration Unified Process*, il faut implémenter l'artefact *Software Development Process Model*. Nous avons fait le choix d'utiliser un atelier de génie logiciel car le niveau de métaprogrammation permet de vérifier plus rapidement les idées novatrices.

Fin 1999, la Direction Scientifique du Cemagref a retenu l'atelier de génie logiciel Objecteering de la société Softeam comme outil de modélisation et de développement des applications informatiques et des systèmes d'information. C'est un logiciel sous licence propriétaire dont le code n'est pas libre. Pour éviter cet inconvénient, les concepteurs d'Objecteering ont implémenté un mécanisme d'extension du métamodèle du langage UML appelé *Profil* qui permet d'enrichir le métamodèle des concepts d'un domaine particulier sans en modifier le métamodèle (Atkinson *et al.*, 2000). Ce mécanisme de *Profil* est décrit dans les spécifications du langage UML (OMG, 2003a, 2005, 2006) et par de nombreux auteurs (Blanc, 2001 ; Blanc *et al.*, 2002 ; Crégut *et al.*, 2005 ; Riccobene *et al.*, 2004 ; Softeam, 2003a) dont la liste exhaustive serait difficile à établir. L'extension du métamodèle est effectuée principalement grâce aux concepts UML de *Stéréotype*, de *Valeur marquée* (cf. paragraphes III.7.1 et III.7.2 de l'Annexe I) et de *Contraintes*.

Étendre un métamodèle avec de nouveaux concepts est une chose. Exploiter les informations que constituent ces nouveaux concepts en est une autre. Pour cela, il faut pouvoir implémenter de nouvelles fonctionnalités dans

l'atelier de génie logiciel. Objecteering dispose d'un langage de métaprogrammation, appelé J (Softeam, 2003a), qui permet de faire de l'introspection du modèle, de vérifier l'état d'un concept, de créer des interfaces, etc. Toutes ces nouvelles fonctionnalités sont attachées au profil.

Soucieux de doter l'établissement d'un outil de modélisation qui puisse être déployé dans différentes équipes, nous avons fait le choix d'implémenter l'artefact Software Development Process Model dans l'atelier de génie logiciel Objecteering.

Toutes les fonctionnalités de création de l'artefact *Software Development Process Model* et de ses sous-modèles ont été implémentées dans un profil appelé *Profil MDA*. Les transformations de gestion des concepts sont aussi réifiées au sein du profil.

Par ailleurs, la réalisation d'un *outil d'aide à la conception de Systèmes d'Information Géographique* étant l'un des objectifs élémentaire fixés, nous avons fait le choix de doter cet outil des langages SIG et pictogrammique (cf. Chapitre 1-V et Chapitre 7). Ces langages ont été étendus aux concepts de spatialité floue et de temporalité floue (Chapitre 8) car aucun des formalismes de conception de *Systèmes d'Information Géographique* n'implémente cette typologie de concepts très utiles pour décrire les concepts thématiques du domaine d'intervention privilégié des scientifiques du Cemagref : l'agriculture et l'environnement.

Ces deux langages ont été implémentés dans un profil indépendant du précédent que nous avons appelé Profil UML-SIG.

Pour réaliser le *Profil UML-SIG*, il faut choisir dans une première étape le concept UML qui va servir de support aux entités référencées et dans une seconde étape celui qui sera utilisé comme support des propriétés spatiales et temporelles des entités référencées.

Bien que très intéressant car il permet souvent d'étendre rapidement et à moindre coût les capacités et les fonctionnalités d'un atelier de génie logiciel, le mécanisme de profil, lorsque l'atelier utilisé est sous licence propriétaire, présente l'inconvénient d'imposer le comportement des concepts UML choisis.

Le concept UML choisi comme support de la spatialité et de la temporalité nous a imposé des contraintes d'implémentation (cf. II) qui ont nécessité de définir des règles de construction des noms internes des concepts SIG composites (cf. III).

Ces contraintes sont à l'origine d'une implémentation limitée des langages SIG et pictogrammique dans le *Profil UML-SIG* (cf. IV).

Sommaire détaillé

I CONCEPTS UML SUPPORTS DES ENTITES REFERENCEES DANS LE PROFIL UML-SIG

Les Entités référencées sont des concepts thématiques qui possèdent des propriétés de spatialité et de temporalité (cf. définition 3). Pour les modéliser en langage UML, il a été nécessaire de mobiliser deux concepts UML : un premier pour représenter le concept thématique annoté par le second qui confère le statut d'entité référencée au premier.

En langage UML, une entité référencée est donc représentée par un couple de deux concepts, concepts dont nous allons approfondir l'étude dans les deux paragraphes suivants.

I.1 Concept UML support de la propriété thématique des entités référencées

Les Entités référencées sont des concepts qui ont des propriétés et, en particulier, celles de spatialité et de temporalité mais elles ont aussi des comportements. En modélisation, les propriétés sont des spécifications statiques alors les comportements relèvent des spécifications dynamiques. Pour modéliser cette double typologie, le concept UML le plus approprié est sans aucun doute le concept de *Classe* (cf. Annexe I-III.1) puisque les concepts UML d'*Attribut* et d'*Opération* lui sont attachés. Le premier permet de représenter les spécifications statiques et le second les spécifications dynamiques.

Tout naturellement, le concept de* Classe *a donc été retenu comme concept UML support de la propriété thématique des Entités référencées.

Ce choix est identique à celui de Perceptory qui est aussi fondé sur le langage UML. Il est équivalent d'une part, à celui MADS puisque le concept *Object* de MADS est homologue à celui de *Classe* en UML (Parent *et al.*, 2006 ; Parent *et al.*, 1997a) et, d'autre part, à celui de STER puisque le concept *Entity Set* est le pendant dans le formalisme Entité/Relation du concept de *Classe* en UML (Tryfona *et al.*, 1998a, 1998b ; Tryfona *et al.*, 1999).

Toutefois, ces trois formalismes utilisent deux autres concepts comme support des entités référencées. Le premier est celui d'*Attribut*[43] et le second est la relation reliant deux concepts qui, en langage UML est généralement instanciée par le concept d'*Association*[53] (Parent *et al.*, 2006 ; Parent *et al.*, 1997a ; Tryfona *et al.*, 1998a, 1998b ; Tryfona *et al.*, 1999).

Dans la phase de faisabilité actuelle du Profil UML-SIG, les concepts d'*Attribut *et de d'*Association *n'ont pas été retenus. Nous avons préféré concentrer l'effort d'implémentation au seul concept de* Classe *et de différer la réflexion et l'implémentation en* Attribut *et en* Association *à une étape ultérieure.

Il faut signaler toutefois que la dualité Attribut/Association (cf. Annexe IV) peut, dans certains cas, être une solution d'autant plus que la transformation d'un attribut en association et inversement a été automatisée. Il faut aussi rappeler que, dans un modèle, une association peut être convertie, dans certaines conditions en *Classe Association* et qu'il est alors possible d'annoter la classe association avec les concepts SIG.

I.2 Concept UML support de spatialité et de temporalité

Comme indiqué au paragraphe III.7.1 de l'Annexe I, le concept de *Stéréotype* est l'élément de modélisation de prédilection pour véhiculer une information capitale ou exprimer une propriété majeure d'un concept. En outre, le langage UML prévoit la possibilité d'associer à chaque stéréotype un pictogramme[44] (OMG, 1999, 2001a, 2003a, 2005, 2006).

Les caractéristiques du concept de *Stéréotype* répondent à nos deux principales préoccupations qui sont d'une part, d'ajouter aux entités référencées les propriétés majeures que sont la spatialité et la temporalité et, d'autre part, d'exprimer ces propriétés via un langage pictogrammique.

Le concept de* Stéréotype *a donc été adopté pour exprimer la spatialité et la temporalité des Entités référencées en langage UML.

---

[43] Le choix d'implémenter les entités référencées sous forme d'attribut dans Perceptory peut, dans certains contextes être en désaccord avec les recommandations de Craig Larman (Larman, 2002a) qui préconise la représentation sous forme d'Association lorsque le concept typant l'attribut est de nature complexe. En particulier, l'annotation des attributs avec des concepts SIG composites ne respecte pas la préconisation de Craig Larman.

[44] Certains ateliers de génie logiciel n'offrent pas cette possibilité.

I.3 En résumé

Lors d'une modélisation en langage UML, une entité référencée est représentée par une* Classe *annotée d'un* Stéréotype *de spatialité et/ou de temporalité.

Le choix du concept de *Stéréotype* n'est pas sans conséquence. Le paragraphe suivant présente les contraintes induites par le concept de *Stéréotype* et les adaptations imposées au langage pictogrammique implémenté dans le *Profil UML-SIG*.

II CONTRAINTES IMPOSEES PAR L'ATELIER DE GENIE LOGICIEL

II.1 Unicité de stéréotypage

Le diagramme de la figure 43 présente un extrait du métamodèle de l'atelier de génie logiciel Objecteering. Il montre les méta-attributs des métaclasses *ModelElement* et *Stéréotype* ainsi que les relations entre ces deux métaclasses (Softeam, 2003a).

Le concept de *Stéréotype* possède deux méta-attributs et en hérite un troisième de la métaclasse *ModelElement*. Le premier, *BaseClass*, spécifie le nom de l'élément de modélisation porteur du stéréotype. Ayant fait le choix de représenter les Entités référencées par le concept UML de *Classe* (cf. I.1), le méta-attribut *BaseClass* aura toujours pour valeur la chaîne de caractères « Class ». Le deuxième, *Icon*, stocke le chemin du fichier image associé au stéréotype. Enfin, le troisième, *Name*, porte le nom du concept thématique ajouté au métamodèle.

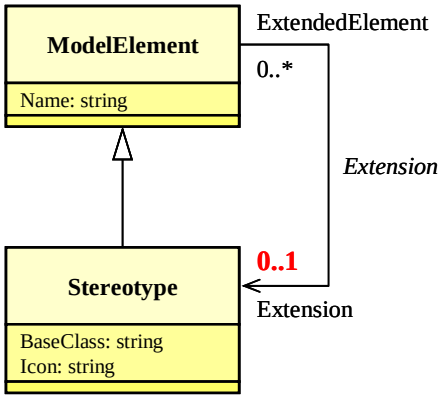


Figure 40 Extrait du métamodèle d'Objecteering.

D'après la figure 43, tout élément de modélisation peut avoir au maximum* un et un seul *stéréotype puisque la multiplicité du méta-rôle de la méta-association* Extension *est 0..1.

Cette multiplicité date de la version 1.3 du langage UML (OMG, 1999). Depuis la version 1.4 d'UML, la multiplicité est de 0..* (OMG, 2001a) et permet le multi-stéréotypage d'un élément de modélisation.

II.2 Unicité de position du pictogramme

Dans l'atelier de génie logiciel Objecteering, les stéréotypes renseignant les classes sont affichés, sous forme textuelle, au-dessus du nom de la classe ou, sous forme pictogrammique, dans le coin supérieur à droite (cf. Annexe I-III.7.1).

La position du stéréotype est gérée en interne par l'atelier de génie logiciel. De plus, elle est* imposée *puisque l'atelier de génie logiciel n'offre aucune fonctionnalité pour repositionner le pictogramme.

II.3 Conséquences sur les langages SIG et pictogrammique

L'unicité de stéréotypage et l'unicité de position du pictogramme ont eu trois conséquences importantes sur les langages SIG et pictogrammique :

- ⇨ La première est toujours liée à l'unicité de stéréotypage en relation avec le méta-attribut *Name*. Il a été nécessaire de définir une règle de gestion des noms des stéréotypes composites de bi-spatialité. Cette règle est présentée au paragraphe III.
- ⇨ La deuxième est induite par l'unicité de stéréotypage. Elle a imposé de réaliser pour chaque stéréotype une image représentant les propriétés de spatialité et de temporalité portées par le nom du stéréotype. Au vu du nombre de combinaisons possibles, nous avons préféré limiter la capacité d'expression du langage pictogrammique du *Profil UML-SIG*. Les concepts SIG finalement implémentés dans le profil sont listés au paragraphe IV.
- ⇨ La troisième résulte de l'unicité de position du pictogramme. Elle a conduit à modifier ou à déroger à certaines conventions du langage pictogrammique définies par l'équipe d'Yvan Bédard (cf. Chapitre 7). Cela s'est traduit par une redéfinition partielle du langage pictogrammique. Les différences entre les deux langages sont présentées au paragraphe V.

III REGLE DE CONSTRUCTION DES NOMS DES CONCEPTS SIG COMPOSITES DE BI-SPATIALITE[45]

Au paragraphe II.1, nous avons vu que *Name* est le seul méta-attribut disponible pour introduire les concepts SIG dans le *Profil UML-SIG*.

Dans la mesure où le nom du concept SIG est simple, créer un stéréotype portant ce nom ne pose aucune difficulté. Par contre, en absence de multi-stéréotypage le seul moyen d'ajouter les concepts composites de bi-spatialité est de mettre en place sous forme de règle un mécanisme de concaténation entre les deux noms des stéréotypes composites de bi-spatialité permettant d'avoir un nom unique.

La règle de construction retenue est d'intercaler entre les deux noms un symbolisme rappelant la loi de composition. Par exemple, le nom d'un stéréotype de spatialité multiple résulte de la concaténation des deux noms des spatialités primitives séparés par l'opérateur « *&&* » équivalent à l'opérateur mathématique $\otimes$ dans plusieurs langages de programmation.

De fait, l'unicité de stéréotypage a conduit à implémenter dans le Profil UML-SIG sous forme de stéréotype le Langage SIG de la figure 98 incluant les noms des singletons composites de bi-spatialité.

Le Tableau 7 fournit, pour chaque type de spatialité composite, la correspondance entre la formulation mathématique et l'opérateur de concaténation des noms ainsi qu'un exemple d'application de la règle :

Type de Spatialité	Loi de composition[46]	Règle de construction	Exemple[47]
Spatialité Multiple	$s^{\otimes} = s_{p_1} \overset{s}{\otimes} s_{p_2}$	$\underline{S_1}$ **&&** $\underline{S_2}$	*Agglomération*<<⊡ *Point* **&&** *Polygone*
Spatialité Alternative	$s^{\oplus} = s_{p_1} \overset{s}{\oplus} s_{p_2}$	$\underline{S_1}$ **||** $\underline{S_2}$	*Bâtiment*<<⊡▣ *Point* **||** *Polygone*

[45] Les concepts composites de bi-temporalité n'ont pas été implémentés dans le Profil UML-SIG (cf. paragraphe suivant)
[46] Cf. les définitions 33, 35, 37 et 38.
[47] Cf. au Chapitre 7 les exemples des paragraphes IV.2.1.1.b, IV.2.1.2.b, IV.2.2.1.b et IV.2.2.2.b.

Spatialité et Temporalité Conjointe	$st^{\overset{ST}{\otimes}} = s_p \overset{ST}{\otimes} t_p$	S **&&** T	*Bâtiment* *Polygone* **&&** *Période*
Spatio-Temporalité	$s^{\overset{s_T}{<<}t} = s_p \overset{s_T}{<<} t_p$	T >> S	*Moissonneuse* *Instant* >> *Point*

Tableau 5 Règle de construction des noms des stéréotypes composites de bi-spatialité.

IV LANGAGE SIG DU PROFIL UML-SIG : DIFFERENCES AVEC PERCEPTORY

Les concepts de spatialité et de temporalité nécessaires à la conception de *Systèmes d'Information Géographique* ont fait l'objet de nombreux travaux au cours de ces dernières années de la part de chercheurs, d'organismes de recherche, d'instances de normalisation, etc. (cf. Chapitre 1 et paragraphe I du Chapitre 7). Les concepts SIG primitifs de spatialité, *Point*, *Ligne* et *Polygone*, et de temporalité, *Instant* et *Période*, sont communs à tous les formalismes, méthodes et outils de modélisation utilisés pour le développement de systèmes d'information et d'applications informatiques du domaine de l'*Information Géographique.*

Ces concepts SIG primitifs auxquels nous avons ajouté le concept Volume*[48] *ont tous été adoptés et implémentés dans le Profil UML-SIG.

Dans un contexte de multi-stéréotypage et de maîtrise de la position du pictogramme, il aurait été possible de réaliser les concepts SIG composites et les pictogrammes associés à partir des concepts SIG primitifs. C'est l'implémentation réalisée dans Perceptory.

Dans le contexte d'unicité de stéréotypage et d'unicité de position, l'implémentation des stéréotypes primitifs ne pose aucun problème mais celle des stéréotypes composites devient rapidement une gageure. En effet, le nombre de combinaisons de noms et d'images à créer devient rapidement prohibitif en fonction du nombre de lois de composition adoptées.

Dans la phase de faisabilité actuelle, nous avons préféré limiter l'implémentation des concepts SIG composites aux quatre lois de composition : la loi d'addition spatiale $\overset{S}{\otimes}$***, la loi de substitution de spatialité*** $\overset{S}{\oplus}$***, la loi d'addition de spatialité et de temporalité*** $\overset{ST}{\otimes}$ ***et la loi d'affectation de temporalité à une spatialité*** $\overset{s_T}{<<}$***. Les concepts SIG dérivés n'ont pas été retenus pour la même raison ainsi que les concepts de spatialité et temporalité particulières. Enfin, la cardinalité des concepts SIG est prise en compte mais la convention pictogrammique associée n'a pas pu être respectée (cf. V.3).***

Tous ces choix sont résumés en figure 44. Elle montre en gris les concepts SIG et les lois de composition qui n'ont pas été implémentés dans le *Profil UML-SIG*.

Malgré ces simplifications, l'application de la règle de construction des noms de stéréotype composite de bi-spatialité (cf. III) aux 6 concepts SIG primitifs et leurs homologues flous (cf. Chapitre 8) combinés aux quatre lois de composition ci-dessus s'est traduite par la création de 176 stéréotypes et le double de fichiers images[49].

Ces choix de simplification ont été adoptés en connaissance de cause car le verrou technologique imposé par l'unicité de stéréotypage sera levé avec la mise sur le marché d'une version de l'atelier de génie logiciel implémentant le langage UML 2.0. Le multi-stéréotype[50] devrait alors permettre de doter le *Profil UML-SIG* des mêmes capacités d'expression pictogrammique que l'atelier de génie logiciel Perceptory.

[48] Voir les raisons en partie Avertissements.
[49] Chaque stéréotype gère 3 pictogrammes de tailles différentes. Ils sont utilisés par l'atelier de génie logiciel suivant le contexte graphique. Dans le profil UML-SIG, il a été possible de réduire le nombre pictogrammes à 2.
[50] La version 6 d'Objecteering qui vient d'être mise sur le marché devrait corriger ce problème puisqu'elle implémente le langage ULM 2.0.

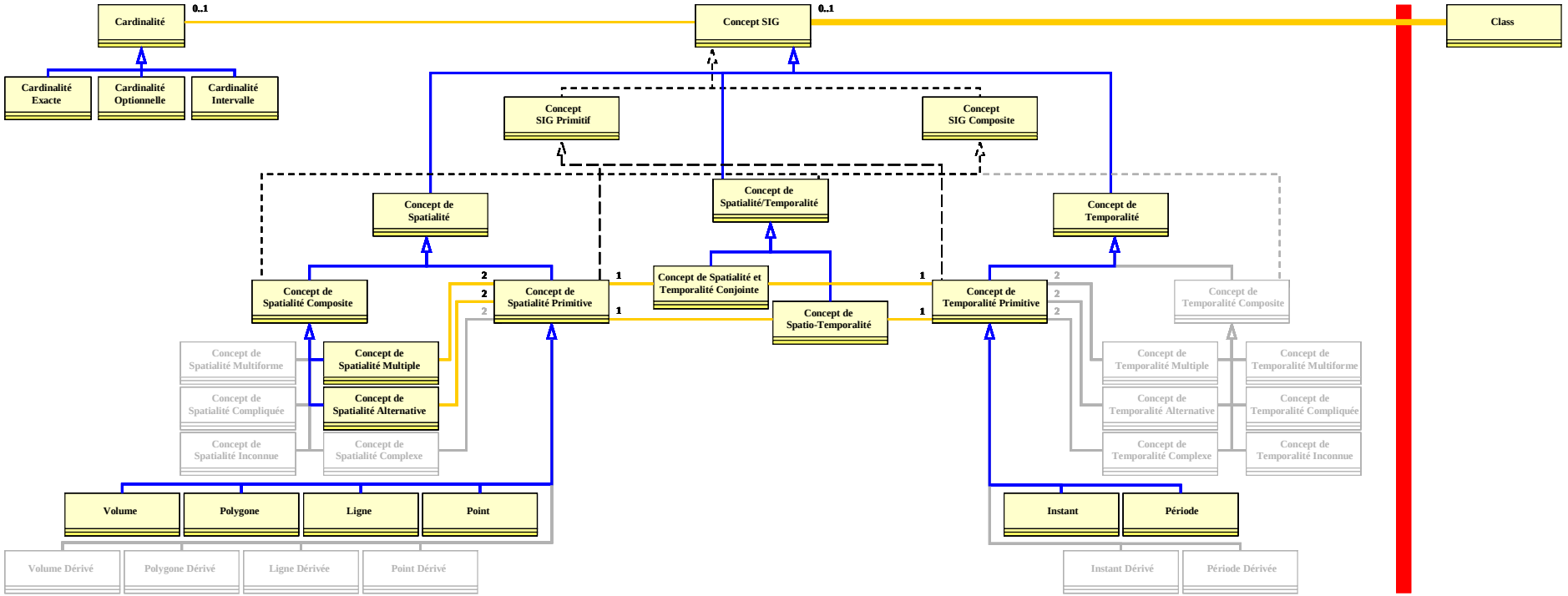


Figure 41 *Métamodèle SIG* représentant en gris les concepts SIG non implémentés dans le *Profil UML-SIG*.

V LANGAGE PICTOGRAMMIQUE DU PROFIL UML-SIG : DIFFERENCES AVEC PERCEPTORY

V.1 Spatialité et temporalité conjointe

La première différence est la plus importante. Elle est due à l'unicité de position du pictogramme (cf. II.2) qui a empêché de respecter la convention pictogrammique de spatialité et temporalité conjointe énoncée au Chapitre 7-IV.2.2.1 qui stipule que les propriétés de spatialité sont annotées à gauche et celles de temporalité à droite (cf. figure 45-Modèle A). Cette convention ne pouvant être respectée, il a été nécessaire d'en adopter une autre.

Étant donné que la loi d'addition de spatialité et de temporalité $\overset{ST}{\otimes}$ est de nature additive, nous avons emprunté la convention pictogrammique de la loi d'addition spatiale $\overset{S}{\otimes}$ pour exprimer la spatialité et de temporalité conjointe. Elle consiste à superposer le pictogramme de spatialité et le pictogramme de temporalité comme le montre le modèle B de la figure 45.

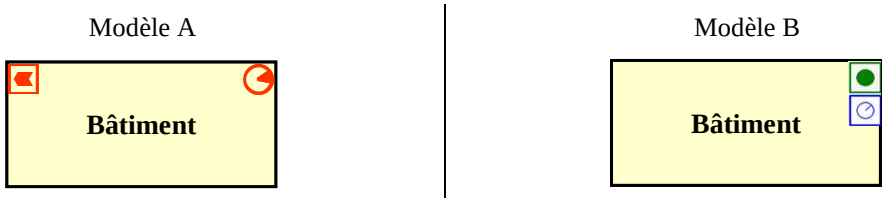


Figure 42 Exemple de pictogramme de spatialité et de temporalité conjointe dans Perceptory (modèle A) et dans le *Profil UML-SIG* (modèle B).

Cependant afin de retrouver en partie la différence de position existant initialement, nous avons introduit la couleur. Les pictogrammes de spatialité ont donc été coloriés en vert et ceux de temporalité en bleu. Le tableau 6 montre les différences de couleurs. Il montre aussi :

- D'une part, une légère différence dans le pictogramme de ligne. Cette différence sera prochainement corrigée par alignement du pictogramme du *Profil UML-SIG* sur celui de Perceptory. Il est vrai que le pictogramme du *Profil UML-SIG* peut induire la perception du concept de *Segment* ce qui ne correspond pas à son implémentation réelle.
- Et, d'autre part, les pictogrammes temporels ont été inclus dans un carré comme ceux de spatialité. Ce choix a été fait car il améliore nous semble-t-il la perception des concepts temporels dérivés. L'inclinaison du pictogramme de spatialité polygonale dérivée apparaît mieux et est plus lisible que celle des pictogrammes de temporalité et .

	Point	*Ligne*	*Polygone*	*Volume*	*Instant*	*Période*
Profil UML-SIG						
Perceptory						

Tableau 6 Pictogrammes SIG primitifs dans Perceptory et le *Profil UML-SIG*.

V.2 Spatio-temporalité

La deuxième différence réside dans la composition des pictogrammes de spatio-temporalité. Dans Perceptory, la spatio-temporalité d'une entité référencée est notée par un pictogramme de temporalité primitive

accolé à droite d'un pictogramme de spatialité (cf. Chapitre 7-IV.2.2.2). Le modèle A de la figure 46 reprend ici le modèle de l'entité référencée *Moissonneuse*<<>> du paragraphe IV.2.2.2.b du Chapitre 7.

Dans le *Profil UML-SIG*, un choix légèrement différent a été effectué afin que la pictogrammie des spatio-temporalités soit en conformité avec l'application d'affectation d'une temporalité à une spatialité (cf. définition 38). Cette loi précise que la propriété de temporalité renseigne celle de spatialité. Aussi, la convention adoptée dans le langage pictogrammique du *Profil UML-SIG* a été d'inclure le pictogramme de temporalité dans le pictogramme de spatialité (cf. figure 46-Modèle B).

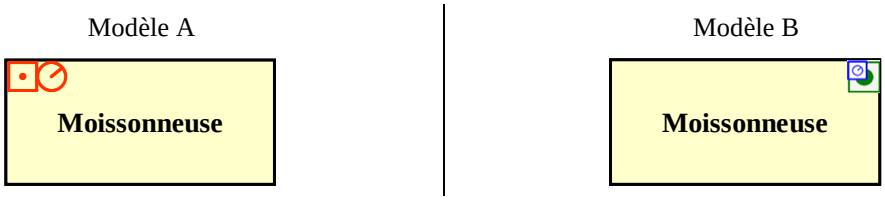


Figure 43 Exemple de pictogramme de spatio-temporalité dans Perceptory (modèle A) et dans le *Profil UML-SIG* (modèle B).

Cette nouvelle convention pictogrammique nécessite une redéfinition de l'ensemble des pictogrammes de spatio-temporalité[51] :

$$\overset{vl}{S_T} = f_{|S_T}(S_T) = \{ \; , \; , \; , \; , \; , \; , \; , \; \}$$

V.3 Cardinalité

La troisième différence porte sur la cardinalité. La convention graphique imposant que la cardinalité soit accolée au pictogramme qu'elle renseigne (cf. Chapitre 7-IV.4) n'a pas pu être respectée car le concept UML *Stéréotype* ne dispose pas de fonctionnalité pour cela.

La cardinalité est donc gérée par le *Profil UML-SIG*, elle est consultable et modifiable au niveau de l'interface de saisie de la spatialité et de la temporalité comme le montre la figure 47 mais elle n'est visible dans le modèle lui-même.

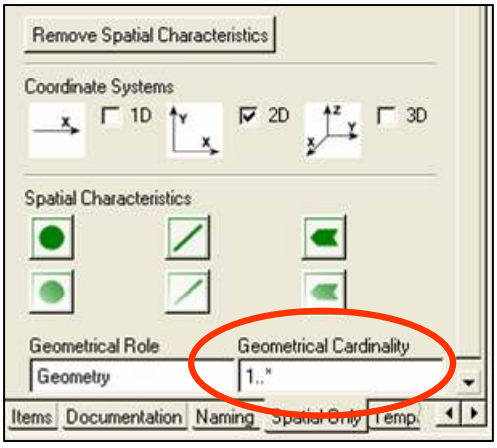


Figure 44 Expression de la cardinalité dans l'interface du *Profil UML-SIG*.

[51] Par souci d'homogénéité par rapport au Chapitre 7, les pictogrammes SIG dérivés ne sont pas introduits.

VI CONCLUSION

Des verrous technologiques imposés par notre choix d'implémenter le *Profil UML-SIG* dans notre atelier de génie logiciel propriétaire ont conduit à limiter la capacité d'expression du langage pictogrammique du *Profil UML-SIG*.

Ces verrous technologiques devraient être levés avec la mise sur le marché de la version 6 de l'atelier de génie logiciel Objecteering qui doit implémenter la version 2.0 du langage UML.

Chapitre 5. Conception des transformations de l'artefact Software Development Process Model

Avertissement : La conception des transformations géomatiques dérive de l'étude terminologique effectuée au Chapitre 7. Afin de bien appréhender la conception des transformations décrites dans le présent chapitre, il est souhaitable de s'imprégner de deux des concepts majeurs décrits au Chapitre 7 : l'application d'affectation **<<** entre l'ensemble ***E*** des entités référencées et l'ensemble ***U*** des concepts SIG (cf. Chapitre 7-I) et le *Métamodèle SIG* dérivé de l'étude terminologique (cf. Chapitre 7-VII). Ce dernier a été étendu aux concepts de spatialité floue et temporalité floue afin de faciliter la modélisation des entités référencées du domaine environnemental (cf. Chapitre 8).

Après avoir défini au Chapitre 3 la méthode *Continuous Integration Unified Process*, l'approche *Software Development Process Approach* et l'artefact de modélisation *Software Development Process Model*, nous avons consacré le Chapitre 4 à décrire les contraintes imposées par l'atelier de génie logiciel.

Reste à concevoir et à implémenter tous les concepts définis afin d'atteindre l'objectif élémentaire visé de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique* adapté à un *processus de développement permettant le prototypage rapide en séance d'analyse* assurant la *capitalisation des connaissances* (cf. Introduction-IV).

La conception des transformations est au cœur du présent chapitre et l'implémentation au cœur du chapitre suivant.

Le paragraphe I pose quelques principes généraux qui ont guidé la conception des transformations afin d'assurer l'évolution du modèle dans un processus full MDA.

Le paragraphe II affine la description de la *Transformation de diffusion* qui assure le transfert des concepts depuis un sous-modèle d'analyse jusqu'aux sous-modèles d'implémentation et introduit la transformation qui permet la rétrodiffusion des concepts. Ce paragraphe présente aussi le principe de prétraitement et de post-traitement associés aux transformations de diffusion et de rétrodiffusion qu'il faut parfois implémenter afin de maintenir le *Software Development Process Model* en cohérence dans le processus itératif adopté.

Les paragraphes III et IV sont consacrés à la description des transformations géomatiques et respectivement SQL qu'il a été nécessaire de concevoir afin que le processus de conception des *Systèmes d'Information Géographique* soit full MDA.

Après avoir défini le concept UML utilisé comme support de l'application d'affectation **<<**, les transformations géomatiques sont décrites au paragraphe III. Elles sont au nombre de quatre.

La première assiste le concepteur lors de la saisie des concepts SIG du langage pictogrammique de Perceptory (cf. Chapitre 7) étendu aux concepts de spatialités floue et de temporalité floue (cf. Chapitre 8). La deuxième facilite la suppression des concepts SIG.

La troisième décrit le *Patron de conception SIG* qui est automatiquement généré au sein de l'artefact *Software Development Process Model* indépendamment du modèle en cours d'élaboration.

La quatrième convertit le langage pictogrammique en éléments de modélisation UML qui établissent une relation entre les entités référencées et les concepts du *Patron de conception SIG*.

Les transformations SQL qui adaptent automatiquement le modèle à la génération de code SQL sont décrites au paragraphe IV. Elles sont au nombre de trois.

La première ajoute la persistence aux classes, prédéfinit une clé primaire et prédétermine le traitement de la relation de généralisation.

La deuxième évite la perte des relations que peut avoir une classe dénudée d'attributs et ayant des opérations avec les classes « environnantes ».

La troisième externalise la relation de composition qui, dans certaines configurations, est à l'origine d'un message d'avertissement récurrent.

Enfin, le paragraphe V conclut.

Sommaire détaillé

I LES PRINCIPES QUI ONT PRESIDE NOTRE DEMARCHE DE CONCEPTION DES TRANSFORMATIONS

La pratique prônée par certaines méthodes de développement de considérer le prototype comme étant un produit « Kleenex », c'est-à-dire « à jeter » (Région Wallonne, 2004) une fois validé, est incompatible avec l'objectif élémentaire de *capitalisation des connaissances* que nous nous sommes fixé. Aussi, nous avons adopté un premier principe pour tenir compte de cet objectif général :

Principe 1 : Le prototype et l'application forment un tout indissociable qui doit évoluer par incréments conformément au principe de la méthode *Continuous Integration Unified Process*.

Le cycle de prototypage rapide a été pensé pour améliorer la capture et la validation des concepts et ce dans un processus hautement itératif. La préoccupation fondamentale est la qualité sémantique du modèle, aussi il n'est nullement besoin d'atteindre LA PERFECTION TECHNIQUE. Cette seconde préoccupation de perfection technique peut être déléguée et réalisée au cours du cycle principal. Les membres de l'équipe ont alors tout le temps pour cela. Cette dichotomie est l'objet du second principe :

Principe 2 : Au cours du cycle de prototypage rapide, la qualité sémantique du modèle est recherchée alors que la qualité technique est réalisée au cours du cycle principal.

C'est ce second principe qui permettra de concevoir des transformations simples résolvant un problème de façon rapide et automatique.

L'imbrication voire la fusion du prototype et de l'application et leur(s) évolution(s) incrémentale(s) conduisent à fixer un troisième principe concernant la conception des transformations :

Principe 3 : Les transformations automatiques du cycle de prototypage rapide ne doivent jamais détruire ou affecter les ajouts de concepts ou les modifications réalisées par le concepteur au cours du cycle principal.

Les trois principes qui viennent d'être énoncés sont généraux et sous-tendent la conception des transformations permettant un processus full MDA pour la conception et le développement de *Systèmes d'Information Géographique*.

II TRANSFORMATIONS DE GESTION DE L'ARTEFACT SOFTWARE DEVELOPMENT PROCESS MODEL

Les transformations décrites dans ce paragraphe ont pour responsabilité de diffuser les concepts et les spécifications entre les différents sous-modèles de l'artefact *Software Development Process Model* et ce en assurant la cohérence de cet artefact dans un processus itératif.

II.1 Transformation de diffusion

Le principe général de la *Transformation de diffusion* $T_{Dif.}^{SM\ S \rightarrow SM\ D}$ a été décrit au paragraphe V du Chapitre 3 mais un mécanisme de diffusion non maîtrisé par le concepteur peut poser rapidement des problèmes lors de la génération de code. Aussi, des annotations d'invalidation de diffusion (Tagged Value) ont été créées afin d'annihiler la diffusion lorsqu'elles renseignent un concept. Le but recherché est d'offrir au concepteur un moyen de piloter la diffusion à son gré.

Par exemple, les concepts relatifs aux interfaces Homme/Machine n'ont pas lieu d'être dans le sous-modèle d'implémentation SQL car ce langage n'est pas adapté. Le concepteur aura tout simplement à marquer les concepts décrivant les interfaces Homme/Machine de l'annotation *{MDA : SQL Diffusion Freezing}* pour

annihiler leur diffusion dans le sous-modèle d'implémentation SQL sans affecter leur diffusion vers d'autres sous-modèles d'implémentation, Java par exemple.

À l'itération N, des opérations de clonage ayant été effectuées aux itérations précédentes, un concept peut avoir un lien de traçabilité de clonage ou non et être marqué ou non d'une annotation d'invalidation de diffusion. À l'itération N et pour un sous-modèle donné, un concept peut se trouver dans l'une des quatre situations suivantes :

- Un concept sans lien de traçabilité de clonage et sans annotation d'invalidation de diffusion, concept *Plot* en figure 45-Before, est un concept introduit par le concepteur ou par une des transformations géomatiques au cours de l'itération N.
- Un concept avec lien de traçabilité de clonage et sans annotation d'invalidation de diffusion, concept *Parcel*, est un concept présent à l'itération N-1 qui a été cloné. Ici, le concepteur a constaté une faute d'orthographe qu'il a corrigée et il a en outre ajouté un attribut *Area*.
- Le concept *Tractor* peut être présent à l'itération N-1 avec l'annotation d'invalidation de diffusion ou bien être ajouté au cours de l'itération N et immédiatement marqué pour annihiler sa diffusion.
- Enfin, un concept avec lien de traçabilité de clonage et avec annotation d'invalidation de diffusion suppose que le concept, *Farmer* en l'occurrence, est présent à l'itération N-1, qu'il est cloné et que le concepteur décide à l'itération N d'annihiler sa diffusion vers les sous-modèles suivants.

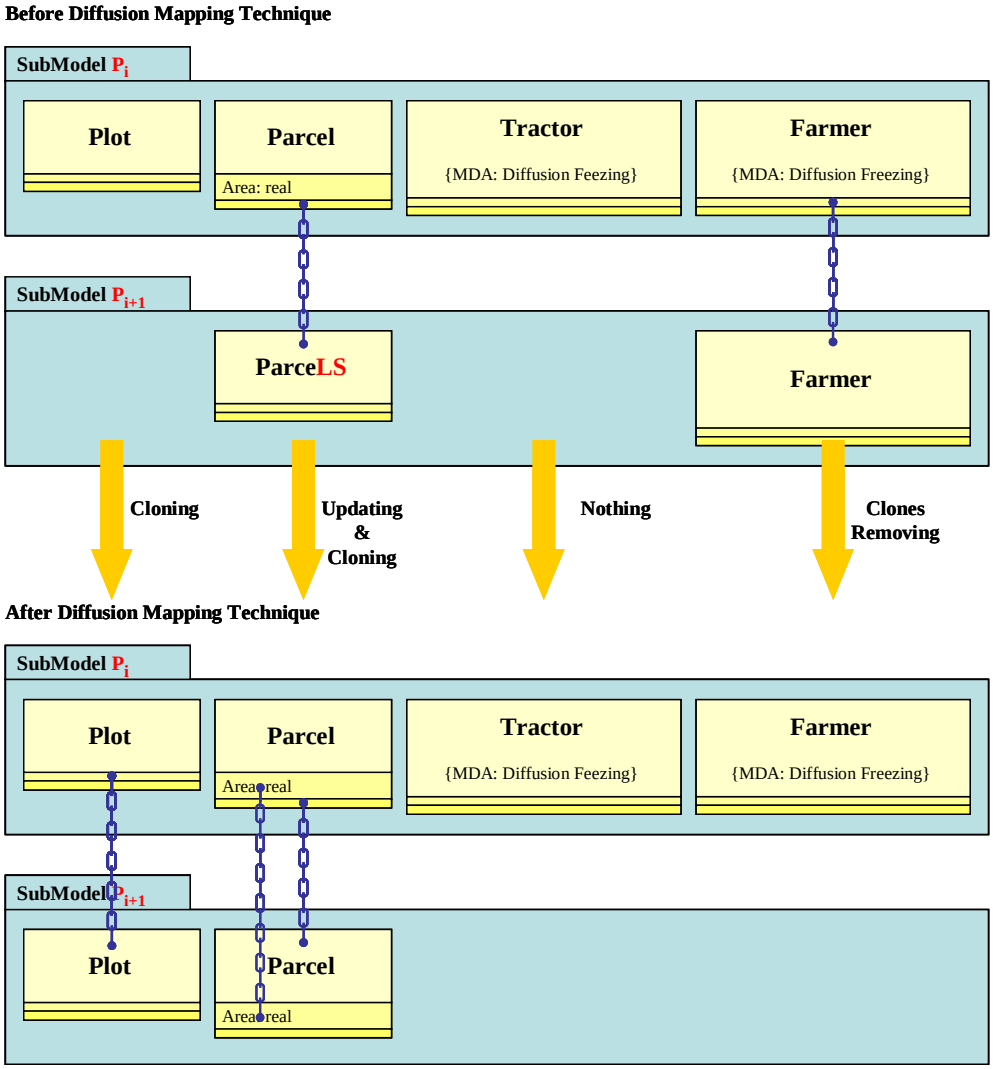


Figure 45 Les différentes opérations de la *Transformation de diffusion*.

La transformation $T_{Dif.}^{SM\ S \rightarrow SM\ D}$ a été conçue pour tenir compte de ces quatre situations. Le concept *Plot* est cloné avec tous les éléments de modélisation attachés -attributs, associations, etc.- s'il y en avait (cf. figure 45-After). Le concept *Parcel* est quant à lui mis à jour (ex : ParceLS remplacé par Parcel) et tous les concepts nouvellement attachés sont clonés par exemple l'attribut Area.

Marqués de l'annotation *{MDA : Diffusion Freezing}*, aucune opération n'est effectuée sur le concept *Tractor* alors que les clones du concept *Farmer* sont simplement et purement supprimés puisque c'est le souhait du concepteur.

La transformation $T_{Dif.}^{SM\ S \rightarrow SM\ D}$ décrite est une transformation complexe qui réalise plusieurs fonctions suivant le contexte local du concept à diffuser, contexte qui est lié aux activités d'ajouts et de modifications de concepts qui sont réalisées naturellement lors de la modélisation.

La *Transformation de diffusion* est typiquement une transformation générique qui s'applique à tous les sous-modèles de l'artefact *Software Development Process Model* excepté aux sous-modèles d'implémentation qui sont en fin de cycle (cf. figure 36). C'est une transformation inter-sous-modèles.

II.2 Transformation de rétrodiffusion et d'annihilation de la rétrodiffusion

Au cours du processus de modélisation, il n'est pas rare que le concepteur constate que des concepts insérés dans un sous-modèle d'implémentation soient d'un niveau de conception ou d'analyse. Son transfert manuel dans le sous-modèle de conception ou d'analyse n'est souvent pas facile. Aussi, l'automatisation de ce transfert rendra le processus de développement beaucoup plus souple et facilitera le travail du concepteur.

Comme la réflexion autour de la diffusion n'a nécessité aucune hypothèse quant à la nature des sous-modèles source et destination, rien n'empêche d'appliquer le même principe de diffusion depuis les sous-modèles d'implémentation vers le sous-modèle d'analyse. C'est l'objet de la ***Transformation de rétrodiffusion*** $T_{Rét.\ Dif.}^{SM\ S \rightarrow SM\ D}$.

Dans sa conception, cette *Transformation de rétrodiffusion* est identique à celle de diffusion au détail près que la rétrodiffusion est annihilée par des annotations de rétrodiffusion (ex : *{MDA : Retrodiffusion Freezing}*) au lieu qu'elles soient de type *Diffusion* (ex : *{MDA : Diffusion Freezing}*).

Bien que très séduisante, la transformation $T_{Rét.\ Dif.}^{SM\ S \rightarrow SM\ D}$ est à manipuler avec précaution à cause de la nature ensembliste des sous-modèles (cf. Chapitre 3-I.2 et les definitions 7 à 12) qui est résumée par la relation :

$$SM_A \subset SM_{PD} \subset SM_{AD} \subset SM_I$$

De ce fait, un sous modèle d'implémentation va contenir un volume de concepts nettement supérieur à celui d'un sous-modèle d'analyse. Si le concepteur ne souhaite rétrodiffuser qu'un seul concept alors il devra annoter tous ceux qu'il ne souhaite pas rétrodiffuser. C'est un travail qui risque de devenir rapidement fastidieux.

Pour faciliter cette opération, une ***Transformation d'annihilation de la rétrodiffusion*** $T_{An.\ Rét.\ Dif.}^{SM \rightarrow SM}$ permet d'annoter tous les concepts exempts de lien de traçabilité avec le sous-modèle de la phase précédente annulant ainsi la rétrodiffusion. Le concepteur peut alors supprimer manuellement le concept ou les quelques concepts qu'il souhaite rétrodiffuser.

Comme pour la *Transformation de diffusion*, celle de rétrodiffusion est générique et s'applique à tous les sous-modèles de l'artefact *Software Development Process Model* sauf, bien entendu, sur le sous-modèle d'analyse. C'est aussi une transformation inter-sous-modèles (cf. figure 36).

La *Transformation d'annihilation de la rétrodiffusion* s'exerce sur les mêmes sous-modèles que la *Transformation de rétrodiffusion*. Par contre, c'est une transformation intra-sous-modèle (cf. figure 36).

II.3 Prétraitement et post-traitement associés aux transformations de diffusion : Principe

Le but de ce paragraphe est de signaler que, à cause du processus itératif de la méthode *Continuous Integration Unified Process*, les transformations de diffusion et de rétrodiffusion nécessitent parfois d'être complétées par un prétraitement et/ou un post-traitement qui s'applique sur le modèle source ou sur le modèle destination.

Il peut exister deux grands types de prétraitements et/ou de post-traitements. Le premier type inclut les prétraitements et les post-traitements qui vérifient la cohérence des sous-modèles source ou destination et signalent au concepteur la non conformité du sous-modèle. La diffusion des concepts est alors annulée. Le second type concerne les prétraitements et/ou les post-traitements qui corrigent des incohérences introduites à chaque cycle itératif par une séquence de transformations du type intra-sous-modèle / inter-sous-modèle.

Étant donné que les prétraitements et/ou post-traitements sont souvent liés à une transformation intra-sous-modèle, leurs descriptions ont été reportées après la transformation intra-sous-modèle dont ils corrigent les effets (cf. III.6.1.3).

En exemple, un post-traitement possible serait l'épuration systématique des spécifications d'implémentation du sous-modèle de conception avancée suite à une *Transformation de rétrodiffusion*. Celui-ci n'a pas été implémenté car la *Transformation d'annihilation de la rétrodiffusion* assure cette fonction.

Généralement, les prétraitements et les post-traitements sont toujours des transformations intra-sous-modèles qui ont la particularité d'être liées à un et un seul sous-modèle (cf. figure 36).

III TRANSFORMATIONS GEOMATIQUES

III.1 Généralités

Au cours de l'analyse, un concept c acquiert le statut d'entité référencée lorsque le concepteur lui affecte une propriété de spatialité ou de temporalité :

- En établissant par exemple une association avec un concept de spatialité ou de temporalité. Auparavant, le concepteur doit créer dans le modèle les concepts *Point*, *Ligne*, etc. et les relations entre ces concepts, il réalise à chaque développement un patron de concept SIG plus ou moins complet. Le problème est qu'il fait cet exercice en présence des acteurs du domaine étudié. Pendant qu'il effectue ce travail, les acteurs sont passifs et attendent qu'il ait fini pour poursuivre l'analyse. ***Cet exercice est non productif en phase d'analyse et source de perte de temps. De plus, si l'attente se prolonge, les acteurs présents pour faire un travail d'analyse risquent de se démobiliser et d'être moins attentif par la suite***.
- En apposant une annotation textuelle, pictogrammique, etc. sur le concept c et en différant son traitement à la phase de conception. L'analyse est alors effectuée à un rythme plus soutenu et avec moins de perte de temps.

Notre méthode Continuous Integration Unified Process s'inscrit dans ce second processus de développement.

En phase de conception, l'annotation textuelle ou pictogrammique peut être interprétée et traitée selon trois processus différents :

- Le premier consiste à finir manuellement le modèle d'une part, en créant un *Patron de conception SIG* et, d'autre part, en établissant les relations entre l'entité référencée et les concepts SIG. Le modèle obtenu peut ensuite être projeté en code dans le langage de programmation cible.
- Le deuxième s'appuie sur une ou plusieurs transformations qui automatisent les opérations manuelles de la méthode précédente. Elle(s) complète(nt) le modèle d'analyse en y insérant le *Patron de conception SIG*. Le modèle obtenu sera, dans une étape ultérieure, projeté en code.
- Le troisième est une variante du précédent car il court-circuite toute l'étape de génération d'un *Patron de conception SIG* et produit directement le code.

L'atelier de génie logiciel Perceptory développé par l'équipe d'Yvan Bédard est typiquement dans le troisième processus de développement. Les annotations pictogrammiques sont projetées en code SQL au cours d'une transformation unique.

Les transformations de Saisie de la Pictogrammie ($T_{Sai.Pict.}^{An\ SM \rightarrow An\ SM}$), de Suppression de la Pictogrammie ($T_{Rem.Pict.}^{An\ SM \rightarrow An\ SM}$), de Génération du Patron de conception SIG ($T_{Pat.SIG}^{PD\ SM \rightarrow PD\ SM}$) et de Traduction des Pictogrammes ($T_{Tra.Picto.}^{PD\ SM \rightarrow PD\ SM}$) que nous allons décrire aux paragraphes III.4, III.5 et III.6 inscrivent notre méthode dans le deuxième processus.

Ci-dessus, nous avons différé le traitement de l'annotation à la phase de conception car la réalisation manuelle d'un *Patron de conception SIG* est longue et non productive. Toutefois, la génération automatique du patron ouvre des perspectives nouvelles en termes de méthode de conduite de projet. En effet, les transformations peuvent être alors mobilisées en phase d'analyse puisque la génération est immédiate. C'est un avantage considérable propre aux processus deux et trois.

La création d'un *Patron de conception SIG*, au lieu de constituer un frein au travail d'analyse comme auparavant, peut devenir, suivant les centres d'intérêts et les compétences des acteurs, un objet d'échange entre le concepteur et ces derniers favorisant par la même occasion la communication entre eux.

La génération instantanée d'un *Patron de conception SIG* associé à la transformation de traduction des annotations en relations de généralisation ou d'association (cf. III.2) permet la mise en œuvre d'un processus de prototypage en phase d'analyse. Face au prototype de l'application, la réactivité et les critiques des acteurs sont bien plus grandes et plus pertinentes que face à un modèle dont ils ne maîtrisent pas souvent toutes les finesses du langage de modélisation. Cette réactivité plus grande améliore la communication et la capture des concepts du domaine et, par voie de conséquence, la qualité finale du modèle et de l'application.

Enfin, le deuxième processus dans lequel nous nous inscrivons offre une souplesse supplémentaire. Certains acteurs, des programmeurs en particulier, perçoivent mieux un modèle « complet » incluant tous les concepts qu'un modèle « partiel » chargé d'annotations mêmes si ces dernières sont de nature pictogrammique. Pour cette typologie d'acteurs, le processus de génération instantanée d'un patron de conception suivi de la conversion des annotations en relations améliore aussi la compréhension et la capture des concepts.

Les transformations de Génération du Patron de conception SIG et de Traduction des Pictogrammes ont été conçues dans cet esprit d'automatisation en vue d'une utilisation en phase d'analyse afin de pouvoir mettre en œuvre la méthode Continuous Integration Unified Process.

Automatiser l'interprétation des annotations facilite la modélisation, mais au préalable il faut saisir, supprimer ou modifier ces notations, c'est l'objet des Transformations de saisie et de suppression de la pictogrammie.

III.2 Concepts UML support de l'application d'affectation <<

Nous avons vu aussi au paragraphe du V du Chapitre 1 qu'il existe deux écoles parmi les formalismes de modélisation des entités référencées :

- ⇨ Une première école utilise implicitement le mécanisme de ***réutilisation en boîte noire***[52] défini (Gamma *et al.*, 1999). C'est *un style de réutilisation fondé sur la composition d'objets*.
- ⇨ La seconde école met en œuvre explicitement le mécanisme de ***réutilisation en boîte blanche***[53], *style de réutilisation fondé sur l'héritage de classe*.

La première comprend la plupart des méthodes ou formalismes sur lesquels des recherches ont été effectuées ces dernières années : CONGOO, MADS, Perceptory, STER, etc.

Toutes ces méthodes ou formalismes créent un attribut géographique ou temporel au sein de l'entité référencée ce qui équivaut à faire une association entre l'entité référencée et le ou les concepts SIG de la pictogrammie de l'entité (cf. Dualité Attribut/Association en Annexe IV). L'aspect implicite vient du fait que l'attribut n'est pas toujours créé au sein de l'entité référencée mais, souvent, il est directement généré dans le code.

Les formalismes et méthodes relevant de la seconde école sont plus rares. La méthode POLLEN et le formalisme ULM-GeoFrame semblent être les seuls utilisant le mécanisme de réutilisation fondé sur l'héritage de classe.

Les deux mécanismes présentant des avantages et des inconvénients, nous avons décidé de laisser le concepteur libre de choisir le mécanisme qui lui convient le mieux par rapport au problème qu'il a à traiter, aux exigences d'implémentation, ou tout autre critère. Il aura à effectuer son choix en début de modélisation via l'interface de configuration de l'atelier de génie logiciel (cf. figure 75).

En langage UML, le concept instanciant le mécanisme de réutilisation en boîte noire est le concept d'*Association* et celui instanciant le mécanisme de réutilisation en boîte blanche est le concept de

[52] Traduction (Gamma *et al.*, 2001) : A style of reuse based on object composition. Composed objects reveal no internal details to each other and are thus analogous to "black boxes".

[53] Traduction (Gamma *et al.*, 2001) : A style of reuse based on class inheritance. A subclass reuses the interface and implementation of its parent class, but it may have access to otherwise private aspects of its parents.

Généralisation/Spécialisation. Ces deux types instanciations de l'application d'affectation << structurent l'ensemble du chapitre.

III.3 Dérivation d'un Patron de conception SIG du Métamodèle SIG

L'objectif premier de la recherche est d'assister les concepteurs dans le développement d'un *Système d'Information Géographique*. Pour atteindre cet objectif, nous avons fait le choix de générer automatiquement le *Patron de conception SIG* (cf. III.1) qui incluent les concepts SIG nécessaires au développement d'une application et les relations qu'ils ont entre eux.

Or, tous les concepts SIG et leurs relations sont présents dans le Métamodèle SIG. Fort de ce constat, le choix a été de dériver du Métamodèle SIG d'un Patron de conception SIG au sens de la définition 81.

Les deux paragraphes suivants traitent d'une part, de cette dérivation et, d'autre part, comment la *Transformation de traduction des pictogrammes* vient en complément à cette dérivation.

III.3.1 *Suppression des spatialités et temporalités de nature informative*

Le *Métamodèle SIG* de la figure 110 est le métamodèle complet dédié au langage UML puisque les entités référencées sont représentées par le concept UML de *Classe*. Ce métamodèle contient un certain nombre de concepts de nature informative dont la fonction première est d'annoter les entités référencées temporairement et ensuite ils sont le plus souvent remplacés. Il s'agit des concepts :

- ⇨ De spatialité dérivée et de temporalité dérivée qui préfigurent la création d'une opération (cf. Chapitre 7-VIII.1).
- ⇨ De spatialité inconnue, compliquée et multiple et leurs homologues temporels qui confèrent le statut d'entité référencée au concept étudié. Le contexte d'utilisation de ces six concepts est décrit au Chapitre 7.

Les seize concepts concernés sont indispensables pour développer un atelier de génie logiciel implémentant le langage pictogrammique de Perceptory mais ils n'ont pas lieu d'être dans un patron de conception. Ils ont donc été supprimés (cf. figure 46).

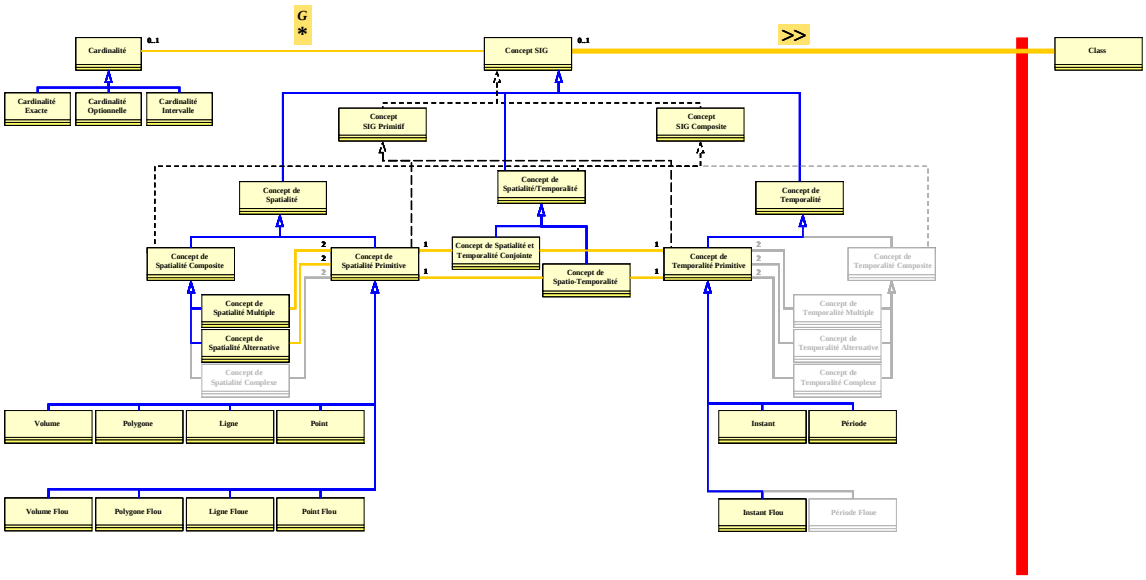


Figure 46 *Métamodèle SIG* - Première étape de simplification.

III.3.2 *Suppression des spatialités composites et des temporalités composites*

Au paragraphe I du Chapitre 4, nous avons évoqué les raisons qui ont conduit à adopter le concept UML de *Classe* pour représenter les entités référencées et le concept de Stéréotype pour exprimer la spatialité et la temporalité. Virtuellement, la taxinomie des entités référencées est réduite au seul concept *Classe*.

La transformation de Génération du Patron de Conceptions SIG ayant été exécutée, les concepts de spatialité (*Point*, *Ligne*, etc.) et temporalité (*Instant*, *Période*, etc.) sont présents dans le modèle du système d'information à modéliser. À ce stade, les entités référencées munies de leur pictogrammie n'ont aucune relation avec les concepts de spatialité primitive et temporalité primitive.

Le but de la Transformation de traduction des pictogrammes est d'établir les relations entre une entité référencée et le ou les concepts correspondant à sa pictogrammie. Cela revient à « matérialiser » l'application d'affectation <<.

À nouveau se pose le problème d'établir cette famille de relations au plus haut niveau de généralisation ou au plus bas niveau de spécialisation (cf. Chapitre 7-VII). Comme précédemment, établir une relation unique au plus haut niveau de généralisation pose les mêmes problèmes d'incohérence syntaxique. Aussi, les relations matérialisant l'application d'affectation **<<** sont établies au plus bas niveau de spécialisation par la *Transformation de traduction des pictogrammes*.

Or, il est possible de matérialiser l'application d'affectation **<<** suivant deux approches différentes. La première consiste à établir une *relation unique entre une entité référencée et le concept SIG associé*. Dans cette perspective, les concepts SIG composite devraient exister dans le *Patron de conception SIG*. Le nombre de concepts du *Patron de conception SIG* serait important et certains d'entre eux seraient rarement utilisés. Dans la seconde approche, la *Transformation de traduction des pictogrammes* étant entièrement automatisée, *plusieurs relations peuvent être établies entre l'entité référencée et les concepts SIG primitifs impliqués dans la pictogrammique*. L'avantage de cette seconde approche est que les lois de compositions sont traitées par ce biais. Le métamodèle de la figure 46 peut être allégé de tous les concepts composites :

- ⇨ Les concepts de spatialité multiple, de spatialité alternative, de spatialité complexe, de spatialité composite et leurs homologues temporels.
- ⇨ Ceux de spatialité et temporalité conjointe, de spatio-temporalité, de spatialité/temporalité et SIG composites.

Bien que plus délicate, la suppression du concept de spatio-temporalité du métamodèle de la figure 46 est possible car c'est un concept à double statut concept SIG/entité référencée. En effet, les concepts de spatio-temporalité appartiennent à l'intersection des deux ensembles ***U*** et ***E*** (cf. Chapitre 7-IV.2.2.2) puisque la temporalité ne renseigne pas le concept thématique de l'entité mais la spatialité. C'est cette propriété mise en évidence lors de la construction des taxinomies terminologiques du Chapitre 7 qui est exploitée au sein de la *Transformation de traduction des pictogrammes*. Cette dernière établira une première relation entre l'entité référencée et le concept de spatialité et une seconde entre les concepts de spatialité et de temporalité.

Les concepts de *Spatialité*, de *Temporalité* et *SIG Composite* n'ayant plus une grande utilité, ils sont aussi supprimés. La figure 47 est le résultat des différentes simplifications effectuées. Elle montre aussi trois exemples d'entités référencées et les relations **<<** qu'elles ont avec les concepts SIG de sa pictogrammie.

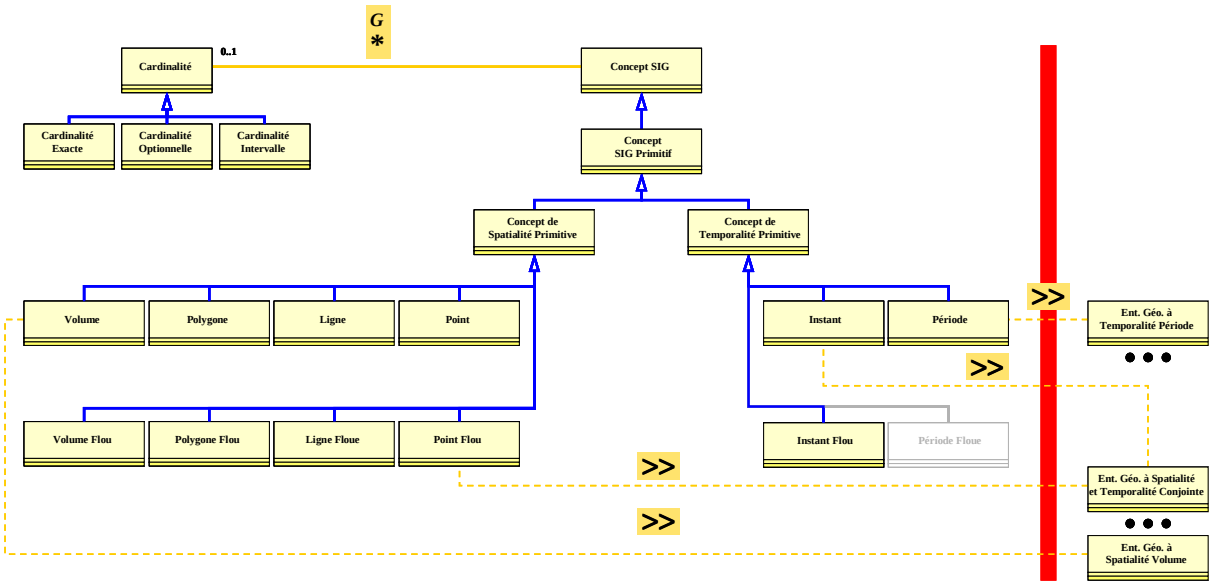


Figure 47 *Métamodèle SIG* - Seconde étape de simplification.

III.3.3 *Traitement de la cardinalité*

L'instanciation de l'application d'affectation << en concept UML de *Généralisation* implique que les concepts de plus bas niveau de spécialisation héritent des propriétés et des comportements de tous les concepts au-dessus de lui. Cela se traduit par le fait qu'une entité référencée va hériter du ou des concepts SIG de sa pictogrammie qui eux-mêmes vont hériter de *Concept SIG*. Aussi quelle que soit l'entité référencée, elle héritera de la loi de composition $*^{G}$ entre *Concept SIG* et *Cardinalité*.

Étant donné que c'est une loi de composition entre concepts SIG et non pas entre entités référencées, cet héritage est conceptuellement incohérent. C'est une première raison qui conduirait à supprimer la loi de composition $*^{G}$ et les concepts de cardinalités.

Dans une instanciation de l'application d'affectation << en concept UML de d'*Association*, ce problème d'héritage ne se pose plus mais la cardinalité des concepts SIG peut être prise en charge par la cardinalité de l'association établie entre l'entité référencée et les concepts SIG primitifs. De ce fait, la loi de composition $*^{G}$ et les concepts de cardinalités ne sont plus nécessaires.

Ce sont les deux raisons qui ont conduit à simplifier le métamodèle de la figure 47 en supprimant la loi de composition $*^{G}$ et les concepts de cardinalités. *Concept SIG* n'étant plus indispensable, il a été lui aussi supprimé.

La partie à gauche de la barre verticale de la figure 48 constitue le cœur du *Patron de conception SIG* qui sera généré par la transformation $T_{Pat.\ SIG}^{PD\ SM \rightarrow PD\ SM}$.

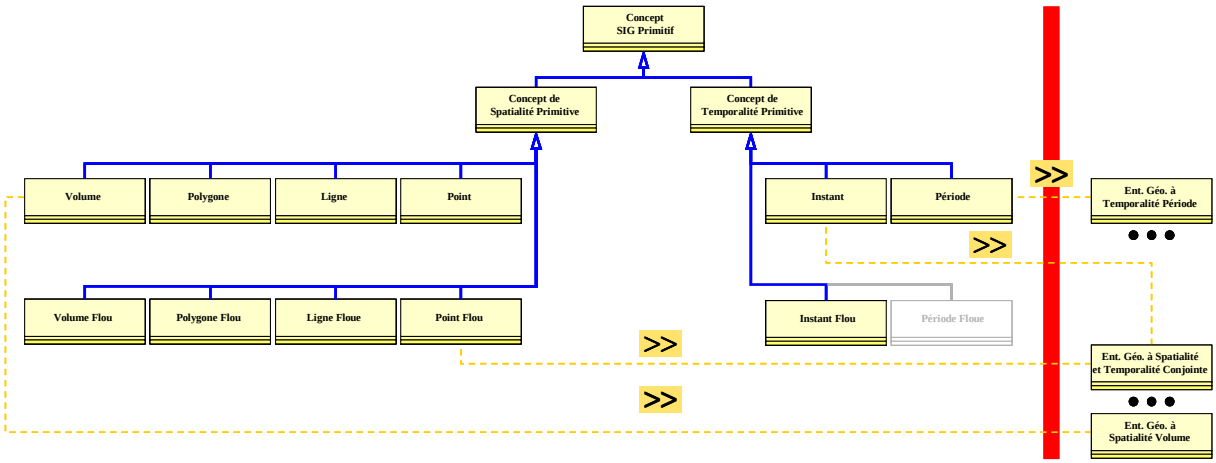


Figure 48 Cœur du *Patron de conception SIG*.

III.3.4 *En résumé*

Finalement, la « matérialisation » de l'application d'affectation entre l'ensemble des entités référencées et celui des concepts SIG soit par une relation de Généralisation/Spécialité soit par une relation d'Association a permis de simplifier de façon drastique le *Métamodèle SIG* de la figure 110 pour extraire le cœur du *Patron de conception SIG* qui sera généré automatiquement par la transformation $T_{Pat.\ SIG}^{PD\ SM \rightarrow PD\ SM}$. Cette extraction a été effectuée en conservant la cohérence avec le *Métamodèle SIG* dérivé des taxinomies terminologiques du Chapitre 7.

Le cœur du *Patron de conception SIG*, épuré des concepts superflus, est constitué des concepts SIG primitifs. Cependant, toute la connaissance du domaine de l'*Information Géographique* n'étant pas exprimée dans ce cœur de patron, il va être complété au paragraphe III.5.3.1 par les relations entre concepts flous et non flous et au paragraphe III.5.3.2 par les relations de nature géométrique entre concepts SIG primitifs.

III.4 Conception des Transformations de saisie et de suppression de la pictogrammie

III.4.1 *Transformation de saisie de la pictogrammie*

Au Chapitre 4, les raisons qui ont amené à limiter la capacité d'expression des langages SIG et pictogrammique du *Profil UML-SIG* ont été exposées. Les spatialités et temporalités actuellement disponibles sont :

- La spatialité primitive et la temporalité primitive.
- La spatialité et temporalité conjointe.
- La spatio-temporalité.
- La spatialité multiple.
- La spatialité alternative.

Lors de la saisie du couple stéréotype/pictogramme, le concepteur doit aussi saisir des informations complémentaires :

- Quel que soit le type d'instanciation de l'application d'affectation <<, il faut indiquer la ou les dimensions du système de coordonnées utilisés pour décrire le domaine modélisé.
- En instanciation d'association, il faut ajouter en outre les rôles, les cardinalités de l'association et, si nécessaire, les contraintes.

Ces informations complémentaires sont toutes introduites via de concept UML de valeur marquée (cf. Annexe I-III.7.2). Les deux paragraphes suivants détaillent les informations saisies simultanément avec la pictogrammie des entités référencées.

III.4.1.1 Instanciation en généralisation

Dans ce type d'instanciations, le couple stéréotype/pictogramme représentant les concepts de temporalité primitive floue ou non floue contient l'information nécessaire et suffisante à la *Transformation de traduction des pictogrammes* (cf. figure 49-Modèle B).

En *Information Géographique*, tout concept de spatialité existe dans un espace donné et est repéré dans un système de coordonnées défini dans cet espace. L'espace le plus communément utilisé est celui qui nous entoure. Il est de dimension 3. Toutefois, une entité référencée peut être repérée dans un sous-espace de l'espace principal, tout dépend de la pratique et/ou des spécifications des acteurs.

Par exemple, si le commanditaire du *Système d'Information Géographique* s'intéresse au suivi temps réel ou semi-temps réel d'un véhicule transportant des marchandises, il a plusieurs options possibles. La première, la plus ancienne, consiste à demander au chauffeur de téléphoner sa position à fréquence régulière. Celui-ci donnera de fait le kilométrage effectué depuis son départ. Implicitement, le chauffeur et le responsable du suivi des marchandises ont adopté le système de coordonnées curviligne matérialisé par la route dont la dimension est 1. Pourtant, la route serpente dans un espace assimilable à un espace 2D si le trajet s'effectue en plaine et, si elle est sinueuse et très pentue, la dimension de l'espace à prendre en compte est 3. Avec l'avènement des technologies GPS, le suivi d'un véhicule de transport de marchandises est devenu plus facile puisque d'une part, le système de positionnement émet régulièrement les coordonnées du véhicule en XY ou en XYZ et, d'autre part, décharge le conducteur d'appeler régulièrement son entreprise..

Cet exemple de gestion de deux systèmes de coordonnées n'est pas unique. Nous avons rencontré un cas similaire dans le projet SIRS Digues dont le but est la gestion de patrimoine des digues du Rhône. Les commanditaires en possession de données remontant à plus d'un siècle souhaitaient les introduire dans le nouveau système d'information sous une forme la plus proche possible des documents existants. Historiquement, la localisation sur une digue est repérée par rapport au cours d'eau qui a un système de coordonnées linéaire (1D) reposant sur le tracé de son axe. Le système linéaire est toujours majoritairement utilisé par le personnel de surveillance des digues car les organismes responsables de l'entretien n'ont pas les moyens d'investir dans la technologie GPS.

Toutefois, l'évolution à la baisse du coût des GPS conduira inévitablement à son adoption dans un futur proche. Le personnel de surveillance repéreront alors les anomalies dans un système de coordonnées 2D voire 3D. D'ailleurs certains syndicats ont déjà adopté cette technique et leurs personnels sont en cours de formation.

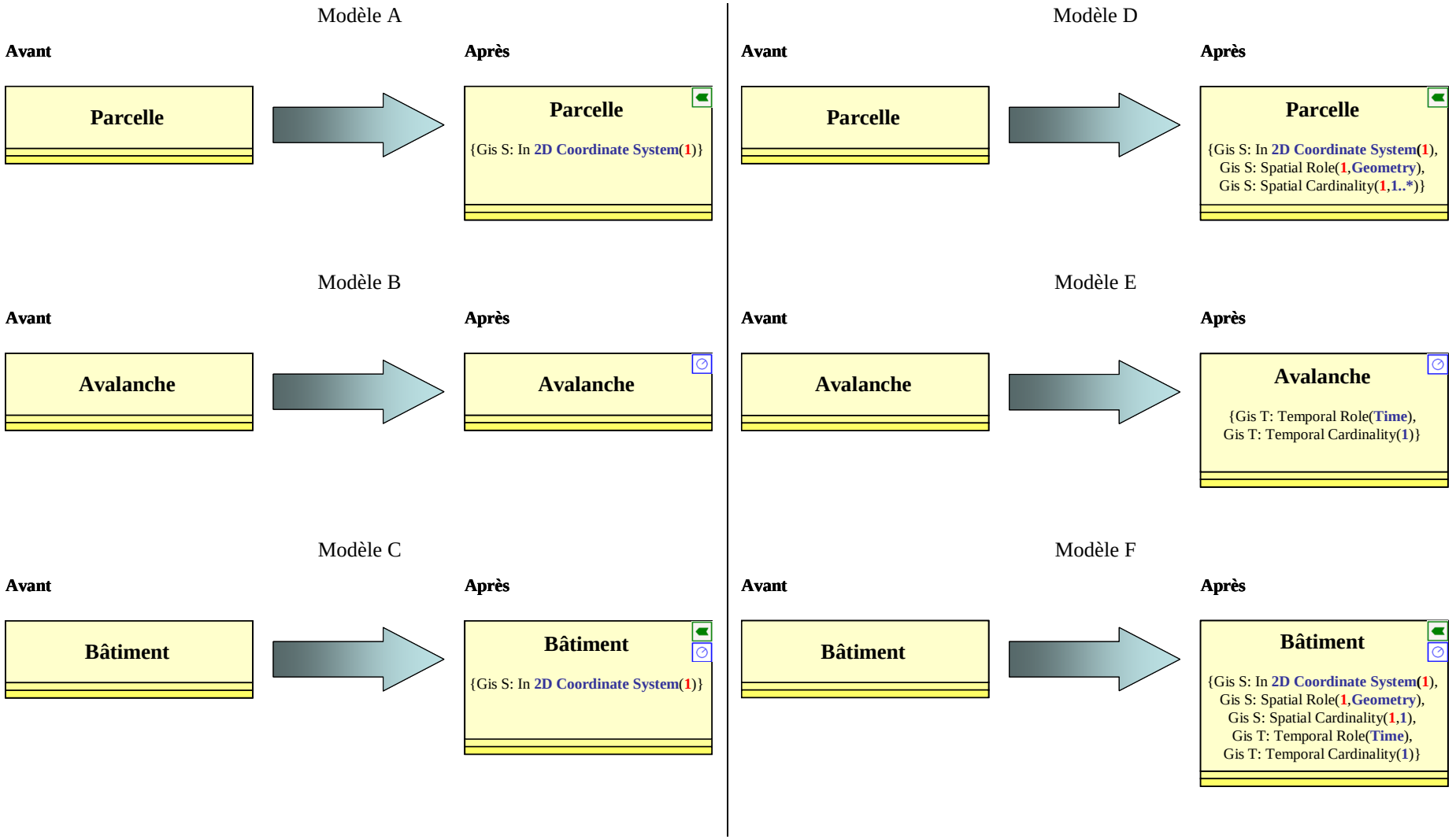


Figure 49 Informations saisies lors de l'annotation des entités référencées suivant le mécanisme de boîte blanche à gauche et le mécanisme de boîte noire à droite.

Dans la phase transitoire, les deux systèmes de localisation cohabitent car la mise en place du dispositif se fait progressivement.

Conscient de cette problématique, nous avons opté pour ajouter systématiquement, lors de la saisie du couple stéréotype/pictogramme, une ou plusieurs valeurs marquées indiquant la ou les dimensions du système de coordonnées satisfaisant la pratique et/ou les besoins de domaine modélisé. L'information relative à la dimension du système de coordonnées est contenue dans le nom des valeurs marquées : *{Gis S: In 1D Coordinate System(1)}*, *{Gis S: In 2D Coordinate System(1)}* et/ou *{Gis S: In 3D Coordinate System(1)}* (cf. figure 49-Modèle A). Les valeurs marquées de dimension ont été dotées d'un index (valeur entre parenthèse) afin de pouvoir gérer les concepts de spatialité multiple et de spatialité alternative. En effet, ceux-ci étant composés de deux spatialités primitives, il faut ajouter deux jeux de valeurs marquées de dimension. Dans ce contexte, l'index prend la valeur 1 ou 2 suivant la spatialité primitive auquel il fait référence sinon il a pour valeur 1.

Pour les concepts de spatialité composite, les valeurs marquées introduites résultent d'une combinaison de deux cas primitif décrits ci-dessus. Le modèle C de la figure 49 montre l'exemple de l'entité référencée *Bâtiment* annotée de la spatialité et temporalité conjointe.

III.4.1.2 Instanciation en association

Pour l'instanciation de l'application d'affectation **<<** en association, nous avons fait le choix d'ajouter deux informations supplémentaires par concept SIG primitif. Ces deux informations deviendront le rôle et la cardinalité des associations au cours de la *Transformation de traduction des pictogrammes*. Elles permettent de compléter le modèle résultant de la transformation. Enfin, pour les concepts de spatialité alternative ou de temporalité alternative, le critère seuil de changement de spatialité ou de temporalité (cf. Chapitre 7-IV.2.1.2) est une information à saisir et à stocker. Étant donné que le critère seuil est une contrainte sur la typologie de la spatialité, nous avons fait le choix du concept UML *Constraint* pour stocker cette information. De ce choix découle le nom de la valeur marquée. Nous approfondirons ces points au paragraphe III.6.1.1.b.

Pour saisir ces informations supplémentaires, les interfaces de l'atelier de génie logiciel (cf. figure 73-Colonne de droite) proposent au concepteur de deux à trois champs supplémentaires selon le type de spatialité ou de temporalité à saisir. Le contenu de ces champs est ensuite stocké dans les valeurs marquées de rôle, de cardinalité ou de contrainte propres au concept primitif. Les valeurs marquées de rôle ont pour nom *{Gis S: Spatial Role(1,Geometry)}* ou *{Gis T: Temporal Role(Time)}* selon qu'elles renseignent une entité référencée à spatialité ou à temporalité, celles relative à la cardinalité ont été nommées *{Gis S: Spatial Cardinality(1,1..*)}* ou *{Gis T: Temporal Cardinality(1)}* et celles de contrainte sont désignées *{Gis S: Spatial Constraint(1, S<500)}* ou *{Gis T: Temporal Constraint(1,T>3)}*.

Les valeurs marquées de temporalité n'ont qu'un seul paramètre qui contient, suivant la typologie de la valeur marquée, l'information sur le rôle, la cardinalité ou la contrainte à stocker. Ces même informations sont gérées et stockées dans le second paramètre des valeurs marquées de spatialité. Ces dernières ont un premier paramètre qui est un index permettant la gestion de deux jeux de valeurs marquées. L'index intervient dans la gestion de deux concepts de spatialités des entités référencées à spatialité multiple ou à spatialité alternative

Comme précédemment, les entités référencées composites sont annotées par une combinaison des valeurs marquées correspondant aux concepts SIG primitifs de la pictogrammie (cf. figure 49-modèle F).

III.4.2 *Transformation de suppression de la pictogrammie*

Toute activité de modélisation suppose l'ajout de spécifications mais aussi de modification ou de suppression. Nous avons vu précédemment qu'un certain nombre d'annotations complémentaires sont associées au couple stéréotype/pictogramme. Supprimer ce couple suppose aussi de supprimer toutes ces annotations supplémentaires. Sans cela, le modèle serait rapidement chargé d'annotations inutiles. Sa qualité serait dégradée.

La *Transformation de suppression de la pictogrammie* $T_{Rem.\ Pict.}^{An\ SM \rightarrow An\ SM}$ a été conçue dans ce but. Lorsqu'un couple stéréotype/pictogramme est supprimé, l'entité référencée est épurée de toute valeur marquée de dimension, de rôle, de cardinalité et, s'il y a au lieu, de contrainte.

III.5 Conception de la Transformation de génération du patron de conception SIG

A la différence des patrons de conception d'Erich Gamma (Gamma *et al.*, 1999) qui sont souvent importés par une action copier/coller, le *Patron de conception SIG* est automatiquement incorporé par la *Transformation de génération du patron de conception SIG* $T_{Pat.\ SIG}^{PD\ SM \rightarrow PD\ SM}$ dans le sous-modèle de conception préliminaire. Il est

ensuite diffusé automatiquement vers un ou plusieurs sous-modèles d'implémentation lesquels sont ensuite projetés en code à l'aide des générateurs de code des ateliers de génie logiciel.

III.5.1 *Architecture générale du Patron de conception SIG*

Le cœur du *Patron de conception SIG* (cf. figure 48) ne prend pas en compte la problématique de la dimension du système de coordonnées évoquée au paragraphe III.4. Ce dernier introduit une contrainte supplémentaire sur la partie spatiale. Autant le concept *Point* existe dans un système de coordonnées 1D, 2D ou 3D autant un concept volumique (*Volume* par exemple[54]) n'existe que dans un système 3D. Ce constat, nous a conduit à générer trois patrons de conception spatiaux qui diffèrent par :

⇨ Les concepts de spatialité primitive de chacun des systèmes de coordonnées :

- En 1D : *Point*/*Ligne* et leur homologue flou *Point Flou*/*Ligne Floue*.
- En 2D : *Point*/*Ligne*/*Polygone* et *Point Flou*/*Ligne Floue*/*Polygone Flou*.
- En 3D : *Point*/*Ligne*/*Polygone*/*Volume* et *Point Flou*/*Ligne Floue*/*Polygone Flou*/*Volume Flou*.

⇨ Le nombre de coordonnées du concept *Point*, X, XY ou XYZ, suivant la dimension du système de coordonnées.

De ce fait, le concepteur a à sa disposition trois patrons de conception spatiaux, un par système de coordonnées, et un patron temporel (cf. figure 50).

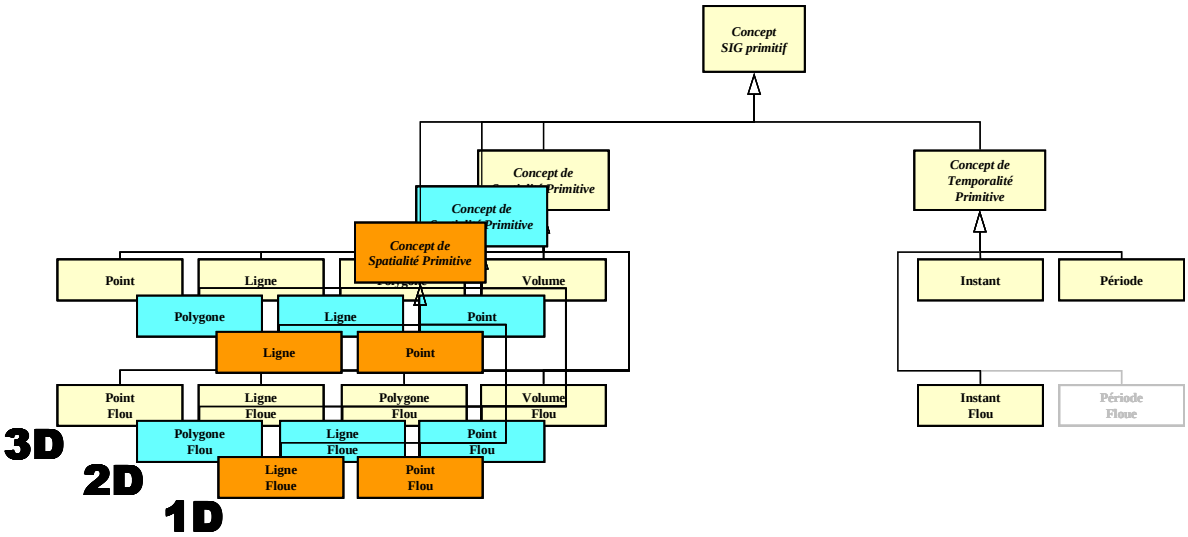


Figure 50 Modèle global du *Patron de conception SIG* simplifié incorporant la problématique de la dimension du système de coordonnées.

Nous avons fait le choix de générer trois patrons de conception spatiaux car le concept de *Point* n'est pas le seul à être vraiment différent. Implicitement, la dimension du système de coordonnées intervient pour les autres concepts lors du calcul de la longueur d'une ligne, du périmètre d'un polygone, etc. Le code de ces fonctionnalités dans les sous-modèles d'implémentation est vraiment différent.

Nous verrons au paragraphe III.2 du Chapitre 6 que ces patrons sont structurés dans trois paquetages en application du principe du développement centré sur l'architecture de la méthode *Continuous Integration Unified Process*.

[54] Nous rappelons que le concept de *Volume* a été juste introduit pour montrer la problématique du 3D et préparer l'évolution du langage SIG du Profil UML-SIG.

III.5.2 *Modèles individuels des concepts SIG primitifs : valeur marquée de primitivité*

Pour renforcer le caractère primitif des concepts SIG du Patron du Conception SIG, les concepts de spatialité et de temporalité flous et non flous sont systématiquement annotés avec une valeur marquée *{Gis : Primitive(val)}*. Le paramètre *val* indique la nature spatiale ou temporelle du concept. Il prend la valeur *Spatial* si la valeur marquée est apposée sur un concept de spatialité et *Temporal* si elle annote un concept de temporalité. La figure 51 montre quelques exemples.

Figure 51 Exemples de valeur marquée de primitivité.

En plus de renforcer le statut primitif, l'adjonction des valeurs marquées *{Gis : Primitive(val)}* présente deux avantages :

- ⇨ Le premier est de permettre une identification plus facile des concepts primitifs de l'ensemble des concepts du *Patron de conception SIG* puisque le concepteur est en droit d'ajouter d'autres concepts s'il en a besoin.
- ⇨ Le second est de différencier le cas échéant le concept SIG primitif généré par la transformation $T_{Pat.\,SIG}^{PD\,SM \rightarrow PD\,SM}$ de ceux réalisés manuellement par le concepteur.

Cette information supplémentaire permet de faire un recoupement d'information et ainsi identifier sans ambiguïté les concepts SIG primitifs du *Patron de conception SIG*. Par exemple, la *Transformation de traduction des pictogrammes* ne s'effectue que si les concepts SIG primitifs de la pictogrammie ont une valeur marquée de primitivité (cf. III.6.1.1). La qualité du modèle en est améliorée.

Il faut tenir compte du fait que le concepteur a toujours la possibilité d'intervenir sur le modèle et peut involontairement le détériorer.

L'ensemble des concepts SIG du *Patron de conception SIG* sont des concepts parfaitement connus depuis fort longtemps et enseignés dès notre enfance. Aussi, tout un chacun est en mesure de faire un modèle des concepts SIG semblable à celui de son voisin. Bien évidemment, ils sont en accord aux modèles normalisés de l'Open Geospatial Consortium Inc. ou de l'ISO (ISO, 2003a ; OGC, 2005).

Les figures 52 et 53 donnent les modèles individuels des concepts SIG non flous retenus qui sont générés par la *Transformation de génération du patron de conception SIG*. Nous verrons le cas des concepts SIG flous au paragraphe III.5.3.1.

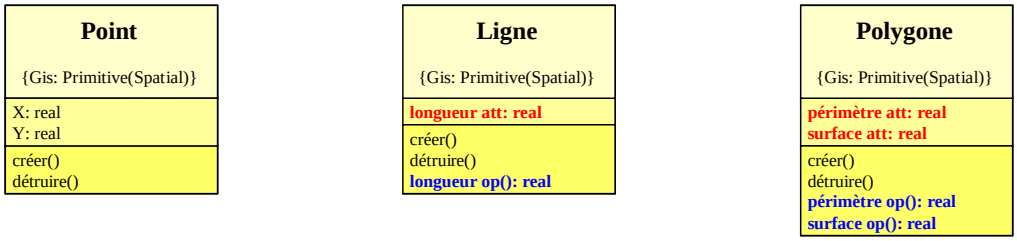


Figure 52 Modèles individuels des concepts SIG primitif de spatialité[55].

Dans un système de coordonnées 2D, le concept de spatialité *Point* possède une abscisse et une ordonnée, propriétés qui sont matérialisées par les attributs X et Y.

La *longueur* est une caractéristique intrinsèque du concept *Ligne* dont le concept UML support est généralement soit le concept d'*Attribut* soit celui d'*Opération*. La première représentation est généralement

[55] Les extensions « att » ou « op » au nom de l'attribut, respectivement de l'opération, évitent les messages d'erreurs dans certains ateliers de génie logiciel. Lorsque le concepteur aura opté pour un type de représentation, il supprimera les extensions.

utilisée lorsque les données proviennent d'une mesure directe de la valeur, la seconde est appliquée quand cette caractéristique résulte d'un calcul. ***Nous avons fait le choix de générer les deux représentations car le concepteur peut ainsi gérer les deux types de données.*** Il peut aussi supprimer la représentation qui ne lui convient pas. Bien que déconseillée, une dernière solution consisterait à utiliser l'opération comme accesseur à l'attribut.

Les deux caractéristiques intrinsèques, *périmètre* et *surface*, du concept *Polygone* sont traitées dans le même esprit. Il en est de même de la propriété *durée* du concept *Période* (cf. figure 53).

Figure 53 Modèles individuels des concepts SIG primitif de temporalité.

Bien entendu, seule la signature des fonctionnalités est créée dans le sous-modèle de conception préliminaire. Le corps proprement dit sera complété dans les sous-modèles d'implémentation en fonction du langage de programmation cible.

Enfin, les concepts SIG ont été dotés systématiquement des constructeurs (*créer()*) et destructeurs (*détruire()*) qui permettent d'instancier et de supprimer des objets. La raison est que souvent ces fonctionnalités participent à la modélisation des spécifications dynamiques décrivant les comportements des objets dans les diagrammes de séquence, de collaboration, etc.

III.5.3 *Patron de conception SIG généré*

III.5.3.1 Intégration des relations entre concept flou / concept non flou

Au paragraphe IV du Chapitre 8, nous avons établi un patron de conception entre un concept primitif flou et son homologue non flou (cf. figure 112). L'avantage d'un patron de conception est qu'il peut être généré automatiquement. La transformation $T^{PD\ SM \rightarrow PD\ SM}_{Pat.\ SIG}$ ajoute les associations d'*Incertitude Floue* entre deux concepts homologues. Elles sont établies *au plus bas niveau de spécialisation* afin d'éviter des incohérences syntaxiques similaires à celles de la matérialisation de l'application d'affectation <<. Le nombre élevé de relations à établir ne pose pas de problème puisqu'elles sont créées automatiquement. Le résultat est le modèle de la figure 54 qui, par souci de simplification, ne fait apparaître ni les modèles individuels des concepts SIG ni les patrons de conception spatiaux 1D et 2D.

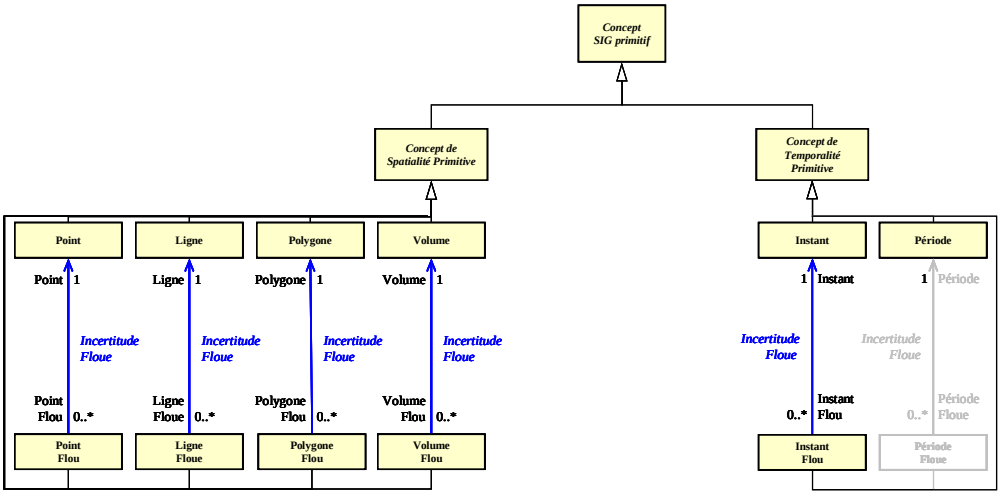


Figure 54 Associations d'*Incertitude Floue* du *Patron de conception SIG*.

III.5.3.2 Intégration des relations de nature élémentaire

A ce stade, la seule connaissance mobilisée est celle de la construction de la taxinomie du langage SIG à laquelle il faut ajouter la contrainte introduite par la dimension du système de coordonnées et celle liée aux concepts flous.

Les relations, que l'on qualifiera *d'élémentaire*, entre concepts de spatialité primitive sont les bases de la géométrie euclidienne. Celles entre les concepts *Instant* et *Période* sont aussi parfaitement connues. Les schémas spatial et temporel de l'ISO (ISO, 2003a) et l'OGC (OGC, 2005) intègrent ces relations élémentaires. La mobilisation de ces connaissances affine le *Patron de conception SIG*. Il est à noter que ces relations se situent au plus bas niveau de spécialisation du *Patron de conception SIG*.

Il nous a été enseigné que le concept de *Ligne* est constitué soit d'une série de points ordonnés soit de tronçons de lignes mis bout à bout. Le patron de conception composite identifié par Erich Gamma (Gamma *et al.*, 1999) couvre ces deux descriptions simultanément. Le patron de conception spatial 1D (cf. figure 55) résulte de cette analyse qui a également été appliquée aux concepts flous.

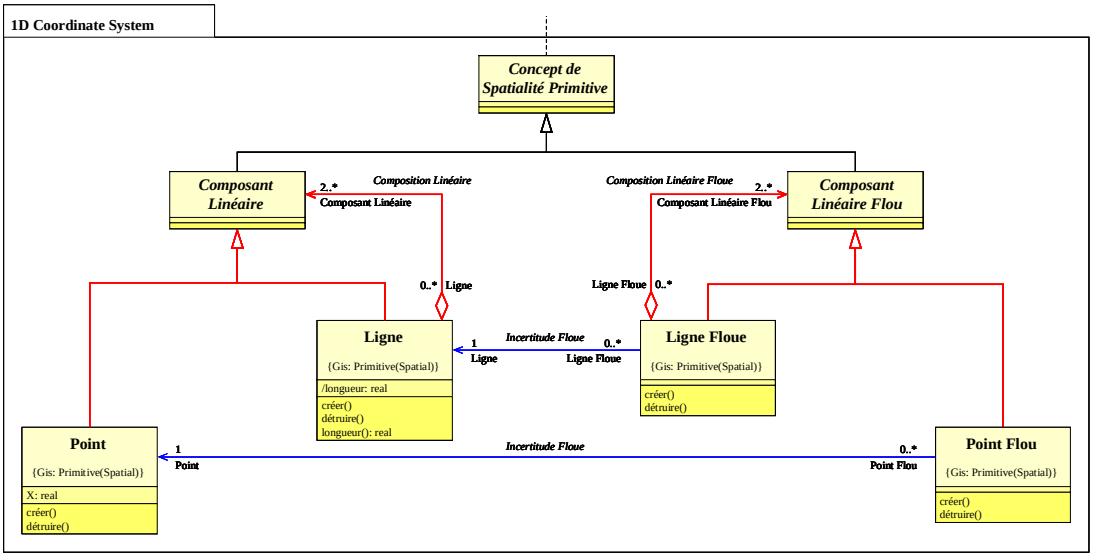


Figure 55 Patron de conception spatial 1D.

Le concept de *Polygone* peut aussi bien être décrit comme étant une succession de points ordonnés ou comme une succession de lignes elles aussi ordonnées. *Point* et *Ligne* sont donc des composants élémentaires d'un polygone. Par ailleurs, souvent lorsqu'un acteur s'intéresse à un ensemble de polygone, s'ils constituent une entité unique, l'acteur s'intéresse aussi à cette entité unique. Cela signifie que le polygone représentant l'entité doit être décomposable en polygones. Comme précédemment, le patron de conception composite permet de couvrir tous ces cas. Le patron de conception spatial 2D incorpore cette description pour les concepts flous et non flous (cf. figure 56).

Comme les deux précédents concepts, celui de *Volume* peut aussi être modélisé autour d'un patron de conception composite (cf. figure 57). Bien que définir un volume par une liste de points ou de lignes[56] soit plus délicat, le patron de conception spatial 3D permet de le faire[57]. Il permet aussi de décrire un volume par les surfaces le délimitant ou par des volumes.

Pour finir, le patron de conception temporel est basé sur la description du concept de *Période* qui débute à un *Instant* donné et qui se termine à un *Instant* final (cf. figure 58).

[56] Ce type de représentation est utilisé dans certains logiciels de Conception Assistée par Ordinateur qui proposent un mode filaire pour décrire les objets volumiques.

[57] Les difficultés de cette représentation seront abordées lors de l'extension des langages SIG et pictogrammique aux concepts volumiques. Il sera sûrement nécessaire d'introduire des contraintes.

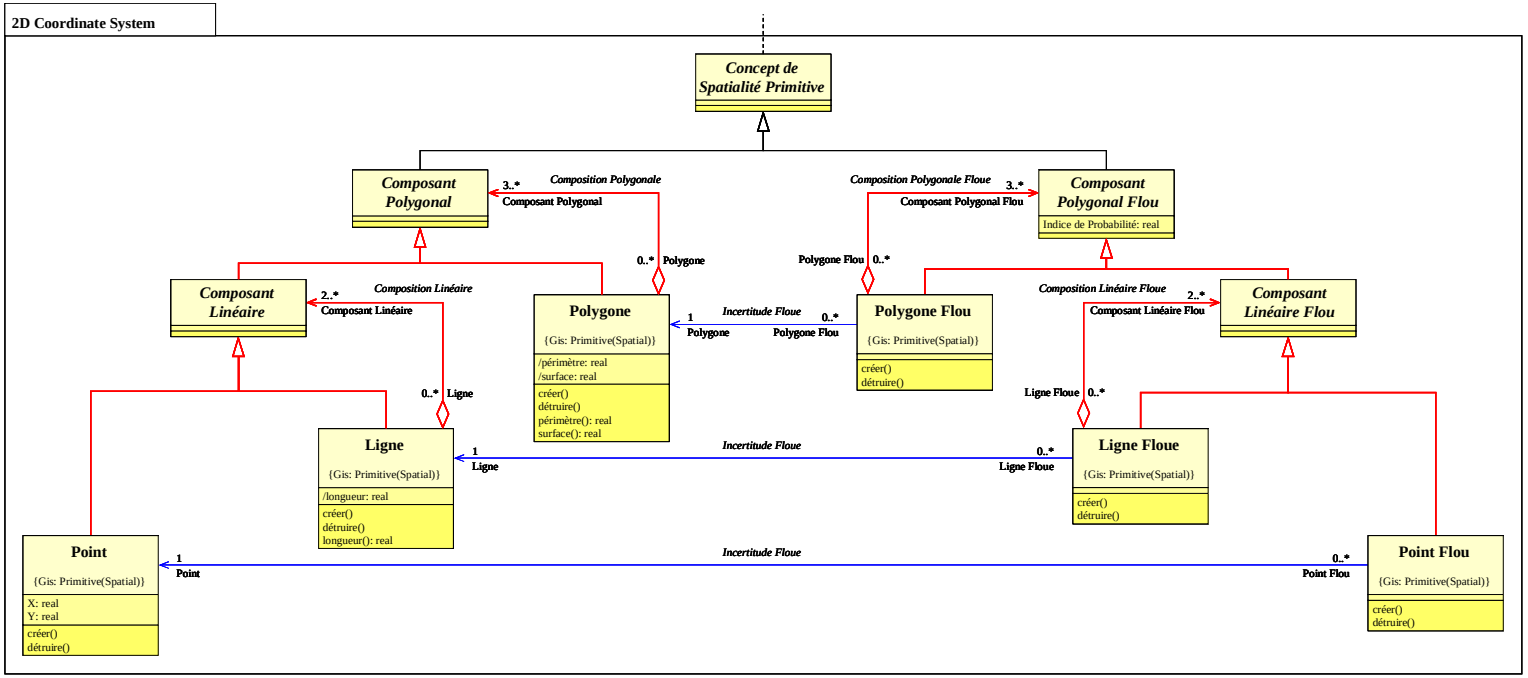


Figure 56 Patron de conception spatial 2D.

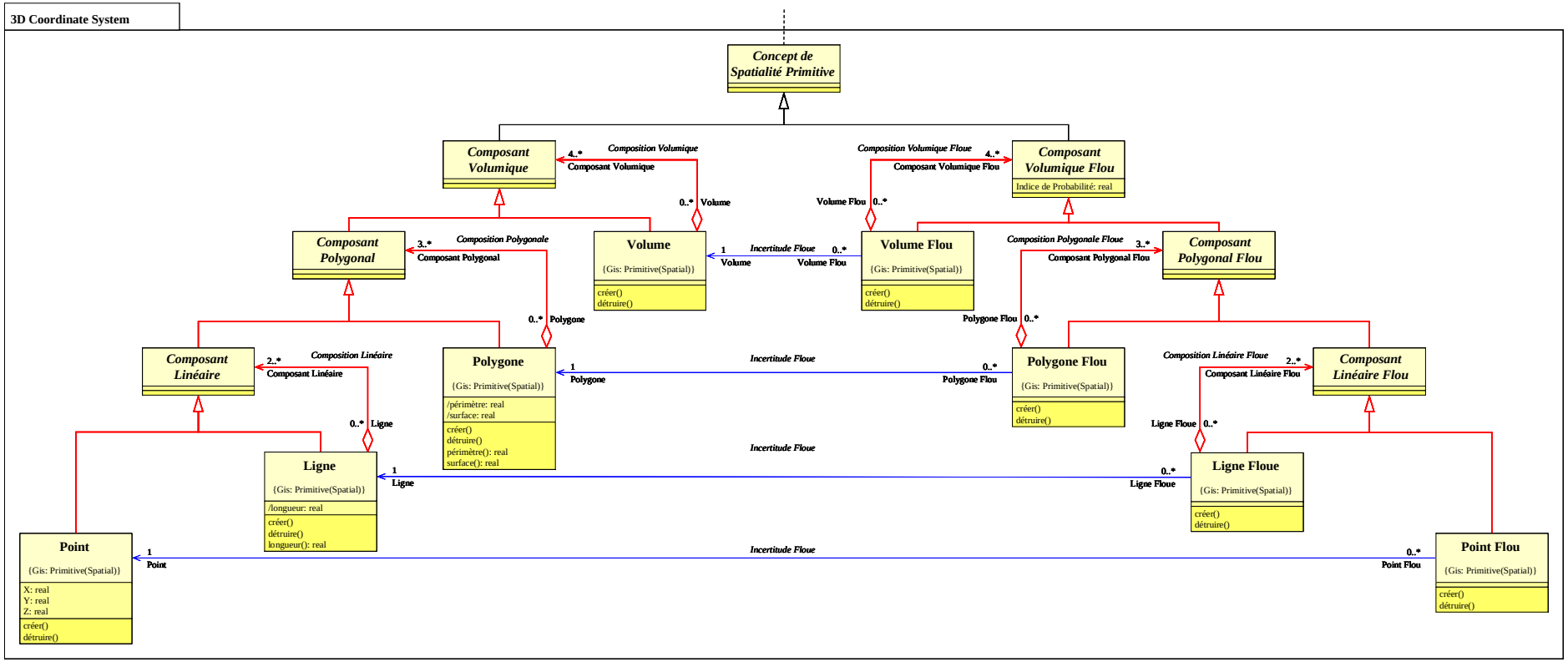


Figure 57 Patron de conception spatial 3D.

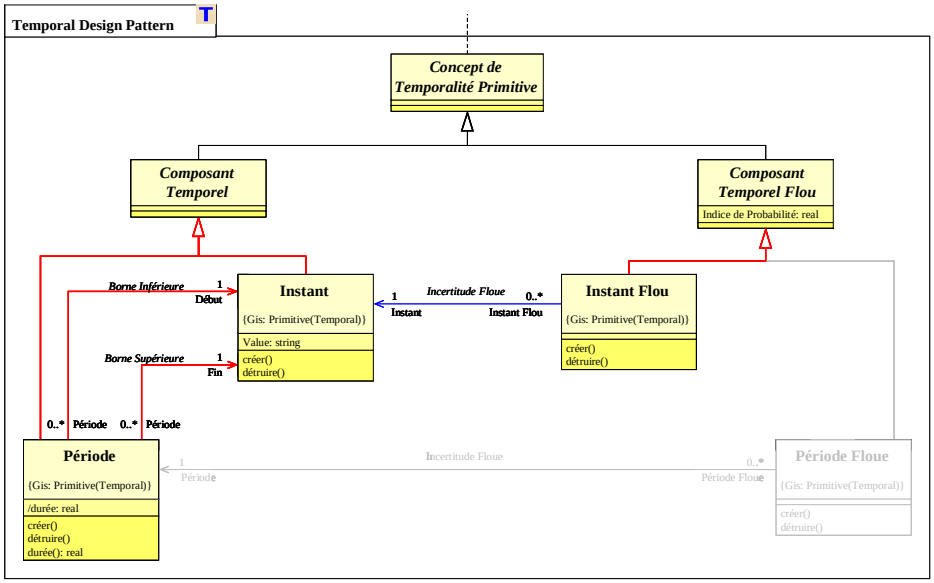


Figure 58 Patron de conception Temporel.

En résumé

La *Transformation de génération du patron de conception SIG* $T^{PD\ SM \rightarrow PD\ SM}_{Pat.\ SIG}$ crée, lorsque le concepteur l'exécute, les trois patrons de conception spatiaux 1D, 2D et 3D ainsi que le patron de conception temporel.

Les concepts SIG primitifs flous et non flous sont tous présents au sein du modèle en cours de réalisation mais ils n'ont aucune relation avec les entités référencées. C'est la *Transformation de traduction des pictogrammes* présentée au paragraphe suivant qui a la charge d'établir ces relations.

III.6 Conception de la Transformation de traduction des pictogrammes

Le *Patron de conception SIG* étant créé dans le sous-modèle de conception préliminaire, l'entité référencée avec sa pictogrammie d'une part et le ou les concepts SIG primitifs correspondant à cette pictogrammie sont totalement dissociés.

Le but de la Transformation de traduction des pictogrammes est d'établir automatiquement une relation entre l'entité référencée et le ou les concepts SIG définis par la pictogrammie de l'entité référencée.

III.6.1 *Description de la transformation*

III.6.1.1 Règle générale

III.6.1.1.a *Instanciation en généralisation*

La *Transformation de traduction des pictogrammes* $T^{PD\ SM \rightarrow PD\ SM}_{Tra.\ Picto.}$ convertit le couple stéréotype/pictogramme en une relation de généralisation entre l'entité référencée et chacun des concepts SIG primitifs impliqués dans sa pictogrammie. Cette règle est commune à tous les concepts de spatialité et de temporalité participant à la pictogrammie des entités référencées.

La figure 59 illustre cette règle. Le concept de spatialité *Polygone* de l'entité référencée *Parcelle* (cf. sous-modèle de conception préliminaire-Avant) constitue l'information pertinente qui permet d'établir la relation de généralisation entre l'entité référencée *Parcelle* et le concept *Polygone* (cf. Après).

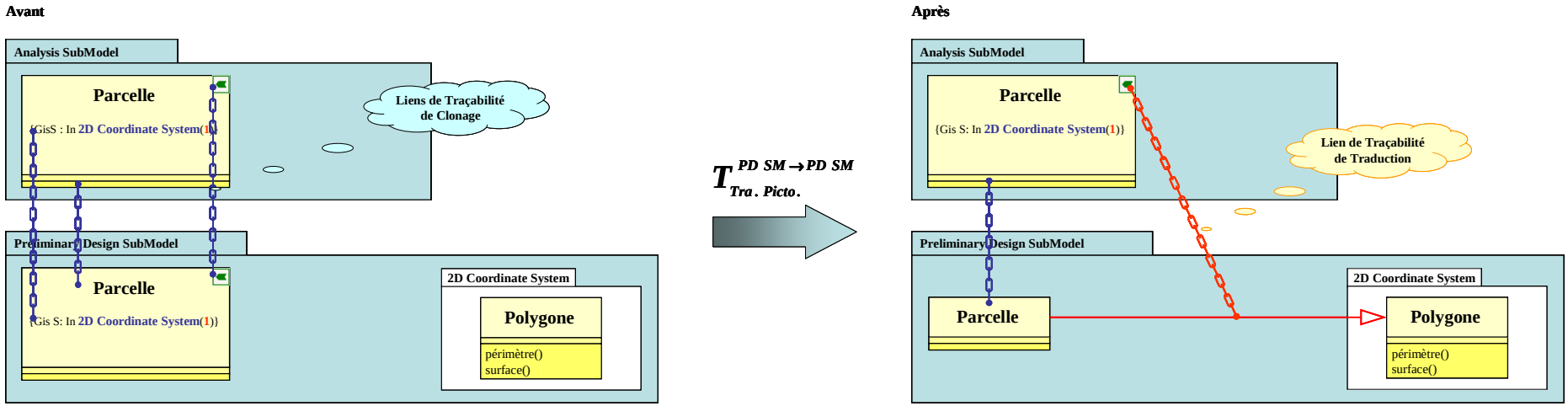


Figure 59 *Transformation de traduction des pictogrammes* en relation de généralisation.

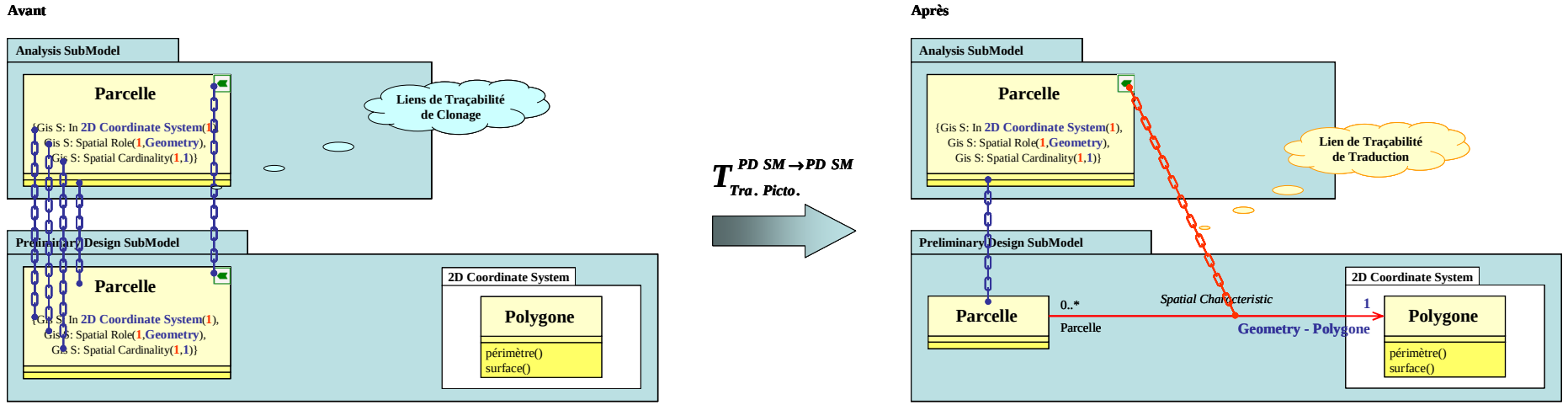


Figure 60 *Transformation de traduction des pictogrammes* en relation d'association.

Toutefois, nous avons vu au paragraphe III.5 que la *Transformation de génération du patron de conception SIG* crée plusieurs concepts *Polygone*, un par patron de conception spatial. La valeur marquée de dimension *{Gis S: In 2D Coordinate System(1)}* indique lequel des concepts *Polygone* est concerné.

Dans l'exemple de la figure 59, le concept de *Polygone* impliqué dans la transformation est celui du patron de conception spatial d'un système de coordonnés 2D. De plus, la transformation ne s'effectue que si le concept *Polygone* est annoté avec la valeur marquée *{Gis: Primitive(Spatial)}*.

Une fois la relation de généralisation établie, le couple stéréotype/pictogramme et la valeur marquée de dimension sont supprimés puisque ces informations sont redondantes avec la relation nouvellement créée. Cette suppression provoque une perte d'information ce qui correspond à une dégradation de la cohérence globale de l'artefact *Software Development Process Model*. Le troisième principe énoncé au paragraphe I est alors enfreint.

Encore une fois, les recommandations de Philippe Desfray (Desfray, 2001) concernant la traçabilité et sa gestion automatisée ont permis d'éviter la perte d'information. Pour ce faire, la transformation $\boldsymbol{T}_{Tra.\,Picto.}^{PD\ SM \rightarrow PD\ SM}$ établit un lien de traçabilité que nous avons désigné ***Lien de Traçabilité de Traduction*** entre le pictogramme de l'entité référencée appartenant au sous-modèle d'analyse et la relation de généralisation.

Nous verrons au paragraphe III.6.1.3 que, dans le processus itératif de la méthode *Continuous Integration Unified Process*, la *Transformation de traduction des pictogrammes* a nécessité d'être complétée par un post-traitement. Le lien de traçabilité de traduction s'est révélé très utile à cette occasion.

La règle de générale décrite ci-dessus s'applique pour tous les concepts SIG primitifs qu'ils soient de nature spatiale ou de nature temporelle.

Elle est toujours valable pour les concepts de spatialité et temporalité conjointe car les deux concepts primitifs sont de nature différente : l'un est spatial et l'autre temporel. Par contre, les concepts de spatio-temporalité, de spatialité multiple et de spatialité alternative posent quelques problèmes que nous allons aborder dans le paragraphe suivant.

III.6.1.1.b *Instanciation en association*

La description précédente de la *Transformation de traduction des pictogrammes* reste valable pour une instanciation de l'application d'affectation << en association. La différence entre les deux types d'instanciations réside dans la nature de la relation et dans la gestion des informations complémentaires afin que le modèle résultant soit sémantiquement cohérent.

La transformation $\boldsymbol{T}_{Tra.\,Picto.}^{PD\ SM \rightarrow PD\ SM}$ établit une relation d'association, nommée *Spatial Characteristic* ou *Temporal Characteristic* (cf. figure 60) selon que le concept SIG en cours de conversion est de nature spatiale ou temporelle. Le modèle obtenu étant sémantiquement incomplet, nous avons défini un mécanisme afin de le compléter en tenant compte au mieux des recommandations de l'Annexe I-III.4.1.

Dans cette association, le nom de l'entité référencée est aussi attribué à son rôle, *Parcelle* dans l'exemple de la figure 60, et sa cardinalité est valuée à 0..*. Afin d'éviter des conflits de nom[58], le rôle du concept SIG primitif est obtenu par concaténation de l'information stockée dans la valeur marquée de rôle avec le nom du concept SIG primitif (par exemple *Geometry - Polygone* en figure 60). La cardinalité provient quant à elle du contenu de la valeur marquée de cardinalité.

Toujours dans un souci de cohérence sémantique du modèle après transformation, les valeurs de rôle et de cardinalité proposées par défaut sont imposées. Le tableau 7 dresse la liste des rôles et des cardinalités suivant la typologie des concepts SIG.

	Rôle(s)	Cardinalité(s)
Spatialité primitive	Fixe : Geometry	Modifiable
Temporalité primitive	Fixe : Time	Modifiable
Spatialité et Temporalité Conjointe	Fixe : Geometry / Time	Modifiable
Spatio-temporalité	Fixe : Geometry / Time	Modifiable

[58] Ce problème se pose pour les entités de spatialité multiple où deux associations auraient les mêmes noms de rôle.

Spatialité Multiple	Modifiable	Fixe : 1
Spatialité Alternative	Fixe : Geometry / Geometry	Fixe : 0..1

Tableau 7 Valeurs des rôles et des cardinalités suivant la typologie des concepts SIG.

Finalement, seuls les rôles de la spatialité multiple sont modifiables, les autres sont nommés automatiquement. Inversement les cardinalités sont majoritairement modifiables excepté celles de spatialité multiple dont la valeur a été fixée à 1 et celles de spatialité alternative qui prend bien évidemment la valeur 0..1.

Toutefois, le concepteur a la possibilité d'intervenir sur les rôles et sur les cardinalités une fois la transformation effectuée s'il souhaite améliorer la cohérence sémantique du modèle. ***Il est déconseillé de modifier les cardinalités des spatialités multiples et des spatialités alternatives.***

Enfin, dans le cas des spatialités alternatives, la transformation $T_{Tra.\ Picto.}^{PD\ SM \rightarrow PD\ SM}$ établit une contrainte d'exclusion mutuelle XOR entre les deux associations *Spatial Characteristic*. La figure 61 montre l'entité référencée à spatialité alternative *Bâtiment* du sous-modèle d'analyse avec sa pictogrammie alternative et ses huit valeurs marquées. Elle montre également son clone du sous-modèle de conception préliminaire une fois la transformation des pictogrammes exécutée.

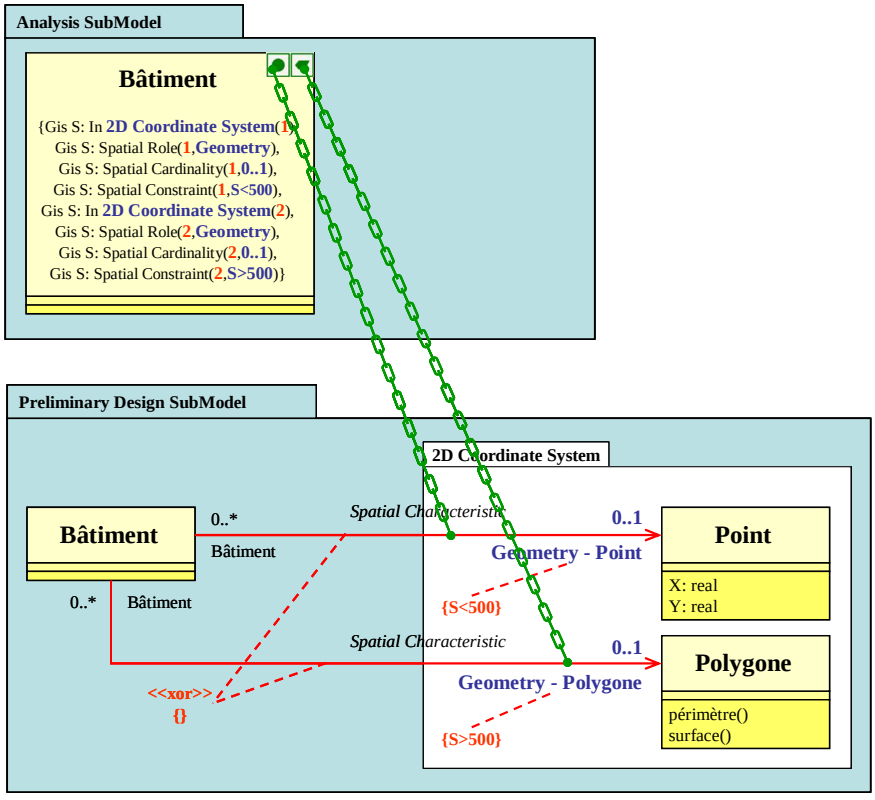


Figure 61 Entité référencée à spatialité alternative : Conversion de la pictogrammie en éléments de modélisation UML.

III.6.1.2 Cas particulier des concepts de spatio-temporalité

Les règles qui viennent d'être décrites pour l'instanciation en généralisation ou en association s'appliquent aux entités de spatialité primitive, de temporalité primitive, de spatialité et temporalité conjointe ainsi qu'à celles de spatialité multiple et de spatialité alternative.

Pour les entités référencées de spatio-temporalité, la conversion du couple stéréotype/pictogramme ne peut pas s'exécuter en une seule transformation $T^{PD\ SM \rightarrow PD\ SM}_{Tra.\ Picto.}$, il faut appliquer deux fois la *Transformation de traduction des pictogrammes*. Cela vient du fait que la temporalité ne porte pas directement sur le concept thématique mais sur le concept de spatialité (cf. définition 38).

La première transformation convertit le couple stéréotype/pictogramme de spatialité et, une fois celle-ci terminée, le concept de spatialité est annoté avec le stéréotype/pictogramme de temporalité (cf. figure 62-État transitoire). La transformation est appliquée une seconde fois pour finaliser la conversion de la pictogrammie (cf. figure 62-État final).

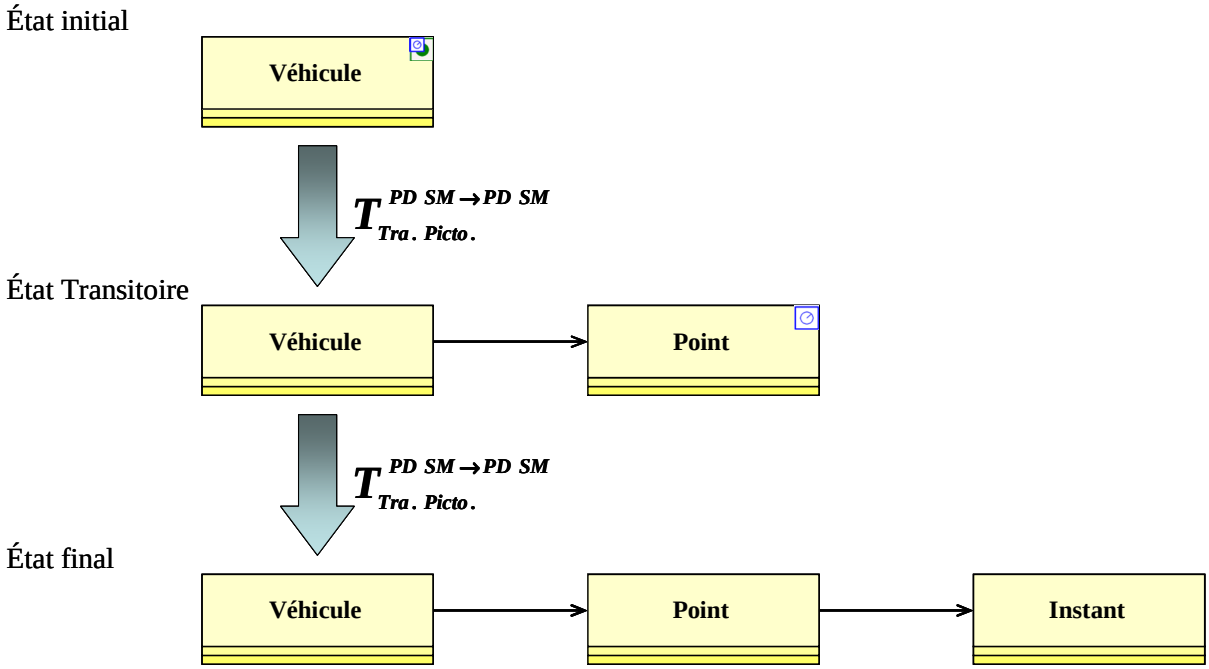


Figure 62 Les deux passes de la transformation $T^{PD\ SM \rightarrow PD\ SM}_{Tra.\ Picto.}$ au cours de la conversion de la spatio-temporalité.

La double transformation produit un modèle cohérent lors de l'instanciation de l'application d'affectation << en association. Par contre, l'instanciation en généralisation pose quelques problèmes de cohérence. En effet, au terme des deux passes, l'entité référencée de spatio-temporalité va hériter des propriétés et des comportements du concept spatialité, *Point* dans notre exemple, qui hérite à tour des propriétés et des comportements du concept de temporalité *Instant*. Le problème ne se pose pas pour l'entité de spatio-temporalité mais pour les autres entités ayant des spatialités sans temporalité, par exemple, une entité référencée *Point d'observation* ayant une spatialité primitive *Point*.

La transformation $T^{PD\ SM \rightarrow PD\ SM}_{Tra.\ Picto.}$ établirait, en application de la règle générale, une généralisation entre l'entité référencée *Point d'observation* et le concept de *Point*. Le problème est que, dans cette configuration, l'entité référencée *Point d'observation* acquiert les propriétés de spatio-temporalité au travers de la généralisation entre le concept de *Point* et le concept de *d'Instant*.

Ce problème nous conduit à interdire la saisie de la spatio-temporalité lors l'instanciation de l'application d'affectation << en généralisation.

III.6.1.3 Post-traitement à la transformation de diffusion

À la fin d'une itération N, le sous-modèle de conception préliminaire est dans l'état « Après » des figures 59 et 60. Au cours de l'itération N+1, le clonage de la *Transformation de diffusion* va re-stéréotyper l'entité référencée et, selon le cas concept SIG de spatialité ou de temporalité et instanciation en généralisation ou en association, elle va ajouter ou non les valeurs marquées de dimension, de rôle, de cardinalité et de contrainte.

La figure 63 montre l'état du sous-modèle de conception préliminaire à l'itération N+1 après l'exécution de la *Transformation de diffusion* et avant post-traitement.

L'entité référencée *Parcelle* a à nouveau un pictogramme *Polygone*, une valeur marquée de dimension *{Gis S: In 2D Coordinate System(1)}* et les liens de traçabilité de clonage de ces éléments de modélisation avec le sous-modèle d'analyse. En instanciation de l'application d'affectation << en association, *Parcelle* aurait en plus une valeur marquée de rôle, une autre de cardinalité et, dans certains cas, de contrainte.

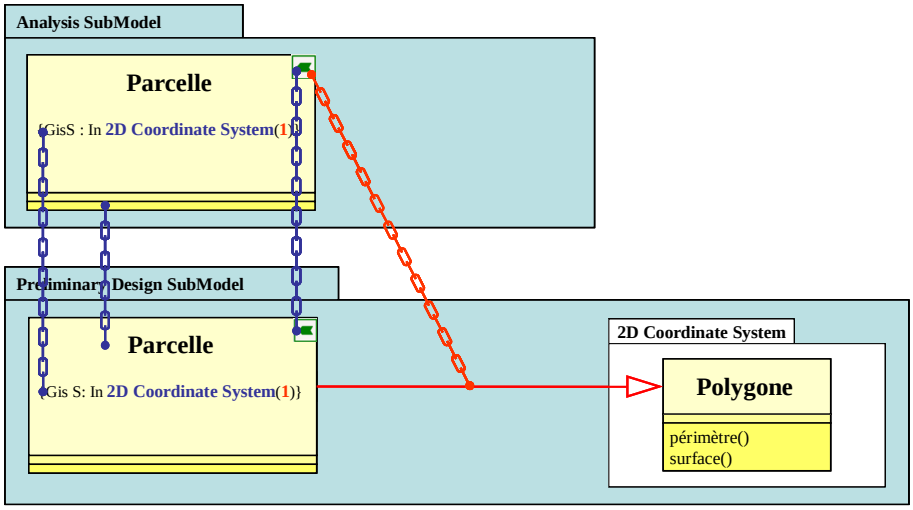


Figure 63 État du modèle suite à la *Transformation de diffusion* et avant post-traitement.

Dans cette configuration, le sous-modèle de conception préliminaire est incohérent car l'entité référencée *Parcelle* a simultanément les annotations de spatialité et une relation de généralisation avec le concept *Polygone*. Il y a redondance d'information.

Afin de reconstituer la cohérence du sous-modèle de conception préliminaire, le post-traitement à la *Transformation de diffusion* a pour responsabilité de supprimer la pictogrammie et les valeurs marquées excédentaires de l'entité référencée ayant déjà subi la *Transformation de traduction des pictogrammes*. Le problème est d'identifier sans ambiguïté ce type d'entités référencées incohérentes. C'est la raison qui a conduit à ajouter le lien de Traçabilité de Traduction.

Étant donné qu'il est établi par la transformation $T_{Tra.\,Picto.}^{PD\ SM \rightarrow PD\ SM}$ une fois celle-ci terminée, sa présence prouve que l'entité référencée a subi au moins une fois la *Transformation de traduction des pictogrammes*.

Un second avantage du lien de Traçabilité de Traduction est qu'il identifie sans ambiguïté les concepts impliqués dans la transformation $T_{Tra.\,Picto.}^{PD\ SM \rightarrow PD\ SM}$. Aussi, doté de ce lien, le sous-modèle contient toute l'information pour réaliser la transformation inverse[59].

III.7 Comparaison concepts ISO / concepts du Patron de conception SIG

Le *Patron de conception SIG* qui est généré par la transformation de Génération des Patrons de Conception $T_{Pat.\,SIG}^{PD\ SM \rightarrow PD\ SM}$ et « relié » au modèle du système étudié par la *Transformation de traduction des pictogrammes* $T_{Tra.\,Picto.}^{PD\ SM \rightarrow PD\ SM}$ permet de modéliser un certain nombre de problématiques rencontrées en cours de modélisation par les concepteurs de *Systèmes d'Information Géographique*.

Le Comité Technique 211 de l'ISO a élaboré les schémas spatiale (ISO, 2003a) et temporel (ISO, 2002) *décrivant les caractéristiques spatiales et temporelles de l'information géographique*.

Bien évidemment, un certain nombre de concepts de l'ISO ont une sémantique identique à celle des concepts SIG du Patron de concepts SIG, d'autres ont des sémantiques proches et d'autres sont totalement différents. Le

[59] Pour l'instant, la transformation inverse n'a pas été implémentée.

but ici n'est pas de faire une étude comparative exhaustive des concepts ISO et des concepts SIG mais simplement de donner les grandes similitudes et les grandes différences. Cette comparaison est principalement centrée sur les concepts couramment utilisés dans les bases de données spatio-temporelles, aussi elle exclut d'une part les concepts de nature topologique (TP_Node, TP_Face, TM_Period, etc.) et ceux représentant les concepts continus comme les courbes de Bézier, les splines, etc.

III.7.1 *Concepts ISO couverts par les concepts SIG*

Afin de les comparer, les concepts ISO et SIG de même sémantique ont été regroupés dans les tableaux 8 et 9. Pour chaque concept ISO, ces tableaux indiquent le principal concept SIG de l'annotation, la cardinalité de l'association **<<** et, lorsque cela est nécessaire, d'une part, le patron de conception composite impliqué et, d'autre part, la nature et le nombre de concepts impliqués dans la composition.

Concept ISO	Concept SIG principal	Cardinalité de l'association **<<**	Patron de conception composite impliqué	Nombre de concepts SIG utilisés dans le patron composite
GM_Point	Point	1		
GM_LineSegment	Ligne	1	Agrégation de *Composition Linéaire*	2 concepts *Points*
GM_LineString	Ligne	1	Agrégation de *Composition Linéaire*	N concepts *Points*
GM_Curve	Ligne	1	Agrégation de *Composition Linéaire*	N concepts *Lignes*
GM_Surface	Polygone	1	Agrégation de *Composition Polygonale*	N concepts *Polygones*
GM_Polygon	Polygone	1	Agrégation de *Composition Polygonale*	1 ou N concepts *Ligne*
GM_Triangle	Polygone	1	Agrégation de *Composition Polygonale*	4 concepts *Points*
GM_Solid	Volume	1	Agrégation de *Composition Volumique*	1 ou N concepts *Polygones*
GM_MultiPoint	Point	N		
GM_MultiCurve	Ligne	N		
GM_MultiSurface	Polygone	N		
GM_MultiSolid	Volume	N		
GM_CompositePoint	Point	1		
GM_CompositeCurve	Ligne	1	Agrégation de *Composition Linéaire*	N concepts *Lignes*
GM_CompositeSurface	Polygone	1	Agrégation de *Composition Polygonale*	N concepts *Polygones*
GM_CompositeSolid	Volume	1	Agrégation de *Composition Volumique*	N concepts *Volumes*

Tableau 8 Comparaison des concepts ISO et SIG de nature spatiale.

Comme le montre tableau 9, le *Patron de conception SIG* permet de créer, grâce à la cardinalité, les concepts de multi-Instant et multi-Période, concepts qui sont absents de la norme ISO 19108 définissant le schéma temporel de l'*Information Géographique*.

Concept ISO	Concept SIG principal	Cardinalité de l'association <<	Patron de conception composite impliqué	Nombre de concepts SIG utilisés dans le patron composite
GM_Instant	Instant	1		
GM_Period	Période	1		
	Instant	N		
	Période	N		

Tableau 9 Comparaison des concepts ISO et SIG de nature temporelle.

La dérivation du *Patron de conception SIG* du *Métamodèle SIG*, l'instanciation en association de l'application d'affectation **<<** et la structuration des concepts SIG primitifs autour de patrons de conception composite imbriqués simplifient de façon drastique les concepts à manipuler. Pour autant, ils sont très complets puisque les quatre concepts SIG de spatialité primitive permettent de modéliser seize concepts ISO.

Cette apparente perte de sémantique n'en est pas une car le *Patron de conception SIG* est généré dans le sous-modèle de conception préliminaire de l'artefact *Software Development Process Model*. Ce sous-modèle est essentiellement dédié aux développeurs du *Système d'Information Géographique* en cours de modélisation alors que les acteurs et le concepteur ne manipulent que les langages SIG et pictogrammique. Ces derniers ne sont accessibles que dans le sous-modèle d'analyse. La capacité d'expression reste donc intacte et les développeurs disposent d'un *Patron de conception SIG* riche au vu du nombre de concepts ISO couverts tout en restant bien plus simple que celui de l'ISO.

III.7.2 *Différences majeures*

Par contre, les normes ISO 19107 et 19108 n'offrent pas au concepteur d'un *Système d'Information Géographique* un certain nombre de concepts de spatialité et de temporalité.

Les concepts de spatialité alternative et temporalité alternative qui sont obtenus très simplement dans le contexte du *Patron de conception SIG* via la cardinalité de l'association **<<** ne sont pas abordés dans ces normes. Les travaux de (Brodeur *et al.*, 2002) confirment cette absence puisque pour implémenter dans une base la spatialité alternative ou la temporalité alternative, il faut créer deux attributs et une contrainte d'exclusion mutuelle.

Les concepts de spatialité et temporalité conjointe eux-aussi ne sont pas définis en tant que tel. Toutefois, comme les concepts de nature spatiale et de nature temporelle de l'ISO sont censés renseigner le concept thématique, ils sont implicitement utilisés.

L'absence des concepts de spatio-temporalité de ces deux normes est plus surprenante. En effet, cette typologie de concepts est utilisée pour annoter tous les concepts thématiques en mouvement ou qui se déforment. Cela représente un nombre très élevé de concepts dont certains sont très couramment utilisés (Véhicule par exemple). Il est vrai que l'étude séparée des concepts spatiaux d'une part et des concepts temporels d'autre part ne facilite pas la définition des concepts qui sont issus d'une combinaison des deux. Ceux-ci sont tous les concepts inclus dans l'ensemble ***ST*** de la taxinomie des concepts SIG (cf. Chapitre 7). (Brodeur *et al.*, 2002) signalaient d'ailleurs que cette famille de concepts devait faire l'objet d'un nouveau thème de travail au sein du TC 211. Cette remarque montre l'intérêt de l'étude terminologique du Chapitre 7.

Bien évidemment, les concepts de spatialité floue et temporalité floue sont absents de ces normes puisqu'ils constituent l'une des contributions du présent travail de recherche. Ces concepts flous ont été définis pour résoudre des cas de figures que les concepteurs rencontrent au cours de la modélisation de *Systèmes d'Information Géographique* sur l'environnement. Le « traitement » en parallèle des concepts flous et non flous a permis de doter les premiers d'un modèle calqué sur celui des concepts non flous, modèle qui est instanciable dans une base de données.

Excepté le concept de *Volume* correspondant à GM_Solid, le *Patron de conception SIG* ne possède aucun des types de solide proposés par l'ISO car l'étude de cette typologie de concepts a été reportée à une phase ultérieure.

Il existe une dernière différence majeure. Elle concerne les schémas spatiaux et temporels de l'ISO et le *Patron de conception SIG*. Cette différence porte sur deux points.

Le premier point découle de la méthode de travail. Dans l'étude terminologique du Chapitre 7, le problème a été posé dans sa globalité c'est-à-dire en prenant en compte les *propriétés spatiales et temporelles* des entités référencées dès le départ alors que les travaux de l'ISO ont séparé les deux types de propriétés. Aussi, alors que l'ISO propose deux schémas séparés, la transformation $T_{Pat.\ SIG}^{PD\ SM \rightarrow PD\ SM}$ génère un seul *Patron de conception SIG* incluant les patrons spatiaux et temporel.

Le second point est relatif à la dimension du système de coordonnées. Ce dernier influe sur le nombre de coordonnées du concept *Point* mais aussi les concepts de spatialité primitive disponible par dimension. Les concepts polygonaux et volumiques n'existant pas dans un système de coordonnées 1D ni dans un système 2D, il n'est pas nécessaire qu'ils soient présents dans le patron destiné à cette dimension. Il en est de même pour les concepts volumiques dans un système de coordonnées 2D. C'est ce constat qui a conduit à générer trois modèles de conception spatiaux (cf. III.5.1) dans le *Patron de conception SIG*. Dans le schéma spatial de l'ISO, il n'y a qu'un seul et même modèle quelle que soit la dimension du système de coordonnées.

Dans le concept GM_Point, la dimension du système de coordonnées est prise en compte via l'attribut *position* de type *DirectPosition* (cf. figure 64). Ce type de donnée possède lui-même un attribut, *coordinate*, qui est une liste séquentielle de valeurs. Le nombre de valeurs informe sur la dimension du système de coordonnées. Or, comme les concepts GM_LineString, GM_Polygon, etc. sont définis in fine comme une liste de GM_Point, rien ne s'oppose à introduire dans cette liste des GM_Point appartenant à des systèmes de coordonnées de dimension différente (1D, 2D et 3D). Dans ce contexte, le *Système d'Information Géographique* ou l'application rencontreront des dysfonctionnements.

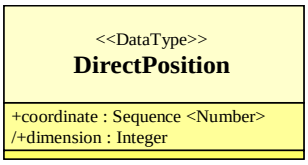


Figure 64 Modèle du concept ISO *DirectPosition* (ISO, 2003a).

C'est un problème syntaxique qui a été résolu dans l'approche proposée ici puisque la transformation $T_{Pat.\ SIG}^{PD\ SM \rightarrow PD\ SM}$ ***génère, dans le Patron de conception SIG, trois patrons de conception spatiaux un par dimension de systèmes de coordonnées.***

IV TRANSFORMATIONS SQL

IV.1 Le générateur de code SQLDesigner dans l'artefact Software Development Process Model

Le sous-modèle d'implémentation SQL obtenu par clonage du sous-modèle de conception avancée (cf. figure 65-État : *After Cloning*) ne renferme aucune des spécifications SQL indispensables à la transformation $T^{SQL\ SM \rightarrow SQL\ P\ SM}$ [60] du générateur de code SQLDesigner.

Dans le fonctionnement actuel de ce générateur, le concepteur n'a pas d'autre alternative que d'ajouter manuellement les spécifications SQL. Dès l'ajout de la première spécification SQL, le sous-modèle d'implémentation SQL est un état intermédiaire (cf. figure 65-État : *SQL Specifications*) où le concepteur a commencé à introduire des spécifications SQL mais n'a pas terminé. Sur les modèles de plusieurs dizaines de

[60] Nom de la transformation $T^{AM \rightarrow PM}$ adapté au contexte du Software Developement Process Model (cf. Annexe V-I).

concepts, cette phase peut prendre plusieurs heures. L'ajout des spécifications SQL s'effectue manuellement. La dernière spécification SQL ajoutée, le sous-modèle d'implémentation se trouve dans l'état *Ready for mapping to Physical SubModel*. Le générateur de code prend alors la relève et crée automatiquement le *sous-modèle physique d'implémentation SQL* au même niveau que le sous-modèle d'implémentation SQL. Le nouveau sous-modèle porte le même nom que le sous-modèle d'implémentation SQL précédé de la chaîne de caractères « MPD_ ».

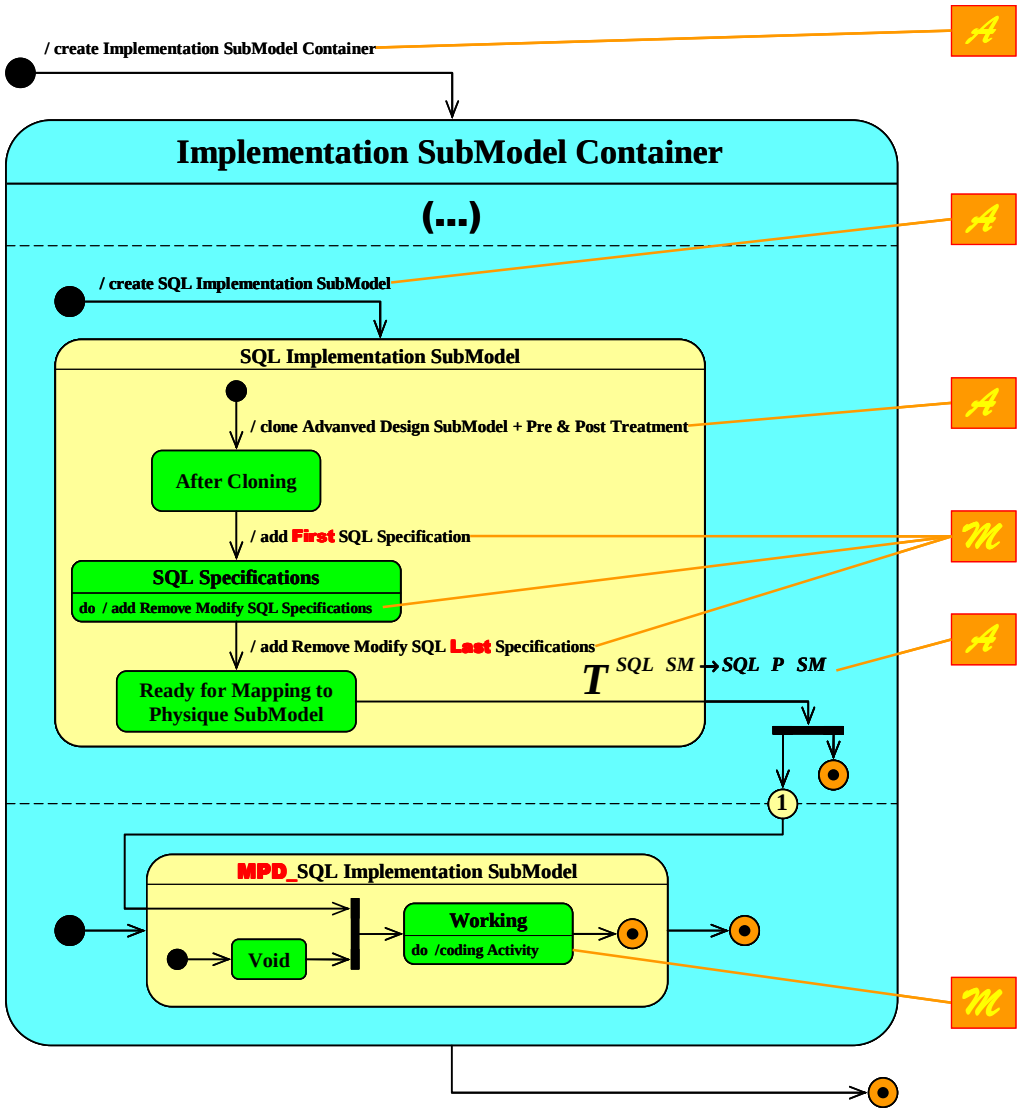


Figure 65 Le générateur de code SQLDesigner dans l'artefact *Software Development Process Model*.

Dans le sous-modèle physique d'implémentation SQL, le concepteur a encore la possibilité d'ajouter des spécifications SQL pour adapter le modèle à un SGBDR cible, de réaliser des requêtes SQL, etc. Toutes ces actions sont aussi réalisées manuellement.

Ayant adopté la méthode *Continuous Integration Unified Process* avec l'objectif d'automatiser la voie technologique SQL, les interventions manuelles doivent être réduites au strict minimum. Les transformations SQL du sous-modèle d'implémentation SQL automatisent le passage de ce sous-modèle entre les états *After Cloning* et *Ready for mapping to Physical SubModel*.

Pour ce faire, il faut ajouter les spécifications SQL de persistance, de clé primaire et de généralisation sous-modèle d'implémentation SQL automatiquement (cf. Annexe V) et faire quelques adaptations du sous-modèle. C'est l'objet des paragraphes suivants.

IV.2 Conception des transformations SQL

IV.2.1 *Transformation de spécification multiple*

Pour qu'une classe soit instanciée en table, elle doit être persistante. Étant donné que les classes filles héritent des caractéristiques des classes mères, elles héritent donc de la persistance (cf. Annexe V-II.1). Aussi, les seules classes à annoter dans un modèle sont les classes isolées[61] et les classes racines d'une hiérarchie de classes. ***L'analyse de ces deux types de classes montre qu'elles se caractérisent par l'absence de classe mère (cf. figures 66 et 67)***. C'est la règle qui a permis de les identifier sans ambiguïté au sein d'un modèle. Nous appellerons C_p l'ensemble des classes persistantes.

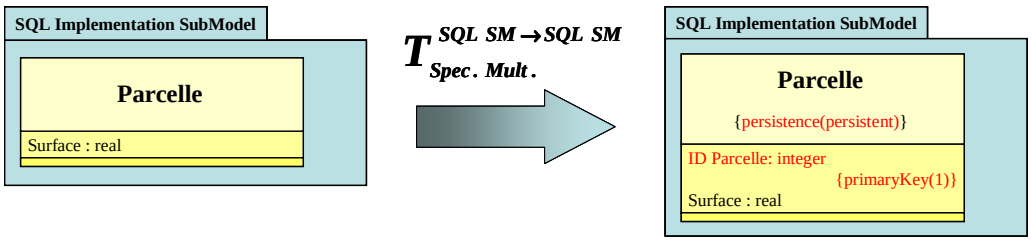


Figure 66 *Transformation de spécification multiple* pour les classes persistantes non racine de hiérarchie.

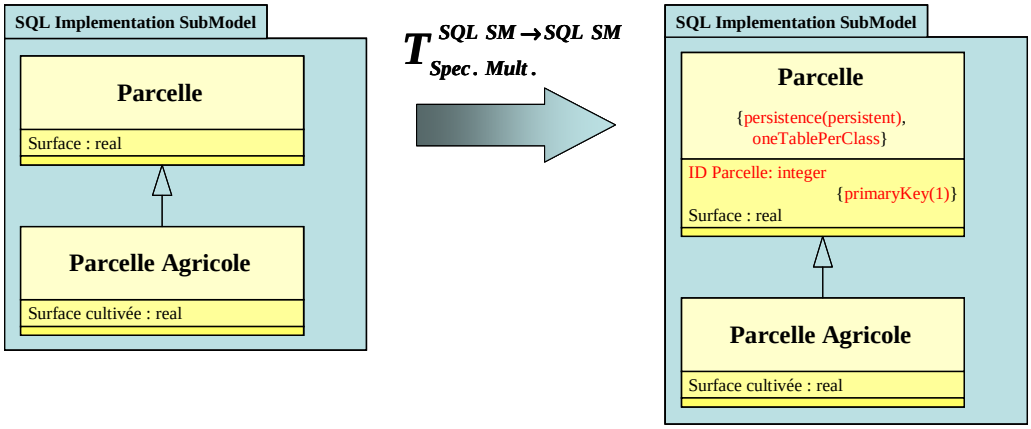


Figure 67 *Transformation de spécification multiple* pour les classes racines.

Bien qu'une clé primaire ne soit obligatoire que sur les classes ayant au moins une relation d'association, ***nous avons fait le choix d'ajouter systématiquement une clé primaire à toutes les classes persistantes***[62] ***(cf. Annexe V-II.2)***. Un attribut annoté est donc systématiquement ajouté à chacune des classes de l'ensemble C_p.

En Annexe V-II.3, sont décrites les différentes techniques permettant de pallier l'absence du mécanisme de généralisation/spécialisation dans la majorité des SGBDR. Parmi les trois techniques proposées par le générateur de code SQLDesigner, ***nous avons adopté celle qui préserve le mieux le formalisme Objet du langage UML c'est-à-dire la technique « Une Table par Classe »***. Il est aussi précisé dans cette rubrique que l'annotation doit être apposée sur classe racine de la hiérarchie de classes. ***Les classes racines se caractérisent par une absence de classes mères et l'existence de classes filles***. Les classes racines sont en fait des classes persistantes ayant des classes filles et forment le sous-ensemble noté C_r de l'ensemble des classes persistantes C_p.

Une première stratégie aurait pu être d'implémenter indépendamment les trois opérations en créant trois transformations individuelles. L'inconvénient de cette stratégie est qu'il faut localiser les classes de l'ensemble

[61] Classe n'ayant aucune classe mère et aucune classe fille.
[62] Un test complémentaire sur l'existence d'AssociationEnd aurait pu exclure les classes sans association.

C_p à trois reprises ce qui constitue une perte de temps. Fort de ce constat, une stratégie plus performante consiste à implémenter dans la même transformation les trois opérations.

La *Transformation de spécification multiple* $T^{SQL\ SM \rightarrow SQL\ SM}_{Spec.\ Mult.}$ résulte de cette réflexion. Elle a la responsabilité d'ajouter :

- La valeur marquée de persistence (*{persistence(persistent)}*) du générateur de code SQLDesigner aux classes de l'ensemble C_p.
- Un attribut dont le nom est obtenu par concaténation de la chaîne de caractères « ID » et du nom de la classe. Cet attribut est aussi annoté avec la valeur marquée *{primaryKey(1)}*, lui affectant le statut de clé primaire (cf. figure 66).
- Enfin, les classes racines de l'ensemble C_r reçoivent en plus la valeur marquée *{oneTablePerClass}* (cf. figure 67).

IV.2.2 *Transformation de suppression des opérations*

Cette deuxième transformation corrige un fonctionnement estimé gênant dans l'objectif de génération d'une base de données. La transformation $T^{SQL\ SM \rightarrow SQL\ P\ SM}$ supprime systématiquement les relations des classes ayant des opérations et dénudées d'attributs (cf. Annexe V-III.1).

La perte des relations étant inacceptable car la structure de la base de données serait fortement altérée, ***nous avons le choix de supprimer les opérations des classes dénuées d'attributs privilégiant ainsi la structure de la base de données***.

La *Transformation de suppression des opérations* $T^{SQL\ SM \rightarrow SQL\ SM}_{Supp.\ Opé.}$ parcourt les éléments du sous-modèle d'implémentation SQL, teste l'absence d'attributs des classes et la présence d'opérations. Dans l'affirmative, elle supprime les opérations s'il y en a (cf. figure 68).

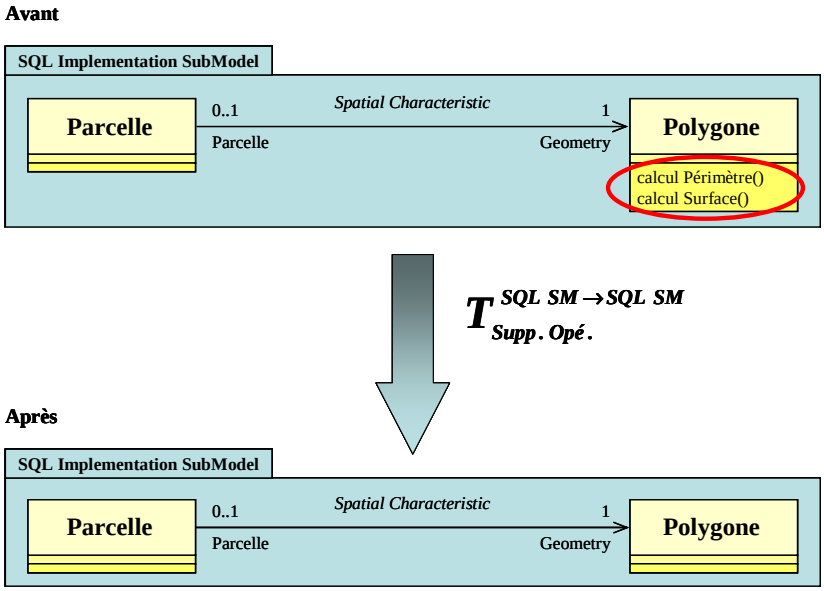


Figure 68 *Transformation de suppression des opérations* d'une classe dénudée d'attributs.

La *Transformation de suppression des opérations* $T^{SQL\ SM \rightarrow SQL\ SM}_{Supp.\ Opé.}$ ne s'appliquant qu'au seul sous-modèle d'implémentation SQL, elle n'altère aucun autre sous-modèle d'implémentation (C++, Java, etc.).

Cette dichotomie des sous-modèles d'implémentation représente un avantage incontestable de l'artefact *Software Development Process Model* qui permet de gérer en cohérence plusieurs sous-modèles.

IV.2.3 *Transformation d'externalisation de la relation de composition*

La troisième transformation a été implémentée afin d'éviter l'affichage d'avertissement systématique lorsque le modèle contient une classe fille avec une relation de composition (cf. figure 169-Motifs A et B).

En soi, ce message n'a aucun caractère de gravité d'autant plus qu'il ne suspend pas l'exécution de la transformation $T^{SQL\ SM \rightarrow SQL\ P\ SM}$ mais lorsqu'il s'affiche de façon récurrente à chaque itération, c'est désagréable surtout lors de l'analyse en présence des acteurs.

Afin de suspendre l'affichage, les relations de composition impliquées dans des motifs A et B ont été systématiquement convertis en tables. La transformation $T^{AM \rightarrow PM}$ effectue cette conversion lorsqu'elle rencontre une relation de composition annotée avec la valeur marquée *{External}*. Les colonnes des tables contiennent les clés primaires des classes d'extrémité de la relation.

La Transformation d'Externalisation de la Composition ($T^{SQL\ SM \rightarrow SQL\ SM}_{Ext.\ Comp.}$) recherche les classes filles dans les hiérarchies de concepts et vérifient si ces dernières ont une relation de composition. Si cette règle est remplie, la transformation $T^{SQL\ SM \rightarrow SQL\ SM}_{Ext.\ Comp.}$ ajoute la relation de la valeur marquée *{External}* (cf. figure 69).

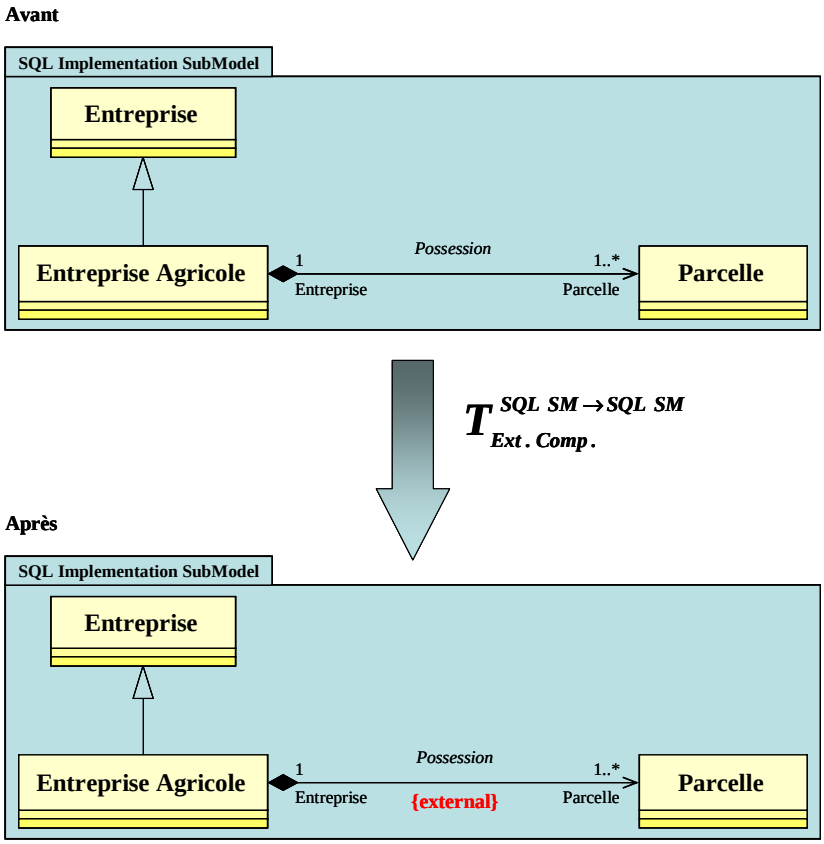


Figure 69 Transformation d'Externalisation des Relations de Composition.

V CONCLUSION

V.1 Transformations de gestion de l'artefact Software Development Process Model

Malgré le caractère multimodèle de l'artefact *Software Development Process Model*, la *Transformation de diffusion* permet à elle seule de gérer cet artefact complexe et ce en assurant sa cohérence. Elle diffuse les concepts depuis le sous-modèle d'analyse jusqu'au(x) sous-modèle(s) d'implémentation sous contrôle du

concepteur puisque ce dernier dispose des annotations d'invalidation de diffusion pour piloter la diffusion des concepts à sa guise.

Une « adaptation marginale » de cette transformation a permis de concevoir la *Transformation de rétrodiffusion* qui, à l'inverse de la précédente, assure la diffusion des concepts depuis un sous-modèle d'implémentation jusqu'au sous-modèle d'analyse.

De plus, l'utilisation conjointe des transformations de diffusion et de rétrodiffusion permet de transférer des concepts d'un sous-modèle d'implémentation à l'autre en rétrodiffusant, dans un premier temps, les concepts depuis le sous-modèle d'implémentation source vers le sous modèle de conception avancée et ensuite en les diffusant vers le sous-modèle d'implémentation cible. L'avantage du mécanisme décrit est que ce transfert est réalisé en maintenant l'ensemble des modèles en cohérence.

À elle deux[63], les transformations de diffusion et de rétrodiffusion réalisent la gestion inter-sous-modèles de l'artefact *Software Development Process Model* tout en garantissant la cohérence de l'ensemble des sous-modèles.

V.2 Transformations géomatiques

Les quatre transformations géomatiques ont été conçues et implémentées en vue d'automatiser la génération et l'évolution du modèle d'un *Système d'Information Géographique*.

Les deux premières, les transformations de Saisie $\boldsymbol{T}_{Sai.\,Pict.}^{An\ SM \rightarrow An\ SM}$ et de Suppression $\boldsymbol{T}_{Rem.\,Pict.}^{An\ SM \rightarrow An\ SM}$ de la Pictogrammie, assistent le concepteur dans son activité de modélisation tout en garantissant la cohérence. Elles s'appliquent au sous-modèle d'analyse de l'artefact *Software Development Process Model* puisque c'est le sous-modèle privilégié où sont introduits les concepts du domaine analysé.

Les transformations de *Génération du Patron de Conception* SIG $\boldsymbol{T}_{Pat.\,SIG}^{PD\ SM \rightarrow PD\ SM}$ et de *Traduction des Pictogrammes* $\boldsymbol{T}_{Tra.\,Picto.}^{PD\ SM \rightarrow PD\ SM}$ sont des transformations du sous-modèle de conception préliminaire de l'artefact *Software Development Process Model*.

La transformation $\boldsymbol{T}_{Pat.\,SIG}^{PD\ SM \rightarrow PD\ SM}$ réifie le *Patron de conception SIG* contenant les concepts SIG primitifs flous et non flous nécessaires à l'implémentation d'un *Système d'Information Géographique* ainsi que les relations entre concepts SIG primitifs. Ce *Patron de conception SIG* est un résultat de l'étude terminologique du Chapitre 7 puisqu'il dérive du *Métamodèle SIG* décrivant le langage SIG.

Dans le contexte itératif de la méthode *Continuous Integration Unified Process*, la *Transformation de traduction des pictogrammes* $\boldsymbol{T}_{Tra.\,Picto.}^{PD\ SM \rightarrow PD\ SM}$ a nécessité de compléter la *Transformation de diffusion* par un post-traitement afin que d'une part, le couple stéréotype/pictogramme et, d'autre part, la relation de généralisation ou d'association restent en cohérence. Cela a été possible grâce à l'utilisation de liens de traçabilité de traduction conformément aux recommandations de Philippe Desfray (Desfray, 2001).

Contrairement à la plupart des ateliers de génie logiciel dédiés à la conception des *Systèmes d'Information Géographique* (Perceptory, MADS, etc.) qui projettent directement un modèle d'analyse/conception en code SQL, les trois transformations géomatiques décrivent pas à pas l'évolution des concepts SIG offrant ainsi la possibilité au concepteur de ré-intervenir sur le modèle et de l'améliorer à chaque phase. Il reste donc maître du processus de développement et de l'évolution du modèle.

Une seconde différence importante avec ces ateliers est que ce couple de transformations réifie le *Patron de conception SIG* alors que, dans ces ateliers de génie logiciel, les concepts SIG sont introduits sous forme d'annotations ou de pictogrammes mais les relations entre concepts SIG restent virtuelles car leur réification est déléguée au SGBDR cible. La réification des concepts SIG et des relations est un atout de l'artefact *Software Development Process Model* car les acteurs du domaine, le concepteur et les développeurs ont accès au *Patron de conception SIG*. S'il ne convient, il peut être adapté au contexte conformément aux préconisations d'Éric Gamma (cf. définition 80).

Enfin, les langages SIG et pictogrammiques définis au Chapitre 7 sont des langages formels au sens de la théorie des langages (cf. Annexe III-III). C'est le cas pour le langage SIG puisque tout concept SIG est un symbole non terminal qui sera être remplacé, en phase de conception préliminaire, par une classe réifiant le

[63] La *Transformation d'annihilation de la rétrodiffusion* $\boldsymbol{T}_{An.\,Rét.\,Dif.}^{SM \rightarrow SM}$ n'est pas comptée car elle est complémentaire à la transformation de rétrodiffusion.

concept SIG et, selon le cas, une relation de généralisation ou d'association entre l'entité référencée et cette classe. La classe et la relation constituent alors les symboles terminaux. C'est aussi le cas pour le langage pictogrammique doté de ses conventions puisque chaque combinaison de pictogrammes (symboles non terminaux) sera réifiée en une classe et une relation (généralisation ou association) qui sont les symboles terminaux selon la théorie des langages.

V.3 Transformations SQL

Les trois transformations du sous-modèle d'implémentation SQL ont été pensées pour faciliter la tâche des concepteurs et permettre de réaliser le processus full MDA exigé par le *processus de développement permettant le prototypage rapide en séance d'analyse*.

Les transformations de Spécification Multiple, de Suppression d'Opérations sont des règles et des connaissances de nature informatique appliquées intuitivement par tout concepteur utilisant le générateur de code SQLDesigner ou imposées par la société en charge du développement de l'application. Nous les avons simplement formalisées et implémentées dans un objectif d'automatisation respectant les principes 1, 2 et 3 du paragraphe I.

La difficulté majeure rencontrée lors de la conception de ces trois transformations SQL a été de les intégrer dans le processus itératif de la méthode *Continuous Integration Unified Process*.

Ces trois transformations SQL sont d'autant plus intéressantes que le nombre de concepts du modèle est important.

V.4 Générale

Doté de seulement neuf transformations[63], l'artefact multimodèle *Software Development Process Model* constitue une plate-forme :

- ⇨ De saisie des spatialités et/ou des temporalités des entités référencées en mettant en œuvre un langage pictogrammique.
- ⇨ De suppression cohérente des spatialités et des temporalités des entités référencées.
- ⇨ De diffusion des concepts entre les différents sous-modèles de cet artefact.
- ⇨ D'enrichissement du sous-modèle de conception préliminaire par un *Patron de conception SIG* réifiant les concepts SIG flous en non flous.
- ⇨ De conversion des pictogrammes des entités référencées en relations.
- ⇨ D'adaptation du sous-modèle d'implémentation SQL au générateur de code de l'atelier de génie logiciel utilisé. Ce dernier prend le relai pour projeter ce sous-modèle en code SQL.

Ces neuf transformations sont associées à des activités de modélisation au sein de l'artefact *Software Development Process Model*, activités qui peuvent être représentées par un diagramme d'états précisant le ou les sous-modèles sur lesquels elles s'appliquent (cf. figure 70).

L'atelier de génie logiciel Objecteering dans lequel va être implémenté l'artefact *Software Development Process Model* (cf. Chapitre 6) et les neuf transformations satisferont l'objectif général fixé puisque cet atelier deviendra un *outil d'aide à la conception de Systèmes d'Information Géographique* doté de la capacité d'expression du langage pictogrammique dédié à l'expression des propriétés spatiales et temporelles des entités référencées.

L'artefact *Software Development Process Model* garantira l'objectif élémentaire de *capitalisation des connaissances* (objectif 3) et les neuf transformations assureront la gestion en cohérence de cet artefact et l'automatisation de l'évolution d'un modèle depuis l'analyse jusqu'à l'implémentation.

L'atelier de génie logiciel Objecteering permettra alors de mettre en œuvre le *processus de développement permettant le prototypage rapide en séance d'analyse* qui fait l'originalité de la méthode *Continuous Integration Unified Process*.

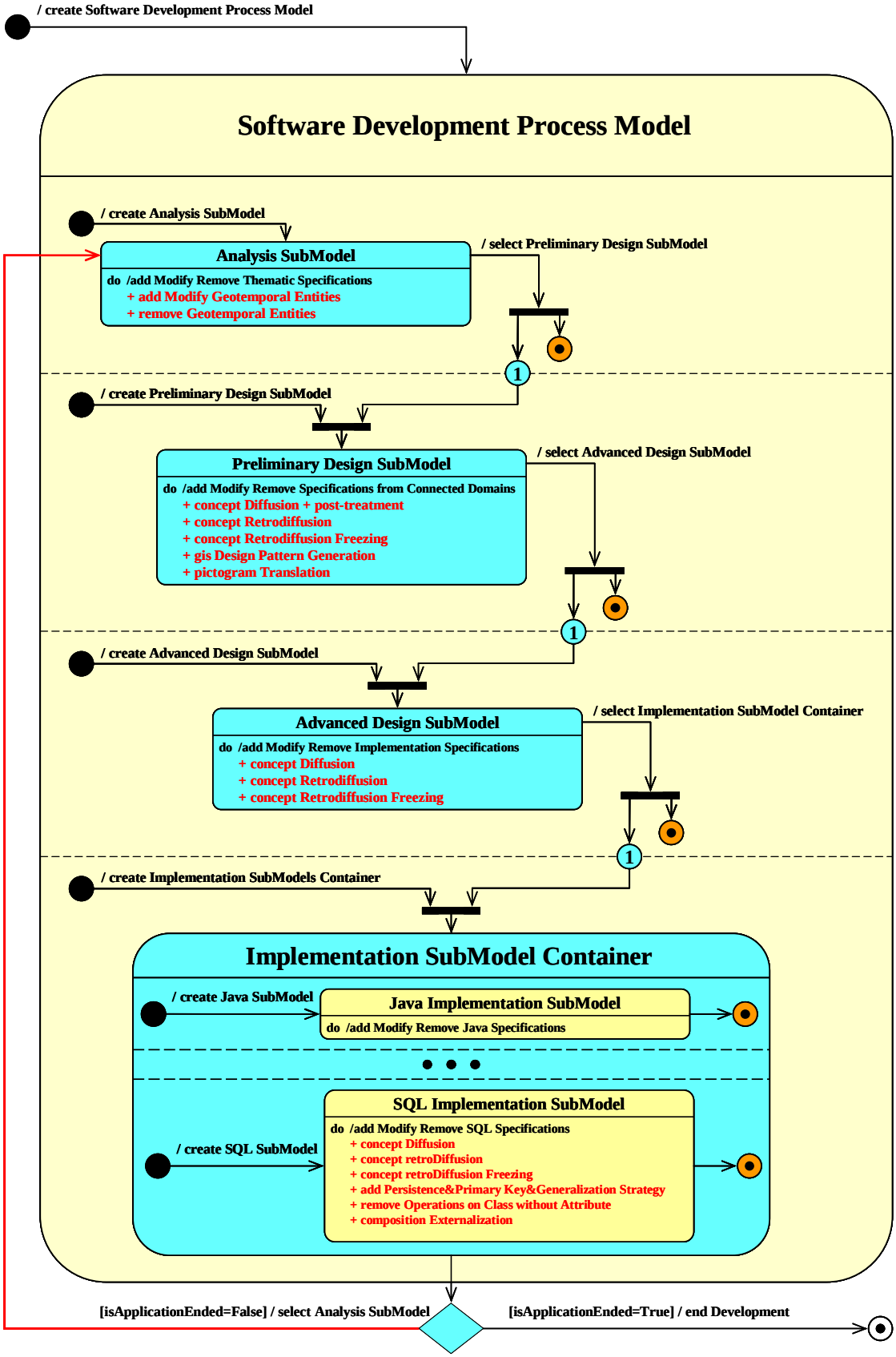


Figure 70 Les activités de modélisation dans l'artefact *Software Development Process Model*.

Chapitre 6. Implémentation de l'artefact Software Development Process Model et des transformations de diffusion de concepts, géomatiques et SQL

Avertissement : L'implémentation des transformations géomatiques faisant référence à plusieurs reprises à l'application d'affectation **<<** entre l'ensemble ***E*** des entités référencées et l'ensemble ***U*** des concepts SIG (cf. Chapitre 7), il est conseillé de s'approprier ce concept avant la lecture du présent chapitre.

L'artefact *Software Development Process Model* permettant la *capitalisation des connaissances* a été conçu et défini au Chapitre 3. La conception des neuf transformations nécessaires à la réalisation d'un *outil d'aide à la conception de Systèmes d'Information Géographique* suivant un *processus de développement permettant le prototypage rapide en séance d'analyse* a été décrite au Chapitre 5.

L'ensemble des concepts et des principes étant posés, la réalisation d'un *outil d'aide à la conception de Systèmes d'Information Géographique* passe par l'implémentation proprement dite dans l'atelier de génie logiciel. C'est l'objet de ce chapitre.

Le paragraphe I décrit l'implémentation de l'artefact *Software Development Process Model* dans l'atelier de génie logiciel.

Les paragraphes II, III et IV présentent les transformations qui automatisent l'évolution des modèles dans un processus full MDA c'est-à-dire depuis l'analyse jusqu'à l'implémentation.

La conclusion (V) rappelle les points essentiels de l'implémentation de l'artefact *Software Development Process Model* et des transformations dans un environnement ergonomique.

Sommaire détaillé

I REIFICATION DE L'ARTEFACT SOFTWARE DEVELOPMENT PROCESS MODEL DANS L'ATELIER DE GENIE LOGICIEL OBJECTEERING

En 2000, Pierre-Alain Muller écrivait d'une part *« un modèle est un paquetage particulier qui permet d'organiser de manière hiérarchique les éléments d'un système selon un point de vue particulier »* et, d'autre part, *« un modèle est une abstraction d'un système, représenté par une arborescence de paquetages »* (Muller *et al.*, 2000). Ces deux extraits montrent que, pour Pierre-Alain Muller, le concept support d'un modèle est le concept UML de *Paquetage* qui permet de structurer un modèle en sous-systèmes.

Les propriétés du concept paquetage correspondent exactement au besoin d'instanciation et de structuration de l'artefact *Software Development Process Model*. En effet, ce dernier est, selon notre vision, le modèle de l'application (cf. Chapitre 3-IV). Cet artefact est constitué de sous-modèles, chacun d'eux représentant une phase du cycle de développement.

Tout naturellement, l'artefact *Software Development Process Model* à été réifié en paquetage et sous-paquetages comme le montre la figure 71.

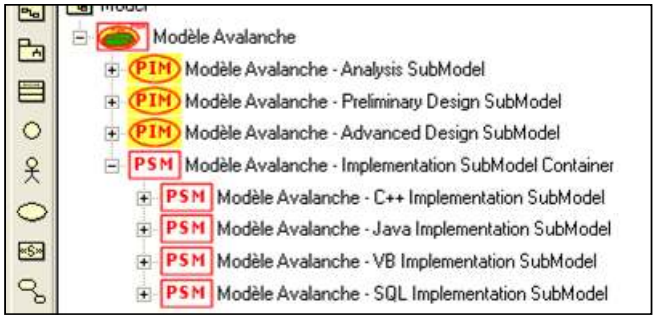


Figure 71 Exemple d'instance de l'artefact *Software Development Process Model*.

Dans cet exemple, l'artefact *Software Development Process Model* est instancié par le paquetage *Modèle Avalanche*, paquetage annoté du stéréotype *<<MDA : Software Development Process Model>>* associé au pictogramme .

Les sous-modèles d'analyse, de conception préliminaire et de conception avancée sont matérialisés par des paquetages fils nommés *Modèle Avalanche - Analysis SubModel*, *Modèle Avalanche - Preliminary Design SubModel* et *Modèle Avalanche - Advanced Design SubModel*. Le statut PIM des paquetages fils est indiqué par le pictogramme PIM attaché au stéréotype *<<MDA : PIM SubModel>>*.

Le container des sous-modèles d'implémentation, appelé *Modèle Avalanche - Implementation SubModel Container*, est aussi un sous-paquetage fils. Puisqu'il regroupe tous les modèles associés à la phase d'implémentation, nous lui avons affecté le statut PSM (stéréotype *<<MDA : PSM SubModel>>*+pictogramme PSM). Dans la phase de faisabilité actuelle, le container peut recevoir jusqu'à quatre sous-modèles d'implémentation correspondant aux quatre générateurs de codes de l'atelier de génie logiciel Objecteering.

Ces sous-modèles d'implémentations sont réifiés en sous-paquetages du container et sont tous stéréotypés *<<MDA : PSM SubModel>>* ce qui facilite leur identification grâce au pictogramme PSM.

Dans notre implémentation, le nom des sous-modèles est obtenu en concaténant le nom du *Software Development Process Model*, *Modèle Avalanche*, et le nom de la phase de développement associée au sous-modèle.

Dans le souci de pérenniser la structure de l'artefact *Software Development Process Model* et afin de garantir sa « pureté » au cours du développement, un certain nombre de fonctionnalités ont été implémentées afin d'assister le concepteur dans la création de cet artefact et dans la vérification de sa cohérence et de sa conformité :

- L'artefact *Software Development Process Model* et sa structuration en sous-paquetages ne peut être créé que si le paquetage racine est stéréotypé *<<MDA : Model>>*.
- Des fonctionnalités permettent de créer pas à pas les sous-modèles les uns après les autres dans cet ordre d'utilisation du cycle de développement.

⇨ Une autre fonctionnalité teste la cohérence de cet artefact (nom, stéréotype et structure des sous-paquetages,) avant toute *Transformation de diffusion*, géomatique ou SQL. En cas de non conformité, le concepteur est informé par un message et la transformation est annulée. De plus, cette fonctionnalité vérifie aussi si des éléments de modélisation (classe, notes, diagrammes, etc.) autres que les paquetages de l'artefact sont présents dans le paquetage racine ou dans le paquetage du container. Dans l'affirmative, un message signale l'existence de ces éléments de modélisation « parasites » au concepteur et l'exécution de la transformation est annihilée.

II TRANSFORMATIONS DE GESTION DE L'ARTEFACT SOFTWARE DEVELOPMENT PROCESS MODEL

Les transformations de diffusion et de rétrodiffusion sont, une fois les principes de conception acquis (cf. Chapitre 5-II), très simples et ne posent aucun problème de développement.

La seule difficulté rencontrée est due aux éléments de modélisation instances des métaclasses *Note*, *ViewBox* et *ViewLink* car il est syntaxiquement impossible de créer des liens de traçabilité. La diffusion des concepts décrite au paragraphe II.1 du Chapitre 5 est impossible. La diffusion et le clonage sont assurés mais la solution adoptée est beaucoup moins satisfaisante et moins fiable.

Les métaclasses *Note* et *ModelElement* héritent de la métaclasse *Element*. Étant donné que le concept *Dependency* dédié à la traçabilité (Softeam, 2003a) a des méta-associations avec la métaclasse *ModelElement* et aucune méta-associations avec la métaclasse *Note*, il est syntaxiquement impossible d'établir des liens de traçabilité entre deux instances de la métaclasse *Note*.

La métaclasse *ViewElement*, métaclasse mère à *ViewBox* et *ViewLink*, est une métaclasse qui hérite directement de la métaclasse *Objet* racine du métamodèle d'Objecteering. Les métaclasses *ViewBox* et *ViewLink* ne peuvent donc pas bénéficier de la propriété *Dependency* qui permet de réaliser la traçabilité entre concepts.

Les figures 72 et 73 montrent deux exemples d'entité géographique, *Site d'Avalanche* et *Avalanche*, appartenant au sous-modèle d'analyse[64] et ayant un clone dans le sous-modèle de conception préliminaire. Ces liens sont facilement identifiables par le biais du symbole <--> et du pictogramme de clonage .

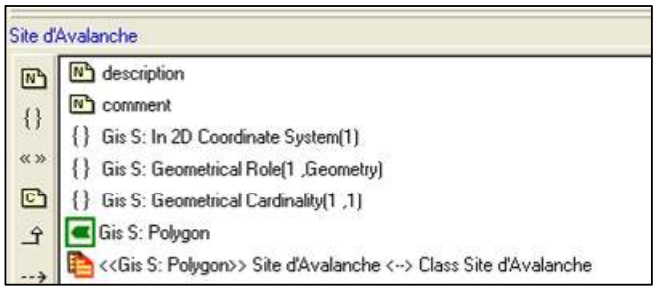


Figure 72 Exemple d'entité référencée à spatialité « Polygone » ayant un lien de traçabilité de clonage.

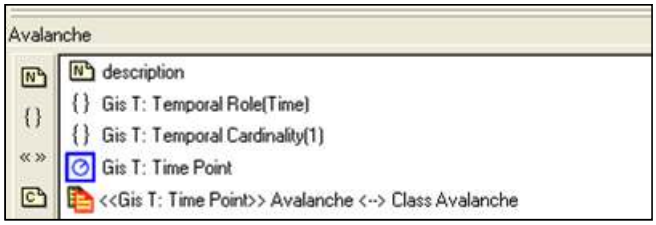


Figure 73 Exemple d'entité référencée à temporalité « Instant » ayant un lien de traçabilité de clonage.

[64] Déduit de la présence des stéréotypes *<<Gis S : Polygon>>* et *<<Gis T : Time Point>>* et des valeurs marquées spatiales et temporelles associées au stéréotype.

Le pictogramme de clonage est associé au stéréotype <<*MDA : Clone Traceability*>> ce qui permet d'identifierau premier coup d'œil les liens de traçabilité de clonage des autres liens de traçabilité (liens de traçabilité de traduction, liens des transformations du SQLDesigner, etc.).

Les concepts *Site d'Avalanche* et *Avalanche* à gauche du symbole <--> sont ceux du sous-modèle d'analyse car ils sont stéréotypés respectivement <<*Gis S : Polygon*>> et <<*Gis T : Time Point*>>. Leurs clones du sous-modèle de conception préliminaire sont à droite de ce symbole (*Class Site Avalanche* et *Class Avalanche*).

III TRANSFORMATIONS GEOMATIQUES

III.1 Transformations de saisie et de suppression de la pictogrammie : Interfaces de saisie

III.1.1 *Transformation de saisie de la pictogrammie*

La Transformation de saisie de la pictogrammie des entités référencées $T_{Sai.\,Pict.}^{An\ SM \rightarrow An\ SM}$ ***étant une transformation spécifique au sous-modèle d'analyse, les interfaces de saisie des concepts SIG ne doivent être proposées au concepteur que lorsque le sous-modèle d'analyse est sélectionné.***

Pour ce faire, la fonctionnalité d'affichage vérifie d'une part que le paquetage sélectionné (cf. figure 74) soit annoté du stéréotype <<*MDA : Analysis SubModel*>> (PIM) et, d'autre part, que le nom du paquetage soit conforme à la règle : nom de l'artefact *Software Development Process Model* (Modèle Avalanche) + « - *Analysis SubModel* ». Si ces deux conditions sont remplies, alors l'atelier de génie logiciel propose une interface de saisie.

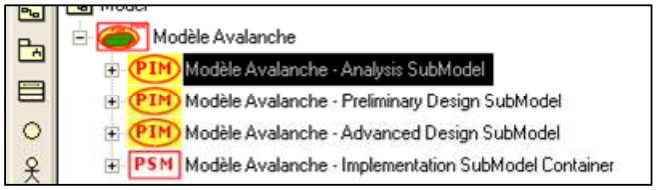


Figure 74 Exemple de sous-modèle d'analyse sélectionné.

Au vu du nombre de spatialités et de temporalités implémentées dans le *Profil UML-SIG*, une interface de saisie par type de spatialités ou de temporalités a été créée. La sélection du type d'interface est effectuée par les onglets. La figure 75 en montre quelques exemples.

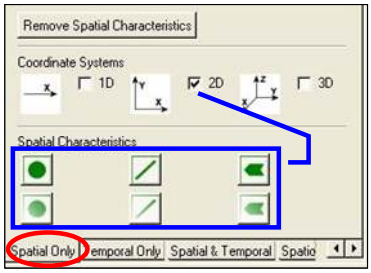
Cette figure présente les interfaces affichées lors la saisie des concepts de spatialité primitive (cf. Interfaces A et D) ou de temporalité primitive (cf. Interfaces B et E). Elle présente également les interfaces proposées au concepteur pour saisir les concepts de spatialité conjointe (cf. Interface C) ou de spatialité alternative (cf. Interface F).

Les interfaces A, B et C correspondent à une instanciation en généralisation de l'application d'affectation << alors que les interfaces D, E et F ne s'affichent que pour une instanciation en association. L'instanciation en généralisation ne nécessite pas de champs de saisie supplémentaire, les interfaces A, B et C ne sont pas munies de champs de saisie supplémentaires alors que les interfaces D, E et F en possèdent puisque l'instanciation en association de l'application d'affectation << impose que les informations de rôle, de cardinalité et de contrainte soient renseignées.

Enfin, les interfaces A et C illustrent l'adaptation au système de coordonnées. Dans un système de coordonnées 2D, les seules spatialités primitives autorisées sont les concepts *Point*, *Ligne* et *Polygone* ainsi que leurs homologues flous (cf. Interface A) puisque les concepts de spatialité volumiques ne peuvent pas exister dans un système de coordonnées 2D. Par contre dans un système de coordonnées 3D, le concepteur peut dispose des concepts de *Volume* et de *Volume flou* (cf. Interface C).

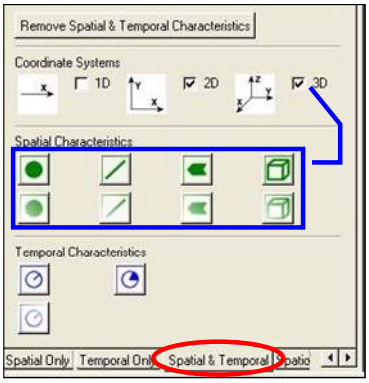
Ces interfaces ont été conçues et implémentées pour qu'elles s'adaptent interactivement au cours de la saisie selon les annotations de spatialité et/ou temporalité de l'entité référencée et la configuration du *Profil UML-SIG*.

Interface A

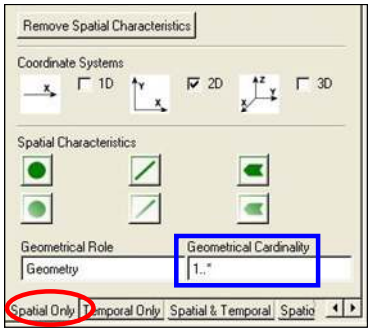


Interface B

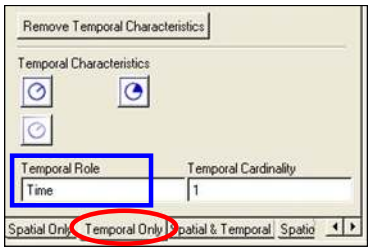
Interface C



Interface D



Interface E



Interface F

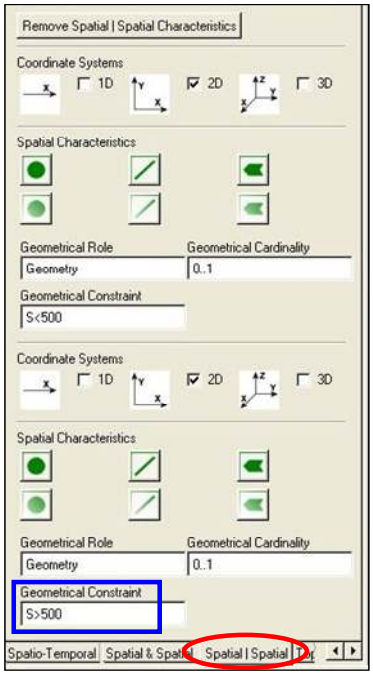


Figure 75 Exemples d'interfaces de saisie suivant une instanciation en généralisation colonne de gauche ou en association colonne de droite.

III.1.2 *Transformation de suppression de la pictogrammie*

L'implémentation de cette transformation ne pose aucune difficulté. Son exécution est déclenchée lorsque le concepteur clique sur le bouton situé au dessus de zone de saisie du système de coordonnées portant le nom de la

typologie de la spatialité ou temporalité (Spatial, Spatial&Spatial, etc.) préfixé par « Remove » et suivi du texte « Characteristics ».

III.2 La Transformation de génération du patron de conception SIG

L'implémentation de la *Transformation de génération du patron de conception SIG* $T_{Pat.\,SIG}^{PD\ SM \rightarrow PD\ SM}$ ne présente aucune difficulté majeure en soi. Le seul problème réside dans son intégration dans le processus de développement itératif. En effet, le *Patron de conception SIG* ne doit être généré qu'une et une seule fois. Un simple test d'existence résout ce problème.

Soucieux de prendre en compte le principe du développement centré sur l'architecture préconisé par la méthode *Continuous Integration Unified Process*, le *Patron de conception SIG* de la figure 48 a été structuré en trois niveaux de paquetages :

- Les éléments de modélisation des patrons de conception spatiaux 1D, 2D, et 3D sont générés dans trois paquetages, nommés respectivement *1D Coordinate system*, *2D Coordinate system* et *3D Coordinate system.*
- Ces trois patrons élémentaires sont regroupés dans un unique paquetage désigné *Spatial Design Patterns*, stéréotypé *<<Gis: Spatial>>* et portant le pictogramme S.
- Les éléments de modélisation du patron de conception temporel sont quant à eux regroupés au sein d'un paquetage portant le nom de *Temporel Design Pattern*. Ce dernier a été décoré du stéréotype *<<Gis: Temporal>>* associé au pictogramme T.
- Enfin, les paquetages *Spatial Design Patterns* et *Temporel Design Pattern* ainsi que la classe Concept SIG Primitif sont inclus dans le paquetage *GIS Design Patterns*. Le couple *<<Gis: GIS>>*/GIS annote ce dernier paquetage global.

La figure 76 illustre cette architecture en trois niveaux. Les couples stéréotype/pictogramme ont été volontairement introduits d'une part, pour faciliter l'indentification visuelle des paquetages SIG dans l'artefact *Software Development Process Model* et, d'autre part, pour piloter plus aisément les transformations ou valider la structure de cet artefact.

Par exemple, le test d'existence limitant à une seule fois la génération du *Patron de conception SIG* vérifie la présence d'un paquetage portant le nom *Spatial Design Patterns* et stéréotypé *<<Gis: GIS>>*.

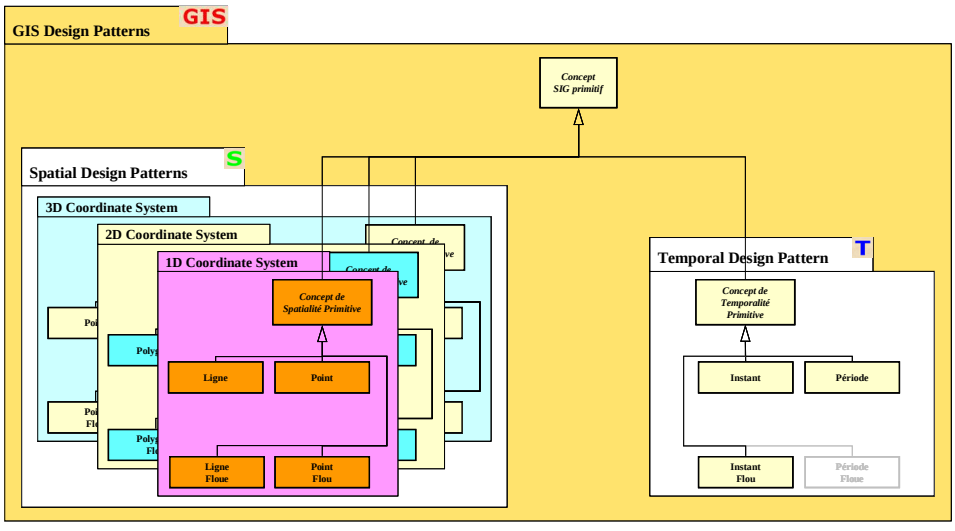


Figure 76 L'architecture générale du *Patron de conception SIG*[65].

[65] Le modèle global du Patron de Conception SIG a été représenté au lieu du patron complet afin d'alléger le dessin.

III.3 Transformation de traduction des pictogrammes et post-traitement à la transformation de diffusion

La difficulté majeure de la *Transformation de traduction des pictogrammes* est son insertion dans le processus itératif de la méthode *Continuous Integration Unified Process*. Ce problème a été abordé et résolu par le post-traitement à la *Transformation de diffusion*.

III.4 Interfaces de contrôle des transformations géomatiques

III.4.1 *Configuration des transformations géomatiques : Panneaux de Configuration*

L'atelier de génie logiciel Objecteering offre la possibilité d'attacher à un profil des panneaux permettant de configurer des paramètres de travail d'un projet. Les figures 77 et 78 présentent les deux panneaux de configuration du *Profil UML-SIG*. Le premier permet de choisir entre les deux types d'instanciation de l'application d'affectation **<<** et le second propose de choisir la dimension par défaut du système de coordonnées.

Avec la configuration choisie en figure 77, la *Transformation de traduction des pictogrammes* $T_{Tra.\,Picto.}^{PD\ SM \rightarrow PD\ SM}$ convertira les couples stéréotype/pictogramme en relation d'association. L'atelier de génie logiciel affichera alors des interfaces de type D, E ou F.

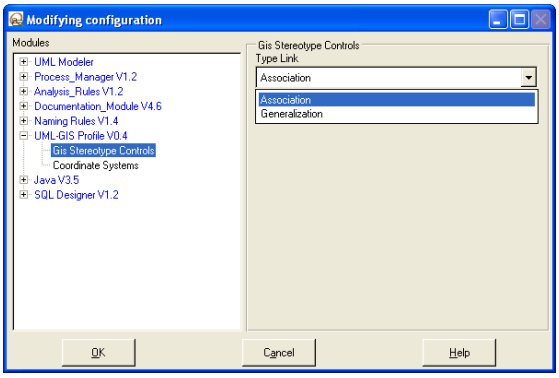


Figure 77 Panneau de configuration du type d'instanciation l'application d'affectation **<<**.

Dans le panneau de configuration de la figure 78, la dimension du système de coordonnées est 2D. Aussi lors de la saisie de la spatialité, les entités référencées seront annotées par défaut avec une valeur marquée de dimension 2D. Le concepteur a ensuite tout loisir d'adapter ce choix par défaut aux spécifications de chacune des entités référencées au travers des interfaces de saisie (cf. figure 75).

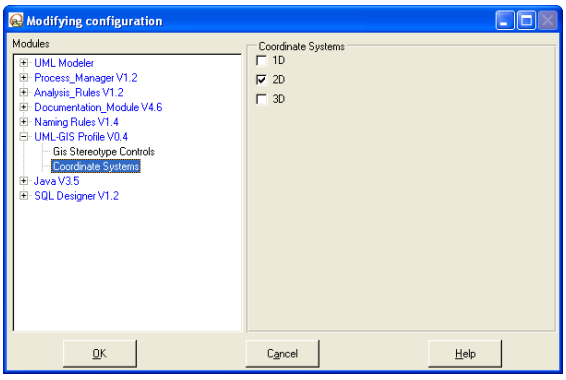


Figure 78 Panneau de configuration de la dimension par défaut du système de coordonnées.

III.4.2 *Interfaces d'exécution des transformations géomatiques*

Les transformations de Génération du Patron de Conception SIG et de Traduction des Pictogrammes sont propres au sous-modèle de conception préliminaire, aussi ces interfaces ne s'affichent que lorsque ce sous-modèle est sélectionné.

Si le concepteur clique sur ce sous-modèle (cf. figure 79) alors l'atelier de génie logiciel présente au concepteur l'une des deux interfaces de la figure 80.

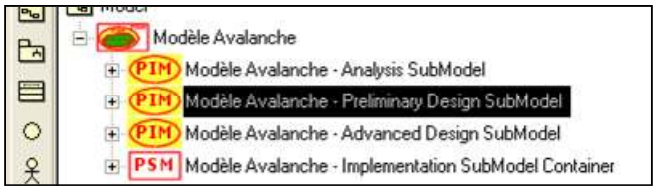
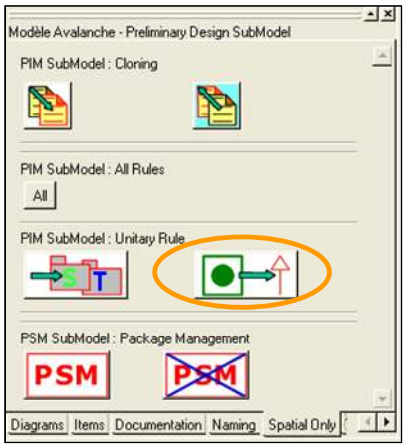


Figure 79 Exemple de sous-modèle de conception préliminaire sélectionné.

L'interface de gauche est proposée si le concepteur a adopté une instanciation en généralisation de l'application d'affectation **<<**. L'interface de droite s'affiche s'il choisit une instanciation en association.

Interface d'instanciation en Généralisation



Interface d'instanciation en Association

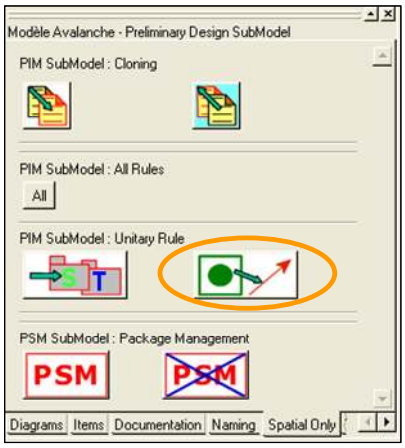


Figure 80 Interfaces d'exécution des transformations géomatiques suivant une instanciation en généralisation (à gauche) ou en association (à droite).

L'exécution de la *Transformation de génération du patron de conception SIG* est déclenchée en cliquant sur le bouton portant le pictogramme  et celle de Traduction des Pictogrammes lorsque le concepteur presse le bouton décoré du pictogramme  ou  . Enfin, le bouton portant l'inscription *All* effectue les transformations les deux transformations l'une après l'autre, la génération du *Patron de conception SIG* en premier lieu suivi de la traduction des pictogrammes.

IV TRANSFORMATIONS SQL

IV.1 Transformation de spécification multiple

La seule difficulté de cette transformation est sa conception dans le contexte itératif de la méthode *Continuous Integration Unified Process*.

En effet, à chaque itération une transformation simple aurait tendance à introduire dans la classe une valeur marquée de persistence, une clé primaire et une valeur marquée de généralisation. Le processus itératif « pollue »

progressivement le modèle en multipliant la même information[66]. Pour éviter cet inconvénient, il est nécessaire de mettre en place une stratégie de tests vérifiant au préalable de l'existence ou non ces spécifications SQL. Si elles n'existent pas, elles sont ajoutées sinon aucun ajout ou modification n'est effectué.

Ces tests de présence permettent de satisfaire au troisième principe qui préside notre démarche de conception du *Profil UML-SIG* (cf. Chapitre 5-I). Que la *Transformation de spécification multiple* soit exécutée au cours du cycle principal ou du cycle de prototypage rapide, elle ne doit en aucun cas détruire des modifications manuelles du concepteur car ce dernier est en droit de faire un tout autre choix technique que celui implémenté par défaut dans la transformation $T_{Spec.\,Mult.}^{SQL\ SM \rightarrow SQL\ SM}$.

IV.2 Transformation de suppression des opérations

La *Transformation de suppression des opérations* $T_{Supp.\,Opé.}^{SQL\ SM \rightarrow SQL\ SM}$ ne pose aucun problème d'implémentation particulier. Dans le processus itératif de la méthode Contiuous Integration Unnified Process, la *Transformation de diffusion* recréera à chaque itération les opérations *calcul Périmètre()* et *calcul Surface()* de la classe *Polygone* du sous-modèle d'implémentation SQL (cf. figure 66). Comme ces deux opérations seront systématiquement supprimées par la transformation de suppression qui lui succédera, la génération de code s'effectuera correctement.

IV.3 Transformation d'externalisation des relations de composition

Lors de l'implémentation, le seul point délicat à prendre en compte dans le processus itératif est de vérifier si la valeur marquée *{External}* existe déjà. Si elle est présente, il n'est pas nécessaire d'en ajouter une seconde puisque cela entraînerait une « pollution » du modèle qui dégraderait sa qualité.

IV.4 Interface d'exécution des transformations SQL

Les transformations SQL décrites ici sont spécifiques au sous-modèle d'implémentation SQL aussi elles ne doivent être appliquées que sur ce sous-modèle et en aucun cas sur les autres sous-modèles.

Pour satisfaire cette règle, la solution la plus opérationnelle a été de n'afficher l'interface des commandes des transformations SQL (cf. figure 81) que si le sous-modèle d'implémentation SQL est sélectionné.

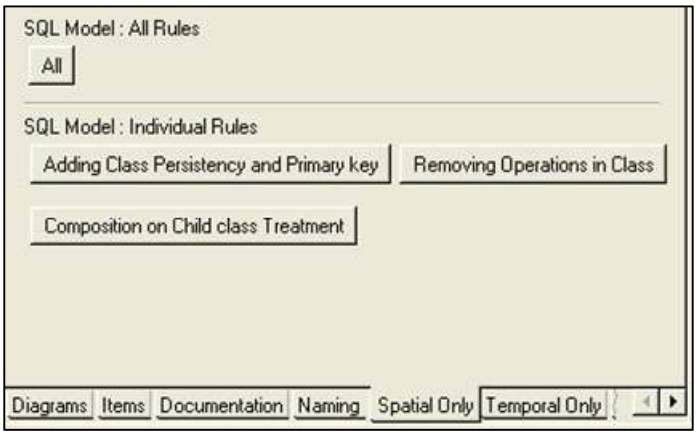


Figure 81 Interface des Transformations SQL.

Dans l'artefact *Software Development Process Model*, le sous-modèle d'implémentation SQL est parfaitement identifiable par le couple stéréotype/extension du nom du paquetage. Il suffit alors de vérifier si le

[66] Dans notre atelier de génie logiciel et si le contrôle de cohérence est activé, la présence de deux attributs (concepts UML support des clés primaires) de même nom dans une classe provoquerait un message d'erreur évitant par là la multiplication d'information. L'ennui est que la transformation est aussi annulée. Par contre, si le contrôle de cohérence est désactivé, rien n'interdit la redondance de la même information.

paquetage sélectionné possède un stéréotype <<*MDA : PSM SubModel*>> (PSM) et si son nom contient l'extension « - *SQL Implementation SubModel* ». La figure 82 montre un exemple.

Via l'interface de la figure 81, le concepteur a la possibilité d'exécuter les transformations de Spécification Multiple, de Suppression des Opérations et d'externalisation des relations de composition en cliquant sur les boutons *Adding Class Persistency and Primary key*, *Removing Operation in Class* ou *Compositon on Child class Treatment*. Le bouton portant l'inscription *All* déclenche l'exécution des trois transformations dans l'ordre énuméré.

Figure 82 Exemple de sous-modèle d'implémentation SQL sélectionné.

V CONCLUSION

Dans un atelier de génie logiciel tel qu'Objecteering doté d'un langage de métaprogrammation puissant, l'implémentation d'une part, des fonctionnalités facilitant la construction de l'artefact *Software Development Process Model*, la vérification de sa cohérence et, d'autre part, des neuf transformations qui permettent d'adopter un processus de développement full MDA est très facile et ne nécessite pas de développements prohibitifs.

Le *Software Development Process Model* et ses sous-modèles ont été réifiés en une structure de paquetage racine et de sous-paquetages qui présente l'avantage de respecter le principe du processus de développement centré sur l'architecture de la méthode Processus Unifié (Chapitre 2-II.1.4).

La difficulté majeure rencontrée lors de l'implémentation des transformations est leur intégration dans le processus itératif de la méthode *Continuous Integration Unified Process*.

Grâce à l'utilisation systématique d'interfaces interactives, l'atelier de génie logiciel Objecteering offre un environnement ergonomique pour la conception et le développement de *Systèmes d'Information Géographique*.

Il faut signaler que la conception de certaines interfaces peut être un moyen de capitalisation de la connaissance d'un domaine. Les interfaces de saisie de la pictogrammie sont un exemple type.

L'affichage des concepts (flou et non flou) de spatialité *Polygone* et *Volume* en fonction de la dimension du système de coordonnées est une connaissance du domaine de l'*Information Géographique* qui, une fois formalisée et implémentée, évite des saisies aberrantes.

Avec l'implémentation, l'objectif général de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique* adapté à un *processus de développement permettant le prototypage rapide en séance d'analyse* assurant la *capitalisation des connaissances* est atteint.

Partie C – Contribution à la Modélisation des Systèmes d'Information Géographique

Chapitre 7. Dérivation d'un Métamodèle SIG de l'étude terminologique des propriétés spatiales et temporelles - Mise en œuvre de la Théorie des ensembles

Confronté à une multitude de méthodes, de formalismes et de normes (cf. Chapitre 1-I) employant des vocables différents souvent pour désigner le même concept de spatialité ou de temporalité, le premier objectif de ce chapitre était de s'approprier ces concepts. Le second objectif était de se doter d'un instrument de comparaison des méthodes et des formalismes.

Afin de réaliser ce double objectif, une étude terminologique du langage pictogrammique de l'atelier de génie logiciel Perceptory a été réalisée car ce langage pictogrammique nous a semblé le plus « riche » pour exprimer la spatialité et/ou la temporalité des entités référencées. Cette étude terminologique a été effectuée en s'appuyant sur la Théorie des ensembles.

Le paragraphe I précise les conventions de notation adoptées dans cette annexe. Le paragraphe II propose des définitions générales des concepts manipulés et les définitions des ensembles regroupant ces concepts.

Le paragraphe III contient les définitions des ensembles relatifs aux spatialités primitives et temporalités primitives qui sont les plus courantes. Ces concepts sont les concepts de base qui entrent dans la combinaison des spatialités composites et des temporalités composites.

Le paragraphe IV définit les lois de composition des spatialités composites et des temporalités composites ainsi que les définitions des ensembles regroupant les concepts composites, les pictogrammes composites et les entités référencées composites.

Le paragraphe V introduit les spatialités dérivées et les temporalités dérivées et le paragraphe VI la spatialité inconnue et la temporalité inconnue. Les ensembles associés à ces nouvelles typologies de spatialités et de temporalités sont définis dans ces paragraphes.

L'approche ensembliste permet de construire des taxinomies terminologiques qui s'avèrent être une représentation ensembliste de la syntaxe entre les concepts manipulés. Un métamodèle du langage pictogrammique a pu être dérivé de ces trois taxinomies. Cette phase de dérivation est décrite au paragraphe VII.

Le paragraphe VIII décrit comment certaines typologiques de spatialités ou de temporalités peuvent être exploitées pour assister le concepteur dans le processus de modélisation.

Le paragraphe IX résume les principaux résultats du présent chapitre.

Sommaire détaillé

I CONVENTIONS DE NOTATION

I.1 Sur les concepts

Quel que soit le domaine modélisé, les acteurs communiquent avec le vocabulaire qu'ils ont l'habitude d'utiliser quotidiennement pour échanger de l'information avec le concepteur et nommer les *concepts* du domaine en cours de modélisation (cf. définition 74). Les *concepts* du domaine en cours de modélisation appartiennent à un ensemble plus vaste qui est l'ensemble des concepts connus ou non. Nous adoptons les conventions suivantes pour noter d'une part, un élément de cet ensemble et, d'autre part, l'ensemble lui-même :

Convention La variable représentant un ***concept*** (cf. définition 74) sera notée $\boldsymbol{c}$.

Convention L'***ensemble des concepts*** $\boldsymbol{c}$ sera noté $\boldsymbol{C}$.

I.2 Sur la terminologie

Les notations de spatialité (cf. définition 1) et de temporalité (cf. définition 2) sont assujetties à la convention :

Convention La variable $\boldsymbol{s}$ sera réservée à la notation de la ***spatialité*** d'un concept thématique et la variable $\boldsymbol{t}$ sera quant à elle destinée à la notation de la ***temporalité***.

I.3 Sur les pictogrammes

Afin de préserver la dualité qui existe entre les concepts de spatialité ou de temporalité et les pictogrammes associés exprimant graphiquement les propriétés de spatialité et de temporalité d'un modèle, la convention suivante a été adoptée :

Convention La variable désignant un pictogrammique sera surmontée de l'abréviation $\boldsymbol{vl}$ (*Visual Language*).

La convention énoncée ci-dessus s'applique à tout type de variable, aussi bien aux éléments d'un ensemble qu'à l'ensemble lui-même. Par exemple, le pictogramme d'un concept $\boldsymbol{c}$ sera noté $\overset{vl}{\boldsymbol{c}}$ et l'ensemble des pictogrammes dual à $\boldsymbol{C}$ sera noté $\overset{vl}{\boldsymbol{C}}$.

II SPATIALITE ET TEMPORALITE : DEFINITIONS GENERALES

II.1 Ensembles de spatialité générale et de temporalité générale

II.1.1 *Ensembles des concepts SIG*

Pour exprimer et renseigner la spatialité et la temporalité *des concepts thématiques,* les acteurs de domaine de l'*Information Géographique* font appel aux concepts *Point, Ligne, Géométrie alternative, Géométrie complexe*, etc. pour la spatialité et/ou aux concepts *Instant, Période*, etc. pour la temporalité. Ils utilisent parfois des concepts plus évolués tels que *Semis (de points), Point mobile, Polygone déformable, Échantillonnage ponctuel.*

Cette rapide énumération à laquelle il serait possible d'ajouter bien d'autres concepts se caractérise par le fait qu'il y a des concepts qui suscitent :

- ⇨ Une perception de spatialité ***pure*** sans aucune perception de temporalité : *Point, Ligne, Géométrie alternative*, Géométrie complexe, Semis (de points), Échantillonnage ponctuelle, etc.
- ⇨ Une perception de temporalité ***pure*** sans aucune perception spatialité : *Instant, Période, Série temporelle*, etc.
- ⇨ Une perception combinant simultanément une perception de spatialité et de temporalité : *Point mobile*.

Cette typologie des perceptions constitue un mode de classification des concepts de spatialité ou de temporalité utilisés en *Information Géographique* qui permet créer une première taxinomie.

Pour ce faire, nous adoptons le formalisme de la ***Théorie naïve des ensembles*** car il présente l'avantage de pouvoir définir un ensemble soit en énumérant un à un les éléments de cet ensemble soit en énonçant une propriété commune des éléments de l'ensemble. Le premier mode de définition en dit en ***extension*** et le second en ***compréhension*** (Amsili, 2005 ; El Methni, Non daté ; Fondation Wikimédia, 2001 ; Iovleff, 2004 ; Savard, Non daté ; Sciences.ch, 2005 ; Zarba, 2005).

Souhaitant dans un premier temps énoncer des définitions générales basées sur la perception, le mode en extension est impossible, seul le mode en compréhension est envisageable dans ce contexte. Pour ce faire, il faut poser des propriétés qui permettront de définir des ensembles souhaités. Nous avons donc formulé une première propriété autour de la spatialité et une seconde autour de la temporalité :

- ⇨ P_S ***= « est un concept qui suscite une perception de spatialité ».***
- ⇨ P_T ***= « est un concept qui suscite une perception de temporalité ».***

Les définitions de P_S et P_T ne préjugent pas de la complexité des concepts. En effet, ils peuvent résulter d'une combinaison de concepts élémentaires. Par exemple, un *Semis de points* ou un *Échantillonnage ponctuel* sont des concepts constitués d'une multitude de points définis suivant les besoins des acteurs. Un *Point mobile* sous-entend une série de points datés.

Munis de ces deux propriétés, il est possible de définir en ***compréhension*** les trois sous-ensembles correspondant aux trois perceptions listées ci-dessus.

Soit c un élément de C :

Définition 13 L'ensemble des ***concepts de spatialité (pure)*** S est l'ensemble des concepts c qui ont la propriété P_S , ***« est un concept qui suscite une perception de spatialité »***, et qui ne possèdent pas la propriété P_T , ***« est un concept qui suscite une perception de temporalité »***.

$$S = \left\{ c \in C \mid P_S(c) \wedge \overline{P_T(c)} \right\}$$

Définition 14 L'ensemble des ***concepts de temporalité (pure)*** T est l'ensemble des concepts c qui ont la propriété P_T , ***« est un concept qui suscite une perception de temporalité »***, et qui ne possèdent pas la propriété P_S , ***« est un concept qui suscite une perception de spatialité »***.

$$T = \left\{ c \in C \mid P_T(c) \wedge \overline{P_S(c)} \right\}$$

Nota Le qualificatif « pure » associé à spatialité ou temporalité a été mis entre parenthèse car il sera la plupart du temps omis sauf lorsque le contexte est ambigu.

Définition 15 L'ensemble des ***concepts de spatialité/temporalité*** ST est l'ensemble des concepts c qui possèdent simultanément les propriétés P_S , ***« est un concept qui suscite une perception de spatialité »***, et P_T , ***« est un concept qui suscite une perception de temporalité »***.

$$ST = \{ c \in C \mid P_S(c) \wedge P_T(c) \}$$

Les ensembles S, T et ST ainsi construits sont des sous-ensembles de C puisque, par construction, leurs éléments appartiennent à C :

$$S \subset C \qquad T \subset C \qquad ST \subset C$$

Les définitions 13, 14 et 15 sont une première étape dans la construction d'une taxinomie, appelée *Taxinomie SIG* qui a les propriétés suivantes :

- ⇨ L'ensemble S n'est composé que des seuls concepts de C caractérisant une spatialité.
- ⇨ L'ensemble T ne regroupe que les seuls concepts de C caractérisant une temporalité.
- ⇨ L'ensemble ST n'est constitué que des concepts de C mobilisant simultanément au moins une spatialité et une temporalité.

Il est possible d'en déduire que les intersections deux à deux de ces trois sous-ensembles de C sont vides. Les sous-ensembles S, T et ST sont donc disjoints :

$$S \cap T = \varnothing \qquad S \cap ST = \varnothing \qquad T \cap ST = \varnothing$$

Les trois ensembles S, T et ST regroupent *tous* les concepts de spatialité, de temporalité ainsi que toutes les combinaisons possibles de ces deux propriétés. Cela a conduit à définir l'ensemble regroupant tous les concepts suscitant une perception de spatialité et/ou de temporalité et à le nommer :

Définition 16 L'ensemble des ***concepts SIG*** U est l'union des trois ensembles S, T et ST.

$$U = S \cup T \cup ST$$

L'ensemble U est aussi sous-ensemble de C puisque tous les éléments de S, de T et de ST sont éléments de C :

$$U \subset C$$

Les définitions 13 à 16 créent la forme générale de la taxinomie SIG sur l'ensemble des concepts de spatialité et de temporalité. Celle-ci a été conçue de telle sorte que, quelque soit le concept de spatialité, de temporalité ou résultant d'une combinaison des deux, il appartient à un et un seul des trois sous-ensembles S, T et ST. Nous avons donc construit une partition $\mathscr{P}(U)$ de l'ensemble U puisque :

- ⇨ Les sous-ensembles S, T et ST ne sont pas vides (nous le verrons aux paragraphes III et IV).
- ⇨ Les sous-ensembles S, T et ST constituent un recouvrement[67] de U (cf. définition 16).
- ⇨ Les sous-ensembles S, T et ST sont disjoints deux à deux.

La figure 83 montre l'ensemble des concepts SIG U et la partition $\mathscr{P}(U)$ formée par les ensembles S, T et ST. L'ensemble ST a été déporté et dessiné à l'extérieur de l'ensemble C afin de bien faire apparaître qu'il est le complémentaire de l'union des ensembles de concepts de spatialité et temporalité ($S \cup T$) dans U :

$$ST = \complement_U (S \cup T)$$

L'ensemble ST a été conçu comme étant le complément à l'union des ensembles S et T car cette définition impose à un concept SIG de n'appartenir qu'à un et un seul de ces ensembles. ***Cette contrainte théorique présente comme avantage pratique d'obliger le chercheur à se poser la question : à quel ensemble appartient un nouveau concept SIG ?*** Cette question est sous-jacente tout au long du chapitre.

Les définitions générales des quatre ensembles S, T, ST et U constituent la première ébauche de la taxinomie des concepts SIG qui permettent de renseigner les propriétés spatiales et temporelles des entités

[67] La réunion des sous-ensembles A_i d'un ensemble A est égale à l'ensemble A lui-même. C'est-à-dire $\bigcup_{A_i \in \mathscr{P}(A)} A_i = A$.

référencées. Dans les deux paragraphes suivant, nous allons utiliser cet acquis pour définir une deuxième taxinomie sur l'ensemble des pictogrammes et une troisième sur l'ensemble des concepts thématiques constituant les entités référencées.

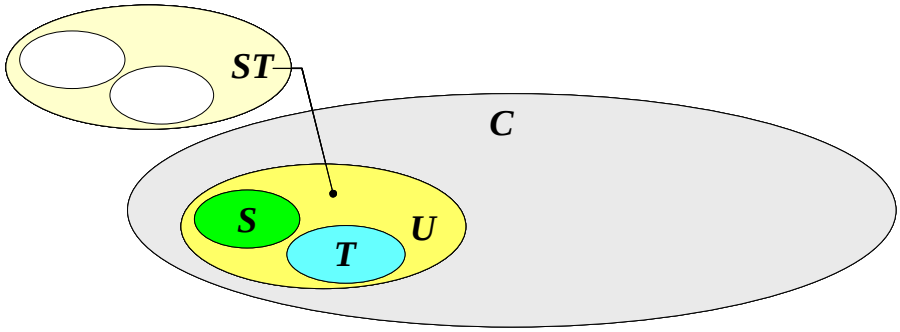


Figure 83 Représentation de la *Taxinomie SIG* et de la partition $\mathscr{P}(U)$.

II.1.2 *Ensembles des pictogrammes SIG*

Nombre de formalismes et de méthodes d'analyse destinées à la conception de *Systèmes d'Information Géographique* ont développés un langage pictogrammique afin de faciliter la communication entre le concepteur et les acteurs du domaine lors de l'analyse.

Pour ce faire, les auteurs des formalismes et méthodes ont associés un pictogramme à chaque concept SIG. Ainsi, le pictogramme[68] ⊡ est associé au concept *Point*, ⊠ à celui de *Ligne* et ◧ au concept de *Polygone*. De manière analogue, le pictogramme ◷ est jumelé au concept d'*Instant* et ◔ à celui de *Période*.

Fort de ce constat, il est possible d'énoncer un couple de propriétés dérivant des définitions P_S et P_T précédentes. Ce nouveau couple de propriétés va permettre de réaliser une classification similaire à celle des concepts SIG. La première est une propriété portant sur les pictogrammes de spatialité et la seconde sur les pictogrammes de temporalité :

⇨ $\overset{vl}{P_S}$ ***= « est un pictogramme associé à un et un seul concept de spatialité (pure) ».***

⇨ $\overset{vl}{P_T}$ ***= « est un pictogramme associé à un et un seul concept de temporalité (pure) ».***

En appliquant une démarche de construction analogue à celle des concepts SIG, il est possible de définir l'ensemble des ***pictogrammes de spatialité (pure)*** $\overset{vl}{S}$, celui des ***pictogrammes de temporalité (pure)*** $\overset{vl}{T}$, celui des ***pictogrammes de spatialité/temporalité*** $\overset{vl}{ST}$ et enfin l'ensemble des ***pictogrammes SIG*** $\overset{vl}{U}$. Ces quatre ensembles constituent la *Taxinomie des Pictogrammes*. Au sein de cette taxinomie, les ensembles $\overset{vl}{S}$, $\overset{vl}{T}$ et $\overset{vl}{ST}$ forment une partition $\mathscr{P}\left(\overset{vl}{U}\right)$ de l'ensemble des pictogrammes SIG $\overset{vl}{U}$. Cette dernière partition est semblable à $\mathscr{P}(U)$.

Étant donné que la taxinomie des Pictogrammes est calquée sur celles des concepts SIG, sa représentation est similaire (cf. figure 84). Comme précédemment, l'ensemble $\overset{vl}{ST}$ a été déporté pour montrer que c'est le complément de l'union des pictogrammes de spatialité pure et de temporalité pure dans $\overset{vl}{U}$:

$$\overset{vl}{ST} = \complement_{\overset{vl}{U}}\left(\overset{vl}{S} \cup \overset{vl}{T}\right)$$

[68] Les pictogrammes peuvent varier légèrement selon le formalisme, la méthode ou l'outil mais l'intention reste la-même.

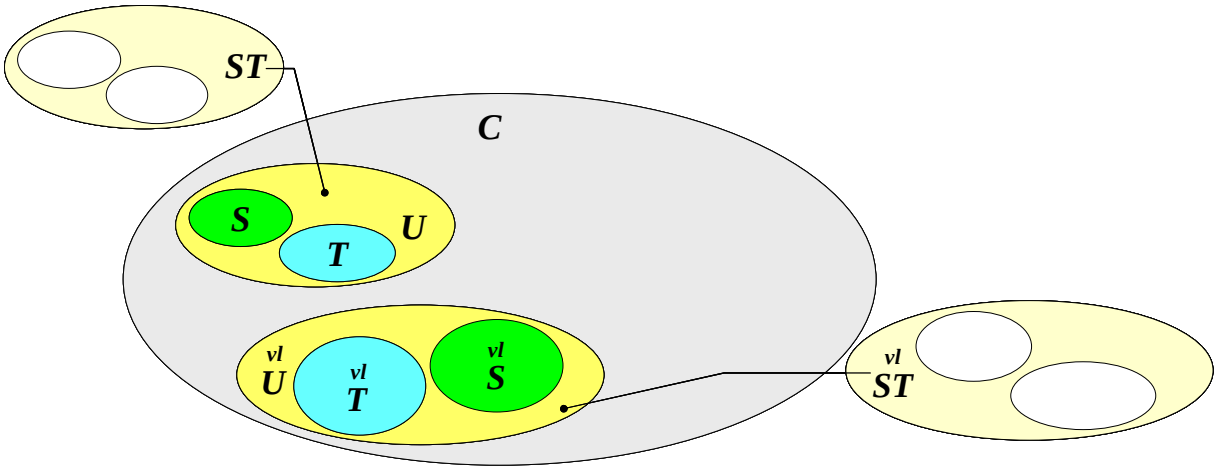


Figure 84 Représentation de la *Taxinomie des Pictogrammes* et de la partition $\mathscr{P}\left(\overset{vl}{U}\right)$.

II.1.3 *Ensembles des entités référencées*

Les concepts SIG étant définis (cf. II.1.1), la réécriture de la définition des entités référencées (cf. définition 3) dans le formalisme de la théorie des ensembles ne posent pas de problème majeur.

Toutefois, comme pour les concepts SIG, nous avons cherché à affiner les typologies des entités référencées afin de construire la *Taxinomie des entités référencées*. Aussi, fort des définitions précédentes, nous avons élaboré deux propriétés une première propriété relative à la spatialité et une seconde concernant la temporalité :

- ⇨ F_S = ***« a au moins un concept de spatialité (pure) ».***
- ⇨ F_T = ***« a au moins un concept de temporalité (pure) ».***

Comme pour les concepts SIG, les définitions de F_S et F_T ne préjugent pas de la complexité de la spatialité et de la temporalité des entités référencées. Nous verrons au paragraphe IV.3 qu'elles peuvent être combinées entre-elles pour former de nouvelles entités référencées plus « conséquentes ».

Par un processus calqué sur celui de construction de la taxinomie SIG, les définitions F_S et F_T sont utilisées pour définir l'ensemble des entités référencées ***à spatialité (pure)*** E_S, celui des entités référencées ***à temporalité (pure)*** E_T, celui des entités référencées ***à spatialité/temporalité*** E_{ST} et l'ensemble des entités référencées E regroupant les autres. Les trois premiers ensembles E_S, E_T et E_{ST} constituent une partition de E et par conséquent toute entité référencée *appartient à un et à un seul* de ces trois sous-ensembles.

Bien entendu, l'architecture des ensembles E, E_S, E_T et E_{ST} est similaire à celle des ensembles, U, S, T et ST (cf. figure 85).

La représentation en déport de l'ensemble E_{ST} a été effectuée pour les mêmes raisons que pour l'ensemble ST (cf. figure 83) c'est-à-dire pour bien montrer sa complémentarité dans E par rapport à l'union des ensembles de spatialité pure E_S et de temporalité pure E_T :

$$E_{ST} = \complement_E(E_S \cup E_T)$$

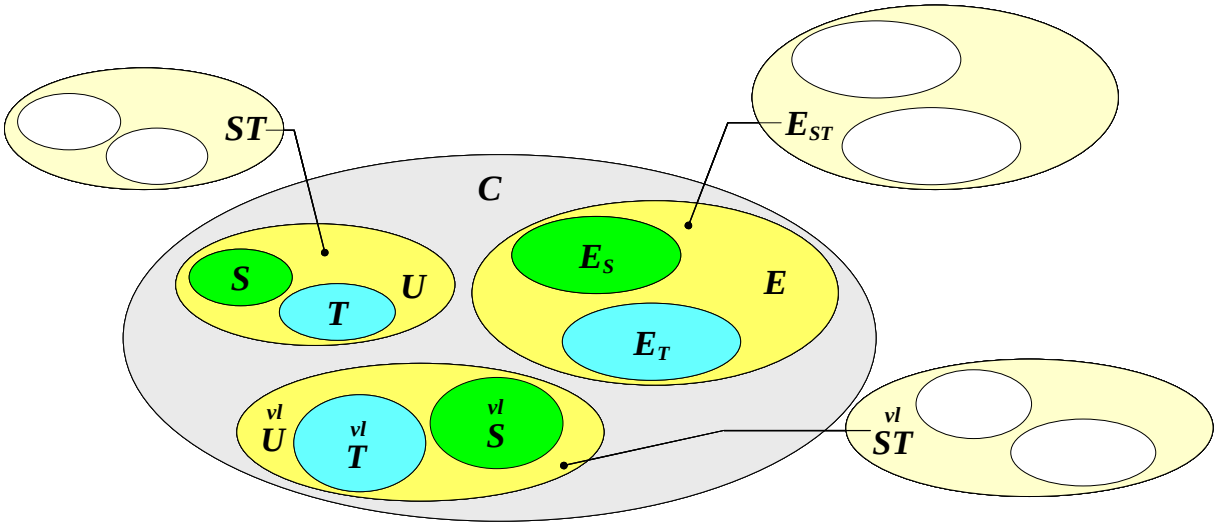


Figure 85 Représentation de la *Taxinomie des entités référencées* et de la partition $\mathscr{P}(E)$[69].

II.2 Relations entre ensembles

Les trois ensembles E, U et $\overset{vl}{U}$ ne sont pas indépendants puis que, d'après la définition 3, un concept thématique acquiert le statut d'entité référencée (appartient E) que si sa spatialité et/ou sa temporalité sont mentionnées. Bien évidemment, ces propriétés sont empruntées à l'ensemble des concepts SIG U.

La pictogrammie a été introduite par de nombreux auteurs afin de faciliter d'une part, la communication avec les acteurs du domaine et, d'autre part, la modélisation. Ces auteurs associent de façon biunivoque un concept SIG de U à un pictogramme de $\overset{vl}{U}$.

La dépendance apparaît aussi dans l'énoncé des 6 propriétés (P_S / P_T, $\overset{vl}{P_S} / \overset{vl}{P_T}$ et F_S / F_T) qui ont permis de les définir puisque des quatre dernières font référence aux concepts SIG définis par les deux premières.

Tout naturellement, nous avons établi des relations entre les trois ensembles E, U et $\overset{vl}{U}$ afin de mettre en correspondance soit des éléments soit des ensembles. Les définitions de ces relations sont explicitées ci-dessous.

II.2.1 *Application d'affectation << entre l'ensemble E des entités référencées et l'ensemble U des concepts SIG*

La définition 3 stipule explicitement qu'un concept thématique c est une entité référencée si et seulement si une spatialité ou une temporalité annote le concept. C'est la raison qui a conduit à définir les propriétés F_S et F_T par rapport aux concepts de spatialité pure et de temporalité pure préalablement définis.

Il y a donc bien une correspondance entre l'ensemble des entités référencées E et celui des concepts SIG U, d'où la définition :

Définition 17 Par définition des ensembles E et U, il existe une application surjective de E vers U, que l'on qualifiera d'affectation et notera $<<$, qui, à une entité référencée e, affecte sa propriété spatiale u.

$$<< : E \longrightarrow U$$
$$e \longrightarrow u$$

[69] Afin de faciliter la compréhension, les ensembles U et E sont représentés séparés mais nous verrons par la suite que ces ensembles ne sont pas disjoints.

Remarque :

La nature surjective de l'application d'affectation $<<$ est intéressante car nous verrons que le nombre de concepts SIG est relativement faible par contre chaque concept SIG peut annoter une infinité d'entités référencées constituant des sous-ensembles dans E. Donc, à chaque singleton associé à un concept SIG correspond un ensemble d'entités référencées :

$$E_{\{u\}} = <<^{-1}(\{u\}) = \{ e \in E \mid <<(e) = u \} \qquad \{u\} \subset U$$

L'intérêt majeur de pousser le raffinement de la taxinomie des entités référencées à l'extrême est d'une part, de préciser la terminologie des entités référencées jusqu'à sa granularité la plus fine et, d'autre part, de préparer la dérivation d'un métamodèle des trois taxinomies (cf. VII).

Le langage UML étant fondamentalement graphique, la complexité des diagrammes amènera à permuter la position des concepts SIG et des entités référencées. Afin que l'orientation des chevrons soit toujours cohérente d'un diagramme à l'autre, la convention graphique suivante a été adoptée :

Convention Les chevrons symbolisant l'application d'affection $<<$ seront toujours orientés vers les entités référencées.

II.2.2 *Fonction f entre l'ensemble U des concepts SIG et l'ensemble $\overset{vl}{U}$ des pictogrammes SIG*

Comme pour les entités référencées, les définitions des propriétés $\overset{vl}{P_S}$ et $\overset{vl}{P_T}$ sont énoncées par rapport aux concepts SIG. De plus, ces définitions stipulent explicitement l'unicité de la correspondance entre concept SIG et pictogramme. Il est donc possible de définir une fonction bijective de l'ensemble des concepts SIG U vers l'ensemble des pictogrammes SIG :

Définition 18 Par définition des ensembles U et $\overset{vl}{U}$, il existe une fonction bijective de l'ensemble des concepts SIG U vers l'ensemble des pictogrammes SIG $\overset{vl}{U}$, que l'on appellera f, qui, à un concept SIG, fait correspondre un et un seul pictogramme.

$$f : U \longrightarrow \overset{vl}{U}$$
$$u \longrightarrow \overset{vl}{u}$$

II.3 En résumé

Les définitions de 13, 14 et 15 relatives à la taxinomie SIG ont été énoncées en compréhension afin qu'elles soient très générales. Les taxinomies des pictogrammes et des entités référencées ont été créées suivant une démarche analogue à celle des concepts SIG.

Ces deux dernières taxinomies sont dépendantes de la taxinomie SIG puisque les propriétés qui ont permis de définir les ensembles en compréhension impliquent les concepts SIG de la première taxinomie. Cette dépendance a été formalisée en définissant d'une part, l'application d'affectation entre l'ensemble des concepts SIG et celui des entités référencées et, d'autre part, une fonction bijective f entre l'ensemble des concepts SIG et l'ensemble des pictogrammes SIG.

La création de partitions au sein des taxinomies a pour avantage de classer tout concept sans ambiguïté dans l'un des trois sous-ensembles de chacune des taxinomies.

Dans le reste du chapitre, les trois taxinomies vont être complétées et affinées afin de préciser la terminologie utilisée par la suite. Les connaissances des concepts SIG et celles des entités référencées vont être mobilisées pour atteindre ce but.

III SPATIALITE PRIMITIVE ET TEMPORALITE PRIMITIVE

Au paragraphe précédent, les ensembles relatifs aux concepts SIG, ceux propres aux pictogrammes et ceux concernant les entités référencées, ont été définis d'un point de vue général. Or, il existe un certain nombre de concepts SIG que les acteurs du domaine de l'*Information Géographique* rencontrent et utilisent de façon récurrente.

Dans ce paragraphe, nous allons nous intéresser à eux car ce sont des concepts de spatialité et temporalité élémentaires qui sont ensuite combinés pour composer des spatialités ou des temporalités plus élaborées. Cela exclue toute combinaison de concepts élémentaires. ***Les concepts de spatialité/temporalité ne sont donc pas traités ici***.

III.1 Ensembles de spatialité primitive et de temporalité primitive

III.1.1 *Ensembles des concepts SIG primitifs*

Tous les travaux de recherche et de normalisation (cf. I) s'accordent sur la nécessité de mettre en œuvre les concepts *Point, Ligne* et *Polygone* pour décrire la spatialité des entités référencées et les concepts *Instant* et *Période* pour renseigner la temporalité. De part la perception suscitée, les concepts de la première énumération sont des éléments de l'ensemble des concepts de spatialité $\boldsymbol{S}$ et ceux de la seconde sont des éléments de l'ensemble des concepts de temporalité $\boldsymbol{T}$:

$\{\boldsymbol{Point}\} \in \boldsymbol{S}$ $\quad$ $\{\boldsymbol{Instant}\} \in \boldsymbol{T}$

$\{\boldsymbol{Ligne}\} \in \boldsymbol{S}$ $\quad$ $\{\boldsymbol{Période}\} \in \boldsymbol{T}$

$\{\boldsymbol{Polygone}\} \in \boldsymbol{S}$

Ces cinq concepts sont les plus utilisés par les acteurs du domaine de l'*Information Géographique*. À eux seuls, ils couvrent une large part des besoins de modélisation. C'est la principale raison qui nous a incités à définir les ensembles constitués par ces concepts et à leurs donner un nom. Nous verrons que les concepts de spatialité et de temporalité plus élaborés du paragraphe IV sont construits par combinaison de ces cinq concepts primitifs et qu'ils sont aussi impliqués dans les notions de ***concept dérivé*** et de ***concept flou*** qui seront abordées au paragraphe V et respectivement au Chapitre 8.

Étant donné que la plupart des autres spatialités et temporalités résultent d'une combinaison de ces cinq concepts, ces derniers seront qualifiés de ***primitif*** afin de bien les identifier.

Les deux énumérations exhaustives consécutives aux travaux de recherche et de normalisation permettent de définir en ***extension*** (Amsili, 2005 ; El Methni, Non daté ; Fondation Wikimédia, 2001 ; Iovleff, 2004 ; Savard, Non daté ; Sciences.ch, 2005 ; Zarba, 2005) d'une part l'ensemble concepts de spatialité primitive et, d'autre part, celui des concepts de temporalité primitive :

Définition 19 L'ensemble des ***concepts de spatialité primitive*** $\boldsymbol{S_p}$ est défini par :

$$\boldsymbol{S_p} = \{\boldsymbol{Point, Ligne, Polygone}\}$$

Définition 20 L'ensemble des ***concepts de temporalité primitive*** $\boldsymbol{T_p}$ est défini par :

$$\boldsymbol{T_p} = \{\boldsymbol{Instant, Période}\}$$

Par définition, les éléments de $\boldsymbol{S_p}$ et de $\boldsymbol{T_p}$ sont éléments de $\boldsymbol{S}$ et respectivement de $\boldsymbol{T}$. Les ensembles nouvellement construits sont donc des sous-ensembles de $\boldsymbol{S}$ et respectivement de $\boldsymbol{T}$:

$\boldsymbol{S_p} \subset \boldsymbol{S}$ $\quad$ $\boldsymbol{T_p} \subset \boldsymbol{T}$

De plus, ils sont disjoints car, par construction, leur intersection est vide :

$$S_p \cap T_p = \varnothing$$

Étant donné que les sous-ensembles S_p et T_p contiennent les concepts les plus couramment utilisés par les acteurs du domaine de l'*Information Géographique*, la définition et la désignation de leur union est naturelle :

Définition 21 L'ensemble des ***concepts SIG primitifs*** U_p est obtenu par union des deux ensembles S_p et T_p.

$$U_p = S_p \cup T_p$$

Par construction, les deux ensembles S_p et T_p forment une partition $\mathscr{P}(U_p)$ de l'ensemble des concepts SIG primitifs U_p puisqu'ils ne sont pas vides, que leur intersection est vide et que leur union est égale à l'ensemble U_p.

Par ailleurs, l'ensemble U_p est un sous-ensemble de U puisque les éléments de S_p et T_p sont aussi éléments de U :

$$\left.\begin{array}{lll} \forall u_p, u_p \in U_p & \textit{soit} \quad u_p \in S_p & \Rightarrow \quad u_p \in S \\ & \textit{soit} \quad u_p \in T_p & \Rightarrow \quad u_p \in T \end{array}\right\} \quad \Rightarrow \quad U_p \subset U$$

III.1.2 *Ensembles des pictogrammes SIG primitifs*

Au paragraphe II.1.2, la fonction bijective f a été définie entre l'ensemble des concepts SIG U et celui des pictogrammes SIG $\overset{vl}{U}$ (cf. définition 18). Nous désignons $f\big|_{S_p}$ et $f\big|_{T_p}$ deux restrictions de cette fonction sur les sous-ensembles des concepts de spatialité primitive S_p et de temporalité primitive T_p :

$$f\big|_{S_p} : \begin{array}{ccc} S_p & \longrightarrow & \overset{vl}{S}_p \\ s & \longrightarrow & \overset{vl}{s} \end{array} \qquad \text{et :} \qquad f\big|_{T_p} : \begin{array}{ccc} T_p & \longrightarrow & \overset{vl}{T}_p \\ t & \longrightarrow & \overset{vl}{t} \end{array}$$

Le faible nombre de concepts de spatialité primitive et de temporalité primitive permet de définir une à une les correspondances entre concept de spatialité primitive et pictogramme associé constituant les graphes des deux fonctions $f\big|_{S_p}$ et $f\big|_{T_p}$:

Définition 22 Le graphe $G^{f|_{S_p}}$ de la restriction $f\big|_{S_p}$ de la fonction f est :

$$G^{f|_{S_p}} = \{(\textit{Point}, \boxdot), (\textit{Ligne}, \boxed{\diagup}), (\textit{Polygone}, \blacksquare)\}$$

Définition 23 Le graphe $G^{f|_{T_p}}$ de la restriction $f\big|_{T_p}$ de la fonction f est :

$$G^{f|_{T_p}} = \{(\textit{Instant}, \odot), (\textit{Période}, \boxed{\diagup})\}$$

Les définitions 22 et 23 permettent de définir les ensembles de pictogrammes primitifs comme étant les images des ensembles des concepts de spatialité primitive S_p et de temporalité primitive T_p :

Définition 24 L'ensemble des ***pictogrammes de spatialité primitive*** $\overset{vl}{S}_p$ est l'image de S_p par la fonction $f\big|_{S_p}$.

$$\overset{vl}{S}_p = f\big|_{S_p}(S_p) = \{\boxdot, \boxed{\nearrow}, \boxed{\blacksquare}\}$$

Définition 25 L'ensemble des ***pictogrammes de temporalité primitive*** $\overset{vl}{T}_p$ est l'image de T_p par la fonction $f\big|_{T_p}$.

$$\overset{vl}{T}_p = f\big|_{T_p}(T_p) = \{\oslash, \oslash\}$$

Par analogie aux concepts SIG primitifs, ces pictogrammes primitifs ont été regroupés au sein d'un ensemble unique les regroupant :

Définition 26 L'ensemble des ***pictogrammes SIG primitifs*** $\overset{vl}{U}_p$ est l'union des ensembles $\overset{vl}{S}_p$ et $\overset{vl}{T}_p$.

$$\overset{vl}{U}_p = \overset{vl}{S}_p \cup \overset{vl}{T}_p \quad \Rightarrow \quad \overset{vl}{U}_p = \{\boxdot, \boxed{\nearrow}, \boxed{\blacksquare}, \oslash, \oslash\}$$

Par définition, les ensembles $\overset{vl}{S}_p$ et $\overset{vl}{T}_p$ sont des sous-ensembles $\overset{vl}{U}_p$ et de plus ils sont disjoints :

$$\overset{vl}{S}_p \subset \overset{vl}{U}_p \qquad \overset{vl}{T}_p \subset \overset{vl}{U}_p \qquad \overset{vl}{S}_p \cap \overset{vl}{T}_p = \varnothing$$

Comme pour l'ensemble U_p des concepts SIG primitifs, les deux sous-ensembles $\overset{vl}{S}_p$ et $\overset{vl}{T}_p$ constituent une partition $\mathscr{P}\left(\overset{vl}{U}_p\right)$ de l'ensemble des pictogrammes SIG primitifs $\overset{vl}{U}_p$.

III.1.3 *Ensembles des entités référencées SIG primitives*

Au paragraphe I, l'application d'affectation $<<$ a été définie de l'ensemble des entités référencées E vers l'ensemble des concepts SIG U (cf. définition 17). Cette application permet donc de définir les sous-ensembles E_{S_p} et E_{T_p} comme étant les images réciproques de S_p et de T_p :

$$<<: \; E_S \longrightarrow S \qquad e_s \longrightarrow s \qquad \text{et :} \qquad <<: \; E_T \longrightarrow T \qquad e_t \longrightarrow t$$

Définition 27 L'ensemble des entités référencées ***à spatialité primitive*** E_{S_p} est l'ensemble des concepts de E_S dont la propriété spatiale s'exprime par un concept de spatialité primitive. C'est image réciproque de S_p par l'application d'affectation $<<$.

$$E_{S_p} = <<^{-1}(S_p) = \{ e_s \in E_S \mid <<(e_s) \in S_p \}$$

Définition 28 L'ensemble des entités référencées ***à temporalité primitive*** E_{T_p} est l'ensemble des concepts de E_T dont la propriété temporelle s'exprime par un concept de temporalité primitive. C'est image réciproque de T_p par l'application d'affectation $<<$.

$$E_{T_p} = <<^{-1}(T_p) = \{ e_t \in E_T \mid <<(e_t) \in T_p \}$$

Comme précédemment, nous définissons et désignons l'union des deux ensembles d'entités référencées de nature primitive :

Définition 29 L'ensemble des entités référencées ***SIG primitives*** E_{U_p} est l'union des ensembles E_{S_p} et E_{T_p}.

$$E_{U_p} = E_{S_p} \cup E_{T_p}$$

En outre, comme les ensembles de départ S_p et T_p sont disjoints, les images réciproques E_{S_T} et E_{T_p} le sont aussi. Ces derniers ensembles constituent donc une partition $\mathscr{P}(E_{U_p})$ de l'ensemble E_{U_p} des entités référencées SIG primitives.

Il est possible d'affiner la taxinomie des entités référencées puisque les concepts primitifs des ensembles de spatialité primitive S_p et de temporalité primitive T_p sont des singletons. À ce titre, ils ont une image réciproque dans les ensembles E_{S_p} ou E_{T_p} (cf. remarque en I). Les ensembles des entités référencées ***à spatialité Point***, ***Ligne***, etc. sont ainsi créés. Ce sont les ensembles les plus détaillés de la taxinomie des entités référencées :

$$E_{\{Point\}} = <<^{-1}(\{Point\}) \qquad E_{\{Point\}} \subset E_{S_p}$$
$$E_{\{Ligne\}} = <<^{-1}(\{Ligne\}) \qquad E_{\{Ligne\}} \subset E_{S_p}$$
$$E_{\{Polygone\}} = <<^{-1}(\{Polygone\}) \qquad E_{\{Polygone\}} \subset E_{S_p}$$

$$E_{\{Instant\}} = <<^{-1}(\{Instant\}) \qquad E_{\{Instant\}} \subset E_{T_p}$$
$$E_{\{Période\}} = <<^{-1}(\{Période\}) \qquad E_{\{Période\}} \subset E_{T_p}$$

Même si l'élément générateur est un singleton, les sous-ensembles $E_{\{Point\}}$, $E_{\{Ligne\}}$, etc. sont infinis. Nous verrons en VII l'intérêt de raffiner à l'extrême les taxinomies et en particulier celle des entités référencées dans la perspective de la dérivation d'un métamodèle.

III.2 Exemples d'entité référencée à spatialité primitive ou à temporalité primitive

Doté du langage pictogrammique, il est facile de renseigner la spatialité et/ou la temporalité d'une entité référencée.

Par exemple, le concept thématique *Bâtiment*$_{<<\blacksquare}$ (cf. figure 86-modèle A) indique que les acteurs du domaine s'intéressent au polygone délimitant les bâtiments et probablement à une au moins des propriétés du concept *Polygone*, périmètre et/ou surface.

Dans un contexte de prévention des risques naturels, la date à laquelle a eu lieu une avalanche est une information majeure. Pour renseigner la date à laquelle a lieu une *Avalanche*, il faut annoter l'entité référencée

avec le pictogramme de temporalité primitive ⏲. La modèle B de la figure 86 montre l'entité référencée à temporalité primitive *Avalanche*<<⏲.

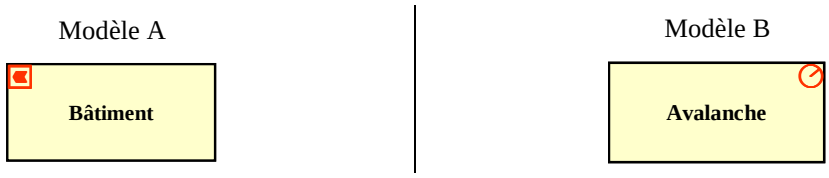


Figure 86 Entités référencées à spatialité primitive (modèle A) ou à temporalité primitive (modèle B).

III.3 En résumé

La création des ensembles $\boldsymbol{U}_p$ et $\overset{vl}{\boldsymbol{U}}_p$ montrent que les ensembles $\boldsymbol{U}$ et $\overset{vl}{\boldsymbol{U}}$ ne sont pas vide. L'ensemble des entités référencées $\boldsymbol{E}$ n'est pas vide non plus puisqu'au moins les concepts *Bâtiment* et *Avalanche* appartiennent à $\boldsymbol{E}$ (cf. exemple en III.2), plus précisément $\boldsymbol{E}_{S_p}$ et $\boldsymbol{E}_{T_p}$.

Les neuf ensembles primitifs créés ici $\boldsymbol{S}_p$, $\boldsymbol{T}_p$ et $\boldsymbol{U}_p$ d'une part, $\overset{vl}{\boldsymbol{S}}_p$, $\overset{vl}{\boldsymbol{T}}_p$ et $\overset{vl}{\boldsymbol{U}}_p$ d'autre part mais aussi $\boldsymbol{E}_{S_T}$, $\boldsymbol{E}_{T_p}$ et $\boldsymbol{E}_{U_p}$ forment trois partitions $\mathscr{P}(\boldsymbol{U}_p)$, $\mathscr{P}\left(\overset{vl}{\boldsymbol{U}}_p\right)$ et $\mathscr{P}(\boldsymbol{E}_{U_p})$. L'avantage d'avoir structuré les concepts SIG, les pictogrammes et les entités référencées au sein de partitions est que chacun d'eux appartient à un et un seul des sous-ensembles évitant par là même toute ambiguïté. La figure 87 montre l'intégration des neuf sous- ensembles dans le diagramme de la figure 85.

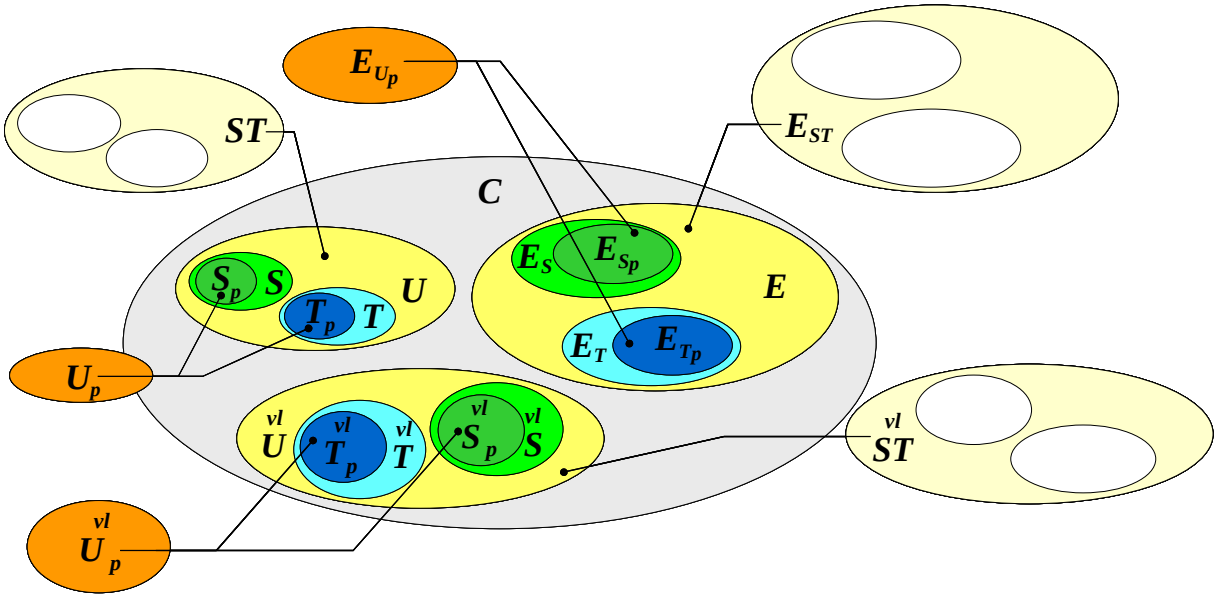


Figure 87 Représentation des *Taxinomie SIG*, *des Pictogrammes* et *des entités référencées* et des partitions $\mathscr{P}(\boldsymbol{U}_p)$, $\mathscr{P}\left(\overset{vl}{\boldsymbol{U}}_p\right)$ et $\mathscr{P}(\boldsymbol{E}_{U_p})$.

Les ensembles $\boldsymbol{U}_p$, $\overset{vl}{\boldsymbol{U}}_p$ et $\boldsymbol{E}_{U_p}$ ont été déportés car il n'est pas possible de les représenter autrement dans ce type de diagramme.

IV SPATIALITE COMPOSITE ET TEMPORALITE COMPOSITE

Au paragraphe II, l'ensemble des concepts SIG, celui des pictogrammes SIG et celui des entités référencées ont été définis successivement. Ces trois ensembles ont été affinés au paragraphe III en intégrant la connaissance des concepts de spatialité primitive et de temporalité primitive. Cette démarche de structuration a permis de construire trois ébauches de taxinomie une par typologie de concepts : la taxinomie SIG, celle des pictogrammes et celle des entités référencées.

Comme cela a été déjà évoqué, les concepts SIG primitifs ne suffisent pas, à eux seuls, à satisfaire tous les besoins d'expression de la spatialité et de temporalité au cours des séances d'analyse. Aussi, les différents travaux de recherche et de normalisation ont conduit à définir des concepts SIG composites qui résultent d'une combinaison de concepts primitifs.

Toutes les descriptions des concepts de spatialité et de temporalité de la littérature et les exemples utilisés par les auteurs pour illustrer leurs propos sont à l'origine de quatre constats :

- ⇨ Le premier est que la plupart des descriptions et des exemples ne ***combinent que deux concepts primitifs*** et qu'au-delà, la compréhension devient difficile.
- ⇨ Le deuxième est que les descriptions et les exemples mettent en œuvre principalement des ***combinaisons soit des concepts de spatialité entre eux, soit des concepts de temporalité entre eux***. La raison implicite est qu'il est plus facile de combiner des concepts de même nature mais plus délicat de combiner des concepts de spatialité et de temporalité.
- ⇨ Le troisième est qu'un certain nombre de combinaisons porte entre :
  - ***Concepts SIG***. Par exemple, les concepts *Point* et *Polygone* vont être combinés pour représenter le centre d'une ville et son étendue.
  - ***Entités référencées***. Ici, les auteurs combinent des entités référencées et créent des conglomérats de taille plus importante.
- ⇨ Le quatrième est que, lorsque les exemples impliquent un nombre élevé de concepts, ***la combinaison est en fait une répétition du même concept***.

Après avoir énoncé les définitions générales des spatialités composites et des temporalités composites, le présent paragraphe est articulé autour de ces quatre constats. Le premier constat a conduit à s'intéresser aux bi-spatialités, bi-temporalités et aux combinaisons des deux, le troisième est à l'origine des macro-spatialités et macro-temporalités et les multi-spatialités et multi-temporalités sont dues au quatrième. Le deuxième constat est « transversal » et crée des typologies au sein des différentes spatialités ou temporalités.

IV.1 Spatialité composite générale et temporalité composite générale

IV.1.1 *Définitions générales*

IV.1.1.1 Ensembles des concepts SIG composites

Les concepts de spatialité primitive et de temporalité primitive sont des propriétés n'impliquant qu'une seule propriété spatiale ou temporelle. Toutes les autres spatialités et/ou temporalités résultent d'une combinaison de concept SIG primitifs. À ce titre, elles sont composites et les concepts SIG aussi. Les ensembles relatifs aux concepts composites se définissent tout naturellement par complémentarité aux ensembles primitifs.

Définition 30 L'ensemble des ***concepts de spatialité composite (pure)*** S_c est l'ensemble complémentaire à S_p dans S.

$$S_c = \complement_S(S_p)$$

Définition 31 L'ensemble des ***concepts de temporalité composite (pure)*** T_c est l'ensemble complémentaire à T_p dans T.

$$T_c = \complement_T(T_p)$$

Par construction, S_c et T_c sont des sous-ensembles de S ou de T, ensembles qui ne contiennent que des concepts de spatialité pure ou de temporalité pure.

Les deux ensembles S_c et T_c ne sont pas les seuls à combiner les concepts SIG. En effet, l'ensemble des concepts de spatialité/temporalité ST impliquent au moins un concept de spatialité et un concept de temporalité (cf. définition 15).

Nous définissons :

Définition 32 L'ensemble des ***concepts SIG composites*** U_c est la l'union des trois ensembles S_c, T_c et ST :

$$U_c = S_c \cup T_c \cup ST$$

Par construction, l'ensemble des concepts SIG composites est inclus dans l'ensemble des concepts SIG U et est le complémentaire à U_p dans U :

$$U_c \subset U \qquad\qquad U_c = \complement_U(U_p)$$

Les trois nouveaux ensembles S_c, T_c et U_c complètent la taxinomie SIG en offrant une double grille de classification des concepts :

- ⇨ La première résulte de la partition $\mathscr{P}_1(U)$ qui regroupe les concepts de spatialité pure (S), de temporalité pure (T) ou ceux combinant spatialité de temporalité (ST).
- ⇨ La seconde est induite par la partition $\mathscr{P}_2(U)$ qui organise les concepts SIG en concepts primitifs (U_p) et en concepts composites (U_c).

La figure 88 montrent la représentation des partitions $\mathscr{P}(S)$, $\mathscr{P}(T)$, $\mathscr{P}(U_p)$, $\mathscr{P}(U_c)$, $\mathscr{P}_1(U)$ et $\mathscr{P}_2(U)$.

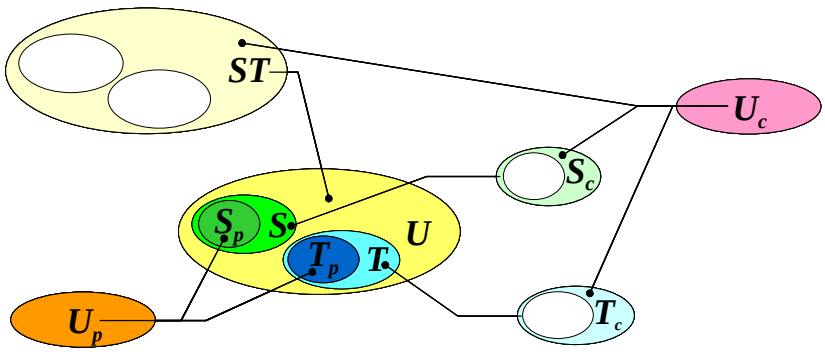


Figure 88 Représentation de la *Taxinomie SIG* et des partitions $\mathscr{P}(S)$, $\mathscr{P}(T)$, $\mathscr{P}(U_p)$, $\mathscr{P}(U_c)$, $\mathscr{P}_1(U)$ et $\mathscr{P}_2(U)$.

Nous avons donc construit une double grille de classification au sein de la taxinomie SIG offrant par là même une double terminologie à chaque concept SIG du domaine de l'*Information Géographique*.

IV.1.1.2 Ensembles des pictogrammes SIG composites

Une démarche analogue permet de définir les ensembles de ***pictogrammes de spatialité composite*** $\overset{vl}{S}_c$, ***de temporalité composite*** $\overset{vl}{T}_c$ et ***SIG composite*** $\overset{vl}{U}_c$.

Comme pour les concepts SIG, ces trois ensembles forment une seconde partition $\mathscr{P}_2\left(\overset{vl}{U}\right)$ de l'ensemble des pictogrammes SIG $\overset{vl}{U}$ offrant ainsi un second mode de classification des pictogrammes au sein de la taxinomie des pictogrammes.

IV.1.1.3 Ensembles des entités référencées SIG composites

En réitérant une troisième fois une démarche semblable à celle de construction des concepts SIG composites, les ensembles des entités référencées ***à spatialité composite*** E_{S_c} , ***à temporalité composite*** E_{T_c} et ***SIG composite*** E_{U_c} ont été définis. Ces trois nouveaux ensembles d'entités référencées organisent une seconde partition $\mathscr{P}_2(E)$ sur l'ensemble des entités référencées E , entités qui peuvent donc être classées suivant une double grille de classification au sein de la taxinomie des entités référencées.

IV.1.2 *En résumé*

Puisque la même démarche a été appliquée trois fois pour définir la taxinomie SIG, celle des pictogrammes et celle des entités référencées, leurs structures générales sont identiques. Nous verrons par la suite qu'elles seront amenées à diverger.

Pour représenter les taxinomies, nous avons adopté un ***formalisme hybride*** empruntant la représentation elliptique des ensembles aux diagrammes de Venn et la relation de généralisation/spécialisation au langage UML pour symboliser l'inclusion d'un ensemble dans un autre. Ce formalisme hybride permet de représenter les dix ensembles définis par taxinomie et les six partitions sur un seul diagramme.

La figure 89 montre les ensembles et les partitions de la taxinomie SIG. Étant donné que les taxinomies des pictogrammes et des entités référencées ont été construites suivant une démarche analogue, deux autres diagrammes similaires auraient pu être dessinés. Ils auraient été identiques à celui-ci.

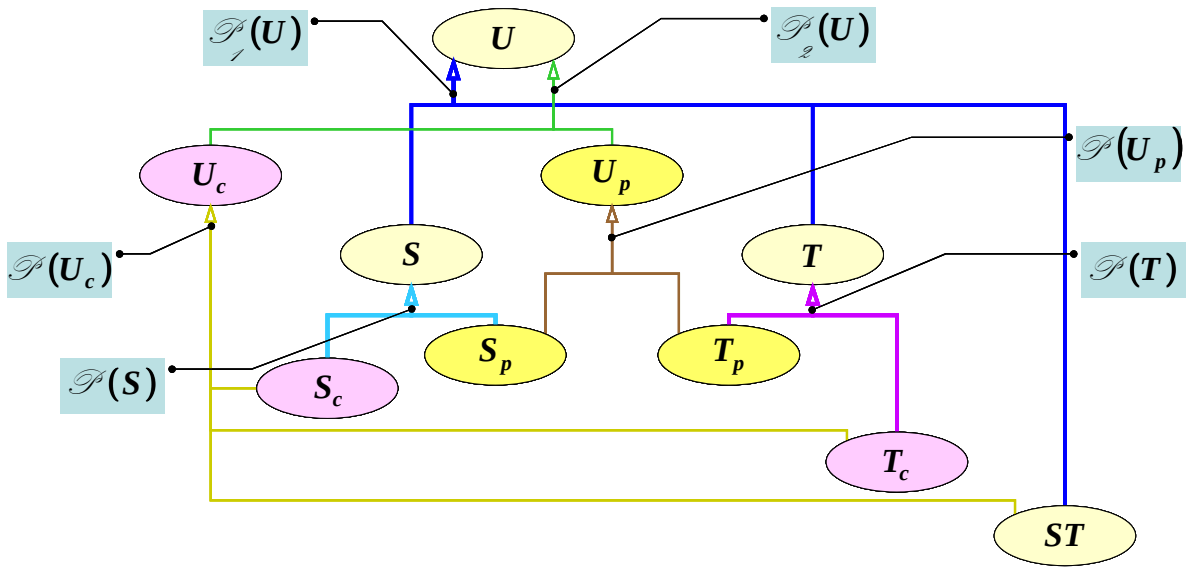


Figure 89 Représentation de la *Taxinomie SIG* suivant le formalisme hybride.

Le diagramme de la figure 89 montre parfaitement les deux grilles de lecture/classification créées pour chacune des taxinomies :

- ⇨ La première résulte de la partition $\mathscr{P}_{/}(U)$ qui structurent les concepts selon qu'ils sont à spatialité pure (S), à temporalité pure (T) ou qu'ils combinent spatialité et temporalité (ST). En outre, les sous-ensembles à spatialité pure et à temporalité pure sont organisés par les partitions :
  - $\mathscr{P}(S)$ composée de S_p et de S_c.
  - $\mathscr{P}(T)$ composée de T_p et de T_c.
- ⇨ La seconde est induite par la partition $\mathscr{P}_{2}(U)$ qui organisent les concepts en concepts primitifs (U_p) ou en concepts composites (U_c). Les sous-ensembles primitifs et composites ont eux aussi été désagrégés suivant les partitions.
  - $\mathscr{P}(U_p)$ composée de S_p et de T_p.
  - $\mathscr{P}(U_c)$ composée de S_c et de T_c.

Le diagramme de la figure 89 sur la taxinomie SIG constitue la structure de base des trois taxinomies. Tout nouveau concept ou nouvel ensemble devra s'insérer dans cette structure comme sous-ensemble de l'un des cinq ensembles « terminaux » S_p, T_p, S_c, T_c ou ST. La vision ensembliste peut être étendue aux éléments d'un ensemble en considérant non pas l'élément mais son singleton.

La structure générale des trois taxinomies offre un cadre de lecture des concepts du domaine de l'*Information Géographique* sans ambiguïté puisque chacun appartient à un et un seul des ensembles « terminaux ». L'effort de formalisation entrepris permet de définir une terminologie plus stricte qui facilte la rédaction du reste du document.

IV.2 Bi-spatialité, bi-temporalité ou mono-spatialité et mono-temporalité

Les spatialités et temporalités abordées ici résultent du premier constat effectué au début paragraphe qui incite à n'étudier, dans un premier temps, que les combinaisons n'impliquant que 2 concepts SIG primitifs, d'où le titre du paragraphe.

Les définitions générales des concepts et des ensembles SIG composites (cf. définitions 15, 30 et 31) sont peu contraignantes puisqu'elles ne stipulent ni la nature de la combinaison des concepts, ni leur nombre, etc. Aussi, tout un chacun peut donc définir sa façon de combiner des concepts SIG entre eux.

Implicitement, cela signifie qu'il y a plusieurs de façons de combiner des concepts SIG et chacune correspondant à une préoccupation propre à son auteur.

Quelque soit la façon de combiner des concepts SIG, l'auteur utilise implicitement une loi de composition interne (Fondation Wikimédia, 2001) de l'ensemble des concepts SIG U sur lui-même puisqu'il est amené à combiner au moins deux concepts SIG et le résultat est un nouveau concept SIG appartenant à U (cf. définitions 13 à 16).

Ayant adopté le langage pictogrammique de Perceptory, nous avons retenu, dans un premier temps, trois mécanismes de combinaison des concepts SIG de cet atelier de génie logiciel :

- ⇨ Le premier permet à une entité référencée de posséder ***plusieurs spatialités ou temporalités***. C'est un mécanisme ***« additif »*** entre concepts SIG qui est alors mis en œuvre.
- ⇨ Le deuxième introduit la possibilité pour une entité référencée ***de pouvoir changer*** de spatialité ou de temporalité. Le mécanisme utilisé est celui de ***« substitution »*** d'une spatialité par une autre spatialité ou d'une temporalité par une autre temporalité. Ce deuxième mécanisme ne peut être appliqué qu'entre concepts de même nature.
- ⇨ Le troisième est particularité des concepts de spatialité qui peuvent avoir des propriétés temporelles pour rendre compte de l'évolution de la propriété spatiale dans le temps (cf. définition 3). Le mécanisme de combinaison utilisé ici est celui de l'***« affectation »*** d'une propriété temporelle à une propriété spatiale.

Le mécanisme additif correspond aux géométries multiples et temporalités multiples définies par Yvan Bédard et son équipe, celui de substitution aux géométries alternatives et temporalités alternatives et d'affectation aux géométries spatio-temporelles.

Les lois de composition additive, et de substitution et d'affectation détaillées dans les paragraphes suivants découlent de cette réflexion. Elles permettent de définir de nouveaux sous-ensembles dans la taxinomie SIG, sous-ensembles qui sont ensuite projetés dans les taxinomies des Pictogrammes et des entités référencées.

Pour exprimer les mécanismes additif et de substitution, nous utilisons l'algèbre de Boole car le mécanisme additif est naturellement décrit par l'opérateur *« et »* et le mécanisme de substitution par l'opérateur *« ou exclusif »*. Dans la suite du document, le mécanisme additif sera représenté par le symbole $\otimes$ et le mécanisme additif de substitution par $\oplus$. Le mécanisme d'affectation sera quant à lui symbolisé par un chevron ($\overset{?}{<<}$) surmonté d'une information précisant la nature de l'application.

La même logique a prévalu pour définir les concepts de bi-spatialité ou bi-temporalité. Dans un premier temps les concepts de spatialité pure et de temporalité pure sont abordés car plus faciles à appréhender et ensuite les concepts impliquant spatialité et temporalité sont définis. Pour chaque cas, la convention pictogrammique de l'atelier de génie logiciel Perceptory est formulée, la définition de la loi de composition est énoncée ce qui permet de définir les ensembles de la taxinomie SIG et de les reporter ensuite dans les deux autres taxinomies. Des exemples viennent illustrer ensuite les nouveaux types de spatialité et/ou de temporalité.

IV.2.1 *Bi-spatialité ou bi-temporalité*

Les concepts de bi-spatialité ou bi-temporalité sont des concepts SIG impliquant dans la combinaison soit deux concepts de spatialité primitive soit deux concepts de temporalité primitive. À ce titre, ce sont des concepts composites mais qui reste de même nature : spatiale ou temporelle. Aussi, les nouveaux concepts appartiennent à l'ensemble des concepts de spatialité $\boldsymbol{S}$ ou de temporalité $\boldsymbol{T}$.

IV.2.1.1 Spatialité multiple ou de temporalité multiple

La spatialité multiple et la temporalité multiple résultant du mécanisme additif permettent d'annoter une entité référencée de deux spatialités ou de temporalités différentes. Dans Perceptory, la représentation des pictogrammes SIG multiples suit la convention pictogrammique suivante :

Convention La représentation graphique d'une ***pictogrammie de spatialité multiple*** ou ***de temporalité multiple*** est obtenue en *superposant l'un au-dessus de l'autre* les pictogrammes SIG primitifs la composant.

IV.2.1.1.a *Définitions des ensembles*

Nous définissons :

Définition 33 La loi d'addition spatiale $\overset{S}{\otimes}$ est une loi de composition commutative et idempotente de $\boldsymbol{S_p \times S_p}$ dans $\boldsymbol{S}$ qui combine, suivant un mécanisme additif, les éléments d'un couple de concepts de spatialité primitive $\left(\boldsymbol{s_{p_1}, s_{P_2}}\right)$ pour créer un nouveau concept $\boldsymbol{s^{\otimes}}$:

$$\overset{S}{\otimes}: \quad \boldsymbol{S_p \times S_p} \longrightarrow \boldsymbol{S}$$
$$\left(\boldsymbol{s_{p_1}, s_{P_2}}\right) \longrightarrow \boldsymbol{s^{\otimes} = s_{p_1} \overset{S}{\otimes} s_{p_2}}$$

Définition 34 La loi d'addition temporelle $\overset{T}{\otimes}$ est une loi de composition commutative[70] et idempotente[71] de $\boldsymbol{T_p \times T_p}$ dans $\boldsymbol{T}$ qui combine, suivant un mécanisme additif, les éléments d'un couple de concepts de temporalité primitive $\left(\boldsymbol{t_{p_1}, t_{P_2}}\right)$ pour créer un nouveau concept $\boldsymbol{t^{\otimes}}$:

[70] $\forall (x,y) \in A^2, \quad x \bullet y = y \bullet x$ (Fondation Wikimédia, 2001 ; Sciences.ch, 2005)

[71] $\forall x \in A, \quad x \bullet x = x$ (Fondation Wikimédia, 2001 ; Sciences.ch, 2005)

$$\overset{T}{\otimes}: \; T_p \times T_p \longrightarrow T$$
$$(t_{p_1}, t_{P_2}) \longrightarrow t^{\otimes} = t_{p_1} \overset{T}{\otimes} t_{p_2}$$

Nota 1 Les lois d'addition spatiale et d'addition temporelle ont été munies de la propriété de commutativité car l'ordre des concepts n'a pas d'importance.

Nota 2 La propriété d'idempotence a été retenue pour les deux lois d'addition spatiale et temporelle afin d'exclure les couples d'un même concept (Point/Point, Instant/Instant, etc.) dont le cas est traité par la multi-spatialité ou la multi-temporalité (cf. IV.4).

Les lois d'addition spatiale $\overset{S}{\otimes}$ et temporelle $\overset{T}{\otimes}$ permettent de définir les ensembles ***des concepts de spatialité multiple*** $S^{\otimes}$ et ***de temporalité multiple*** $T^{\otimes}$:

$$S^{\otimes} = \left\{ s \in S \;\middle|\; s = s_{p_1} \overset{S}{\otimes} s_{p_2} \quad \textit{avec } s_{p_1} \in S_p \wedge s_{p2} \in S_p \wedge s_{p_1} \neq s_{p_2} \right\} \quad \text{soit :}$$

$$S^{\otimes} = \left\{ \begin{array}{ll} \textit{Point} \overset{S}{\otimes} \textit{Ligne}, & \textit{Point} \overset{S}{\otimes} \textit{Polygone}, \\ \textit{Ligne} \overset{S}{\otimes} \textit{Polygone} & \end{array} \right\}$$

$$T^{\otimes} = \left\{ t \in T \;\middle|\; t = t_{p_1} \overset{T}{\otimes} t_{p_2} \quad \textit{avec } t_{p_1} \in T_p \wedge t_{p2} \in T_p \wedge t_{p_1} \neq t_{p_2} \right\} \quad \text{soit :}$$

$$T^{\otimes} = \left\{ \textit{Instant} \overset{T}{\otimes} \textit{Période} \right\}$$

Il est à noter que les nouveaux concepts $s^{\otimes}$ et $t^{\otimes}$ n'appartiennent pas à S_p ou à T_p puisqu'ils sont composés de deux concepts primitifs. Par définition de S_c et de T_c, ils appartiennent à S_c ou à T_c puisqu'ils combinent soit des concepts de spatialité primitive soit des concepts de temporalité primitive. Les ensembles $S^{\otimes}$ et $T^{\otimes}$ sont donc des sous-ensembles de S_c ou de T_c :

$$S^{\otimes} \subset S_c \qquad\qquad T^{\otimes} \subset T_c$$

En appliquant les restrictions $f|_{S^{\otimes}}$ et $f|_{T^{\otimes}}$ aux ensembles ***des concepts de spatialité multiple*** $S^{\otimes}$ et ***de temporalité multiple***, nous obtenons les ensembles ***des pictogrammes de spatialité multiple*** $\overset{vl}{S}^{\otimes}$ et ***de temporalité multiple*** $\overset{vl}{T}^{\otimes}$, sous-ensemble de $\overset{vl}{S}_c$ et respectivement de $\overset{vl}{T}_c$:

$$\overset{vl}{S}^{\otimes} = f|_{S^{\otimes}}(S^{\otimes}) = \left\{ \begin{array}{c} \boxdot \; \boxdot \; \boxed{\diagup} \\ \boxed{\diagup}, \boxed{\blacksquare}, \boxed{\blacksquare} \end{array} \right\} \qquad \overset{vl}{S}^{\otimes} \subset \overset{vl}{S}_c$$

$$\overset{vl}{T}^{\otimes} = f|_{T^{\otimes}}(T^{\otimes}) = \left\{ \begin{array}{c} \oslash \\ \oslash \end{array} \right\} \qquad \overset{vl}{T}^{\otimes} \subset \overset{vl}{T}_c$$

Grâce à l'application d'affectation $<<$, les images réciproques $E_{S^{\otimes}}$ et $E_{T^{\otimes}}$ des ensembles $S^{\otimes}$ et $T^{\otimes}$ peuvent être définies dans les ensembles respectifs des entités référencées composites E_{S_c} et E_{T_c} :

$$E_{S^{\otimes}} = <<^{-1}(S^{\otimes}) \qquad E_{S^{\otimes}} \subset E_{S_c}$$

$$E_{T^{\otimes}} = <<^{-1}(T^{\otimes}) \qquad E_{T^{\otimes}} \subset E_{T_c}$$

Comme précédemment, l'application d'affectation **<<** peut être appliquée aux singletons des concepts primitifs et créer ainsi les ensembles ***des entités référencées à spatialité multiple Point/Ligne***, ***à spatialité multiple Point/Polygone***, etc., sous-ensembles de celui des entités référencées à spatialité composite E_{S_c} :

$$E_{\left\{Point \overset{S}{\otimes} Ligne\right\}} =<<^{-1}\left(\left\{Point \overset{S}{\otimes} Ligne\right\}\right) \qquad E_{\left\{Point \overset{S}{\otimes} Ligne\right\}} \subset E_{S^{\otimes}}$$

$$E_{\left\{Point \overset{S}{\otimes} Polygone\right\}} =<<^{-1}\left(\left\{Point \overset{S}{\otimes} Polygone\right\}\right) \qquad E_{\left\{Point \overset{S}{\otimes} Polygone\right\}} \subset E_{S^{\otimes}}$$

$$E_{\left\{Ligne \overset{S}{\otimes} Polygone\right\}} =<<^{-1}\left(\left\{Ligne \overset{S}{\otimes} Polygone\right\}\right) \qquad E_{\left\{Ligne \overset{S}{\otimes} Polygone\right\}} \subset E_{S^{\otimes}}$$

La même démarche peut être appliquée à l'ensemble des concepts de temporalité multiple $T^{\otimes}$:

$$E_{\left\{Instant \overset{T}{\otimes} Période\right\}} =<<^{-1}\left(\left\{Instant \overset{T}{\otimes} Période\right\}\right) \qquad E_{\left\{Instant \overset{T}{\otimes} Période\right\}} \subset E_{T^{\otimes}}$$

Par définition de la spatialité multiple et de la temporalité multiple, les deux spatialités primitives ou les deux temporalités primitives renseignent le concept thématique de l'entité référencée.

Le paragraphe suivant donne deux exemples d'entités référencées annotées avec des concepts de spatialité multiple ou de temporalité multiple. Ils permettent d'illustrer et de mieux appréhender la notion de concept multiple.

IV.2.1.1.b *Exemples d'entité référencée à spatialité multiple ou à temporalité multiple*

Les exemples d'entités référencées à spatialité multiple (*Agglomération*) et à temporalité multiple (*Festival*) ci-dessous sont extraits des communications de l'équipe d'Yvan Bédard (Bédard, 1998, 1999c).

Dans le modèle A de la figure 90, l'entité référencée $Agglomération_{<<\boxdot}$ est annotée du pictogramme de spatialité multiple $s^{\overset{vl}{\otimes}} = \boxdot$ qui combine les pictogrammes de spatialité primitives *Point* (⊡) et *Polygone* (◙).

Avec cette pictogrammie, cette entité référencée permet d'attribuer deux propriétés spatiales simultanément à chacune de ces instances : le centre de l'agglomération via la spatialité ponctuelle et son emprise territoriale grâce à la spatialité polygonale.

De façon analogue, le pictogramme de spatialité multiple $t^{\overset{vl}{\otimes}} = \begin{matrix}\oslash\\ \medbullet\end{matrix}$ renseignant la temporalité de l'entité référencée $Festival_{<<\begin{matrix}\oslash\\ \medbullet\end{matrix}}$ (cf. figure 90-modèle B) indique que les organisateurs de ce type d'événements s'intéressent simultanément à un moment précis du festival (⊘), la prestation de l'artiste principal par exemple, ainsi qu'à la période où a lieu le festival (◔). Chaque instance de l'entité référencée $Festival_{<<\begin{matrix}\oslash\\ \medbullet\end{matrix}}$ sera renseignée avec ces deux informations.

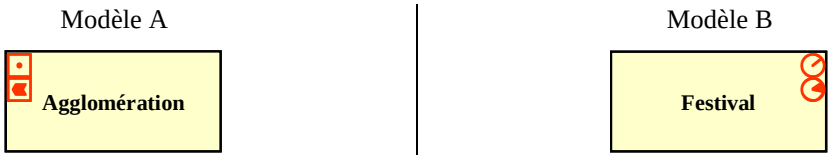


Figure 90 Entités référencées à spatialité multiple (modèle A) ou à temporalité multiple (modèle B).

IV.2.1.2 Spatialité alternative ou de temporalité alternative

La spatialité alternative et la temporalité alternative permettent de modéliser des entités référencées dont les instances n'ont pas toutes le même type de spatialité de temporalités. Souvent, le changement de spatialité ou de

temporalité est lié à un critère défini par l'acteur du domaine. Ce critère agit comme un **« *seuil* »** : au-dessous, les instances de l'entité référencée ont une certaine spatialité ou temporalité, au-dessus, ils en ont une autre.

Comme pour les autres pictogrammies, Yvan Bédard et son équipe ont adopté la convention suivante pour représenter graphiquement les concepts alternatifs :

Convention Dans la représentation graphique d'une ***pictogrammie de spatialité alternative*** ou ***de temporalité alternative***, les pictogrammes SIG primitifs la composant sont *accolés sans aucun espace*.

IV.2.1.2.a *Définitions des ensembles*

Nous définissons :

Définition 35 La loi de substitution de spatialité $\overset{S}{\oplus}$ est une loi de composition commutative et idempotente de $S_p \times S_p$ dans S qui combine, suivant un mécanisme de substitution, les éléments d'un couple de concepts de spatialité primitive (s_{p_1}, s_{P_2}) pour créer un nouveau concept $s^{\oplus}$:

$$\begin{aligned} \overset{S}{\oplus}: \quad & S_p \times S_p \longrightarrow S \\ & (s_{p_1}, s_{P_2}) \longrightarrow s^{\oplus} = s_{p_1} \overset{S}{\oplus} s_{p_2} \end{aligned}$$

Définition 36 La loi de substitution de temporalité $\overset{T}{\oplus}$ est une loi de composition commutative et idempotente de $T_p \times T_p$ dans T qui combine, suivant un mécanisme de substitution, les éléments d'un couple de concepts de temporalité primitive (t_{p_1}, t_{P_2}) pour créer un nouveau concept $t^{\oplus}$

$$\begin{aligned} \overset{T}{\oplus}: \quad & T_p \times T_p \longrightarrow T \\ & (t_{p_1}, t_{p_2}) \longrightarrow t^{\oplus} = t_{p_1} \overset{T}{\oplus} t_{p_2} \end{aligned}$$

Nota 1 Les lois de substitution de spatialité ou de temporalité ont été dotées de la propriété de commutativité afin de ne pas prévilégier l'ordre des concepts.

Nota 2 Les combinaisons d'un même concept (Période/Période, Polygone/Polygone, etc.) ont été exclues des définitions 35 et 36 car substituer un concept par un concept de même type n'a pas de sens. C'est la raison pour laquelle la propriété d'idempotence a été retenue.

Les deux lois de substitution $\overset{S}{\oplus}$ et $\overset{T}{\oplus}$ permettent de définir les ensembles ***des concepts de spatialité alternative*** $S^{\oplus}$ et ***de temporalité alternative*** $T^{\oplus}$ de la même façon que précédemment :

$$S^{\oplus} = \left\{ s \in S \;\middle|\; s = s_{p_1} \overset{S}{\oplus} s_{p_2} \quad \textit{avec } s_{p_1} \in S_p \wedge s_{p2} \in S_p \wedge s_{p_1} \neq s_{p_2} \right\} \qquad \text{soit :}$$

$$S^{\oplus} = \left\{ \begin{array}{ll} \textit{Point} \overset{S}{\oplus} \textit{Ligne}, & \textit{Point} \overset{S}{\oplus} \textit{Polygone}, \\ \textit{Ligne} \overset{S}{\oplus} \textit{Polygone} & \end{array} \right\} \qquad S^{\oplus} \subset S_c$$

$$T^{\oplus} = \left\{ t \in T \;\middle|\; t = t_{p_1} \overset{T}{\oplus} t_{p_2} \quad \textit{avec } t_{p_1} \in T_p \wedge t_{p2} \in T_p \wedge t_{p_1} \neq t_{p_2} \right\} \qquad \text{soit :}$$

$$T^{\oplus} = \left\{ \textit{Instant} \overset{T}{\oplus} \textit{Période} \right\} \qquad S^{\oplus} \subset S_c$$

Pour les mêmes raisons que précédemment, les concepts $s^{\oplus}$ et $t^{\oplus}$ sont définis en combinant deux concepts primitifs, ils appartiennent donc aussi à S_c ou à T_c. Les ensembles $S^{\oplus}$ et $T^{\oplus}$ sont des sous-ensembles de S_c et respectivement de T_c :

$$S^{\oplus} \subset S_c \qquad\qquad T^{\oplus} \subset T_c$$

De même, les ensembles ***des pictogrammes de spatialité alternative*** $\overset{vl}{S}{}^{\oplus}$ et ***de temporalité alternative*** $\overset{vl}{T}{}^{\oplus}$ sont définis comme étant les images des restrictions $f|_{S^{\oplus}}$ et $f|_{S^{\oplus}}$:

$$\overset{vl}{S}{}^{\oplus} = f|_{S^{\oplus}}\left(S^{\oplus}\right) = \{\boxdot, \boxdot, \boxdot\} \qquad \overset{vl}{S}{}^{\oplus} \subset \overset{vl}{S}_c$$

$$\overset{vl}{T}{}^{\oplus} = f|_{T^{\oplus}}\left(T^{\oplus}\right) = \{\oslash\} \qquad \overset{vl}{T}{}^{\oplus} \subset \overset{vl}{T}_c$$

L'application d'affectation $<<$ permet de définir les images réciproques $E_{S^{\oplus}}$ et $E_{T^{\oplus}}$ des ensembles $S^{\oplus}$ et $T^{\oplus}$. Les images $E_{S^{\oplus}}$ et $E_{T^{\oplus}}$ sont des sous-ensembles des entités référencées composites E_{S_c} ou E_{T_c} :

$$E_{S^{\oplus}} = <<^{-1}\left(S^{\oplus}\right) \qquad E_{S^{\oplus}} \subset E_{S_c}$$

$$E_{T^{\oplus}} = <<^{-1}\left(T^{v}\right) \qquad E_{T^{\oplus}} \subset E_{T_c}$$

Les singletons de $S^{\oplus}$ et $T^{\oplus}$ permettent quant à eux de raffiner la taxinomie des entités référencées en définissant les images réciproques. Ces dernières sont les ensembles ***des entités référencées à spatialité alternative Point/Ligne***, ***à spatialité alternative Point/Polygone***, etc., sous-ensembles de $E_{S^{\oplus}}$ ou $E_{T^{\oplus}}$:

$$E_{\left\{Point \overset{S}{\oplus} Ligne\right\}} = <<^{-1}\left(\left\{Point \overset{S}{\oplus} Ligne\right\}\right) \qquad E_{\left\{Point \overset{S}{\oplus} Ligne\right\}} \subset E_{S^{\oplus}}$$

$$E_{\left\{Point \overset{S}{\oplus} Polygone\right\}} = <<^{-1}\left(\left\{Point \overset{S}{\oplus} Polygone\right\}\right) \qquad E_{\left\{Point \overset{S}{\oplus} Polygone\right\}} \subset E_{S^{\oplus}}$$

$$E_{\left\{Ligne \overset{S}{\oplus} Polygone\right\}} = <<^{-1}\left(\left\{Ligne \overset{S}{\oplus} Polygone\right\}\right) \qquad E_{\left\{Ligne \overset{S}{\oplus} Polygone\right\}} \subset E_{S^{\oplus}}$$

$$E_{\left\{Instant \overset{T}{\oplus} Période\right\}} = <<^{-1}\left(\left\{Instant \overset{T}{\oplus} Période\right\}\right) \qquad E_{\left\{Instant \overset{T}{\oplus} Période\right\}} \subset E_{T^{\oplus}}$$

Par définition de la spatialité alternative et de la temporalité alternative, les deux spatialités primitives ou les deux temporalités primitives renseignent toutes deux le concept thématique de l'entité référencée.

Les deux exemples du paragraphe suivant montrent l'utilisation des concepts de spatialité alternative ou de temporalité alternative.

IV.2.1.2.b *Exemples d'entité référencée à spatialité alternative ou à temporalité alternative*

Les entités référencées *Bâtiment* et *Feu* (cf. figure 91), souvent utilisées par Yvan Bédard (Bédard, 1998, 1999c), sont parfaitement adaptées à illustration des notions de spatialité alternative et de temporalité alternative.

L'entité référencée $Bâtiment_{<<\boxdot}$ annotée avec le pictogramme $\overset{vl}{s}{}^{\oplus} = \boxdot$ traduit le fait que les instances de cette entité peuvent avoir une spatialité, soit ponctuelle, soit polygonale. ***Dans cet exemple précis, le critère de changement de spatialité est une valeur de surface*** (500 m^2 par exemple) :

- ⇨ Valeur au-dessous de laquelle les instances de l'entité *Bâtiment*<< > n'auront que les coordonnées positionnant le bâtiment dans un référentiel.
- ⇨ Valeur au-dessus de laquelle, le polygone délimitant une instance de l'entité *Bâtiment*<< > sera complètement défini.

Il en est de même pour le pictogramme $t^{\oplus}_{vl}$ = de l'entité référencée *Feu*<< >. La sémantique associée est que lorsque la durée du feu est inférieure à une certaine valeur (10 min par exemple) seule la date ( ) à laquelle a eu lieu le feu est enregistrée alors que, si elle est supérieure, les dates ( ) de début et de fin seront alors toute deux consignées.

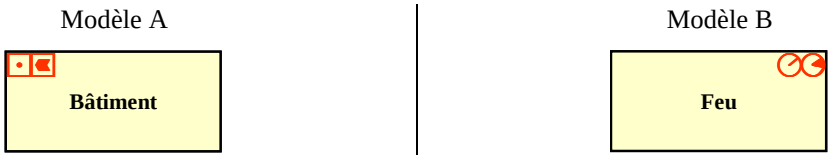


Figure 91 Entités référencées à spatialité alternative (modèle A) ou à temporalité alternative (modèle B).

IV.2.2 *Mono-spatialité et mono-temporalité*

Les concepts de spatialité/temporalité sont des concepts SIG qui résultent de la combinaison d'un concept de spatialité primitive et d'un concept de temporalité primitive. Ceux sont donc des concepts composites impliquant des primitives de natures différentes. Nous verrons que les nouveaux concepts appartiennent à l'ensemble des concepts de spatialité/temporalité $\boldsymbol{ST}$.

IV.2.2.1 Spatialité et temporalité conjointe

Comme pour la spatialité multiple et la temporalité multiple, la spatialité et temporalité conjointe résulte du mécanisme additif entre concept SIG mais avec la différence que la nature des concepts est ici différente.

Au niveau du langage pictogrammique, les membres de l'équipe de Perceptory aurait pu faire un choix similaire à celui des pictogrammes de spatialité multiple ou de temporalité multiple puisque le mécanisme de combinaison est le même. Toutefois, afin de mieux différencier visuellement spatialité et temporalité, ils ont préféré la convention :

Convention Dans une ***pictogrammie de spatialité et temporalité conjointe***, le pictogramme de spatialité primitive est *placé à droite* dans la représentation graphique de l'entité référencée et celui de temporalité primitive *à gauche.*

IV.2.2.1.a *Définition des ensembles*

Nous définissons :

Définition 37 La loi d'addition de spatialité et de temporalité $\overset{ST}{\otimes}$ est une loi de composition commutative de $\boldsymbol{S_p \times T_p}$ dans $\boldsymbol{ST}$ qui combine, suivant un mécanisme additif, les concepts de spatialité primitive et de temporalité primitive du couple (s_p, t_P) pour créer un nouveau concept $st^{\otimes}$:

$$\overset{ST}{\otimes} : \; S_p \times T_p \longrightarrow ST$$
$$(s_p, t_P) \longrightarrow st^{\otimes} = s_p \overset{ST}{\otimes} t_p$$

Nota La loi d'addition de spatialité et de temporalité a été munie de la propriété de commutativité car l'ordre des concepts n'a pas d'importance.

La loi d'addition de spatialité et de temporalité $\overset{ST}{\otimes}$ ainsi définie permet de créer l'ensemble ***des concepts de spatialité et temporalité conjointe*** $ST^{\otimes}$:

$$ST^{\otimes} = \left\{ st \in ST \;\middle|\; st = s_p \overset{ST}{\otimes} t_p \quad avec\ s_p \in S_p \wedge t_p \in T_p \right\} \qquad \text{soit :}$$

$$ST^{\otimes} = \left\{ \begin{array}{ll} Point \overset{ST}{\otimes} Instant, & Point \overset{ST}{\otimes} Période, \\ Ligne \overset{ST}{\otimes} Instant, & Ligne \overset{ST}{\otimes} Période, \\ Polygone \overset{ST}{\otimes} Instant, & Polygone \overset{ST}{\otimes} Période \end{array} \right\}$$

Par construction, les concepts $st^{\otimes}$ appartiennent tous à l'ensemble ST puisqu'ils ont simultanément une spatialité et une temporalité. L'ensemble des concepts de spatialité et temporalité conjointe $ST^{\otimes}$ est donc un sous-ensemble de ST .

$$ST^{\otimes} \subset ST$$

L'ensemble des concepts de spatialité et temporalité conjointe permet de définir, via la restriction $f\big|_{ST^{\otimes}}$, l'ensemble ***des pictogrammes de spatialité et temporalité conjointe*** $\overset{vl}{ST}^{\otimes}$ qui est un sous-ensemble $\overset{vl}{ST}$:

$$\overset{vl}{ST}^{\otimes} = f\big|_{ST^{\otimes}}\left(ST^{\otimes}\right) = \left\{ \begin{array}{llll} \boxdot\!| & \oslash\!| & \boxdot\!| & \circlearrowright\!| \\ \boxtimes\!| & \oslash\!| & \boxtimes\!| & \circlearrowright\!| \\ \blacksquare\!| & \oslash\!| & \blacksquare\!| & \circlearrowright\!| \end{array} \right\}^{72} \qquad \overset{vl}{ST}^{\otimes} \subset \overset{vl}{ST}$$

Comme précédemment, l'application d'affectation $<<$ permet de définir l'image réciproque de $ST^{\otimes}$ dans l'ensemble des entités référencées. L'ensemble $E_{ST^{\otimes}}$ est un sous-ensemble des entités référencées à spatialité/temporalité E_{ST} :

$$E_{ST^{\otimes}} = <<^{-1}\left(ST^{\otimes}\right) \qquad E_{ST^{\otimes}} \subset E_{ST}$$

L'application d'affectation $<<$ permet de raffiner la taxinomie des entités référencées en créant les images réciproques des singletons de $ST^{\otimes}$: l'ensemble ***des entités référencées à spatialité et temporalité conjointe Point/Instant***, ***à spatialité et temporalité conjointe Point/Période***, etc. Tous ces ensembles sont sous-ensembles de $E_{ST^{\otimes}}$:

$$\begin{array}{llll} E_{\left\{Point \overset{ST}{\otimes} Instant\right\}} & = <<^{-1}\left(\left\{Point \overset{ST}{\otimes} Instant\right\}\right) & E_{\left\{Point \overset{ST}{\otimes} Instant\right\}} & \subset E_{ST^{\otimes}} \\ E_{\left\{Point \overset{ST}{\otimes} Période\right\}} & = <<^{-1}\left(\left\{Point \overset{ST}{\otimes} Période\right\}\right) & E_{\left\{Point \overset{ST}{\otimes} Période\right\}} & \subset E_{ST^{\otimes}} \\ \ldots & \ldots & \ldots & \ldots \\ E_{\left\{Polygone \overset{ST}{\otimes} Période\right\}} & = <<^{-1}\left(\left\{Polygone \overset{ST}{\otimes} Période\right\}\right) & E_{\left\{Polygone \overset{ST}{\otimes} Période\right\}} & \subset E_{ST^{\otimes}} \end{array}$$

Par définition de la spatialité et temporalité conjointe, la spatialité primitive et la temporalité primitive renseignent toutes deux le niveau thématique de l'entité référencée.

Dans le paragraphe suivant, un exemple montre l'utilisation de spatialité et temporalité conjointe pour annoter une entité référencée.

---

[72] Les traits droits latéraux symbolisent les limites de la représentation graphique d'une entité référencée.

IV.2.2.1.b *Exemple d'entité référencée à spatialité et temporalité conjointe*

Pour illustrer la Spatialité et Temporalité Conjointe, l'entité référencée *Bâtiment*<<▣<<◔ de la figure 92 est l'exemple même de cette famille. En effet, ce modèle signifie que la spatialité et la temporalité sont découplées et renseignent toutes deux le concept thématique. L'entité référencée *Bâtiment*<<▣<<◔ a une spatialité polygonale (▣), indépendante de la propriété spatiale, et le service du foncier d'une mairie par exemple souhaite disposer de la date de construction de chacun des bâtiments mais aussi leur date de démolition (◔).

Figure 92 Entité référencée à spatialité et temporalité conjointe.

IV.2.2.2 Spatio-temporalité

La spatio-temporalité combine un concept de spatialité primitive avec un concept de temporalité primitive mais, à la différence de la spatialité et temporalité conjointe, les deux concepts SIG primitifs ne renseignent pas tous les deux le niveau thématique de la l'entité référencée. En fait, l'entité référencée est renseignée par la spatialité elle même renseignée par la temporalité. C'est un processus d'annotation en cascade.

Nous verrons d'ailleurs, que les nouveaux concepts ont double statut puisque d'une part, ils sont créés en combinant des concepts SIG primitifs et, d'autre part, ils sont conformes à la définition des entités référencées.

La convention pictogrammique adoptée par Yvan Bédard et son équipe pour exprimer visuellement la spatio-temporalité est :

Convention Dans une ***pictogrammie de spatio-temporalité***, le pictogramme de temporalité primitive est *accolé sans aucun espace à droite* du pictogramme de spatialité primitive.

IV.2.2.2.a *Définition des ensembles*

Nous définissons :

Définition 38 La loi d'affectation de temporalité à une spatialité $\overset{S_T}{<<}$ est une loi de composition non commutative de $S_p \times T_p$ dans ST qui affecte la temporalité primitive du couple (s_p, t_P) à la spatialité primitive pour créer un nouveau concept $s^{<<t}$:

$$\overset{S_T}{<<}: \; S_p \times T_p \longrightarrow ST$$
$$(s_p, t_P) \longrightarrow s^{<<t} = s_p \overset{S_T}{<<} t_p$$

Nota La loi de composition d'affectation de temporalité à une spatialité suit la même convention d'orientation des chevrons que l'application d'affectation $<<$ (cf. I).

La loi de d'affectation de temporalité à une spatialité $\overset{S_T}{<<}$ permet de créer l'ensemble ***des concepts de spatio-temporalité*** S_T :

$$S_T = \left\{ st \in ST \;\middle|\; st = s_p \overset{S_T}{<<} t_p \quad avec\ s_p \in S_p \wedge t_p \in T_p \right\}$$ soit :

$$S_T = \left\{ \begin{array}{ll} Instant \overset{S_T}{>>} Point, & Période \overset{S_T}{>>} Point, \\ Instant \overset{S_T}{>>} Ligne, & Période \overset{S_T}{>>} Ligne, \\ Instant \overset{S_T}{>>} Polygone, & Période \overset{S_T}{>>} Polygone \end{array} \right\}$$

Selon la définition 38, un concept $s^{<<t}$ combine un concept de spatialité primitive avec un concept de temporalité primitive. C'est donc un élément de ST. L'ensemble S_T est donc un sous-ensemble de ST :

$$\forall s^{<<t}, \quad s^{<<t} \in ST \qquad \Rightarrow \qquad S_T \subset ST$$

Par ailleurs, la spatialité de $s^{<<t}$ a une propriété de temporalité aussi, au titre de la définition 3, c'est une entité géographie qui ne peut qu'appartenir qu'à l'ensemble des entités référencées à spatialité/temporalité E_{ST} puisque les autres sous-ensembles ne contiennent que des éléments purement spatiaux ou purement temporels. Donc le nouvel ensemble S_T est aussi sous-ensemble de E_{ST} :

$$\forall s^{<<t}, \quad s^{<<t} \in E_{ST} \qquad \Rightarrow \qquad S_T \subset E_{ST}$$

Ceci montre que l'intersection de l'ensemble des entités référencées et celui des concepts SIG n'est pas vide et que certains concepts sont à la fois concepts SIG et entités référencées.

Cette remarque est particulièrement importante et trouve son utilité dans l'application de la *Transformation de traduction des pictogrammes* aux concepts de spatio-temporalité (cf. Chapitre 5-III.6.1.2).

La restriction $f|_{S_T}$ permet de définir l'ensemble ***des pictogrammes spatio-temporalité*** $\overset{vl}{S}_T$ qui est un sous-ensemble $\overset{vl}{ST}$:

$$\overset{vl}{S}_T = f|_{S_T}(S_T) = \{\boxdot\oslash, \boxdot\circledcirc, \boxtimes\oslash, \boxtimes\circledcirc, \blacksquare\oslash, \blacksquare\circledcirc\} \qquad \overset{vl}{S}_T \subset \overset{vl}{ST}$$

Comme précédemment, l'application d'affectation $<<$ permet de définir l'image réciproque de S_T dans l'ensemble des entités référencées. L'ensemble E_{S_T} est aussi un sous-ensemble des entités référencées à spatialité/temporalité E_{ST} :

$$E_{S_T} = <<^{-1}(S_T) \qquad E_{S_T} \subset E_{ST}$$

L'application d'affectation $<<$ permet de raffiner la taxinomie des entités référencées en créant les images réciproques des singletons de S_T : l'ensemble ***des entités référencées à spatio-temporalité Point/Instant***, ***à spatio-temporalité Point/Période***, etc. Tous ces ensembles sont sous-ensembles de E_{S_T} :

$$\begin{array}{llll} E_{\left\{Instant \overset{S_T}{>>} Point\right\}} & = <<^{-1}\left(\left\{Instant \overset{S_T}{>>} Point\right\}\right) & E_{\left\{Instant \overset{S_T}{>>} Point\right\}} & \subset E_{S_T} \\ E_{\left\{Période \overset{S_T}{>>} Point\right\}} & = <<^{-1}\left(\left\{Période \overset{S_T}{>>} Point\right\}\right) & E_{\left\{Période \overset{S_T}{>>} Point\right\}} & \subset E_{S_T} \\ \ldots & \ldots & \ldots & \ldots \\ E_{\left\{Période \overset{S_T}{>>} Polygone\right\}} & = <<^{-1}\left(\left\{Période \overset{S_T}{>>} Polygone\right\}\right) & E_{\left\{Période \overset{S_T}{>>} Polygone\right\}} & \subset E_{S_T} \end{array}$$

Contrairement aux autres, la temporalité primitive renseigne la spatialité primitive qui renseigne à son tour l'entité référencée à spatio-temporalité. Ici, la spatialité et la temporalité ne sont pas au même niveau

d'affectation car elles s'appliquent suivant un processus en cascade. Le paragraphe suivant illustre, par deux exemples, l'utilisation de la spatio-temporalité.

IV.2.2.2.b *Exemples d'entité référencée à spatio-temporalité*

Les entités référencées *Moissonneuse*<<⊡ <<⊘ et *Épidémie*<<▣ <<⊘ de la figure 93 portent les pictogrammes de spatio-temporalité ⊡⊘ et respectivement ▣⊘ pour pouvoir prendre en compte le fait que la spatialité, *Point* (⊡) et respectivement *Polygone* (▣), évoluent dans le temps.

Par exemple pour l'entité *Moissonneuse*<<▣ <<⊘, le *Système d'Information Géographique* en cours de modélisation va permettre de relever les positions successives de la moissonneuse au cours du travail et, si cette dernière a été équipée d'un capteur de pesage du grain en continue, il sera possible d'établir une cartographie de la variabilité du rendement de la parcelle, information qui pourra être exploitée l'année suivante pour optimiser le rendement de la parcelle.

Les études portant sur l'évolution géographique d'une épidémie sont fondamentalement des entités référencées à spatio-temporalité (*Épidémie*<<▣ <<⊘) puisque l'étendue de la maladie est l'objet même de l'étude. Le pictogramme ▣⊘ répond à ce besoin d'acquisition stockage de données. En effet, il est parfaitement adapté puisque le polygone représentant l'étendue de la maladie pourra être consigné avec la date du relevé. Les épidémiologistes disposeront alors d'une succession de polygones datés représentant la zone affectée par la maladie.

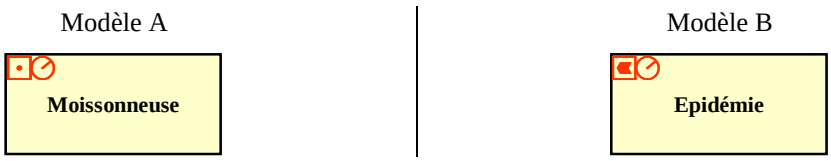


Figure 93 Entités référencées à spatio-temporalité.

IV.3 Macro-spatialité et macro-temporalité

La macro-spatialité et macro-temporalité résultent du troisième constat effectué au début paragraphe selon lequel certaines combinaisons portent sur les entités référencées et non par sur les concepts SIG.

Les premiers échanges au cours de l'analyse d'un *Système d'Information Géographique* font germer les entités référencées facilement identifiables par les acteurs du domaine. Ceux sont souvent des entités référencées qui incorporent d'autres entités référencées ayant un niveau de granularité plus fin. Ces premières entités identifiées sont de véritables conglomérats qui acquièrent de fait les spatialités et les temporalités individuelles des entités référencées formant le conglomérat. Par exemple, les entités référencées *Réseau d'autobus* et *Monuments Historiques*[73] sont des concepts thématiques très généralistes qui, suivant la vision qu'en ont les acteurs, englobent un nombre important de concepts : *Réseaux des circuits, Arrêts (d'Autobus)*, etc. pour la première et *Statue, Rue historique*, etc. pour la seconde. L'existence de ces conglomérats montre qu'il existe des modes d'organisation dans l'ensemble des entités référencées tout comme il y en a dans l'ensemble des concepts SIG. Yvan Bédard a identifié deux modes d'organisation des conglomérats d'entités référencées qui correspondent aux deux relations majeures du langage UML :

- ⇨ La ***composition*** qui consiste à agréger des entités référencées de granularité plus fine dans un conglomérat. De ce fait, la spatialité et la temporalité du conglomérat se déduisent des propriétés individuelles de chacune des entités référencées de granularité plus fine.
- ⇨ La ***généralisation*** qui revient à « factoriser » dans un conglomérat des entités référencées de « plus faible abstraction » ayant des propriétés communes du point de vue de l'application. La spatialité et la temporalité du conglomérat doivent donc « fédérer » toutes les spatialités et temporalités individuelles des entités de plus faible abstraction.

[73] Exemples extraits de (Bédard, 1998, 1999c).

Un résultat important de la description des deux mécanismes de construction des macro-entités référencées est qu'il est toujours possible d'affiner l'analyse d'une macro-entité et de l'« éclater », soit par désagrégation, soit par spécialisation. Nous verrons ces deux processus dans les exemples des paragraphes IV.3.1.3 et IV.3.2.3.

De fait, une macro-entité référencée est un concept thématique qui ***résume*** en son sein tout un ensemble d'entités référencées et qui ***masque*** le modèle détaillé des entités référencées élémentaires.

Nous allons utiliser les deux modes d'organisation identifiés pour définir et construire les ensembles relatifs aux macro-entités référencées ainsi que les concepts d'annotation et les pictogrammes.

Il est à noter que, contrairement aux autres spatialités et temporalités, c'est le mode de raffinement des entités référencées qui pilote celui de la taxinomie SIG et de celle des pictogrammes.

IV.3.1 *Macro-entités référencées d'agrégation - Spatialité complexe et temporalité complexe - spatialité compliquée et temporalité compliquée*

Dans ce paragraphe, nous allons nous intéresser aux conglomérats d'entités référencées dont le mode d'organisation est la composition. Afin d'annoter la spatialité et la temporalité de ces conglomérats, deux approches sont possibles dans Perceptory :

- ⇨ La première est dans le même esprit des spatialités composites précédentes. C'est-à-dire qu'une loi de composition combine les concepts SIG primitifs des entités référencées présentes dans le conglomérat. Ce sont les *Spatialités Complexes* et *Temporalités Complexes*.
- ⇨ La seconde simplifie à l'extrême la pictogrammie puisqu'elle remplace tous les pictogrammes de la *Spatialité Complexe* ou de la *Temporalité Complexe* lorsque leur interprétation devient difficile. Ce sont la *Spatialité Compliquée* et la *Temporalité Compliquée*.

Les paragraphes suivants décrivent ces deux types d'organisation existant dans les conglomérats d'entités référencées.

IV.3.1.1 Définitions

IV.3.1.1.a *Macro-entités référencées d'agrégation*

Nous définissons :

Définition 39 La loi d'agrégation $\overset{A}{\circ}$ est une loi de composition non commutative de $E \times E$ dans E qui combine suivant un mécanisme d'agrégation les entités référencées du couple (e_1, e_2) pour créer une ***macro-entité référencée d'agrégation*** me^A.

$$\overset{A}{\circ} : \quad E \times E \longrightarrow E$$
$$(e_1, e_2) \longrightarrow me^A = e_1 \overset{A}{\circ} e_2$$

Nota La loi d'agrégation est non commutative car, si un concept A est un agrégat et un concept B un composant, l'inverse n'est pas vrai.

La loi d'agrégation $\overset{A}{\circ}$ établie il est alors possible de définir l'ensemble des macro-entités référencées d'agrégation :

$$ME^A = \left\{ e \in E \;\middle|\; e = e_1 \overset{A}{\circ} e_2 \quad avec\ e_1 \in E \wedge e_2 \in E \right\}$$

Les entités référencées étant des concepts fondamentalement composites, ils appartiennent donc à l'ensemble des entités référencées composites E_{U_c} :

$$\forall me^A, \quad me^A \in E_{U_c} \qquad \Rightarrow \qquad ME^A \subset E_{U_c}$$

IV.3.1.1.b *Spatialité complexe et temporalité complexe*

Nous définissons :

Définition 40 La loi d'agrégation spatiale $\overset{A_S}{\circ}$ est une loi de composition non commutative idempotente de $S_p \times S$ dans S qui constitue un conglomérat s° avec les concepts de spatialité primitive (s_p, s) :

$$\overset{A_S}{\circ} : S_p \times S \longrightarrow S$$
$$(s_p, s) \longrightarrow s^\circ = s_p \overset{A_S}{\circ} s$$

Définition 41 La loi d'agrégation temporelle $\overset{A_T}{\circ}$ est une loi de composition non commutative idempotente de $T_p \times T_p$ dans T qui constitue un conglomérat t° avec les concepts de temporalité primitive (t_{p_1}, t_{P_2}) :

$$\overset{A_T}{\circ} : T_p \times T_p \longrightarrow T$$
$$(t_{p_1}, t_{P_2}) \longrightarrow t^\circ = t_{p_1} \overset{A_T}{\circ} t_{p_2}$$

Les lois d'agrégation spatiale et temporelle $\overset{A_S}{\circ}$ et $\overset{A_T}{\circ}$ permettent de définir les ensembles ***des concepts de spatialité complexe*** S° et ***de temporalité complexe*** T° :

$$S^\circ = \left\{ s_c \in S_c \;\middle|\; s_c = s_p \overset{A_S}{\circ} s \quad \textit{avec } s_p \in S_p \wedge s \in S \wedge s_p \neq s \right\} \qquad \text{soit :}$$

$$S^\circ = \left\{ \begin{array}{ll} \textit{Point} \overset{A_S}{\circ} \textit{Ligne}, & \textit{Point} \overset{A_S}{\circ} \textit{Polygone}, \\ \textit{Ligne} \overset{A_S}{\circ} \textit{Polygone}, & \textit{Point} \overset{A_S}{\circ} \textit{Ligne} \overset{A_S}{\circ} \textit{Polygone} \end{array} \right\}$$

$$T^\circ = \left\{ t_c \in T_c \;\middle|\; t_c = t_{p_1} \overset{A_T}{\circ} t_{p_2} \quad \textit{avec } t_{p_1} \in T_p \wedge t_{p2} \in T_p \wedge t_{p_1} \neq t_{p_2} \right\} \qquad \text{soit :}$$

$$T^\circ = \left\{ \textit{Instant} \overset{A_T}{\circ} \textit{Période} \right\}$$

Par définition, ce sont de sous-ensembles de S_c ou de T_c :

$$S^\circ \subset S_c \qquad\qquad T^\circ \subset T_c$$

Il est possible de définir les images réciproques E_{S° et E_{T° des ensembles S° et T° en utilisant l'application d'affectation $<<$. Les ensembles E_{S° et E_{T° sont des sous-ensembles des macro-entités référencées d'agrégation ME^A :

$$E_{S^\circ} = <<^{-1}(S^\circ) \qquad\qquad E_{S^\circ} \subset ME^A$$
$$E_{T^\circ} = <<^{-1}(T^\circ) \qquad\qquad E_{T^\circ} \subset ME^A$$

Les singletons des ensembles S° et T° permettent d'affiner les ensembles des macro-entités référencées en créant les sous-ensembles ***des macro-entités référencées à spatialité complexe Point/Ligne***, ***à spatialité complexe Point/Polygone***, etc. Tous ces ensembles issus des singletons de spatialité complexe ou de temporalité complexe sont des sous-ensembles de $E_{S^{\circ}}$ ou de $E_{T^{\circ}}$:

$$E_{\left\{Point \overset{A_S}{\circ} Ligne\right\}} = <<^{-1}\left(\left\{Point \overset{A_S}{\circ} Ligne\right\}\right) \qquad E_{\left\{Point \overset{A_S}{\circ} Ligne\right\}} \subset E_{S^{\circ}}$$

$$E_{\left\{Point \overset{A_S}{\circ} Polygone\right\}} = <<^{-1}\left(\left\{Point \overset{A_S}{\circ} Polygone\right\}\right) \qquad E_{\left\{Point \overset{A_S}{\circ} Polygone\right\}} \subset E_{S^{\circ}}$$

$$E_{\left\{Ligne \overset{A_S}{\circ} Polygone\right\}} = <<^{-1}\left(\left\{Ligne \overset{A_S}{\circ} Polygone\right\}\right) \qquad E_{\left\{Ligne \overset{A_S}{\circ} Polygone\right\}} \subset E_{S^{\circ}}$$

$$E_{\left\{Point \overset{A_S}{\circ} Ligne \overset{A_S}{\circ} Polygone\right\}} = <<^{-1}\left(\left\{Point \overset{A_S}{\circ} Ligne \overset{A_S}{\circ} Polygone\right\}\right) \qquad E_{\left\{Point \overset{A_S}{\circ} Ligne \overset{A_S}{\circ} Polygone\right\}} \subset E_{S^{\circ}}$$

$$E_{\left\{Instant \overset{A_T}{\circ} Période\right\}} = <<^{-1}\left(\left\{Instant \overset{A_T}{\circ} Période\right\}\right) \qquad E_{\left\{Instant \overset{A_T}{\circ} Périoden\right\}} \subset E_{T^{\circ}}$$

IV.3.1.1.c *Spatialité compliquée et temporalité compliquée*

La spatialité complexe et la temporalité complexe offrent une capacité accrue pour exprimer les propriétés spatiales et temporelles des entités référencées lors de l'analyse d'un *Système d'Information Géographique*.

Toutefois, lorsqu'un nombre élevé de pictogrammes annote une entité référencée, la sémantique suscitée par une multitude de pictogrammes s'avère difficile à interpréter. Il en résulte alors des incohérences pouvant entraîner des dysfonctionnements du *Système d'Information Géographique* final.

Afin de simplifier la sémantique suscitée par une multitude de pictogrammes, l'équipe de Perceptory a introduit la spatialité compliquée et la temporalité compliquée. Ces deux concepts remplacent tous les concepts de spatialité ou de temporalité des entités référencées impliquées dans la macro-entité référencée d'agrégation.

Définition 42 Le ***concept de spatialité compliquée*** s_C est destiné à résumer la composition de la spatialité d'une entité référencée d'agrégation. Il substitue tous les concepts individuels de spatialité des entités référencées impliquées dans la macro-entité référencée d'agrégation. C'est un concept fondamentalement composite aussi il appartient à l'ensemble S_c car il suscite une perception de spatialité pure.

$$s_C \in S_c \quad \Rightarrow \quad \{Spatialité\ Compliquée\} = \{s_C\} \subset S_c$$

Définition 43 Le ***concept temporalité compliquée*** t_C est destiné à résumer la composition de la temporalité d'une entité référencée d'agrégation. Il substitue tous les concepts individuels de temporalité des entités référencées impliquées dans la macro-entité référencée d'agrégation. C'est un concept fondamentalement composite aussi il appartient à l'ensemble T_c car il suscite une perception de temporalité pure.

$$t_C \in T_c \quad \Rightarrow \quad \{Temporalité\ Compliquée\} = \{t_C\} \subset T_c$$

Les deux nouveaux singletons permettent de raffiner l'ensemble ME^A définissant les deux sous-ensembles de macro-entités référencées d'agrégation : celui de spatialité et celui de temporalité :

$$E_{\{s_C\}} = <<^{-1}(\{s_C\}) = \{e_s \in E_S \mid <<(e_s) = s_C\}$$

$$E_{\{t_C\}} = <<^{-1}(\{t_C\}) = \{e_t \in E_T \mid <<(e_t) = t_C\}$$

Les sous-ensembles $E_{\{s_C\}}$ et $E_{\{t_C\}}$ ne constituent pas une partition de ME^A car ces deux concepts ne suffisent pour couvrir toutes les combinaisons de spatialité et/ou de temporalité. En particulier, le cas des macro-entités d'agrégation dont certaines entités référencées ont une temporalité n'est pas pris en compte.

Par définition des concepts de spatialité compliquée s_C et temporalité compliquée t_C, les images réciproques des deux singletons sont inclues dans ME^A :

$$E_{\{s_C\}} \subset ME^A \qquad\qquad E_{\{t_C\}} \subset ME^A$$

Les deux nouveaux concepts s_C et t_C étant définis comme éléments de S_c et de T_c, leur définition ne nécessite aucune redéfinition d'ensembles (S, T et ST).

IV.3.1.2 Pictogrammie : Définitions

IV.3.1.2.a *Spatialité complexe et temporalité complexe*

Comme pour les autres types de spatialité ou de temporalité, l'équipe d'Yvan Bédard applique une convention pictogrammique permettant d'identifier visuellement dans un modèle les macro-entités référencées d'agrégation.

Convention Dans une ***pictogrammie de spatialité complexe***, le pictogramme est une combinaison des pictogrammes individuels des entités référencées *regroupés au sein d'un même rectangle englobant.*

Nota Attention cette convention pictogrammique ne concerne que la pictogrammie de spatialité. Le seul pictogramme de temporalité complexe ne respecte pas cette convention.

Les deux ensembles de concepts de spatialité complexe S° et T° ont des images dans $\overset{vl}{S}$ et $\overset{vl}{T}$ qui sont obtenues par les restrictions $f|_{S^\circ}$ et $f|_{T^\circ}$. Ce sont les ensembles ***des pictogrammes de spatialité complexe*** $\overset{vl}{S}^\circ$ et ***de temporalité complexe*** $\overset{vl}{T}^\circ$, sous-ensembles de $\overset{vl}{S}_c$ et respectivement de $\overset{vl}{T}_c$:

$$\overset{vl}{S}^\circ = f|_{S^\circ}(S^\circ) = \{\boxed{\cdot\,\nearrow},\boxed{\cdot\,\blacktriangleleft},\boxed{\nearrow\,\blacktriangleleft},\boxed{\cdot\,\nearrow\,\blacktriangleleft}\} \qquad \overset{vl}{S}^\circ \subset \overset{vl}{S}_c$$

$$\overset{vl}{T}^\circ = f|_{T^\circ}(T^\circ) = \{\boxed{\text{◔}}\} \qquad \overset{vl}{T}^\circ \subset \overset{vl}{T}_c$$

Il est à noter qu'il n'y qu'un seul pictogramme de temporalité complexe.

IV.3.1.2.b *Spatialité compliquée et temporalité compliquée*

Le nombre de concepts de l'ensemble U_c est infini puisqu'il dépend du nombre de concepts primitifs, du nombre de lois de composition que souhaite exprimer l'auteur du langage, etc. Il est donc impossible de définir en extension ou en compréhension son graphe $G^{f|_{s_C}}$. Par contre, il est possible de définir en extension le graphe de singletons particuliers comme ceux des concepts de spatialité compliquée $\{s_C\}$ ou de temporalité compliquée $\{t_C\}$:

Définition 44 Le graphe $G^{f|_{\{s_C\}}}$ de la restriction $f|_{\{s_C\}}$ de la fonction f est :

$$G^{f|_{\{s_C\}}} = \{(s_C, \boxed{!})\}$$

Définition 45 Le graphe $G^{f|_{\{t_C\}}}$ de la restriction $f|_{\{t_C\}}$ de la fonction f est :

$$G^{f|\{t_C\}} = \{(t_C, \textcircled{!})\}$$

Les graphes $G^{f|\{s_C\}}$ et $G^{f|\{t_C\}}$ de ces singletons sont des sous-ensembles de $G^{f|s_C}$. Les pictogrammes $\overset{sl}{s_C}$ et $\overset{sl}{t_C}$, images des concepts s_C et t_C, appartiennent aux ensembles des pictogrammes de spatialité $\overset{vl}{S_c}$ et de temporalité $\overset{vl}{T_c}$:

$$\overset{vl}{s_C} \in \overset{vl}{S_c} \quad \Rightarrow \quad \{\boxed{!}\} \subset \overset{vl}{S_c} \qquad \text{et :} \qquad \overset{vl}{t_C} \in \overset{vl}{T_c} \quad \Rightarrow \quad \{\textcircled{!}\} \subset \overset{vl}{T_c}$$

Au paragraphe suivant, le même exemple de macro-entité référencée d'agrégation est traité en spatialité complexe et en spatialité compliquée.

IV.3.1.3 Exemple de macro-entité référencée d'agrégation

IV.3.1.3.a *Macro-entité référencée à spatialité complexe*

L'exemple de l'entité référencée *Réseau d'autobus* (Bédard, 1998, 1999c) est parfaitement bien adapté pour illustrer le mécanisme d'agrégation. La description textuelle suivante accompagne l'exemple : l'entité référencée *Réseau d'autobus* *se compose d'une agrégation* *de lignes (réseaux des circuits), de points (arrêts), de polygones (terminus), etc.* Cette description contient deux informations essentielles :

- La première concerne la spatialité de l'entité référencée *Réseau d'autobus*. Sa spatialité inclut la géométrie linéaire du réseau routier, la géométrie ponctuelle des arrêts d'autobus et la géométrie polygonale des terminus.
- La seconde porte sur le mécanisme de construction de l'entité référencée *Réseau d'autobus*. La description stipule qu'elle *se compose d'une agrégation* des entités référencées *Circuit*$_{<<}$, *Arrêt*$_{<<}$ et *Terminus*$_{<<}$. La granularité spatiale de ces dernières est plus faible que celle de l'entité *Réseau d'autobus*.

L'entité référencée *Réseau d'autobus* est l'exemple même d'une entité basée sur le mécanisme d'agrégation décrit plus haut. C'est donc bien une macro-entité référencée d'agrégation dont le modèle « résumé » est celui de la figure 94.

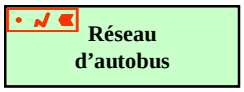


Figure 94 Macro-entité référencée à spatialité complexe.

Au début de l'analyse, le modèle de figure 94 suffit amplement pour représenter le concept thématique majeur qu'est le réseau d'autobus. Cette macro-entité référencée va permettre d'initier la modélisation et va faciliter la communication entre les acteurs du domaine et le concepteur. Mais il est fort probable que le modèle de figure 94 ne soit pas suffisamment détaillé et que des spécifications majeures propres à des entités référencées de plus faible granularité soient absentes. Les acteurs du domaine en collaboration avec le concepteur vont être obligés de décomposer la macro-entité référencée *Réseau d'autobus* et aboutiront au modèle de la figure 95.

Les modèles des figures 94 et 95 sont équivalents. Le premier résume en une seule macro-entité référencée le modèle détaillé au second plan de la figure 95.

L'examen du processus de modélisation montre que le concept SIG complexe n'est pas propre à la macro-entité référencée d'agrégation *Réseau d'autobus* mais au rôle que les acteurs et le concepteur du système lui font jouer au cours de la modélisation. En effet, au démarrage de l'analyse, elle est censée représenter à elle seule un conglomérat de plusieurs entités référencées et de résumer tout un modèle. Par contre en phase avancée d'analyse, elle perd sa spatialité car les entités réellement spatiales sont celles qui la composent.

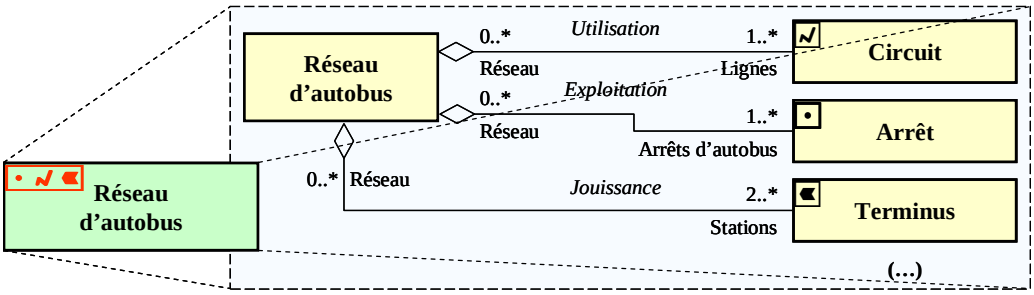


Figure 95 Modèle détaillé de la macro-entité référencée à spatialité complexe.

IV.3.1.3.b *Macro-entité référencée à spatialité compliquée*

Dans l'exemple précédent, la macro-entité référencée d'agrégation *Réseau d'autobus*<< >> est annotée avec le pictogramme SIG complexe . Ce dernier attribue le statut de macro-entité référencée d'agrégation au concept thématique *Réseau d'autobus*<< >>.

En fait, la macro-entité référencée *Réseau d'autobus*<< >> masque un modèle plus détaillé dont les entités élémentaires sont *Circuits*<<, *Arrêts*<< *et Terminus*<<. Il est vrai que la perception suscitée par le pictogramme (cf. figure 96-modèle A) est loin d'être évident à interpréter au premier abord.

C'est ce qui a amené Yvan Bédard à introduire le concept de *Spatialité Compliquée* qui remplace le pictogramme SIG complexe par le pictogramme (cf. figure 96-modèle B).

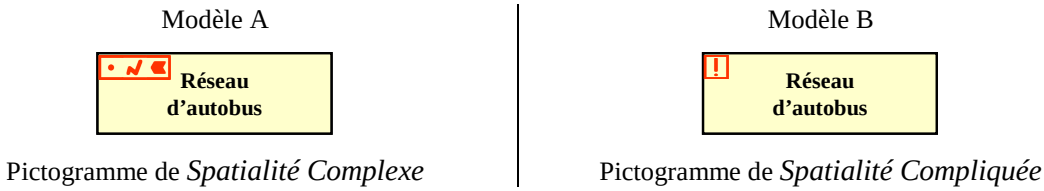


Figure 96 Macro-entité référencée à spatialité compliquée.

Les modèles A et B sont équivalents. La différence entre les deux réside dans le rôle qu'ils jouent, rôle qui est souvent temporaire et qui évolue au cours du développement. Dans les prémices de l'analyse, le modèle B suffit amplement pour représenter le concept majeur du domaine analysé et pour initier un dialogue et des échanges constructifs avec les acteurs du domaine.

Il est fort probable que, l'analyse progressant, le contenu de la macro-entité référencée *Réseau d'autobus*<< > soit précisée et le pictogramme de spatialité compliquée soit remplacé, dans une première étape, par celui de spatialité complexe du modèle A accompagné de préférence d'une description textuelle et, dans une seconde étape, par le modèle détaillé de la figure 95.

IV.3.2 *Macro-entités référencées de généralisation - Spatialité multiforme et temporalité multiforme*

Les conglomérats étudiés dans ce paragraphe sont ceux qui sont construit via le mécanisme de généralisation présenté au début du paragraphe (cf. IV.3).

Les conglomérats créés suivant ce mécanisme sont très hétérogènes. Aussi, l'explicitation de la spatialité ou de la temporalité par composition des spatialités et temporalités des entités référencées de granularité plus fine est difficile ce qui a conduit l'équipe d'Yvan Bédard à résumer cette pictogrammie par deux concepts : la *Spatialité Multiforme* et la *Temporalité Multiforme*.

IV.3.2.1 Définitions

IV.3.2.1.a *Macro-entités référencées de généralisation*

Nous définissons :

Définition 46 La loi de généralisation $\overset{G}{\circ}$ est une loi de composition non commutative et idempotente de $E \times E$ dans E qui combine suivant un mécanisme de généralisation les entités référencées du couple (e_1, e_2) pour créer une ***macro-entité référencée de généralisation*** me^G.

$$\begin{aligned} \overset{G}{\circ} : \quad E \times E &\longrightarrow E \\ (e_1, e_2) &\longrightarrow me^G = e_1 \overset{G}{\circ} e_2 \end{aligned}$$

Nota 1 La loi de généralisation est non commutative car, si un concept A est plus abstrait qu'un concept B, l'inverse est impossible.

Nota 2 La propriété d'idempotence a été retenue pour la loi de généralisation car un concept ne peut pas être plus abstrait que lui-même.

La loi de généralisation $\overset{G}{\circ}$ formulée permet de définir l'ensemble des macro-entités référencées de généralisation :

$$ME^G = \left\{ me^G \in E \;\middle|\; e = e_1 \overset{G}{\circ} e_2 \quad avec\ e_1 \in E \wedge e_2 \in E, \right\}$$

Les entités référencées sont des concepts fondamentalement composites aussi ils appartiennent à l'ensemble E_{U_c} :

$$\forall me^G, \quad me^G \in E_{U_c} \qquad \Rightarrow \qquad ME^G \subset E_{U_c}$$

IV.3.2.1.b *Spatialité multiforme et temporalité multiforme*

Nous définissons :

Définition 47 Le ***concept de spatialité multiforme*** s_M est destiné à résumer la complexité de la spatialité d'une entité référencée de généralisation. Il substitue tous les concepts individuels de spatialité des entités référencées impliquées dans la macro-entité référencée de généralisation. C'est un concept fondamentalement composite aussi il appartient à l'ensemble S_c car il suscite une perception de spatialité pure.

$$s_M \in S_c \qquad \Rightarrow \qquad \{ Spatialité\ Multiforme \} = \{ s_M \} \subset S_c$$

Définition 48 Le ***concept temporalité multiforme*** t_M est destiné à résumer la complexité de la temporalité d'une entité référencée de généralisation. Il substitue tous les concepts individuels de temporalité des entités référencées impliquées dans la macro-entité référencée de généralisation. C'est un concept fondamentalement composite aussi il appartient à l'ensemble T_c car il suscite une perception de temporalité pure.

$$t_M \in T_c \qquad \Rightarrow \qquad \{ Temporalité\ Multiforme \} = \{ t_M \} \subset T_c$$

Les deux singletons permettent de raffiner l'ensemble ME^G créant deux sous-ensembles : l'ensemble des macro-entités référencées de généralisation de spatialité et celui des macro-entités référencées de généralisation de temporalité :

$$E_{\{s_M\}} = <<^{-1}(\{s_M\}) = \{ e_s \in E_S \mid <<(e_s) = s_M \}$$

$$E_{\{t_M\}} = <<^{-1}(\{t_M\}) = \{ e_t \in E_T \mid <<(e_t) = t_M \}$$

Il est à noter que $E_{\{s_M\}}$ et $E_{\{t_M\}}$ ne constituent pas une partition de ME^G puisque, comme pour la spatialité compliqué et la temporalité compliquée, les deux concepts définis ne suffisent pas à couvrir toutes les combinaisons de spatialité et/ou de temporalité possibles et en particulier si des entités référencées de la macro-entité de généralisation ont une temporalité.

Par définition des concepts de spatialité multiforme s_M et temporalité multiforme t_M, les images réciproques des deux singletons sont inclues dans ME^G :

$$E_{\{s_M\}} \subset ME^G \qquad\qquad E_{\{t_M\}} \subset ME^G$$

IV.3.2.2 Pictogrammie : Définitions

Les définitions 47 et 48 ne nécessitent aucunement la redéfinition des ensembles S ou T puisque les nouveaux concepts s_M et t_M ne sont que des éléments parmi d'autres.

Par contre, l'utilisation d'une pictogrammie pour exprimer la spatialité multiforme et la temporalité multiforme dans une représentation graphique conduit à définir le graphe des restrictions $f|_{\{s_M\}}$ et $f|_{\{t_M\}}$ de la fonction f :

Définition 49 Le graphe $G^{f|_{\{s_M\}}}$ de la restriction $f|_{\{s_M\}}$ de la fonction f est :

$$G^{f|_{\{s_M\}}} = \{(s_M, \boxtimes)\}$$

Définition 50 Le graphe $G^{f|_{\{t_M\}}}$ de la restriction $f|_{\{t_M\}}$ de la fonction f est :

$$G^{f|_{\{t_M\}}} = \{(t_M, \circledast)\}$$

Comme pour la spatialité compliquée et la temporalité compliquée, les graphes $G^{f|_{\{s_M\}}}$ et $G^{f|_{\{t_M\}}}$ des singletons $\{s_M\}$ et $\{t_M\}$ sont éléments de $G^{f|_{sC}}$. Les définitions 49 et 50 remplacent dans le reste du document les définitions 44 et 45 du paragraphe IV.3.1.2.

Les pictogrammes $\overset{sl}{s_M}$ et $\overset{sl}{t_M}$, images des concepts s_M et t_M, appartiennent aux ensembles des pictogrammes de spatialité $\overset{vl}{S_c}$ et de temporalité $\overset{vl}{T_c}$ qui, comme leurs images réciproques, n'ont pas besoin d'être redéfinis :

$$\overset{vl}{s_M} \in \overset{vl}{S_c} \quad \Rightarrow \quad \{\boxtimes\} \subset \overset{vl}{S_c} \qquad \text{et :} \qquad \overset{vl}{t_M} \in \overset{vl}{T_c} \quad \Rightarrow \quad \{\circledast\} \subset \overset{vl}{T_c}$$

Le paragraphe suivant présente un exemple de macro-entité référencée de généralisation et l'évolution possible de cette macro-entité au cours du processus de modélisation.

IV.3.2.3 Exemple de macro-entité référencée à spatialité multiforme

L'exemple illustrant le mécanisme de généralisation est aussi cité par Yvan Bédard (Bédard, 1999c) dans ces communications. Il est construit autour de l'entité référencée *Monuments Historiques* qui peut être soit une *Statue*, soit une *Rue historique*, soit un *Parc*, soit encore un *Quartier Historique*, etc. La description de l'entité référencée *Monuments Historiques* informe sur :

- Sa spatialité : la localisation ( ) est la propriété intéressante afin de pouvoir situer les objets *Statue*, comme l'est le tracé ( ) pour les objets *Rue historique*, l'emprise au sol ( ) pour les objets *Parc*, la géométrie complexe ( ) des objets *Quartier Historique* ( ), etc. L'énumération étant inachevée et vu la multitude de monuments existants, tout laisse à penser qu'il faut envisager tous les types de spatialité.
- Son mécanisme de construction. L'énumération des concepts thématiques laisse transparaître que l'entité *Monuments Historiques* est un ensemble d'entités référencées qui n'ont pas forcément de relations entre-elles : *Statue*<<, *Rue historique*<<, *Parc*<<, *Quartier Historique*<<, etc. Le mécanisme de généralisation est bien sous-jacent dans cette description.

L'entité référencée *Monuments Historiques* est donc une macro-entité référencée dont le modèle de structuration sous-jacent est basé sur le mécanisme de généralisation/spécialisation.

Au début de l'analyse, le modèle de cette description pourrait être résumé par la seule classe *Monuments Historiques* comme le montre le modèle A de la figure 97. Face à ce modèle, il est peu probable que quiconque analysant le modèle soit alerté de l'intérêt des propriétés spatiales des différents types de *Monuments Historiques* et que ces propriétés sont une caractéristique essentielle pour les acteurs du domaine.

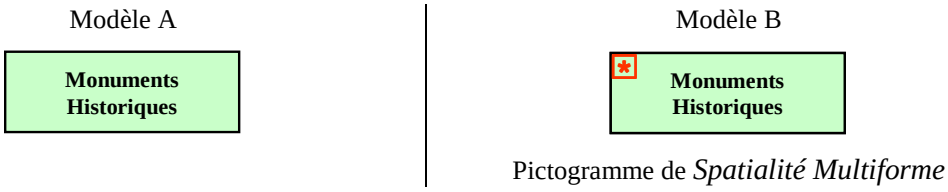


Figure 97 Macro-entité référencée à spatialité multiforme.

Tel quel, le concept *Monuments Historiques* du modèle A ne présente pas les caractéristiques d'une entité référencée, c'est-à-dire posséder une spatialité et/ou une temporalité (cf. définition 3). Quiconque confronté à ce modèle est en droit de se demander s'il agit d'une entité référencée. En effet, le modèle A pourrait très bien être utilisé pour réaliser une application de gestion documentaire ou financière des monuments sans aucun besoin de localisation.

Le pictogramme de spatialité multiforme du modèle B introduit toute la différence. Il indique en premier lieu que *Monuments Historiques*<< a le statut d'une entité référencée, en deuxième lieu que c'est une macro-entité référencée de généralisation et donc qu'elle représente un ensemble d'entités référencées de type très différents et, enfin en troisième lieu, que le concepteur doit attacher une grande importance à la spatialité de cette macro-entité ou des entités référencées la composant. L'intérêt de la spatialité multiforme est d'introduire une information qui impose des obligations aux acteurs du domaine mais aussi au concepteur. Tout l'intérêt est là.

Concentrer toute l'information contenue dans la description textuelle du début en une seule classe est très réducteur. Aussi, il est évident dès que les acteurs du domaine souhaiteront préciser certains détails, la macro-entité référencée *Monuments Historiques* sera obligatoirement éclatée et les modèles détaillés des deuxième et troisième plans de la figure 98 seront réalisés.

La macro-entité référencée de généralisation *Monuments Historiques*<< sera alors spécialisée par des entités référencées (*Statue*<<, *Rue historique*<<, *Parc*<<) voire d'autres macro-entités (*Quartier Historique*<<) et, si cela est nécessaire, ces dernières seront à leur tour raffinées ou décomposées. Dans la figure 98, l'entité référencée *Quartier Historique*<< est une macro-entité référencée complexe décomposable en entités plus élémentaires telles que *Rue*<<, *Place*<<, etc.

Le processus de raffinement et de décomposition décrit ici montre que l'existence des macro-entités référencées *Monuments Historiques*<< et *Quartier Historique*<< est fortement conditionnée au bon vouloir des acteurs du domaine et des besoins de représentation détaillée ou non. En fait, les macro-entités utilisées au début de la modélisation ne sont souvent qu'une étape dans le processus de modélisation.

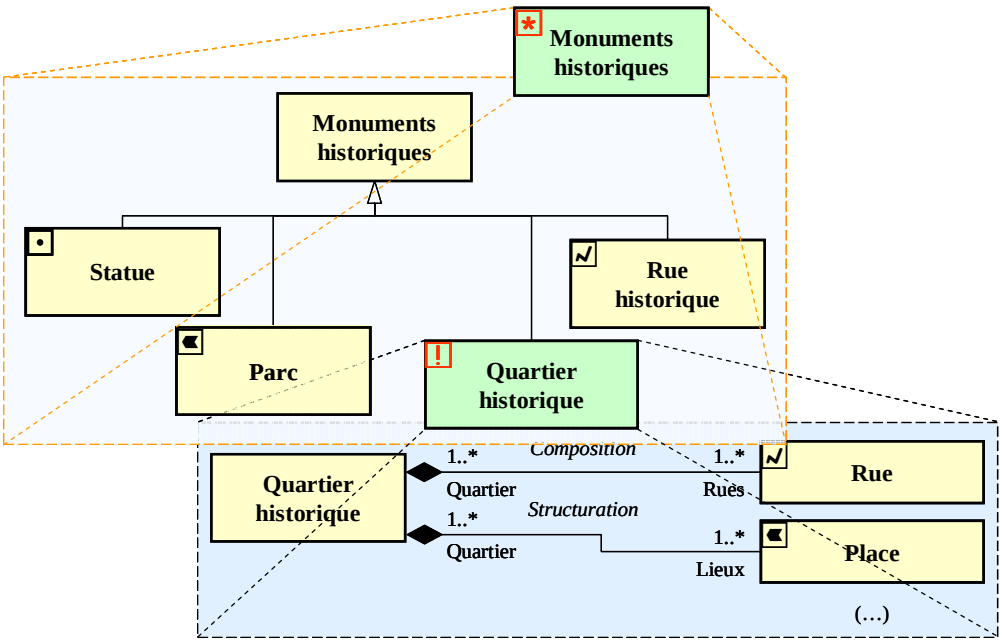


Figure 98 Modèle détaillé de la macro-entité référencée à spatialité multiforme.

IV.3.3 *Ensemble des macro-entités référencées : Définition*

Dans la même logique de ce chapitre, nous définissons l'union des macro-entités référencées des paragraphes IV.3.1 et IV.3.2 :

Définition 51 L'ensemble des ***macro-entités référencées*** $\boldsymbol{ME}$ est l'union des deux ensembles des macro-entités référencées d'agrégation $\boldsymbol{ME^A}$ et de généralisation $\boldsymbol{ME^G}$.

$$\boldsymbol{ME = ME^A \cup ME^G}$$

Étant donné que les ensembles des macro-entités référencées d'agrégation $\boldsymbol{ME^A}$ et de généralisation $\boldsymbol{ME^G}$ sont des sous-ensembles de celui des entités composites $\boldsymbol{E_{U_c}}$, l'ensemble des macro-entités référencées $\boldsymbol{ME}$ est inclus dans l'ensemble entités composites $\boldsymbol{E_{U_c}}$:

$$\left.\begin{array}{l} \boldsymbol{ME^A \subset E_{U_c}} \\ \boldsymbol{ME^G \subset E_{U_c}} \end{array}\right\} \quad \Rightarrow \quad \boldsymbol{ME \subset E_{U_c}}$$

IV.4 Multi-spatialité et multi-temporalité

La macro-spatialité et macro-temporalité résultent du quatrième constat effectué au début du paragraphe qui stipule que certains concepts sont obtenus par répétition du même concept primitif.

Les ensembles étudiés dans ce paragraphe correspondent au troisième constat mentionné au début du paragraphe IV évoquant la répétition d'un concept de spatialité et/ou de spatialité.

En cours d'analyse, il n'est pas rare de rencontrer des entités référencées telles que *Prélèvements Ponctuels* dont la propriété spatiale ou temporelle est constituée de la répétition d'une propriété primitive. L'entité *Prélèvements Ponctuels* est un bon exemple pour illustrer cette problématique.

Sa spatialité est suscitée par le mot *Ponctuels* qui indique d'une part, que le concept de spatialité primitive est *Point* et, d'autre part, qu'il y en a plusieurs puisque il est au pluriel. C'est un concept de spatialité composite même si la nature du concept de la spatialité élémentaire est simple.

Bien entendu, il est important d'exprimer la multi-spatialité et/ou la multi-temporalité dans les modèles en synergie avec les différents types de pictogrammes. La convention suivante a été définie par l'équipe d'Yvan Bédard :

Convention La cardinalité est *accolée à droite du pictogramme* qu'elle renseigne.

Nota Dans le cas d'une pictogrammie composite à plusieurs pictogrammes, cette convention sous-entend que la cardinalité relative au premier pictogramme elle est placée entre les deux pictogrammes.

IV.4.1 *Définitions*

Nous définissons :

Définition 52 La loi de cardinalité générale $\overset{G}{*}$ est une loi de composition externe commutative de $\mathbb{N}\times U$ dans U qui combine, suivant un mécanisme additif, λ fois le concept SIG primitif du couple (λ,u) pour créer un nouveau concept u_λ :

$$\begin{array}{rccl} \overset{G}{*}: & \mathbb{N}\times U & \longrightarrow & U \\ & (\lambda,u) & \longrightarrow & u_\lambda = \lambda \overset{G}{*} u \end{array}$$

Pour les valeurs de $\lambda = 0$ et $\lambda = 1$, cette la loi de cardinalité générale ne présente pas un grand intérêt du point de vue ensembliste mais sa définition a été énoncée afin de pouvoir annoter les concepts SIG et les pictogrammes de la cardinalité optionnelle du langage UML qui sera introduite lors de la dérivation des taxinomies en métamodèle (cf. VII). Pour les besoins ensembliste, nous définissons une nouvelle loi excluant les valeurs de $\lambda = 0$ et $\lambda = 1$:

Définition 53 La loi de cardinalité $*$ est une loi de composition externe commutative de $\mathbb{N}-[0;1]\times U$ dans U^λ qui combine, suivant un mécanisme additif, λ fois le concept SIG primitif du couple (λ,u) pour créer un nouveau concept u_λ :

$$\begin{array}{rccl} *: & \mathbb{N}-[0;1]\times U & \longrightarrow & U^\lambda \\ & (\lambda,u) & \longrightarrow & u^\lambda = \lambda * u \end{array}$$

Nota Pour $\lambda = 2$ et $u \in S_p$ ou $u \in T_p$, la loi de cardinalité $*$ traite les cas exclus par les lois d'addition spatiale et d'addition temporelle (cf. définitions 33 et 34).

La loi de cardinalité $*$ permet de définir de façon générale l'ensemble ***des concepts multi-spatialité et/ou multi-temporalité*** U_λ :

$$U^\lambda = \left\{ u^\lambda \in U \;\middle|\; \lambda \in \mathbb{N}-[0;1] \wedge u \in U,\quad u^\lambda = \lambda * u \right\}$$

La loi de cardinalité $*$ peut être appliquée à chacun des sous-ensembles de U et donner ainsi l'ensemble des concepts de multi-spatialité et/ou multi-temporalité associé :

- ⇨ À S_p va correspondre l'ensemble des concepts de multi-spatialité primitive S_p^λ.
- ⇨ À S va correspondre l'ensemble des concepts de multi-spatialité S^λ.
- ⇨ À ST va correspondre l'ensemble des concepts de multi-spatialité/temporalité ST^λ.
- ⇨ À $T^\otimes$ va correspondre l'ensemble des concepts de multi-temporalité multiple $T^{\otimes\lambda}$.

⇨ À $\{Ligne\}$ va correspondre l'ensemble des concepts de spatialité multi-Ligne $\{Ligne\}^{\lambda}$.

⇨ Etc.

Une propriété importante de tous ces ensembles est qu'ils sont sous-ensembles de l'ensemble composite associé à l'ensemble « générateur » comme cela est démontré ci-après. Soit $*\big|_{S_p}$ la restriction de la loi de cardinalité $*$ à l'ensemble S_p, alors il est possible d'écrire :

$$\forall s_p, \forall \lambda, \quad s^{\lambda} = \lambda *\big|_{S_p} s_p = \underbrace{s_p \overset{S}{\otimes} s_p \ldots s_p \overset{S}{\otimes} s_p}_{\lambda} \quad \Rightarrow \quad s^{\lambda} \in S_c \quad \Rightarrow \quad S^{\lambda} \subseteq S_c$$

Soit $*\big|_{S_c}$ la restriction de la loi de cardinalité $*$ à l'ensemble S_c et puisque s_c est combinaison d'au moins de concepts SIG primitifs, il est possible d'écrire :

$$\forall s_c, \forall \lambda, \quad s^{\lambda} = \lambda *\big|_{S_c} s_c \quad \Rightarrow \quad s^{\lambda} \in S_c \quad \Rightarrow \quad S^{\lambda} \subset S_c$$

Bien évidemment, tous ces ensembles ont une image dans l'ensemble des pictogrammes et une image réciproque dans celui des entités référencées. Par souci de simplification, les images et images réciproques ne seront pas énumérées ici.

IV.4.2 *Exemple d'entité référencée à spatialité multipoints*

L'entité référencée *Prélèvements Ponctuels*$_{<<\boxdot 10}$ annotée du pictogramme ⊡ suivi de la valeur 10 (cf. figure 99) signifie que chaque échantillon du plan d'échantillonnage modélisé est constitué de *10 et exactement 10* prélèvements effectués en 10 points de coordonnées connues ou mesurées.

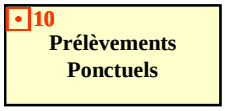


Figure 99 Entité référencée à spatialité multipoints.

IV.5 En résumé

Les concepts SIG composites ouvrent une capacité d'expression quasi-infinie pour représenter la spatialité ou la temporalité des entités référencées au cours de l'analyse d'un *Système d'Information Géographique*. Ce constat est d'autant plus vrai que les auteurs des méthodes, des formalismes, etc. peuvent définir les lois de composition qu'ils souhaitent afin d'exprimer une problématique.

Cette capacité d'expression a son revers de la médaille. En effet, la sémantique devient de plus en plus difficile à appréhender avec l'augmentation du nombre de lois de composition adoptées. Aussi, il est essentiel que le concepteur utilisant un atelier de génie logiciel doté des concepts SIG maîtrise parfaitement les concepts composites et, à travers eux, les lois de composition. Sans cette maîtrise, des risques de dysfonctionnements du *Système d'Information Géographique* final ou de l'application informatique développée sont à craindre.

Pour les bi-spatialités, les bi-temporalités, les multi-spatialités et les multi-temporalités, ce sont les modes de combinaison des propriétés spatiales et temporelles qui ont permis de définir les lois de composition entre concepts SIG primitifs alors que, pour les macro-spatialités et les macro-temporalités, ce sont les lois de compositions entre entités référencées qui ont permis de définir les concepts de spatialité et de temporalité.

La fonction f étant bijective, les taxinomies SIG et des pictogrammes sont totalement identiques. Par contre au terme de ce paragraphe, la taxinomie SIG présente des différences avec celles des entités référencées. Bien qu'ils auraient pu être imaginés, il n'existe aucun concept dans la taxinomie SIG image de l'ensemble des macro-entités référencées ME, ni de celui des macro-entités référencées d'agrégation ME^{A} ni enfin de celui des macro-entités référencées de généralisation ME^{G} (cf. figure 100).

Cette différence est normale puisque ce sont les lois de composition des entités référencées qui ont amenées Yvan Bédard à définir des concepts SIG et des pictogrammes permettant d'annoter les macro-entités référencées. Il est à noter que les trois ensembles $\boldsymbol{ME}$, $\boldsymbol{ME}^{\boldsymbol{A}}$ et $\boldsymbol{ME}^{\boldsymbol{G}}$ constituent une grille de lecture complémentaire des entités référencées SIG composites puisque ce sont des sous-ensembles de $\boldsymbol{E}_{U_c}$ (cf. figure 100).

Cette différence est confirmée par les concepts spatio-temporalités. En effet, ces derniers ont été créés comme étant une combinaison de concepts SIG primitifs. C'est la loi d'affection de temporalité à une spatialité $\boldsymbol{s}_T$ << qui a permis de la définir. Toutefois, la définition de ces concepts leur attribue aussi le statut d'entité référencée (cf. IV.2.2.2). Ce sont donc des concepts qui appartiennent aux deux taxinomies comme le montre la figure 100.

V SPATIALITE DERIVEE ET TEMPORALITE DERIVEE

Dans le domaine de l'*Information Géographique*, les notions de spatialité dérivée et de temporalité dérivée sont particulièrement intéressantes pour exprimer au cours de la modélisation que les acteurs intéressent à la spatialité ou à la temporalité d'un concept thématique dont les propriétés spatiales et temporelles ne lui sont pas propres mais qu'elles peuvent être déduites de celles d'entités référencées avec lesquelles il est en relation.

La notion de dérivation a été empruntée par Yvan Bédard et son équipe au langage UML (OMG, 2003a) et adapté au domaine de l'*Information Géographique* (Proulx *et al.*, 2002). Ils ont conservé la même sémantique c'est-à-dire que la spatialité dérivée ou la temporalité dérivée d'une entité référencée est le résultat d'un traitement numérique portant sur les spatialités et/ou les temporalités d'entités référencées autre que celle annotée avec le concept de dérivation.

Dans le langage UML, la notion de dérivation est symbolisée par une barre oblique (slash) placée devant les attributs dérivés ou des rôles dérivés des relations d'association. L'idée d'inclinaison a été reprise par l'équipe d'Yvan Bédard pour différencier les pictogrammes « simples » des pictogrammes « dérivés ». La convention pictogrammique utilisée est donc :

Convention Dans la représentation graphique d'une ***pictogrammie dérivée***, les pictogrammes sont *inclinés* par similarité avec la barre oblique du langage UML.

V.1 Concept dérivé : Définition

Définition 54 Un ***concept SIG dérivé*** $\boldsymbol{u}$, annotant une entité référencée $\boldsymbol{e}$, est un concept SIG indiquant que la spatialité et/ou la temporalité qu'il suscite sont déduites par traitement numérique des spatialités et/ou des temporalités d'entités référencées $(\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_i, \ldots)$ autres que $\boldsymbol{e}$.

Cette définition montre explicitement que l'entité référencée annotée avec un concept de spatialité dérivée ou de temporalité dérivée n'est pas *« propriétaire »* de ces propriétés spatiales et temporelles qui sont l'essence même d'une entité référencée au sens de la définition 3. Par contre, la spatialité et la temporalité peuvent être reconstruites autant que de besoin. Bien évidemment, lorsque le processus de reconstruction de la spatialité ou de la temporalité est long, elles sont rendues persistantes, par exemple dans une base de données, et attachées à l'entité référencée annotée du concept de dérivation. Ce n'est pas pour autant qu'elles lui appartiennent. Dans un tel contexte, il est primordial qu'un processus de mise à jour soit clairement défini et de préférence automatisé.

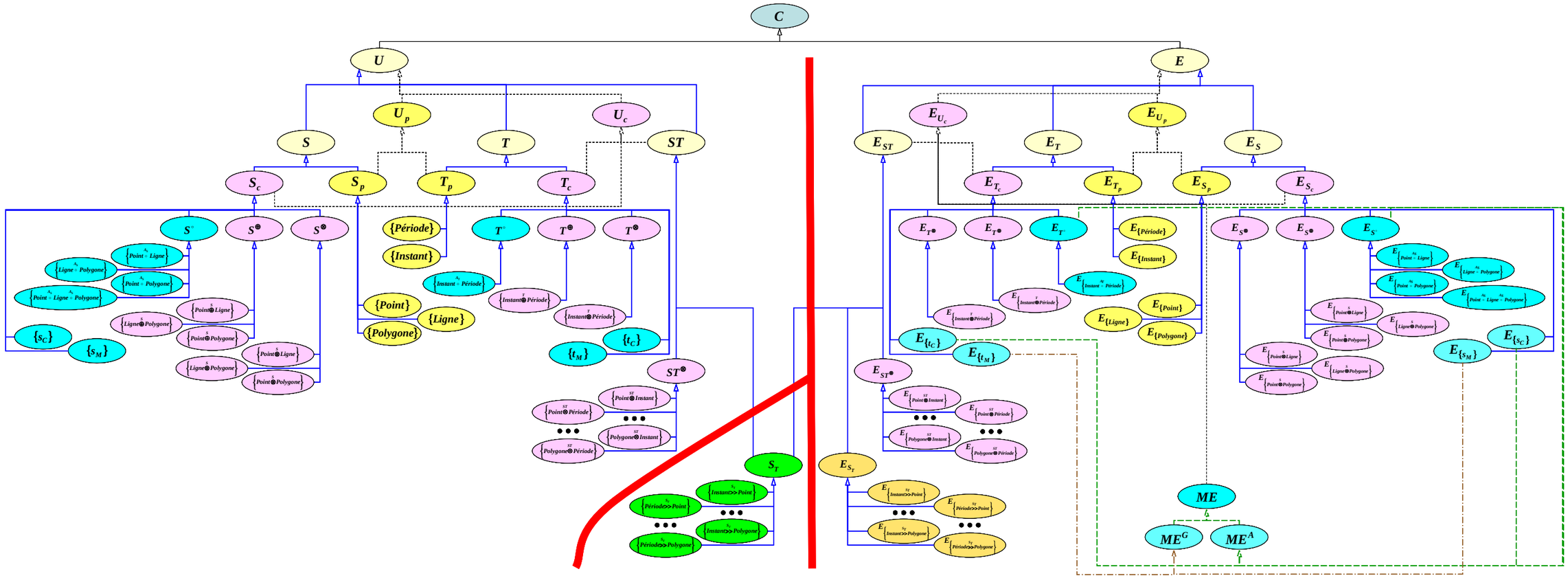


Figure 100 Représentation des taxinomies SIG et des entités référencées suivant le formalisme hybride[74].

[74] Afin de simplifier le diagramme, tous ensembles relatifs aux multi-spatialités et multi-temporalités n'ont pas été représentés car cela aurait doublé le nombre d'ensembles.

V.2 Redéfinition des ensembles[75]

V.2.1 *Primitifs*

L'application de la définition 54 aux deux ensembles des concepts primitifs S_p et T_p a pour conséquence de doubler les concepts SIG primitifs d'un concept dual dérivé. Aussi, les définitions 19 et 20 des ensembles S_p et T_p ne sont plus valables et doivent être reformulées en incluant les concepts dérivés.

Vu le petit nombre de concepts SIG primitifs (trois spatiaux et deux temporels), il est assez facile de redéfinir en extension les deux ensembles des concepts primitifs S_p et T_p :

Définition 55 L'ensemble des ***concepts de spatialité primitive*** S_p est défini par :

$$S_p = \begin{Bmatrix} \textit{Point}, & \textit{Ligne}, & \textit{Polygone}, \\ \textit{Point Dérivé}, & \textit{Ligne Dérivée}, & \textit{Polygone Dérivé} \end{Bmatrix}$$

Définition 56 L'ensemble des ***concepts de spatialité primitive*** T_p est défini par :

$$T_p = \begin{Bmatrix} \textit{Instant}, & \textit{Période}, \\ \textit{Instant Dérivé}, & \textit{Période Dérivée} \end{Bmatrix}$$

Nota 1 Nous aurions pu définir dans chacun des ensembles S_p et T_p deux sous-ensembles : l'un spécifique aux concepts « primitifs simples » et l'autre consacré aux concepts « primitifs dérivés ». Pour l'instant, nous n'avons pas jugé indispensable de définir et de nommer des ensembles spécifiques aux concepts dérivés.

Nota 2 Les nouveaux concepts de spatialité dérivée et de temporalité dérivée n'ont pas nécessité la reformulation des définitions relatives aux ensembles S, T et U car leurs énoncés sont suffisamment généraux pour inclure ces cinq nouveaux concepts (cf. définitions 13, 14 et 16).

Les nouvelles définitions 55 et 56 des ensembles des concepts primitifs S_p et T_p remplaceront dans reste du document les définitions 19 et 20 énoncées en III.1.1. La redéfinition des ensembles S_p et T_p implique de redéfinir d'une part, des graphes de la fonction f et, d'autre part, les ensembles des pictogrammes primitifs.

Le nombre de concepts de spatialité primitive et de temporalité primitive étant restreint, il est possible de définir une à une les correspondances entre concepts de spatialité primitive et pictogrammes associés, correspondances qui constituent les graphes des deux fonctions $f|_{S_p}$ et $f|_{T_p}$:

Définition 57 Le graphe $G^{f|_{S_p}}$ de la fonction $f|_{S_p}$ est :

$$G^{f|_{S_p}} = \begin{Bmatrix} (\textit{Point}, \boxdot), & (\textit{Ligne}, \boxtimes), & (\textit{Polygone}, \blacksquare), \\ (\textit{Point Dérivé}, \boxdot), & (\textit{Ligne Dérivée}, \boxtimes), & (\textit{Polygone Dérivé}, \blacksquare) \end{Bmatrix}$$

Définition 58 Le graphe $G^{f|_{T_p}}$ de la fonction $f|_{T_p}$ est :

[75] Tous les nouveaux ensembles relatifs aux concepts dérivés de ce paragraphe viennent compléter ou enrichir les taxinomies SIG, des pictogrammes et des entités référencées. Au terme du paragraphe, la figure 100 devrait être reprise en intégrant ces nouveaux ensembles mais, étant donné que le diagramme serait surchargé, elle n'a pas été redessinée.

$$G^{f|_{T_p}} = \begin{Bmatrix} (Instant, \oslash), & (Période, \ominus), \\ (Instant\ Dérivé, \oslash), & (Période\ Dérivée, \ominus) \end{Bmatrix}$$

Les graphes des fonctions $f|_{S_p}$ et $f|_{T_p}$ étant définis, les deux ensembles des pictogrammes SIG primitifs $\overset{vl}{S}_p$ et $\overset{vl}{T}_p$ peuvent être à leur tour redéfinis ainsi que leur union $\overset{vl}{U}_p$:

Définition 59 L'ensemble des ***pictogrammes de spatialité primitive*** $\overset{vl}{S}_p$ est l'image de S_p par la fonction $f|_{S_p}$.

$$\overset{vl}{S}_p = f|_{S_p}(S_p) = \begin{Bmatrix} \boxdot, & \boxslash, & \boxleftarrow, \\ \boxdot, & \boxslash, & \boxleftarrow \end{Bmatrix}$$

Définition 60 L'ensemble des ***pictogrammes de temporalité primitive*** $\overset{vl}{T}_p$ est l'image de T_p par la fonction $f|_{T_p}$.

$$\overset{vl}{T}_p = f|_{T_p}(T_p) = \begin{Bmatrix} \oslash, & \ominus, \\ \oslash, & \ominus \end{Bmatrix}$$

Définition 61 L'ensemble des ***pictogrammes SIG primitifs*** $\overset{vl}{U}_p$ est l'union des ensembles $\overset{vl}{S}_p$ et $\overset{vl}{T}_p$.

$$\overset{vl}{U}_p = \overset{vl}{S}_p \cup \overset{vl}{T}_p \quad \Rightarrow \quad \overset{vl}{U}_p = \begin{Bmatrix} \boxdot, & \boxslash, & \boxleftarrow, & \oslash, & \ominus, \\ \boxdot, & \boxslash, & \boxleftarrow, & \oslash, & \ominus \end{Bmatrix}$$

Comme pour les concepts SIG, les définitions de ces trois nouveaux ensembles remplacent celles énoncées en III.1.2 (définitions 24, 25 et 26).

V.2.2 *Composites*

Les lois de composition s'appliquant aussi aux nouveaux concepts, il faudrait, afin de compléter les trois taxinomies, redéfinir le contenu des tous les ensembles de concepts SIG composites $S^{\otimes}$, $T^{\otimes}$, $S^{\oplus}$, $T^{\oplus}$, $ST^{\oplus}$, S_T, S° et T°, de leurs images dans l'ensemble des pictogrammes et des images réciproques dans l'ensemble des entités référencées en incluant aussi les images réciproques des singletons.

Bien que cela à été réalisé lors de l'implémentation les concepts SIG et les pictogrammes associés dans le *Profil UML-SIG*, nous n'avons pas jugé utile de faire des listes de noms de concepts composites ou des listes de pictogrammes composites. C'est la raison pour laquelle, les nouveaux ensembles ne sont pas redéfinis ici.

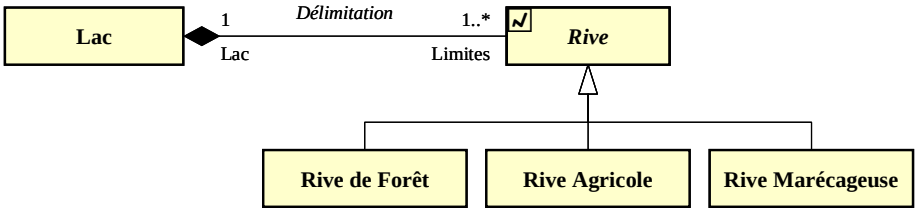
Le paragraphe suivant montre un exemple d'utilisation de la spatialité dérivée au cours de la modélisation d'une application ou d'un *Système d'Information Géographique*.

V.3 Exemple d'entité référencée à spatialité dérivée

Le modèle A de la figure 101 montre un concept thématique *Lac* sans spatialité or ce concept suscite une perception de spatialité polygonale. Au titre de la définition 3, le concept *Lac* n'est pas une entité référencée. En effet, même si d'autres concepts associés à *Lac* sont des entités référencées (*Rive*$_{<<\boxslash}$), le concept *Lac* pourrait

être utilisé dans une application ne s'intéressant pas à la spatialité. Par exemple, une base de données recensant les noms les lacs d'une région ou d'un pays.

Modèle A - Sans pictogramme de spatialité dérivée



Modèle B - Avec pictogramme de spatialité dérivée

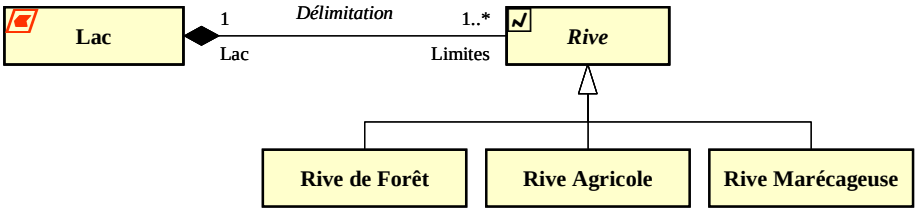


Figure 101 Illustration de la notion de concept SIG dérivé.

Par contre, il est évident que, face au modèle A, un acteur quel qu'il soit suggérera le développement d'une fonctionnalité permettant de déterminer, par un traitement numérique, le polygone délimitant la zone couverte par un lac puisque l'information existe au sein des entités référencées *Rive*<<▣ . Renseigner le concept de *Lac* avec le pictogramme de polygone dérivé (▣) entre dans cette logique. L'avantage d'introduire cette annotation supplémentaire est :

- D'une part, que le concept de *Lac*<<▣ acquière le statut d'entité référencée ce qui n'était pas le cas dans le modèle A. Cela suppose que sa propriété spatiale devient une caractéristique fondamentale dont il faudra se soucier lors de l'implémentation de *Système d'Information Géographique* mais aussi en termes d'acquisition de données.
- Et, d'autre part, que le pictogramme dérivé représente une information interprétable qui permet d'améliorer le processus de modélisation, point que nous aborderons au paragraphe VIII.1.

V.4 En résumé

Les concepts de *Spatialité dérivée* et de *Temporalité dérivée* sont des concepts qui complètent la panoplie des concepts primitifs et composites. Ils permettent d'augmenter la capacité d'expression des propriétés spatiales et temporelles des entités référencées ce qui rend la modélisation des *Systèmes d'Information Géographique* plus aisée.

Cependant, les concepts dérivés ne sont pas de même nature que les précédents car ils ne peuvent pas être réifiés en *Patron de conception SIG* nous le verrons au Chapitre 5-III.3.1.

Nous verrons aussi au paragraphe VIII.1 que la présence des concepts dérivés constitue une information qui peut être utilisée pour piloter le processus de modélisation des *Systèmes d'Information Géographique*, information qui va permettre de concevoir des fonctionnalités améliorant le processus de modélisation.

Les notions de spatialité dérivée et de temporalité dérivée s'appliquent à tout concept SIG qu'il soit primitif ou composite. Toutefois, l'utilisation de cette notion avec des concepts composites peut devenir rapidement problématique car les concepts dérivés risquent d'être sujets à interprétations multiples. La qualité du modèle peut en souffrir. Aussi, il est conseillé de mettre en œuvre la spatialité dérivée et la temporalité dérivée avec prudence.

Afin de ne pas surcharger le diagramme, la figure 100 n'a pas été réactualisée avec les concepts de spatialité dérivée et de temporalité dérivée.

VI SPATIALITE INCONNUE ET TEMPORALITE INCONNUE[76]

Lorsque les acteurs du domaine étudié et le concepteur sont en train d'établir les spécifications d'un *Système d'Information Géographique*, il n'est pas rare que la spatialité ou la temporalité de certaines entités référencées soient inconnues au moment de la modélisation (cf. l'exemple de l'entité référencée *Coin de Nature Vierge* en VI.3). Toutefois, ce sont des propriétés qui intéressent les acteurs du domaine.

Les concepts de *Spatialité inconnue* et de *Temporalité inconnue* ont été introduits par Yvan Bédard afin de proposer une solution à ce problème.

Ces deux concepts permettent soit d'exprimer l'intérêt porté à la spatialité et à la temporalité d'un concept thématique alors qu'elles sont réellement inconnues soit de renseigner sur le champ l'existence d'une spatialité ou d'une temporalité dont la nature exacte fera l'objet d'une recherche ultérieure.

VI.1 Concept de spatialité inconnue et de temporalité inconnue : Définitions

Définition 62 Le ***concept de spatialité inconnue*** s_I est destiné à renseigner la méconnaissance ou l'absence d'information sur la spatialité d'une entité référencée. C'est un concept de l'ensemble S_c.

$$s_I \in S_c \quad \Rightarrow \quad \{Spatialité\ Inconnue\} = \{s_I\} \subset S_c$$

Définition 63 Le ***concept temporalité inconnue*** t_I est destiné à renseigner la méconnaissance ou l'absence d'information sur la temporalité d'une entité référencée. C'est un concept de l'ensemble T_c.

$$t_I \in T_c \quad \Rightarrow \quad \{Temporalité\ Inconnue\} = \{t_I\} \subset T_c$$

Nota La principale raison qui a incitée à affecter les concepts inconnus parmi les composites est justement la perception d'« inconnue » suscitée.

Les définitions 62 et 63 qu'impliquent aucune redéfinition des ensembles S_c ou T_c puisque les nouveaux concepts s_I et t_I sont éléments de ces ensembles.

VI.2 Pictogrammie : Définitions

Étant donné qu'une représentation pictogrammique des concepts inconnus a été définie par l'équipe de Perceptory, il a été nécessaire de définir les graphes des restrictions $f\big|_{\{s_I\}}$ et $f\big|_{\{t_I\}}$ de la fonction f aux ensembles S et T, définitions qui sont effectuées en extension au vu du faible nombre de concepts auxquels il faut associer un pictogramme :

Définition 64 Le graphe $G^{f|_{\{s_I\}}}$ de la restriction $f\big|_{\{s_I\}}$ de la fonction f est :

[76] Dans le langage pictogrammique de Perceptory, Yvan Bédard distingue la « géométrie inconnue » pour une occurrence de « la géométrie pas encore déterminée » qui s'applique à une classe entière. Ces nuances sémantiques sont difficiles à exprimer dans Perceptory puisque ce dernier ne propose que les diagrammes de classes. Seule la sémantique contextuelle du modèle et des notes permettent de distinguer ces nuances. Disposant d'un atelier de génie logiciel doté des diagrammes de classe et d'objets (occurrences), nous avons fait le choix d'exploiter ce potentiel pour dissocier ces deux nuances. En effet, il est possible de faire une extension du métamodèle d'UML pour les instances et une autre pour les classes. Le langage pictogrammique peut être alors utilisé pour annoter une instance sans annoter la classe dont elle est issue (géométrie inconnue) comme il est possible d'annoter une classe sans annoter les instances de cette classe (la géométrie pas encore déterminée).

$$G^{f|_{\{s_I\}}} = \{(s_I, ?)\}$$

Définition 65 Le graphe $G^{f|_{\{s_I\}}}$ de la restriction $f|_{\{t_I\}}$ de la fonction f est :

$$G^{f|_{\{t_I\}}} = \{(t_I, ?)\}$$

Comme pour les singletons précédents, les graphes $G^{f|_{\{s_I\}}}$ et $G^{f|_{\{t_I\}}}$ associés aux concepts inconnus sont éléments de $G^{f|_{sc}}$. Les définitions 64 et 65 remplacent dans le reste du document les définitions 49 et 50 du paragraphe IV.3.2.2.

Les pictogrammes $\overset{sl}{s_I}$ et $\overset{sl}{t_I}$, images des concepts s_I et t_I , appartiennent aux ensembles des pictogrammes de spatialité $\overset{vl}{S_c}$ et de temporalité $\overset{vl}{T_c}$:

$$\overset{vl}{s_I} \in \overset{vl}{S_c} \quad \Rightarrow \quad \{?\} \subset \overset{vl}{S_c} \qquad \text{et :} \qquad \overset{vl}{t_I} \in \overset{vl}{T_c} \quad \Rightarrow \quad \{?\} \subset \overset{vl}{T_c}$$

VI.3 Exemples d'entité référencée à spatialité inconnue ou temporalité inconnue

Dernièrement, une équipe de chercheurs a découvert un *Coin de Nature Vierge* dans la jungle des montagnes de Foja en Nouvelle-Guinée[77]. Aux dires de ces chercheurs, les limites de cette zone sont totalement inconnues. Pour modéliser l'entité référencée *Coin de Nature Vierge*, les chercheurs sont donc face à deux options :

- ⇨ Soit ils font le modèle sans se soucier de la spatialité de cette entité abandonnant ainsi toute information spatiale, l'étendue de la zone par exemple. C'est une option plausible mais étonnante de la part de chercheurs.
- ⇨ Soit ils manifestent dès le départ leur intérêt pour la spatialité en annotant l'entité *Coin de Nature Vierge*$_{<<?}$ avec le pictogramme de spatialité inconnue (cf. figure 102) même si, dans l'immédiat, ils ne disposent d'aucun élément pour déterminer cette spatialité.

Figure 102 Exemple d'entité référencée à spatialité inconnue.

D'ailleurs, il est fort probable que la priorité des prochaines missions soit justement de mieux définir cette zone donc sa spatialité. Il faudra alors vraisemblablement remplacer le pictogramme de spatialité inconnue par celui de spatialité polygonale. Le pictogramme de spatialité inconnue aura servi à annoter temporairement la méconnaissance de la spatialité.

Les recherches sur la naissance de l'Univers ont amené les chercheurs à estimer son origine à 15 milliards d'années[78], estimation qui a été possible grâce à la *Théorie de la Relativité Générale*. Cette dernière permet d'expliquer l'expansion de l'Univers à partir de 10^{-43}s après le *Big Bang* mais, aux dires des chercheurs, elle est inadaptée pour remonter le temps jusqu'à l'instant zéro. Aussi, la ***« date précise »*** du *Big Bang* n'est pas connue et *nécessitera de nombreuses recherches et probablement l'élaboration d'une théorie plus adaptée que Théorie de la Relativité Générale*[78].

[77] Sciences&Vie Junior – Avril 2006 – Numéro 199
[78] Sciences&Vie Junior – Avril 2001 – Hors série numéro 44

Au cours du processus de modélisation de l'Univers, marquer le concept *Big Bang*$_{<<?}$ avec le pictogramme de temporalité inconnue permet de tenir compte de cette réalité tout en rappelant que la propriété importante est justement la temporalité.

Figure 103 Exemple d'entité référencée à temporalité inconnue.

VI.4 En résumé

Les concepts de *Spatialité inconnue* et de *Temporalité inconnue* sont des concepts qui complètent la série de concepts primitifs, composites et dérivés. En termes de capacité d'expression, leur impact est moindre que les précédents mais l'intérêt principal de les utiliser est d'attribuer à des concepts thématiques le statut d'entité référencée ce qui impose de s'intéresser à la spatialité et/ou à la temporalité de ces entités même si elle est inconnue au moment de la modélisation.

Dans la plupart des cas, la spatialité inconnue et la temporalité inconnue sont remplacées par une spatialité et/ou une temporalité classique dès qu'elles sont connues suite à des recherches documentaires ou des travaux plus fondamentaux.

Au paragraphe VIII.2, nous verrons en particulier l'intérêt que présentent les concepts inconnus et l'impact qu'ils peuvent avoir sur le processus de modélisation. Leur présence dans un modèle constitue une information pertinente permettant de piloter le processus de modélisation des *Systèmes d'Information Géographique*. En cela, ils modifient les pratiques de modélisation.

Afin de ne pas surcharger le diagramme, la figure 100 représentant les deux taxinomies SIG et des entités référencées n'a pas été réactualisée avec les concepts de spatialité inconnue et de temporalité inconnue.

VII DERIVATION D'UN METAMODELE A PARTIR DES 3 TAXINOMIES

Les deux taxinomies de la figure 100 représentent le modèle d'organisation des concepts SIG et des entités référencées suivant une triple grille de lecture : nature, complexité et mode de combinaison des entités référencées. Les concepts de spatialité dérivée et inconnue ainsi que ceux de temporalité dérivée et inconnue sont absents parce qu'ils n'ont pas été représentés sur le diagramme afin de ne pas le surcharger. La taxinomie de pictogrammes n'a pas été représentée car elle est identique à celle des concepts SIG.

Le langage UML définit le concept de *Classe* comme un concept permettant de décrire un ensemble d'objets partageant les mêmes spécifications des propriétés, des contraintes et de la sémantique (OMG, 2005). Cette définition est dans le même esprit que la définition en compréhension des ensembles utilisés dans les premières étapes de construction des trois taxinomies.

En formalisme ensembliste, les ensembles sont constitués d'éléments alors qu'en langage UML, une classe représente une famille d'objets donc un ensemble d'objets de même type.

Suite à ce constat, nous avons substitué les ellipses du formalisme hybride de la figure 100 par le concept de *Classe* du langage UML. Ce choix est conforté par la définition du concept de *Généralisation* où il est précisé que chaque instance d'un *Classifieur* est aussi indirectement une instance d'un *Classifieur* plus général (OMG, 2005). Le diagramme de la figure 100 peut donc être réécrit en langage UML (cf. figures 105 et 106).

Dans la perspective d'établir un métamodèle permettant de représenter des entités référencées conformément aux définitions précédemment énoncées, il faut ajouter :

- ⇨ Les huit lois de composition propres à la taxinomie SIG (cf. figure 105) : $\overset{S}{\otimes}$, $\overset{T}{\otimes}$, $\overset{S}{\oplus}$, $\overset{T}{\oplus}$, $\overset{A_S}{\circ}$, $\overset{A_T}{\circ}$, $\overset{ST}{\otimes}$ et $\overset{S_T}{<<}$.
- ⇨ Les deux lois de composition la taxinomie des entités référencées (cf. figure 106) : $\overset{A}{\circ}$ et $\overset{G}{\circ}$.

- ⇨ La loi de cardinalité générale[79] $\overset{G}{*}$ qui, comme elle s'applique à tout concept, peut être factorisé (cf. figure 105) au niveau de la classe $\boldsymbol{U}$.
- ⇨ L'application d'affectation $<<$.

Dans le diagramme de la figure 100, il serait tentant d'établir une association unique au plus haut niveau de généralisation c'est-à-dire entre les classes $\boldsymbol{E}$ et $\boldsymbol{U}$ pour « matérialiser » l'application d'affectation $<<$ ce qui serait conforme à sa définition (cf. définition 17).

L'ennui c'est que ce choix introduit, en langage UML, des incohérences par rapport aux définitions précédentes. En effet, dans une arborescence de généralisation, les classes filles héritent de toutes les propriétés des classes mères.

Un programmeur utilisant un modèle avec une relation unique au plus haut niveau de généralisation pourrait alors associer une instance *Entité Référencée Point* à une instance *Polygone*, *Instant*, etc. ce qui est ne correspond pas aux définitions précédentes de ces entités (cf. figure 104).

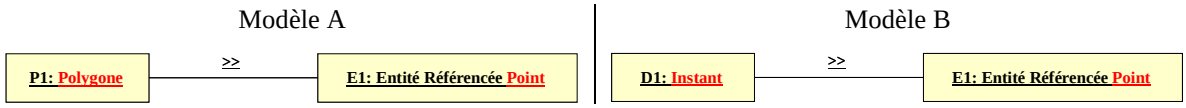


Figure 104 Exemples d'incohérences introduites par une relation unique au plus haut niveau de généralisation.

Pour éviter ces incohérences, il faut établir les relations au bas niveau de spécialisation c'est-à-dire entre la classe représentant un concept SIG et celle représentant l'image réciproque du concept SIG dans l'ensemble des entités référencées (cf. figure 107).

Le diagramme de la figure 107 décrit la syntaxe entre concepts SIG, entre entités référencées mais aussi entre concepts SIG et entités référencées. Le diagramme de la figure 107 est donc le métamodèle d'une application permettant de modéliser des Systèmes d'Information Géographique conformes aux définitions énoncées précédemment.

Si on s'intéresse au modèle constitué des seuls concepts SIG (cf. figure 105), lui aussi décrit la syntaxe entre concepts SIG. C'est donc le métamodèle d'un langage que nous appellerons ***Langage SIG***. Par analogie, on parlera de ***Langage Pictogrammique*** pour qualifier le métamodèle issu de la Taxinomie des Pictogrammes.

Dans un contexte de conception d'un atelier de génie logiciel dédié à la modélisation de *Systèmes d'Information Géographique*, certains choix permettent de simplifier ce métamodèle.

Le premier consiste à choisir un seul concept comme support des entités référencées, le concept UML de *Classe* par exemple. C'est le choix effectué dans l'atelier de génie logiciel Perceptory. Alors, la classe $\boldsymbol{E}$ est remplacée par le concept *Classe*. Les entités référencées étant représentées par un concept unique, les incohérences syntaxiques évoquées ci-dessus ne sont plus possibles. L'application d'affectation peut être représentée par une relation au plus haut niveau de généralisation (cf. figure 108). Dans ce contexte, les lois de composition $\overset{A}{\circ}$ et $\overset{G}{\circ}$ sont assurées par le métamodèle du langage UML.

Le second choix repose sur le constat que les classes représentant un singleton composite n'ont plus d'utilité excepté celles de spatialité compliquée et de spatialité multiforme ainsi que leurs équivalents de temporalité. En effet, les concepts composites peuvent être obtenus en combinant les concepts primitifs.

Suite aux simplifications apportées, le diagramme de la figure 108 constitue donc le métamodèle[80] d'un atelier de génie logiciel mettant en œuvre le langage UML pour modéliser des entités référencées. C'est ce métamodèle qui nous appellerons Métamodèle SIG.

---

[79] Avec la loi cardinalité normale, il n'est pas possible de décorer un concept de la cardinalité alternative d'UML d'où la préférence donné ici à la loi de cardinalité générale.

[80] Afin de réifier les lois de composition d'addition, de substitution, les associations ont été établies avec les classes *concept de spatialité primitive* et/ou *concept de temporalité primitive* conformément aux définitions de lois de composition. Dans une perspective de développement, des patrons composition impliquant aussi les classes *concept de spatialité* et/ou *concept de temporalité* seraient plus adaptés et plus performants.

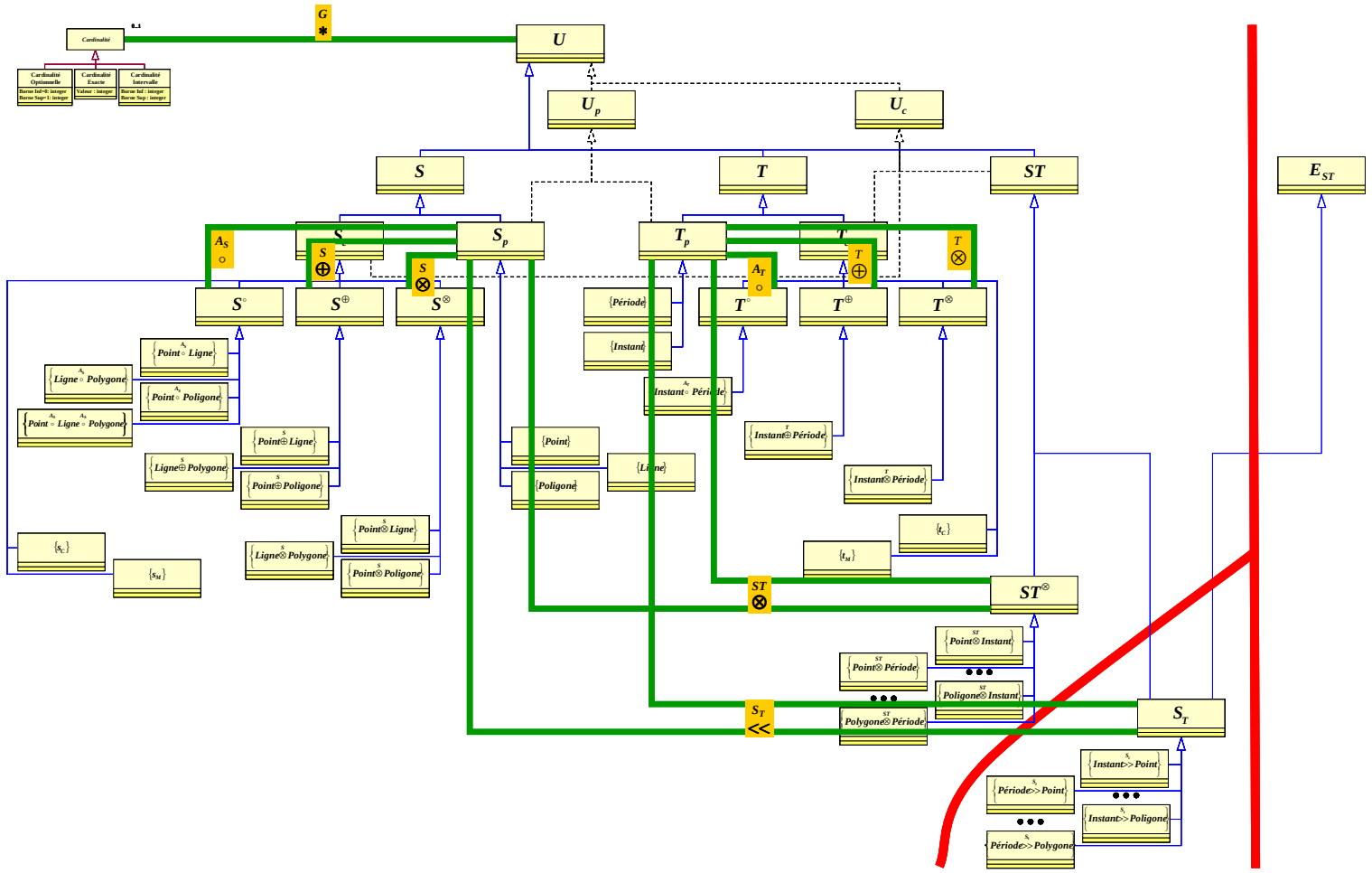


Figure 105 Représentation de la taxinomie SIG en langage UML.

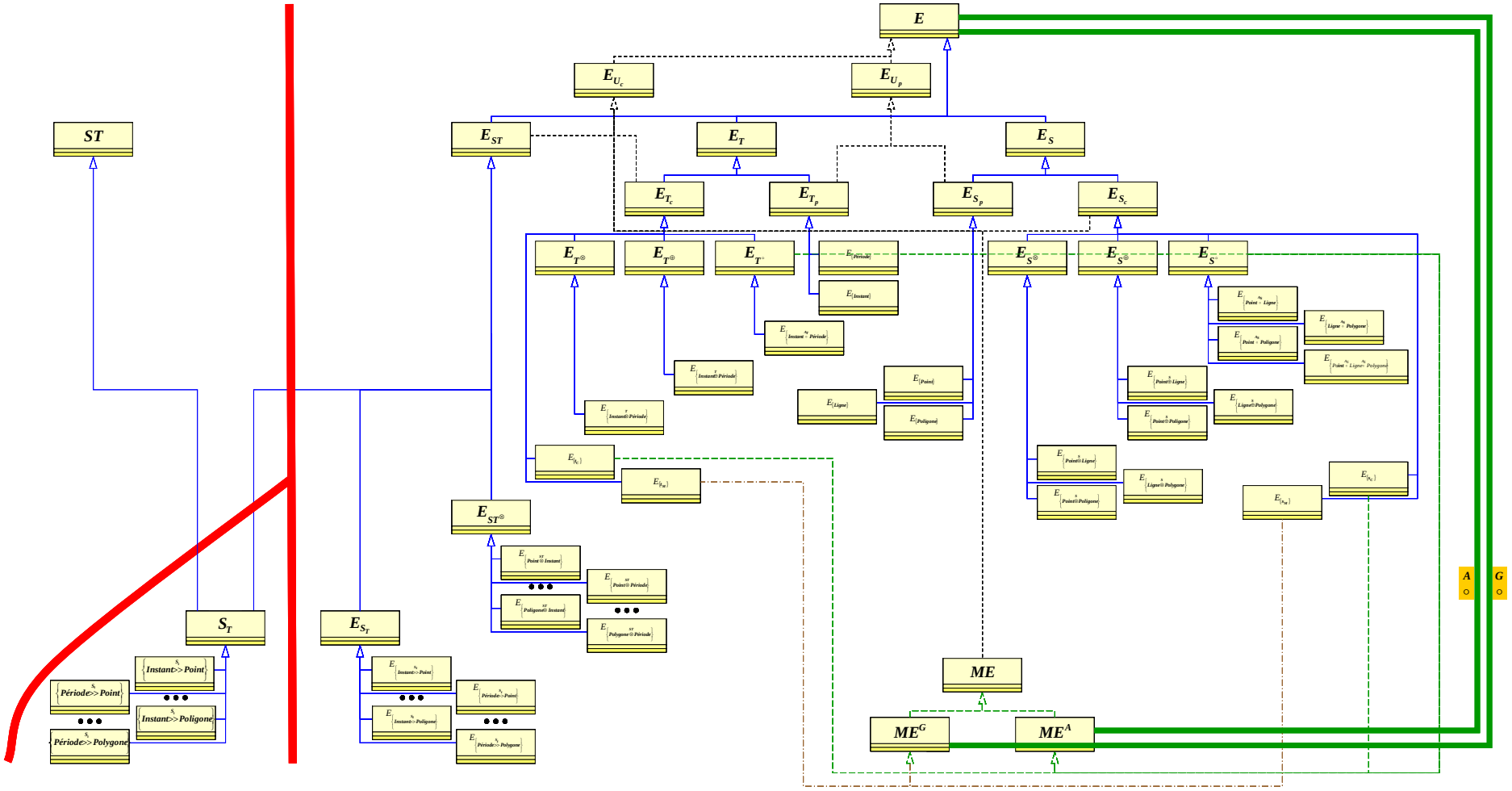


Figure 106 Représentation de la taxinomie des entités référencées en langage UML.

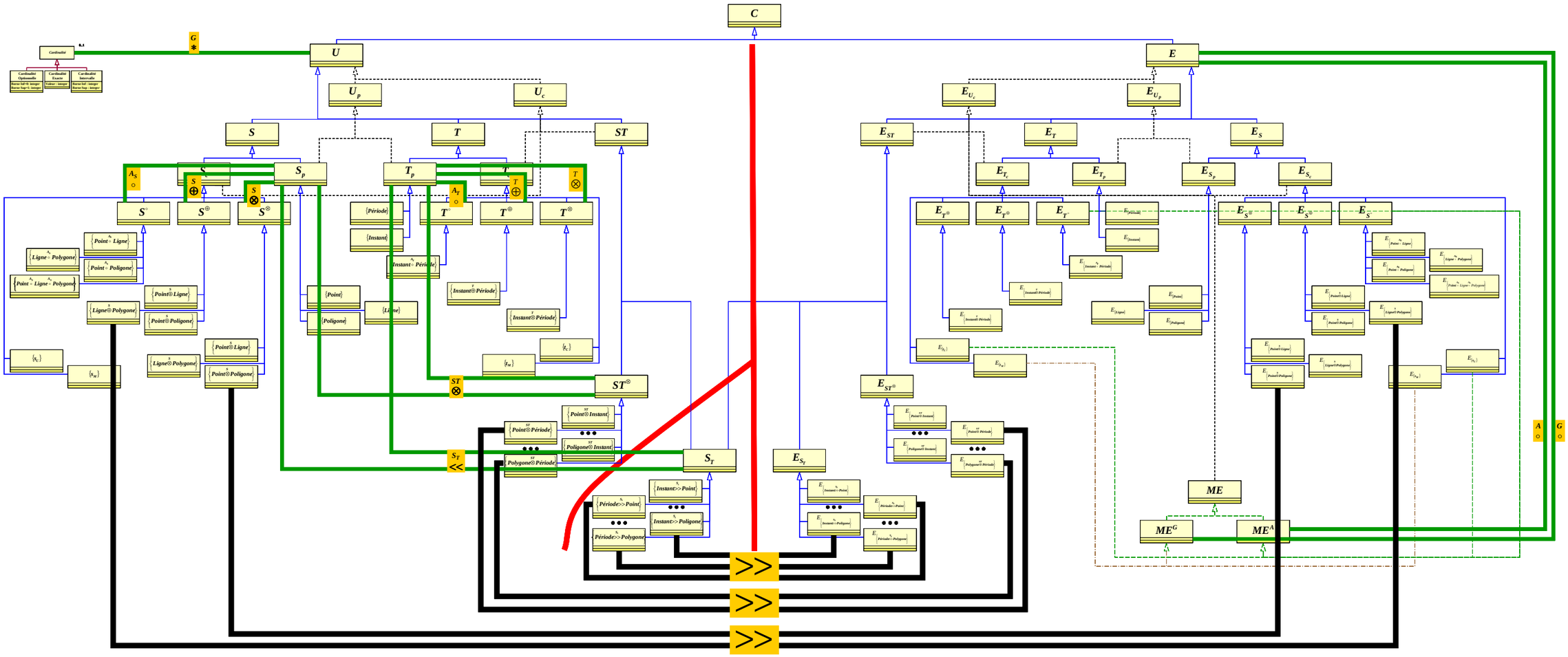


Figure 107 Représentation des taxinomies SIG et des entités référencées ainsi que l'application d'affectation **<<** en langage UML.

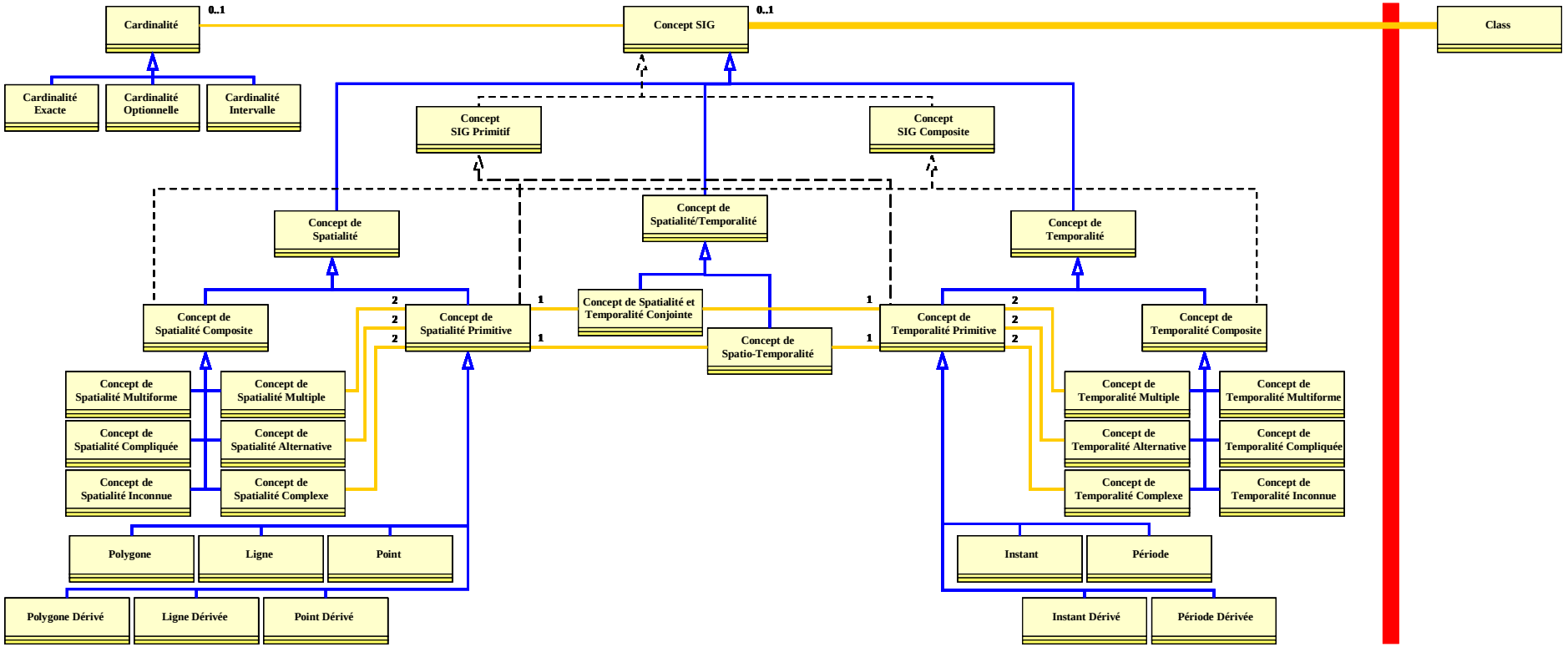


Figure 108 *Métamodèle SIG.*

VIII AMELIORATIONS DU PROCESSUS DE MODELISATION GRACE AUX SPATIALITES ET TEMPORALITES DERIVEES, INCONNUES, COMPLIQUEES ET MULTIFORMES

Lorsque le concepteur d'un *Système d'Information Géographique* ou d'une application informatique utilise un atelier de génie logiciel, la spatialité et/ou la temporalité sont souvent introduites sous forme d'annotations sur le concept thématique. Ces dernières sont soit de nature textuelle, soit de nature pictogrammique, soit encore les deux à la fois. Le choix entre l'une de ces trois formes est souvent conditionné par l'outil informatique utilisé.

Quelle que soit la forme utilisée, les annotations constituent une information essentielle du concept thématique. En effet, elles confèrent le statut d'entité référencée au concept thématique qu'elle annote, statut qu'elle n'aurait pas si elle ne portait aucune annotation (cf. définition 3). Elle oblige par conséquence le concepteur à attacher une grande importance à la spatialité et à la temporalité du concept thématique.

Les annotations de spatialités et temporalités dérivées, inconnues, compliquées et multiformes peuvent être utilisées dans un contexte de pilotage du processus de modélisation comme nous allons le voir dans les paragraphes suivants.

VIII.1 Spatialité dérivée et temporalité dérivée

Les concepts dérivés supposent qu'il existe une opération qui va permettra de déterminer la spatialité et/ou la temporalité des entités référencées qu'ils annotent. Dans le modèle d'analyse, la présence de cette opération n'est pas essentielle par contre elle devient ***obligatoire*** dans le modèle d'implémentation afin que les générateurs de code puissent engendrer sa signature dans le code.

Dans un atelier de génie logiciel possédant un langage de métaprogrammation, il est possible d'implémenter une transformation qui vérifie cette cohérence, c'est-à-dire l'existence d'une opération dont le type de retour correspond au concept dérivé.

En phase d'implémentation, l'entité référencée *Lac*<<▰ de la figure 101-Modèle B doit avoir une opération de concaténation de lignes (*concaténationDeLignes()*<<▣) dont le type de retour doit être un polygone (cf. figure 109).

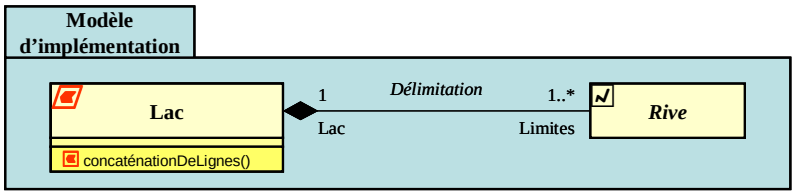


Figure 109 Illustration de l'opération de spatialité associée au SIG dérivé.

Dans ce type d'atelier de génie logiciel, il est même possible de conditionner la génération de code à l'existence de cette opération. Si la condition n'est pas remplie, l'atelier de génie logiciel délivre alors un message d'erreur avec obligation pour le concepteur ou le programmeur d'apporter une solution.

VIII.2 Spatialité inconnue et temporalité inconnue

Lors de la modélisation d'un système, il n'est pas rare que la spatialité et la temporalité de certains concepts thématiques ne soient pas connues de façon précise lors de la modélisation et qu'il soit nécessaire d'effectuer des recherches pour trouver le renseignement.

Confronté à cette situation, plutôt que de ne pas annoter le concept thématique, il vaut mieux le faire avec l'une des annotations de la spatialité inconnue ou de temporalité inconnue ainsi le concept thématique acquière le statut d'entité référencée.

Mais les concepts inconnus peuvent être mis en œuvre dans un autre contexte. Le démarrage d'un projet est un période assez particulière où ont lieu d'intenses réflexions, de nombreux échanges et où les concepts affluent de façon continue. Il est souvent difficile de tenir le rythme auquel des concepts affluent surtout si une analyse détaillée est effectuée.

Une autre approche de travail consiste à introduire les concepts dans le modèle au fur et à mesure de leur arrivée et de reporter l'analyse détaillée et en particulier la description de la spatialité et/ou de la temporalité. Entre-temps, les concepts continuent à arriver et les questionnements autour des premiers concepts s'estompent. Quand il faut reprendre l'analyse détaillée, l'intégralité du modèle doit être scrutée. C'est un travail qui devient vite fastidieux lorsque le modèle a plusieurs dizaines de concepts voire plus.

Les concepts inconnus peuvent être utilisés judicieusement dans ce contexte. En effet, dès qu'un nouveau concept est introduit dans le modèle, si sa spatialité et/ou sa temporalité a une importance thématique, il est possible de le marquer avec l'un des deux concepts inconnus, lui attribuant le statut d'entités référencées.

Les concepts inconnus sont une information qui facilite ensuite la recherche des entités référencées. Dans les ateliers de génie logiciel munis d'un niveau de métaprogrammation, il est possible d'implémenter une fonctionnalité d'introspection systématique du modèle qui va lister les concepts inconnus. Il est alors plus facile d'identifier les entités référencées dont l'analyse doit être poursuivie. De plus, cela va permettre de prioriser l'ordre d'analyse car souvent les entités référencées ne sont pas totalement indépendantes.

Dans les deux contextes décrits ci-dessus, les concepts inconnus sont utilisés comme ***« indicateur de contrôle »*** de l'avancement de la recherche d'information, premier contexte, et de l'analyse, second contexte. La modélisation avançant et les concepts se précisant, il est souvent possible de lever les indéterminations et les concepts inconnus sont alors remplacés par des concepts SIG. Les entités référencées à spatialité inconnue ou temporalité inconnue peuvent donc changer de spatialité ou de temporalité.

Bien évidemment, il existe des entités référencées pour lesquelles la spatialité et/ou la temporalité sont vraiment inconnues, celles-ci resteront annotées avec des concepts inconnus[81] sachant que rien n'est jamais figé et qu'un jour peut être ces propriétés pourront être renseignées comme ce sera vraisemblablement le cas pour l'entité référencée *Coin de Nature Vierge*<<? du paragraphe VI.3.

Dans ces deux contextes, les concepts inconnus sont utilisés pour assister le processus de modélisation et contrôler la qualité du modèle. En effet tant qu'il reste au sein du modèle des concepts inconnus, l'analyse n'est pas terminée, sauf si les propriétés spatiales et temporelles sont vraiment inconnues.

Aussi, il est crucial pour la qualité du modèle que la fonctionnalité d'introspection soit mise en œuvre en fin de la séance d'analyse pour identifier les entités qui auraient été oubliées et, si c'est le cas, d'effectuer leur analyse tant que les acteurs du domaine sont présents.

VIII.3 Spatialité compliquée et temporalité compliquée - Spatialité multiforme et temporalité multiforme

Dans les prémices de l'analyse, les macro-entités référencées suffisent souvent amplement pour résumer les modèles sous-jacents à chacune d'elles. Elles permettent d'initier le dialogue et des échanges constructifs entre les acteurs du domaine thématique et le concepteur du *Système d'Information Géographique*. Elles sont fondamentales à ce stade. Il est primordial de les annoter avec les concepts de spatialité compliquée ou multiforme ou avec leurs homologues temporels afin de leur conférer le statut d'entité référencée et de bien les différencier des concepts exempts de spatialité et/ou de temporalité.

L'analyse progressant, les acteurs du domaine et le concepteur vont se trouver confrontés à faire un choix entre deux options sur le devenir des macro-entités référencées :

- ⇨ Soit ils les conservent en l'état par manque d'information pour décrire les entités de granularité plus fine ou parce que le niveau de granularité correspond aux besoins des acteurs.
- ⇨ Soit ils détaillent le modèle des macro-entités référencées et les désagrègent.

Si au cours du processus de développement, le concepteur vient à mettre en œuvre la seconde option, les macro-entités référencées initiales sont des entités qui évoluent au cours du cycle de modélisation. Les macro-entités initiales peuvent donc être vues comme des entités référencées de ***première analyse qu'il faut ensuite affiner***. Lors du raffinement, elles subissent un changement d'état qui se traduit par un modèle détaillé et souvent par la perte de l'annotation lui conférant le statut de macro-entité (cf. figures 95 et 98). Ce changement d'état constitue une information permettant de piloter le processus de modélisation. En effet, la présence des annotations particulières montrent que la désagrégation de l'entité n'a pas eu lieu.

Dans les ateliers de génie logiciel dotés d'un langage de métaprogrammation, il est assez facile d'implémenter une fonctionnalité d'introspection du modèle listant les macro-entités référencées ayant toujours

[81] Une valeur marquée supplémentaire permettrait de distinguer les deux types d'utilisation des concepts de spatialité inconnue et de temporalité inconnue. Il serait alors possible d'exclure les entités référencées dont la spatialité ou la temporalité est réellement inconnue de la recherche.

leurs annotations. L'édition à la demande de la liste des macro-entités référencées non raffinées permet au concepteur de prioriser l'ordre d'analyse des macro-entités. En effet, s'il y a plusieurs macro-entités au sein d'un modèle elles sont souvent dépendantes.

Effectuer une introspection du modèle avant le départ des acteurs du domaine permet comme précédemment de vérifier si toutes les macro-entités ont été analysées et affinées. Si ce n'est pas le cas, il est encore possible de les traiter tant que les acteurs du domaine sont présents. Sans ce dernier contrôle, la désagrégation des macro-entités référencées aurait été effectuée par le concepteur soit par lui-même, c'est le pire des cas, soit après avoir recuilli les informations auprès les acteurs par courrier, par téléphone, etc. Dans ces conditions, la capture de l'information est moins favorables qu'en séance d'analyse.

Lorsque le modèle est de petite taille, la fonctionnalité d'introspection n'apporte pas de gain considérable car il est relativement facile de ne pas oublier les macro-entités référencées à raffiner. Par contre, lorsque le modèle implique une centaine de concepts voire plus, elle facilite le processus de modélisation et surtout elle en contrôle la qualité.

IX CONCLUSION

L'exercice de formalisation de la terminologie mené dans cet annexe à l'aide de la Théorie des ensembles a permis d'élaborer le diagramme de la figure 100 qui correspond à une lecture ensembliste de la spatialité et de la temporalité des entités référencées. Le résultat de cet exercice est une triple taxinomie (SIG, des pictogrammes et des entités référencées). Chacune de ces trois taxinomies peut être abordées suivant une triple grille de lecture :

- La première décline la nature des propriétés de l'entité référencée : ***spatialité***, ***temporalité*** ou ***spatialité/temporalité***. Elle structure les concepts SIG et leurs homologues des deux autres taxinomies suivant une partition complète (exemple : $\mathscr{P}'(\boldsymbol{U})$ pour l'ensemble $\boldsymbol{U}$).
- La seconde exprime la complexité des caractéristiques de spatialité et de temporalité d'une entité référencée : ***primitive*** ou ***composite***. Comme la précédente, elle réalise une partition complète ($\mathscr{P}_2(\boldsymbol{U})$) de l'ensemble des concepts SIG. L'ensemble des pictogrammes et celui des entités référencées ont une partition similaire.
- La troisième résulte d'une vision UML du mode d'organisation des entités référencées, ***d'agrégation*** ou ***de généralisation***, mode d'organisation lié aux deux principales relations du langage UML. Étant donné qu'au sein du langage UML il existe d'autres types de relations, cette troisième grille de lecture est une partition incomplète de l'ensemble des concepts SIG $\boldsymbol{U}$ et des ensembles homologues.

Au sein de chacune des taxinomies, les trois grilles de lecture évoquées ci-dessus permettent de classer les concepts de façon unique.

Les huit lois de composition entre concepts SIG permettent de faire des combinaisons de concepts SIG primitifs. Elles correspondent au langage pictogrammique de l'atelier de génie logiciel Perceptory. Le nombre de lois de composition n'est pas limité, il est donc toujours possible de définir de nouvelles lois de composition pour répondre à un besoin spécifique.

Les trois taxinomies ont permis de dériver le *Métamodèle SIG* (cf. figure 108) conforme aux lois de composition du langage pictogrammique.

Cet exercice théorique a mis en exergue l'intérêt des concepts de spatialité dérivées, inconnues, compliquées et multiformes et leurs homologues temporels pour assister le concepteur dans le processus de modélisation.

L'exercice de formalisation de la terminologie mené dans ce chapitre a permis d'atteindre les deux objectifs fixés initialement : l'appropriation des concepts du langage pictogrammique de l'atelier de génie logiciel Perceptory et création du *Métamodèle SIG* qui est utilisé au Chapitre 1 comme instrument de comparaison entre méthodes ou entre formalismes.

Chapitre 8. Concepts de spatialité floue et de temporalité floue - Extension du Métamodèle SIG

Avertissement : Les concepts de spatialité floue et de temporalité floue qui sont définis dans le présent chapitre sont ensuite intégrés dans le *Métamodèle SIG* dérivé de l'étude terminologique du Chapitre 7, il est donc recommandé de s'imprégner des concepts majeurs décrits au Chapitre 7.

Les concepts de spatialité et de temporalité du langage SIG (cf. Chapitre 7) ne permettent pas de décrire l'ensemble des situations rencontrées dans le domaine d'intervention privilégié du Cemagref : l'agriculture et l'environnement.

Aussi, l'*outil d'aide à la conception de Systèmes d'Information Géographique* qui a été réalisé dans le cadre de cette recherche sera utilisé pour la conception de *Systèmes d'Information Géographique* de ce domaine. Or, de nombreuses entités référencées de ce domaine ont une nature floue ou suscite une impression de floue.

L'étude bibliographique sur les méthodes et les formalismes de conception de *Systèmes d'Information Géographique* (cf. Chapitre 1) a permis de constater que les auteurs de deux formalismes, CONGOO (Pantazis *et al.*, 1997) et MADS (Shu *et al.*, 2003), avaient mené une réflexion sur la description de cette typologie de concepts.

La problématique posée par les entités référencées incertaines est parfaitement décrite dans la communication de (Pantazis *et al.*, 1997) présentant le formalisme CONGOO. Malheureusement, les auteurs ne donnent aucune définition précise de ces concepts.

Cette lacune qui a été comblée avec la communication de (Shu *et al.*, 2003) qui propose une extension du formalisme MADS prenant en compte les concepts de spatialité floue et de temporalité floue.

Bien que la réflexion théorique ait été menée, il n'existe pas à ce jour d'outil d'informatique permettant de décrire la spatialité et/ou la temporalité des entités référencées dont les limites sont incertaines puisque le formalisme CONGOO n'a jamais été implémenté (cf. Chapitre 1-IV.1) et que le prototype implémentant le formalisme MADS est au stade recherche (cf. Chapitre 1-IV.3).

L'absence d'outil informatique implémentant un formalisme de description des entités référencées floues ont conduit à étendre le Métamodèle SIG (cf. figure 106) afin d'enrichir le langage pictogrammique de l'atelier de génie logiciel Perceptory avec des concepts de spatialité floue et de temporalité floue.

Le paragraphe I présente la problématique au travers de la description de deux exemples. Le paragraphe II pose la définition générale d'un concept floue, la convention pictogrammique adoptée et redéfinie un certain nombre de définitions du Chapitre 7 influencées par l'introduction de ces nouveaux concepts. Le paragraphe III présente le *Métamodèle SIG* étendu aux nouveaux concepts de spatialité floue et de temporalité floue. Le paragraphe IV identifie et décrit le patron de conception qui facilitera la génération automatique de ces concepts. Le paragraphe V traite les deux exemples du paragraphe I présentant la problématique à l'aide des concepts flous précédemment définis. Enfin, le paragraphe VI conclue le propos autour des concepts de spatialité floue et de temporalité floue.

Sommaire détaillé

I LA PROBLEMATIQUE

Les but des deux exemples présentés ci-après est d'illustrer une typologie de problèmes rencontrés par les concepteurs de *Systèmes d'Information Géographique* impossible à modéliser avec les concepts SIG décrits au Chapitre 7.

Le premier exemple est issu du domaine de la télédétection mais la problématique qu'il pose est fréquemment rencontrée dès que l'information pertinente est extraite de mesures de terrain ou autre par un traitement numérique. Les systèmes d'acquisition de données (capteurs, GPS, etc.) et les traitements associés (photo interprétation, extraction automatique d'objets d'une image, etc.) introduisent souvent une incertitude sur la position ou la frontière des entités référencées, incertitude qu'il est souhaitable de prendre en compte dès l'analyse. Cet exemple traite de l'extraction de l'entité référencée *Parcelle* d'une image satellitale.

Le second exemple concerne l'entité référencée *Zone Marécageuse* qui correspond à une typologie de paysage courante en description des territoires. Le nom même de cette entité référencée évoque une sensation d'incertitude de sa limite.

I.1 Exemple de l'entité référencée Parcelle

L'extraction d'objets pertinents des images satellitales est de plus en plus effectuée automatiquement via des traitements numériques qui ont pour but de définir et de délimiter la géométrie des objets pertinents (polygones, lignes, etc.) d'une thématique (occupation du sol, hydrologie, etc.).

Par exemple, une parcelle de blé (objet vectoriel[82] recherché) dans une image (donnée raster[83]) est souvent identifiée par le rayonnement émis par chacun des pixels appartenant à la parcelle (cf. figure 110). Une opération de segmentation va permettre de discriminer les pixels ayant un rayonnement proche de celui du blé des autres. Les premiers forment l'agrégat des pixels de blé. Une seconde opération va ensuite déterminer le polygone délimitant l'agrégat des pixels de blé.

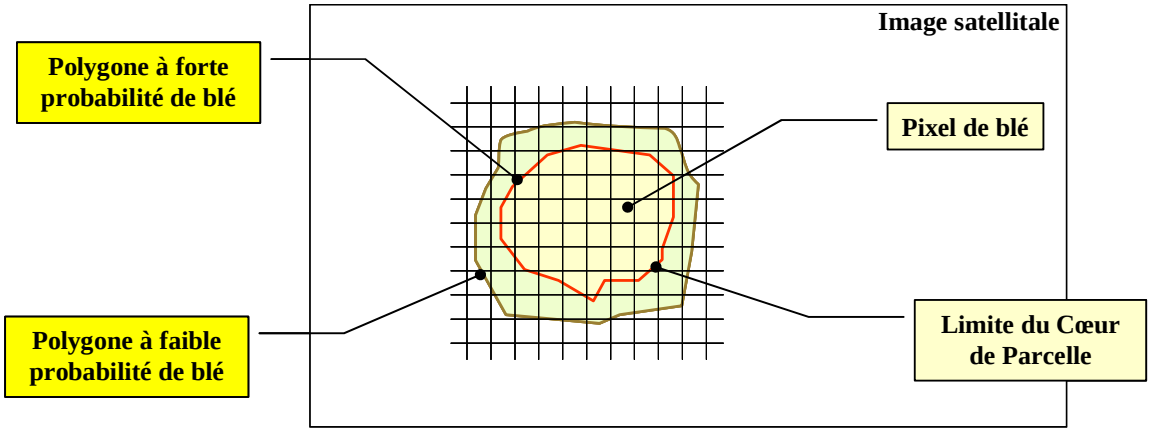


Figure 110 Extraction les limites d'une parcelle de blé dans une image satellitale.

De ces traitements numériques, celui relatif à la segmentation est cruciale car l'affectation d'un pixel de blé à l'agrégat résulte souvent de la comparaison de son rayonnement à une valeur de référence I_0 déduite de mesures de terrain, de laboratoire, etc. La valeur de référence I_0 joue un rôle d'inclusion ou d'exclusion des pixels dans l'agrégat. Étant donné que de nombreux paramètres peuvent influencer la qualité de la valeur de référence, adopter une valeur unique n'est pas toujours la meilleure solution.

Une stratégie alternative serait, par exemple, de prendre en compte l'erreur absolue ΔI propre à la valeur de référence, définissant ainsi un intervalle de valeurs possibles $I \in [I_0 - \Delta I_0, I_0 + \Delta I_0]$. Appliqué avec la borne supérieure, le traitement de segmentation permettra de définir un cœur de parcelle dont la probabilité de présence

[82] Un objet vectoriel est un objet dont la propriété de spatialité est l'une des géométries Point, Ligne ou Polygone.
[83] Une donnée raster est une donnée sur un espace continu qui a été discrétisée suivant un maillage régulier de points ou de cellules.

de blé est certaine. Appliqué avec la borne inférieure, le traitement délimitera un second polygone incluant a priori le premier dont la probabilité de présence de blé est plus faible en périphérie.

Ces deux polygones et l'indice de probabilité associés constituent une information qui pourrait être utilisée dans le cadre d'attribution des aides de la **Politique Agricole Commune**.

I.2 Exemple de l'entité référencée Zone Marécageuse

Dans la perspective de réaliser un *Système d'Information Géographique* permettant d'assurer une meilleure gestion d'un territoire et une meilleure coordination des aménagements à réaliser, le concepteur du *Système d'Information Géographique* réunit les acteurs du territoire pour analyser et décrire les différentes composantes afin d'établir un modèle le plus pertinent possible par rapport aux préoccupations des acteurs.

L'une des composantes de ce territoire est un lac situé dans une vallée traversée par un cours d'eau dont les abords représentent une zone de conflit d'intérêts où se côtoie une zone agricole, une zone forestière très riche d'un point de vu faunistique et une zone marécageuse.

Les élus locaux soucieux du développement de cette vallée envisagent de favoriser le tourisme autour de la richesse faunistique de la zone forestière en imposant la contrainte de ne pas dégrader cette zone avec des équipements hôteliers entre autres. Cela signifie qu'il faut reporter la construction des équipements sur les autres zones. Bien entendu, les acteurs agricoles s'opposent à la construction des équipements en zone agricole et suggèrent de rendre constructible et viable la zone marécageuse. L'étude de cette hypothèse suppose d'avoir une bonne connaissance du territoire et en particulier de la zone marécageuse.

Les acteurs de ce territoire réunis en séance d'analyse réalisent la description suivante du lac : la forêt occupe une partie de la rive gauche du lac, la majeure partie de la rive droite est quant à elle bordée par la zone agricole et la zone marécageuse s'étend sur toute la partie amont du lac (cf. figure 111). Les acteurs sont aussi en mesure de préciser l'étendue du marécage en période de pluviosité exceptionnelle définissant ainsi la ligne des plus hautes eaux et en période d'étiage délimitant la ligne des eaux permanentes.

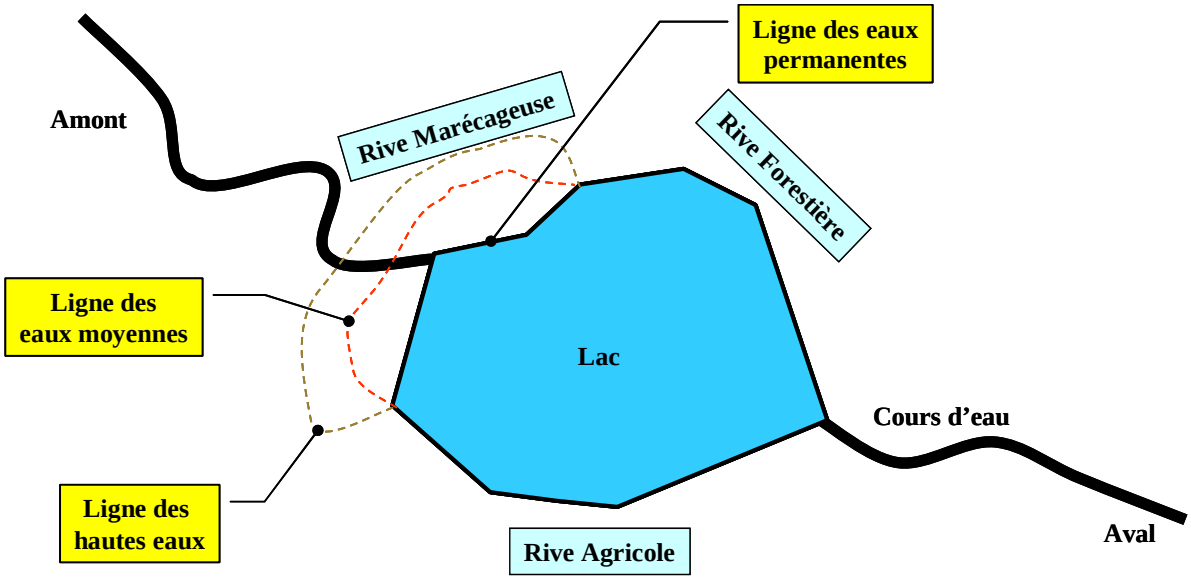


Figure 111 Les différentes limites de l'entité référencée *Rive Marécageuse.*

Étant liée à une pluviosité exceptionnelle, la ligne des plus hautes eaux a une probabilité de présence d'eau faible alors que la ligne des eaux permanentes a une probabilité maximale de 100%. Entre ces deux lignes, ils existent toutes les variantes possibles dont une ligne de probabilité moyenne (ligne des eaux moyennes).

Avec les rives forestière et agricole, les deux lignes extrêmes de la zone marécageuse constituent du point de vu géomatique deux polygones représentant les enveloppes minimale et maximale du lac, les rives forestière et agricole étant des éléments communs aux deux polygones.

I.3 En résumé

Les deux exemples ci-dessus montrent des entités référencées *Parcelle* et *Rive Marécageuse* dont au moins une propriété de spatialité est de nature floue soit à cause de problèmes technologiques et de traitements numériques (*Parcelle*), soit à cause de la nature même de l'entité qui suscite une perception d'incertitude (*Rive Marécageuse*).

Le but de ce chapitre est de proposer des concepts SIG qui permettent d'identifier et de renseigner, dès la phase d'analyse, les entités référencées dont la spatialité et la temporalité sont connues avec un certain niveau d'incertitude. Nous définirons au paragraphe II ci-dessous les concepts SIG et les conventions pictogrammiques qui permettent de décrire les concepts de spatialité floue et temporalité floue qui viennent compléter les langages SIG et pictogrammique du Chapitre 7.

II SPATIALITE PRIMITIVE FLOUE ET DE TEMPORALITE PRIMITIVE FLOUE

Un des résultats importants du Chapitre 7 est que les concepts SIG composites résultent de ***« combinaisons »*** de concepts SIG primitifs. Ainsi, en définissant des ***concepts primitifs flous*** et en les incluant dans le *Métamodèle SIG* de la figure 106, ils contribueront automatiquement aux lois de composition.

Le but de ce paragraphe est en premier lieu de définir les concepts SIG flous (cf. II.1), en deuxième lieu de préciser la convention pictogrammique permettant de les identifier visuellement (cf. II.2) et en troisième lieu de redéfinir les ensembles des concepts SIG primitifs et des pictogrammes SIG primitifs ainsi que les ensembles des entités référencées images réciproques des nouveaux singletons (cf. II.3).

II.1 Définition des concepts SIG primitifs flous

Les concepts flous ont été introduits aussi bien pour les propriétés de spatialité que de temporalité. La définition générale relative à ces concepts est :

> **Définition 66** Un ***concept SIG primitif flou*** est un concept SIG doté d'un indice de probabilité, à valeur dans l'intervalle **[0,1]**, portant sur le concept thématique d'une entité référencée.

Cette définition ne posant aucune restriction quant à la nature du concept SIG primitif, elle s'applique aussi bien aux concepts dérivés qu'aux concepts non dérivés. Aussi, les concepts SIG de spatialité primitive *Point*, *Ligne*, *Polygone* et *Volume*[84] et de temporalité primitive *Instant* et *Période* ainsi que leurs équivalents dérivés ont chacun d'eux un concept homologue flou.

II.2 Définition de la convention pictogrammique des concepts SIG primitifs flous

Afin de différencier visuellement cette série de nouveaux concepts des concepts SIG primitifs non flous, la convention pictogrammique suivante a été fixée :

> **Convention** Un ***pictogramme SIG primitif flou*** est un pictogramme SIG primitif dont le pictogramme du concept de spatialité ou de temporalité s'estompe suivant la diagonale issue du coin haut à droite au coin bas à gauche. L'atténuation des *pictogrammes de spatialité* augmente suivant cette direction et dans la direction opposée pour les *pictogrammes de temporalité*.

En appliquant cette convention pictogrammique, le pictogramme de spatialité polygonale  a pour homologue le pictogramme flou  , le pictogramme  est quant à lui l'homologue de  , etc. Tous les nouveaux pictogrammes flous sont listés au paragraphe suivant.

[84] Cf. Avertissements.

II.3 Redéfinition des ensembles

Tous les nouveaux ensembles relatifs aux concepts flous de ce chapitre viennent enrichir la taxinomie SIG, celle des pictogrammes et celle des entités référencées. Au terme du paragraphe, la figure 98 devrait être reprise en intégrant tous ces nouveaux ensembles mais, étant donné que le diagramme serait surchargé, elle n'a pas été redessinée.

II.3.1 *Primitifs*

L'adoption des nouveaux concepts nécessite de redéfinir les deux ensembles de concepts primitifs S_p et T_p. Ces définitions sont énoncées en extension :

Définition 67 L'ensemble des ***concepts de spatialité primitive*** S_p est défini par :

$$S_p = \begin{Bmatrix} \textit{Point}, & \textit{Point Dérivé}, & \textit{Point Flou}, & \textit{Point Flou Dérivé}, \\ \textit{Ligne}, & \textit{Ligne Dérivée}, & \textit{Ligne Floue}, & \textit{Ligne Floue Dérivée}, \\ \textit{Polygone}, & \textit{Polygone Dérivé}, & \textit{Polygone Flou}, & \textit{Polygone Flou Dérivé}, \\ \textit{Volume}, & \textit{Volume Dérivé}, & \textit{Volume Flou}, & \textit{Volume Flou Dérivé} \end{Bmatrix}$$

Définition 68 L'ensemble des ***concepts de temporalité primitive*** T_p est défini par :

$$T_p = \begin{Bmatrix} \textit{Instant}, & \textit{Instant Dérivé}, & \textit{Instant Flou}, & \textit{Instant Flou Dérivé}, \\ \textit{Période}, & \textit{Période Dérivée}, & \textit{Période Floue}, & \textit{Période Floue Dérivée} \end{Bmatrix}$$

Le nombre de concepts étant limité, les graphes $G^{f|_{S_p}}$ et $G^{f|_{T_p}}$ des restrictions de la fonction f aux ensembles de spatialité primitive S_p et de temporalité primitive T_p sont redéfinis en extension :

Définition 69 Le graphe $G^{f|_{S_p}}$ de la fonction $f|_{S_p}$ est :

$$G^{f|_{S_p}} = \begin{Bmatrix} (\textit{Point},), & (\textit{Point Dérivé},), & (\textit{Point Flou},), & (\textit{Point Flou Dérivé},), \\ (\textit{Ligne},), & (\textit{Ligne Dérivée},), & (\textit{Ligne Floue},), & (\textit{Ligne Floue Dérivée},), \\ (\textit{Polygone},), & (\textit{Polygone Dérivé},), & (\textit{Polygone Flou},), & (\textit{Polygone Flou Dérivé},), \\ (\textit{Volume},), & (\textit{Volume Dérivé},), & (\textit{Volume Flou},), & (\textit{Volume Flou Dérivé},) \end{Bmatrix}$$

Définition 70 Le graphe $G^{f|_{T_p}}$ de la fonction $f|_{T_p}$ est :

$$G^{f|_{T_p}} = \begin{Bmatrix} (\text{Instant},), & (\text{Instant Dérivé},), & (\text{Instant Flou},), & (\text{Instant Flou Dérivé},), \\ (\text{Période},), & (\text{Période Dérivée},), & (\text{Période Floue},), & (\text{Période Floue Dérivée},) \end{Bmatrix}$$

Les graphes des fonctions $f|_{S_p}$ et $f|_{T_p}$ étant définis, il est possible de redéfinir les images des ensembles des concepts de spatialité primitive S_p et de temporalité primitive T_p dans l'ensemble des pictogrammes ainsi que l'ensemble des pictogrammes SIG primitifs $\overset{vl}{U}_p$:

Définition 71 L'ensemble des ***pictogrammes de spatialité primitive*** $\overset{vl}{S}_p$ est l'image de S_p par la fonction $f|_{S_p}$:

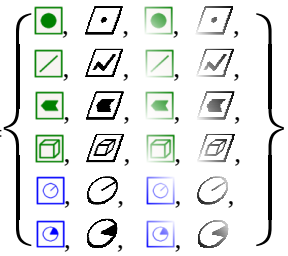
$$\overset{vl}{S_p} = f|_{S_p}(S_p) = \left\{ \right\}$$

Définition 72 L'ensemble des ***pictogrammes de temporalité primitive*** $\overset{vl}{T_p}$ est l'image de T_p par la fonction $f\big|_{T_p}$:

$$\overset{vl}{T_p} = f|_{T_p}(T_p) = \left\{ \right\}$$

Définition 73 L'ensemble des ***pictogrammes SIG primitifs*** $\overset{vl}{U_p}$ est l'union des ensembles $\overset{vl}{S_p}$ et $\overset{vl}{T_p}$.

$$\overset{vl}{S_p} = \overset{vl}{S_p} \cup \overset{vl}{T_p} = \Rightarrow \overset{vl}{S_p} = \left\{ \right\}$$



Comme pour les concepts SIG, les définitions de ces trois nouveaux ensembles remplacent celles énoncées au Chapitre 7-V.2 (cf. définitions 59, 60 et 61).

II.3.2 *Composites*

L'adjonction de tout nouveau concepts aux ensembles de concepts primitifs S_p et T_p suppose une redéfinition également du contenu des ensembles de concepts SIG composites $S^{\otimes}$, $T^{\otimes}$, $S^{\oplus}$, $T^{\oplus}$, $ST^{\oplus}$, S_T, S° et T°, de leurs images dans l'ensemble des pictogrammes et des images réciproques dans l'ensemble des entités référencées en incluant aussi les images réciproques des singletons.

Comme pour les concepts composites dérivés, nous n'avons pas jugé utile de faire des listes de noms de concepts composites ou des listes de pictogrammes composites ni ceux de l'ensemble des entités référencées images réciproques des singletons. C'est la raison pour laquelle, les nouveaux ensembles ne sont pas redéfinis ici bien qu'ils existent.

III INCIDENCE SUR LE METAMODELE SIG

Tout nouveau concept ajouté aux taxinomies implique une évolution du *Métamodèle SIG* de la figure 106. Ce dernier s'enrichit de la douzaine de concepts flous définis dans le présent chapitre. La figure 112 montre les huit métaclasses de spatialité floue et les quatre de temporalité floue qui ont été ajoutées pour mettre en conformité le métamodèle avec les nouvelles définitions des ensembles S_p, T_p, $\overset{vl}{S_p}$ et $\overset{vl}{T_p}$. La figure 112 montre aussi les cinq concepts flous (*Point Flou*, *Ligne Floue*, *Polygone Flou*, *Volume Flou* et *Instant Flou*) qui sont implémentés dans la phase de recherche actuelle. Les concepts flous dérivés sont dessinés en gris car ils seront implémentés ultérieurement dans le *Profil UML-SIG* lorsque l'atelier de génie logiciel permettra le multi-stéréotypage (cf. Chapitre 4-II.1).

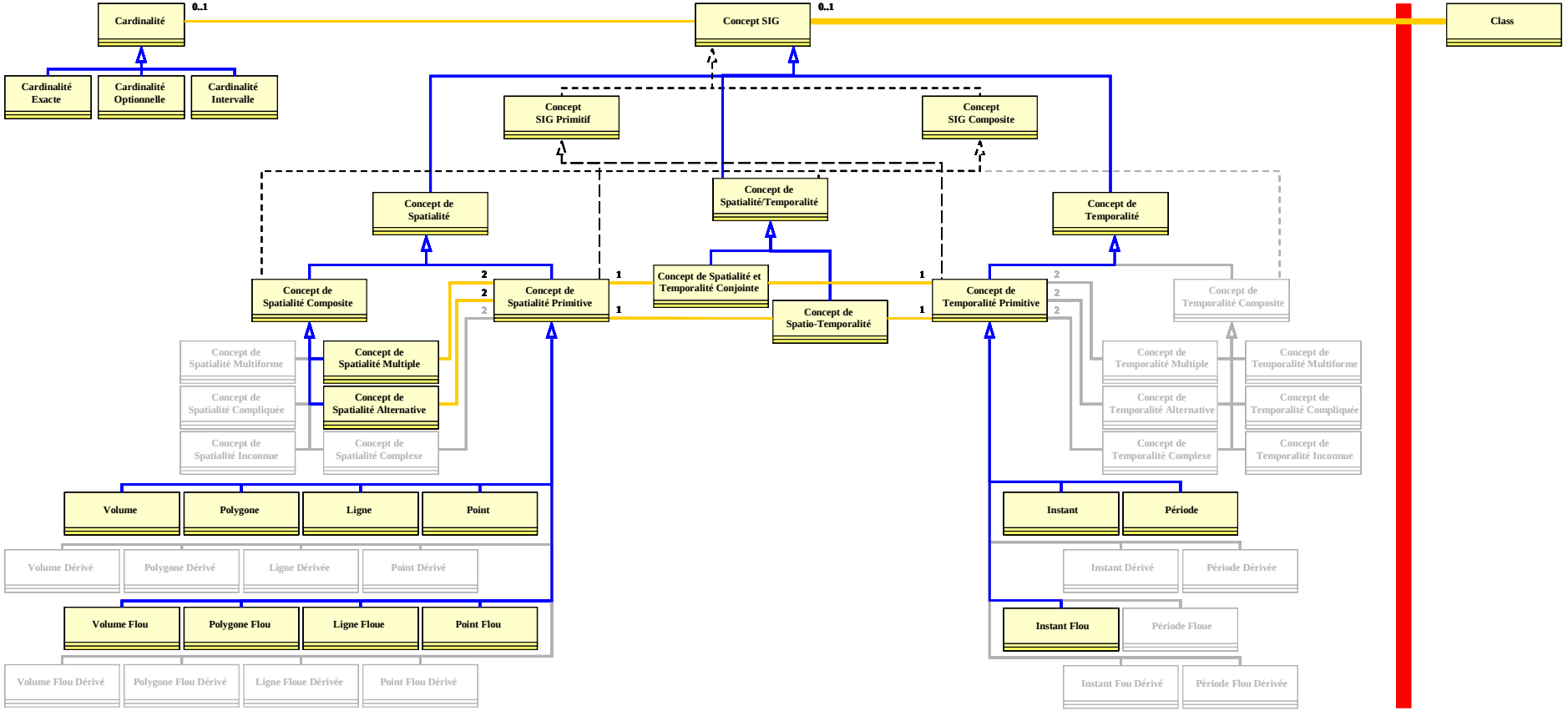


Figure 112 *Métamodèle SIG* intégrant les concepts flous et montrant en gris les concepts SIG non implémentés dans le *Profil UML-SIG*.

Au stade actuel de la recherche, le concept de temporalité *Période Floue* n'a pas été implémenté car il est plus délicat à définir d'où sa coloration en gris. En effet, l'incertitude peut porter sur la durée ou sur les dates de début et de fin. Par exemple, il est possible de déterminer précisément la durée d'une mesure (température, pression, etc.) avec un chronomètre sans pour autant connaître l'instant précis du début de la mesure. La durée est alors parfaitement connue mais sa position dans le temps est imprécise ou inconnue.

Inversement, sauf si une instrumentation adéquate a été installée sur le site, l'heure exacte de début et de fin d'une avalanche est souvent déterminée à dire d'acteurs. Ces derniers n'ont pas le réflexe de regarder une montre lorsque l'avalanche se déclenche. Au mieux, ils vont donner une estimation de l'heure à laquelle a commencé l'avalanche et à laquelle elle s'est terminée.

IV DEFINITION D'UN PATRON DE CONCEPTION ENTRE CONCEPTS SIG PRIMITIFS FLOUS ET NON FLOUS

Les concepts SIG primitifs flous ont été définis par rapport à leurs homologues non flous (cf. définition 66) en établissant une relation, nommée *Incertitude Floue*, entre concepts homologues. En figure 113, cette relation a été instanciée par une association entre le concept de *Polygone Flou* et celui de *Polygone* constituant de fait un « mini-modèle » entre ce couple de concepts. Pour compléter le mini-modèle, les cardinalités et les rôles de l'association ont été définis en construisant les phrases sémantiquement cohérentes du point de vu thématique en appliquant la méthode suggérée en Annexe I-III.4.1 :

- ⇨ Un objet *Polygone Flou* est en relation d'*Incertitude Floue* avec un et un seul (*1*) objet *Polygone* qui joue le rôle de *Polygone*.
- ⇨ Un objet *Polygone* est en relation *Incertitude Floue* avec *0* à *N* (*) objets *Polygone Flou* qui joue le rôle de *Polygone Flou*.

Bien entendu, le concept de *Polygone Flou* a un indice de probabilité permettant de valuer l'incertitude sous-jacente aux entités référencées.

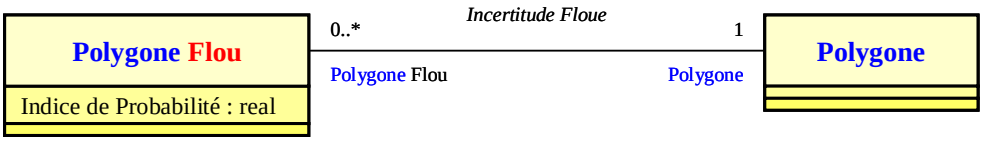


Figure 113 Mini-modèle entre le concept de *Polygone Flou* et son homologue non flou.

Une analyse approfondie de la figure 113 montre que le terme Polygone peut être remplacé par n'importe quel autre terme désignant un concept SIG primitif. Aussi, un modèle plus générique peut être obtenu en substituant le concept de *Polygone* par une variable **[u]** à valeur dans les ensembles S_p et T_p des définitions 55 et respectivement 56[85].

Le mini-modèle obtenu a toutes les caractéristiques d'un patron de conception entre couple de concepts flous et non flous. La figure 114 montre la généralisation du mini-modèle de la figure 113.

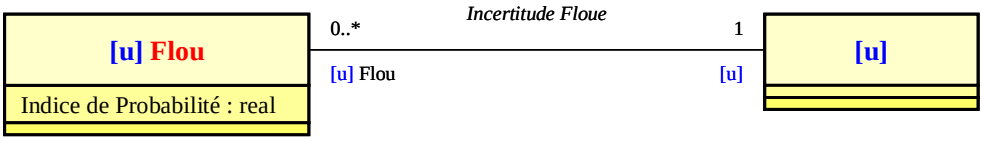


Figure 114 Patron de conception entre un concept SIG primitif flou et son homologue non flou.

[85] Attention : les ensembles S_p et T_p des définitions 19 et 20 sont incomplets car les concepts dérivés ont été introduits ultérieurement et les ensembles S_p et T_p des définitions 67 et 3 incluent les concepts SIG primitif flous en cours de définition.

L'avantage d'avoir établi un patron de concept entre couple de concepts flou et non flou est que la création d'un patron de conception est entièrement programmable dans un atelier de génie logiciel doté d'un langage de métaprogrammation (cf. Chapitre 5.III.5).

V APPLICATION DE LA SPATIALITE FLOUE ET DE LA TEMPORALITE FLOUE AUX EXEMPLES DE PRESENTATION DE LA PROBLEMATIQUE

Les exemples du paragraphe I.1 et I.2 ayant permis de poser la problématique de l'incertitude sont repris ici pour montrer comment les concepts de spatialité floue peuvent être mobilisés en phase d'analyse pour améliorer l'expression des propriétés spatiales des entités référencées au sein du diagramme de classe. La modélisation a été menée à son terme en décrivant les objets informatiques représentant les objets thématiques des figures 110 et 111.

V.1 Exemple de l'entité référencée Parcelle

En cours d'analyse, le concepteur d'un *Système d'Information Géographique* confronté à la modélisation de l'entité référencée *Parcelle* extraites d'image satellitale (cf. I.1) utilisera, s'il en dispose, du concept de *Polygone Flou* et annotera l'entité référencée *Parcelle* avec le pictogramme [86] (cf. figure 115).

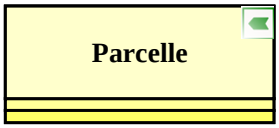


Figure 115 Modèle de l'entité référencée *Parcelle.*

L'entité référencée de la figure 115 permettra à un programmeur en phase d'implémentation d'instancier un objet informatique *Ma Parcelle* (cf. figure 116) qui sera en relation[87] avec deux polygones flous nommés *FP Haut* et *FP Bas.*

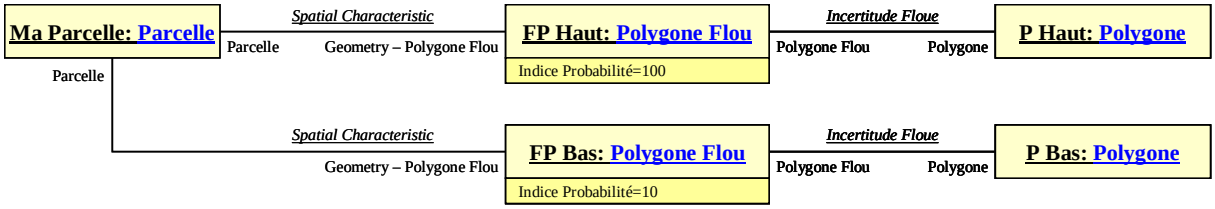


Figure 116 Modèle de l'objet informatique représentant la parcelle de la figure 110.

Le premier représentera le cœur de parcelle et aura un indice de probabilité par exemple de 100%. Il sera obtenu par segmentation de l'image avec la borne supérieure des rayonnements possibles alors que le second englobera en plus la frange de pixels suite à une segmentation avec la borne inférieure. L'indice de probabilité de ce dernier polygone flou sera par exemple de 10%. Chacun de ces deux polygones flous instanciera un objet polygone (*P Haut* et *P Bas*) conformément au patron de conception entre concepts flous et non flous (cf. IV).

[86] Ici, il aurait été préférable d'utiliser un pictogramme dérivé puisque les limites de l'entité référencée *Parcelle* sont déduites d'un traitement numérique. N'ayant pas implémenté les concepts dérivés pour les raisons évoquées au Chapitre 4-IV, nous avons utilisé le concept primitif non dérivé.

[87] La relation entre l'objet *maParcelle* et les polygones flous *FP Haut* et *FP Bas* est détaillée au Chapitre 5-III.5.3.1.

V.2 Exemple de l'entité référencée Zone Marécageuse

En cours d'analyse, le concepteur en charge du développement du *Système d'Information Géographique* destiné à la gestion du territoire décrit au paragraphe I.2 créera dans un premier temps les entités référencées *Lac*[5] et *Rive*[5] puisque ce sont des composantes essentielles du territoire. Ces deux entités sont en relation de *Composition* et, dans cette relation, le concept de Rive joue le rôle de *Limite*.

Pour transcrire le fait que les rives bordées par la forêt, par la zone agricole et par le marécage ont des propriétés et des comportements différents, le concepteur spécialisera le concept de *Rive* avec les entités référencées *Rive Forestière*, *Rive Agricole* et *Rive Marécageuse*.

Les deux premières entités référencées étant des linéaires parfaitement délimités, le concepteur les annotera avec le pictogramme  . Par contre, il appose le pictogramme flou  sur la rive marécageuse puisque la limite de l'eau dans cette zone est imprécise et que les acteurs du territoire souhaitent disposer d'une information précise sur la probabilité de présence d'eau dans la zone marécageuse. Le modèle de la figure 117 est le résultat de cette analyse.

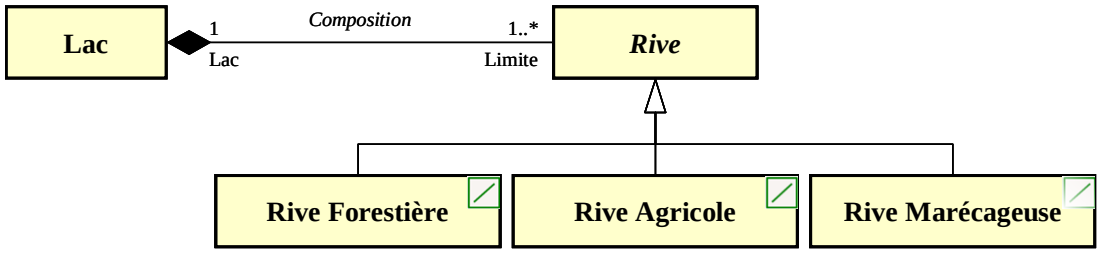


Figure 117 Modèle de l'entité référencée *Lac* issue de la description du territoire.

Grâce à ce modèle, un programmeur pourra instancier en phase d'implémentation un objet informatique *Mon Lac* (cf. figure 118) correspondant à la description faite par les acteurs. Pour cela, il instanciera un premier objet de typologie *Rive Forestière* qu'il pourra nommer *Rive Gauche*, un deuxième de typologie *Rive Agricole* qu'il désignera *Rive Droite* et un troisième objet de typologie *Rive Marécageuse* qu'il qualifiera de *Rive Amont*. Les deux objets *Rive Gauche* et *Rive Droite* ayant des propriétés de spatialité linéaire auront une relation avec des instances *Rive Ga.* et respectivement *Rive Dr.* de la classe *Ligne*. L'entité référencée *Rive Marécageuse* étant de nature floue, elle aura quant à elle une série de relations vers des instances de la classe *Ligne floue* (*FL Haute*, *FL Moy.* et *FL Basse*) correspondant chacun à la limite de la zone marécageuse en fonction de la probabilité de présence en eau. Ces trois objets flous ont chacun une relation d'*Incertitude Floue* avec un objet *Ligne* (*F Haute*, *F Moy.* et *F Basse*), objet porteur des coordonnées.

VI CONCLUSION

En phase de modélisation, les concepts flous viennent enrichir la capacité d'expression des langages SIG et pictogrammique. Doté seulement de trente concepts, ces langages offrent une panoplie de concepts et de pictogrammes qui permet de modéliser la plupart des entités référencées rencontrées de façon relativement fine. En effet, cette trentaine de concepts primitifs peuvent être combinés entre-eux pour décrire des entités référencées composites, à spatialité et temporalité conjointe, à spatio-temporalité, à spatialité ou temporalité complexe ou bien encore les différents types de spatialité particulière et de temporalité particulière.

Dans la phase actuelle de faisabilité, seulement onze concepts SIG et les pictogrammes associés ont été implémentés dans le *Profil UML-SIG* car l'atelier de génie logiciel utilisé ne supporte pas le multi-stéréotypage (cf. Chapitre 4-II.1) ce qui ne saurait tarder avec la mise sur le marché d'une nouvelle version implémentant UML 2.0.

Les concepts de spatialité floue et de temporalité floue définis dans ce chapitre sont une solution adaptée à la problématique posée au début du chapitre. Les figures 116 et 118 montrent les objets informatiques que le programmeur pourra instancier lors de l'implémentation. Ces deux objets informatiques correspondent bien à la description du concept thématique des figures 110 et respectivement 111.

Il est à noter que, bien que (Shu *et al.*, 2003) aient posé des définitions, le dernier ouvrage consacré au formalisme MADS (Parent *et al.*, 2006) élude les concepts de spatialité floue et de temporalité floue. En tout état de cause, l'application informatique créée dans le cadre du projet européen MurMur n'implémente pas cette typologie de concepts. Le projet MurMur s'est achevé en 2002 (MurMur, 2000) et les travaux de Hong Shu sont postérieurs.

L'atelier de génie logiciel Objecteering doté du Profil UML-SIG est donc le premier outil de conception de Systèmes d'Informations Géographique permettant de façon opérationnelle la modélisation des concepts de spatialité floue et de temporalité floue et propose un modèle implémenté quelque soit la base de données.

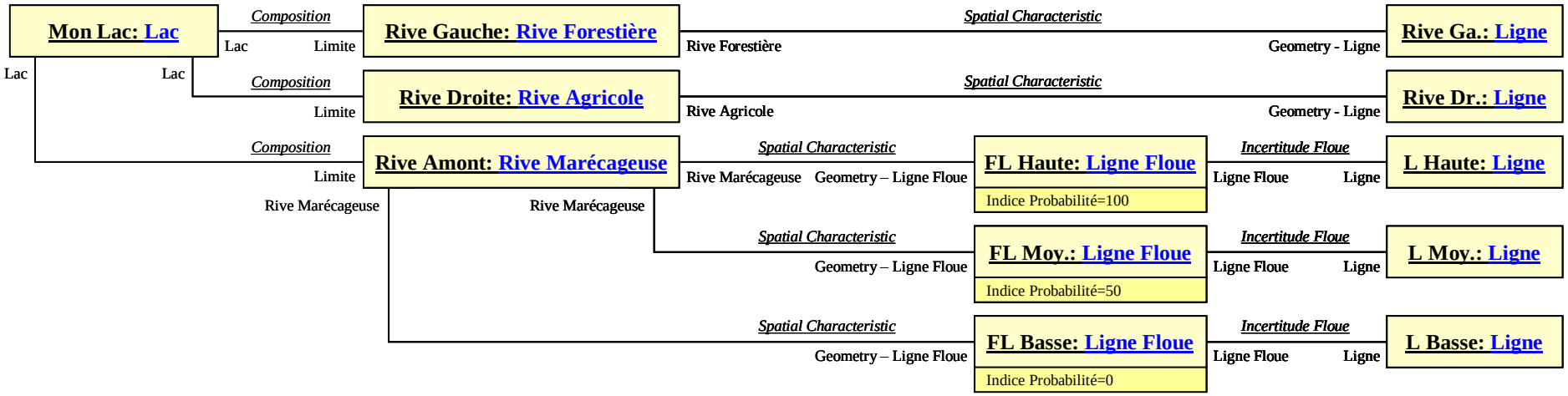


Figure 118 Modèle de l'objet informatique représentant le lac de la figure 111.

Conclusion Générale

Comme cela a été évoqué en Introduction, le présent travail de recherche est à la conjonction de l'Information Géographie et de l'Informatique. De ce fait, une partie de la contribution relève du domaine Géographique et une autre du domaine Informatique. Cette dichotomie, qui a sous-tendue les travaux de recherche, a structuré le contenu du mémoire et structure aussi celui de la conclusion.

L'objectif général de la recherche était de :

> Réaliser un ***outil d'aide à la conception de Systèmes d'Information Géographique*** adapté à un ***processus de développement permettant le prototypage rapide en séance d'analyse*** et assurant la ***capitalisation des connaissances***.

(cf. Introduction-IV)

Cet objectif a été atteint puisque l'outil implémenté dans l'atelier de génie logiciel Objecteering permet de concevoir des *Systèmes d'Information Géographique*, l'artefact multimodèle *Software Development Process Model* structure et gère les connaissances mobilisées au cours du développement et les transformations automatisent l'évolution des modèles depuis l'analyse jusqu'à l'implémentation suivant un processus full MDA.

L'outil a été implémenté en mettant en œuvre le mécanisme de profil, mécanisme qui permet d'étendre un métamodèle, celui d'UML en l'occurrence. En interne cet outil est lui aussi structuré par la dichotomie Information Géographie/Informatique puisqu'il est constitué de deux profils :

- ⇨ Un ***Profil UML-SIG*** mettant à la disposition du concepteur un langage pictogrammique[88] qui :
  - Permet d'exprimer la spatialité floue et la temporalité floue des entités référencées.
  - Facilite la saisie des spatialités et des temporalités des entités référencées ainsi que la dimension du système de coordonnées dans lequel elles sont représentées.
- ⇨ Un ***Profil MDA*** offrant un cadre de conception et de développement des applications qui :
  - Permet quant à lui de mettre en œuvre la méthode *Software Development Process Approach*, approche conçue spécialement pour réaliser une application suivant un *processus de développement permettant le prototypage rapide en séance d'analyse*.
  - Assure la *capitalisation des connaissances* au sein de l'artefact multimodèle *Software Development Process Model*.

I CONTRIBUTION GEOMATIQUE : DERIVATION D'UN METAMODELE SIG ET D'UN PATRON DE CONCEPTION SIG

Une ***première contribution*** de nature géomatique concerne la méthode de construction du *Métamodèle SIG*. Cette contribution est induite par l'objectif élémentaire de créer un *outil d'aide à la conception de Systèmes d'Information Géographique*. Pour décrire la spatialité et la temporalité des entités manipulées en *Information Géographique*, il était nécessaire d'adopter un formalisme ou un langage. Pour ce faire, la première étape classique à toute recherche consistait à faire une étude bibliographique des méthodes et formalismes existants.

[88] La capacité d'expression du langage pictogrammique implémenté a été limitée à cause de contraintes techniques rencontrées, contraintes qui viennent d'être levées (cf. Chapitre 4-II)

La difficulté à comparer ces méthodes et ces formalismes a conduit à imaginer une technique facilitant cette comparaison. Elle s'appuie sur le choix a priori d'un *formalisme de référence* qui est transcrit en *métamodèle de référence*. Une fois ce dernier établi, les concepts des autres méthodes ou formalismes sont reportées sur le *métamodèle de référence*, la comparaison est alors plus synthétique et donc plus aisée (cf. Chapitre 1-IV).

Dans la présente recherche, le *formalisme de référence* adopté est le langage pictogrammique de l'atelier de génie logiciel Perceptory lequel a permis le définir le *Métamodèle SIG*. Ce dernier a été établi en appliquant la Théorie des ensembles à la terminologique des propriétés spatiales et temporelles. Cette lecture ensembliste des concepts du domaine a permis de construire une taxinomie générale constituée de trois sous-taxinomies :

- ⇨ La *Taxinomie SIG* qui comprend les concepts de spatialité et de temporalité.
- ⇨ La *Taxinomie des Pictogrammes* composée des pictogrammes représentant les concepts de spatialité et de temporalité précédents.
- ⇨ La *Taxinomie des entités référencées* constituée des concepts thèmatiques du système étudié annotés de concepts de spatialité et de temporalité.

Ayant fait le constat de la proximité conceptuelle de la notion de sous-ensemble de la théorie des ensembles et de la notion de généralisation/spécialisation du langage UML (cf. figure 106), il a été possible de représenter la taxinomie générale suivant un *formalisme hybride* empruntant la représentation ellipsoïdale des concepts aux diagrammes de Venn et la représentation de la généralisation/spécialisation au langage UML (cf. Chapitre 7-IV.1.2).

La représentation d'une taxinomie suivant le formalisme hydride (cf. figure 98) étant similaire à celle d'un diagramme UML, il a été assez facile de définir un mécanisme de dérivation de la taxinomie générale pour obtenir un diagramme UML. Ce dernier est un métamodèle et pas un modèle car, pour construire la taxinomie générale, il a été nécessaire d'identifier et de définir d'une part, les relations entre les trois sous-taxinomies et, d'autre part, les lois de composition des concepts d'une même sous-taxinomie. Ces relations et ces lois de composition constituent la syntaxe du langage du domaine, SIG en l'occurrence.

La méthode de dérivation d'un métamodèle est totalement généralisable à d'autres domaines. La principale difficulté de cette méthode réside dans la formalisation mathématique des concepts et des lois de composition.

Une ***deuxième contribution*** de nature géomatique est liée à la création du *Patron de conception SIG*. Par construction, le *Métamodèle SIG* renferme tous les concepts de spatialité et de temporalité du domaine de l'*Information Géographique*. Ayant fait le constat qu'un certain nombre de concepts de ce métamodèle sont de nature informative[89] (cf. Chapitre 7-VIII), il a été assez simple d'épurer le *Métamodèle SIG* de tous les concepts « superflus » pour l'implémentation d'un *Système d'Information Géographique*. Les concepts restants constituent le cœur du *Patron de conception SIG* (cf. Chapitre 5-III.5).

Enfin, une ***troisième contribution*** de nature géomatique est liée à l'extension du langage pictogrammique de Perceptory aux concepts de spatialité floue et temporalité floue. Doté d'un langage pictogrammique étendu, l'*outil d'aide à la conception de Systèmes d'Information Géographique* implémenté dans Objecteering est le seul, à notre connaissance, proposant un formalisme de description des spatialités floues et temporalités floues des entités référencées. Le formalisme de description est, grâce aux travaux en ingénierie des modèles, exploité pour produire de code SQL permettant de créer la base de données d'un *Système d'Information Géographique*.

II CONTRIBUTION EN INGENIERIE DES MODELES

La méthode de conduite de projet *Continuous Integration Unified Process* est une ***première contribution*** à l'ingénierie des modèles. Cette méthode permet de mettre en œuvre un *processus de développement permettant le prototypage rapide en séance d'analyse*. Le prototype étant l'objet technique qui améliore la communication entre les acteurs du domaine étudié et le concepteur. Cette méthode permet de rechercher l'excellence sémantique du modèle au cours du cycle de prototypage rapide et de reporter la recherche de l'excellence technique au cycle principal.

La *Software Development Process Approach* constitue la ***seconde contribution*** à l'ingénierie des modèles. Cette nouvelle approche généralise l'idée de *capitalisation des connaissances* sous-jacente dans l'approche MDA. Elle est fondée sur le constat que la capitalisation des connaissances est une préoccupation constante du

[89] La spatialité inconnue et la temporalité inconnue par exemple ou bien la spatialité dérivée et la temporalité dérivée qui indique que le concept annoté doit posséder une opération effectuant un calcul sur la spatialité et/ou la temporalité des entités référencées.

développement d'une application. Ce constat a conduit à associer un modèle à chacune des phases du cycle de développement.

Le *Software Development Process Model* est la ***troisième contribution*** à l'ingénierie des modèles. Cet artefact multimodèle est monolithique car il contient tous les concepts, les spécifications, etc. mobilisés pour le développement de l'application. **Selon notre vision, le *Software Development Process Model* est LE MODÈLE de l'application**. C'est un artefact multimodèle qui structure et gère en cohérence, grâce à l'architecture de traçabilité de clonage, les sous-modèles associés aux phases du cycle de développement. Ces sous-modèles peuvent être vus comme étant des plans de capitalisation.

Les neuf transformations qui automatisent l'évolution des modèles depuis l'analyse jusqu'à l'implémentation sont la ***quatrième contribution*** à l'ingénierie des modèles.

La *Transformation de diffusion* et celle de rétrodiffusion sont fondamentales car, associées à l'architecture de clonage, elles assurent le fonctionnement et une grande partie du maintien en cohérence de l'artefact *Software Development Process Model*. En complément de ces deux transformations, il a été nécessaire de concevoir, lorsque cela est nécessaire, des prétraitements et post-traitements pour corriger les incohérences introduites par le processus itératif adopté. Ces prétraitements et post-traitements sont spécifiques à la diffusion des concepts entre deux sous-modèles.

Sans l'architecture de traçabilité de clonage ou sans les transformations de diffusion et de rétrodiffusion ou bien encore sans les prétraitements et les post-traitements, la gestion du *Software Development Process Model* serait impossible. C'est l'articulation de cet ensemble qui permet de gérer en cohérence cet artefact.

Quatre transformations de nature géomatique permettent d'une part, la saisie des propriétés spatiales et/ou temporelles des entités référencées sous forme d'annotations et, d'autre part, la conversion de ces annotations en éléments de modélisation UML interprétables par les générateurs de code de l'atelier de génie logiciel.

Les *Transformations de saisie et de suppression de la pictogrammie* facilitent ces opérations mais ont aussi pour responsabilité de maintenir en cohérence le *Software Development Process Model*. Par exemple, la suppression de la spatialité (Point, Ligne, etc.) doit s'accompagner de celle de la dimension du système de coordonnées (1D, 2D et/ou 3D) afin que le modèle ne soit pas « pollué ».

Les *Transformations de Génération du Patron de conception SIG et de Traduction des Pictogrammes* interprétent les annotations de spatialité et de temporalité et enrichissent le modèle des concepts SIG qui seront implémentés dans la base de données.

La première génère un patron de conception spécifique au domaine de l'*Information Géographique* ne contenant que des concepts de spatialité et/ou de temporalité (Instant et Période) et les relations entre ces concepts. Ce *Patron de conception SIG* est dérivé de la taxinomie générale effectué lors de l'étude terminologique. Une fois généré, il peut être amélioré à volonté par le concepteur si le besoin s'en fait sentir.

Le modèle en cours de réalisation et le *Patron de conception SIG* étant dissociés, la *Transformation de traduction des pictogrammes* réalise le couplage. De fait, cette transformation instancie l'application d'affectation **<<** des propriétés spatiales et/ou temporelles aux entités référencées. Cette application a été identifiée et définie lors de l'étude terminologique.

Enfin, les trois transformations SQL (*Transformation de spécification multiple*, celle de suppression des opérations et celle d'externalisation de la relation de composition) adaptent le modèle diffusé depuis le sous-modèle de conception préliminaire au générateur de code SQLDesigner de l'atelier de génie logiciel Objecteering. Leur particularité principale est d'avoir été implémentée pour fonctionner suivant le processus itératif de la méthode de conduite de projet *Continuous Integration Unified Process*.

Perspectives

Les perspectives de recherche présentées ici sont soit des travaux à cours terme qui s'inscrivent dans la finalisation des travaux actuels (cf. III), soit des travaux à moyen terme qui vont permettre d'explorer de nouvelles pistes de recherche (cf. IV).

III À COURT TERME

III.1 Implémentation complète du Langage Pictogrammique

La nouvelle version de l'atelier de génie logiciel Objecteering, commercialisée récemment, implémente le langage UML 2.0 et le multi-stéréotypage levant ainsi le verrou technologique rencontré au cours de cette recherche (cf. Chapitre 4-II).

Une **première perspective** va consister à implémenter le langage pictogrammique de Perceptory étendu aux concepts de spatialité floue et de temporalité floue dans sa totalité et particulièrement la spatialité dérivée et temporalité dérivée. La capacité d'expression de l'*outil d'aide à la conception de Systèmes d'Information Géographique* sera ainsi décuplée.

III.2 Mise en œuvre de la méthode Continuous Integration Unified Process : Génération automatique des Interfaces Homme/Machine

Comme indiqué en Introduction au stade actuel de la recherche, la méthode *Continuous Integration Unified Process* ne peut pas être pleinement appliquée car les interfaces Homme/Machine ne sont pas générées automatiquement.

Une **deuxième perspective** est donc de concevoir et d'implémenter les transformations de génération automatique des interfaces Homme/Machine. Une analyse rapide a permis de fixer les grandes lignes du mécanisme de génération. Dans l'immédiat, deux transformations ont été identifiées :

- ⇨ La première transformation, de nature PIM/PIM, sera appliquée au sein de sous-modèle de conception avancée. Elle générera automatiquement une description indépendante de toute plate-forme de l'interface Homme/Machine. Le concept à afficher *People* sera dérivé en *People UI* qui lui décrit la forme de l'affichage (cf. figure 119). Cette première étape présente l'avantage de permettre une intervention manuelle du concepteur si le besoin s'en fait sentir. En effet, il est fort probable que l'interface ne correspond pas à la vision que s'en font les acteurs, dans ce contexte, le concepteur doit pouvoir modifier l'interface afin qu'elle soit conforme aux besoins des acteurs.
- ⇨ Après diffusion du concept *People UI*, la seconde transformation, de nature PSM/PSM, aura lieu sur les sous-modèles d'implémentation. Elle sera en charge de produire l'implémentation finale des interfaces Homme/Machine ce qui nécessitera de mobiliser des concepts propres à chacun des langages de programmation. Comme le montre la figure 119, les sous-modèles d'implémentation présenteront de légères différences.

L'avantage de cette démarche est qu'elle est entièrement automatisable à l'exception des quelques retouches qu'aura à faire le concepteur pour rendre conforme l'interface à la vision des acteurs.

Ce volet de la recherche va être effectué au cours du premier semestre 2007 dans le cadre du projet de *Conception d'Observatoires de Pratiques Territorialisées* (COPT) financé par *Agence Nationale de la*

Recherche. L'objectif général de ce projet est de *mener une réflexion sur la construction des dispositifs d'observation des pratiques agricoles au niveau du territoire*.

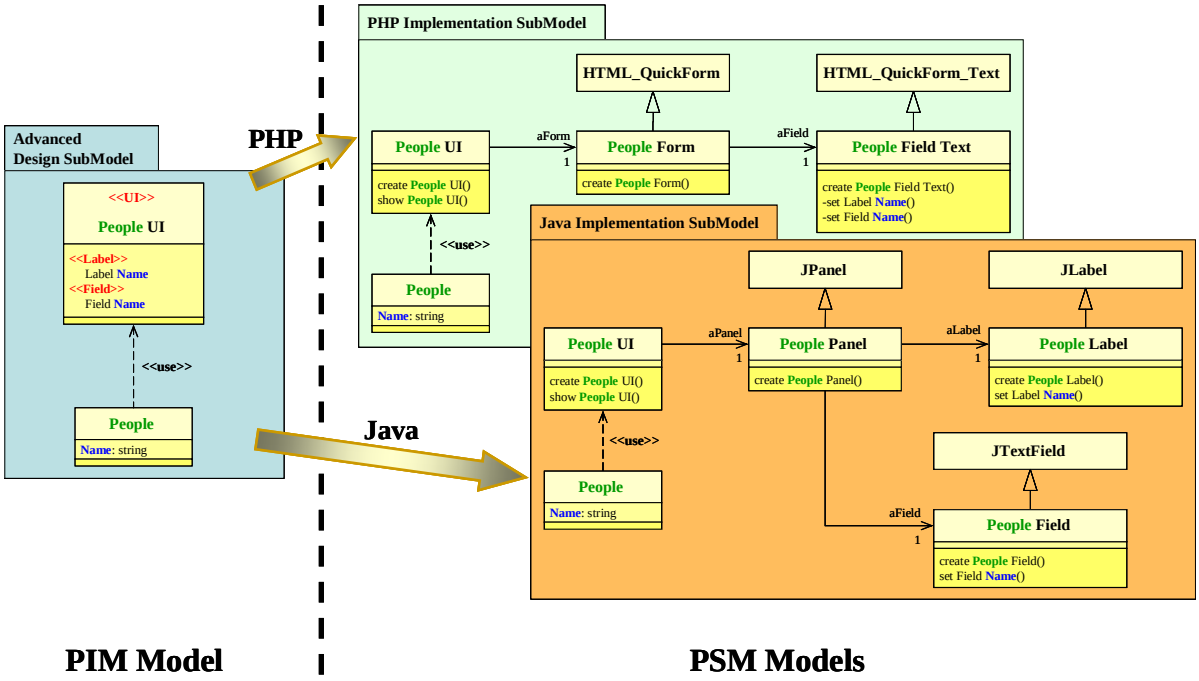


Figure 119 Principe de la transformation de génération automatique des interfaces Homme/Machine.

Le Work Package 2, intitulé *Conception d'un schéma type d'observatoire et utilisation du langage UML*, prévoit :

- ⇨ De créer un *générateur d'applications* simple pouvant produire un modèle générique des observatoires.
- ⇨ D'implémenter ce modèle générique dans un Système d'Information de façon la plus automatique possible. C'est dans cette opération que s'intègre la génération automatique des interfaces Homme/Machine.

La génération du modèle générique et son implémentation étant deux opérations découplées, le modèle générique pourra être adapté à la situation locale du territoire par les acteurs du domaine.

Les interfaces Homme/Machine prévues dans le projet COPT sont des interfaces simples permettant la saisie de données alphanumériques et de leur restitution sous forme alphanumérique. La saisie et la représentation sous forme cartographique seront abordées ultérieurement dans le cadre d'un autre projet.

À la demande des utilisateurs, une architecture client-serveur a été adoptée afin de pouvoir accéder à l'application via Internet ou un Intranet. Le choix du langage PHP est issu de cette exigence des utilisateurs. L'atelier de génie logiciel utilisé ne possédant pas de générateur de code PHP, un générateur minimal va être développé.

III.3 Publication

Une **troisième perspective** est la publication d'une partie des travaux dans un ouvrage, intitulé *Advances in Modelling Agricultural Systems*, coédité par Petraq Papajorgji et Panos Pardalos. Trois chapitres de cet ouvrage intégreront les réflexions et les acquits de la présente recherche :

- ⇨ The *Model Driven Architecture* (MDA) approach: a framework for developing complex agricultural systems (Papajorgji, P. & Miralles, A.).
- ⇨ A new methodology to automate the transformation of GIS models in an iterative development process (Miralles, A. & Libourel, T.).
- ⇨ Application of Model Transformation Paradigm in Agriculture: a simple farm system case study (Miralles, A. & Libourel, T.).

IV À MOYEN TERME : THESE SUR LA FACTORISATION DE MODELES

Une **quatrième perspective**, la plus intéressante du point de vue scientifique, est une nouvelle piste de recherche qui vient prolonger des travaux de recherche du LIRMM effectués au sein de l'équipe-projet *Données Objets Composants pour les systèmes complexes* (équipe D'OC).

Cette piste de recherche est dans la continuité des travaux présentés ici. Un des constats majeurs de la présente recherche est que l'analyse est une phase cruciale du développement d'une application car c'est lors de cette phase que les concepts du domaine et les exigences des acteurs sont transmises au concepteur de l'application. Aussi, la qualité finale de l'application et son adéquation aux besoins des acteurs dépend beaucoup du résultat de cette phase.

Fort de ce constat, il est important de mettre en œuvre tous les moyens, méthodes, pratiques, etc. qui peuvent faciliter la capture des concepts et des exigences. La méthode *Continuous Integration Unified Process* a été conçue dans ce seul but. Son cycle de prototypage rapide en phase d'analyse permet d'accroître la communication entre acteurs et concepteur ce qui facilite la capture les concepts du domaine et les exigences des acteurs.

La méthode *Continuous Integration Unified Process* comme beaucoup d'autres est fondée sur l'hypothèse que tous les acteurs concernés sont présents lors de l'analyse du système ou d'un sous-système. Dans ce contexte, l'analyse est faite en groupe et un modèle unique est produit (cf. figure 120-Analyse en groupe). Ce modèle unique est le résultat de la confrontation des différentes perceptions et visions qu'ont les acteurs du système étudié.

Or, l'hypothèse de présence simultanée des acteurs n'est pas toujours vérifiée. Dans certains cas, les experts du domaine ont des plannings très chargés ce qui les rend difficilement *mobilisable simultanément de façon répétitive et soutenue* (Introduction-II.1) comme l'exige les méthodes de conduite de projet récentes. Dans ce contexte, il est préférable de changer d'hypothèse et d'admettre que les acteurs peuvent effectuer l'analyse du système individuellement ou par typologie d'acteurs. La conséquence est qu'il y a de forte chance qu'il n'y ait plus qu'un seul modèle d'analyse mais plusieurs, à la limite un par acteur (cf. figure 120-Analyse en individuelle).

Les difficultés techniques consécutives à ce contexte de travail résident alors dans la *gestion de plusieurs modèles* d'analyses et dans la *factorisation de ces modèles* en un seul « modèle générique » (cf. figure 120-Analyse en individuelle).

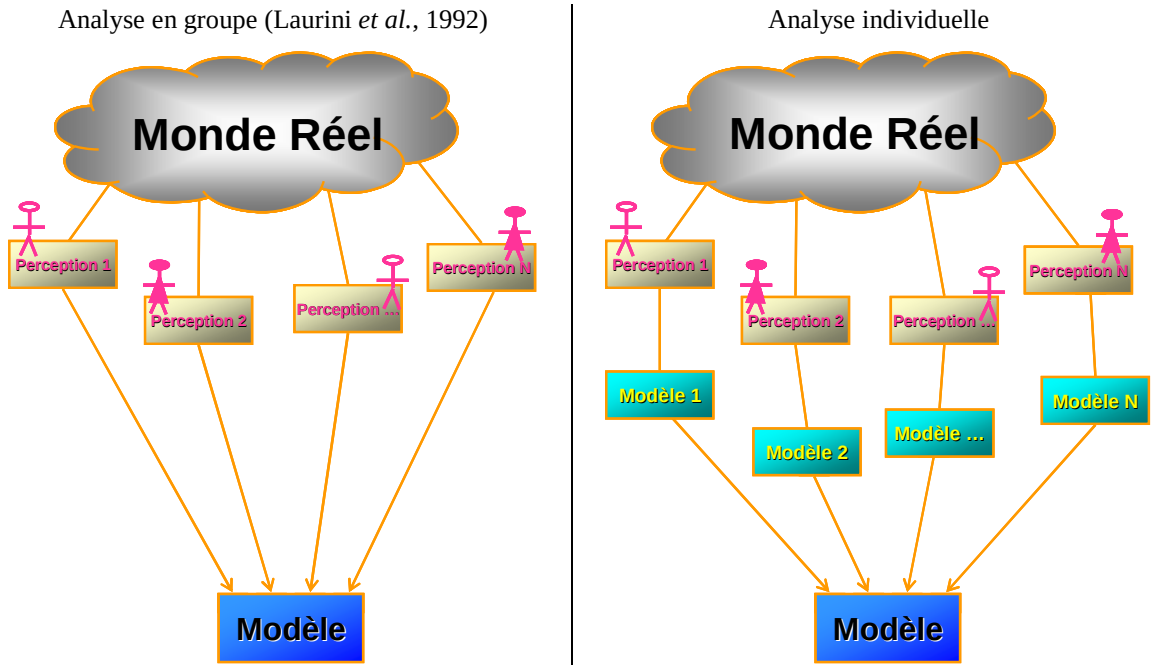


Figure 120 Approches d'analyse en groupe et d'analyse individuelle.

Cette problématique de factorisation intervient aussi dans d'autres contextes. C'est fréquemment le cas en *Information Géographique* où les entités spatiales et/ou temporelles sont souvent le seul objet technique qui fédère les points de vue des différents acteurs sur un territoire. De ce fait, il n'est pas rare qu'il existe des données disséminées dans différents services, organismes, etc.

Dans un processus de modélisation, les premières étapes de l'analyse se nourrissent de l'inventaire de l'existant or les données disséminées font partie de cet existant. Qu'elles soient bien structurées ou non, les données ont souvent un schéma connu. Comme précédemment, les difficultés techniques sont la *gestion de plusieurs schémas* issus de services, organismes, etc. différents et la *factorisation de plusieurs schémas* en un « schéma générique ».

L'objectif qu'aura à satisfaire cette nouvelle thèse sera de concevoir les mécanismes et les transformations permettant de factoriser les concepts disséminés au sein de plusieurs modèles afin de produire un modèle générique unique. Le titre provisoire de cette thèse est *Transformation automatisée de modèles*[90] *de Systèmes d'Information Géographique*.

Ce travail de recherche sera effectué en cotutelle entre l'Université Laval (Canada) et l'Université Montpellier II et mobilisera les compétences de l'équipe d'Yvan Bédard (Université Laval), de l'équipe-projet D'OC (LIRMM) et de l'UMR TETIS (Cemagref). Comme évoqué précédemment, ce travail de recherche s'inscrira dans la continuité des travaux effectués à :

- ⇨ L'Université Laval par (Brodeur, 2004) sur la proximité géosémantique, par Tarek Sboui sur la gestion des ontologies géospatiales, par Mohamed Bakillah sur l'évolution de la sémantique des objets spatiaux dans un cube de données, etc.
- ⇨ L'Université Montpellier II par (Huchard, 2003) et (Roume, 2004) sur les treillis et les propriétés mathématiques des sous hiérarchies de Gallois.

Le travail de recherche bénéficiera de l'apport des travaux présentés ici puisque l'une des difficultés sera la gestion de plusieurs modèles or le *Software Development Process Model* est un artefact conçu pour assumer cette gestion.

Par ailleurs, le concepteur pourra toujours profiter de facilités offertes par la méthode *Continuous Integration Unified Process* avec son cycle de prototypage rapide lors de l'analyse du système avec des acteurs individuels.

[90] Selon le cas il peut s'agir du modèle d'analyse ou du schéma de la base de données spatio-temporelles.

Principe structurant la recherche

Dans cette phase de conclusion des travaux et de synthèse d'idées, il est intéressant de revenir sur deux des trois principes qui ont présidé à la conception de la panoplie de transformations de modèles qui permettent de faire évoluer automatiquement un modèle depuis l'analyse jusqu'à l'implémentation (cf. Chapitre 5- I). Les deux principes qui nous intéressent sont repris in extenso ci-après :

Principe 1 : Le prototype et l'application forment un tout indissociable qui doit évoluer par incréments conformément au principe de la méthode *Continuous Integration Unified Process*.

Principe 3 : Les transformations automatiques du cycle de prototypage rapide ne doivent jamais détruire ou affecter les ajouts de concepts ou les modifications réalisées par le concepteur au cours du cycle principal.

L'application de ces deux principes revient à faire de la *capitalisation des connaissances* rejoignant par la même occasion l'objectif général de la recherche. Ces deux principes relèvent en fait d'un *principe plus général* selon lequel :

Principe structurant la recherche

Au cours du processus de développement d'une application, il faut favoriser à tout instant et par tous les moyens la capitalisation des connaissances et limiter autant que faire ce peut la destruction d'une connaissance.

Références bibliographiques

ALEXANDER, C., ISHIKAWE, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I. & ANGEL, S. (1977). *A Pattern Language*, New York

AMSILI, P. (2005). *Chapitre 1 : Bases mathématiques de la théorie des langages formels : Rappels de th´eorie des ensembles*, http://www.linguist.jussieu.fr/~amsili/Ens/poly-li063-1.pdf, Dernier accès: Mars 2006.

ARGOUML. (Site Web non daté). *Site ArgoUML*, http://argouml.tigris.org/.

ATKINSON, C. & KÜHNE, T. (2000, 2-6 October). *Strict Profiles: Why and How*. UML 2000 -- The Unified Modeling Language, Third International Conference, York, UK, pp. 309-322.

BECK, K. (2000). *eXtreme Programming explained - Embrace change*. Addison-Wesley. 190p.

BEDARD, Y. (1998, Dernière mise à jour: 06 avril 2004). *Site Perceptory*, http://sirs.scg.ulaval.ca/perceptory/, Dernier accès: Janvier 2005.

BÉDARD, Y. (1999a). Visual Modeling of Spatial Databases Towards Spatial Extensions and UML. *Geomatica, Vol. 53, n°2*, pp. 169-186.

BEDARD, Y. (1999b, 15 décembre). *PERCEPTORY : Nouvel Atelier de Génie Logiciel pour la modélisation des bases de données spatiales et spatio-temporelles.* Journée scientifique du Cemagref-Engref, Montpellier - France

BEDARD, Y. (1999c). Visual modelling of spatial databases: towards spatial PVL and UML. *Geomatica, Vol. 53, n°2*, pp. 169-186.

BÉDARD, Y. & PAQUETTE, F. (1989, 2-7 April). *Extending entity/relationship formalism for spatial information systems, AUTO-CARTO 9.* Ninth International Symposium on Computer-Assisted Cartography, Baltimore (United-States), pp. 818-827.

BEDARD, Y., PAGEAU, J. & CARON, C. (1992, 1-14 August). *Spatial Data Modeling: The Modul-R Formalism and CASE Technology.* ISPRS Symposium, Washington (United-States), pp. 10p.

BÉDARD, Y., NORMAND, P. & LARRIVÉE, S. (1998). *Modélisation des contraintes d'intégrité spatiale*. Rapport de recherche remis au Service de la cartographie MRN. 76p.

BEDARD, Y., LARRIVEE, S., PROULX, M.-J. & NADEAU, M. (2004, November). *Modeling Geospatial Databases with Plug-ins for Visual Languages: A Pragmatic Approach and the Impacts of 16 Years of Research and Experimentations on Perceptory.* ER Workshops 2004 CoMoGIS, Shanghai (China), pp. 17-30.

BÉDARD, Y., CARON, C., MAAMAR, Z., MOULIN, B. & VALLIÈRE, D. (1996). Adapting Data Models for the Design of Spatio-temporal Databases. *Computer, Environment and Urban Systems, Vol. 20, n°1*, pp. 19-41.

BÉDARD, Y., VAN CHESTEIN, Y., LARRIVÉE, S., NORMAND, P. & BOURGON, J. F. (1999). *Recherche et développement relative à la mise en place d'une nouvelle approche de gestion et d'exploitation de la BNDT*. Rapport de recherche remis au CITS. 75p.

BELAUNDE, M. (2004). *Petit point sur le RFP MOF - Query/View/Transformation*, http://meta.lip6.fr/index.php?lng=fr, http://meta.lip6.fr/file/presentation/04_09_02_MB.zip, Dernier accès: Septembre 2005.

BELAUNDE, M., BÉZIVIN, J. & MARVIE, R. (2004). *Transformations et modèles de plates-formes*. Dans *Ingénierie des modèles - Logiciels et systèmes (ARAGO 30)* (pp. 89-103). Lavoisier - Tec & Doc.

BENARD, J.-L. (2001). *Méthodes agiles (1) - Panorama*, http://www.devreference.net/devrefv205.pdf, Dernier accès: Septembre 2004.

BENARD, J.-L. (2002a). *Méthodes agiles (7) - RAD*, http://www.devreference.net/devrefv211.pdf, Dernier accès: Septembre 2004.

BENARD, J.-L. (2002b). *Méthodes agiles (7) - Unified Process*, http://www.devreference.net/devrefv212.pdf, Dernier accès: Septembre 2004.

BENARD, J.-L. (2002c). *Méthodes agiles (2) - eXtreme Programming*, http://www.devreference.net/devrefv206.pdf, Dernier accès: Septembre 2004.

BÉNARD, J.-L., BOSSAVIT, L., MÉDINA, R. & WILLIAMS, D. (2002). *Gestion de projet eXtreme Programming* (1ére éd.). Eyrolles. 298p.

BENNETT, S., MCROBB, S. & FARMER, R. (1999). *Object-oriented Systems Analysis and Design Using UML*. McGraw-Hill Publishing Company. 516p.

BÉZIVIN, J., DUPÉ, G., JOUAULT, F., PITETTE, G. & ROUGUI, J. E. (2003). *First experiments with the ATL model transformation language: Transforming XSLT into XQuery*. 2nd OOPSLA Workshop on Generative Techniques in the context of MDA, Anaheim (California), pp. 18.

BEZIVIN, J., BLAY-FORNARINO, M., BOUZEGHOUB, M., ESTUBLIER, J. & FAVRE, J.-M. (2005). *Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles - Rapport de Synthèse*. CNRS. 14p.

BILLEN, R., PANTAZIS, D. & CORNÉLIS, B. (2004). *Application of ER and CONGOO formalisms in a spatial database-reengineering project*. Department of Geography & Geomatics, University of Glasgow. 15p.

BLANC, X. (2001). *Échanges de Spécifications Hétérogènes et Réparties*. 145p.

BLANC, X. (2005). *MDA en action - Ingénierie logicielle guidée par les modèles*. Eyrolles. 270p.

BLANC, X. & DESFRAY, P. (2002). *Génie Logiciel - MDA (4) - Du profil UML au composant MDA*, http://www.devreference.net/devrefv222.pdf, Dernier accès: Septembre 2004.

BLAY-FORNARINO, M. & FRANCHI, P. (2005). *Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles - Espace Technologique "Langages" et IDM*. CNRS. 23p.

BOEHM, B. & BOSE, P. (1994). *A Collaborative Spiral Software Process Model Based on Theory W*. 3rd International Conference on Software Process, Reston, USA, pp. 18.

BOEHM, B. W. (1988, May). A Spiral Model of Software Development and Enhancement. *IEEE Computer, 21*, pp. 61-72.

BOEHM, B. W. & ROSS, R. (1989). Theory-W Software Project Management: Principles and Examples. *IEEE Transactions on Software Engineering, Vol. 15, n°7*, pp. 902-916.

BOEHM, B. W. & BOSE, P. (1994). *A Collaborative Spiral Software Process Model Based on Theoy W*. Third International Conference on the Software Process, pp. 19.

BOEHM, B. W. & TURNER, R. (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional.

BOEHM, B. W., ALEXANDER, E., KWAN, J., PORT, D., SHAH, A. & MADACHY, R. (1998). Using the WinWin Spiral Model: A Case Study. *IEEE Computer*, pp. 33-44.

BOOCH, G., RUMBAUGH, J. & JACOBSON, I. (2000). *Guide de l'utilisateur UML* (2° éd. - Trad.: Société Alinter). Eyrolles. 500p.

BORGES, K. A. V., DAVIS JR., C. A. & LAENDER, A. H. F. (2001). OMT-G: An Object-Oriented Data Model for Geographic Applications. *GeoInformatica, Vol. 5, n°3*, pp. 221-260.

BOUZEGHOUB, M. (2005). *Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles - Espace Technologique "Bases de Données"*

et l'IDM. CNRS. 11p.

BRODEUR, J. (2004). *Interopérabilité des données géospatiales : Élaboration du concept de proximité géosémantique*. Thèse de doctorat. Université Laval. Québec. 262p.

BRODEUR, J., BEDARD, Y. & PROULX, M.-J. (2002, November 10-11). *Modelling Geospatial Application Databases using UML-based Repositories Aligned with International Standards in Geomatics.* ACMGIS 2000, Washington DC (USA), pp. 113-130.

CARLI, E. (2005, 30-31 Mars). *Processus Agile dans le Domaine de la Défense*. Les Journées du Centre de Maîtrise des Systèmes et du Logiciel, Paris

CARON, C. (1991). *Nouveau formalisme de modélisation conceptuelle adapté aux SIRS*. Thèse. Université Laval. 247p.

CARREZ, C. (2003). *Projet RNTL - Assemblage de Composants par Contrats en environnement Ouvert et Réparti*, http://www.infres.enst.fr/projets/accord/2004.

CASEMAKER INC. (2000). *What is Rapid Application Development?*, http://www.casemaker.com/download/products/totem/rad_wp.pdf, Dernier accès: July 2005.

CHAMPOUX, P. (1997). *Étude des tâches et de l'utilité d'un outil à la phase "d'analyse" dans un projet de géomatisation d'entreprise*. Thèse de doctorat. Université Laval. Sainte-Foy. 150p.

CHEN, P. P.-S. (1976). The entity-relationship model-toward a unified view of data. *ACM Transactions on Database Systems, Vol. 1, n°1*, pp. 9-36.

CLAIS, S. (2003). *Étude Comparative des Systèmes de Gestion de Bases de Données Spatiales* (Stage). Université Montpellier II - IUP Génie Mathématique et Informatique. 65p.

CLARAMUNT, C., COULONDRE, S. & LIBOUREL, T. (1997). Autour des méthodes orientées objet pour la conception des SIG. *Revue internationale de géomatique, Vol. 7, n°3-4*, pp. 233-257.

CMSL. (2005, 30-31 Mars). *Séminaire Méthodes Agiles.* Les Journées du Centre de Maîtrise des Systèmes et du Logiciel, Paris

CNRS. (2005). *Stocker de l'information dans des matériaux moléculaires par voie optique*, http://www.techno-science.net/?onglet=news&news=1404, Dernier accès: Juillet 2005.

COMMISSION EUROPEENNE. (Site Web non daté, Dernière mise à jour: 25 Août 2006). *GALILEO - Système européen de navigation par satellite*, http://ec.europa.eu/dgs/energy_transport/galileo/index_fr.htm.

CONNELL, J. L. & SHAFFER, L. B. (1989). *Structured Rapid Prototyping, An Evolutionnary Approach to Software Development*. Yourdon Press. 299p.

CRÉGUT, X., EBERSOLD, S., NASSAR, M. & COULETTE, B. (2005). *Un patron de génération de code pour le profil VUML*. Objets, Composants et Modèles 2005, Berne, Suisse, pp. 5-11.

CROS, T. (2001). *La conception dans l'Extreme Programming*, http://www.devreference.net/devrefv201.pdf, Dernier accès: Septembre 2004.

DESFRAY, P. (1994). *Object Engineering - The Fourth Dimension*. Paris. Addison Wesley. 342p.

DESFRAY, P. (2001). *MDA - When a major software industry trend meets our toolset, implemented since 1994*, http://www.objecteering.com/pdf/whitepapers/us/mda.pdf, http://www.omg.org/mda/mda_files/MDA-Softeam-WhitePaper.pdf, Dernier accès: April 2005.

D'SOUZA, D. F. & WILLS, A. C. (1998). *Objects, Components, Frameworks with UML: The Catalysis Approach*. Addison-Wesley. 785p.

EGYED, A. & BOEHM, B. (1998). *Telecooperation Experience with the WinWin System.* 15th IFIP World Computer Congress (WCC), Vienna, Austria and Budapest, Hungary, pp. 37-46.

EL METHNI, M. (Non daté). *Mathématiques Appliquées aux Sciences Humaines et Sociales*, http://brassens.upmf-grenoble.fr/IMSS/MathSHS/MASS1/Alg1/Cours/Tab1.htm, Dernier accès: Mars 2006.

EUROPEAN COMMISSION. (Non daté). *MODELling solution for softWARE systems*, http://www.modelware-ist.org/, http://www.modelware-ist.org/index.php?option=com_content&task=view&id=19&Itemid=36, Dernier accès: September 2006.

FARCET, N. (2003). *MDA assessment and deployment within THALES*, http://aristote1.aristote.asso.fr/Presentations/CEA-EDF-2003/Conferences/NicolasFarcet/MIRROR-EDF-CEA-summer-school-2003-publication.ppt, Dernier accès: August 2006.

FAVRE, J.-M. & ESTUBLIER, J. (2005). *Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles - Concepts et relations de base pour l'Ingénierie par les Modèles*. CNRS. 9p.

FAYET, E. (2002). *Forum Utilisateurs Rational - Le discours de la méthode*, http://www.devreference.net/devrefv220.pdf, Dernier accès: Septembre 2004.

FILHO, J. L. & IOCHPE, C. (1999a). *Specifying analysis patterns for geographic databases on the basis of a conceptual framework*. Dans *Proceedings of the 7th ACM international symposium on Advances in geographic information systems* (pp. 7-13). Kansas City, Missouri, United States. ACM Press.

FILHO, J. L. & IOCHPE, C. (1999b). Um Estudo sobre Modelos Conceituais de Dados para Projeto de Bancos de Dados Geográficos. *Revista IP-Informática Pública, Vol. 1, n°2*, pp. 37-90.

FILHO, J. L., SODRÉ, V. D. F., DALTIO, J., RODRIGUES JÚNIOR, M. F. & VILELA, V. M. (2004). *A CASE Tool for Geographic Database Design Supporting Analysis Patterns* (3289 éd.). 43-54p.

FONDATION WIKIMEDIA. (2001, Dernière mise à jour: Avril 2005). *Wikipédia - Encyclopédie Libre*, http://fr.wikipedia.org/wiki/Accueil, Dernier accès: Avril 2005.

FONDATION WIKIMEDIA. (2005, Dernière mise à jour: 12 August 2005). *Rational Unified Process*, http://en.wikipedia.org/wiki/Rational_Unified_Process, Dernier accès: August 2005.

FOWLER, M. (2002). *Refactoring: Improving the Design of Existing Code* (9th éd.). Addison-Wesley. 431p.

FOWLER, M. (2004a). *Refactoring Home Page*, http://www.refactoring.com/.

FOWLER, M. (2004b). *UML 2.0* (3° éd.). CampusPress. 201p.

FOWLER, M. & HIGHSMITH, J. (2001a, Dernière mise à jour: August 2001). *The Agile Manifesto*, http://www.sdmagazine.com/.

FOWLER, M. & SCOTT, K. (2001b). *UML, Le Tout en Poche* (2° éd.). CampusPress. 226p.

FOWLER, M., BECK, K., BRANT, J., OPDYKE, W. & ROBERTS, D. (2002). *Refactoring: Improving the Design of Existing Code* (2° éd.). Boston. Addison-Wesley. 443p.

GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. (1999). *Design patterns - Catalogue de modèles de conceptions réutilisables* (- Trad.: J.-M. Lasvergères). Vuibert. 480p.

GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. (2001). *Design patterns - Elements of Reusable Object-Oriented Software* (1st éd.). Addison Wesley Professional. 416p.

GAYTE, O., LIBOUREL, T., CHEYLAN, J.-P. & LARDON, S. (1997). *Conception des Systèmes d'information sur environnement*. Paris. Hermés. 153p.

GERARD, S., TERRIER, F., DUBOIS, H., MRAIDHA, C. & BAUDRY, B. (2005). *Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles - Les systèmes temps-réel et l'IDM*. CNRS. 20p.

GIRE, S. (2004, Dernière mise à jour: Septembre 2004). *Compilation - Théorie des langages*, http://fastnet.univ-brest.fr/~gire/LIBRE_COURS/COMPIL/cours.pdf, Dernier accès: Mai 2005.

GONNOT, B. (2005, 30-31 Mars). *L'évolution d'XP à AGF-AM - L'expérience Positive*. Les Journées du Centre de Maîtrise des Systèmes et du Logiciel, Paris

GORDILLO, S., BALAGUER, F. & DAS NEVES, F. (1997). *Generating the Architecture of GIS Applications with Design Patterns*. GIS 97, Las Vegas (USA), pp. 30-34.

GORDILLO, S., BALAGUER, F., MOSTACCIO, C. & DAS NEVES, F. (1999). Developing GIS Applications with Objects: A Design Patterns Approach. *GeoInformatica, Vol. 3, n°1*, pp. 7-32.

GRØNMO, R. & OLDEVIK, J. (2005, 23-25 February). *An Empirical Study of the UML Model Transformation Tool (UMT)*. INTEROP-ESA - First International Conference on Interoperability of Enterprise Software and Applications, Geneva (Switzerland)

GUIMOND, L.-E. (2005). *Conception d'un environnement de découverte des besoins pour le développement de solutions SOLAP*. Thèse. Université Laval. Québec. 124p.

HUCHARD, M. (2003). *Classification de classes dans les approches à objets - Algorithmes et applications*. Mémoire pour une habilitation à diriger des recherches. Université Montpellier II. Montpellier.

IOVLEFF, S. (2004). *Mathématiques Pour l'Informatique I : Théorie des Ensembles et Relations*, http://www.iut-info.univ-lille1.fr/~iovleff/pub/Teaching/MathInfo1/Annee04-05/Polys/Poly.pdf, Dernier accès: Mars 2006.

ISO. (1999). *ISO/IEC TR 15504-5 - Technologies de l'information - Évaluation des procédés - Partie 5 : Un modèle d'évaluation et guide des indicateurs*.

ISO. (2002). *ISO 19108 - Geographic Information - Temporal schema*.

ISO. (2003a). *ISO 19107 - Geographic Information - Spatial schema*.

ISO. (2003b). *ISO 19111 - Geographic Information - Spatial referencing by coordinates*.

ISO. (2003c). *ISO/IEC 15504-2 - Technologies de l'information - Évaluation des procédés - Partie 2 : Exécution d'une évaluation*.

ISO. (2004a). *ISO/IEC 15504-1 - Technologies de l'information - Évaluation des procédés - Partie 1 : Concepts et vocabulaire*.

ISO. (2004b). *ISO/IEC 15504-3 - Technologies de l'information - Évaluation des procédés - Partie 3 : Réalisation d'une évaluation*.

ISO. (2004c). *ISO/IEC 15504-4 - Technologies de l'information - Évaluation des procédés - Partie 4 : Conseils sur l'utilisation pour l'amélioration de processus et la détermination de capacité de processus*.

ISO. (2005). *ISO 19109 - Geographic Information - Rules for application schema*.

ISO. (Target year: 2007). *ISO/CD 19141 - Geographic information -- Schema for moving features*, http://www.iso.org/iso/en/CatalogueListPage.CatalogueList?COMMID=4637&scopelist=PROGRAMME, Dernier accès: August 2006.

JACOBSON, I. (2003). *Use Cases -- Yesterday, Today, and Tomorrow*, http://www.ivarjacobson.com/html/content/publications_papers.html, http://www.ivarjacobson.com/publications/uc/UseCases_TheRationalEdge_Mar2003.pdf, Dernier accès: August 2005.

JACOBSON, I., BOOCH, G. & RUMBAUGH, J. (1999). *The Unified Software Development Process.* Addison Wesley. 463p.

JEFFRIES, R. (2001, Dernière mise à jour: August). *What is Extreme Programming?*, http://www.xprogramming.com/xpmag/whatisxp.htm, Dernier accès: July 2005.

JEFFRIES, R. (2004, Dernière mise à jour: June). *Chrysler Payroll*, http://c2.com/cgi/wiki?RonJeffries2005.

JEZEQUEL, J.-M., GERARD, S., MRAIDHA, C. & BAUDRY, B. (2005). *Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles - Approche unificatrice par les modèles.* CNRS. 9p.

JOUAULT, F. & KURTEV, I. (2006, April, 23-27). *On the Architectural Alignment of ATL and QVT.* SAC'06, Dijon (France), pp. 8.

KLEPPE, A. (2004). *Interview with Anneke Kleppe*, http://www.codegeneration.net/tiki-read_article.php?articleId=21, Dernier accès: August 2006.

KLEPPE, A., WARMER, J. & BAST, W. (2003). *MDA Explained: The Model Driven Architecture-Practice and Promise*. Addison-Wesley Professional. 170p.

KÖSTERS, G., PAGEL, B. & SIX, H. (1997). GIS-application development with GeoOOA. *International journal of GIS, Vol. 11, n°4*, pp. 307-355.

KOSUTH, P. (2005). Présentation synthétique de l'UMR TETIS. pp. 1.

KRUCHTEN, P. B. (1999). *The Rational Unified Process: An introduction* (3rd Edition éd.). Addison-Wesley Professional. 336p.

KRUCHTEN, P. B. (2000). *Introduction au Rational Unified Process*. Éditions Eyrolles. 282p.

KRUCHTEN, P. B. (Site Web non daté). http://philippe.kruchten.com/, Dernier accès: August 2005.

LARMAN, C. (2002a). *UML et les Design Patterns* (2° éd. - Trad.: M.-C. Baland & D. Maisons). CampusPress. 672p.

LARMAN, C. (2002b). *Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and the Unified Process* (2° éd.). Prentice Hall PTR. 627p.

LARRIVEE, S., BEDARD, Y. & POULIOT, J. (2005). *How to Enrich the Semantics of Geospatial Databases by Properly Expressing 3D Objects in a Conceptual Schema.* OTM Workshops 2005, pp. 999 – 1008.

LARRIVEE, S., BEDARD, Y. & POULIOT, J. (2006). Fondement de la modélisation conceptuelle des bases de données géospatiales 3D. *Revue internationale de géomatique, Vol. 16, n°1*, pp. 9-27.

LAURINI, R. & THOMPSON, D. (1992). *Fundamentals of spatial information systems.* San Diego. Academic Press Inc. 680p.

LBATH, A. (1997). *AIGLE : Un environnement pour la conception et la génération automatique d'applications géomatiques*. Thèse de doctorat. Institut National des Sciences Appliquées. Lyon.

LBATH, A. & PINET, F. (2000, 10-11 November). *The Development and Customization of GIS-based Applications and Web-based GIS Applications with the CASE tool AIGLE.* 8th ACM Symposium on Advances in Geographic Information Systems, Washington D.C, pp. 194-196.

LEGRAIN, M., GARNIER, Y. & VINCIGUERRA, M. (2002). *Le petit Larousse grand format 2003 en couleurs*. Larousse. 1885p.

MACEDO DE AMORIM, K. (2004). *Modélisation d'aspects qualité de service en UML : application aux composants logiciels*. Thèse. Université de Rennes I. Rennes. http://www.irisa.fr/triskell/publis/2004/Macedo04a.pdf. 169p.

MARTIN, J. (1991). *Rapid Application Development*. Macmillan Publishing Co., Inc. 788p.

MARVIE, R., DUCHIEN, L. & BLAY-FORNARINO, M. (2005). *Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles - Plate-formes d'exécution et l'IDM*. CNRS. 13p.

MCCONNELL, S. C. (1998). *Requirements Development. Software Project Survival Guide: How to Be Sure Your First Important Project Isn't Your Last.* Dans (pp. 288). Redmond (Washington State). Microsoft Press.

MCCRACKEN, D. D. & JACKSON, M. (1982). Life Cycle Concept Considered Harmful. *ACM Software Engineeting Notes*, pp. 29-32.

MEDINA, R. (2005, 30-31 Mars). *Retour d'expérience XP dans un contexte industriel.* Les Journées du Centre de Maîtrise des Systèmes et du Logiciel, Paris

MILLAN, T., PERCEBOIS, C., LEBLANC, H. & BAZEX, P. (2004). Projet NEPTUNE : Vérification statique des modèles. *Génie Logiciel, n°69*, pp. 45-50.

MILLER, J. & MUKERJI, J. (2001). *Model Driven Architecture (MDA)*, http://www.omg.org/cgi-bin/apps/doc?ormsc/01-07-01.pdf, Dernier accès: September 2004.

MILLER, J. & MUKERJI, J. (2003). *MDA Guide Version 1.0.1*, http://www.omg.org/cgi-bin/doc?omg/03-06-01, Dernier accès: May 2006.

MULLER, P.-A. & GAERTNER, N. (2000). *Modélisation objet avec UML* (2° éd.). Eyrolles. 520p.

MURMUR. (2000). *Multi-representations and multiple resolutions in geographic databases*, http://lbdwww.epfl.ch/e/MurMur/, Dernier accès: Janvier 2005.

MYSQL. (2006). *Manuel de référence MySQL 5.0*, http://dev.mysql.com/doc/, http://downloads.mysql.com/docs/refman-5.0-fr.a4.pdf2006.

NORMAND, P. (1999). *Modélisation des contraintes d'intégrité spatiale : théorie et exemples d'application*. Thèse. Université Laval. Québec. 95p.

OGC. (2005). *Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture*, Dernier accès: January 2006.

OMG. (1999, Dernière mise à jour: June). *Unified Modeling Language - Specification - Version 1.3*.

OMG. (2001a, Dernière mise à jour: September). *Unified Modeling Language - Specification - Version 1.4*, http://www.omg.org/cgi-bin/apps/doc?formal/01-09-67.pdf, Dernier accès: Junuary 2003.

OMG. (2001b). *Common Warehouse Metamodel - Version 1.0*, http://www.omg.org/cgi-bin/doc?ad/2001-02-01, Dernier accès: June 2004.

OMG. (2002a, Dernière mise à jour: April). *Meta Object Facility Specification - Version 1.4*, http://www.omg.org/cgi-bin/doc?formal/2002-04-03.pdf.

OMG. (2002b). *MOF 2.0 Query/Views/Transformations ad/2002-04-10*, http://www.omg.org/docs/ad/02-04-10.pdf, Dernier accès: January 2005.

OMG. (2003a, Dernière mise à jour: March). *Unified Modeling Language - Specification - Version 1.5*, http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.pdf.

OMG. (2003b, Dernière mise à jour: November). *UML 2.0 Infrastructure Specification*, http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-15.pdf.

OMG. (2005, Dernière mise à jour: August). *Unified Modeling Language: Superstructure - Version 2.0*, http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf, Dernier accès: June 2006.

OMG. (2006, Dernière mise à jour: March). *Unified Modeling Language: Infrastructure - Version 2.0*, http://www.omg.org/cgi-bin/apps/doc?formal/05-07-05.pdf, Dernier accès: June 2006.

PANTAZIS, D. & DONNAY, J. P. (1996). *La conception SIG : méthode et formalisme*. Paris. Hermés. 343p.

PANTAZIS, D. & DONNAY, J.-P. (1997). Objets géographiques à limites indéterminés - Modélisation et intégration dans un modèle conceptuel de données. *Revue internationale de géomatique, Vol. 7, n°2*, pp. 159-186.

PAQUETTE, F. (1990). *Conception d'un système d'information à référence spatiale pour la gestion et l'aménagement de la Forêt Montmorency*. Mémoire de maîtrise. Université Laval. Sainte-Foy. 126p.

PARENT, C., SPACCAPIETRA, S. & ZIMANYI, E. (2006). *Conceptual Modeling for traditional and Spatio-Temporal Applications: The MADS Approach*. Springer-Verlag New York, Inc.

PARENT, C., SPACCAPIETRA, S., ZIMÁNYI, E., DONINI, P., PLAZANET, C. & VANGENOT, C. (1998, July). *Modeling Spatial Data in the MADS Conceptual Model*. 8th Int. Symp. on Spatial Data Handling, Vancouver (Canada), pp. 12.

PARENT, C., SPACCAPIETRA, S., ZIMÁNYI, E., DONINI, P., PLAZANET, C., VANGENOT, C., ROGNON, N. & CRAUSAZ, P.-A. (1997a). MADS, modèle conceptuel spatio-temporel. *Revue Internationale de Géomatique, Vol. 7, n°3-4*, pp. 317-352.

PARENT, C., SPACCAPIETRA, S., ZIMANYI, E., DONINI, P., PLAZANET, C., VANGENOT, C., ROGNON, N., J, P. & CRAUSAZ, P. (1997b). MADS: un modèle conceptuel pour des applications spatio-temporelles. *Revue Internationale de Géomatique, Vol. 7, n°3-4*, pp. 317 - 352.

PICHAT, M. & SOLNON, C. (1997). *Théorie des Langages*, http://bat710.univ-lyon1.fr/~csolnon/langages.html, Dernier accès: Mai 2005.

POSTGIS. (2005). *PostGis Web Site*, http://postgis.refractions.net/, Dernier accès: September 2006.

PRICE, R., TRYFONA, N. & JENSEN, C. S. (1999). *A Conceptual Modeling Language for Spatiotemporal Applications* (Chorochronos Technical Report CH-99-20). 21p.

PRINTZ, J. (2005, 30-31 Mars). *Les Méthodes Agiles : Évolution, Révolution, Mode, Bluff, ...?!!! - Séminaire Méthodes Agiles.* Les Journées du Centre de Maîtrise des Systèmes et du Logiciel, Paris

PROULX, M.-J., LARRIVEE, S. & BEDARD, Y. (2002). *Représentation multiple et généralisation avec UML et l'outil Perceptory*. Dans *Généralisation et représentation multiple* (pp. 113-130). Paris. Hermès Sciences Publications.

QVT PROJECT. (Non Daté). *Welcome to the QVT Project web page*, http://modfact.lip6.fr/qvtP.html.

RAD-TOOL. (2004). *RAD-TOOL*, http://www.rad-tool.com/, Dernier accès: June 2005.

REGION WALLONNE. (2004). *Le prototypage : Définition et objectifs*, Dernier accès: Septembre 2006.

RENOLEN, A. (1997, 1-3 June). *Conceptual Modelling and Spatiotemporal Information Systems: How to Model the Real World.* ScanGIS'97 - The 6th Scandinavian Research Conference on GIS, Stockholm (Sweden), pp. 22.

RICCOBENE, E., ROSTI, A. & SCANDURRA, P. (2004). *Improving SoC Design Flow by Means of MDA and UML Profiles.* 3rd Workshop in Software Model Engineering, Lisbon, Portugal, pp. 8.

ROBERT, P. (1984). *Le Petit Robert - Dictionnaire alphabétique et analogique de la langue française*. Les Dictionnaires Le Robert. 2172p.

ROCHA, L. V. D. (2001). *Framework Conceitual Temporal para Aplicações de Sistemas de Informações Geográficas*. Universidade Federal do Rio Grande Do Sul. Porto Alegre. http://www.inf.ufrgs.br/~nina/Dissertacao/LucianaRocha.pdf#search=%22geoframe-T%3A%20um%22. 137p.

ROCHA, L. V. D., EDELWEISS, N. & IOCHPE, C. (2001). *GeoFrame-T: A Temporal Conceptual Framework for Data modeling.* 9th ACM international symposium on Advances in geographic information systems, Atlanta, Georgia (USA), pp. 124-129.

ROQUES, P. (2002). *La démarche agile au service du e-business: Première partie*, http://www.dotnetguru.org/articles/Methodes/AgileDotNet.htm.

ROQUES, P. & VALLEE, F. (2002). *UML en Action - De l'analyse des besoins à la conception en Java* (2° éd.). Eyrolles. 388p.

ROUME, C. (2004). *Analyse et restructuration de hiérarchies de classes*. Thèse. Université Montpellier II. Montpellier. 211p.

ROYCE, W. W. (1970, August). *Managing The Development of Large Software Systems.* IEEE Westcon, Monterey, California, United States, pp. 1-9.

RUMBAUGH, J., BLAHA, M., EDDY, F., PREMERLANI, W. & LORENSEN, W. (1995). *OMT - Médélisation et conception orientées objet* (- Trad.: A.-B. Fontaine, G.-P. Reich & V. Zaïm). Masson - Prentice-Hall. 536p.

SAINT MARTIN, P. (2002a). *UML - Le retour en grâce (1)*, http://www.devreference.net/devrefv220.pdf, Dernier accès: Septembre 2004.

SAINT MARTIN, P. (2002b). *UML - Le retour en grâce (2)*, http://www.devreference.net/devrefv221.pdf, Dernier accès: Septembre 2004.

SAVARD, G. (Non daté). *Cours de Logique et mathématiques discrètes*, http://www.seg.etsmtl.ca/GSavard/mat210/index.html, Dernier accès: Mars 2006.

SCHRAMM, W. (1954). How Communication Works. The Process and Effects of Mass Communication. Urbana (Illinois): University of Illinois Press.

SCIENCES.CH. (2005). *Éléments de Mathématiques Appliquées*, http://www.sciences.ch/dwnldbl/telecharger.php3, Dernier accès: Mars 2006.

SHEKHAR, S., COYLE, M., GOYAL, B., LIU, D.-R. & SARKAR, S. (1997). *Data Models in Geographic Information Systems*, pp. 103--111.

SHU, H., SPACCAPIETRA, S., PARENT, C. & SEDAS, D. Q. (2003, 19-20 March). *Uncertainty of Geographic Information and its Support in MADS.* 2nd International Symposium on Spatial Data Quality, Hong Kong (China)

SIGONNEAU, B. (2003). *Concept de vue et génie logiciel*. DEA. http://www.irisa.fr/lande/sigonneau/publications/articles/biblio_dea.pdf. 14p.

SILAGHI, R., FONDEMENT, F. & STROHMEIER, A. (2004, June). *"Weaving" MTL Model Transformations.* 2nd International Workshop on Model Driven Architecture, Foundations and Applications, Linköping University (Sweden), pp. 15.

SODRÉ, V. D. F., FILHO, J. L., VILELA, V. M. & ANDRADE, M. V. A. (2005). *Improving productivity and quality of GIS databases design using an analysis pattern catalog.* Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling, Newcastle, New South Wales (Australia), pp. 107-114.

SOFTEAM. (2003a). *Objecteering/UML - Objecteering/J Libraries User guide - Objecteering/Metamodel User guide - Version 5.2.2*. 491p.

SOFTEAM. (2003b). *Objecteering/UML - Objecteering/SQL Designer User guide - Version 5.2.2*. 236p.

SOFTEAM. (2005). Formation sur les Modèles Objet et UML [Transparents].

SRIPLAKICH, P. (2003). *Techniques des transformations de modèles basées sur la méta-modélisation*. Mémoire du D.E.A. Systèmes Informatiques Répartis. Université Pierre et Marie Curie. http://modfact.lip6.fr/ModFactWeb/qvt/SimpleTRL.pdf. 59p.

TAFT, D. K. (2004, Dernière mise à jour: March). *What Is Bill Gates Thinking?*, http://www.eweek.com/article2/0,1895,1556075,00.asp, http://www.eweek.com/article2/0,1895,1556081,00.asp, Dernier accès: Juily 2005.

THE MIDDLEWARE COMPANY. (2003). *Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach - Productivity Analysis* (Case Study). 18p.

THE STANDISH GROUP. (2001). *Extreme CHAOS*. The Standish Group International. 12p.

TRAN, P. (2005). *Les patterns, oui. MDA, non*, http://www.weblmi.com/sections/actualites/developpeur_referenc/les_patterns_oui._m/, Dernier accès: June 2005.

TRYFONA, N. & JENSEN, C. S. (1998a). *CHOROCHRONOS: A Network for Spatiotemporal Database Systems - Conceptual Data Modeling for Spatiotemporal Applications* (Technical report n°. CH-98-08). European Commission DG XII Science, Research and Development. 25p.

TRYFONA, N. & JENSEN, C. S. (1998b). *CHOROCHRONOS: A Network for Spatiotemporal Database Systems - A Component-based Conceptual Model for Spatiotemporal Design* (Technical report n°. CH-98-10). European Commission DG XII Science, Research and Development. 21p.

TRYFONA, N. & JENSEN, C. S. (2000, 19-21 March). *Using Abstractions for Spatio-Temporal Conceptual Modeling*. 2000 ACM Symposium on Applied Computing, Villa Olmo (Italy), pp. 313-322.

TRYFONA, N., PFOSER, D. & HADZILACOS, T. (1997). *Modeling Behavior of geographic objects : an experience with the object modeling technique*. CASE'97, Barcelone

TRYFONA, N., ANDERSEN, S., MOGENSEN, S. R. & JENSEN, C. S. (1999, 26-29 August). *A Methodology and a Tool for Spatiotemporal Database Design.* Seventh Hellenic Conference on Informatics, Ioannina (Greece), pp. 53-60.

TURBAN, E. & ARONSON, J. E. (2001). *Decision Support Systems and Intelligent Systems* (6° éd.). Prentice-Hall. 867p.

UNIVERSITY OF CALIFORNIA. (1997a). *Rapid Application Development (RAD)*, http://sysdev.ucdavis.edu/WEBADM/document/rad_toc.htm, Dernier accès: June 2005.

UNIVERSITY OF CALIFORNIA. (1997b). *Rapid Application Development (RAD) - Essential Aspects*, http://sysdev.ucdavis.edu/WEBADM/document/rad-essentials.htm, Dernier accès: Juin 2005.

VANGENOT, C. (2001). *Multi-représentation dans les bases de données géographiques*. Thèse de docteur es sciences. Université de Bourgogne. http://biblion.epfl.ch/EPFL/theses/2001/2430/2430_abs.pdf#search=%22Multi-repr%C3%A9sentation%20dans%20les%20bases%20de%20donn%C3%A9es%20g%C3%A9ographiques%22.

VANGENOT, C., PARENT, C. & SPACCAPIETRA, S. (2002). *Modélisation et manipulation de données spatiales avec multireprésentation dans le modèle MADS*. Dans *Généralisation et représentation multiple* (pp. 93-112). Hermès.

VICKOFF, J.-P. (1995). *RAD, le Développement d'application Client-Serveur*.

VICKOFF, J.-P. (1999, Août). RAD 2 : la conduite de projet "haute performance". *Forum Logiciel*, pp. 15.

VICKOFF, J.-P. (2000). *Méthode RAD - Éléments fondamentaux*, http://mapage.noos.fr/rad/radmetho.pdf, 32p.

VICKOFF, J.-P. (Site Web non daté). *Portail des DSI et chefs de projets SI & NTIC*, http://www.rad.fr/, Dernier accès: Juin 2005.

VOSS, A. (1998). *Dictionnaire de l'Informatique et de l'Internet* (Édition Septembre 98 éd. - Trad.: C. Stoll, H. Bertrand & F. Ligier). Micro Application.

VOYER, P. (2000). *Tableau de bord de Gestion et indicateurs de performance* (2° éd.). Sainte-Foy. Presses de l'Université du Québec. 446p.

WEB NTIC. (2003, Dernière mise à jour: Mars). *Dichotomie MOE MOA*, http://phortail.org/webntic/Dichotomie-MOE-MOA-28.html, Dernier accès: Juillet 2005.

YVON, F. & DEMAILLE, A. (2003). *Théorie des langages - Notes de cours*, http://www.lrde.epita.fr/~akim/THL/th-langage.pdf, Dernier accès: Mai 2005.

ZARBA, C. G. (2005). *Lecture Notes on Theoretical Foundations of Computer Science - Naive Set Theory*, http://theory.stanford.edu/~zarba/notes.html, http://theory.stanford.edu/~zarba/rainbow/ch01.pdf, Dernier accès: March 2006.

ZIMÁNYI, E., PARENT, C., SPACCAPIETRA, S., EL OSTA, E., BURNET, R. & BALLEY, S. (2002). *MurMur Project - Multi Representations Multi Resolutions - Workpackage 7 - Coordination - Deliverable 23 - Final Report* (n°. MM-WP7-DLA-023). European Community. 36p.

Glossaire

I CONCEPT

Les définitions ci-après extraites de dictionnaires ou de l'encyclopédie en ligne Wikipédia font toutes références à l'aspect abstrait ou immatériel de la notion de concept :

Définition du Petit Larousse *(Legrain et al., 2002) : Représentation générale et abstraite d'un objet, d'un ensemble d'objets. […]*

Définition du Petit Robert *(Robert, 1984) : Représentation mentale générale et abstraite d'un objet.*

Définition du site Wikipédia *(Fondation Wikimédia, 2001) : On nomme concept une idée abstraite et générale. La notion de corpulence (rapport de la taille et du poids) constitue par exemple un concept.*

La définition retenue est :

Définition 74 Un ***concept*** est le *signifié* d'un objet, d'une idée, d'une pensée, d'une abstraction, etc.

II CONCEPT THEMATIQUE

Définition 75 Un ***concept thématique*** est un concept du domaine à modéliser.

III SPECIFICATION

Définition Grady Booch 1 *(Booch et al., 2000) (en implémentation) : Description textuelle de la syntaxe et de la sémantique des briques de base particulières.*

Définition Grady Booch 2 *(Booch et al., 2000) (en réalisation) : Description déclarative de ce qu'est ou fait une chose.*

Définition OMG *(OMG, 2003a) : A declarative description of what something is or does. Contrast: implementation.*

Définition Pierre-Alain Muller *(Muller et al., 2000) : Description exhaustive d'un élément de modélisation.*

Les quatre définitions ci-dessus sont assez proches toutefois nous préférons adopter une définition plus générale impliquant le vocabulaire préalablement défini :

Définition 76 Une ***spécification*** est la description exhaustive d'un concept.

IV SPECIFICATION METIER

Définition 77 Une ***spécification métier*** est la spécification d'un concept thématique.

V SPECIFICATION TECHNIQUE

Définition 78 Une ***spécification technique*** est la spécification d'un concept informatique.

VI ARTEFACT

Définition Grady Booch *(Booch et al., 2000) : Information utilisée ou produite par un processus de développement logiciel.*

Définition Ivar Jacobson *(Jacobson et al., 1999) : A tangible piece of information that (1), is created, changed, and used by workers when performing activities, (2) represents an area of responsibility, and (3) is likely to be put under separate version control. An artifact can be a model, a model element, or a document.*

Définition Pierre-Alain Muller *(Muller et al., 2000) : Élément d'information, produit ou utilisé lors d'une activité de développement.*

Définition OMG MOF 1 *(OMG, 2002b) : A piece of information that is used or produced by a software development process. An artifact can be a model, a description, or software. Synonym: product.*

Définition OMG UML 2 *(OMG, 2003a) : A physical piece of information that is used or produced by a software development process. Examples of Artifacts include models, source files, scripts, and binary executable files. An artifact may constitute the implementation of a deployable component. Synonym: product Contrast: component.*

Définition OMG UML 3 *(OMG, 2005) : An artifact is the specification of a physical piece of information that is used or produced by a software development process, or by deployment and operation of a system. Examples of artifacts include model files, source files, scripts, and binary executable files, a table in a database system, a development deliverable, or a word-processing document, a mail message.*

Les définitions ci-dessus diffèrent d'un auteur à l'autre et montrent l'évolution du concept *Artefact* au sein même de l'OMG. Bien que leurs formulations différent, le sens de ces définitions converge vers l'idée que :

Définition 79 Un ***artefact*** est une information ou un fragment d'information *utilisé* ou *produit* au cours du développement d'une application.

Dans notre contexte de recherche, il est important de souligner que les modèles sont des artefacts créés au début du développement et qu'ils évoluent constamment au cours du processus de développement.

VII PATRON DE CONCEPTION

Les deux définitions les plus couramment citées dans la littérature informatique sont celles (Alexander *et al.*, 1977) et de (Gamma *et al.*, 1999) :

Définition de Christopher Alexander et al. *(Alexander et al., 1977) : Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the*

solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

Définition[91] ***d'Éric Gamma et al.*** *(Gamma et al., 1999) : Systématiquement, un modèle de conception, nomme, commente et explicite une démarche générale dont relèvent des problèmes de conception observables fréquemment dans les systèmes orientés objet. Le modèle décrit le problème, la solution, et dit quand appliquer la solution, et quelles sont ses conséquences. Il fournit également des conseils pour le développement et des exemples de codage. La solution est un ensemble organisé d'objets et de classes, utilisés pour résoudre le problème. La solution est particularisée et développée afin de résoudre le problème dans un contexte spécifique.*

La définition d'Éric Gamma est en accord avec celle de Christopher Alexander et complète avantageusement cette dernière sur trois points : la nécessité de préciser les conséquences induites par l'utilisation du patron de conception, l'exigence de fournir des conseils et des exemples de développements et la possibilité d'adapter la solution à son problème.

Ce dernier point est particulièrement important en termes de développement car il n'est pas rare d'être obligé de réaliser des adaptations du patron de conception pour satisfaire les impératifs du domaine ou des acteurs.

Nous nous utiliserons le concept de *Patron de conception* au sens défini par Éric Gamma :

Définition 80 Systématiquement, un modèle de conception, nomme, commente et explicite une démarche générale dont relèvent des problèmes de conception observables fréquemment dans les systèmes orientés objet. Le modèle décrit le problème, la solution, et dit quand appliquer la solution, et quelles sont ses conséquences. Il fournit également des conseils pour le développement et des exemples de codage. La solution est un ensemble organisé d'objets et de classes, utilisés pour résoudre le problème. La solution est particularisée et développée afin de résoudre le problème dans un contexte spécifique.

VIII PATRON DE CONCEPTION SIG

Les concepts de spatialité (*Point*, *Ligne* et *Polygone*) et de temporalité (*Instant* et *Période*) sont des concepts connus depuis fort longtemps qui ont des relations stables entre eux. Ils constituent des mini-modèles récurrents qui sont systématiquement mobilisés à l'occasion du développement des *Systèmes d'Information Géographique*. La figure 121 illustre deux mini-modèles entre les concepts *Point* et *Ligne*.

Même s'il existe plusieurs variantes, le nombre de mini-modèles n'est pas infini. De plus, ces variantes sont relativement bien connues comme le montrent les deux mini-modèles de la figure 121.

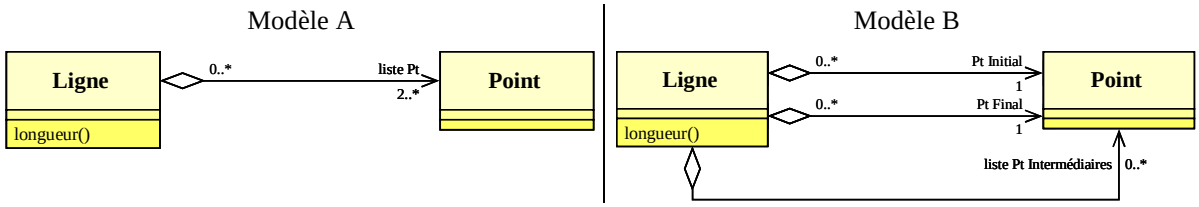


Figure 121 Exemples de mini-modèles.

Les schémas spatial et temporel proposés par les normes ISO 19107 (ISO, 2003a) et respectivement 19108 (ISO, 2002) sont aussi des (mini-)modèles qui décrivent les concepts de spatialité et de temporalité et leurs relations. À la différence de ceux réalisés par un concepteur ou de la figure 121, ces schémas sont des variantes reconnues et acceptées au niveau international.

[91] **Version originale** (Gamma *et al.*, 2001) : A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context.

De nombreux mini-modèles du domaine de l'*Information Géographique* présentant toutes les caractéristiques des patrons de conception[92], nous avons défini le concept de *Patron de conception SIG* :

Définition 81 Un ***Patron de conception SIG*** est un patron de conception ne faisant intervenir que des concepts SIG[93].

Dans le domaine de l'*Information Géographique*, cette terminologie est déjà utilisée pour désigner les mini-modèles existants entre concepts de spatialité et/ou de temporalité (Tryfona *et al.*, 1998b) ou pour décrire les mini-modèles entre entités référencées (Gordillo *et al.*, 1997 ; Gordillo *et al.*, 1999). Silvia Gordillo montre que les entités géographiques peuvent être organisées autour de patrons de conception d'Éric Gamma (patron de conception composite, décorateur, builder, etc. (Gamma *et al.*, 1999)) permettant ainsi d'exprimer par exemple la structure emboîtée des concepts d'un quartier ou de faciliter l'organisation en couche des entités géographiques en vue d'une représentation cartographique, etc.

[92] Au sens de la définition 80.
[93] Cf. définition 16 au Chapitre 7.

Acronymes

2TUP	2 Track Unified Process
ADS	Adaptive Software Development
AGL	Atelier de Génie Logiciel
C3	Chrysler Comprehensive Compensation
CMM	Capability Maturity Model
CNES	Centre National d'Etudes spatiales
CNIG	Conseil National de l'Information Géographique
COPT	Conception d'Observatoires de Pratiques Territorialisées
CWM	Common Warehouse Metamodel
D'OC	Données Objets Composants pour les systèmes complexes
DSDM	Dynamic Systems Development Method
GPS	Global Positioning System
IGN	Institut Géographique National
IHM	Interface Homme/Machine
INSA	Institut National des Sciences Appliquées
ISO	International Organization for Standardization
LIMOS	Laboratoire d'Informatique, de Modelisation et d'Optimisation des Systemes.
LIRMM	Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
MADS	Modeling of Application Data with Spatio-temporal features
MOA	Maîtrise d'ouvrage
MOE	Maîtrise d'œuvre
MOF	Meta Object Facility
OGC	Open Geospatial Consortium, Inc.
OMG	Object Management Group
OMT	Object Modeling Technique
ONF	Office National des Forêts
ONIC	Office national Interprofessionnel des Céréales
OOSE	Object-Oriented Software Engineering
PAC	Politique Agricole Commune
PVL	Plug-in for Visual Language
RAD	Rapid Application Development
ROP	Rational Objectory Process
RUP®	Rational Unified Process®
SGBD	Systèmes de Gestion de Bases de Données
SGBDR	Systèmes de Gestion de Bases de Données Relationnelles

SIG	Système d'Information Géographique
SIRS	Système d'Information à Référence Spatiale
SWAT	Skilled Workers With Advanced Tools
TC 211	Technical Comity 211 - Comité Technique 211
UM II	Université Montpellier II
UML	Unified Modeling Language
UMR 3S	Unité Mixte de Recherche Structures et Systèmes Spatiaux
UMR TETIS	Unité Mixte de Recherche Territoires, Environnement, Télédétection et Information Spatiale
UP	Unified Process - Processus Unifié
XP	eXtreme Programming

Annexe I. Rappels sur le Langage UML

Bill Gates[94] :
Modeling is the future, so every company that's working on this I think it's great (Taft, 2004)

La présente recherche a été effectuée au Cemagref, établissement dont les domaines d'intervention sont principalement l'agriculture et l'environnement. La plupart du temps, le cœur de compétence des scientifiques du Cemagref n'est pas l'informatique et la modélisation d'applications informatiques.

Cette annexe leur est destinée ainsi qu'à toute personne n'ayant pas ou ayant une connaissance limitée du langage Unified Modeling Language (UML) car, tout au long du présent mémoire, le lecteur a à lire et à décrypter des modèles UML représentant une formalisation du système étudié.

Le but de cette annexe est de présenter rapidement *l'objectif* du langage UML, son *historique*, les *concepts* permettant de modéliser la structure d'un système, le symbolisme graphique associé du langage UML et, enfin, les *diagrammes* représentant les différents points de vue sur le système. Les diagrammes constituent la force du langage UML car ils facilitent la communication entre les acteurs du domaine modélisé et le concepteur.

Voulu par ces concepteurs lors de sa création, le langage UML ne couvre ni le cycle de développement de l'application, ni la gestion financière, ni celle des ressources humaines ou des moyens matériels (Sigonneau, 2003), éléments qui sont du ressort de la méthode de conduite de projet.

Le langage UML participe donc à l'élaboration d'une application en s'intégrant à une méthode de conduite de projet plus ou moins "rigide/cartésienne" selon la sensibilité du chef de projet et l'entreprise ayant obtenue le marché. D'autres outils de gestion de projet, de gestion de personnel, etc. viennent compléter l'outil de modélisation adopté pour gérer tout le processus de développement permettant la réalisation d'une application.

Nous signalons que le langage UML n'est qu'un langage de modélisation parmi d'autres. Par contre, il semble s'imposer comme langage de modélisation car de plus en plus de communautés l'adoptent.

[94] Traduction (Blanc, 2005) : Modéliser est le futur, et je pense que les sociétés qui travaillent dans ce domaine ont raison.

Sommaire détaillé

I LE LANGAGE UML

Le langage *Unified Modeling Language* (UML) a été standardisé par l'Object Management Group (OMG), consortium international réunissant des utilisateurs, des chercheurs, des développeurs et des industriels. Il permet de décrire les systèmes avec l'objectif final de produire une application informatique. ***C'est un langage relevant des formalismes objets et qui permet de spécifier, de visualiser, de construire et de documenter les concepts d'un système (OMG, 2002a, 2003a)***. La composante visuelle dote ce langage d'une capacité de ***communication*** qui favorise l'échange entre les concepteurs de l'application et les acteurs du domaine. Les spécifications du langage UML représentent la convergence des meilleures pratiques en technologie objet de l'industrie informatique.

En outre, l'Object Management Group a formalisé et modélisé les concepts et les relations manipulés par le langage lui-même au sein d'un modèle désigné par *métamodèle d'UML* qui décrit la syntaxe du langage. Les instances du métamodèle sont les *ateliers de génie logiciel*. Ils offrent un cadre informatique mettant en œuvre le langage UML.

II HISTORIQUE D'UML

Le langage UML est né de la mise en commun des trois plus importantes méthodes de modélisation orientées objet du début des années 90 (cf. figure 122) au sein d'un même langage (OMG, 1999) :

- La méthode Booch dont l'auteur Grady Booch travaillait au développement de systèmes en Ada chez *Rational Software Corporation*.
- La méthode *Object Modeling Technique* (OMT) développée par Jim Rumbaugh qui dirigeait une équipe de recherche chez *General Electric*.
- La méthode *Object-Oriented Software Engineering* (OOSE) résultant des travaux d'Ivar Jacobson sur les commutateurs téléphoniques chez son employeur Ericsson. La méthode OOSE était la première à introduire le concept des cas d'utilisation (*use case*).

Le langage UML a bénéficié de bien d'autres travaux comme ceux de Sally Shlaer et Steve Mellor sur l'analyse et la conception de systèmes ou ceux de Jim Odell et James Martin en génie logiciel mais aussi des nombreux acquis de la communauté Smalltalk (Fowler, 2004b ; Fowler *et al.*, 2001b ; Muller *et al.*, 2000). La fusion de ces trois méthodes a été réalisée très rapidement dans les années 94-96 car Jim Rumbaugh rejoignit Grady Booch chez Rational Software Corporation en 1994. En 1995, cette dernière acquit la société Objectory AB d'Ivar Jacobson.

Tous trois, souvent surnommés *Les trois amigos*, rédigèrent la version 0.9 d'UML qui fut publiée à OOPSALA 96 en juin 1996, texte qui sera soumis à l'OMG en janvier 1997 sous le numéro de version 1.0. Cette version sera amendée par des propositions émanant des membres de l'OMG et donnera lieu à la version 1.1 soumise en septembre 1997 et adoptée en novembre de la même année. En juin 1998, la version 1.2 apportera des modifications mineures au langage UML alors que la version 1.3 fera l'objet de modifications plus conséquentes comme l'intégration du langage de contraintes *Object Constraint Language* (OCL) ou la génération d'interfaces via le langage *Interface Definition Language* (IDL). Cette dernière sera soumise en juin 1999 et adoptée en mars 2000. La version 1.4, adoptée en septembre 2001, introduira des concepts et des spécifications détaillées sur les notions de composants et de profils. Adoptée en mars 2003, la version 1.5 ajoutera la sémantique des actions.

La version 2.0 a été l'occasion d'une refonte majeure du langage UML dont les spécifications ont été mises en accord avec celles du *Meta Object Facility* (MOF) méta-métamodèle du langage UML. Pour ce faire, les membres de l'OMG ont extrait les concepts identiques du langage UML et du MOF et les ont mutualisés au sein d'une nouvelle architecture dénommée UML 2.0 Infrastructure. Cette nouvelle organisation a permis d'alléger les spécifications d'UML 2.0 et celles du MOF 2.0. Les nouvelles spécifications d'UML sont décrites dans le document intitulé UML 2.0 Superstructure. L'infrastructure d'UML 2.0 a été adoptée en septembre 2003 et la superstructure en octobre 2003. La version UML 2.0 est l'aboutissement d'une collaboration intensive de 3 ans entre les membres de l'OMG.

Entre les versions 1.0 et 2.0, sept ans se sont écoulés et sept versions ont été adoptées par les membres de l'OMG ce qui correspond à une version en moyenne par an. Ceci montre l'intérêt et le dynamisme de cette communauté.

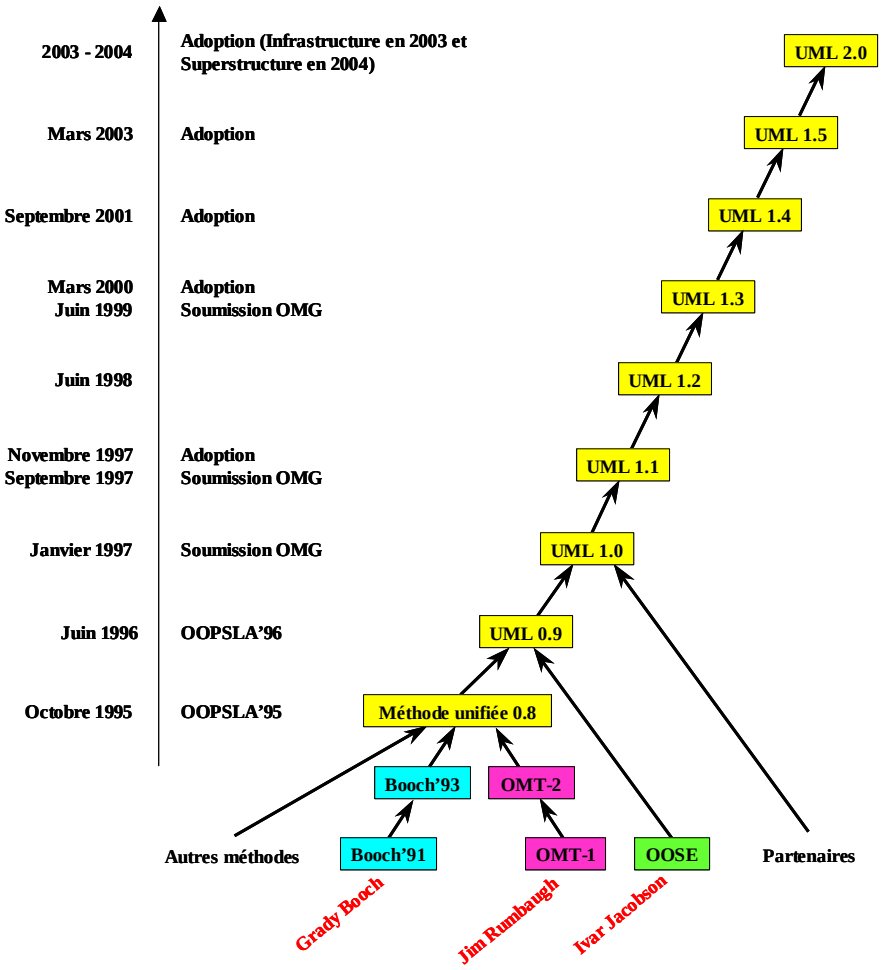


Figure 122 Genèse d'UML et principales étapes de standardisation et d'évolution.

III LES CONCEPTS UML ET LEURS ANNOTATIONS

Le but de ce paragraphe est de donner au lecteur non initié les bases minimales permettant de lire et de décrypter un modèle UML. Il ne s'agit pas de faire une présentation exhaustive des concepts UML mais plutôt de rappeler ceux qui sont incontournables. Bien d'autres concepts existent mais ils ne sont pas évoqués ici. Pour en savoir plus, le lecteur pourra se référer aux nombreux ouvrages sur le langage UML (Bennett *et al.*, 1999 ; Booch *et al.*, 2000 ; Fowler, 2004b ; Fowler *et al.*, 2001b ; Larman, 2002a, 2002b ; Muller *et al.*, 2000 ; OMG, 1999, 2003b ; Roques *et al.*, 2002 ; Saint Martin, 2002a, 2002b).

III.1 Le concept UML de Classe

Une *Classe* est un concept UML qui permet de représenter un concept thématique réel (*Tracteur*) ou virtuel (*Figure Géométrique*) ayant des caractéristiques génériques, c'est-à-dire des propriétés communes et/ou des comportements identiques. Le concept thématique modélisé par l'élément de modélisation *Classe* est généralement un ***ensemble*** ou ***une collection d'objets*** ayant tous les mêmes propriétés et les mêmes comportements.

Une *Classe* résulte donc de la démarche intellectuelle d'isoler par la pensée les propriétés et les comportements d'un concept thématique (matériel, virtuel, etc.). Dans une classe, les propriétés sont représentées par le concept UML d'*Attribut* et les comportements par le concept UML d'*Opération*.

Le signifiant[95] du concept UML *Classe*, c'est-à-dire sa représentation graphique autrement dit sa notation, est un rectangle comportant trois[96] champs (cf. figure 123-modèle de gauche) : un champ supérieur portant le nom du concept thématique modélisé, un champ central dédié au concept UML d'*Attribut* et un champ inférieur réservé à celui d'*Opérations*. Pour faciliter la lisibilité des modèles, certains ateliers de génie logiciel offrent la possibilité de réduire, sans perte d'information, le rectangle au seul champ supérieur portant le nom du concept modélisé (cf. figure 123-modèle de droite).

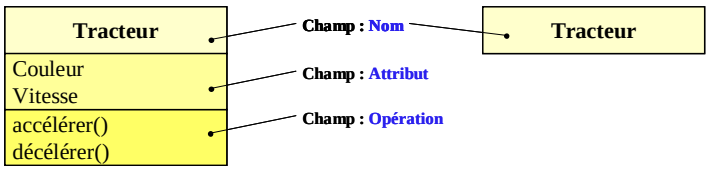


Figure 123 Exemple de *Classe*.

La figure 123 montre l'exemple d'une classe *Tracteur* modélisant le concept réel de *Tracteur* qui a comme propriétés génériques la couleur et la vitesse et comme comportements communs de pouvoir *Accélérer* et *Décélérer*. La classe *Tracteur* énumère donc les propriétés (couleur/vitesse) du concept thématique *Tracteur* et liste les comportements (accélérer/décélérer) sans préciser comment ces derniers sont déclenchés ou comment ils s'articulent entre eux.

Le concept UML de *Classe* est l'élément de modélisation central participant à la description de la structure statique d'un système. Une classe regroupe une famille (ensemble) ou une collection d'éléments ayant tous les mêmes propriétés et les mêmes comportements.

III.2 Le concept UML d'Instance

Le concept UML d'*Instance* permet de modéliser un des éléments de la collection représentée par la classe d'origine. En formalisme objet, cet élément de modélisation est appelé indifféremment *Instance* ou *Objet*. La représentation graphique (signifiant UML) du concept UML d'*Instance* est représentée par un rectangle à 2 champs : un champ supérieur où apparaît le nom de l'instance et un champ inférieur portant les propriétés valuées de l'instance. Le nom de l'instance est toujours souligné. Il est composé du nom proprement dit de l'instance situé à gauche des deux-points et suivi du nom de la classe d'origine (cf. figure 124-modèle de gauche). Dans le champ inférieur, les attributs de la classe d'origine sont listés et une valeur leur est affectée. C'est l'instance qui réagira selon les comportements définis au niveau de la classe.

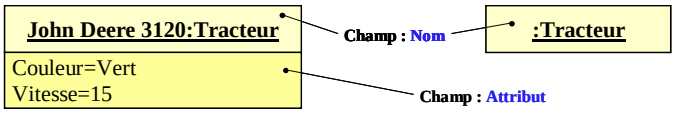


Figure 124 Exemple d'une *Instance nommée* à gauche et d'une *Instance anonyme* à droite.

Une instance ayant un nom tel que décrit ci-avant s'appelle *Instance nommée*. Le langage UML a également défini les *Instances anonymes*, c'est-à-dire sans nom à droite des deux-points (cf. figure 124-modèle de droite). Ces dernières sont souvent utilisées pour expliciter des traitements sous forme générique. Comme pour les classes, certains ateliers de génie logiciel permettent de masquer le champ inférieur pour faciliter la lecture des digrammes.

L'exemple de la figure 124 montre l'*Instance* de la classe *Tracteur* qui s'appelle *John Deere 3120* et qui modélise le tracteur de la figure 125. Cet objet *John Deere 3120* est de *Couleur* verte et sa *Vitesse* est de 15 km/h. Il possède les comportements définis par la classe *Tracteur* c'est-à-dire la possibilité d'accélérer et de décélérer afin d'adapter la vitesse à l'environnement.

---

[95] Au sens de la Théorie du langage (cf. Annexe III).

[96] Des champs supplémentaires peuvent être introduits sous le champ opération pour représenter les exceptions et/ou les responsabilités (Muller *et al.*, 2000).

Figure 125 Tracteur John Deere 3120.

Les *Instances* sont créées soit lorsque l'application est initialisée soit par le programmeur pour réaliser un traitement. Suivant les langages de programmation, c'est une *Opération create()* qui permet sa réification. Lorsque son existence n'est plus nécessaire, elle est détruit par une autre *Opération* souvent appelée *destroy()*. Une instance a donc une durée de vie variable qui peut atteindre au maximum celle d'une instance de l'application.

Le concept UML d'*Instance* est l'élément de modélisation qui permet de s'intéresser à un élément particulier d'un ensemble ou d'une collection d'objets décrit par une *Classe*. La valeur de ces propriétés peut évoluer au cours sa durée de vie.

III.3 Le concept UML de Lien

Le concept UML de *Lien* permet de matérialiser une relation entre les objets de deux ensembles. C'est une instance d'une relation entre classes, concept que nous traiterons au § III.4. L'exemple de la figure 126 montre les relations entre un ensemble d'*Exploitants* agricoles et un ensemble de *Tracteurs* disponibles chez des concessionnaires proposant un service de location ou de prêt.

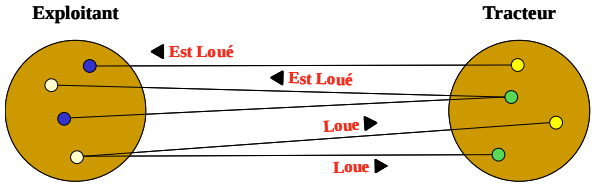


Figure 126 Exemple de relations entre un ensemble d'*Exploitants* et un ensemble de *Tracteurs*.

La notation graphique (signifiant UML) d'un *Lien* est un trait entre les deux objets en relation (cf. figure 127). Le concept de *Lien* a possède les décorations suivantes :

- Un nom *Location* situé entre les deux instances et calligraphié en italique. Étant donné qu'il matérialise la relation existant entre deux *Instances*, il est donc souligné comme ces dernières.
- Deux rôles, *Client* et *Équipement* situés aux extrémités du *Lien*, qui représentent l'implication des classes adjacentes. Ce sont aussi les rôles joués par les classes comme nous allons voir ci-dessous.

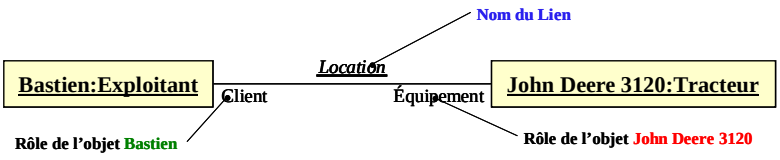


Figure 127 Exemple de *Lien*.

En phase d'analyse, le nom et les rôles peuvent ne pas être spécifiés. Ils seront complétés au fur et à mesure de l'avancement du projet. Nous verrons dans le § III.4 comment le nom et les rôles du *Lien* sont déterminés afin d'obtenir un modèle ayant un sens thématique.

III.4 Relations entre classes

Le langage UML met à disposition des concepteurs trois types de relations pour exprimer le rapport existant entre deux classes : le concept UML d'*Association*, celui d'*Agrégation Partagée* et celui d'*Agrégations de Composition*. Concernant les deux derniers types, nous utilisons une terminologie proche de celle de Craig Larman (Larman, 2002a) qui exprime mieux les notions à modéliser. Nous allons décrire ci-dessous ces trois types de relations.

III.4.1 *Le concept UML d'Association*

Le concept UML d'*Association* est employé pour mettre en correspondance deux classes[97], c'est-à-dire deux ensembles d'objets. Elle matérialise une relation stable entre les deux classes. De la même façon que le concept de *Classe* représente une famille d'*Instances*, le concept d'*Association* « factorise » une famille de *Liens*. Dans sa forme la plus simple, la représentation graphique (signifiant UML) d'une *Association* est un trait reliant les deux classes impliquées dans la relation. La nature de l'*Association* est donnée par son nom en italique situé entre les deux classes (cf. figure 128). Quelquefois, il est omis. La décoration de la figure 128 est souvent suffisante en phase d'analyse.

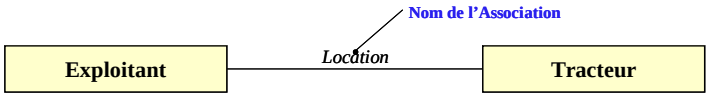


Figure 128 Exemple d'*Association* - décoration simplifiée.

En phase de conception (cf. figure 129), elle est complétée aux extrémités par des rôles et des cardinalités relatives à la classe adjacente[98].

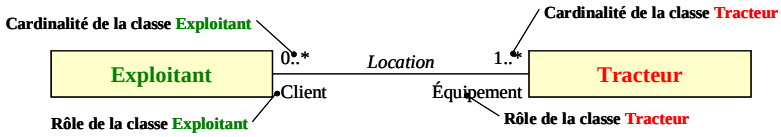


Figure 129 Exemple d'*Association* - décoration complète.

L'exercice de nommer l'*Association* et de définir le rôle et la cardinalité de chacune des classes situées aux extrémités ne doit pas être fait à la légère car, dans le cas contraire, le modèle est sémantiquement soit incomplet, soit incompréhensible et ensuite il est assez difficile d'appréhender le sens du modèle. Une méthode simple pour obtenir un modèle sémantiquement correct consiste à qualifier l'*Association* et les *Rôles* en puisant dans la terminologie du domaine afin d'établir des phrases du type :

- ⇨ Exemple de la figure 128 : le concept d'*Exploitant* s'appuie sur le concept de *Tracteur* par le biais de la relation *Location*.
- ⇨ Exemple de la figure 129 :
  - Un *Exploitant* est en relation de *Location* avec *1 à n Tracteur(s)* qui joue le rôle d'*Équipement*.
  - Un *Tracteur* est en relation de *Location* avec *0 à n Exploitant(s)* qui joue le rôle de *Client*.

[97] Le langage UML possède aussi le concept d'Association N-aire pour mettre en correspondance une ou plusieurs classes avec une relation de même nature.

[98] Ici nous ne nous attarderons pas sur la navigabilité des associations, décoration des extrémités nécessaire en phase d'implémentation comme le décrit parfaitement (Fowler *et al.*, 2001b).

Dans l'exemple de la figure 129 , la première phrase caractérise complètement la relation dans le sens *Exploitant / Tracteur* et la seconde détaille le sens *Tracteur / Exploitant*. Les deux phrases se complètent mutuellement pour définir totalement le modèle et toute les deux sont nécessaires pour obtenir un modèle qui ait du sens.

C'est un exercice délicat qui, s'il n'est pas fait correctement, donne très rapidement un modèle sémantiquement illisible. Aussi, le moyen de parvenir à un modèle compréhensible est d'effectuer cet exercice avec les acteurs du domaine car ceux sont eux qui connaissent le mieux les relations et les rôles entre les objets.

III.4.2 *Le concept UML d'Agrégation*

Le concept UML d'*Agrégation* est une association particulière traduisant un couplage fort entre deux concepts ou d'appartenance d'un concept à un autre, c'est-à-dire la subordination d'un concept à une autre. Elle permet de modéliser la relation existant entre la classe d'un concept composite et la classe d'un concept composant subordonné ou appartenant au concept composite. Elle met donc en correspondance un ensemble d'objets composites et un ensemble d'objets composants. Elle introduit donc une dissymétrie dans la relation puisque l'un des concepts a un rôle prépondérant. Il en existe deux types les *Agrégations Partagées* et les *Agrégations de Composition*. Beaucoup d'auteurs (Bennett *et al.*, 1999 ; Booch *et al.*, 2000 ; Fowler, 2004b ; Muller *et al.*, 2000 ; OMG, 2003a) emploient une terminologie plus concise en qualifiant la première d'*Agrégation* et la seconde de *Composition* mais le vocabulaire que nous utiliserons et que l'on retrouve chez Craig Larman (Larman, 2002a) traduit mieux la différence entre les deux concepts, différence que nous allons voir ci-après.

III.4.2.1 Le concept UML d'Agrégation Partagée souvent qualifié plus sommairement d'Agrégation

Une *Agrégation Partagée* est une agrégation faible dont la destruction physique de l'objet composite ***n'entraîne pas*** la destruction de ***tous*** les objets composants. Nous détaillerons ce point à travers l'exemple de la figure 131 mais au préalable nous allons préciser la notation. La représentation graphique (signifiant UML) d'une *Agrégation Partagée* est analogue à celle d'une association à l'exception près du losange évidé décorant le rôle du concept composite. Cette extrémité porte généralement la cardinalité 1 à n.

La figure 130 donne un exemple d'*Agrégation Partagée* qui met en relation le concept composite d'*Équipe* avec l'une des composantes d'une équipe que sont les *Joueurs*. Ce modèle a un sens puisqu'il est possible de formuler les deux phrases suivantes :

- Une *Équipe* est en relation de *Composition* avec *1 à n Joueurs* qui jouent le rôle de *Composant*.
- Un *Joueur* est en relation de *Composition* avec *1 à n Équipes* qui jouent le rôle de *Groupe*.

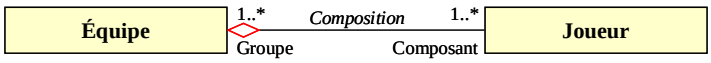


Figure 130 Exemple d'*Agrégation Partagée*.

Ce modèle de classe permet d'instancier deux objets représentant des équipes féminines (cf. figure 131) : une *Équipe de Basket* et une *Équipe de Tennis*. Trois joueuses de ces deux équipes ont pour prénom : *Chantal* pour la première, *Alicia* pour la seconde et *Claire* pour la troisième.

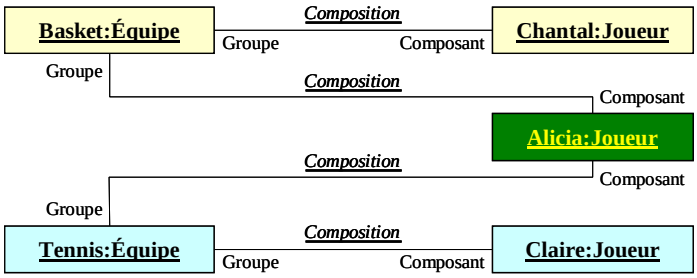


Figure 131 Modèle d'instances de deux équipes partageant la joueuse Alicia.

Les joueuses *Chantal* et *Claire* appartiennent exclusivement à l'*Équipe de Basket* et respectivement à l'*Équipe de Tennis* mais *Alicia* est une joueuse impliquée dans les deux équipes. Si pour des raisons de rentabilité par exemple, l'*Équipe de Basket* doit être dissoute ce qui se traduit en informatique par la suppression de l'objet, les objets la composant peuvent être détruits sauf bien évidemment l'objet *Alicia* qui doit continuer à exister afin de maintenir la cohérence de l'*Équipe de Tennis*.

III.4.2.2 Le concept UML d'Agrégation de Composition généralement appelé plus simplement Composition

Une *Agrégation de Composition* est une agrégation forte dont la destruction physique de l'objet composite ***entraîne*** la destruction de ***tous*** les objets composants. Sa représentation graphique (signifiant UML) est identique à celle d'une agrégation de composition sauf que le losange est plein. Le composite a une cardinalité de 1, c'est la raison pour laquelle elle est souvent omise.

La figure 132 donne un exemple d'une *Agrégation de Composition* qui met en relation le concept composite de *Voiture* avec le concept de Siège, un de ses constituants d'une voiture. Ce modèle est sémantiquement correct puisque les phrases suivantes peuvent être construites :

- Une *Voiture* est en relation de *Possession* avec *1 à 4 Sièges* qui jouent le rôle d'*Équipement*.
- Un *Siège* est en relation de *Possession* avec *1 Voiture* qui joue le rôle de *Support* de fixation.

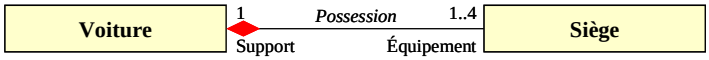


Figure 132 Exemple d'*Agrégation de Composition*.

De part la définition de l'*Agrégation de Composition*, le modèle de la figure 132 oblige le concepteur à réifier des objets voitures n'ayant en commun aucun objet siège comme l'illustre la figure 133 ce qui est thématique cohérent. Les deux agrégats sont totalement indépendants et la destruction de l'agrégat *Renault 5* n'a aucune incidence sur l'agrégat *Peugeot 205*.

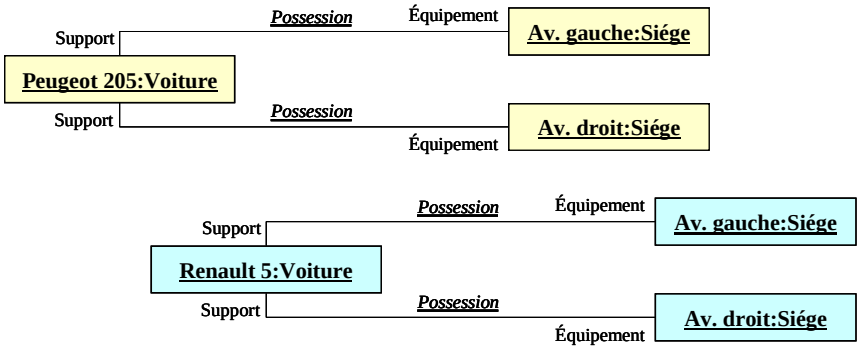


Figure 133 Modèle d'instances de deux voitures ne partageant aucun objet.

III.5 Le concept UML de Paquetage

Le concept UML de *Paquetage* est un concept de structuration des modèles qui regroupe les concepts thématiques d'un même domaine. Aussi, pour des modèles complexes, il est possible d'imbriquer les *Paquetages* entre eux. Cette structuration interne ne doit pas être anarchique, aléatoire ou dépendre du bon vouloir du concepteur, il faut qu'elle soit sémantiquement correcte c'est-à-dire qu'elle est un sens du point de vue thématique. Nous reviendrons sur ce point dans les exemples ci-dessous. La représentation graphique (signifiant UML) d'un *Paquetage* est un dossier portant le nom du paquetage au centre si le contenu du paquetage est masqué ou dans le rectangle supérieur si le contenu est détaillé (cf. figure 134).

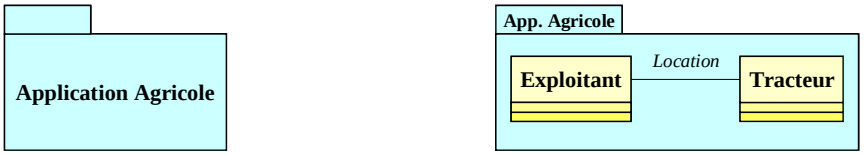


Figure 134 Exemple de *Paquetages*.

Lors de la conception de *Systèmes d'Information Géographique*, les géomaticiens manipulent constamment des concepts ayant des propriétés de spatialité (*Point*, *Ligne* et *Polygone*) ou de temporalité (*Instant* et *Période*). Une structuration possible de ces propriétés est de regrouper les propriétés de même nature au sein de paquetages dédiés : les concepts primaires de spatialité dans un premier paquetage nommé *Concepts Spatiaux* et les concepts primaires de temporalité dans un second paquetage appelé *Concepts Temporels*. Ces deux paquetages élémentaires peuvent être eux-mêmes inclus dans un troisième paquetage contenant tous les concepts primaires de spatialité et de temporalité indispensables à la conception et au développement des *Systèmes d'Information Géographique*. La figure 135 montre cette organisation.

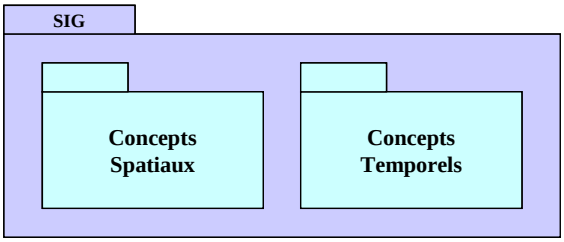


Figure 135 Exemple de structuration thématique des *Paquetages*.

La structuration interne des concepts permet aussi de faire cohabiter des concepts portant le même nom mais n'ayant pas le même sens car ils appartiennent à des domaines thématiques différents. Dans ce contexte, le *Paquetages* a une fonction d'***espace de nommage***.

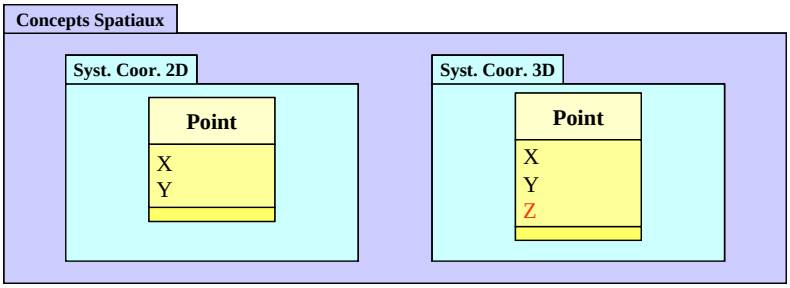


Figure 136 Illustration de la fonction d'espace de nommage d'un *Paquetage*.

Cet exemple montre le cas du concept *Point* dont les propriétés sont différentes suivant que l'entité référencée est repérée dans un système de coordonnées 2D ou 3D. Dans le premier cas, la classe *Point* a deux attributs X et Y alors que dans le second car elle en a trois X, Y et Z. Ici, le concept UML de *Paquetage* est utilisé pour faire cohabiter les deux classes de même nom *Point* en confinant chacune d'elle dans un *Paquetage* représentant le système de coordonnées correspondant. Chacun de ces deux paquetages constitue un espace de nommage reposant sur un concept thématique structurant.

III.6 Le concept UML de Spécialisation/Généralisation

Le concept UML de *Spécialisation* est un mécanisme qui permet d'exprimer le fait qu'un concept est un sous-ensemble d'un autre concept de niveau supérieur d'abstraction. Le premier est souvent qualifié de fils et le second de père.

La partie gauche de la figure 137 montre la relation d'inclusion ensembliste entre le concept d'*Humain* qui est un ensemble contenant le sous-ensemble des humains *Salariés* lui-même comprenant le sous-ensemble des *Informaticiens*. Lorsque ces trois concepts sont modélisés en langage UML par des classes, la propriété *Taille* de l'*Humain* va être répétée dans les concepts *Salarié* et *Informaticien* et son aptitude à *marcher()* aussi. Il en est de même pour la propriété *Salaire* et le comportement *travailler()* du concept *Salarié* vis-à-vis du concept d'*Informaticien*. Dans le modèle de droite de la figure 137, rien n'indique que les trois classes *Humain*, *Salarié* et *Informaticien* sont dans une relation ensembliste.

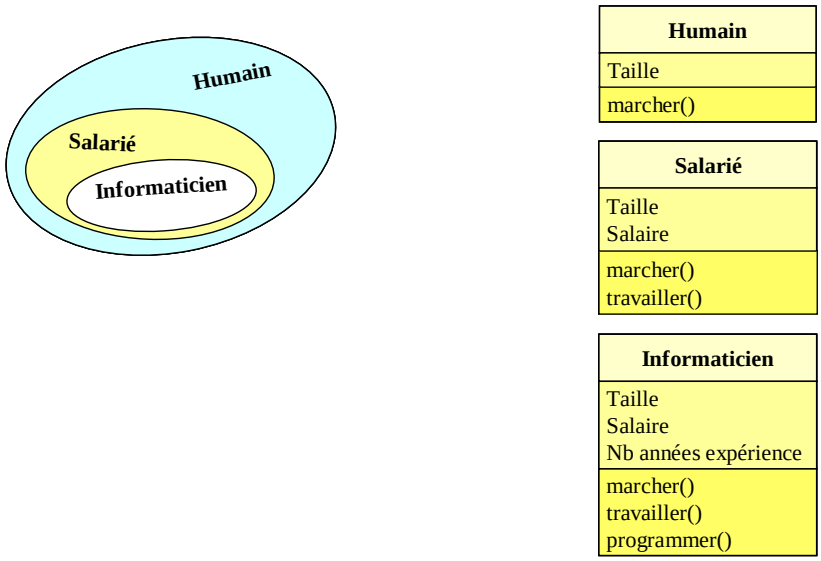


Figure 137 Exemple de trois concepts liés par une relation d'inclusion.

Le concept UML de *Spécialisation* a été défini pour exprimer la relation ensembliste entre un concept père et un concept fils. Ce mécanisme s'applique aussi bien aux classes qu'aux paquetages[99]. On parle alors de classe fille et de classe mère, de paquetage fil et paquetage père. Si un concept est sous-ensemble d'un autre concept, il en hérite les propriétés et les comportements du concept père ce qui permet de simplifier la notation et par voie de conséquence accroît la lisibilité du modèle.

La représentation graphique (signifiant UML) est une flèche à extrémité triangulaire évidée reliant les deux classes et orientée de la classe fille vers la classe mère. L'organisation des classes résultante forme une *hiérarchie de classes* (cf. figure 138-modèle de gauche).

La partie gauche de la figure 138 montre un exemple de relation de *Spécialisation* entre les concepts *Humain*, *Salarié* et *Informaticien*. Dans les classes filles, seuls les propriétés et les comportements supplémentaires sont mentionnés. Toutefois, elles héritent des propriétés et des comportements classes mères.

La partie droite de la figure 138 illustre une relation de *Généralisation* entre paquetages. Dans cet exemple, les concepts de spatialité et de temporalité sont des sous-ensembles des concepts abstraction supérieure que sont les concepts SIG utilisés dans le domaine de l'*Information Géographique*.

Le processus de modélisation qui vient d'être décrit part du concept très abstrait d'*Humain* pour atteindre le concept plus précis d'*Informaticien*. On parle alors de *Spécialisation*. Une autre démarche de modélisation serait de créer les concepts de granularité très fine (*Informaticiens*, *Mécaniciens*, etc.) et de trouver des concepts d'abstraction supérieure factorisant ou mutualisant les propriétés et les comportements des concepts fils. On parle alors de *Généralisation*. *Spécialisation* et *Généralisation* sont deux mécanismes duaux de construction d'une hiérarchie de classes ou de paquetage.

[99] Il est possible d'utiliser ce mécanisme entre d'autres concepts UML que nous n'aborderons pas ici.

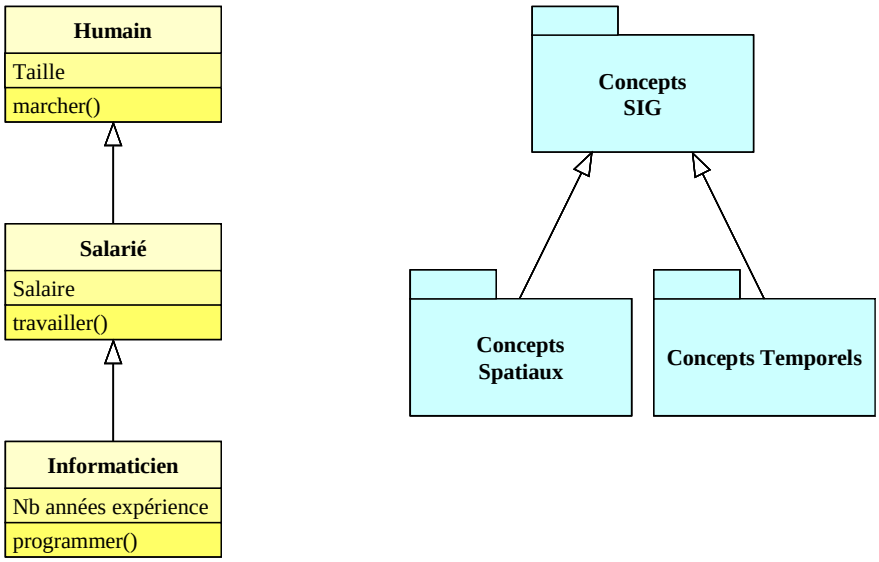


Figure 138 Exemple de *Spécialisation/Généralisation.*

III.7 Annotations supplémentaires des éléments de modélisation

Malgré les nombreux concepts et les éléments de modélisation associés définis par UML pour modéliser un domaine en vue de produire une application informatique, il est impossible d'imaginer tous les cas et toutes les nuances possibles qui peuvent être rencontrés au cours de l'analyse d'un système. Face à cette difficulté, les concepteurs d'UML ont doté ce langage d'annotations permettant son extension sémantique. Nous n'aborderons ici que les deux concepts les plus employés dans le présent document : le *Stéréotype* et la *Valeur marquée.*

III.7.1 *Le concept de Stéréotype*

Le concept UML de *Stéréotype* est un type d'annotation, souvent thématique, ajoutant une sémantique particulière et supplémentaire à un élément de modélisation. Il permet d'étendre et de préciser la sémantique de l'élément de modélisation. C'est un concept qui s'applique à tous les éléments de modélisation du langage UML. Il existe quelques exceptions souvent liées à l'instanciation d'UML dans l'atelier de génie logiciel. Un *Stéréotype* est employé lorsque le concepteur veut mettre exergue un ***concept métier/thématique majeur***. Sa représentation graphique (signifiant UML) est constituée du nom du concept thématique encadré par des chevrons ouvrant et fermant (cf. figure 139). Il est porté par le concept UML stéréotypé.

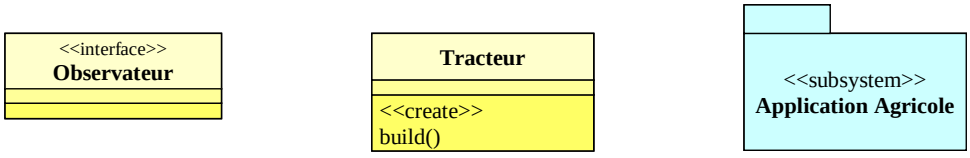


Figure 139 Exemple de *Stéréotypes* standards du langage UML.

Le *Stéréotype* <<interface>> est porté par la classe *Observateur* de la figure 139 et signifie que cette classe offre un service. Le *Stéréotype* <<create>> renseigne l'opération *build()* de la classe *Tracteur* et indique que cette opération est un constructeur. Le paquetage *Application Agricole* est stéréotypé <<subsystem>> indiquant ainsi qu'il fait partie d'une application informatique plus importante. Les stéréotypes qui viennent d'être évoqués sont des stéréotypes standards du langage UML qui en propose bien d'autres.

Certains ateliers de génie logiciel offre la possibilité de créer de nouveaux stéréotypes afin d'augmenter la sémantique disponible. Nous avons mis à profit ce mécanisme d'extension, appelé Profil, pour réaliser la

présente recherche aussi bien pour les transformations géomatiques dédiées aux *Systèmes d'Information Géographique* (cf. Chapitre 5-III) que pour les transformations SQL (cf. Chapitre 5-IV).

Nous avons en particulier étendu le langage UML (cf. Chapitres 4 et 5) en ajoutant les concepts de spatialité (*Point*, *Ligne* et *Polygone*) via le concept UML de *Stéréotype*. Pour ce faire, nous avons défini les stéréotypes suivants : <<Gis S : Point>>, <<Gis S : Line>> et <<Gis S : Polygon>>.

Nous avons adopté l'élément de modélisation *Stéréotype* pour exprimer la spatialité et la temporalité car ceux sont les propriétés majeures caractérisant les entités référencées. Ceux sont elles qui confèrent le statut d'entité référencée à un concept thématique.

En outre, certains ateliers de génie logiciel offrent la possibilité d'associer un pictogramme à l'élément de modélisation *Stéréotype* introduisant par là même un langage visuel. C'est le cas de l'atelier de génie logiciel Objecteering utilisé au Cemagref. La figure 140 illustre la classe *Parcelle* portant le *Stéréotype* <<Gis S : Polygon>> sous la forme littérale à gauche ou sous la forme pictogrammique à droite. Cette annotation ajoute une information supplémentaire en précisant le type de spatialité de l'entité référencée *Parcelle* : les objets parcelles ont une géométrie polygonale. L'ajout d'une annotation textuelle ou pictogrammique à la classe *Parcelle* facilite la communication entre le concept et les acteurs du domaine ce qui améliore la qualité du modèle.

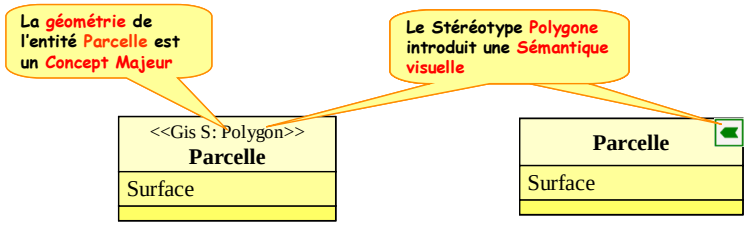


Figure 140 Exemples de classes stéréotypées étendant le langage UML au domaine géomatique.

III.7.2 *Le concept de Valeur marquée (Tagged Value)*

Comme le concept de stéréotype, celui de *Valeur marquée* est une annotation qui introduit, sur l'élément de modélisation, une information supplémentaire mais de ***« moindre importance »*** que celle ajoutée par un stéréotype. De la même façon, elle permet d'étendre la sémantique du langage UML pour un domaine particulier mettant à disposition du concepteur de nouveaux concepts métiers ou thématiques. Comme le stéréotype, elle peut s'appliquer à tous les éléments de modélisation du langage UML à quelques exceptions près. À la différence de ces derniers, elles ne peuvent pas être associées à un pictogramme. Par contre, elle offre la possibilité d'introduire des paramètres. Sa représentation graphique (signifiant UML) est composée du nom du concept entre accolades s'il n'y a pas de paramètres (cf. figure 141) ou du nom du concept suivi de sa valeur, le tout entre accolades (cf. figure 142-modèle de gauche).

Figure 141 Exemple de *Valeur marquée* standard du langage UML.

Bien que le langage UML soit standardisé, son instanciation dans les ateliers de génie logiciel souffre souvent de différences minimes. C'est le cas de notre atelier de génie logiciel concernant les *Valeurs marquées*. Nous le signalons pour que le lecteur ne soit pas perturbé par cette différence de notation car nous avons utilisé les *Valeurs marquées* intensivement dans le Chapitre 5 en complément de l'information de spatialité des entités référencées et cette notation se retrouve dans les diagrammes produits avec notre atelier de génie logiciel. Dans ce dernier, la différence porte sur la représentation des paramètres. Dans une *Valeur marquée*, il est possible d'avoir plusieurs paramètres séparés par une virgule (cf. figure 142-modèle de droite). Ils sont listés entre parenthèse après le nom de la *Valeur marquée*.

La figure 141 montre la *Valeur marquée {persistence}* de la classe *Digue* qui permettra, par exemple, projeter cette classe en base de données. L'attribut *Pi* de la classe *Cercle* est étiqueté par la *Valeur marquée {frozen}* indiquant que cet attribut est une constante et qu'à ce titre il est "gelé".

La figure 142, partie de gauche, illustre l'utilisation de *Valeurs marquées* dont l'information vient compléter la géométrie linéaire de la classe *Digue*. Le nom de cette *Valeur marquée* permet de préciser qu'elle apporte une information sur le système de coordonnées dans lequel est repérée la géométrie. Ces *Valeurs marquées* sont affectées de la dimension du système de coordonnées, 1D pour la première et 2D pour la seconde. Dans cet exemple, un objet digue linaire pourra être repéré :

- Dans un système de coordonnées à une dimension (1D) représentant l'abscisse curviligne sur la digue c'est-à-dire la distance parcourue le long de la digue depuis une origine.
- Dans un système de coordonnées à deux dimensions (2D) afin de pouvoir dessiner par exemple sa forme exacte sur un plan.

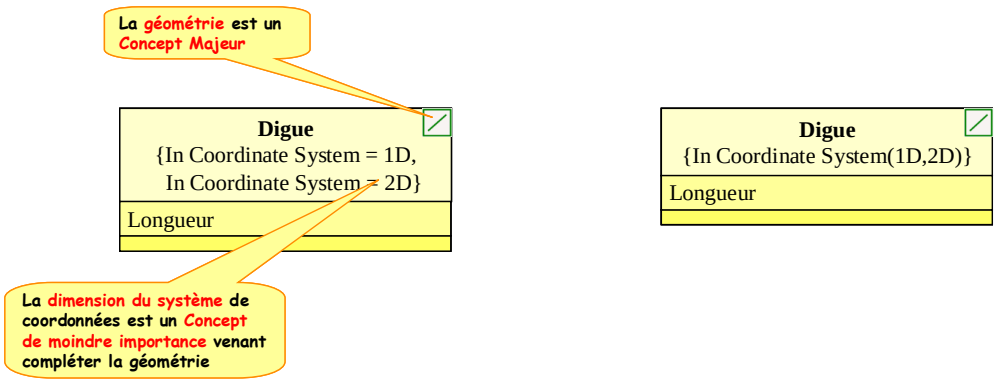


Figure 142 Exemple de *Valeur marquée* complétant l'information géométrique portée par le stéréotype.

Les *Valeurs marquées* de la classe *Digue* de gauche correspond à l'annotation selon les spécifications standards du langage UML. En partie droite, nous donnons l'annotation équivalente dans notre atelier de génie logiciel.

IV Les Diagrammes UML

La conception du langage UML a été centrée sur la communication. Aussi, les auteurs d'UML ont élaboré neuf diagrammes qui constituent différents points de vue sur le système.

Le *Diagramme des Cas d'Utilisation* permet en premier lieu de délimiter le système et donc sa portée. C'est aussi un outil de recensement et de description des cas d'utilisation du système qui concrétisent le comportement du système du point de vue de l'utilisateur. Il permet d'inventorier et de montrer les acteurs interférant avec le système ainsi que leurs relations avec les cas d'utilisation.

Les *Diagrammes de Classes* et *d'Objets* sont de vues complémentaires d'une même réalité (Muller *et al.*, 2000). Le premier représente une abstraction de cette réalité, représentation focalisée sur la structure, alors que le second en est une instance.

Le *Diagramme de Classes* est probablement le plus utilisé des diagrammes car il est utilisé pour décrire la structure statique du système en capturant tous les concepts composant le système et en particulier les concepts métiers via le concept UML de classe. Il saisit aussi les correspondances entre concepts au travers de relations. Il met en œuvre principalement les concepts UML de *Classe*, d'*Attribut* et d'*Opération* concepts indispensable pour la description des classes, d'*Association*, d'*Agrégation Partagée* et d'*Agrégation de composition*, de *Généralisation* et de *Paquetages*. La description inclut aussi les propriétés et les comportements liés aux concepts mais aussi les contraintes sur ces concepts. C'est un diagramme très complet.

Un *Diagramme d'Objets* représente une instance particulière d'un diagramme de classe ce qui sous-entend qu'un diagramme de classe peut générer plusieurs diagramme d'objets. Il modélise les instances des éléments participant à l'organisation structurelle d'un concept et montre les relations existant entre ces instances. Il donne une vue statique du système à un instant donné. Effectivement, certains objets sont créés au cours du fonctionnement de l'application afin de permettre la réalisation d'une fonctionnalité. Aussi, ils montrent souvent l'état d'un système avant ou après une interaction.

Les *Diagrammes de Séquence* et *de Collaboration* sont deux diagrammes dédiés à la modélisation des comportements dynamiques des applications. Ce sont des diagrammes homologues exprimant les mêmes concepts sous des formes différentes. Aussi, il est donc possible de convertir un diagramme de séquence en un diagrammes de collaboration et réciproquement et ce sans perte d'information (Booch *et al.*, 2000).

Le *Diagramme de Séquence* est un diagramme complémentaire au diagramme d'objets exprimant les interactions entre les objets. Ces interactions sont des messages échangés en réception ou en émission par les objets. Il met en évidence l'ordre chronologique des messages présentant ainsi le flot temporel des échanges suivant un axe vertical descendant. Il montre aussi l'existence, la création et la destruction au cours du temps des objets de l'application donnant ainsi un cliché sur leur durée de vie. Il informe sur les périodes d'activation des objets lorsqu'ils envoient ou reçoivent des messages. Pour finir, il fait apparaître les structures de contrôles tels que les boucles, les branchements conditionnels ou concurrents. De ce fait, il est particulièrement adapté à la modélisation des systèmes temps réels. Son inconvénient majeur est de devenir rapidement illisible lorsqu'il implique plusieurs instances ou que les échanges de messages sont complexes, le *Diagramme de Collaboration* lui est alors préféré.

Le *Diagramme de Collaboration* est équivalent au digramme de séquence mais avec une représentation plus spatiale des objets (Muller *et al.*, 2000) montrant ainsi leurs collaborations. Aussi, il ne décrit pas explicitement l'enchaînement des messages envoyés ou reçus par les objets, leur durée de vie ni leur temps d'activité. Pour pallier cette absence de dimension temporelle des échanges de messages, ces derniers portent un numéro de séquence. Ils permettent comme pour les diagrammes de séquence d'exprimer les structures de contrôles. Le diagramme de collaboration peut être vu comme un diagramme d'objets auquel ont été ajoutés les messages échangés.

Les *Diagrammes d'États-Transitions* et *d'Activités* viennent compléter la panoplie des diagrammes permettant d'exprimer la dynamique du système. Ce sont deux diagrammes complémentaires car le premier met l'accent sur les états du système alors que le second focalise l'intérêt du concepteur sur les transitions entre états (Muller *et al.*, 2000).

Le *Diagramme d'États-Transitions* est particulièrement adapté à la modélisation des systèmes à états finis comme les automates. Il montre l'état des objets ou des systèmes complexes, les transitions entre états et les événements déclenchant le changement d'état. Il modélise aussi la transition entre états concurrents, la synchronisation d'événements, etc. Un exemple classique est la description du cycle de fonctionnement d'un vérin hydraulique qui ne peut être que ouvert ou fermé. Le cycle de fonctionnement d'une machine à laver la vaisselle -rinçage, lavage et séchage- est un autre exemple.

Le *Diagramme d'Activités* montre le flot d'exécution des actions entre les objets d'un système. Il représente la disjonction et la synchronisation d'événements mais aussi la dichotomie des transitions pouvant faire évoluer un objet dans un état vers deux états distincts. À cette occasion, l'objet peut changer d'état et de nature. Le diagramme d'activités est parfaitement adapté à la modélisation de l'organisation fonctionnelle d'une entreprise et des échanges entre services.

Les *Diagrammes de Composants* et *de Déploiement* sont des diagrammes complémentaires car le premier est utilisé pour décrire les composants logiciels réalisant des services alors que le second montre leur localisation sur des équipements et les connexions entre ces derniers.

Le *Diagramme de Composants* illustre l'architecture logicielle d'une application informatique. C'est une représentation statique des entités logicielles offrant un service que sont les composants (fichiers de configuration, librairies de fonctions mathématiques ou de dessins, exécutables d'un serveur mail, etc.). Il met aussi en exergue les relations entre les composants logiciels de l'application. Le diagramme de composants est donc une vue sur les services rendus par les composants logiciels du système. Il permet de modéliser la réutilisation de composants logiciels préalablement développés évitant ainsi un surcroît de travail.

Le *Diagramme de Déploiement* montre l'architecture matérielle des équipements nécessaires au bon fonctionnement d'une application -ordinateurs, imprimantes, serveurs, etc.- et les relations entre ces matériels. Ces équipements constituent les nœuds de ce réseau et les relations correspondent aux types de connexions et aux protocoles d'échanges -RS 232, ADSL, TCP/IP, etc.- entre nœuds. Il donne aussi la localisation physique de composants logiciels dans les nœuds.

V CONCLUSION

UML est un langage au sens de la *Théorie des Langages* puisqu'il possède des concepts propres (*Paquetage, Classe, Attribut, Instance*, etc.) qui sont les signifiés du langage et des signifiants précis relatifs à chacun des concepts. Ces signifiants procurent à UML une représentation graphique parfaitement structurée. Comme nous l'avons vu, une classe est représentée par un rectangle ayant de 1 à 3 champs, chaque champ ayant une décoration précise, une association est symbolisée par un trait portant un certain nombre d'informations la définissant complètement, etc. L'agencement et mise en relation des concepts d'UML sont régis par des règles qui en constituent la syntaxe (ou grammaire).

UML est un langage de modélisation qui date d'une dizaine d'années. Depuis, il est en constante évolution comme le montre son histoire récente. Entre les versions 1.0 et 2.0, sept ans se sont écoulés et sept versions ont été adoptées par les membres de l'Object Management Group, consortium en charge de sa révision. Une version en moyenne par an. Ceci montre le dynamisme de cette communauté et l'intérêt croissant pour ce langage de modélisation. Ce dynamisme se convertit en faiblesse car UML atteint des niveaux de complexité tels qu'il devient quasiment impossible d'en appréhender toutes les finesses. Les deux documents définissant l'Infrastructure et la Superstructure d'UML 2.0 représentent 1000 pages. Aussi, il est difficile pour des modélisateurs occasionnels d'assimiler tous les aspects du langage et à plus forte raison les acteurs d'un système qui doivent participer à la définition des spécifications de l'application.

Souhaité par ces concepteurs, c'est un langage qui s’appuie sur le formalisme objet, formalisme dont il tire toute sa puissance. En effet, le formalisme objet est plus facile à assimiler ce qui permet d’améliorer la communication entre acteurs et concepteurs. Il en résulte que le système étudié est mieux décrit par les acteurs.

Son point fort réside dans son pouvoir de communication entre les analystes, les concepteurs, les acteurs du domaine et les utilisateurs. Il a été conçu dans ce but. Ceux sont les neuf diagrammes qui facilitent cette communication.

C'est un langage de modélisation qui est de plus en plus utilisé dans des communautés exogènes à l'informatique comme la normalisation ISO[100] de concepts agricoles et environnementaux.

[100] International Organization for Standardization

Annexe II. Description plus détaillée des méthodes Rapid Application Development, Processus Unifié et eXtreme Programming

Cette annexe fait la description détaillée de trois méthodes de conduite de projet majeures. Cette description a permis de s'imprégner des problèmes et des difficultés que rencontrent tout chef de projet, concepteur ou programmeur lors du développement d'une application. Ces méthodes proposent des solutions à ces problèmes et à ces difficultés.

Sommaire détaillé

I RAPID APPLICATION DEVELOPMENT (RAD)

Cette méthode est apparue dans les années 80 selon (Bénard, 2002a) à la suite d'un double constat. Le premier concerne la carence voire l'absence de communication entre les programmeurs et les utilisateurs, déficiences qui se traduisent par la réalisation d'applications inadaptées aux besoins des utilisateurs. Le second est relatif aux méthodes de conduite de projet telle que Merise qui allonge des délais de production et par conséquent les coûts de réalisation.

Le fondateur de cette méthode est James Martin auteur de nombreux ouvrages dont celui du même nom que la méthode, *Rapid Application Development* (Martin, 1991). Cette méthode a été adaptée au contexte français et promue en France entre autres par Jean-Pierre Vickoff, pleinement en accord avec les idées de James Martin. Il est auteur de l'ouvrage *RAD, le Développement d'application client-serveur* (Vickoff, 1995) et du site Internet http://www.rad.fr/ (Vickoff, Site Web non daté).

Aux dires mêmes de son auteur, l'objectif de cette méthode de conduite de projet est de changer radicalement le processus de développement afin de réaliser *très rapidement* des applications *de qualité*. Plus récemment, d'autres auteurs ont introduit un troisième objectif concernant la *maîtrise des coûts*. (Vickoff, Site Web non daté) voire des *coûts très bas* (RAD-TOOL, 2004 ; University of California, 1997a). Il est particulièrement intéressant de constater que les adeptes de cette méthode mettent au même niveau les quatre ingrédients -équipe, méthodologie, gestion et outils- nécessaires au développement d'une application (CASEMaker Inc., 2000 ; University of California, 1997b). Selon eux, ils constituent les aspects essentiels de la méthode RAD. Si l'un d'eux est défaillant, la vitesse de développement ne sera pas maximale. Il est donc primordial d'après eux d'arriver à mettre en œuvre une coordination parfaite de ces ingrédients. Pour bien monter l'importance qu'ils attachent à cette coordination, l'équipe de l'université de Davis a schématisé cette recommandation par quatre piliers identiques soutenant un parallélépipède symbolisant la méthode RAD (cf. figure 143).

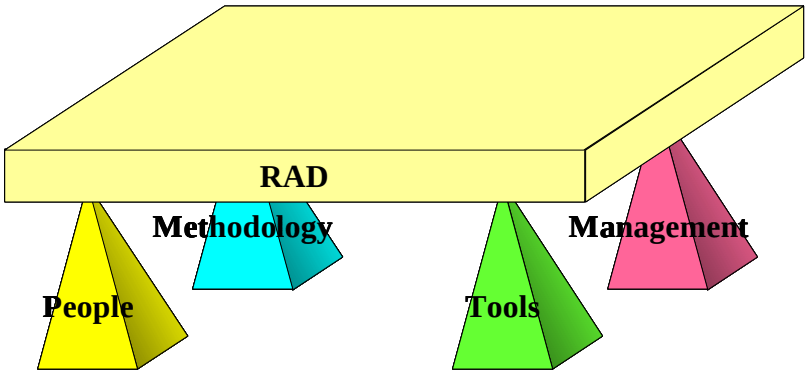


Figure 143 Les quatre aspects essentiels de RAD.

Les utilisateurs de cette méthode insistent particulièrement sur le fait que le triptyque *développement rapide, qualité et faible coût* vont de pair si une méthode de conduite de projet appropriée est utilisée (University of California, 1997a). La méthode RAD a aussi été conçue pour éviter la propagation des erreurs fonctionnelles et techniques des méthodes traditionnelles, appelé parfois "effet tunnel" (Web Ntic, 2003). Cette propagation a un *impact direct sur la qualité de l'application, les délais, le budget et la visibilité.*

Comme toute méthode de conduite de projet, la méthode RAD mobilise différentes "ingrédients". Cette méthode formalise l'*organisation de l'équipe*, la *méthodologie* employée et le *cycle de développement* mise en œuvre, la *gestion* du projet et des membres de l'équipe et les *outils* à préconiser afin d'atteindre le résultat escompté.

I.1 Organisation de l'équipe

Dérivant des méthodes classiques, elle est fortement imprégnée des mêmes fondamentaux : la séparation des rôles et des responsabilités entre la maîtrise d'ouvrage (MOA) et la maîtrise d'œuvre (MOE). La première définie les spécifications de l'application en relation avec le client et les communiquent à la seconde qui est chargée de trouver des solutions aux problèmes. La méthode RAD a volontairement accentué cette dichotomie par rapport aux méthodes classiques afin de bien dissocier les rôles et les responsabilités de la maîtrise d'ouvrage et de la

maîtrise d'œuvre renforçant ainsi leurs prérogatives. Afin que ce couple collabore et échange des informations sans souci, le concepteur de la méthode a mis en place une structure d'animation dénommée *Groupe d'Animation et de Rapport* dont le rôle principal est d'une part, de formaliser les mécanismes de communication entre la maîtrise d'ouvrage et la maîtrise d'œuvre et, d'autre part, d'encourager et de faciliter cette communication.

Ci-après, nous allons détailler les responsabilités de chacun des rôles -maîtrise d'ouvrage et la maîtrise d'œuvre- ainsi que celles de la structure d'animation.

I.1.1 *Maîtrise d'ouvrage*

La maîtrise d'ouvrage consiste à définir les objectifs et les besoins des utilisateurs de l'application, à valider les solutions proposées ainsi qu'à préparer et piloter les inéluctables changements intervenant en cours de développement. Trois acteurs majeurs interviennent en maîtrise d'ouvrage : le *Maître d'ouvrage*, le *Coordinateur délégué* et le *Responsable de la qualité fonctionnelle*.

Le *Maître d'ouvrage* a la responsabilité de fixer les objectifs -produit, coût et délai- et les orientations du projet ainsi que la validation finale des solutions proposées par la maîtrise d'œuvre. Pour ce faire, il rédige les documents de cadrage du projet dans lesquels il mentionne les objectifs stratégiques, fonctionnels, technologies, organisationnels ainsi que les contraintes propre au projet. Ces documents de cadrage constituent le référentiel pour toute décision. Avec le concours du client entre autres, il analyse les solutions proposées par la maîtrise d'œuvre en regard des exigences exprimées et, celle qui semble la plus pertinente, est validée. Il préside aussi le comité de pilotage.

Le *Coordinateur délégué* coordonne les activités de la maîtrise d'ouvrage et fait le suivi des objectifs et des activités de la maîtrise d'œuvre. Il s'intéresse essentiellement aux produits réalisés, aux coûts et à la charge de travail ainsi qu'aux délais de réalisation.

La responsabilité du *Responsable de la qualité fonctionnelle* est de s'assurer de la cohérence et de la qualité fonctionnelle de l'application, c'est-à-dire de s'assurer si l'application correspond bien aux besoins de l'utilisateur. Pour ce faire, il contrôle en particulier la cohérence des décisions et des choix relatifs aux aspects fonctionnels de l'application. Il est en charge des tests de validation de la qualité fonctionnelle.

I.1.2 *Maîtrise d'œuvre*

Ses responsabilités sont de définir et de soumettre à la maîtrise d'ouvrage des solutions techniques et méthodologiques permettant de répondant aux problèmes posés, de produire et de fournir des livrables intermédiaires pour montrer l'évolution du produit et obtenir un feedback et de respecter le Plan d'Assurance Qualité décidé par l'entreprise ou la maîtrise d'ouvrage. Comme pour la maîtrise d'ouvrage, trois acteurs contribuent au bon fonctionnement de ce rôle : le *Maître d'œuvre*, le *Pilote de Projet informatique* et le *Responsable par domaine*.

Le *Maître d'œuvre* a pour charge de proposer des objectifs, coûts et délais en autres, et les orientations du projet. Il arrête les propositions des solutions qui vont être transmises à la maîtrise d'ouvrage ainsi que celles qui seront implémentées. Il décide également des moyens et des méthodes à mettre en œuvre au sein de la maîtrise d'œuvre et il se coordonne, s'il y a lieu, avec les autres maîtrises d'œuvre du projet et en particulier avec celles de la sous-traitance.

Le *Pilote de Projet informatique* coordonne les différentes tâches de la maîtrise d'œuvre. Il participe à la validation des propositions de solution et les solutions qui seront in fine implémentées. Il évalue et met en œuvre les moyens de la maîtrise d'œuvre. Il est en particulier en charge de définir le planning de production ainsi que les méthodes et outils pour réaliser l'application.

Enfin, le *Responsable par domaine* pilote la réalisation d'un sous-ensemble du produit et suit l'équipe de production.

I.1.3 *Groupe d'animation et de Rapport (GAR)*

Le *Groupe d'Animation et de Rapport* a la charge de la logistique des réunions, du thème et du déroulement de ces réunions, de la formalisation des besoins exprimés par le client, de la gestion des comptes-rendus et de la planification générale des tâches. Dans un *Groupe d'Animation et de Rapport*, il y a deux acteurs qui ont des fonctions essentielles au bon déroulement des réunions : l'*Animateur* et le *Rapporteur*.

L'*Animateur* est un acteur qui doit mener et orienter les débats au cours des réunions de façon à assurer le bon déroulement de celle-ci. Pour cela, il doit intervenir sur la forme des débats mais en aucun cas sur le fond. Son rôle est aussi d'une part, de recentrer les échanges et les énergies des participants sur les thèmes initiaux de

la réunion en synthétisant et reformulant l'essentiel des propos et, d'autre part, de favoriser l'expression des opinions des uns et des autres en les questionnant pour qu'ils approfondissent une idée, qu'ils précisent un problème, etc. Une des qualités premières de l'*Animateur* doit être sa neutralité afin de pouvoir formuler des avis impartial. Pour cela, il doit être indépendant des deux maîtrises et doit donc être rattaché fonctionnellement soit à la direction générale (B. Boehm *et al.*, 1994 ; Egyed *et al.*, 1998) soit à un service médiateur du service informatique en charge du projet et du service demandeur de l'application (Vickoff, 2000).

Le *Rapporteur* est souvent un analyste concepteur confirmé ayant une forte expérience qui doit lui permettre de modéliser le système en temps réel lors des réunions consacrées à la conception. Il doit être en mesure de produire une documentation automatiquement et de la maintenir à jour. Pour satisfaire ces deux obligations, la méthode RAD préconise fortement l'utilisation d'un atelier de génie logiciel. Il est responsable aussi de la réalisation des comptes-rendus de réunions.

I.2 Méthodologie et Cycle de développement

Le cycle de développement préconisé par son auteur est un cycle court de 90 jours optimum avec un maximum de 120 jours (Martin, 1991). La borne inférieure a été déplacée (Vickoff, 2000) afin de pouvoir appliquer cette méthode au développement d'applications de taille réduite. Un projet développé suivant la méthode RAD demande d'engager des ressources humaines compétentes donc coûteuse. Aussi, il est inconcevable de ne pas dégager les moyens financiers nécessaires pour que l'équipe puisse disposer de salles de travail correctement agencées, d'outils de modélisation et de développement performants, des vidéoprojecteurs, etc. Sans ce contexte fonctionnel, la démotivation peut gagner rapidement l'équipe pénalisant ainsi soit la qualité du développement soit la durée du projet, voire les deux. Il en résulte en général un surcoût. Le cycle de développement selon la méthode RAD se compose de trois phases principales voulues pas son auteur) -une phase de *Cadrage*, une phase de *Design* et une phase de *Construction*- auxquelles a été ajoutées ultérieurement une phase d'*Initialisation* et une phase de *Finalisation*. La figure 144 illustre ces cinq phases.

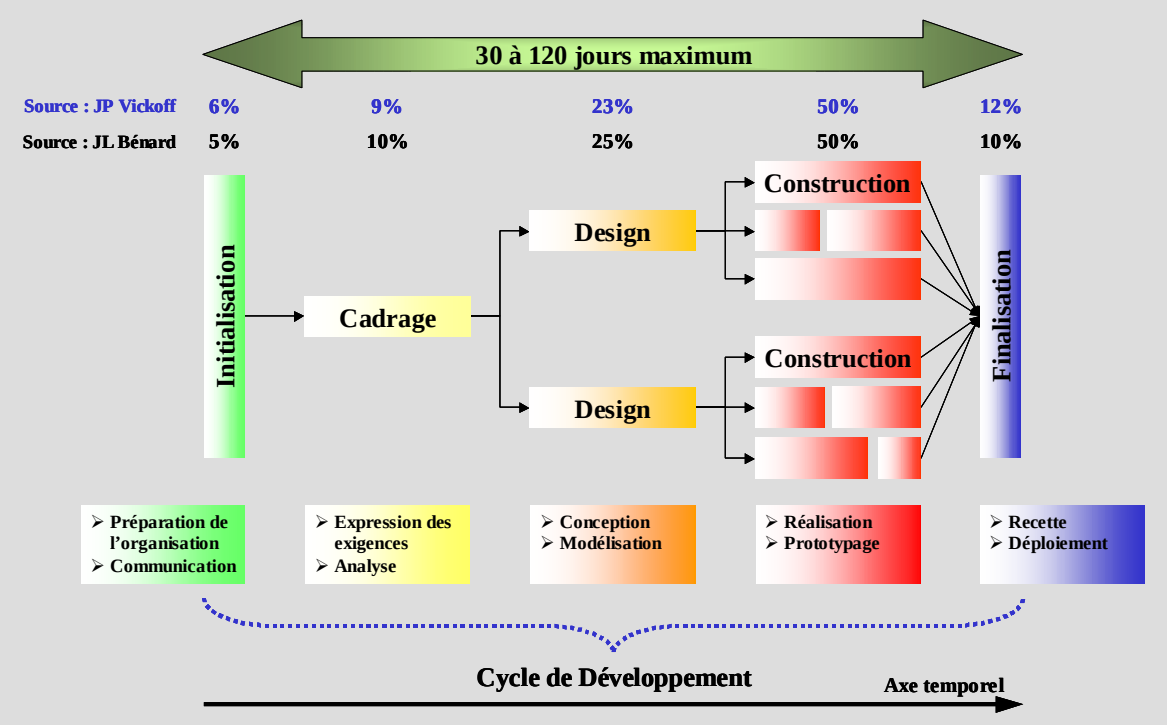


Figure 144 Cycle de développement d'un projet RAD (Bénard, 2002a ; Vickoff, 2000).

Selon les auteurs (Bénard, 2002a ; Vickoff, 2000), le temps imparti à chacune d'elles varie légèrement. La phase d'*Initialisation* représente 5 à 6% du temps prévue pour le projet, la phase de *Cadrage* en occupe de 9 à 10%, la phase de *Design* en sollicite quant à elle de 23 à 25%, la phase de *Construction* reste la plus gourmande avec 50% de ce temps et la phase de *Finalisation* en accapare 10 à 12%.

I.2.1 *Phase d'Initialisation (préparation de l'organisation et communication)*

La phase d'*Initialisation* démarre par la présentation aux acteurs du projet du cadre formel de la méthode de conduite RAD et des contraintes qui en découlent. Au cours de cette phase, la définition du projet dans ses grandes lignes est effectuée avec la participation active des utilisateurs ainsi que la structuration en domaines de l'application. La charge de travail est cadrée au cours de cette phase et permet d'en déduire les moyens nécessaires au bon déroulement du projet. C'est aussi l'occasion de recenser les spécialités rares qui seront nécessaires en cours de projet et de cibler les personnes ressources à engager ou à solliciter. Un *Plan de communication* est produit. Il liste l'ensemble des acteurs du projet. Une première réunion, appelée *Réunion de lancement*, rassemble, pendant une demi-journée à une journée, l'ensemble des acteurs impliqués dans le projet. La spécification des exigences y est effectuée en collaboration avec des utilisateurs et les tâches préliminaires y sont réparties. Les utilisateurs impliqués dans le projet sont sélectionnés en fonction de leur motivation réelle. Le choix d'utilisateurs motivés et aptes à formuler leurs savoir-faire ou leurs observations est primordial pour produire une application conforme aux besoins. Ils constituent une *force de proposition fonctionnelle* (Vickoff, 2000).

La réunion de lancement ponctue aussi le début du projet. Toutes ces actions permettent d'initier une dynamique de projet entre les différents acteurs.

I.2.2 *Phase de Cadrage (analyse et expression des exigences)*

La phase de *Cadrage* suit un processus *top-down* pour la définition des choix stratégiques et opérationnels. La Direction Générale fixe des objectifs hiérarchisés du projet et la maîtrise d'ouvrage détermine le cahier des charges ou bien la hiérarchie des fonctionnalités à réaliser. Ce sont essentiellement les décideurs qui sont impliqués dans cette phase et bien évidemment des utilisateurs. Ces derniers interviennent dans la spécification détaillée des fonctionnalités. Pour ce faire, les domaines précédemment définis sont découpés en thèmes (Vickoff, 2000). Les utilisateurs expriment leurs besoins au sein de commissions consacrées à un thème particulier. Si un domaine est relativement bien cerné, il ne fait pas l'objet d'une étude particulière. Par contre, si le domaine n'est pas bien maîtrisé ou si certaines fonctionnalités présentent des difficultés de compréhension ou de réalisation, son analyse est approfondie avec une panoplie plus large d'interlocuteurs. Des techniques d'entretiens de groupe sont employées pour faciliter l'expression des besoins et l'affinement des spécifications. La modélisation est aussi largement employée comme outil de communication entre les utilisateurs et les responsables du projet. Des séances de vidéoprojection permettent de faire évoluer de façon interactive un modèle au sein d'un atelier de génie logiciel. La durée des séances de travail consacrées à une commission oscille entre de 2 et 5 jours suivant l'importance, en termes de volume de travail, du thème. En fin de phase de *Cadrage*, si les exigences et les ressources ne sont pas en adéquation, les fonctionnelles de l'application sont classées suivant un niveau de priorité en termes de retour sur investissement.

Au cours de cette phase, l'animateur amène les acteurs de cette phase à stabiliser les exigences, à bloquer les budgets, à s'engager sur des délais et à fournir des plans stratégique, fonctionnel, technologique et organisationnel.

I.2.3 *Phase de Design (conception et modélisation)*

La première partie de la phase de *Design* relève aussi d'un processus *top-down*. Les utilisateurs détaillent davantage les spécifications de l'application au cours de séances de travail en commun avec les développeurs. Afin de faciliter la communication entre ces deux types d'acteurs, l'utilisation intensive d'un atelier de génie logiciel est recommandée et, si possible, en association d'un vidéoprojecteur dans une salle dédiée au travail en groupe. Jean-Pierre Vickoff (Vickoff, 2000) souligne l'utilité de la modélisation afin de réduire la complexité de la réalité et accélérer la compréhension du problème. Toutefois, il rappelle que cette *opération [NDLR : la modélisation] est considérée comme parasitaire en regard* de l'implémentation réelle des fonctionnalités demandées. Les utilisateurs et les développeurs portent donc leur effort principal sur la modélisation des données intervenant dans l'application et sur celle des traitements. Plusieurs formalismes de modélisation peuvent être en œuvre si nécessaire puisque les modèles sont avant tout des produits pour favoriser la communication entre utilisateurs et développeurs. La maîtrise d'œuvre doit s'attacher à structurer l'application afin de pouvoir lui conférer une grande capacité dévolution et donc une meilleure pérennité. La méthode RAD préconise de commencer l'implémentation par la construction des thèmes les plus stables et de réaliser les plus instables en dernier. Un premier prototype est produit à cette occasion. Il permet d'apprécier l'ergonomie générale de l'application et l'enchaînement des grandes fonctionnalités. Les utilisateurs interviennent également pour donner leur avis sur ce prototype et le valider s'il répond à leurs attentes. Les séances de travail sur les thèmes ont une durée qui varie de 4 à 8 jours. La documentation technique de l'application est l'un des produits de cette phase. Elle est automatiquement générée par l'atelier de génie logiciel.

Au cours de cette phase, la parallélisation du développement peut être envisagé pour des projets de grand envergure (Vickoff, 1999) comme le montre la figure 144.

I.2.4 *Phase de Construction (réalisation, prototypage)*

La phase de *Construction* relève quant à elle d'un processus *bottom-up*. C'est la phase pendant laquelle l'essentiel du code est produit. Elle fusionne l'étape classique de spécification détaillée, de réalisation, de tests unitaires et d'intégration ainsi qu'une partie de tests de recette. Pour assurer cette production, la méthode RAD utilise plusieurs techniques pour réaliser les fonctionnalités demandées par les utilisateurs. Elle s'appuie sur la constitution d'équipes ***SWAT***[101] comprenant 4 à 6 concepteurs/développeurs. La méthode RAD exige une équipe SWAT soit construite avec des concepteurs/développeurs ayant comme qualités compétence, autonomie et démocratie. En outre, ils doivent être complémentaires. La ***validation des fonctionnalités est permanente*** c'est-à-dire que les utilisateurs les définissent lors des séances de travail et les entérinent à la séance suivante. Ce processus de validation permanente est la meilleure garantie pour assurer que les fonctionnalités implémentées répondent bien aux besoins des utilisateurs. En outre, des réunions plénières, appelées ***Focus***, sont l'occasion de présenter l'avancement du projet et de le valider dans toutes ses dimensions. Les focus font suite à l'implémentation par le SWAT de plusieurs fonctionnalités majeures. La méthode RAD définie des ***normes de codage*** afin d'obtenir code uniforme. Elles sont publiées et acceptées par tous. Par ailleurs, la méthode RAD met en œuvre la ***revue de code*** qui consiste à une vérification croisée de la conformité des codes écrits aux normes adoptées. En outre, la technique orientée "planning" du ***jalon zéro-défaut*** -revue de code suivie d'une intégration et d'une validation complète- permet de contrôler l'avancement du projet en matière de visibilité et de qualité entre deux réunions de focus. La méthode RAD préconise que l'application soit maintenue en ***état de livraison permanente*** c'est-à-dire que après chaque focus confirmant un jalon zéro-défaut, l'application est dans un état de livrable même s'il est à fonctionnalités réduites. Le livrable est conservé en l'état afin de pouvoir être présenté à tout moment et utilisé à des fins de tests ou de validation. Bien que l'application ait été structurée en phase de Design, cette première structuration ne suffit pas à garantir son évolutivité ultérieure car en phase de Construction, la conception se poursuit simultanément à la réalisation. Les concepteurs/développeurs doivent donc intégrer de nouveaux concepts, implémenter de nouvelles fonctionnalités en s'appuyant sur des ***techniques de conception en vue de la modification*** ultérieure de l'application. Ces techniques sont mises en œuvre pour permettre une évolution continue de l'application depuis la conception, l'implémentation et jusqu'à la maintenance. Une ***structure de Construction itérative et incrémentation*** doit être privilégiée au cours de cette phase. En début de la phase de Construction, un plan d'évolution du prototype doit être définit. Il segmente l'ensemble des fonctionnalités à réaliser et précise au cours de quel focus elles doivent être réalisées. Au cours de cette phase, une sérialisation du développement peut venir s'ajouter à la parallélisation entamée en phase de Design (cf. figure 144). Certains auteurs parlent de développement semi-itératif[102].

La documentation utilisateur vient compléter la documentation technique de l'application et un site pilote est ouvert pour la validation des différentes évolutions de l'application.

I.2.5 *Phase de Finalisation (recette et déploiement)*

L'objectif de la phase de *Finalisation* est l'homologation du produit final. Elle implique principalement la maîtrise d'ouvrage. Pour ce faire, les tests de recette sont mis en œuvre. L'implication constante des utilisateurs tout au long du projet et la validation des spécifications et des prototypes qu'ils opèrent à chaque stade se traduisent par une simplification et un allégement de l'étape de recette. Cette phase comprend également le déploiement de l'application dans son intégralité sur le site pilote, sa mise en exploitation et le suivi du site afin de pallier les éventuels bogues. Un bilan global du projet est effectué. Il inclut la synthèse des connaissances acquises au cours du projet dans le but de pérenniser les savoir-faire développés. Enfin, l'application finalisée est transférée au service maintenance avec toute la documentation relative au projet, en particulier les documentations technique et utilisateur du produit. Pour finir, elle intègre également la formation des utilisateurs au nouveau produit.

La phase de *Finalisation* conclue un projet développé suivant la méthode RAD.

I.3 Gestion

Le développement d'une application avec comme objectif principal une réalisation rapide est un processus complexe. Il est impossible d'atteindre cet objectif si des d'obstacles bureaucratiques ou politiques viennent

[101] SWAT : Skilled Workers With Advanced Tools.

[102] Précision de Jean-Pierre Vickoff (Vickoff, 2000) : Merise ou SDMS utilisent un cycle en cascade, alors que DSDM ou RUP préconisent un cycle totalement itératif.

encombrer ou perturber ce processus déjà complexe (University of California, 1997b). Aussi, l'équipe de direction du projet doit mettre en œuvre une gestion de "qualité", critère indispensable pour atteindre objectif fixé. En particulier, elle aura à gérer un ***changement de culture***, ***engager et gérer*** le personnel des équipes SWAT, ***motiver les acteurs*** du projet (responsables, utilisateurs, développeurs, etc.) et enfin démontrer, au travers ***d'indicateurs***, que le développement suivant la méthode RAD est rapide, de qualité et rentable.

Les tâches à mener par l'équipe de direction ne sont pas faciles à accomplir et pour y arriver il faut qu'elle-même soit convaincue du bien fondé de la méthode qu'elle compte mettre en œuvre. Aussi, il est essentiel que l'équipe de direction ait totalement intégré les concepts de la méthode RAD et qu'elle y soit "entièrement dévouée" (University of California, 1997b). Le succès du projet en dépend.

Le développement rapide d'une application selon la méthode RAD suppose de mettre en œuvre de nouvelles technologies de développement basées sur la réutilisation de code existant, sur l'utilisation d'atelier de génie logiciel et sur une bonne gestion de la base de connaissances de l'équipe. Un tel changement de mode de travail demande un vrai ***changement de culture*** de travail au sein de la communauté des Techniques de l'Information. Aussi surprenant que cela puisse paraître, c'est une tâche difficile que devra mener l'équipe de direction car les dirigeants et les professionnels de la communauté sont souvent réticents à adopter de nouvelles technologies. L'attitude des développeurs est primordiale pour le bon déroulement du projet. Il va de soi qu'une attitude positive sera bénéfique au projet alors qu'une attitude négative peut perturber fortement le fonctionnement de l'équipe et du projet. Cette dernière se manifeste assez souvent sous la forme d’"astucieux sabotage" de la part des développeurs réticents à abandonner leurs anciennes technologies, leur but étant de démontrer que les nouvelles technologies sont inappropriées. Plusieurs organismes et entreprises ont ainsi obtenu une faible productivité en mettant en œuvre de nouvelles technologies de développement, comme les ateliers de génie logiciel. En fait, ces développeurs sont confortablement installés dans leurs anciennes technologies et se refusent à accroître leurs connaissances et leurs savoir-faire (University of California, 1997b). Aussi, l'équipe de direction, parfaitement informée de la méthode RAD et convaincue de son utilité, devra faire pression pour qu'elle soit appliquée dans son intégralité.

En méthode de conduite de projet RAD, les équipes SWAT sont au cœur du dispositif et d'elles dépend le succès ou non du projet. Aussi, la ***constitution et la gestion des équipes SWAT*** sont un vrai challenge qu'aura à relever l'équipe de direction. Une équipe SWAT doit être construite avec des concepteurs/développeurs ayant comme qualités principales la compétence, l'autonomie et la démocratie. En outre, ils doivent être techniquement complémentaires. L'équipe de direction doit faire en sorte de limiter au maximum le turn-over au sein des équipes car il nuit directement à la productivité du développement. Pour ce faire, elle doit maintenir un haut niveau de motivation de l'équipe et faire attention de ne pas en décourager les membres. Une situation classique et assez fréquente est la gestion des heures supplémentaires. Tout projet développé s'accompagne souvent de périodes de travail très intensif et les développeurs font alors de nombreuses heures supplémentaires. Un bon chef de projet doit être en mesure d'évaluer à quel moment l'utilisation massive d'heures supplémentaires fait "plonger" la productivité. Il doit alors être capable de stopper le développement afin que les équipes se reposent. Un autre exemple de gestion important pour soutenir la motivation au sein des équipes est mettre, à la fin du projet, les équipes SWAT en congés avant d'entreprendre un autre développement intensif. Ces pratiques permettent de diminuer le découragement des développeurs et de les conserver motivés dans leur travail réduisant ainsi le turn-over.

En gestion quotidienne, la tâche la plus importante et la plus délicate qu'aura à résoudre l'équipe de direction est que les développeurs conservent un haut ***niveau de motivation***. Pour ce faire, les responsables doivent, en premier lieu, clairement identifier les objectifs de chacune des équipes SWAT et ensuite ils doivent motiver les développeurs afin d'atteindre les objectifs fixés. Un des moyens pour y arriver est de porter à connaissance le niveau de productivité de chacune des équipes créant ainsi une émulation entre elles. Par ailleurs, les responsables doivent s'assurer que les développeurs sont individuellement motivés et qu'aucun d'entre eux n'est "isolé". Aussi, l'équipe de direction doit faire appel à toutes les techniques de motivation possibles afin d'accroître la cohésion et l'excellence du groupe.

Une bonne gestion de projet suppose de définir ***des indicateurs de performances*** permettant d'évaluer différents critères comme la vitesse de codage, la productivité, la qualité, etc. L'équipe de direction doit donc mettre en place des indicateurs pour mesurer ces critères et les communiquer et les faire admettre aux autres acteurs du projet. Généralement ces indicateurs de performances peuvent être directement transposés en coûts financiers pour le projet. Ces indicateurs sont indispensables à une entreprise ou à une organisation afin d'évaluer le délai et les ressources nécessaires, donc le coût, de futurs projets. Selon leur granularité, ces indicateurs sont aussi utilisés pour évaluer la performance des équipes SWAT et des développeurs. Mis à disposition de l'ensemble des acteurs du projet, ils deviennent un outil de motivation important car chaque développeur essaye d'améliorer ses propres performances. Fréquemment, l'indicateur de performance le plus important est si oui ou non le développement de l'application ou d'un sous-système a été livré dans les délais. D'autres indicateurs plus

fins peuvent être utilisés pour évaluer la productivité (nombre de lignes de code …), la qualité (nombre de bogues, de défauts de conception, de modifications pour améliorer l'ergonomie …), etc.

I.4 Outils

Selon (University of California, 1997b), le développement rapide d'applications, bon marché et de bonne qualité suppose l'automatisation massif de l'ensemble du processus, c'est-à-dire la mise en œuvre d'outils permettant de réaliser la planification du développement, de faire l'analyse du domaine, d'effectuer la conception de l'application mais aussi de produire du code automatiquement. Le but étant bien évidemment que chacun de ces outils exploitent les informations des autres au travers d'une base de connaissances. Ces différentes activités de développement correspondent à des fonctionnalités instanciées dans quatre types d'outils :

- Des langages non procéduraux permettant aux développeurs d'exprimer le résultat sans montrer les détails de développement.
- Des outils permettant un processus itératif du développement en cours, processus qui consiste en une succession de raffinements et tests de l'application.
- Des ateliers de génie logiciel permettant d'effectuer graphiquement la modélisation et la conception de l'application.
- Des générateurs de code permettant de projeter le modèle graphique de nature contemplative dans un langage de programmation qui est lui un modèle productif.

La méthode RAD préconise l'utilisation d'un atelier de génie logiciel intégrateur offrant en son sein ces quatre types de fonctionnalités.

Le formalisme de modélisation n'est pas du tout imposé par la méthode RAD. Toutefois, les auteurs (Vickoff, Site Web non daté ; Web Ntic, 2003) suggèrent l'utilisation soit de la méthode Merise soit l'approche américaine des flux (DFD[103]) soit encore le langage UML.

I.5 En résumé

La méthode de conduite de projet ***Rapid Application Development*** est une méthode qui se positionne comme précurseur des méthodes Agiles. C'est une méthode globale qui s'intéresse à l'ensemble du dispositif de développement : l'équipe, le processus de développement, la gestion du projet et des membres de l'équipe et les outils. La mise en œuvre de cette méthode est un vrai challenge que doit relever l'équipe de direction et en particulier le chef de projet. L'université de Californie (University of California, 1997b) a identifié douze commandements que le chef de projet et l'équipe de direction doivent suivre pour surmonter ce challenge :

- Utiliser une méthode de conduite de projet adaptée à un développement rapide et de qualité.
- S'assurer d'une participation active des acteurs du domaine et des utilisateurs.
- Sélectionner des développeurs doués et de haut niveau décidé à relever le challenge.
- Choisir les développeurs près à travailler en équipe.
- Conserver les bonnes équipes sur plusieurs projets pour forger un esprit d'équipe.
- Proposer aux développeurs les meilleures formations possibles sur la méthodologie RAD et les nouvelles technologies de développement.
- Fixer clairement des objectifs et définir les indicateurs de productivité des équipes associés permettant une classification régulière de leurs performances.
- Structurer le travail afin que chaque développeur réalise un travail intéressant et créatif.
- Rendre le processus de développement aussi agréable que possible.
- Utiliser des techniques de motivations.
- Faire attention d'éviter le découragement des développeurs.
- Mettre en œuvre les plus puissants outils de gestion et de développement existants.

[103] DFD : Data Flow Design.

I.6 Remarques et commentaires

Jean-Pierre Vickoff (Vickoff, 2000) attire l'attention du lecteur sur l'aspect ***illusoire de vouloir développer un projet stratégique avec des ressources à temps partiel ou sous contrainte de temps***.

Cette méthode a selon Jean-Louis Bénard (Bénard, 2002a) permis de formaliser les rôles de maîtrise d'ouvrage et de maîtrise d'œuvre et d'en préciser l'articulation fonctionnelle via le Groupe d'animation. C'est aussi l'une des premières méthodes à avoir intégré le principe du développement itératif (semi-itératif selon certains auteurs) des applications, avec production d'un prototype validé par les acteurs à chaque itération. Elle donnait ainsi la parole aux acteurs du domaine et tenait compte de leurs observations en cours de développement se démarquant ainsi des méthodes traditionnelles. Elle introduisait un début de flexibilité au cours du développement. Toutefois, le développement itératif d'une méthode RAD n'est pas poussé au paroxysme comme cela est le cas en eXtreme Programming.

Jean-Louis Bénard rappelle aussi qu'une méthode de conduite de projet RAD doit être rapide pour être efficace, ce qui se traduit par l'utilisation de techniques de modélisation rigoureuses mais simplifiées. Aucune recommandation n'est faite quant à utilisation d'un langage de modélisation en particulier -Merise, UML ou tout autre- par contre il est conseillé de limiter au strict nécessaire le nombre de diagrammes utilisés.

(Muller *et al.*, 2000) soulignent que le défaut de cette méthode réside dans le fait qu'elle incite les développeurs à travailler *vite et mal* au détriment des développements *vite et bien* car ils sont sous l'influence des utilisateurs dont le seul but est la rapidité de réalisation. Par exemple, il arrive souvent qu'avec une méthode RAD le code des interfaces utilisateurs soit noyé dans celui de l'application elle-même. Selon Pierre-Alain Muller, c'est une conséquence directe de l'absence de structuration, d'abstractions en couche, etc. Ce commentaire de Pierre-Alain Muller est en contradiction avec la présentation de la méthode par Jean-Pierre Vickoff (Vickoff, Site Web non daté) qui souligne d'une part, que cette controverse est injustifiée car le premier objectif de la méthode RAD est la *qualité* du développement et que, d'autre part, les partisans de cette méthode considèrent qu'elle *n'est pas une voie royale ou laxiste* et que *son efficacité impose formation et rigueur*. Jean-Pierre Vickoff complète ces propos en rappelant que la méthode RAD répond aux exigences des 6 niveaux du CMM[104][105], procédure d'évaluation des processus de développement d'applications (Muller *et al.*, 2000).

II LA METHODE DE CONDUITE DE PROJET PROCESSUS UNIFIE (UP)

II.1 Historique

Cette méthode est l'aboutissement de pratiques de développements logiciels qui s'étalent sur trois décennies (Jacobson *et al.*, 1999). Au cours du temps, elle a évoluée, alimentée par de nombreuses réflexions, de pratiques et d'expériences vécues. La figure 145 retrace la genèse de cette méthode.

Les idées qui donneront naissance à la méthode de conduite Processus Unifié datent de la fin des années soixante alors qu'Ivar Jacobson travaillé chez Ericsson (Jacobson, 2003). En 1987, Ivar Jacobson quitte la société Ericsson et fonde sa propre société, Objectory AB à Stockholm, afin de mettre en œuvre ses idées. Il développe plusieurs versions (1.0 - 3.8) du produit Objectory[106] entre 1987 et 1995. En 1995, Rational Software Corporation fait l'acquisition d'Objectory AB et Ivar Jacobson rejoint ainsi Grady Booch et James Rumbaugh. Les trois *amigos* réunis donneront naissance au texte fondateur d'UML en juin 1996, texte qui sera soumis à l'OMG en janvier 1997 et adopté en novembre 1997 (cf. Annexe I-II).

Avant le rachat d'Objectory AB, la société Rational Software Corporation avait développé sa propre méthode intitulée *The Rational Approach*. Après le rachat, les deux méthodes ont été fusionnées car elles présentaient chacune des complémentarités fortes. *Rational Objectory Process* est le fruit de cette fusion. Entre 1995 et

[104] Le CMM (Capability Maturity Model) a été défini par Watts S. Humphrey de l'université de Carnegie Mellon fin des années 80. Cette procédure d'évaluation du processus de développement a été normalisée et a donné lieu à la série des cinq normes ISO 15504 connues sous le titre général *Évaluation des procédés*. Chaque partie a son propre titre : celui de la partie 1 est *Concepts et vocabulaire* (ISO, 2004a), celui de la partie 2 est *Exécution d'une évaluation* (ISO, 2003c), celui de la partie 3 est *Réalisation d'une évaluation* (ISO, 2004b), celui de la partie 4 est *Conseils sur l'utilisation pour l'amélioration de processus et la détermination de capacité de processus* (ISO, 2004c) et enfin celui de la partie 5 est *Un modèle d'évaluation et guide des indicateurs* (ISO, 1999). Cette dernière est en cours de révision et son titre devrait changer prochainement (titre provisoire : *Un modèle d'évaluation des procédés exemplaire*).

[105] Depuis la rédaction de certains articles ou documents cités en référence, les groupes de travail de l'ISO ont continué à faire évoluer cette norme. Ils ont modifié d'une part, sa structure même en recomposant les 9 parties initiales et ne les ramenant à 5 et, d'autre part, le nombre de niveaux du CMM en augmentant de 5 à 6.

[106] Objectory : contraction de Object Factory.

1998, Rational Software Corporation achète ou fusionne avec plusieurs autres sociétés qui chacune apporte son expertise en processus de développement.

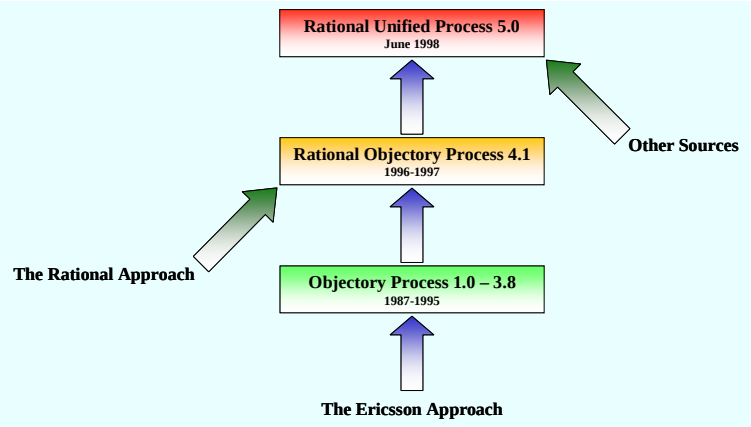


Figure 145 Genèse de la méthode Processus Unifié (Jacobson *et al.*, 1999).

Philippe Kruchten, directeur de cette activité chez Rational Software Corporation, et son équipe mèneront à bien l'intégration de toutes ces méthodes et les unifieront en une seule qui sera baptisée *Rational Unified Process*® (*RUP*®). Cette méthode fut implémentée dans un outil du même nom lequel sera commercialisé en juin 1998. Il accompagnera la sortie de ce produit d'un ouvrage, intitulé *The Rational Unified Process* (Kruchten, 1999)[107], devenu un des "best-sellers" en génie logiciel avec un volume de vente de 170 000 exemplaires et traduit en 10 langages(Kruchten, Site Web non daté). En 2003, la société IBM a racheté Rational Software Corporation qui est devenu la cinquième marque du groupe *IBM software brands*.

La terminologie *Processus Unifié* est généralement utilisée par les organismes ou entreprises qui désirent appliquer les principes de la méthode sans mettre en œuvre les outils proposés par la société Rational Software Corporation (Fowler, 2004b).

La méthode Processus Unifié puise donc son origine dans une longue expérience de pratiques concernant le génie logiciel appliqué à une multitude de domaines tels que les télécommunications, l'aéronautique, la défense, les transports, etc. Elle cristallise les enseignements tirés de la conception d'applications très diverses faisant d'elle ***un recueil des meilleures pratiques récentes de développement logiciel***. C'est devenu une méthode phare.

Par sa genèse, la méthode Processus Unifié est associée au langage UML mais elle en est totalement dissociée (Fowler, 2004b ; Millan *et al.*, 2004). Les créateurs d'UML ont délibérément omis de définir une méthode pour mener à bien le développement d'une application (Fayet, 2002) afin de laisser chaque concepteur libre de la démarche la mieux adaptée à son environnement professionnel. Ivar Jacobson a formalisé les différents éléments intervenant dans un projet de développement d'une application informatique. Ce modèle est connu sous le nom *4 Ps : Personnel (Acteur), Projet, Produit et Processus de développement d'une application*[108] (cf. figure 146).

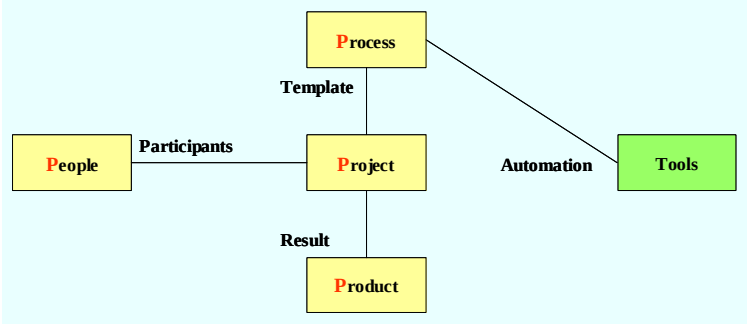


Figure 146 Les 4 Ps du développement d'une application (Jacobson *et al.*, 1999).

[107] La version française est intitulée *Introduction au Rational Unified Process* (Kruchten, 2000).
[108] The Four Ps : People, Project, Product, and Process in Software Development.

Ce modèle exprime que le ***résultat*** d'un ***projet*** est un ***produit*** qui nécessite des ***participants***[109] pour être mené à bien. La réalisation du projet s'effectue suivant un ***guide***[110] qui définit, organise et explique les étapes successives du ***processus*** de développement. Pour mettre en œuvre le processus de développement, il faut des ***outils*** de planification du projet, d'expression des besoins, de modélisation, etc. qui automatisent certaines activités.

Sa genèse explique pourquoi la méthode Processus Unifié est classée parmi les *méthodes traditionnelles* de conduite de projet, bien qu'ayant, par certains aspects, les caractéristiques des méthodes agiles (Bénard, 2001).

II.2 Équipe

Selon Ivar Jacobson, les acteurs d'un projet de développement informatique viennent d'horizons différents et chacun d'eux ont un apport, des attentes et des fonctions différentes. Les commanditaires vont financer le projet car ils ont un besoin à satisfaire, les analystes et les concepteurs vont capturer les concepts du système étudié et concevoir l'architecture de l'application ou du système d'information, les programmeurs vont quant à eux réaliser le développement proprement dit et testeront l'application et enfin les utilisateurs bénéficieront des avantages de la nouvelle application. Aussi, au vu de la diversité d'acteurs impliqués dans un projet informatique, Ivar Jacobson considère que le ***développement d'une application doit être centré sur les acteurs*** du projet. Dit autrement, la méthode de développement doit être de nature participative afin que le produit final corresponde aux attentes de l'ensemble des acteurs.

Dans son analyse du fonctionnement de l'équipe, Ivar Jacobson fait remarquer que la structuration du projet et la gestion de l'équipe ont une incidence directe sur les acteurs du projet et en particulier sur leur implication et leur comportement. Parmi cette diversité d'acteurs, les programmeurs constituent une population particulière car ce sont eux qui in fine vont réaliser l'application. Leur adhésion au projet est donc cruciale. Ivar Jacobson relève un certain nombre d'impressions ou de faits qui peuvent soit favoriser l'adhésion des acteurs en général soit la rejeter.

La ***faisabilité du projet*** est la première impression. La plupart des acteurs n'aiment pas participer à un projet qu'ils considèrent irréalisable. Selon Ivar Jacobson, l'adoption d'un processus de développement itératif met rapidement en exergue cette faiblesse et le projet peut alors être réévalué ou abandonné.

De la même manière, si les acteurs perçoivent que la ***gestion du risque*** n'a pas été faite correctement et que l'analyse n'a pas été suffisamment poussée pour les réduire, ils vont être *mal dans leur peau* et leur implication sera moins bonne. Une recherche des risques majeurs dans les phases préliminaires amoindrit ce problème.

Dans le but d'une meilleure efficacité et une meilleure productivité, l'***organisation de l'équipe*** passe souvent par une structuration en petits groupes de six à huit personnes. Une telle structuration vient renforcer le sentiment de responsabilité des développeurs qui souvent sont attachés aux développements dont ils ont la charge. Certains vont jusqu'à s'identifier aux développements. Par contre, l'organisation en petits groupes peut conduire l'architecte à découper l'application en sous-systèmes dans un but de faciliter la répartition de la charge de travail entre les groupes. Un bon architecte devra alors être plus précis dans la définition des interfaces entre sous-systèmes et composants répartis entre ces groupes de développement.

La ***planification du projet*** peut agir également sur le moral et le comportement des acteurs. Si ces derniers estiment que la programmation est utopique, leur moral va plonger. Il est donc essentiel d'éviter cela en leur faisant percevoir exactement ce que l'application doit faire, en leur communiquant une planification raisonnable et en leur accordant le temps nécessaire pour réaliser leurs tâches. Les acteurs aiment savoir ce qu'ils font. Pour ce faire, il est crucial que l'architecture fournisse une description détaillée à chacun des acteurs afin qu'ils aient une ***compréhension du projet*** dans sa globalité et une vision de leur travail dans le contexte global.

Enfin, les acteurs doivent avoir le ***sentiment du travail accompli***. Il est possible de transmettre ce sentiment via le processus de développement itératif préconisé par la méthode. Des feedbacks et des validations à chaque itération augmente la qualité du développement et in fine le sentiment du travail accompli.

En méthode de conduite Processus Unifié, les utilisateurs constituent une autre population sensible et incontournable d'acteurs. En effet, ce sont eux qui vont fournir une part importante des spécifications de l'application qui permettront de satisfaire leurs besoins. En outre, ce sont eux aussi qui auront la charge de valider les produits ou les prototypes au terme de chaque itération.

[109] Acteur dans notre contexte.
[110] Méthode dans notre contexte.

II.3 Méthode et cycle de développement

II.3.1 *Les Principes de la méthode*

Pour qu'un processus de développement soit qualifié de Processus Unifié, il faut qu'il prenne en compte quatre grands principes : il doit être *conduit par les cas d'utilisations, centré sur l'architecture, itératif et incrémental* et *piloté par les risques* de façon systématique (Bénard, 2002b ; Jacobson *et al.*, 1999 ; Larman, 2002a ; Muller *et al.*, 2000 ; Roques, 2002 ; Roques *et al.*, 2002).

II.3.1.1 Conduit par les cas d'utilisation

Le but de tout projet informatique est de reproduire le comportement d'un système afin de satisfaire un besoin ou une exigence. Aussi, il faut d'une part, mobiliser la connaissance qu'ont les acteurs d'un système à travers la description des concepts du domaine (aspects structurels) et les comportements de ces concepts (aspects dynamiques) et, d'autre part, de bien d'identifier les besoins ou les exigences des utilisateurs (aspects fonctionnels).

Pour ce faire, il est indispensable de mettre en œuvre une technique de capture de cette connaissance et de ces besoins et exigences. Pour cela, le Processus Unifié propose d'utiliser le concept des cas d'utilisation et celui des scénarios (Booch *et al.*, 2000).

Les cas d'utilisation ont été introduits par Ivar Jacobson en 1987(Jacobson, 2003). Il en donnait la définition suivante : *un cas d'utilisation est une séquence de transactions accomplies entre un utilisateur et un système au cours d'un dialogue*. Ce concept novateur a tellement été plébiscité par les modélisateurs qu'Ivar Jacobson a été lui-même surpris du son succès. Cette définition de 1987 reste compatible avec l'instrumentation qui a été faite de ce concept dans le langage UML (Jacobson, 2003).

Les cas d'utilisation capturent les comportements du futur système, ils en montrent son fonctionnement, ils identifient *qui* et *quoi* interfèrent avec le système et ce que doit faire le système, ils vérifient que toutes les exigences des utilisateurs ont bien été identifiées. C'est un instrument de planification.

D'après Ivar Jacobson, il y a plusieurs raisons pour lesquelles le concept des cas utilisations est bon et est très apprécié et universellement adopté mais deux d'entre elles émergent du lot :

- ⇨ La première vient de sa facilité et de sa capacité à capturer, dans un cadre très formel, les exigences fonctionnelles d'un système avec une approche très intuitive. Les utilisateurs et les concepteurs d'une application n'ont pas à apprendre une notation et des mécanismes très complexes. Leur lecture et leur compréhension sont donc aisées. Leur évolution en est facilitée et ne demande pas d'effort particulier.
- ⇨ La seconde résulte du fait qu'il permet de guider tout le processus de développement d'un système : analyse, conception, implémentation et tests. Ils fournissent un cadre conceptuel pour assurer la cohérence du développement depuis les exigences jusqu'aux tests (Booch *et al.*, 2000). Les concepts structurels sont appréhendés lors de la description des cas d'utilisation et constituent généralement les classes du système. La rédaction de cette description est souvent l'occasion de préciser les Interfaces Homme/Machine. La définition de ces dernières est un moyen de sérier les tâches accomplies par les acteurs du domaine. Une fois l'intégration effectuée en fin d'itération, les tests sont déployés pour vérifier si les différentes tâches décrites par le cas d'utilisation ont été réalisées et si elles exécutent correctement, validant implicitement le cas d'utilisation lui-même. Ils permettent aussi au chef de projet d'évaluer le volume de travail de chacun des tâches donc du cas d'utilisation dans sa globalité. Pour lui c'est également un outil de gestion puisqu'il va attribuer la réalisation d'un cas d'utilisation ou d'une des tâches à une équipe ou à un développeur qui en devient responsable.

Les cas d'utilisation interviennent donc tout au long du processus de développement et sont au centre du processus. Ils ont été conçus pour tenir compte des besoins des utilisateurs donc, une fois implémenté, un cas d'utilisation doit satisfaire a priori l'utilisateur. Dans le cas contraire, ils ont été mal définis.

En outre, selon (Jacobson *et al.*, 1999), la traçabilité entre les cas d'utilisation et les autres éléments de la modélisation facilitent le maintient en cohérence du système surtout si les exigences évoluent.

Tous ces éléments font que les ***cas d'utilisation pilotent*** le développement car ils servent de références tout au long du processus.

II.3.1.2 Développement centré sur l'architecture

Le développement d'applications complexes est un vrai challenge car il implique un nombre élevé de connaissances de domaines divers. La mise en vrac de ces connaissances dans un modèle ou dans du code pose

d'énormes problèmes au niveau de la compréhension du système et de son fonctionnement, compréhension qui aura une incidence directe sur l'évolution ultérieure de l'application. S'il est difficile de comprendre son fonctionnement interne, il y a peu de chance qu'il soit possible de modifier une fonctionnalité d'une application ou d'en ajouter de nouvelles. Ce constat a conduit les concepteurs à construire des architectures afin de structurer leur application. Cette structuration est nécessaire pour les quatre raisons suivantes (Jacobson *et al.*, 1999) :

- La ***compréhension du système*** bien entendu : pour les systèmes complexes, la connaissance des acteurs doit être structurée en sous-systèmes (Booch *et al.*, 2000 ; Roques, 2002) car ils sont difficiles à appréhender dans leur globalité par un seul acteur. Souvent, un acteur n'a qu'une connaissance partielle du système et c'est l'assemblage de ces connaissances partielles qui permet au concepteur de reconstituer la totalité du système.
- L'***organisation du développement*** : une fois la structuration métier étant réalisée assurant ainsi une bonne compréhension du système, il n'est pas rare que l'architecte organise l'application suivant d'autres critères. Le regroupement selon des concepts informatiques indispensables à la réalisation de l'application est vraisemblablement le plus aisé à mettre en œuvre dans un deuxième temps. Ces structurations étant établies, les briques de logiciel à réaliser restent beaucoup trop conséquentes dans le cadre de gros projet impliquant plusieurs dizaines de développeurs voire plusieurs centaines. L'architecte doit alors décomposer chacune des briques en sous-systèmes afin de distribuer le travail par équipe et par développeur. Cette structuration oblige l'architecte à détailler, à tous les niveaux d'organisation, les interfaces entre sous-systèmes pour que, lors de l'intégration, l'ensemble fonctionne correctement.
- La ***réutilisation de composants*** existants : le but de toute organisation ou de toute entreprise est de réaliser un produit dans les meilleurs délais et à moindre coût. Dans ce contexte, tous les développements existants seront réutilisés dans la mesure du possible. Si une application est répartie sur plusieurs machines et que ces entités doivent communiquer via Internet, le responsable du projet va imposer l'utilisation des composants de bases de la communication Internet pour des questions d'économie mais aussi pour la stabilité du code existant. En effet, redévelopper des composants logiciels sous-entend de les fiabiliser avant de les mettre en service, il vaut donc mieux utiliser l'existant. La structuration du développement va permettre d'identifier les composants de l'application qui ont déjà été développés à d'autres occasions et de décider de l'opportunité ou non de les utiliser ou de les acquérir.
- L'***évolution du système*** : s'il y a un point qui est certain c'est que l'application va évoluer (Jacobson *et al.*, 1999). Elle évolue naturellement au cours du développement jusqu'à sa stabilisation. Ensuite, les concepteurs peuvent être amenés à y ré-intervenir pour des raisons techniques, par exemple l'évolution du système d'exploitation, mais aussi parce que le client souhaite introduire de nouvelles fonctionnalités. Dans un cas comme dans l'autre, il est primordial que l'application soit facilement modifiable. Dans ce contexte, une application bien structurée dont l'organisation est compréhensible sera plus aisée à modifier qu'une application non structurée.

La structuration en sous-systèmes facilite la capitalisation des connaissances mais aussi, la maintenance et l'évolution de l'application. Elle fournit l'architecture du système qui doit être *simple* et très *intuitive* (Muller *et al.*, 2000) afin que les acteurs du système puissent se l'approprier facilement lors des phases de validation. *Elle est conçue avec et pour la réalisation* (Muller *et al.*, 2000).

Cette structuration ne doit pas être une simple vue de l'esprit formalisée par une *description du système sous forme graphique ou textuelle mais elle doit être* effective et *matérialisée* via un modèle dans un atelier de génie logiciel (Roques, 2002 ; Roques *et al.*, 2002).

Le concept des cas d'utilisation est un des outils privilégiés d'assistance à cette structuration. La terminologie ***développement centré sur l'architecture*** est le qualificatif consacré à ce travail de structuration que doit impérativement effectuer tout concepteur relevant de la méthode Processus Unifié.

II.3.1.3 Développement itératif et incrémental

Le processus itératif et incrémental du développement reste sans aucun doute l'exigence la plus importante (Larman, 2002a ; Roques, 2002). C'est principalement ce principe qui confère au Processus Unifié certains aspects des méthodes de conduite de projet agiles.

Le qualificatif d'***itératif*** s'applique à *la gestion d'une succession de* versions exécutables (Booch *et al.*, 2000 ; Muller *et al.*, 2000) alors que celui d'***incrémental*** porte sur l'évolution temporelle de l'application, évolution qui est obtenue par ajout de nouveaux concepts ou de nouvelles spécifications formant le nouvel incrément. Un incrément est donc la différence entre l'état du modèle ou de l'application en début d'itération et en fin d'itération (Jacobson *et al.*, 1999 ; Roques *et al.*, 2002). Tout naturellement, ***le modèle validé à l'itération N devient le modèle d'entrée de l'itération N+1. Un processus itératif induit inévitablement une évolution incrémentale du***

développement de l'application, concepts et/ou architecture. Ivar Jacobson exprime autrement ce constat en écrivant : *le résultat d'une itération est un incrément* (Jacobson *et al.*, 1999).

La méthode de conduite de projet Processus Unifié prévoit que le développement d'une application soit découpé en itérations de courte durée -2 à 4 semaines- *en réaction aux méthodes dites "merisiennes" (cycle en V, etc.) qui préconisent des cycles de développement long* (Bénard, 2001). À chaque itération, les spécifications sont validées par les acteurs (Roques *et al.*, 2002) et un sous-système exécutable est produit. Ce processus itératif de développement est le meilleur garant d'une part, pour produire un exécutable conforme aux spécifications et aux exigences des acteurs améliorant de fait la qualité du produit final et, d'autre part, pour permettre de suivre *l'avancement du projet* (Bénard, 2002b), le *contrôle des coûts et des délais* (Roques *et al.*, 2002). Sur ce point, Martin Fowler a une position beaucoup plus tranchée en déclarant ***Ne faites appel au développement itératif que pour les projets que vous voulez voir aboutir*** (Larman, 2002a).

Le processus de développement itératif et incrémentale facilite la capitalisation des connaissances du domaine parce qu'il est toujours possible d'ajouter des concepts nouveaux ou bien oubliés par un acteur lors d'une itération précédente. D'autre part, ce type de processus permet de "mixer" les connaissances d'acteurs ayant des points de vue différents sur le même concept (Booch *et al.*, 2000). Il permet aussi la capitalisation *des enseignements des cycles précédents* (Bénard, 2002b).

II.3.1.4 Piloté par les risques

Les risques sont peuvent mettre en difficulté un projet voire le faire échouer (Jacobson *et al.*, 1999). Ce dernier distingue les ***risques non techniques*** des ***risques techniques***.

Les ***risques non techniques*** sont essentiellement liés à la gestion du projet. Souvent, ils sont décelés par les responsables du projet lorsqu'ils s'aperçoivent que le projet dérive par rapport au planning initial. Voici quelques exemples de cette typologie de risques :

- ⇨ Absence, dans l'organisme ou l'entreprise en charge de la réalisation, de compétence dans des domaines ou des technologies clés du projet.
- ⇨ Décision d'implémenter certaines parties de l'application avec un nouveau langage de programmation non maîtriser par l'organisme ou l'entreprise.
- ⇨ Accepter un projet avec des délais souhaités/imposés par le client difficiles à tenir, etc.

Mais en amont des risques non techniques évoqués par Ivar Jacobson, il existe des typologies de risques majeurs dont le chef de projet doit se soucier : ces sont les ***risques financiers*** et les ***risques commerciaux*** (Muller *et al.*, 2000).

Concernant les ***risques financiers***, la question fondamentale est de s'assurer que le commanditaire de l'application a les moyens financiers pour mener à terme la réalisation de l'application. De nombreux projets sont abandonnés dès la phase de préétude car le coût est disproportionné par rapport aux bénéfices escomptés. Mais il arrive aussi que les moyens financiers soient étalés sur une longue période et qu'entre temps les objectifs changent ou deviennent obsolètes rendant caduque le projet lui-même.

Les ***risques commerciaux*** ne doivent pas être négligés lorsque l'application est destinée à être commercialisée. Ils sont liés à la situation du marché dans lequel va être introduit le produit. Le danger est bien entendu une réactivité forte de la concurrence et qu'elle monopolise le marché avant la sortie du produit. Dans ce contexte, il peut être vital pour le commanditaire de réaliser une version minimale du produit pour l'introduire rapidement et commencer à occuper le marché. C'est un problème de stratégie commerciale qui aura des conséquences directes sur le développement et architecture de l'application finale car, pendant le développement, l'application minimale doit pouvoir être suivie et déboguée. Bien évidemment, cette application minimale doit être un produit de qualité avec très peu de défauts et de dysfonctionnements afin de ne pas compromettre le succès commercial de l'application finale.

Même si la liste n'est pas exhaustive, Ivar Jacobson énumère quatre grandes typologies de ***risques techniques*** liés au développement proprement dit de l'application :

- ⇨ Ceux induits par l'***adoption de nouvelles technologies*** : nous pouvons citer comme exemple les processus répartis sur différents nœuds qui vont poser des difficultés de synchronisation, difficultés qui n'existeraient pas si le processus s'exécutait sur un seul nœud.
- ⇨ Ceux ***liés à l'architecture*** : D'après Ivar Jacobson, ces risquent sont tellement importants que la méthode Processus Unifié a été conçue pour les prendre en compte en facilitant l'adaptation des architectures qui vont pouvoir être ajusté itération après itération et devenir ainsi plus robuste. La robustesse de l'architecture présente un autre avantage et non des moindres, c'est de repérer les patrons de conception,

les composants réutilisables et les frameworks nécessaires à la réalisation de l'application. Leur identification permet ensuite de se poser la question s'il faut les réaliser ou s'il faut les acheter. L'achat n'est pas exempte de problèmes. En effet, il peut arriver qu'un composant ou un framework ait été réalisé dans un contexte particulier et qu'il ne soit pas suffisamment générique pour être appliqué au nouveau projet.

- Ceux relatifs à la ***construction d'un bon système*** : ce risque met en relief l'importance de discerner les exigences aussi bien fonctionnelles que non fonctionnelles de l'application (Jacobson *et al.*, 1999). Le plus délicat pour les concepteurs du système consiste à identifier et à préciser ces fonctionnalités majeures rapidement afin de les implémenter très tôt permettant ainsi de sérier en cascade les nouvelles difficultés s'il y en a.
- Ceux découlant des ***performances du produit final*** : dans certains contextes le temps d'exécution une fonctionnalité est une variable importante. Par exemple, les applications temps réel imposent souvent que les traitements soient réalisés dans un délai fixé à l'avance. C'est une contrainte qu'il faudra donc vérifier. Dans le même ordre d'idée, un nombre de 10 000 connexions par heure à un site Internet impliquent des contraintes fonctionnelles d'un niveau bien supérieur à celles d'un site qui aurait une centaine de connexions par heure. Ce type de contrainte entraîne des choix de structure de répartition orientant la modélisation vers un type de solutions.

L'identification précoce des risques est largement tributaire de la compétence non seulement de l'analyste mais aussi de celle de tous les acteurs -utilisateurs et développeurs- impliqués dans le projet. Des acteurs expérimentés seront capables de déceler très tôt les risques liés au développement d'une application. Ces problèmes sont listés par tous les acteurs et discutés au cours de réunions de travail consacrées au risque. Le but de ces séances n'est pas de résoudre le problème mais de bien les identifier et de les classer par ordre de priorité.

La plupart de ces risques potentiels ont été estimés et des solutions, techniques ou de gestion, ont été réfléchies pour chacun d'eux. Par exemple, certains risques peuvent être évités simplement en modifiant légèrement des exigences sans pour autant bouleverser le projet. D'autres peuvent être traités en organisant de façon judicieuse les activités au cours d'une itération (Jacobson *et al.*, 1999 ; Roques, 2002).

Parmi l'ensemble des solutions, il en est une qui réduit de façon conséquente le niveau de risque qu'il soit d'origine technique ou de gestion : il s'agit du processus itératif. En effet, l'intégration dans l'application de l'incrément produit pendant l'itération va mettre en exergue les problèmes techniques et les bogues du code réalisé. Avant de poursuivre le développement, l'équipe devra résoudre ces problèmes et apporter une solution aux bogues sans quoi le projet entrera en régression. Il serait suicidaire de faire autrement. Des intégrations fréquentes stabilisent l'architecture et les composants rapidement réduisant ainsi les risques pour les itérations ultérieures. La figure 147 montre les évolutions du niveau de risque d'un cycle itératif et d'un cycle en cascade.

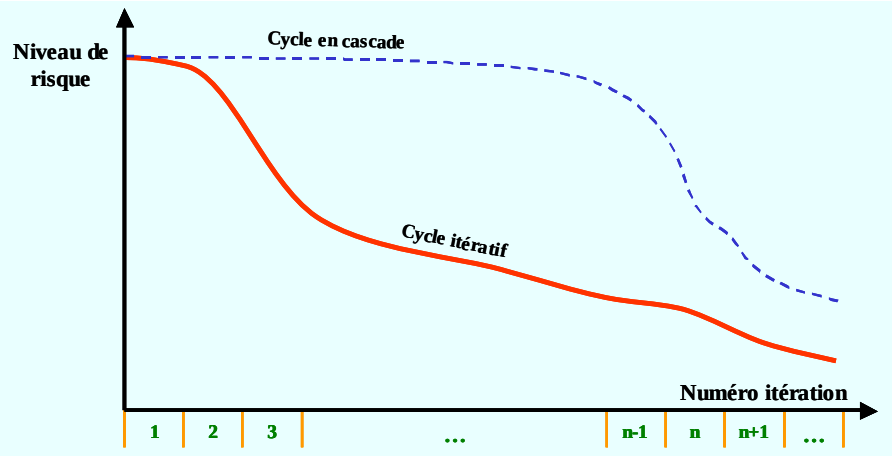


Figure 147 Courbe d'évolution des risques au cours des itérations (Jacobson *et al.*, 1999 ; Muller *et al.*, 2000).

Au niveau gestion, le processus itératif va aussi mettre en exergue la dérive du planning puisque, en début de chaque itération, une replanification est effectuée impliquant forcément un bilan -travaux et délais- de l'itération précédente. Il en résulte une détection précoce de la dérive du projet permettant de prendre les mesures adéquates : utilisation d'heures supplémentaires, embauche de personnel, réduction de la portée du projet, etc.

Le ***pilotage par les risques*** d'un projet permet de prendre en compte des problèmes très tôt voire à leur origine et de les traiter par anticipation.

II.3.2 *Le cycle de développement*

II.3.2.1 Généralités sur le cycle de développement

Le cycle de développement d'une application suivant une méthode de conduite de projet Processus Unifié comprend quatre phases : l'***inception***, l'***élaboration***, la ***construction*** et la ***transition*** (Bénard, 2002b ; Jacobson *et al.*, 1999 ; Kruchten, 1999 ; Larman, 2002a). Le concepteur d'une application met en œuvre tout au long de ces quatre phases des *activités*[111] (Bénard, 2002b ; Muller *et al.*, 2000) très diverses comme l'*étude d'opportunité*, l'*expression des besoins*, l'*analyse*, la *conception*, l'*implémentation*, les *tests* et le *déploiement* (cf. figure 148-Axe vertical).

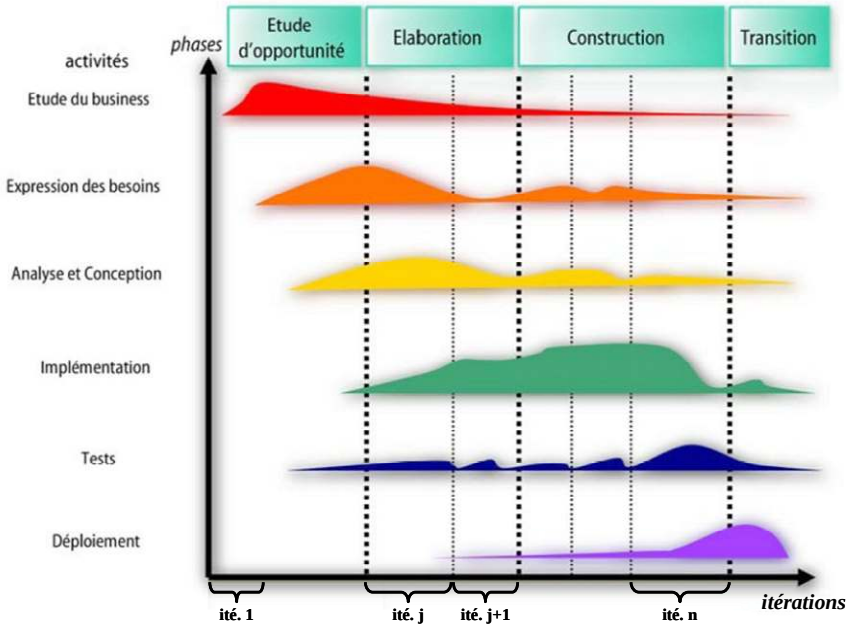


Figure 148 Activités de la méthode Processus Unifié et intensité de production (Booch *et al.*, 2000 ; Jacobson *et al.*, 1999 ; Kruchten, 1999).

La réalisation des différentes activités suppose un effort de production dont intensité varie suivant la phase de développement mais aussi au cours de l'itération. La figure 148 montre un exemple. Les courbes relatives à chacune des activités montrent l'effort à déployer, itération après itération, tout au long d'un processus de développement. Ce graphique est un exemple qu'il faut adapter au contexte et aux besoins de chaque projet de développement.

Étant donné que les compétences mobilisées pour la réalisation d'une activité sont souvent spécifiques aux tâches à accomplir, la représentation suivant la figure 148 de l'effort de production constitue un outil, pour le responsable du projet, de gestion des ressources humaines et des compétences requises, itération après itération, tout au long du projet.

L'analyse de cette figure montre aussi que l'étude d'opportunité a lieu en début de projet, le déploiement intervient principalement en fin de projet ce qui est naturel et que les tests sont répartis tout au long des phases avec une intensité majeure en fin d'itération.

[111] Certains auteurs (Larman, 2002a ; Roques, 2002) préfèrent utiliser le terme de discipline. Auparavant, le terme *Workflow* était utilisé (Booch *et al.*, 2000) mais il a été abandonné en 2000 et remplacé par le terme *discipline* suite aux travaux de standardisation de l'OMG (Larman, 2002a)

II.3.2.2 Description d'une itération type et du processus itératif

Qu'elle que soit la phase, les activités d'une itération sont toujours les mêmes et interviennent toujours suivant la même séquence (cf. figure 149). Le cœur d'itération est constitué par une activité d'expression des besoins, suivie des activités d'analyse et de conception pour finir par les activités l'implémentation des nouveaux concepts et de tests.

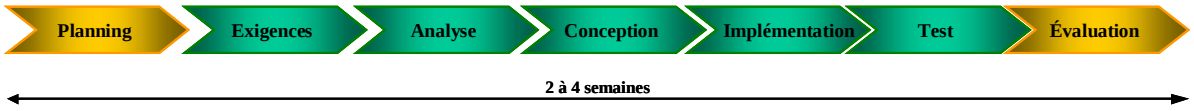


Figure 149 Activités intervenants dans une itération type (Jacobson *et al.*, 1999 ; Kruchten, 1999).

Ces cinq activités du cœur d'itération sont précédées par une activité de planification de l'itération et suivies d'une évaluation du travail effectué. L'activité de planification va détailler le travail à faire au cours de l'itération et l'équipe va examiner en particulier comment les risques résiduels peuvent affecter les autres itérations. L'activité d'évaluation en fin d'itération est essentiellement focalisée d'une part, sur l'analyse des nouvelles exigences décelées en cours d'itération et, d'autre part, sur l'incidence des développements de l'itération sur les anciennes exigences.

La méthode de conduite de projet Processus Unifié préconise une durée d'itération de 2 à 4 semaines. Une durée courte de 2 semaines diminuera les risques, réduira le nombre de défauts, etc. alors que une durée longue permettra de construire des incréments plus conséquent. La durée d'itération peut être utilisée comme variable de gestion des risques. En effet, si une équipe démarre un projet dans un nouveau domaine les risques sont plus élevés que si elle avait à développer une application pour un domaine connu. Dans le premier cas, un cycle court de 2 semaines est conseillé car il permettra de suivre plus finement l'évolution de l'application alors que, dans le second cas, un cycle de 4 semaines peut être accepté car il est fort probable que des composants développés au cours de projets précédents puissent être mobilisés. ***Par contre une fois fixée, la durée doit être la même pour toutes les itérations***. C'est une exigence de la méthode.

Le choix entre les deux extrêmes va dépendre de la sensibilité du chef de projet, de la disponibilité des acteurs à participer à l'analyse et à la définition des exigences mais aussi à la validation des produits de chaque itération.

Au cours d'une itération, le développement peut être vu comme un miniprojet (Jacobson *et al.*, 1999 ; Muller *et al.*, 2000) constituant une nouvelle version de l'application à usage interne. Cette nouvelle version est plus complète et plus robustesse que la précédente.

Comme le montre la figure 148, les itérations se succèdent dans le temps les unes après les autres pour réaliser les quatre phases du cycle de développement de l'application (inception, élaboration, construction, transition). Ce processus itératif du développement peut se représenter par le schéma de la figure 150.

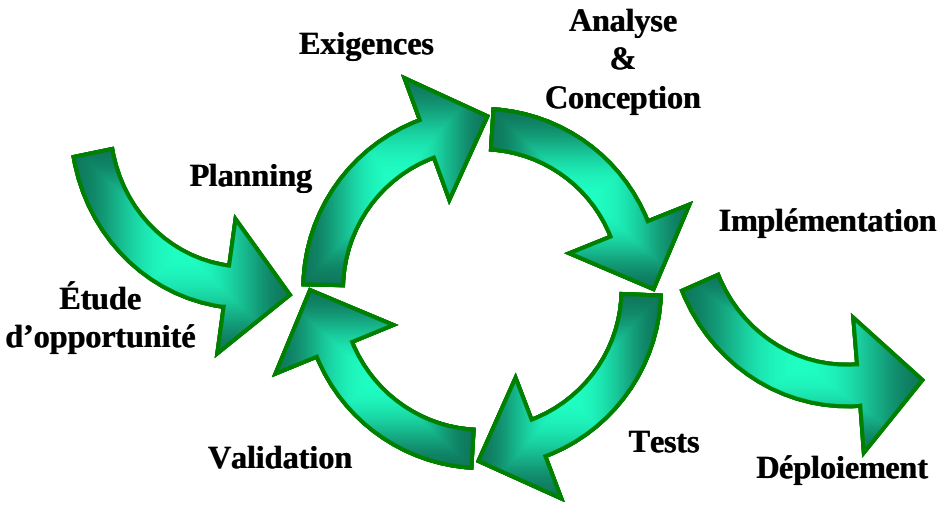


Figure 150 Cycle itérative de la méthode Processus Unifié (Kruchten, 1999).

Tel que décrit, les itérations doivent se succéder les unes aux autres sans recouvrement. Dans les faits, il arrive que les itérations se chevauchent car la tentation est forte pour une chef de projet d'accélérer le développement afin de réduire le coût global du projet (cf. figure 151). Ivar Jacobson insiste sur le fait qu'un chef de projet ne devrait pas accepter de commencer une nouvelle itération avant que les objectifs de l'itération en cours soient atteints. Lorsque cela se produit, il conseille de limiter le recouvrement au strict minimum car le résultat d'une itération est, par principe même de la méthode, le socle de travail de l'itération suivante, il faut donc terminer une itération avant d'en entamer une autre.

Figure 151 Recouvrement des itérations (Jacobson *et al.*, 1999).

Pierre-Alain Muller pense que certaines *itérations peuvent être conduites en parallèle* (Muller *et al.*, 2000). Toutefois, il ne précise par les conditions d'emploi de la pratique de recouvrement.

II.3.2.3 Les Phases du cycle de développement

II.3.2.3.a *L'inception*

L'***inception*** (Booch *et al.*, 2000) est une phase d'ébauche du projet que les certains auteurs qualifient d'***initialisation*** (Larman, 2002a ; Roques, 2002), d'***étude d'opportunité*** (Bénard, 2002b ; Muller *et al.*, 2000) ou bien encore de ***préétude*** (Roques *et al.*, 2002).

En début du cycle de développement, l'équipe ne dispose en tout et pour tout que d'une description très sommaire du système qui tient souvent en une page (Jacobson *et al.*, 1999). Le but de cette phase va donc être de donner du corps au système. Pour ce faire, un premier point va être de définir la finalité du système (Booch *et al.*, 2000 ; Larman, 2002a ; Muller *et al.*, 2000) et un second de délimiter le périmètre du domaine d'étude (Bénard, 2002b ; Booch *et al.*, 2000 ; Larman, 2002a). Une fois les frontières du système fixées, les spécifications du produit final (Muller *et al.*, 2000) vont pouvoir être établies de concert avec les acteurs du domaine c'est à dire que les principales exigences, les fonctionnalités clés et les contraintes majeures vont être identifiés (Bénard, 2002b) et listées. Sauf cas particulier, leur description sera plutôt sommaire, description qui sera reprise et approfondie en phase d'élaboration. En suivant ce processus d'analyse, il est évident qu'au cours de cette description en « surface » du domaine seul les principaux cas d'utilisation sont abordés et que les autres seront découverts en phase d'élaboration. À ce stade, 10 % environ des cas d'utilisation sont implémentés (Bénard, 2002b) et la création d'un prototype exécutable peut servir de preuve pour la validité du concept (Booch *et al.*, 2000). Ivar Jacobson est quant à lui plus réservé sur le nombre de cas d'utilisation à implémenter, nombre qu'il situe autour de quelques pourcents.

Cette description du système est l'occasion aussi de cerner les risques majeurs pouvant mettre en péril le projet (Bénard, 2002b ; Booch *et al.*, 2000). Une première liste des risques est constituée, liste qui sera ensuite mise à jour régulièrement tout au long du projet au moins à chaque itération. Une première estimation financière complète ce travail (Larman, 2002a).

Muni de ces éléments et d'une étude de marché (Muller *et al.*, 2000), le chef de projet va pouvoir discuter avec le commanditaire de la faisabilité du projet (Bénard, 2002b ; Roques, 2002) et de la suite à donner : poursuite ou l'arrêt du projet (Roques, 2002). Dans une situation mitigée où les éléments de décision ne sont ni franchement en faveur d'un arrêt ou ni franchement en faveur d'une poursuite, la solution souvent proposée par les chefs de projet est de financer la tranche supplémentaire du développement, c'est-à-dire la phase d'élaboration au terme de laquelle tous les cas d'utilisation sont identifiés. L'incertitude financière est alors mieux maîtrisée comme le montre la figure 152.

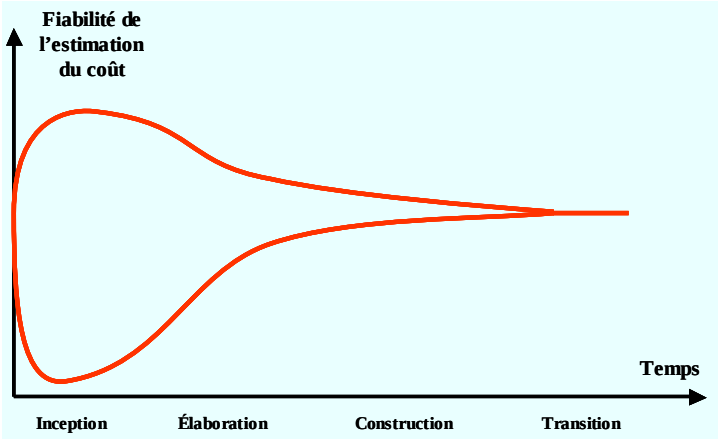


Figure 152 Incertitude sur l'estimation du coût d'un projet (Muller *et al.*, 2000).

II.3.2.3.b *L'élaboration*

Au cours de la phase d'***élaboration***, trois objectifs sont poursuivis en parallèle : l'identification et la description de la majeure partie des besoins acteurs (Booch *et al.*, 2000 ; Muller *et al.*, 2000 ; Roques, 2002), la construction -une description textuelle ne suffit pas- de l'architecture du système (Bénard, 2002b ; Booch *et al.*, 2000 ; Roques, 2002) et enfin la détermination et la révision des risques majeurs du projet (Bénard, 2002b ; Booch *et al.*, 2000 ; Roques, 2002).

L'architecte doit s'intéresser à la compréhension globale du système c'est-à-dire le domaine à couvrir et les besoins fonctionnels ou non fonctionnels à satisfaire. Pour ce faire, il doit alterner la construction de l'architecture du système avec la définition des fonctionnalités au travers des cas d'utilisation. Ce sont deux tâches à mener en parallèle car elles s'influencent mutuellement. (Jacobson *et al.*, 1999) conseillent en outre d'osciller entre une approche très large du système à modéliser sans entrer dans les détails et une approche scrutant en profondeur un point très précis. Pour imager des deux approches Ivar Jacobson utilisent les métaphores *« mile wide and inch deep »* pour la première et *« inch wide and mile deep »* pour la seconde.

Les principales tâches à réaliser au cours de la phase d'élaboration sont l'identification des exigences des acteurs agissant sur le système et la description des cas d'utilisation. Ces derniers doivent être décrits de façon assez précise pour bien comprendre les besoins. Ils doivent être structurés afin de bien appréhender l'enchaînement des actions et un ordre de priorités doit être établi. Ce travail ainsi que celui d'analyse et de conception est réalisé de façon itérative comme le préconise la méthode. Pour les projets de taille réduite, une paire d'itérations peut suffire mais les projets plus conséquents peuvent nécessiter plusieurs itérations. Les activités d'implémentation et de test sont assez réduites lors de cette phase mais la réalisation d'un prototype l'implémentant les principaux cas d'utilisation (Booch *et al.*, 2000) peut aider les utilisateurs à mieux cerner leurs spécifications et leurs besoins stabilisant par la même occasion les concepts et l'architecture. En phase d'élaboration, (Jacobson *et al.*, 1999) considèrent que les interfaces Homme/Machine ne sont pas des produits majeurs du prototype aussi il déconseille de les réaliser. Pour lui, cette préconisation est une règle générale à respecter.

Au cours de cette phase, un plan de développement est également établi (Bénard, 2002b ; Muller *et al.*, 2000) incluant la détermination des ressources, la conception et la validation de l'architecture du logiciel (Muller *et al.*, 2000).

Au terme de cette phase, 80 % des cas d'utilisation sont dénombrés (Bénard, 2002b ; Jacobson *et al.*, 1999) et un *noyau architectural du système* (Larman, 2002a) est réalisé.

II.3.2.3.c *La construction*

La phase de **construction** consiste à concevoir et à implémenter l'ensemble des éléments opérationnels (autres que ceux de l'architecture de base) (Roques, 2002). Leur approfondissement et leur implémentation vont être les tâches essentielles de la phase de construction (Jacobson *et al.*, 1999). Craig Larman résume cette phase par la métaphore : *on met de la chair au squelette créé lors de la phase d'élaboration* (Larman, 2002a).

En début de phase, l'équipe dispose d'une l'architecture de l'application définie dans ses grandes lignes, architecture qui va être complétée au fur et à mesure des itérations. Le chef de projet, l'architecte et les développeurs chevronnés vont établir un ordre de priorité pour réaliser les cas d'utilisation. Cet ordre va être

rediscuté, affiné et modifié s'il y a lieu à chaque itération. Ce travail d'ordonnancement est primordial et doit être fait sérieusement afin que le processus de développement soit en progression constante et éviter des retours en arrière sur le code déjà réalisé. Bien évidemment, la réalisation proprement dite des fonctionnalités est l'activité majeure de cette phase, complétée par les nombreux tests : tests d'intégration, de performances, etc.

Pour les projets importants, les développements peuvent être menés en parallèle (Booch *et al.*, 2000) au cours d'une même itération dans la mesure où les sous-systèmes sont indépendants. Une autre tâche essentielle de cette phase consiste à ré-identifier à lister les risques potentiels du développement et à tenir constamment à jour cette liste en raffinant ceux qui ont déjà été identifiés et en ajoutant les nouveaux. Ce suivi précis des risques permet de les prendre en compte lors des planifications des itérations et d'atténuer ainsi leur portée.

Cette phase se caractérise par la production d'une version bêta de l'application (Bénard, 2002b ; Booch *et al.*, 2000 ; Muller *et al.*, 2000) et du manuel utilisateur correspondant (Bénard, 2002b) et du modèle du système. C'est évidemment *la phase la plus consommatrice de ressources* (Roques, 2002).

II.3.2.3.d *La transition*

La ***transition*** est la phase de transfert de l'application vers les utilisateurs (Booch *et al.*, 2000 ; Roques, 2002) et de son déploiement dans un environnement opérationnel (Jacobson *et al.*, 1999). Quelquefois, avant son déploiement final, l'application est mise en service dans un environnement préfigurant cet environnement opérationnel en vue de consolider les spécifications, les fonctionnalités et corriger les éventuels bogues. À ce stade, la version de l'application déployée est la version *bêta* de la phase de construction. Elle peut encore faire l'objet d'améliorations ou d'ajouts de nouvelles fonctionnalités mineures (Booch *et al.*, 2000) mais la priorité est donnée au traitement des bogues. Sa mise en service exige que les bases de données soient transférées de l'ancien système vers le nouveau et que l'application soit conditionnée avec son manuel d'utilisation et sa documentation technique. C'est aussi au cours de cette phase qu'un logiciel d'installation est créé.

Dans certaines circonstances, l'équipe peut avoir comme mission d'aider les utilisateurs à gérer l'ancien système et le nouveau en parallèle (Jacobson *et al.*, 1999). La formation des utilisateurs (Muller *et al.*, 2000 ; Roques, 2002), le support technique (Muller *et al.*, 2000) et la maintenance (Muller *et al.*, 2000) sont mobilisés pour la réussite du projet. À ce stade, *on décide si les objectifs du projet sont atteints* (Booch *et al.*, 2000).

Le cycle de développement s'achève par *soit sur une nouvelle étude d'opportunité soit une élaboration ou une construction* (Bénard, 2002b) en vue de réitérer, si cela est nécessaire, *un autre cycle de vie du projet de développement* (Booch *et al.*, 2000). Une étape importante de cette phase est d'établir un recueil des *enseignements tirés du projet pour améliorer le processus de développement et appliquer tout cela au projet suivant* (Booch *et al.*, 2000).

II.3.2.3.e *En résumé*

Par principe, la méthode de conduite de projet Processus Unifié est centrée sur les cas d'utilisation qui guident le développement et la gestion du projet tout au long des phases d'inception, d'élaboration, de construction et de transition. La capture des cas d'utilisation n'est pas concentrée au début du projet mais étalée sur les trois phases d'inception, d'élaboration et de construction.

Afin de synthétiser l'évolution des cas d'utilisation au cours du projet, Ivar Jacobson a rassemblé au sein d'un tableau les pourcentages de cas d'utilisation identifiés, décrits, analysés et enfin implémentés et testés selon la phase (cf. tableau 10).

L'analyse de ce tableau montre que, bien que 80% soit identifiés au terme de la phase d'élaboration, seulement moins de 10% sont implémentés et testés. Les 90% restant sont implémentés au cours de phase de construction. Ce tableau montre aussi que plus aucun cas d'utilisation ne doit être créé en phase de transition, phase qui est consacrée à la mise en service de l'application.

	Modélisation métier/thématique accomplie	Cas d'utilisation identifiés	Cas d'utilisation décrits	Cas d'utilisation analysés	Cas d'utilisation implémentés et testés
Inception	50 à 70%	50%	10%	5%	Quelques concepts permettant la réalisation d'un prototype
Élaboration	presque 100%	80% ou plus	40 à 80%	20 à 40%	moins de 10%
Construction	100%	100%	100%	presque 100%	100%
Transition					

Tableau 10 Évolution[112] d'un projet structuré par les cas d'utilisation selon la méthode Processus Unifié (Jacobson *et al.*, 1999).

II.4 Outils

II.4.1.1 Généralités

Selon (Jacobson *et al.*, 1999), le développement d'applications étant réalisé par des hommes, un besoin d'uniformisation des modes opératoires s'impose pour rendre le développement plus homogène et efficace. La figure 146 montre que le processus de développement d'une application met en œuvre des outils afin d'assister et d'automatiser sa réalisation. Ivar Jacobson manifeste un avis beaucoup plus tranché sur ce sujet puisqu'il considère qu'*aujourd'hui, il est impensable de développer une application sans outil*. Il ajoute aussi que *le processus et les outils constituent une suite, ces derniers étant inséparables du processus*. Pour lui, le couple processus/outil constitue un subtil équilibre : les ***outils utilisés ont un impact sur le processus*** mais, inversement, le ***processus influence les outils*** mis en œuvre pour le développement.

Selon les fonctionnalités offertes par l'outil, le travail quotidien des développeurs est plus ou moins facile. C'est le cas en particulier de certaines tâches connexes au code, cœur du développement, et qui sont fastidieuses et répétitives. Par exemple, l'utilisation d'un outil découplant la gestion du modèle et de la documentation ou du code imposera aux développeurs des opérations de mise à jour de la documentation ou du code par rapport au modèle. Les outils étant séparés, cela amènera le développeur à faire manuellement ces opérations qui deviendront très rapidement fastidieuses. Inévitablement, les mises à jour seront de moins en moins fréquentes ayant pour conséquence une dérive de ces trois produits et un manque de cohérence. Dans ce contexte, chaque mise à jour deviendra alors une opération lourde de plus en plus difficile à réaliser et donc pénible. Cette absence de cohérence régulière est source de problèmes et de bogues lors de l'intégration des sous-systèmes développés séparément par des groupes de développeurs. La conséquence est une perte de productivité. Les outils sont donc mis en œuvre pour automatiser, partiellement ou totalement, certaines activités, afin d'augmenter la productivité et la qualité du produit final et de réduire des délais de réalisation.

Selon Ivar Jacobson, *le processus est la seule raison d'utiliser un outil*. Le but de ce dernier est d'automatiser au mieux le processus et de le rendre le plus facile possible. Aussi, le choix d'un processus va inciter le chef de projet à favoriser ou à imposer les outils les mieux adaptés au processus adopté. Si d'un point de vue contractuel, une documentation précise doit être fournie avec l'application, il est évidemment que le responsable du projet choisira les outils de modélisation et de développement les plus adaptés à automatiser cette tâche afin de la rendre la moins pénible possible augmentant par la même occasion la qualité du produit final. Par exemple, un outil de modélisation possédant une fonctionnalité de gestion et de génération de la documentation facilitera la production de cette dernière.

[112] Les valeurs contenues dans ce tableau constituent des grandes masses permettant de fixer les idées. Elles peuvent donc faire l'objet de d'ajustements sensibles selon le projet.

II.4.1.2 L'Outil RUP®

Pour faciliter la mise en œuvre de la méthode Processus Unifié, la société Rational Software Corporation l'a instrumenté dans son produit commercial, Rational Unified Process® (RUP®), *qui affine, met à jour et détaille le Processus Unifié plus général* (Larman, 2002a).

Ce produit *est constitué d'une documentation cohérente et bien conçue, présentée sous forme de pages HTML* (Fayet, 2002 ; Larman, 2002a) *et couplé à une base de connaissances consultable sur l'Intranet de l'équipe de développement* (Fayet, 2002). Il permet l'interfaçage avec d'autres outils comme des outils d'expression des besoins, des outils de modélisation UML, etc. En outre, ce produit est *configurable mais aussi actualisable régulièrement* (Fayet, 2002) ce qui lui confère une très grande souplesse d'utilisation.

Pour Philippe Kruchten en charge du Rational Unified Process® chez Rational Software Corporation, *le RUP® ne doit pas être vu comme un livre de recettes de cuisines à suivre à la lettre mais beaucoup plus comme un guide de savoir-faire à adapter dans le contexte du projet* (Fayet, 2002).

II.5 En résumé

Le plus important des quatre principes de la méthode de conduite de projet Processus Unifié reste sans aucun doute la ***conduite itérative et incrémentale du projet*** (Larman, 2002a ; Roques, 2002). C'est finalement grâce à ce principe que les ajustements successifs peuvent être faits autant au niveau du modèle qu'au niveau du code. Il permet également de réduire les défauts et les risques encourus au cours du projet. La citation de Martin Fowler en début de chapitre renforce ce constat au point où elle peut être réinterprétée de la façon suivante : *si on veut que des projets de développement échoués, il suffit d'adopter une méthode de conduite de projet sans processus itératif* réinterprétation lourde de conséquence…

Le second principe majeur est la ***conduite centrée sur les cas d'utilisation*** qui *doivent être vus comme un fil rouge* (Muller *et al.*, 2000) permettant l'expression des spécifications et des besoins des acteurs du domaine, l'implémentation proprement dite du système, les tests et mais aussi la gestion des risques. *Ce fil rouge banalise les différentes activités de développement* selon Pierre-Alain Muller.

De par son origine, la méthode Processus Unifié préconise le langage UML qui incorpore le formalisme des cas d'utilisation. Ils forment donc un couple opérationnel permettant de réaliser des modèles en langage UML suivant une méthode de type Processus Unifié. D'autres outils comme ceux de planification de projet peuvent venir compléter la panoplie. La société Rational Software Corporation a instancié la méthode proprement dite dans son outil connu sous le nom de *Rational Unified Process* qui assiste les chefs de projet tout au long du projet.

II.6 Remarques et limites de la méthode Processus Unifié

Concernant le mode de transcription des cas d'utilisation, Pierre-Alain Muller et (Jacobson *et al.*, 1999) semblent avoir des approches légèrement différentes mais qui ont une conséquence non négligeable.

Le premier conseille de les exprimer *dans les termes des utilisateurs, sous une forme textuelle, loin de tout formalisme informatique* à charge des analystes de faire la *transition vers une représentation informatique*. Alors que le second préconise de décrire les cas d'utilisation à l'aide d'un atelier de génie logiciel. La forme textuelle présente l'inconvénient majeur qui est qu'un certain nombre d'information existe sur deux supports physiques différents (forme textuelle + code). Dans ce cas, des problèmes de maintien de la cohérence des informations apparaîtront rapidement. L'utilisation d'un atelier de génie logiciel rendra plus facile le maintien de la cohérence et, si l'atelier de génie logiciel dispose d'une fonctionnalité de génération automatique de code, une documentation à jour sera disponible à tout moment. C'est un gage de qualité.

Il faut souligner que, bien que très attrayant, la méthode de conduite de projet Processus Unifié n'est pas adapté à tous les projets de développement informatique. En effet, c'est une méthode qui, par l'instrumentation mobilisée, devient rapidement coûteuse (Fondation Wikimédia, 2005) et qui ne peut donc être déployée que pour des développements à gros budget comme ceux pouvant exister dans le secteur de l'aéronautique et de l'espace, de la défense, des télécommunications, etc. En recherche, il arrive souvent qu'un développement ponctuel soit effectué à l'occasion d'une recherche très ciblée dans le temps en vue d'explorer un nouveau domaine de recherche. Or, la mise en œuvre d'une méthode de conduite de projet traditionnelle, type Processus Unifié, induit une surcharge de travail et un allongement de la durée du projet (Bénard, 2002a) qui se traduisent par des coûts supplémentaires. Dans ce contexte de recherche, les surcoûts sont souvent insolvables au vu de la taille réduite du projet. L'application d'une méthode relevant des concepts de l'agilité est alors préférable à une absence totale de méthode.

III eXTREME PROGRAMMING (XP)

L'eXtreme Programming est une méthode de conduite de projet qui relève résolument des méthodes *agiles*. Sa formalisation est attribuée à Kent Beck, Ward Cunningham et Ron Jeffries, tous trois experts en développement logiciel et souvent surnommés *les trois extremos*. L'acte de fondateur de cette méthode est la publication de l'ouvrage de Kent Beck *Extreme Programming Explained* (Beck, 2000). Cet ouvrage est le résultat de tout un travail de mise au point et d'expérimentation de cette méthode au cours du projet de développement logiciel C3 Payroll[113] chez Chrysler. Bien d'autres contributeurs ont participé et alimenté les réflexions autour de cette méthode, les plus importants étant : Martin Fowler auteur de plusieurs ouvrages dont les deux références citées en bibliographie (Fowler, 2002 ; Fowler *et al.*, 2001b) et Robert Cecil Martin président de la société Object Mentor.

L'objectif de cette méthode est, tout en les réduisant au strict minimum, d'organiser les activités connexes mais indispensables à la production de code dans un processus reproductible. Son originalité réside bien dans l'organisation *légère* des activités connexes au développement a contrario des méthodes de développement traditionnelles qui privilégient une planification forte. EXtreme Programming est une méthode de conduite de projet dont le processus de développement est aussi itératif c'est-à-dire que *le produit est en conception et développement constant* (Cros, 2001). Une très large gamme d'activités est concernée par cet objectif de minimisation de l'activité : la planification d'un projet, l'organisation de l'équipe de développement etc. mais aussi des relations avec le commanditaire.

Pour mettre en œuvre cette méthode, les concepteurs d'eXtreme Programming associent le développement d'une application à un *jeu de rôle* entre le client et les développeurs (Beck, 2000 ; Bénard *et al.*, 2002) avec pour objectif de tirer le meilleur parti de leur collaboration et maximiser ainsi la valeur de l'application en cours de développement.

Nous allons présenter dans ce qui suit les quatre valeurs sur lesquelles la méthode est fondée (cf. III.1), les différents rôles à assurer au sein d'une équipe eXtreme Programming (cf. III.2) et les douze pratiques (cf. III.3) qui définissent cette méthode.

III.1 Les quatre valeurs d'eXtreme Programming

La méthode eXtreme Programming rassemble sa communauté autour de quatre valeurs qui en constituent l'essence même :

La **communication** : les concepteurs de cette méthode ont érigé la communication en valeur fondamentale car de nombreux projets de développement ont échoué à cause d'une mauvaise communication entre les différents acteurs du projet : commanditaire, utilisateurs, chef de projet, équipe de développement, etc.

La **simplicité** : l'objectif premier de tout développement étant la satisfaction du commanditaire, la simplicité du développement est une valeur vitale pour obtenir un code compréhensible et ne pas développer des gadgets inutiles au commanditaire.

Le **feedback** : c'est une valeur essentielle aux développeurs qui doivent avoir à tout moment un retour d'information rapide et de qualité de la part du commanditaire et de tous les autres membres de l'équipe.

Le **courage** : cette valeur est une qualité humaine incontournable d'après les auteurs de la méthode car ils estiment que les acteurs utilisant cette méthode doivent avoir la détermination et la bravoure de dire les choses telles qu'elles sont et de prendre des décisions difficiles comme *changer une structure de code ou de le jeter* (Cros, 2001).

III.2 L'équipe

Dans tout projet, la constitution de l'équipe par le chef de projet est une gageure qui conditionne sa réussite ou son échec. Les relations entre les membres de l'équipe sont fondamentales et peuvent faire échouer un projet même s'il ne présente aucune difficulté technique majeure. La réunion de personnes, aussi brillantes et compétentes soient-elles, au sein d'une même équipe ne suffit pas pour garantir une bonne communication entre elles et le bon déroulement du projet. Le chef de projet doit donc veiller à construire un esprit d'équipe et surtout à ce qu'il perdure tout au long du projet. Dans sa tâche, il est aidé d'une part, par la définition d'un certain nombre de rôles qui doivent être assumés et assurés au sein de l'équipe et, d'autres part, par des pratiques de fonctionnement collaboratives que nous détaillerons dans le paragraphe suivant (cf. III.3). Un projet eXtreme

[113] C3 : Chrysler Comprehensive Compensation, projet destiné à assurer le payement sans faille des employés de chez Chrysler au passage de l'an 2000 (Jeffries, 2004).

Programming implique une dizaine de membres. C'est une taille qui semble optimale au vu des retours d'expérience évoqués lors du séminaire sur les Méthodes Agiles organisé par le Centre de Maîtrise des Systèmes et du Logiciel (CMSL, 2005).

III.2.1 *Programmeur (Programmer)*

Un des rôles parmi les plus importants est celui du *programmeur* car il est au cœur même de la méthode eXtreme Programming. Le membre de l'équipe assumant ce rôle doit savoir communiquer avec les autres acteurs du projet dont le client, autre rôle majeur de l'équipe. En outre, il doit écrire le code de l'application, mettre en place les tests d'évaluation de la qualité de l'application, fractionner l'application en briques élémentaires, regrouper les briques de façon cohérente, déplacer certaines d'elles, utiliser un système de nommage précis explicitant l'intention du code, etc.

Au vu du large panel de compétences et de qualités humaines exigés par ce rôle, il incombe souvent à des *programmeurs* confirmés. Heureusement en raison du fonctionnement collaboratif que nous verrons plus loin, tous les *programmeurs* de l'équipe n'ont pas besoin d'être confirmés. Comme nous le verrons, le travail en binôme va permettre aux *programmeurs* non confirmés d'acquérir les compétences et les qualités humaines nécessaires, enseignements qu'ils vont pouvoir réinvestir immédiatement en transmettant leurs nouvelles compétentes à d'autres membres et en prenant en charge des responsabilités.

Le Programmeur est au **cœur** de la méthode eXtreme Programming.

III.2.2 *Client (Customer)*

C'est un rôle clé dans une équipe eXtreme Programming. Il a la responsabilité de définir ce que doit faire l'application et la façon dont cela doit être fait. Pour ce faire, il doit communiquer ces informations au programmeur.

Dans un projet eXtreme Programming, le *client* doit être convaincu des vertus de la simplicité afin de ne pas solliciter des gadgets inutiles ou irréalisables. Il doit savoir faire preuve de courage lorsqu'il engage les programmeurs dans le développement d'une fonctionnalité qui s'avère être soit non attendue par les utilisateurs finaux de l'application soit une erreur de jugement. Par dessus tout, il doit favoriser la communication et le feedback. C'est la communication qui va lui permettre de bien expliquer les attendus de l'application aux programmeurs afin d'obtenir un produit adapté à ses besoins. C'est le processus de feedback rapide avec les programmeurs et les utilisateurs finaux qui permettra au client d'identifier ses propres erreurs de jugement avant que le projet ne dérive trop.

Une première activité essentielle de ce rôle consiste à décrire les fonctionnalités de l'application ou les interactions entre utilisateurs et application sous forme de *scénarios clients* (*user stories*) en les rédigeant manuellement sur des fiches bristols. Comme toute histoire, ces scénarios clients ont un début et une fin. Ces scénarios clients s'apparentent aux cas d'utilisation d'UML. Une deuxième activité clé est de définir les tests de recettes pour le membre de l'équipe jouant le rôle de testeur. Là encore la tâche est difficile car le *client* devra d'une part, décrire le test à implémenter et, d'autre part, fournir les éléments chiffrés pour réaliser ces tests.

Enfin, la troisième activité majeure est de prendre des décisions tout au long du projet pour lever ou préciser les insuffisances de spécifications initiales.

Ce rôle n'est pas obligatoirement tenu par le commanditaire ou son représentant. Dans ce cas là, c'est souvent un membre de l'équipe de développement. Mais selon Kent Beck, le meilleur *client* est l'acteur qui devra utiliser l'application en cours de développement et qui a la capacité en outre de prospection pour entrevoir la façon dont le problème peut être résolu et dont les processus s'enchaînent.

Une des pratiques suggérée par cette méthode pour faciliter la communication et les échanges entre le *client* et les programmeurs est que le *client* soit membre à part entière de l'équipe et, si ce rôle est tenu par le commanditaire ou son représentant, qu'il s'installe dans les mêmes locaux que l'équipe.

Le Client est le **décideur du contenu technique** de l'application, rôle difficile à assumer et qui nécessite beaucoup de courage.

III.2.3 *Testeur (Tester)*

Le *testeur* travaille de concert avec le client. Sa principale activité réside à implémenter les tests -de recettes- définis par le client. Pour ce faire, il doit aider le client à exprimer les tests, à les matérialiser et à évaluer leur pertinence. Toutefois, il peut aussi être amené à déconseiller des fonctionnalités imaginées par le client ou des

technologies utilisés par les programmeurs car les unes comme les autres sont difficilement testables. Pour que l'application soit entièrement testable, il peut solliciter certaines adaptations du code aux programmeurs.

Le testeur est le **bras technique** du client.

III.2.4 *Tracker*

Le rôle du *Tracker* est de s'assurer du bon avancement du projet au cours d'une itération afin de mettre en évidence très tôt une éventuelle dérive du projet. Pour ce faire, deux à trois fois par semaine, il rencontre les programmeurs les uns après les autres et s'enquiert de l'avancement de la tâche en cours et leur demande une nouvelle réévaluation. Une fois cet exercice fait, s'il a des craintes ou s'il note l'émergence d'un problème, il en réfère au coach qui est le seul habilité à décider des mesures à prendre.

Ce rôle demande beaucoup de doigté car le *Tracker* doit dialoguer avec les programmeurs et maintenir un niveau de confiance élevé pour obtenir les informations nécessaires pour assumer sa tâche mais il doit surtout éviter de mettre les programmeurs sous pression.

Pour Kent Beck, le Tracker est la **conscience** de l'équipe et son **historien**.

III.2.5 *Coach*

Le *coach* est responsable du développement dans son ensemble. À ce titre, il est le garant du bon fonctionnement de la méthode. Pour ce faire, il doit avoir une parfaite connaissance de la méthode eXtreme Programming afin de pouvoir trouver des alternatives lorsque les problèmes de fonctionnement de l'équipe apparaissent, alternatives qui doivent rester dans l'esprit de la méthode. Il doit s'assurer d'une part, que chacun joue son rôle et que les pratiques eXtreme Programming que nous verrons plus loin sont respectées et, d'autre part, que l'équipe *fait ce qu'elle dit qu'elle allait faire*. Il doit favoriser la créativité de l'équipe plutôt que d'imposer ses propres solutions. Il doit rester calme dans toutes les circonstances surtout lorsque tout le monde panique.

L'objectif absolu du *coach* est que l'équipe puisse fonctionner sans lui. Petit à petit, il doit être moins directif et laisser les membres de l'équipe trouver leur propre dynamique et mettre en œuvre leurs propres solutions.

Pour Kent Beck, la difficulté la plus importante qu'il ait rencontrée dans ce rôle est de se convaincre qu'il travaillait bien en œuvrant indirectement.

Par extension de la métaphore de Pierre-Alain Muller sur le bateau traversant l'océan (cf. Annexe II), le Coach est le **skipper** d'un projet qu'il doit mener à bon port. Il a la rude tâche de dire les choses telles qu'elles sont (Bénard *et al.*, 2002). Pour cela, il doit faire preuve de beaucoup de courage.

III.2.6 *Consultant*[114]

Dans son fonctionnement, un projet eXtreme Programming a peu de chance de former des spécialistes puisque le travail en binôme avec un turn-over important favorise la diffusion de la connaissance et implique les programmeurs à tous les niveaux de l'application.

Ce mode de fonctionnement est une force car il permet une flexibilité grande mais c'est aussi une faiblesse car l'équipe peu avoir besoin de connaissances très pointues du ressort de spécialistes. Dans ces cas, l'équipe peut faire appel à des *consultants* dont la seule tâche est de mettre à disposition ses connaissances pour que les programmeurs élaborent leur propre solution. En aucun cas, il ne doit suggérer ou livrer une solution clé en main.

Le Consultant est une **ressource technique** de l'équipe. Il doit opérer par **enseignement** et non pas par injonction.

III.2.7 *Manager (Big Boss)*

Généralement, c'est lui qui a la responsabilité du projet. À ce titre, il assure la gestion matérielle de l'équipe dont le recrutement de nouveaux programmeurs, testeurs ou autres. Bien entendu, de par ces fonctions, ils réclament des comptes et demandent à l'équipe de faire ses preuves. Il attend d'elle des résultats tangibles et palpables. Ce suivi évite généralement toute dérive de l'équipe. Toutefois, lorsque l'équipe rencontre des difficultés, il doit avoir le courage de laisser la méthode se dérouler sans intervenir.

---

[114] Le rôle de Consultant n'est mentionné que dans l'ouvrage de Kent Beck (Beck, 2000) et dans l'article de Jean-Louis Bénard (Bénard, 2002c). Apparemment ce rôle a été abandonné : (Bénard *et al.*, 2002), http://www.extremeprogramming.org/, http://www.xprogramming.com/, etc.

Il a la charge d'informer les partenaires et le commanditaire de la progression du développement en cours et de communiquer à l'équipe les informations recueillies auprès de ces personnes. Il s'informe donc auprès du commanditaire si le projet évolue selon ses desiderata et s'il est satisfait.

Souvent, c'est le supérieur hiérarchique de programmeurs et parfois du client si ce rôle a été confié à un membre de l'équipe. Contrairement au coach, il ne fait pas partie intégrante de l'équipe.

Le Manager est le **responsable** du projet.

III.3 Les Pratiques

Une équipe opérant selon cette méthode concentre principalement son énergie sur la production de code sans pour autant faire de concessions sur la qualité du produit final. C'est une méthode bien adaptée aux petites équipes de développement impliquant une dizaine de personnes dont l'objectif est de produire des logiciels rapidement tout en se dotant d'un fort potentiel de réaction aux changements. Cette dernière propriété permet de répondre d'une part, aux évolutions inéluctables des spécifications souvent sur initiative du client et, d'autre part, à la prise en compte de spécifications incomplètes de l'analyse.

L'eXtreme Programming concentre son action sur douze pratiques qu'il est possible de classer en trois grandes catégories :

Les *pratiques de gestion de projet* qui définissent la planification et la relation avec le client.

Les *pratiques de collaboration* qui structurent et organisent le fonctionnement de l'équipe de développement.

Les *pratiques de programmation* qui définissent des principes de production et de gestion de code.

III.3.1 *Stratégie de Planification*

En eXtreme Programming, la planification est l'art de deviner ce qui seraient *bon d'implémenter* au cours de l'itération (Beck, 2000). Pour atteindre ce but, il est nécessaire d'effectuer un certain nombre de tâches : provoquer des réunions de l'équipe aussi souvent que cela est nécessaire, y décider de l'étendu du développement et des priorités, estimer les coûts et programmer le développement, donner confiance aux membres de l'équipe et, enfin, faire en sorte que le contexte et l'ambiance entre les membres de l'équipe soient favorables pour réaliser et faciliter le feedback.

III.3.1.1 Processus de planification collective (Planning game)

Dans le jeu de rôle adopté, les concepteurs utilisant une méthode eXtreme Programming fractionnent le processus de développement d'une application (*cycle de développement*) selon un double niveau d'itérations (cf. figure 153). Le premier niveau (*itération de livraison*) est calé sur la date de livraison des produits (livrables) qui seront présentés aux clients pour validation. Ces itérations de livraison comprennent elle-même une ou plusieurs *itérations de développement*[115] au cours desquelles sont réellement codés les concepts et les fonctionnalités définis par le client. Les activités des deux niveaux d'itérations sont identiques :

- ⇨ Une première phase d'*exploration* (*exploration*) initie l'itération. La durée de cette phase est courte. Son but est d'identifier et de bien assimiler les concepts à implémenter et d'évaluer le coût en termes de développement.
- ⇨ Elle est suivie de la phase d'*engagement* (*commitment*) très courte où sont décidés collectivement les concepts à implémenter au cours de l'itération.
- ⇨ La troisième phase est celle de *pilotage* (*steering*). De par sa durée, c'est la principale acticité de l'itération. Au cours de celle-ci sont réalisées les fonctionnalités demandées par le client.

Généralement, la durée des itérations de livraison oscille entre 1 et 3 semaines. Une durée de 1 semaine permet de mieux interagir avec les membres de l'équipe et de déceler plus rapidement une éventuelle dérive du projet. L'inconvénient majeur est que le volume de concepts et de fonctionnalités implémentées est restreint. Inversement lorsque la durée de l'itération est portée à 3 semaines, la dérive du projet devient plus difficile à discerner mais par contre l'incrément implémenté est plus conséquent. Aussi, des itérations de livraison de 2 semaines semblent être un bon compromis. Plusieurs auteurs conseillent cette durée.

Le nombre d'itérations de développement dépend quant à lui du volume de code à réaliser pour construire un nouvel incrément de logiciel. Si l'équipe s'engage à produire une fonctionnalité majeure très complexe et très

[115] Nous avons préféré utiliser cette terminologie à celle d'*itération* généralement consacrée par de nombreux auteurs. Nous avons fait ce choix afin de mieux différencier les deux niveaux d'itération.

difficile -c'est souvent le cas en début de projet- l'itération de développement pourra être confondue avec celle de livraison. Par contre, si l'équipe décide de faire des fonctionnalités simples, l'itération de livraison pourra être composée de plusieurs itérations de développement.

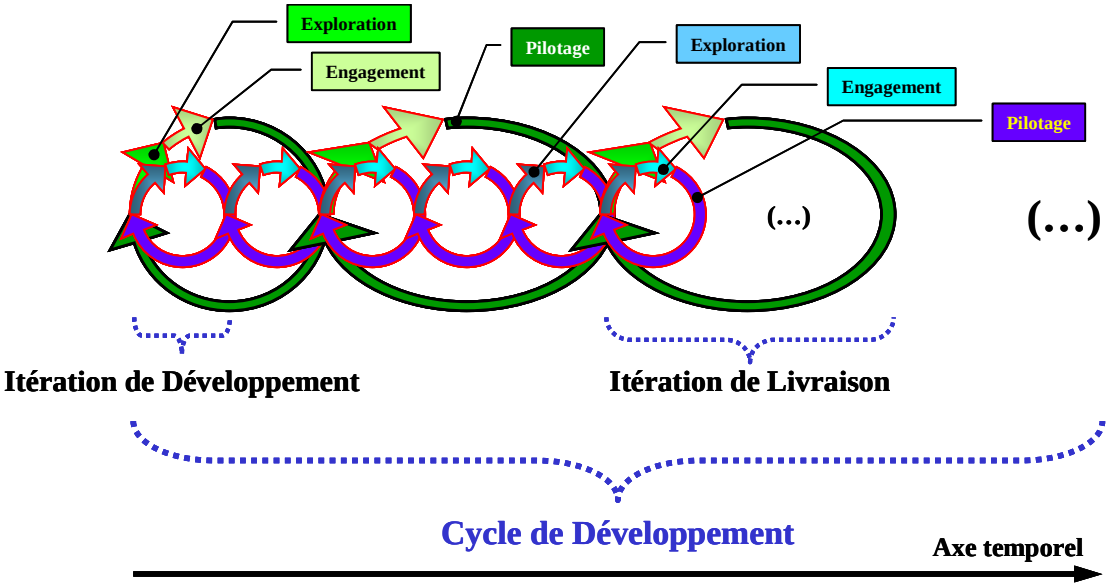


Figure 153 Illustration du cycle de développement, des itérations de livraison et des itérations de développement en eXtreme Programming (Bénard *et al.*, 2002).

III.3.1.2 Planification des Itérations de Livraison

Au début d'une itération de livraison, en *phase d'exploration*, le client et le programmeur se réunissent et le client décrit sous forme de scénarios (*user stories*) les fonctionnalités de l'application. Le client consigne par écrit ces scénarios souvent avec l'aide du coach. Ces scénarios doivent présenter certaines qualités : ils doivent être facilement estimables et leur suivi doit être possible. Cela signifie d'une part que leur granularité doit être suffisamment fine pour que les développeurs ait un aperçu complet sur le développement à réaliser pour chacun d'eux et, d'autre part, ils doivent être testables, tests qui, tout au long du projet, permettront de vérifier que la fonctionnalité est toujours correctement implémentée et intégrée dans son environnement. Les auteurs de la méthode conseillent d'écrire ces scénarios sur des fiches cartonnées au format A5. Ils ont jugé cette solution plus pratique que l'usage de logiciels de gestion de projet. Une fiche de scénario type (cf. figure 154) contient la description de la fonctionnalité à implémenter bien évidemment, la priorité donnée à cette fonctionnalité par le client, le risque technique est évalué principalement par les programmeurs et une estimation de la charge de travail nécessaire pour construire cette fonctionnalité.

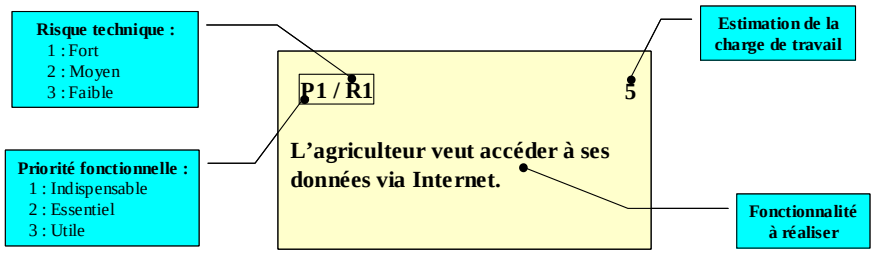


Figure 154 Fiche de scénario type (Bénard *et al.*, 2002).

L'estimation de la charge de développement pour implémenter une fonctionnalité n'est pas aisée. Elle comprend, outre le temps proprement dit de développement, le temps consacré aux réunions, le temps dédié à des tâches d'administration ou de gestion, les congés des membres de l'équipe, etc. Il est donc délicat de définir une unité de temps. Dans la pratique, les auteurs conseillent de construire sa propre unité de temps en affectant à

chaque fonctionnalité un certain nombre de points et de calculer le nombre de points des fonctionnalités correctement implémentées dans les premières itérations. Rapidement, cette unité converge vers une valeur calendaire stable. Pour les itérations suivantes, le client sait qu'il disposera d'un capital de points par itération qu'il devra consommer au mieux.

Lorsque certains scénarios se révèlent être trop importants ou portent sur une durée de développement trop longue, le programmeur peut demander au client de le fractionner en scénarios de granularité plus fine. A contrario, si des scénarios ont une granularité trop fine, ils peuvent être regroupés en un seul scénario. Le regroupement de scénarios n'est possible que s'ils sont cohérents du point de vue thématique.

La *phase d'exploration* se termine lorsque le client a exprimé tous ses besoins et que les scénarios sont explicités sous forme de fiche avec leur priorité, leur risque et leur coût.

Alors, la *phase d'engagement* peut démarrer. Elle consiste à dispatcher les scénarios au sein des itérations pour établir un plan de livraison des livrables tout au long du projet. C'est un plan assez vague au départ à cause en particulier de l'unité de temps qui n'est pas stabilisée mais, comme à chaque itération de livraison il sera revu, il deviendra de plus en plus précis grâce à l'expérience acquise par l'équipe sur son propre fonctionnement. Cette phase comporte une tâche essentielle qui consiste à trier les scénarios selon deux logiques :

- ⇨ Une logique « client » d'après le niveau de priorité des fonctionnalités qu'il a fixé.
- ⇨ Une logique « développeur » qui estime le risque technique lié à la réalisation de la fonctionnalité.

Ce double tri permet de classer les scénarios en neuf catégories selon l'utilité de la fonctionnalité et du risque à la réaliser comme le montre la figure 155. Le choix par client des scénarios à implémenter est alors plus facile. Il doit sélectionner parmi les scénarios restant à implémenter ceux qui sont vitaux pour l'application tout en n'excédant pas le capital dont ils disposent dans l'itération. Le plan de livraison qu'il en résulte est un élément de travail absolument pas figé qui évolue selon les nouveaux besoins exprimés par le client, le raffinement des spécifications et bien entendu des difficultés techniques rencontrées.

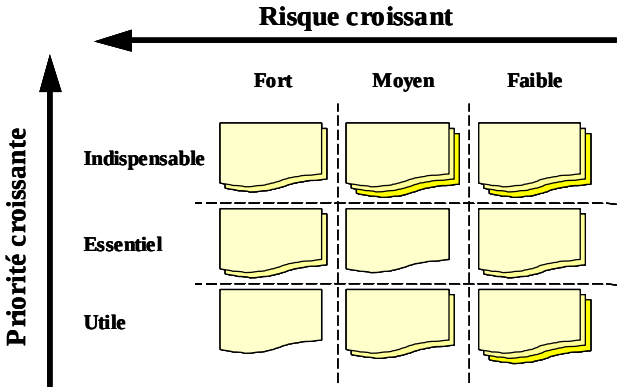


Figure 155 Tableau de classement des scénarios (Bénard *et al.*, 2002).

La *phase de pilotage* de l'itération de livraison est découpée en *itérations de développement* que nous verrons dans le paragraphe III.3.1.3. C'est au cours de cette phase que les concepts et les fonctionnalités sont réalisés ainsi que les tests permettant de vérifier qu'une fonctionnalité a été codée et intégrée correctement à son environnement. C'est aussi au cours de cette phase que sont traités les inévitables bogues. En cas des bogues, la démarche suggérée par la méthode eXtreme Programming pour les corriger est soit de modifier le scénario à l'origine du bogue soit de transformer ces bogues en nouveaux scénarios client. Dans les deux cas, les programmeurs avec l'aide du client doivent concevoir de nouveaux tests permettant de reproduire le bogue si la même anomalie se produit et de ne générer aucun bogue dans le cas contraire.

La fin de l'itération est formalisée par la livraison du produit au jour J et à l'heure H même si toutes les fonctionnalités ne sont pas terminées. Une équipe eXtreme Programming ne doit pas jouer avec le délai. Le coût -nombres de points- des fonctionnalités réellement implémentées au cours de l'itération est calculé afin de mettre à jour l'unité de temps.

La livraison est une étape importante d'un projet eXtreme Programming et les auteurs de la méthode considère que c'est l'occasion pour l'équipe complète de fêter l'événement en faisant une activité hors du cadre

professionnel, un bon repas par exemple (Bénard *et al.*, 2002) mais toute autre activité est possible le but étant de favoriser la communication entre les membres de l'équipe et d'aplanir les éventuels prémisse de problèmes.

III.3.1.3 Planification des Itérations de Développement

Tout comme les itérations de livraison, celles de développement sont aussi décomposées en trois phases : la phase d'exploration, la phase d'engagement et la phase de pilotage.

Au cours de la *phase d'exploration*, les scénarios clients sont ré-analysés conjointement par le client et les programmeurs afin de les décomposer en *tâches élémentaires* n'impliquant pas plus de une ou deux journées de travail. Ces tâches élémentaires sont elles aussi rédigées sur des fiches[116] mais, à la différence de celles de scénarios, elles n'y sont pas estimées car la méthode eXtreme Programming considère que l'estimation est du ressort du programmeur en charge de la réaliser. Étant donné qu'à ce stade les tâches ne sont pas affectées à un programmeur, l'estimation est différée à la phase d'engagement.

En *phase d'engagement*, les développeurs se partagent les tâches à réaliser sur la base du volontariat. Il devient alors responsable de la tâche à construire et donne une estimation temps du travail à faire. Il fait son estimation en se référant, au début du projet, à son expérience antérieure et, en cours de projet, au code réalisé lors des précédentes itérations de développement. Tous ces éléments permettent alors de planifier l'itération en cours. Souvent, le programme de travail est écrit sur un tableau. Il est essentiel que la charge de travail soit répartie équitablement entre les programmeurs. C'est-à-dire que la somme des estimations temps doit être grosso modo identique pour les développeurs[117]. Si ce n'est pas le cas, les programmeurs surchargés doivent céder les tâches à des collègues. Comme pour les scénarios, c'est aussi l'occasion de scinder ou de fusionner des tâches si cela s'avère nécessaire. Ces discussions peuvent aussi induire la création ou l'abandon de tâches.

La *phase de pilotage* commence seulement lorsque toute la gestion et la planification est effectuée. Les développeurs s'entendent pour travailler en binôme et développer en commun les tâches dont ils ont la charge. Les raisons de ce mode de travail sont expliquées en III.3.7. L'ordre de réalisation des tâches est de leur initiative. Une fois la tâche implémentée, ils l'indiquent sur le tableau où le programme de travail est inscrit. Afin de faire un suivi de l'avancement du projet, deux à trois fois par semaine le tracker fait le tour des développeurs et s'assurent du bon déroulement des tâches (cf. III.2.4). S'il note des dérives, il en fait part au coach. Une fois par jour, l'équipe se réunit pour des réunions (*stand-up meeting*) d'une dizaine de minutes afin de faire le point rapidement de la situation et des difficultés de la journée précédente, sans pour cela rentrer dans les détails techniques peuvent être l'objet d'une miniréunion prolongeant les stand-up meetings entre membres concernés. Ces stand-up meetings sont aussi l'occasion d'une part, d'évoquer la difficulté de réalisation de certaines tâches ou certains scénarios dans le temps imparti et de demander au client de les reformuler, de les différer ou de la supprimer si cela est possible et, d'autre part, de faire émerger les besoins particuliers comme la nécessité de renforcer l'équipe, celle de faire appel à un spécialiste pour définir le schéma d'une base de données, etc. En un mot, leur but est d'organiser la journée. L'itération de développement prend fin avec une livraison intermédiaire de l'application afin que le client puisse la tester et faire des retours.

Comme pour la fin des itérations de développement, les auteurs de la méthode suggèrent d'octroyer à l'équipe le droit de se détendre en faisant une activité externe au cadre professionnel.

III.3.2 *Livraisons fréquentes (Frequent releases)*

Les livrables doivent incorporer des incréments logiciels de taille aussi réduite que possible, chacun de ces incréments devant correspondre à l'implémentation d'une spécification métier. L'équipe de développement est donc amenée à produire de nouvelles versions de l'application à un rythme élevé. Cette fréquence élevée de livraison est une garantie pour le client et l'équipe que le produit de l'itération est conforme aux attentes du client. Le client est rassuré en continue du bon avancement du projet car il voit le produit évoluer itération après itération en fonction des spécifications qu'il a formulées. Les développeurs sont quant à eux confortés dans les choix techniques qu'ils ont effectués en cours d'itération puisque le produit obtenu satisfait le client. Ce mode de fonctionnement axé sur un couplage fort entre le client et les programmeurs et associé au processus itératif et incrémental du développement a toutes les chances de parvenir à un produit final conforme aux besoins du client, besoins qui évoluent souvent entre le début et la fin du projet.

La date de livraison d'un livrable est définie par le client. La périodicité des livraisons peut varier suivant le type d'application mais aussi en fonction des pratiques d'intégration de l'équipe. Muni de bonnes pratiques et d'outils d'intégration adéquats, des livraisons quotidiennes voire plus peuvent être envisagées, excepté pour les

[116] Jean-Louis Bénard (Bénard *et al.*, 2002) suggère l'utilisation de la méthode des cartes Classes-Responsabilités-Collaboration (CRC).

[117] Bien sûr c'est le ratio nombre de points sur le temps de présence qu'il faut considérer afin de prendre en compte les congés des membres de l'équipe, les mi-temps, etc.

applications nécessitant une validation externe comme les logiciels pour téléphones portables. Le rythme conseillé par les auteurs de la méthode est de déployer un livrable toutes les deux à trois semaines même si le rythme d'une semaine est envisageable. Pendant ce délai, l'équipe de développement peut réaliser/intégrer un nombre de fonctionnalités suffisamment « visibles » par le client. La date de livraison étant immuablement fixée par le client, l'équipe ne dispose que du volume de fonctionnalités implémentées entre deux versions comme variable d'ajustement de son processus de développement. Si l'équipe a du retard dans son développement, elle supprimera des fonctionnalités par contre si elle est en avance elle peut en ajouter. Le choix des nouvelles fonctionnalités à implémenter est toujours conditionné à la même contrainte : à la date de livraison, toutes les fonctionnalités de l'incrément doivent être opérationnelles.

III.3.3 *Métaphore (Metaphor)*

Dans son ouvrage, Kent Beck suggère de désigner et de décrire structurellement et fonctionnellement le système à développer par des métaphores. L'idéal étant de parvenir à désigner l'application par une seule métaphore. Quelquefois, les métaphores sont simplistes et naïves mais leurs seuls objectifs est d'aider les acteurs du projet à appréhender les concepts majeurs de l'application et les relations à établir entre ces concepts pour qu'ils échangent des informations.

Les équipes eXtreme Programming parvenant à qualifier par des métaphores l'application et ses sous-systèmes améliorent fortement d'une part, leur communication interne et par conséquent la qualité du produit final et, d'autre, leur communication externe offrant ainsi aux gestionnaires du projet et au client une visibilité et une lisibilité du projet bien meilleure.

L'une des métaphores les plus connues est le *bureau de Windows* qui donnent un accès directs aux applications ou aux fichiers permettent une meilleure ergonomie. Il est aussi possible de citer le concept *client-serveur* qui est au cœur de toute communication via Internet.

III.3.4 *Conception simple (Simple design)*

Selon les auteurs de la méthode, la conception d'une application est correcte si à tout moment le code en cours de développement satisfait les exigences les suivantes :

- ⇨ Le code doit subir avec succès tous les tests mis en place par les programmeurs pour contrôler la qualité du code, les exigences métiers du client, etc.
- ⇨ Le code doit être exempt de toute redondance : pour ce faire, il faut favoriser le développement de fonctionnalités génériques écrites une et une seule fois, par exemple la recherche d'un fichier dans un répertoire et ses sous-répertoires. Toute dérogation à cette règle se traduit par une maintenabilité plus difficile et fait chuter la qualité du développement.
- ⇨ Le code doit exprimer clairement l'intention des programmeurs lorsqu'ils implémentent un nouveau concept ou une nouvelle spécification qu'elle corresponde au métier du client ou informatique.
- ⇨ Le code doit être le plus concis possible c'est-à-dire qu'il faut limiter le nombre de classes, d'attributs, de méthodes etc. mais aussi chercher à réduire au strict minimum le nombre de lignes de code.

Pour Kent Beck, tout sous-ensemble de code doit être analysé et justifié à la lecture de cette grille d'exigences. Tout code dérogeant à l'une d'elle doit être récrit ou supprimé.

La première exigence est remplie si les tests sont validés lors de l'étape d'intégration continue (cf. III.3.9). Le processus de remaniement (cf. III.3.6) prend en compte la seconde exigence. Le code dupliqué est factorisé en continue de part le mode de travail de l'équipe ou recherché systématiquement grâce aux fonctionnalités de certains environnements de développement. Les exigences trois et quatre sont quant à elles aussi directement liées au mode de travail de l'équipe. La programmation en binôme (cf. III.3.7) et le turn-over important des membres de l'équipe au sein des binômes permettent de revenir sur l'intention d'un code donc de le partager, de vérifier sa simplicité et de l'épurer au maximum. Ce processus agit de la même façon sur l'architecture de l'application.

Faire simple suppose aussi de ne pas implémenter des fonctionnalités non définies par les spécifications. L'équipe doit à tout moment réfréner son désir de créer des fonctionnalités surnuméraires. Des tests devront être élaborés pour vérifier la qualité de ces fonctionnalités et, en outre, il faudra les prendre en compte dans la maintenance et l'évolution de l'application. Cela engendre des surcoûts.

III.3.5 *Stratégie de Test*

Comme le souligne Kent Beck (Beck, 2000), personne ne veut entendre parler de tests. Pourtant, tout le monde s'accorde sur leur importance. Comme le rappelle très judicieusement Ron Jeffries (Jeffries, 2001), le feedback est une *obsession de la méthode eXtreme Programming*. Or un feedback de qualité implique de tester le code de façon systématique en mettant en place tout un arsenal de tests de qualité et fiables. Une équipe eXtreme Programming peut instrumenter quasiment 100% de son code avec des tests. Leur utilisation systématique oblige souvent les développeurs à changer leur façon de travailler pour tenir compte des recommandations préconisées par la méthode :

- ⇨ Concevoir des tests simples et indépendants les uns des autres.
- ⇨ Les écrire avant l'implémentation de la fonctionnalité car l'expérience montre qu'un test est rarement écrit si le programmeur code d'abord la fonctionnalité. Il a souvent le sentiment de perdre son temps en écrivant le test. Comme dit Kent Beck, les programmeurs ont souvent une grande confiance en eux, confiance qui les rend aveugles.
- ⇨ Automatiser leur utilisation afin qu'ils soient systématiquement appliqués en phase d'intégration. Cela permet :
  - D'éprouver la totalité des fonctionnalités implémentées et avec moins de risque d'erreur que si la vérification était réalisée manuellement.
  - De s'assurer de la cohérence d'une fonctionnalité vis-à-vis du reste du code et de mettre en évidence les dysfonctionnements introduits par le codage ou le remaniement d'autres fonctionnalités.

Les tests sont construits pour toute la durée de vie du projet. Ils sont créés pour valider d'une part, les fonctionnalités sollicitées par le client (tests de recette) et, d'autre part, celles relatives à des concepts informatiques (tests unitaires) utilisés ou imaginés par les programmeurs comme les interfaces Homme/Machine ou des bases de données. Au travers les tests, la conception du développement est affinée par rapport à une conception traditionnelle.

La pratique des tests systématiques et automatiques permet souvent de ***réduire d'un facteur cinq*** les bogues de conception ou de développement d'une application (CMSL, 2005).

III.3.5.1 Tests de recette (Acceptance test/Customer test)

Par le biais des *tests de recette*, le client maîtrise la qualité du produit final. Ces tests constituent un *examen de passage* (Bénard *et al.*, 2002) pour le projet, examen qui, en cas de succès, met souvent en route le processus de facturation des développements effectués. La pertinence de ces tests dépend essentiellement de la faculté du client à décrire la fonctionnalité dont il a besoin. Un client capable de spécifier dans le détail une fonctionnalité est aussi en mesure de dire dans quelles conditions cette fonctionnalité peut être appliquée ou non. Le développeur peut alors en déduire un ensemble de tests qu'il peut coder et les intégrer au développement.

III.3.5.2 Tests unitaires (Unit test/Developer test)

Les *tests unitaires* sont l'ensemble des tests écrits par le développeur autre que les tests de recettes. Leur but essentiel est de prévenir la régression des fonctionnalités déjà écrites. Par exemple, un programmeur impliqué dans un projet de serveur Internet a pris la responsabilité d'implémenter la fonctionnalité sécurisant l'envoie d'un message. Il décide en accord avec le client d'adopter un algorithme de chiffrement à clés asymétriques, RSA[118] par exemple. Le test unitaire permettant de vérifier si le couple de fonctionnalités codage/décodage est et reste opérationnel au cours du projet serait :

- ⇨ D'appliquer la fonctionnalité de codage à un message de référence, un message codé serait obtenu.
- ⇨ D'appliquer au message codé la fonctionnalité de décodage, le résultat serait un message décodé.
- ⇨ Le test proprement dit consisterait à comparer le texte décodé au texte de référence. S'ils sont identiques les deux fonctionnalités sont opérationnelles sinon cela signifie qu'un problème est apparu. Le travail du développeur consisterait alors à déceler l'origine de ce bogue et à affiner ou à créer un nouveau test avant d'apporter des modifications à l'une ou l'autre des deux fonctionnalités.

Dans un tel cas, notre développeur aurait écrit les signatures des fonctionnalités de codage et de décodage et ensuite il aurait implémenté le code du test lui-même en offrant la possibilité de paramétrer le texte de référence.

[118] Acronyme constitué à partir des initiales du nom des inventeurs : Rivest, Shamir et Adleman.

Il pourrait ainsi l'instancier le teste sur plusieurs textes de référence afin d'avoir une meilleure assurance de la stabilité du code des deux fonctionnalités.

Les tests unitaires sont une façon de documenter le code produit par les développeurs car ils décrivent avec précision le fonctionnement de chaque entité de code. Par cet aspect là, les tests unitaires sont très proches *des notions de contrat de la conception par contrat prônées par Bertrand Meyer* (Bénard *et al.*, 2002).

III.3.6 *Remaniement (Refactoring)*

Pour les concepteurs de la méthode eXtreme Programming, le maintient d'une documentation à jour est difficile et, de ce fait, elle diverge à plus ou moins long terme du code proprement dit. Pour une équipe eXtreme Programming, le code représente le livrable le plus fiable. Cela les oblige à produire un code parfaitement compréhensible pour assurer sa maintenabilité à long terme et son évolution. Un premier moyen d'y parvenir est de bannir la duplication de code remplissant ainsi l'un des commandements majeurs de la méthode eXtreme Programming : *Once And Only Once* c'est-à-dire réaliser une et une seule fois une partie de code. Un second moyen est de structurer le code conformément aux intentions afin d'en faciliter la lecture.

Par exemple, le code permettant la lecture ligne à ligne d'un fichier texte de configuration d'une application ou de données sera toujours le même, seul changera le nom et le chemin pour atteindre le fichier. L'écriture de cette fonctionnalité évitera la duplication inutile et redondante de code. Le travail en binôme et le turn-over des membres de l'équipe au sein même des binômes induit une relecture fréquente du code ce qui facilite la mise en évidence de code redondant en intention, code qu'il faudra factoriser. (Fowler, 2004a ; Fowler *et al.*, 2002) ont montré et expérimenté que la factorisation du code est possible de façon simple et systématique en opérant méthodiquement par petites transformations validées à chaque fois par les tests des codes modifiés. Si un test n'est pas satisfait, il est très facile de revenir à la version précédente car les évolutions sont de faible ampleur.

Le remaniement doit être aussi l'occasion de scinder les codes très volumineux impliquant de nombreuses lignes. Les codes volumineux mettent souvent bout à bout différentes intentions pour réaliser une fonctionnalité majeure. La décomposition du code en fonctionnalités ou en intentions de granularité plus petite permet dans un premier temps de mieux structurer le code en le rendant ainsi plus compréhensible, plus maintenable, etc. Cette décomposition présente un second avantage qui est de mettre à disposition des autres binômes des fonctions unitaires qui auraient été noyées sans cet effort de remaniement.

III.3.7 *Programmation en binôme (Pair programming)*

Le développement en binôme est un des fondements de la méthode de conduite de projet eXtreme Programming. Ce mode de travail consiste à développer une application à deux sur un même poste de travail, l'un des développeurs jouant le rôle de *pilote* et l'autre de *copilote*. Le pilote est celui qui écrit le code et manipule les outils de développement alors que la fonction du copilote est plus stratégique. Elle consiste à assimiler et à comprendre en continue le code écrit et par voie de conséquence à critiquer au sens positif du terme l'implémentation en cours en proposant d'autres solutions, en imaginant de nouveaux tests, etc. le rôle de copilote est un rôle primordial. Bien entendu, les deux programmeurs changent de rôle plusieurs fois par jour. Le bénéfice immédiat de ce mode de travail est que l'attention des deux programmeurs est plus soutenue tout au long de la journée. Cela se traduit par un code plus clair, une conception plus robuste car elle découle des discussions/négociations entre les deux développeurs, une meilleure prise ne compte des règles de codage établies par l'équipe de programmation et par voie de conséquence une meilleure qualité du code.

Un binôme n'est pas établi à ad vitam aeternam. Les programmeurs permutent fréquemment de partenaire en fonction de leurs compétences et des aspects techniques à coder. Une recommandation forte de cette méthode est d'associer un programmeur compétent sur un type de tâche avec un collègue moins expérimenté afin de lui faire profiter de son expérience. Cette mobilité intra-équipe est un moyen pour les développeurs d'assimiler et de partager les connaissances métiers relatives à l'application, d'acquérir des compétences techniques nouvelles, d'avoir une meilleure vue d'ensemble du projet et, surtout, d'avoir le sentiment de produire une application collectivement.

Cette forme de travail collective opère un nivellement par le haut des compétences des membres de l'équipe. Elle soulève toutefois des questions quant au mode d'intervention des spécialistes -base de données, interface Homme/Machine par exemple- au sein d'une équipe. La méthode eXtreme Programming préconisent que ces spécialistes jouent le rôle de consultants.

Le travail en binôme est un mode de fonctionnement clé pour que les programmeurs s'approprient le code et s'engagent sur la *responsabilité collective du code*.

III.3.8 *Responsabilité collective du code (Collective code ownership)*

La méthode eXtreme Programming met en ligne de mire la responsabilité commune du code. En effet, en eXtreme Programming c'est l'équipe entière qui est garante du bon fonctionnement de l'application et de la qualité du code produit et non pas un individu en particulier. De ce fait, tout binôme peut intervenir pour améliorer ou remanier (refactoring) le code produit par d'autres binômes. Ainsi, le code est relu non pas seulement par le copilote mais par d'autres binômes.

Ce partage de responsabilité est d'autant mieux accepté que tout est en mis en œuvre pour que le fonctionnement de l'équipe soit optimal : programmation en binôme avec un partage optimal de la connaissance métier, règles de codage communes établies au démarrage du projet, intégration en continue du code, tests de cohérence des spécifications, etc.

III.3.9 *Intégration continue (Continuous integration)*

La pratique de l'***Intégration continue*** consiste en incorporer le code développé dans la version commune de l'application. Cette pratique permet à l'équipe de disposer d'un livrable de l'application en permanence. Ce type de fonctionnement supprime la phase d'intégration des méthodes traditionnelles, phase au cours de laquelle des anomalies ou des dysfonctionnements apparaissent souvent dus à des lacunes de spécifications.

Dans un projet eXtreme Programming, la fréquence d'intégration est au maximum journalière mais peut être horaire, cela dépend uniquement de la complexité de la fonctionnalité ou du composant. Si une fonctionnalité ou un composant n'est pas codé dans la journée, le lendemain il est scindé afin de faciliter sa réalisation.

En pratiquant l'intégration continue, les binômes mettent en continu leurs développements à la disposition des autres membres de l'équipe. Ces derniers peuvent alors interagir avec le code en effectuant des tests, en le remaniant, etc. Cette pratique contribue donc à améliorer la qualité globale de l'application.

Lors de l'intégration, les tests unitaires et de recette jouent un rôle très important car ils vont permettre de s'assurer que la cohérence du code produit localement sur une machine reste intacte dans version commune. Dans le cas contraire, le binôme, avec l'aide des autres membres de l'équipe, doit modifier et remanier le code tant que tous les tests ne sont pas positifs.

III.3.10 *Rythme durable (Sustainable pace - 40-Hour Week)*

Pour les auteurs de cette méthode, l'un des soucis majeurs du coach est que l'équipe ait un rythme de développement soutenu dans la durée. Pour ce faire, les membres de l'équipe doivent rester disponibles du point de vue intellectuel et passionnés au cours de la journée et, une fois rentrés chez eux, ils doivent être fatigués et satisfait de leur journée. On retrouve ce même souci de disponibilité et de fatigue sur le cycle hebdomadaire. Pendant le week-end, le programmeur doit penser à tout autre chose qu'à son travail pour arriver le lundi matin au bureau rempli d'énergie et d'idées. Pour Kent Beck, il est naturel que le programmeur soit fatigué en fin de semaine mais il faut aussi qu'il soit satisfait de son travail.

Pour ce faire, les auteurs de cette méthode estiment qu'une équipe doit avoir un rythme de travail optimal c'est-à-dire que les développeurs doivent disposer de suffisamment de temps pour réaliser un développement de qualité mais aussi qu'ils doivent avoir du temps pour se détendre, concession que le coach doit leur octroyer sans difficultés. Effectivement, des programmeurs fatigués ou confrontés à des problèmes -familiaux ou autres- n'ont pas la disponibilité intellectuelle pour concevoir et réaliser des codes simples, des tests adaptés, etc. Par expérience, certains auteurs (Bénard *et al.*, 2002) considèrent qu'une activité de développement soutenu de 6 h par jour est une durée au-delà de laquelle la qualité du code décline. Bien évidemment, comme le souligne Kent Beck la durée de concentration est fonction de chacun des membres de l'équipe. Certains resteront parfaitement concentrés durant 35 h par semaine et d'autres 45 h mais en aucun cas une personne pourra rester concentrée 60 h.

Ce constat sous-entend que le recours systématique à des heures supplémentaires est fortement déconseillé par les auteurs de la méthode eXtreme Programming. Si l'équipe fait souvent appel à des heures supplémentaires pour réaliser les développements dans le temps imparti, cela signifie qu'elle a un problème structurel ou fonctionnel qu'il faut d'identifier rapidement afin d'y apporter une solution. Il n'est pas interdit d'effectuer des heures supplémentaires dans la mesure où celles-ci sont focalisées sur une courte période et que la nécessité de faire des heures supplémentaires est occasionnelle.

III.3.11 *Client sur site (On-site customer)*

La pratique du ***Client sur site*** consiste à favoriser la communication entre le client et les programmeurs en accueillant en permanence le client ou son représentant au sein de l'équipe garantissant ainsi une forte réactivité et un feedback élevé. Le but principal de cette pratique est de pallier l'absence de spécifications détaillées.

La tâche principale du client est la rédaction des scénarios qui vont permettre de coder les fonctionnalités de l'application. Une seconde tâche tout aussi importante est la détermination des tests de recette que le testeur doit implémenter pour valider une fonctionnalité. Le client intervient en fixant les priorités entre fonctionnalités, en précisant les spécifications qui n'ont pas été définies auparavant ou qui sont restées floues lors des précédentes discussions, etc.

Sa présence dans l'équipe lui permet de voir le résultat immédiat de son travail de spécifications et d'apprécier l'évolution de l'application. Cette proximité lui permet aussi d'évaluer rapidement la pertinence de ses spécifications. Si le projet diverge ou prend du retard, il s'en rendra compte immédiatement.

III.3.12 *Règles de codage (Coding standards)*

Afin que les programmeurs appréhendent rapidement le code écrit par les autres membres de l'équipe, un minimum de règles de codage doit être volontairement admises et respectées par tous. Lorsque de nouveaux membres rejoignent l'équipe, le travail en binôme favorise la transmission de ces règles aux nouveaux venus.

Dans un projet eXtreme Programming, ces règles sont rarement écrites explicitement mais plutôt illustrées par des exemples dans les premiers fichiers de l'application. Ces règles concernent essentiellement la présentation du code et des commentaires mais aussi les conventions de nommage des concepts métiers ou informatiques implémentés :

- ***Présentation du code*** : la valeur de l'indentation et le type de caractères utilisés -espaces ou tabulation- pour y parvenir doivent être définis et les environnements de développement configurés pour ces paramètres.
- ***Commentaires*** : en eXtreme Programming, le code est le meilleur support des concepts et des intentions métiers ou informatiques, aussi les auteurs de cette méthode prônent le remaniement du code et en particulier la séparation des intentions plutôt qu'une documentation pléthorique et un code difficilement compréhensible ; les commentaires sont donc brefs et concis et en nombre limité.
- ***Règles de nommage*** : afin qu'un code soit rapidement appropriable par tous les membres d'une équipe, il faut que les noms des concepts informatiques soient homogènes. Selon le langage de programmation, les compilateurs ou les interpréteurs imposent des contraintes (rejet des espaces, des caractères accentués, etc.) qui obligent les programmeurs à dériver un *nom informatique* à partir de *nom thématique*. Par exemple, si plusieurs mots constituent le nom d'un concept thématique, le nom de la classe qui va représenter l'objet informatique peut être obtenu en supprimant les espaces entre mots et en mettant en lettre capitale le premier caractère de chaque mot (*NomDeClasse*). Les programmeurs ont l'habitude de ce genre d'exercice, l'important c'est que tous les membres d'une équipe adoptent la même règle.

Le respect des règles de codage est crucial pour que les programmeurs aient le sentiment de construire une application collectivement et qu'ils s'engagent leur responsabilité de façon collective.

III.4 Remarques et limites de la méthode eXtreme Programming

Comme le rappelle (Boehm *et al.*, 2004), cette méthode de conduite de projet est parfaitement adaptée à la réalisation de projets de petites tailles. Il semble toutefois qu'elle atteigne ses propres limites avec des équipes de plus de vingt personnes. Les auteurs soulignent deux autres limites : le principe ***Tu n'en auras pas besoin*** (YAGNI[119]) et la pratique de *Conception simple*.

(Beck, 2000) lui-même identifie un certain nombre de limites à sa méthode. Les plus importantes sont la séparation géographique des équipes et les cycles longs de feedback ou d'intégration. C'est parfaitement compréhensible car ces trois limites agissent directement sur la communication qui devient de ce fait plus difficile.

Lors du séminaire sur les Méthodes Agiles (CMSL, 2005), le retour d'expérience sur la méthode eXtreme Programming a montré que la collaboration entre équipes XP et équipes traditionnelles pouvait s'avérer difficile.

[119] YAGNI: You arent gonna need it.

D'après les intervenants (Carli, 2005 ; Gonnot, 2005 ; Medina, 2005), ces difficultés résultent avant tout du manque de communication entre équipes, communication qui devrait être en principe assurée par la maîtrise d'ouvrage.

Au cours de ce séminaire, Jaques Printz évoque une autre difficulté majeure pour l'exercice de sa profession d'enseignant (Printz, 2005) : *i1 ne sait pas comment construire un enseignement autour de la valeur de courage préconisée par la méthose eXtreme Programming*. Bien qu'étant présentée comme étant l'une des valeurs majeures de la méthode, les auteurs ne donnent aucune piste pour enseigner le *courage*.

Concernant l'utilisation ou non du langage UML en eXtreme Programming, la position de (Bénard *et al.*, 2002) est en opposition avec celle (Cros, 2001). Les premiers manifestent une opposition ferme à l'emploi du langage UML et à l'utilisation des patrons de conception. Ils rappellent aussi dans leur ouvrage que ces techniques ne sont pas recommandées par la méthode eXtreme Programming. A contrario, Thierry Cros justifie l'usage d'UML en argumentant autour de la communication. Il rappelle très justement que les concepteurs d'UML ont imaginé ce langage pour améliorer la communication entre acteurs participant au développement d'une application (cf. Annexe I-I). Étant donné que la communication est l'une des valeurs fondamentales de la méthode eXtreme Programming, Thierry Cros considère qu'il n'y a aucune raison de se priver de ce vecteur de communication.

Par ailleurs, discourant sur l'utopie des spécifications complètes et immuables des méthodes traditionnelles, (Bénard *et al.*, 2002) expriment un constat extrêmement fort par rapport aux outils de modélisation. Nous rapportons in extenso ces propos : *Mais ces outils de modélisation [NDLR: UML et AGL] n'apportent pas de changement réellement significatif sur le déroulement des projets de développement, et il convient de se demander si la solution à ces problèmes de spécification et de conception passe part un effort supplémentaire de modélisation... Ne serait-il pas envisageable au contraire d'adopter une approche radicalement différente qui accorde une importance moindre à cette phase initiale ?*

Les propos de la première phrase amènent à se poser la question de la pertinence des efforts de recherche effectués à travers le monde autour de la modélisation des applications informatiques et de la spécification des besoins des commanditaires. Ces propos concernent aussi les travaux de standardisation de l'OMG. La solution sous-jacente à la question qu'ils posent est qu'il faudrait réfléchir à autres modes de travail prenant davantage en compte les activités connexes à la réalisation d'une application comme la planification d'un projet, l'organisation de l'équipe de développement, les relations avec le commanditaire, etc.

Annexe III. Rappels sur les Langages, les Langages Naturels et la Théorie des Langages

Le processus de développement d'une application implique les deux grandes typologies de langages existant :

- ⇨ Les langages naturels utilisés par les individus pour exprimer et communiquer des concepts ou des pensées. Ils sont nécessaires tout au long du processus mais essentiels en phase d'analyse, phase au cours de laquelle les acteurs décrivent leur domaine et détaillent les fonctionnalités de l'application.
- ⇨ Les langages de programmation qui permettent de communiquer et de piloter les ordinateurs. Ils sont principalement utilisés en phase d'implémentation de l'application.

L'objectif de cette partie est de rappeler les concepts fondamentaux qui sous-tendent d'une part, les définitions des langages naturels et, d'autre part, la conception de langages de programmation. L'étude théorique de ces derniers relève de la *Théorie des Langages*.

Les concepts de cette annexe permettent de mieux appréhender les travaux effectués autour de l'expression des propriétés de spatialité et de temporalité des entités référencées à l'aide de pictogrammes. L'objectif visé en adoptant ce mode d'expression est d'accroître la capacité de communication des langages de modélisation existants tout en l'adaptant au domaine de l'*Information Géographique*.

Sommaire détaillé

I CONCEPTS DE LANGUE ET DE LANGAGE EN CONTEXTE GENERAL

I.1 Définitions

Les définitions relatives aux concepts de *Langue* ou de *Langage* et au concept associé de *Signe linguistique* sont extraites soit des dictionnaires *Le Petit Larousse* (Legrain *et al.*, 2002) et *Le Petit Robert* (Robert, 1984) soit de l'*Encyclopédie Libre Wikipédia* (Fondation Wikimédia, 2001), encyclopédie en ligne sur Internet.

I.1.1 *Langue*

Définition du Petit Larousse : *Système de signes verbaux propre à une communauté d'individus qui l'utilisent pour s'exprimer et communiquer entre eux.*

Définition 1 du Petit Robert : *Système d'expression du mental et de la communication, commun à un groupe social (communauté linguistique).*

Définition 2 du Petit Robert : *Système abstrait de signes, "somme des images verbales emmagasinées chez tous les individus" (Saussure).*

Définition du site Wikipédia : *Une langue est un système de signes linguistiques vocaux, graphiques ou gestuels qui permet la communication entre les individus. Une définition linguistique de la langue précise que c'est un système de signes doublement articulés, c'est-à-dire que la construction du sens se fait à deux niveaux d'articulation. On trouve tout d'abord celui des entités signifiantes (morphèmes et lexèmes ou monèmes) formant les énoncés puis celui des unités distinctives de sens (phonèmes) formant les unités signifiantes. […].*

On distingue généralement la langue (système de signes) et le langage (faculté humaine mise en œuvre au moyen d'un tel système). On distingue également, depuis Ferdinand de Saussure, la langue et la parole (c'est-à-dire l'utilisation effective du système de la langue par les locuteurs). […].

Une langue vivante est rarement un système uniforme et rigide, elle varie généralement selon le lieu géographique (dialectes), le milieu social (sociolectes) et les individus (idiolectes) et, bien sûr, selon le temps, ce qui fait que, considérée à un moment donné, une langue est toujours en évolution et contient plusieurs états. Par exemple, le système phonologique des langues est en évolution constante, ce qu'étudie la phonétique historique.

I.1.2 *Langage*

Définition 1 du Petit Larousse : *Faculté propre à l'homme d'exprimer et de communiquer sa pensée au moyen d'un système de signes vocaux ou graphiques, ce système.*

Définition 2 du Petit Larousse : *Système structuré de signes non vocaux remplissant une fonction de communication. Langage gestuel.*

Définition 3 du Petit Larousse : *Manière de parler propre à un groupe social ou professionnel, à une discipline, etc. Le langage administratif.*

Définition 4 du Petit Larousse : *Ensemble des procédés utilisés par un artiste dans l'expression de ses sentiments et de sa conception du monde. Langage de la peinture.*

Définition du Petit Robert : *Fonction d'expression de la pensée et de la communication entre les hommes, mise en œuvre au moyen d'un système de signes vocaux (parole) et éventuellement de signes graphiques (écriture) qui constitue une langue.*

Définition du site Wikipédia : *Le langage est la faculté de mettre en œuvre un système de signes linguistiques (qui constituent la langue) permettant la communication et l'expression de la pensée.*

I.1.3 *Signes Linguistiques ou Système de Signes*

Définition du site Wikipédia : *Un signe linguistique est une entité formée par la réunion d'un signifié (un concept) et d'un signifiant (une forme sonore ou image acoustique). Par exemple, le mot français arbre est un signe linguistique associant le concept d'arbre à la forme sonore /arbr/. La notion de signe linguistique a été définie par Ferdinand de Saussure dans son Cours de linguistique générale. Saussure a souligné quatre caractéristiques importantes du signe linguistique :*

- *l'arbitraire du signe : le lien entre le signifiant et le signifié est arbitraire (c'est-à-dire immotivé), car un même concept peut être associé à des images acoustiques différentes selon les langues.*
- *Le caractère linéaire du signifiant : «le signifiant, étant de nature auditive, se déroule dans le temps». Les éléments des signifiants se présentent donc obligatoirement les uns après les autres, selon une succession linéaire : ils forment une chaîne.*
- *L'immutabilité du signe : le signifiant associé à un concept donné s'impose à la communauté linguistique : un locuteur ne peut décider de le modifier arbitrairement.*
- *La mutabilité du signe : les signes linguistiques peuvent néanmoins être modifiés par le temps, par l'évolution linguistique (modification du signifiant, du signifié ou de leur rapport).*

La science étudiant les signes linguistiques est la sémiotique (ou sémiologie, selon le sens donné à ce dernier terme). Celle étudiant les concepts d'un point de vue linguistique est la sémantique, tandis que la phonétique et la phonologie décrivent l'image sonore des signes linguistiques.

I.2 Discussion

Les définitions du concept de *Langue*, énoncées par le Petit Larousse et par le site Wikipédia (cf. I.1.1), précisent deux points essentiels :

⇨ Une langue est utilisée dans un *but* d'exprimer et de transmettre d'une pensée.

⇨ La communication est réalisée au *moyen* d'un *Système de signes* appelé aussi *Signes linguistiques*.

D'après les définitions du site Wikipédia, le concept de *Langue* est le système de signes linguistiques qui permet d'échanger de l'information alors que le concept de *Langage* est la faculté des individus de mettre en œuvre le système de *Signes linguistiques* donc la langue (cf. I.1.2). Une autre information importante apportée par ces définitions concerne la forme des *Signes linguistiques* qui peut être sonore, graphique ou bien gestuelle.

Par ailleurs, la définition d'une *Signe linguistique* introduite par Ferdinand de Saussure dans son Cours de linguistique générale (cf. I.1.3) dissocie le *Concept* à exprimer ou à communiquer de la représentation matérielle utilisée pour véhiculer ce *Concept*. En linguistique, le *Concept* est qualifié de *Signifié* et la représentation matérielle de *Signifiant*.

Étant donné qu'il existe trois vecteurs de communications qui sont le son, le graphisme et le gestuel, un même *Signifié* a au moins trois *Signifiants* : un *Signifiant* sonore, un *Signifiant* graphique et un *Signifiant* gestuel.

Le premier vecteur correspond à la communication entre individus via la parole, le deuxième à la transmission d'information via l'écriture ou le dessin et le troisième constitue le principal mode de communication des muets (cf. définition 2 du concept *Langage* énoncée par le Petit Larousse).

En pratique, lorsque deux individus dialoguent en présence l'un de l'autre, ils utilisent rarement qu'un seul des trois vecteurs mais plutôt une combinaison des trois dans le but de faciliter l'assimilation des concepts par leurs interlocuteurs ce qui a pour conséquence d'augmenter le débit de la communication. La combinaison optimale des trois vecteurs dépend bien sûr du corpus de connaissance des individus, de leur aisance commune à les manipuler, de leur capacité intellectuelle, etc. Aussi, l'action de communication est une opération complexe.

Dans certaines conditions, les individus mettent en œuvre qu'un seul de ces vecteurs et la communication devient plus difficile voire laborieuse. C'est particulièrement le cas des muets dialoguant exclusivement au travers le langage des signes, langage pouvant être considéré comme instance du vecteur gestuel. C'est aussi le cas lorsqu'un touriste rédige une longue lettre pour décrire un paysage à un membre de sa famille alors qu'il serait plus simple de lui envoyer un dessin ou une photographie.

Pour souligner la complexité de la communication, il nous a semblé intéressant de rappeler qu'une langue -vivante- est rarement un système uniforme, rigide et statique (cf. définition du concept de *Langue*). Les signes linguistiques d'une langue peuvent changer de signification suivant la localisation géographique des individus introduisant des ajustements locaux que les linguistes nomment dialectes, suivant le milieu social des individus, adaptations que les linguistes désignent par sociolectes (cf. troisième définition du concept *Langage* et celle du concept *Langue* donné par le Petit Larousse), et suivant les individus eux-mêmes, variantes que les linguistes qualifient d'idiolectes.

A ces trois modes d'adaptation, il faut rajouter l'évolution temporelle d'une langue vivante où de nouveaux signifiants peuvent voir le jour pour désigner de nouveaux signifiés (concepts), des signifiés peuvent disparaître

de l'usage courant entraînant parfois la perte du signifiant et des signifiants peuvent changer de sens et être associés à d'autres signifiés. Ces mécanismes d'évolution d'une langue montrent parfaitement :

- ⇨ D'une part, le caractère arbitraire, donc immotivé, de la relation entre le signifié et le signifiant, caractéristique que Ferdinand de Saussure avait mis en exergue dans les années 1900 (cf. définition du signe linguistique ci-dessus).
- ⇨ Et, d'autre part, la mutabilité temporelle d'un signe linguistique, caractéristique que Ferdinand de Saussure avait aussi identifié.

Toute cette analyse montre que les éléments stables de la communication sont les signifiés (concepts) et non pas les signifiants (termes désignant les concepts). La conséquence de ce constat est qu'il faut, dans la mesure du possible, attacher plus d'importance au signifié d'un signe linguistique qu'à son signifiant.

II LANGUE NATURELLE ET LANGAGE NATUREL

II.1 Définitions

La définition du concept de *Langue Naturelle* est extraite des notes de cours sur la *Théorie des langages* de François Yvon et d'Akim Demaille (Yvon *et al.*, 2003) et celle de *Langage Naturel* du Dictionnaire de l'Informatique et de l'Internet (Voss, 1998).

Bien que définie dans un contexte informatique, ces deux définitions apportent un certain nombre d'idées ou de précisions d'ordre général intéressantes qui positionnent les langages naturels par rapport aux langages de programmations.

II.1.1 *Langue Naturelle*

***Extrait de Cours de François Yvon** : Par langue naturelle, on entend tout simplement les langues qui sont parlées (parfois aussi écrites) par les humains. Les langues humaines sont, à de multiples niveaux, des systèmes de symboles :*

- *les suites de sons articulées pour échanger de l'information s'analysent, en dépit de la variabilité acoustique, comme une séquence linéaire unidimensionnelle de symboles choisis parmi un inventaire fini, ceux que l'on utilise dans les transcriptions phonétiques. Toute suite de son n'est pas pour autant nécessairement une phrase articulable, encore moins une phrase compréhensible ;*
- *les systèmes d'écriture utilisent universellement un alphabet fini de signes (ceux du français sont des symboles alphabétiques) permettant de représenter les mots sous la forme d'une suite linéaire de ces signes. Là encore, si tout mot se représente comme une suite de lettres, la réciproque est loin d'être vraie ! Les suites de lettres qui sont des mots se trouvent dans les dictionnaires[120] ;*
- *si l'on admet, en première approximation, que les dictionnaires représentent un nombre fini de mots, alors les phrases de la langue sont aussi des séquences d'éléments pris dans un inventaire fini (le dictionnaire, justement). Toute suite de mots n'est pas une phrase grammaticalement correcte, et toutes les phrases grammaticalement correctes ne sont pas nécessairement compréhensibles.*

II.1.2 *Langage Naturel*

***Définition du Dictionnaire de l'Informatique et de l'Internet** : C'est le contraire d'un langage artificiel, en particulier d'un langage de programmation. Sa complexité est redoutable et, de ce fait, on considère que son traitement par ordinateur est du domaine de l'intelligence artificielle. Il a pourtant inspiré la construction des langages artificiels (certaines langues d'origine naturelle sont devenues artificielles dans une large mesure) en définissant une syntaxe (ensemble de règles pour construire des unités plus grandes à partir des plus petites) et une sémantique (attribution de catégories donnant un sens aux unités manipulées par la syntaxe). La concision du langage naturel est, du point de vue d'une machine qui l'analyse, source de nombreuses ambiguïtés : dans l'expression*

[120] Pas toutes : penser aux formes conjuguées, aux noms propres, aux emprunts, aux néologismes, aux argots…

piège à puces électronique par exemple, l'être humain comprend spontanément qu'il ne s'agit pas d'une puce électronique, mais d'un piège électronique, tandis que la machine a besoin d'une laborieuse analyse du contexte pour arriver à la même conclusion.

II.2 Discussion

Les notes de cours de François Yvon rappellent qu'une langue est constituée un système de symboles sonores et de caractères. En cela, il est en accord avec les définitions du § I.1.1 même si François Yvon a omis de citer le langage des signes. Dans le dernier item, il fait deux constats majeurs :

- ⇨ Le premier est qu'une suite de mots ne constitue pas toujours une phrase grammaticalement correcte.
- ⇨ Le second est qu'une phrase grammaticalement correcte n'est pas forcément compréhensible.

Aussi, pour qu'une phrase soit compréhensible, propriété indispensable à une bonne communication entre individus, il faut que la suite de mots soit non seulement grammaticalement correcte mais aussi compréhensible. Ces deux constats effectués sur le signifiant graphique sont transposables aux signifiants sonores ou gestuels de la langue.

La définition du *Langage naturel* proposée par le Dictionnaire de l'Informatique et de l'Internet est « amusante » puisqu'elle est énoncée en antinomie aux *Langages artificiels* dont les langages de programmation. Elle vient confirmer la nature complexe des langues et des langages, complexité dont l'une des formes est l'ambiguïté introduite, apparemment, par la recherche de concision lors d'un échange d'information. Cette l'ambiguïté est la principale source difficultés pour envisager le traitement automatique des langues naturelles et des langages naturels par ordinateur.

Ces deux définitions sont intéressants aussi car elles introduisent entre autres les concepts d'*Alphabet*, de *Mot*, de *Grammaire* et de *Syntaxe* sur lesquels repose la *Théorie des langages*.

III CONCEPT DE LANGAGE EN CONTEXTE INFORMATIQUE : THEORIE DES LANGAGES

Dans ce paragraphe nous nous intéresserons qu'au seul signifiant graphique de la communication puisque c'est le seul, pour l'instant, à être utilisé en informatique pour concevoir et produire des applications.

III.1 Rappel Historique

En premier lieu, il faut rappeler qu'un ordinateur a son propre langage (Yvon *et al.*, 2003) qui est composé de suites de zéros et de uns[121], souvent manipulés sous forme hexadécimale (Gire, 2004). Ce langage, communément appelé langage binaire, est abscons. De ce fait, il est particulièrement difficile de le mettre en œuvre pour réaliser des applications mais le plus difficile est probablement de maintenir et de faire évoluer l'application.

Très rapidement, il a été remplacé par l'assembleur, premier langage d'abstraction supérieure permettant de communiquer avec un ordinateur. En effet, la micro-instruction *MOV R0 R1* est plus compréhensible par un programmeur que la suite de chiffres hexadécimaux *0A 43* (Gire, 2004). La maintenabilité de l'application devient aussi plus facile. L'assembleur est projeté en langage binaire soit par un *compilateur* soit par un *interpréteur*, le premier effectue la traduction complète du programme en une seule opération alors que le second réalise la traduction à chaque micro-instruction (Gire, 2004 ; Pichat *et al.*, 1997).

L'assembleur présentait et présente toujours deux inconvénients majeurs. Le premier est l'absence, dans le jeu de micro-instructions de base, de certaines opérations complexes fort utiles au développement de l'application, la récursivité par exemple. Le second est lié à l'évolution proprement dite des processeurs. Ces derniers deviennent de plus en plus complexes et offre de nouvelles fonctionnalités permettant d'accroître les vitesses de calcul, de traitement de l'information, d'échanges entre le processeur et la mémoire vive, etc. Or, la mise en œuvre de ces nouvelles fonctionnalités s'effectue au travers de nouvelles micro-instructions. Aussi, il faudrait adapter continuellement un programme à chaque nouvelle famille de processeur pour optimiser son fonctionnement.

[121] Dans les années à venir, le langage binaire pourrait être remplacé par un langage à trois états (CNRS, 2005).

Pour pallier ces inconvénients et introduire des fonctionnalités de plus haut niveau d'abstraction, de nombreux langages ont été conçues (Basic, C, C++, Java, Smalltalk…) et les compilateurs ou interpréteurs associés développés.

III.2 Définitions

Les définitions de ce paragraphe on été empruntées soit aux notes de cours de (Gire, 2004), (Pichat *et al.*, 1997) et (Yvon *et al.*, 2003) soit aux dictionnaires ou à l'encyclopédie en ligne cités au § I.1 soit encore à l'ouvrage de (Kleppe *et al.*, 2003) sur l'approche *Model Driven Architecture*.

III.2.1 *Théorie des Langages*

Définition du site Wikipédia *: La théorie des langages a pour objectif de comprendre le fonctionnement des langages, vus comme moyen de communication, d'un point de vue mathématiques. Un langage est un ensemble de mots. Un mot (ou lexème) est une combinaison de signes élémentaires. L'ensemble de ces signes élémentaires est appelé alphabet. La fonction associant l'alphabet au langage est appelée grammaire. On peut associer à une grammaire un automate permettant de déterminer si un mot fait partie d'un langage. Parmi les applications pratiques de la théorie des langages, on trouve en particulier les compilateurs, en informatique.*

III.2.2 *Alphabet*

Définition de Sophie Gire *: On appelle alphabet un ensemble fini non vide* ***A*** *de symboles (lettres de un ou plusieurs caractères).*

Définition de Michèle Pichat *: Un alphabet, noté* $\mathcal{A}$*, est un ensemble non vide de symboles (ou lettres).*

Définition du Petit Larousse *: Liste de toutes les lettres servant à transcrire les sons d'une langue et énumérées selon un ordre conventionnel.*

Définition du Petit Robert *: Système de signes graphiques (lettres) servant à la transcription des sons (consonnes, voyelles) d'une langue ; série des lettres, rangées dans un ordre traditionnel.*

Définition du site Wikipédia *: Un alphabet (d'alpha et bêta, les deux premières lettres de l'alphabet grec) est un ensemble de symboles utilisé pour représenter plus ou moins précisément les phonèmes d'une langue. […]*

III.2.3 *Mot*

Définition de Sophie Gire *: On appelle un mot toute séquence finie d'éléments de* ***A****.*

Définition de Michèle Pichat *: Un mot, défini sur un alphabet* $\mathcal{A}$*, est une suite finie d'éléments de* $\mathcal{A}$*.*

Définition du Petit Larousse*(Legrain et al., 2002) : Élément de la langue constitué d'un ou de plusieurs phonèmes et susceptible d'une transcription graphique comprise entre deux blancs.*

Définition du site Wikipédia *: Dans le langage courant, un mot est une suite de caractères graphiques ou de sons formant une unité sémantique et pouvant être distingués par un séparateur (blanc typographique à l'écrit, pause à l'oral).*

III.2.4 *Langage et Langage Formel*

III.2.4.1 Langage

Définition d'Anneke Kleppe *: In this book we use the word language as a synonym for Well-Defined Language.*

Définition de Sophie Gire *: On appelle langage sur un alphabet* ***A*** *tout sous-ensemble de* ***A****.*

Définition de Michèle Pichat *: Un langage, défini sur un alphabet* $\mathcal{A}$*, est un ensemble de mots définis sur* $\mathcal{A}$*.*

Définition du Petit Robert *: Ensemble codé de signes utilisé pour la programmation de problèmes spécifiques (scientifiques, de gestion, etc.) permettant de formuler des instructions adaptées à un calculateur électronique.*

III.2.4.2 Langage Formel

Définition d'Anneke Kleppe *: A language with well-defined form (syntax) and meaning (semantics), which is suitable for automated interpretation by a computer.*

Définition du site Wikipédia *: Dans de nombreux contextes (scientifique, légal, etc.) l'on désigne par langage formel un mode d'expression plus formalisé et plus précis (les deux n'allant pas nécessairement de pair) que le langage de tous les jours (voir langage naturel). En mathématiques, logique et informatique, un langage formel est formé :*

- *d'un ensemble de mots obéissant à des règles logiques (grammaire formelle ou syntaxe) strictes.*
- *éventuellement d'une sémantique sous-jacente.*

La force des langages formels est de pouvoir faire abstraction d'une telle sémantique, ce qui rend les théories réutilisables dans plusieurs modèles. Ainsi, alors qu'un calcul particulier de paye ou de matrice inverse restera toujours un calcul de paye ou de matrice inverse, un théorème sur les groupes s'appliquera aussi bien sur l'ensemble des entiers que sur les transformations du cube de Rubik.

III.2.5 *Grammaire*

Définition de Sophie Gire *: Une grammaire est la donnée de $G=(V_T,V_N,S,P)$ où :*

- *V_T est un ensemble non vide de symboles terminaux (alphabet terminal).*
- *V_N est un ensemble de symboles non terminaux, avec $V_T \cap V_N = \varnothing$.*
- *S est un symbole terminal $\in V_N$ appelé axiome.*
- *P est un ensemble de règles de production (règles de réécritures).*

Définition de Michèle Pichat *: Une grammaire est un quadruplet G=(T,N,S,R) tel que :*

- *T est le vocabulaire terminal, i.e. l'alphabet sur lequel est décrit le langage.*
- *N est le vocabulaire non terminal, i.e. l'ensemble des symboles qui n'apparaissent pas dans les mots générés, mais qui sont utilisés au cours de la génération. Un symbole non terminal désigne une "catégorie syntaxique".*
- *R est l'ensemble des règles dites de réécriture ou de production de la forme : $u1 \rightarrow u2$ […].*
- *$S \in N$ est le symbole de départ ou axiome. C'est à partir de ce symbole non terminal que l'on commence la génération de mots au moyen des règles de la grammaire.*

Définition de François Yvon *: Une grammaire syntagmatique G est définie par un quadruplet (V,Σ,P,S), où V, Σ et P désignent respectivement des ensembles de variables (ou non terminaux), de terminaux, et de productions. V et Σ sont des ensembles finis de symboles tels que $V \cap \Sigma = \varnothing$. Les productions sont des éléments de $(V \cup \Sigma)^* \times (V \cup \Sigma)^*$, que l'on note sous la forme $\alpha \rightarrow \beta$. S est un élément distingué de V, appelé le symbole initial ou encore l'axiome de la grammaire.*

Définition du Petit Larousse *: Ensemble des règles phonétiques, écrites et orales d'une langue ; étude et description de ces règles.*

Définition du Petit Robert *(Robert, 1984) : Ensemble des règles à suivre pour parler et écrire correctement une langue.*

Définition 1 du site Wikipédia *: La grammaire est l'étude des règles qui régissent une langue permettant de construire des énoncés reconnus corrects par les locuteurs natifs d'une langue donnée. Elle comporte plusieurs disciplines : phonétique, phonologie, morphologie, syntaxe et sémantique.*

Définition 2 du site Wikipédia *: La linguistique et l'informatique utilisent la notion de grammaire formelle, qui précise les règles de syntaxe d'un langage.*

III.3 Discussion

Le but de la *Théorie des langages* (cf. III.2.1) est d'étudier, sous l'angle mathématique, les langages permettant à un utilisateur de communiquer avec un ordinateur. Cela sous-entend la définition et la mise en œuvre d'un système de signes linguistiques (cf. I) qui constituent l'*Alphabet* du *Langage* et dont les combinaisons forment des *Mots* organisés en « phrases » respectant une *Grammaire*.

Au travers du formalisme mathématique, cette discipline cherche à abstraire les langages afin d'une part, d'améliorer le dialogue homme/machine et, d'autre part, de les rendre interprétables automatiquement par les ordinateurs. Les principaux produits autour de ces travaux sont les compilateurs ou les interpréteurs.

Le concept d'*Alphabet* regroupe tous les symboles, lettres ou caractères, utilisés pour matérialiser une information dans un *Langage* (cf. III.2.2). La formulation mathématique des deux définitions énoncées par Sophie Gire et de Michèle Pichat sont équivalentes. Il est à noter que, d'après ces définitions, le système de signes linguistiques n'est pas limité aux seules lettres de l'alphabet latin des langues d'Europe Occidentale. Bien qu'établis dans un contexte général, les trois dernières définitions ne sont pas en contradiction avec les deux premières définitions plus scientifiques.

Le concept de *Mot* est un premier niveau d'organisation qui associe un certain nombre fini de caractères graphiques dans une séquence bien précise (cf. III.2.3). Comme cela était prévisible, les définitions formulées par Sophie Gire et Michèle Pichat sont équivalentes puisqu'elles s'appuient sur le même formalisme mathématique. Sans surprise aussi, les définitions du Petit Larousse et du site Wikipédia, formulées dans un contexte général, sont en accord avec les précédentes. Néanmoins, elles introduisent l'existence d'un signe linguistique particulier qui fait office de séparateur entre les mots. Dans la plupart des langues d'Europe Occidentale, ce dernier est le caractère *Blanc*.

En faisant un rapprochement avec les concepts linguistiques, un *Mot* est le signifiant graphique d'un signifié ou concept.

Nous ne nous attarderons pas sur la définition du concept de *Langage* énoncée par Anneke Kleppe qui renvoie à celle du concept de *Langage formel* (cf. III.2.4.1). Hormis cette définition, les trois autres sont équivalentes, puisque, selon Sophie Gire, ***A**** est l'ensemble infini contenant tous les mots qu'il est possible de construire à partir de l'alphabet ***A***.

L'intérêt de la définition du Petit Robert est d'introduire le concept d'*Instruction* équivalent informatique de celui de phrase en linguistique. Le concept d'*Instruction* est un niveau d'organisation des *Mots* via des règles logiques strictes, souvent qualifiées de règles syntaxiques. Elles constituent la *Grammaire* du langage (cf. III.2.4.2 et III.2.5).

Les définitions d'Anneke Kleppe et du site Wikipédia autour du concept de *Langage formel* (cf. III.2.4.2) mettent en exergue la nécessité pour un *Langage* de posséder une sémantique et une syntaxe stricte afin que son interprétation puisse être automatisée par un ordinateur.

Il est important de souligner qu'en contexte informatique, les définitions du concept de *Langage* sont identiques en intention à celles du concept *Langue* en contexte général. En effet, les deux concepts, chacun dans sont contexte, font référence à un ensemble de signes. Dans ses notes de cours, Sophie Gire précise que le concept de *Langue* est plutôt réservé à la communication entre individus et que celui de *Langage* est plutôt dédié à la communication entre un individu et un ordinateur.

Bien que de nature différente (généraliste ou mathématique), toutes les définitions du concept de *Grammaire* (cf. III.2.5) font référence à un ensemble de règles syntaxiques qui imposent une structuration et un ordonnancement des mots (symboles, vocabulaire ou variables) au sein des instructions.

Les définitions plus scientifiques de Sophie Gire, de Michèle Pichat et de François Yvon introduisent des concepts d'*Axiome* et d'ensembles de *Symboles terminaux* et *non terminaux*. Cette terminologie, très hermétique pour les non initiés, désigne en fait des concepts familiers lorsqu'ils sont appliqués au langage naturel.

Pour illustrer la signification de cette terminologie, Sophie Gire présente, dans ses notes de cours, l'exemple d'une grammaire $G=(V_T,V_N,S,P)$ où :

⇨ V_T, ensemble des symboles terminaux, est constitué des mots :

- V_T = {il, elle, parle, est, devient, court, reste, sympa, vite}.

⇨ V_N, ensemble des symboles non terminal, est composé des mots :

- V_N = {PHRASE, PRONOM, VERBE, COMPLEMENT, VERBETAT, VERBACTION}.

⇨ S, l'axiome de la grammaire, est le symbole non terminal :

- S = PHRASE.

⇨ P, ensemble de règles de production, est défini par :

- P = { PHRASE → PRONOM VERBE COMPLEMENT

PRONOM → il | elle
VERBE → VERBETAT | VERBACTION
VERBETAT → est | devient | reste
VERBACTION → parle | court
COMPLEMENT → sympa | vite }

L'analyse des mots impliqués dans cet exemple est très intéressante. Elle montre d'une part que les symboles terminaux constituent l'information à transmettre (sympa, court, etc.) au moyen du concept de PHRASE, axiome de la grammaire et, d'autre part, que les symboles non terminaux sont les concepts structurels (PRONOM, COMPLEMENT, etc.) de l'axiome. Enfin, six règles de production définissent le contenu ou l'organisation des symboles non terminaux et tout particulièrement l'organisation de l'axiome.

La représentation ensembliste de la figure 156 illustre les deux ensembles V_T et V_N de symboles terminaux et respectivement non terminaux. Elle montre aussi les sous-ensembles de symboles terminaux, appelés ***catégories syntaxiques*** par Michèle Pichat, résultant de l'analyse des règles de production.

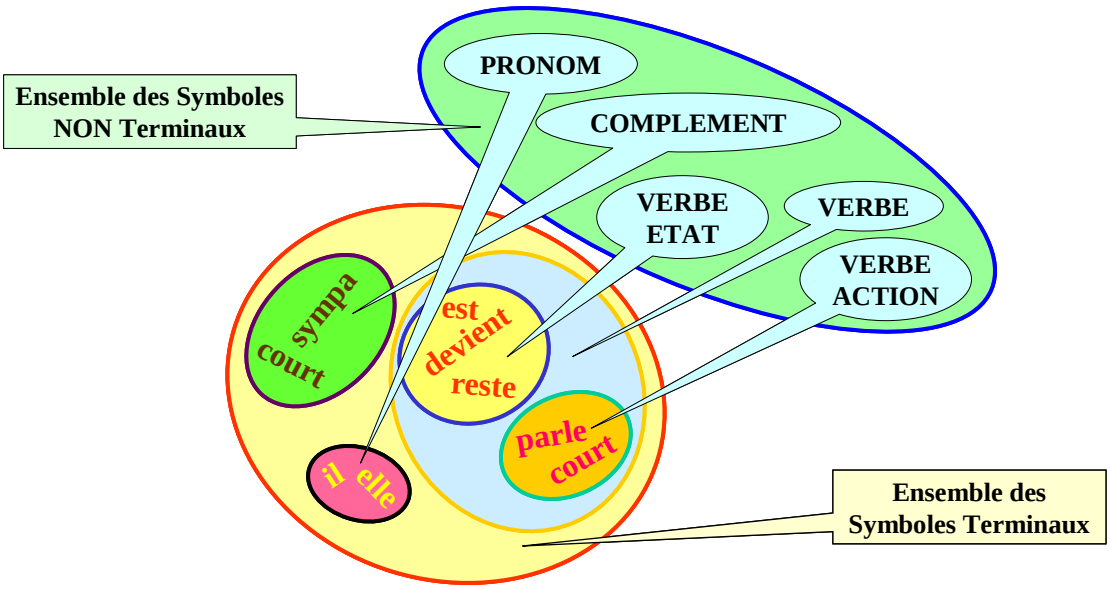


Figure 156 Représentation des ensembles des symboles terminaux et non terminaux.

La première et la troisième règle définissent l'organisation des catégories syntaxiques entre elles. Dans l'exemple ci-dessus, une phrase est constituée de la succession d'un pronom, d'un verbe et d'un complément et un verbe peut être soit un verbe d'état ou un verbe d'action. Les quatre autres règles définissent les symboles terminaux de chacun des catégories syntaxiques.

La grammaire G de cet exemple permet de construire seulement vingt phrases syntaxiquement correctes dont certaines sont cohérentes du point de vue sémantique et d'autres sont totalement incohérentes. Par exemple, la phrase *il parle vite* a un sens que tout un chacun peut comprendre alors que la phrase *elle court sympa* est sémantiquement incorrecte.

Les phrases sémantiquement incorrectes ne peuvent pas être identifiées par l'analyse syntaxique mais par l'analyse sémantique complémentaire à la première.

IV EN RESUME

En tout premier lieu, il faut rappeler qu'un langage est un vecteur de communication entre des individus ou entre un individu et une machine. Il permet de véhiculer des signifiés (concepts) via une combinaison de signifiants sonore, graphique ou gestuel.

Les langages naturels varient au grès de la localisation géographique des communautés les utilisant, des groupes sociaux ou socioprofessionnels et des individus eux mêmes. Entre outre, ils évoluent dans le temps :

- ⇨ Certains concepts sont abandonnés car ils n'ont plus cours.
- ⇨ D'autres naissent en concomitance en particulier de l'avancée des technologies ou des sciences.
- ⇨ D'autres, enfin, changent de signifié ou de signifiant sous la pression de certains groupes sociaux ou socioprofessionnels.

Les variantes géographiques ou sociales et les mutations temporelles ont rendu les *Langages naturels* complexes, ambigus et difficilement exploitables pour établir un échange d'information fiable avec les machines et tout particulièrement avec les ordinateurs.

La technologie des lampes et des semi-conducteurs ensuite a permis la création et le développement de l'industrie informatique mais elle a imposé le langage binaire[121] comme langage de communication interne des ordinateurs. Ce langage reste pour l'instant le seul le langage compréhensible par les ordinateurs. La difficulté de s'approprier et de maîtriser le langage binaire et celle de maintenir un programme développé en langage binaire ont amené les chercheurs à définir des langages de plus haut niveau d'abstraction plus faciles à maîtriser et à mettre en œuvre.

Actuellement, un certain nombre de chercheurs effectue des recherches sur des langages graphiques voire sur des langages pictogrammiques. Le langage UML (OMG, 2005, 2006) est un bel exemple de la première catégorie et le langage pictogrammique de Perceptory (Bédard, 1999b ; Bédard *et al.*, 2004) pour la modélisation de bases de données spatio-temporelles fait partie de la seconde catégorie.

Dans l'immédiat, ces travaux n'incluent pas la génération de code binaire mais c'est une perspective qui n'est pas impossible à atteindre à moyen terme.

Annexe IV. Dualité entre les concepts UML d’attribut et d’association

Le but de cette annexe est de rappeler la proximité conceptuelle existant entre le concept UML d’*Attribut* et celui d’*Association*. De ce fait, ces deux concepts peuvent être utilisés indifféremment pour modéliser un concept thématique.

La différence essentielle entre ces deux concepts réside dans leurs représentations graphiques dont l’une est sémantiquement un peu plus riche et facilite par la même la lecture du modèle car cette sémantique supplémentaire contient des informations précieuses qui fluidifient la communication entre les acteurs du domaine et le concepteur.

La dualité entre ces deux concepts a permis de concevoir et d’implémenter des transformations automatisant la conversion d’un *Attribut* en *Association* et inversement.

Sommaire détaillé

I CONCEPTS UML D'ATTRIBUT ET D'ASSOCIATION : DIFFERENCE CONCEPTUELLE ?

(Fowler *et al.*, 2001b) affirment qu'il n'y a pas de différence conceptuelle entre le concept d'*Attribut* et celui d'*Association*. Cette affirmation transparaît aussi dans les propos de Craig Larman (Larman, 2002a) lorsqu'il conseille de réserver l'utilisation du concept d'*Attribut* aux seuls concepts dont le type de donnée est simple (booléen, date, nombre, chaîne, heure) et de modéliser les concepts complexes par des *Association*. (Muller *et al.*, 2000) abondent dans ce sens en qualifiant de pseudo-attribut le concept UML de *Rôle* qui est l'un des concepts de décoration d'une association.

Suite à ces affirmations, la comparaison de la modélisation d'un concept sous forme d'*Attribut* et sous forme d'*Association* s'imposait.

L'utilisation du concept d'*Attribut* dans les figures 157 et 158 pour indiquer qu'un aéroport a un nom est aisément compréhensible par une grande majorité d'acteurs.

Par contre, la modélisation des escales effectuées par un avion sous forme d'*Attribut* (cf. figure 157) s'avère moins évidente. Selon Craig Larman, *le concept d'aéroport est un concept complexe qui occupe plusieurs kilomètres carrés*. Ce n'est pas la seule raison qui préside ce choix : un aéroport est un concept complexe composé de concepts eux-mêmes complexes tel que la tour de contrôle, les pistes d'atterrissage, les aérogares, etc. De plus, un aéroport est une entité ayant une dynamique propre et des évolutions dans le temps. C'est sa composition et sa dynamique qui font qu'un aéroport un concept complexe.

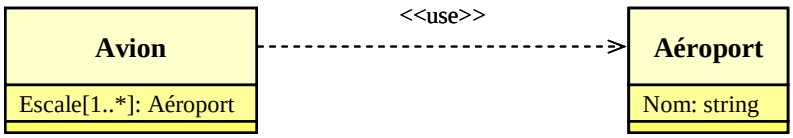


Figure 157 Modélisation sous forme *d'Attribut*.

Il est vrai que la modélisation du concept d'*Aéroport* via une *Association* (cf. figure 158) est plus facile à comprendre pour la plupart des acteurs car elle est sémantiquement plus riche et précise mieux d'une part, la nature de la relation (*Vol*) qui relie le concept d'*Avion* à celui d'*Aéroport* -information absente du modèle précédent- et, d'autre part, les rôles joués par les deux concepts.

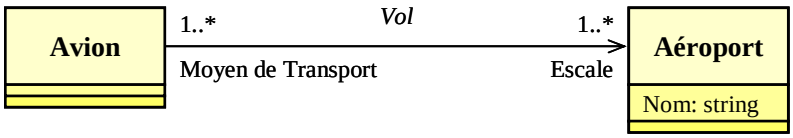


Figure 158 Modélisation sous forme d'*Association.*

L'analyse des figures 157 et 158 montre que les concepts d'*Attribut* et d'*Association* sont très proches l'un de l'autre et ce qui les différencie est un complément de sémantique pour la représentation en *Association*. L'affirmation de Martin Fowler est confirmée par l'analyse de ces deux modèles et donne la perception d'une dualité entre les concepts d'*Attribut* et d'*Association*.

Cette dualité est encore plus palpable en cours de modélisation car il n'est pas rare que le concepteur matérialise un concept thématique via un *Attribut* et qu'ensuite, au vu de sa complexité, il le transforme en *Association*. Inversement, il va convertir un concept thématique initialement matérialisé sous forme d'*Association* en *Attribut* si les propriétés du concept thématique qui sont retenues sont des propriétés simples.

Implicitement, le concepteur effectue des transformations de modèle et il devra soit supprimer des informations existantes ce qui ne pose a priori pas de problème majeur, soit il devra compléter certains points de décoration ce qui est plus délicat.

L'existence de cette dualité et les pratiques des modélisateurs nous ont conduits à concevoir et à implémenter des transformations automatisant la conversion de ces deux concepts. Elles font l'objet des deux paragraphes suivants où les mécanismes de conversion sont décrits via des exemples.

II TRANSFORMATION D'UN ATTRIBUT EN ASSOCIATION

La transformation réalisant la conversion d'un attribut en association sera désignée par $T_{Att\ To\ Ass}$. Elle est en charge de transformer automatiquement l'attribut *Escale* de la classe *Avion* en une association (cf. figure 159). L'attribut *Escale* possède trois informations majeures : son nom *Escale*, sa multiplicité *1..** et son type *Aéroport*. Ces informations représentent un capital de connaissances à conserver au cours de la transformation et elles doivent être reportées dans le nouveau modèle.

La comparaison du concept de *Rôle* à un pseudo-attribut par (Muller *et al.*, 2000) désigne le concept de *Rôle* comme le meilleur candidat pour porter le nom de l'attribut. La multiplicité de l'attribut est tout naturellement transposée vers celle de l'extrémité de l'association.

Fort de ces constats, la conception de la transformation $T_{Att\ To\ Ass}$ devient très simple. Nous lui avons assignée la responsabilité d'établir une relation d'association entre la classe *Avion* et la classe *Aéroport*, classe typant l'attribut. Après, elle nomme *Escale* le rôle de la classe *Aéroport* et lui attribue la multiplicité *1..**. Ensuite, elle ajoute une navigabilité à l'extrémité de l'association portée par la classe *Aéroport*. Enfin et ce dans un souci de cohérence du modèle, l'attribut *Escale* est supprimé ainsi que, s'il existe, le lien de dépendance *<<Use>>* entre la classe *Avion* et la classe *Aéroport*.

À ce stade, toutes les informations contenues dans l'attribut *Escale* sont reprises dans la nouvelle représentation. Elles suffisent à la plupart des générateurs de code projetant le modèle dans un langage programmation objet. Par contre, elles ne suffisent pas pour un générateur de code SQL.

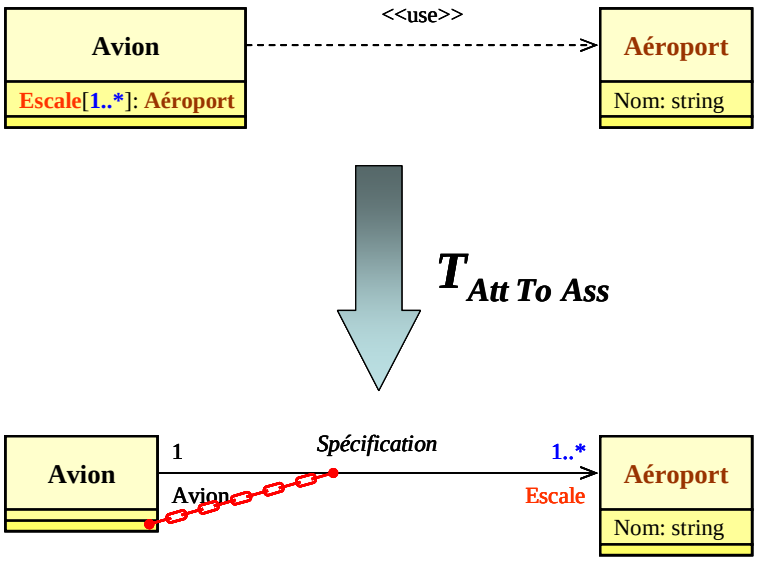


Figure 159 Exemple illustrant la Transformation $T_{Att\ To\ Ass}$.

Cependant, en termes de modélisation, le modèle obtenu est sémantiquement incomplet. Aussi, même si l'ajout systématique d'information s'avère toujours délicat, la transformation $T_{Att\ To\ Ass}$ complète le modèle conformément aux recommandations de l'Annexe I-III.4.1 : l'association est systématiquement nommée *Spécification*, le nom *Avion* de la classe est attribué au nom du rôle de la classe *Avion*, la multiplicité de cette extrémité est valuée à *1* puisqu'il n'y a qu'un seul attribut.

Le modèle résultant n'a pas la qualité sémantique d'un modèle effectué par un concepteur comme le montre les deux phrases :

⇨ Un *Avion* est en relation de *Spécification* avec *1 à n Aéroport(s)* qui joue le rôle d'*Escale(s)*.

⇨ Un *Aéroport* est en relation de *Spécification* avec *1 Avion* qui joue le rôle d'*Avion.*

dont la construction est recommandée en Annexe I-III.4.1. Ces phrases ne sont pas sémantiquement ***parfaites*** mais elles restent cohérentes et compréhensibles par des acteurs.

Pour finir et ce dans une perspective d'implémenter la transformation inverse, la transformation $T_{Att\ To\ Ass}$ relie la classe *Avion* à la nouvelle association par un lien de traçabilité. Ce dernier permet de distinguer sans ambiguïté les associations créées par la transformation $T_{Att\ To\ Ass}$ de celles établies manuellement qui n'ont pas de lien de traçabilité. Ce lien constitue une information pertinente indiquant l'origine de l'association. La disposition du lien a aussi été réfléchie afin d'identifier sans ambiguïté la classe de rattachement de l'attribut[122].

Bien que le modèle final ne soit pas sémantiquement idéal, le plus important est d'offrir au concepteur d'une part, une fonctionnalité lui permettant d'alterner entre les deux formes de modélisation automatiquement et, d'autre part, la possibilité d'intervenir sur le modèle manuellement une fois la transformation réalisée. Le concepteur améliore alors la sémantique et la qualité du modèle. Dans ce cas de figure, la transformation inverse doit conserver les modifications manuelles car elles constituent un capital à ne pas dilapider.

III TRANSFORMATION D'UNE ASSOCIATION EN ATTRIBUT

La transformation $T_{Ass\ To\ Att}$ ne présente quant à elle aucune difficulté conceptuelle puisqu'elle crée un attribut (cf. figure 160) *Escale* ayant une multiplicité *1..** qui sont respectivement le nom et la multiplicité de la classe *Aéroport* dans l'association *Vol*. Le type du nouvel attribut est bien entendu la classe *Aéroport*. Pour finir, la transformation supprime l'association *Vol* et établit un lien de dépendance stéréotypé <<*use*>> entre les deux classes.

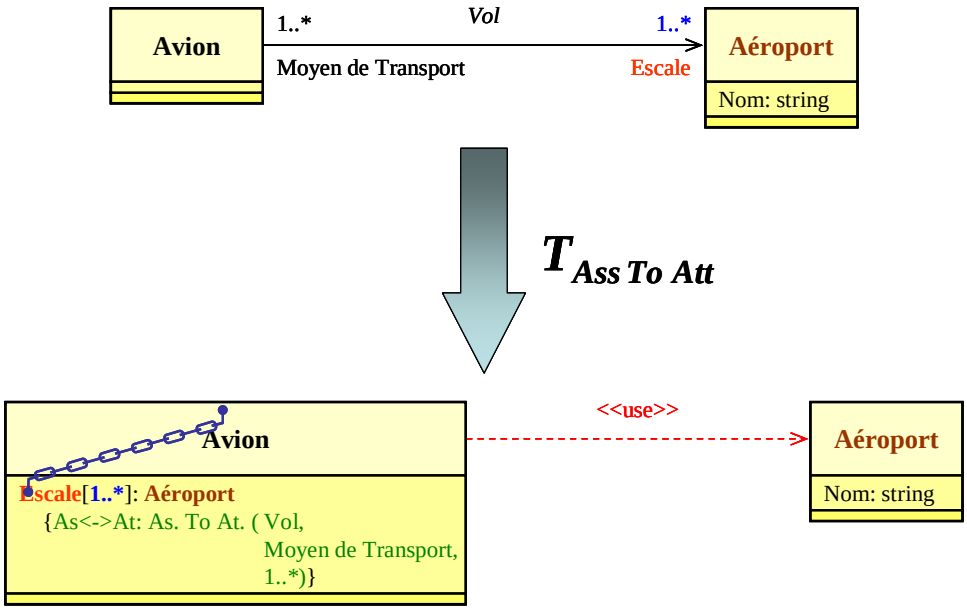


Figure 160 Exemple illustrant la Transformation $T_{Ass\ To\ Att}$.

[122] La navigabilité ne nous semble pas être une information suffisamment fiable pour renseigner la classe de rattachement de l'attribut car rien n'interdit au concepteur de modifier la navigabilité une fois la transformation effectuée.

Au cours de cette transformation, le nom de l'association, le rôle et la multiplicité du concept *Avion* sont superflus. Ce n'est pas parce que ces informations sont superflues qu'elles ne sont plus utiles. Elles constituent un capital de connaissance à ne pas perdre qui rendrait impossible la transformation inverse.

Pour pallier ce problème, la transformation $\boldsymbol{T_{Ass\ To\ Att}}$ ajoute, au nouvel attribut *Escale*, une valeur marquée (*{As<-> : As. To At.(Vol, Moyen de Transport, 1..*)}*) contenant les trois informations à conserver.

Bien qu'il ne soit pas indispensable à la transformation inverse[123], notre souci d'homogénéité avec la transformation précédente nous a conduits à établir un lien de traçabilité entre le classe *Avion* et l'attribut *Escale* permettant ainsi d'identifier plus facilement et sans ambiguïté les attributs résultant de la transformation $\boldsymbol{T_{Ass\ To\ Att}}$.

IV CONCLUSION

Les transformations $\boldsymbol{T_{Att\ To\ Ass}}$ et $\boldsymbol{T_{Ass\ To\ Att}}$ offrent la possibilité d'alterner automatiquement entre une modélisation attributaire et une modélisation en association d'un concept thématique. La conséquence directe est une souplesse beaucoup plus grande au cours du processus de modélisation rendant ainsi moins crucial le respect des recommandations de Craig Larman puisque à tout moment le concepteur peut alterner entre les deux représentations.

Ces deux transformations sont totalement génériques et à ce titre elles peuvent être mises œuvre sur n'importe quel modèle UML. Elles peuvent ainsi être appliquées aux différents sous-modèles de l'artefact *Software Development Process Model*. Dans ce contexte, elles seront soit des transformations PIM/PIM, soit des transformations PSM/PSM selon le statut du sous-modèle sur lequel elles sont appliquées.

Il faut signaler que la dualité entre les concepts d'*Attribut* et d'*Association* est utilisée depuis longtemps par les générateurs de code des langages de programmation objet (Java, C++…). Ces générateurs transforment implicitement les associations en attributs suivant le principe de la transformation $\boldsymbol{T_{Ass\ To\ Att}}$ à la différence près que le lien de dépendance, la valeur marquée et le lien de traçabilité ne sont instanciés.

[123] Doté de la valeur marquée *{As<-> : As. To At.(…)}*, l'attribut contient l'information suffisante pour reconstruire le modèle initial.

Annexe V. Description succincte du générateur de code SQLDesigner de l'atelier de génie logiciel Objecteering

L'objectif de cette annexe est de rappeler le processus de projection d'un modèle UML en code SQL avec l'atelier de génie logiciel Objecteering, code qui permet ensuite de créer la base de données relationnelle conforme au modèle.

La compréhension du processus de projection permet de mieux saisir la conception des transformations SQL adaptant le sous-modèle d'implémentation SQL au Générateur de Code SQLDesigner (cf. Chapitre 5-IV).

Le rappel effectué dans cette annexe n'est pas exhaustif car il est limité aux seules spécifications d'implémentation SQL utilisées dans les Transformations SQL du paragraphe IV du Chapitre 5.

Sommaire détaillé

I LES ETAPES DE LA GENERATION DE CODE SQL

Le module SQLDesigner est un générateur de code qui projette un diagramme de Classe UML en code SQL, code qui permet ensuite de créer une base de données relationnelle. Pour ce faire, la société Softeam a étendu le métamodèle de l'atelier de génie logiciel en y introduisant les concepts du langage SQL. Le concepteur dispose des concepts SQL aux travers d'annotations. Il introduit ces dernières dans les concepts thématiques du *modèle d'analyse* (AM), *aussi appelé modèle logique* (Softeam, 2003b). Une première transformation ($T^{AM \rightarrow PM}$) *génère le modèle physique* (PM) (cf. figure 161) qui, à son tour, peut être annoté afin de définir certains paramètres de la génération de code SQL.

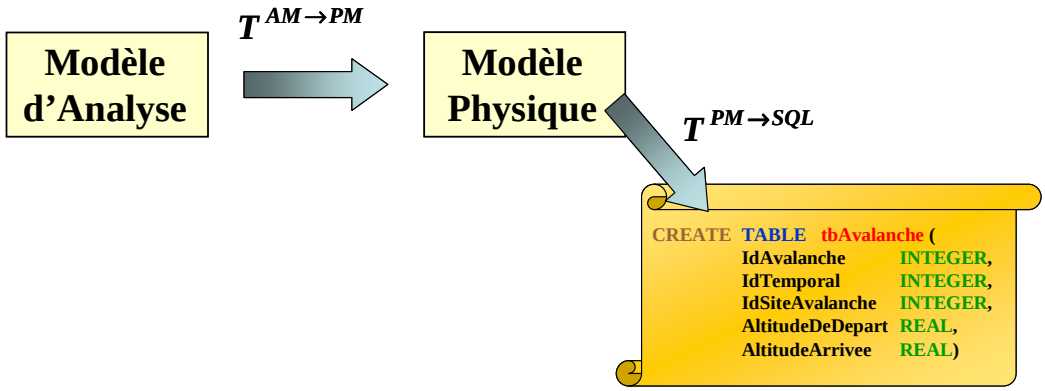


Figure 161 Les étapes de modélisation et de génération d'une base de données relationnelle (Softeam, 2003b).

La génération proprement dite est effectuée par la transformation $T^{PM \rightarrow SQL}$ qui applique les règles du Langage de Définition des Données[124] pour projeter le modèle physique en code SQL. La figure 162 illustre un exemple de règle de projection. Le concept UML de *Classe* est converti en *Table*, le concept d'*Attribut* en *Colonne* et le concept d'*Instance* correspond à un *Tuple*.

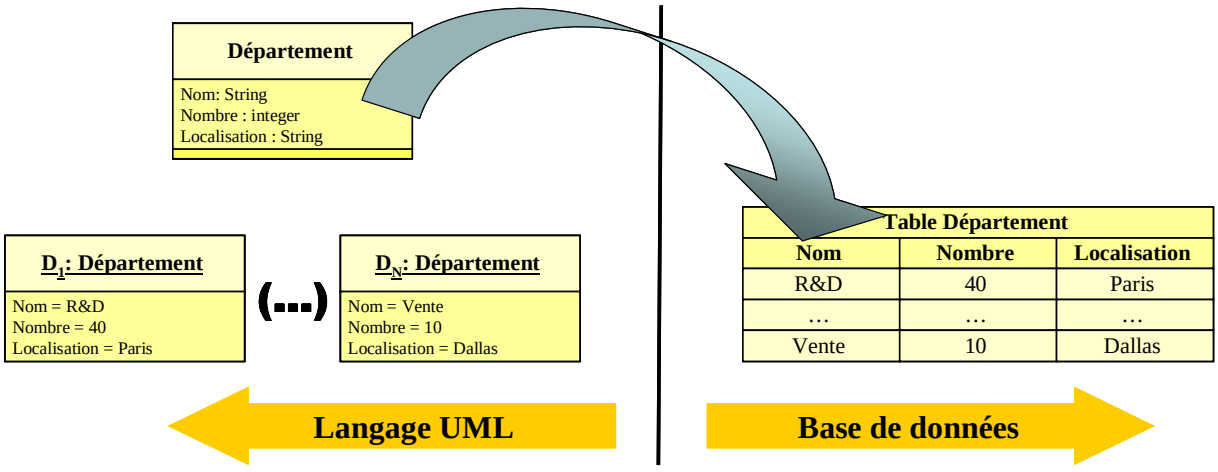


Figure 162 Exemple de règle de projection Langage de Définition des Données (Softeam, 2003b).

Ainsi, la classe *Département* du modèle d'analyse devient la table *Département* dans la base de données, les attributs *Nom*, *Nombre* et *Localisation* de la classe sont les colonnes de la table et le contenu des instances $D_1 \ldots D_N$ de la classe sont les valeurs des tuples de la table.

[124] LDD ou en anglais DDL pour Data Definition Language.

Le générateur de code SQLDesigner produit le code SQL permettant de créer la table *Département*, les colonnes *Nom*, *Nombre* et *Localisation* en tenant compte du type et du format des variables (cf. figure 163). Il crée aussi le code SQL de suppression des tables et des principales contraintes d'intégrité.

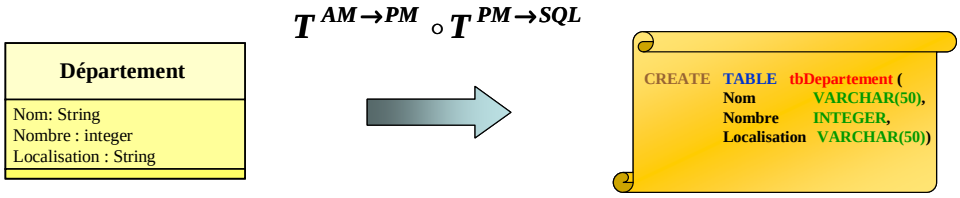


Figure 163 Code SQL issu de la classe *Département* (Softeam, 2003b).

Malheureusement, dans la version actuelle, la règle de projection du concept UML d'*Instance* n'a pas été implémentée dans le générateur de code SQLDesigner.

Pour que la transformation $T^{PM \to SQL}$ puisse s'exécuter correctement, le modèle physique doit contenir un certain nombre de spécifications de nature SQL sans lesquelles la transformation n'est pas possible.

Dans le paragraphe suivant, nous rappelons succinctement les principales spécifications SQL indispensables à la transformation $T^{PM \to SQL}$.

II SPECIFICATIONS SQL MAJEURES

II.1 Persistence

Les bases de données ont été conçues pour réaliser le service de persistence des données. Les tables étant un des éléments d'une base de données, elles sont elles-mêmes persistantes. Le concept de *Table* résultant de la projection du concept de *Classe*, il est naturel que ce dernier ait une spécification de persistence afin que la transformation $T^{PM \to SQL}$ s'effectue.

La spécification de persistance est introduite dans la classe par une valeur marquée de type *persistence* propre au générateur de code SQLDesigner. Cette valeur marquée possède un paramètre qui indique le type de la persistence *persistent* ou *transient* (cf. figure 164).

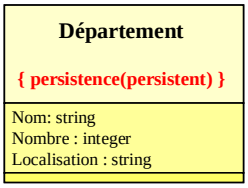


Figure 164 Exemple d'annotation de persistance.

II.2 Clé primaire

Les classes persistantes ayant des relations d'association, d'agrégation ou de composition avec d'autres classes doivent impérativement posséder un attribut faisant office de clé primaire. Dans la pratique, le concepteur décide en relation avec les acteurs du domaine de l'attribut qui jouera ce rôle.

Afin d'indiquer l'attribut ayant cette fonction, le concepteur a à sa disposition la valeur marquée *{primaryKey(index)}* qu'il aura à ajouter (Softeam, 2003b). Lorsque plusieurs attributs sont nécessaires pour constituer une clé primaire, l'indice de l'attribut dans la composition de la clé primaire est indiqué par le paramètre *index*. En figure 165, ce paramètre a la valeur 1.

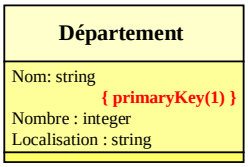


Figure 165 Exemple d'annotation de clé primaire.

II.3 De généralisation/spécialisation : Techniques de projection de la relation

Le mécanisme généralisation/spécialisation n'étant pas implémenté dans de nombreux Système de Gestion de Bases de Données Relationnelles[125], le générateur de code SQLDesigner propose au concepteur trois techniques pour pallier cette carence (Softeam, 2003b). Le concepteur doit choisir une de ces techniques et cette information doit être introduite sous forme de valeur marquée dans la ***classe racine*** de la hiérarchie de généralisation. Ensuite, la transformation $T^{AM \rightarrow PM}$ interprète cette information pour réaliser la projection en code SQL.

II.3.1 *Technique Une Table par Classe Concrète*

La première technique différencie les classes abstraites des classes concrètes (non abstraites). Seules ces dernières sont converties en tables dans le modèle physique. Qu'elles soient abstraites ou concrètes, les attributs des classes mères sont recopiés dans les classes filles par la transformation $T^{AM \rightarrow PM}$. Pour appliquer cette technique, le concepteur doit apposer la valeur marquée *{oneTablePerConcreteClass}* sur la classe racine de la hiérarchie de classes.

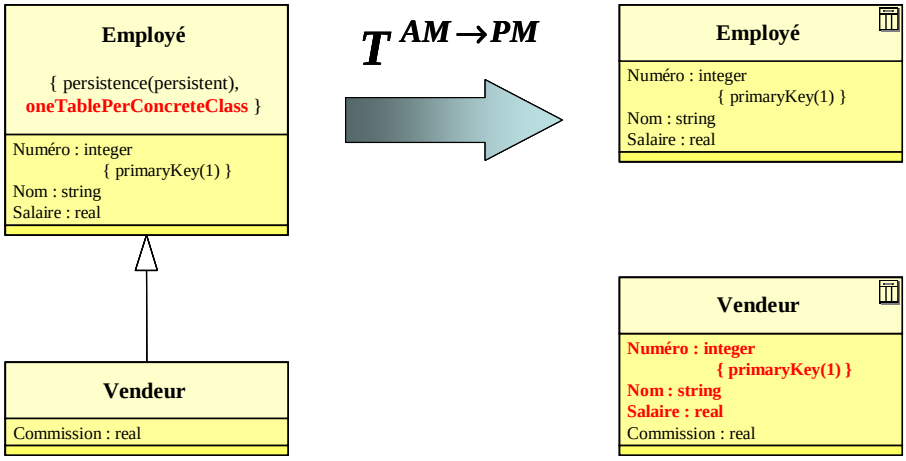


Figure 166 Illustration de la Transformation $T^{AM \rightarrow PM}$ pour la technique *une table par classe concrète* (Softeam, 2003b).

II.3.2 *Technique Une Table par Classe*

La seconde technique ne fait aucune distinction entre les classes abstraites et les classes concrètes. La transformation $T^{AM \rightarrow PM}$ crée systématiquement une table par classe dans le modèle physique (cf. figure 167). Cette transformation recopie la clé primaire de la classe racine dans toutes les classes filles et établie une contrainte d'intégrité référentielle entre la clé primaire d'une classe et celle de sa classe mère. Cette technique est effectuée par le générateur de code SQLDesigner lorsque la valeur marquée *{oneTablePerClass}* annote la classe racine.

[125] Il est à signaler que PostgreSQL 8.0 possède le mécanisme d'héritage.

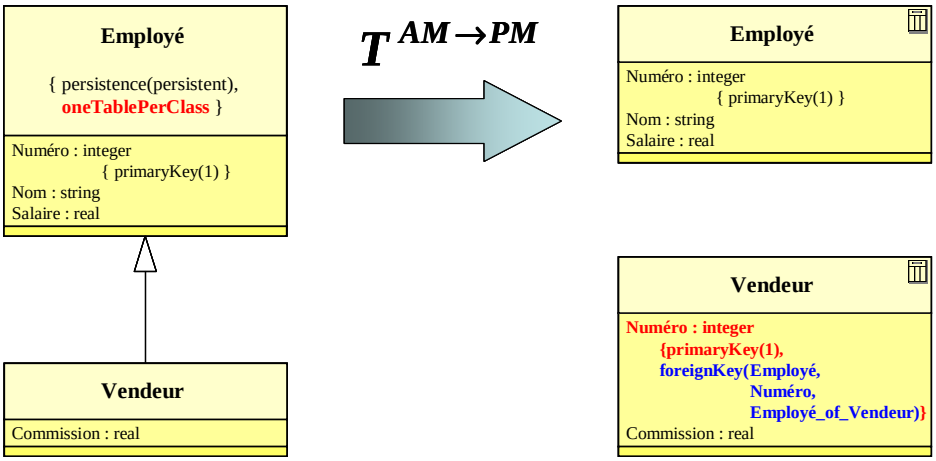


Figure 167 Illustration de la Transformation $T^{AM \rightarrow PM}$ pour la technique *une table par classe* (Softeam, 2003b).

II.3.3 *Technique Une Seule Table*

La troisième technique fusionne toutes les classes de la hiérarchie en une seule classe en regroupant les attributs des classes filles au sein de la classe racine (cf. figure 168). En outre, la transformation $T^{AM \rightarrow PM}$ ajoute un attribut supplémentaire (*Type*) qui permet de préciser la typologie de l'objet. C'est la présence de la valeur marquée *{oneTable}* sur la classe racine qui déclenche cette option.

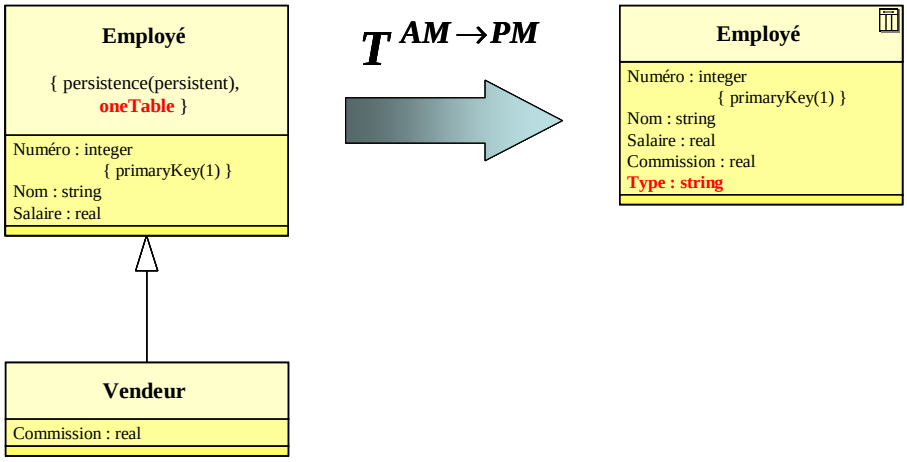


Figure 168 Illustration de la Transformation $T^{AM \rightarrow PM}$ pour la technique *une seule table* (Softeam, 2003b).

III INCONVENIENTS INDUITS PAR LA TRANSFORMATION $T^{AM \rightarrow PM}$

III.1 Perte des relations

Les classes dénuées attributs et possédant une ou plusieurs opérations (cf. classe Polygone - figure 169-sous-modèle d'implémentation SQL) sont considérées par la transformation $T^{AM \rightarrow PM}$ comme étant des procédures stockées. Au cours cette transformation elles sont :

- D'une part, automatiquement annotées avec le stéréotype <<*procedureClass*>>.
- D'autre part, les relations d'association et de généralisation qu'elles ont avec d'autres classes sont systématiquement supprimées (cf. classe Polygone - figure 169-modèle physique MPD_SQL Implementation SubModel).

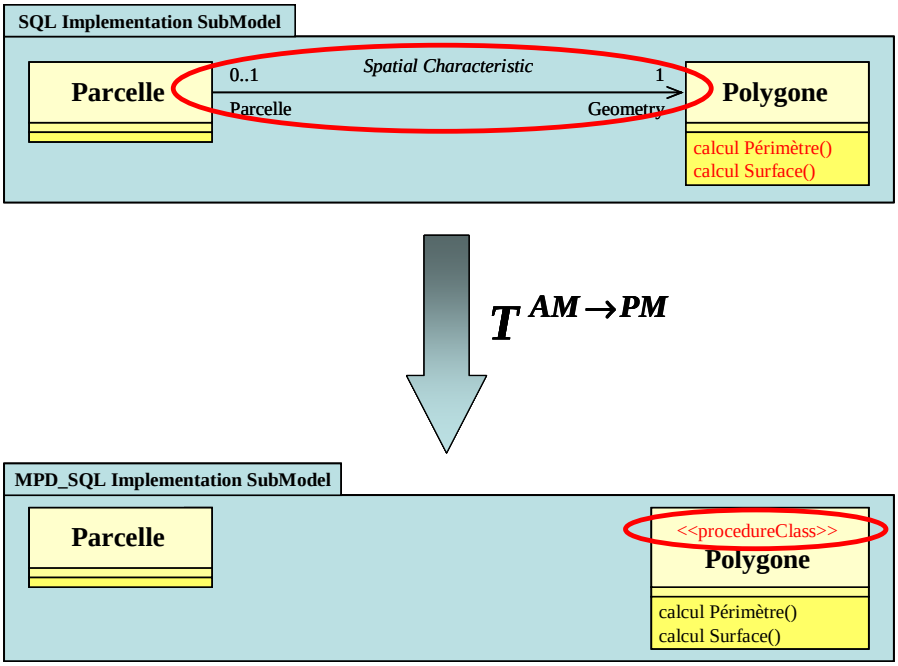


Figure 169 Modifications introduites par la transformation $T^{AM \rightarrow PM}$.

III.2 Affichage récurrent

La transformation $T^{AM \rightarrow PM}$ du générateur de code SQLDesigner affiche de façon récurrente le message de la figure 170 lorsque de modèle en cours de réalisation contient au moins l'un des deux motifs de la figure 171, c'est-à-dire lorsqu'une classe fille a une relation de composition.

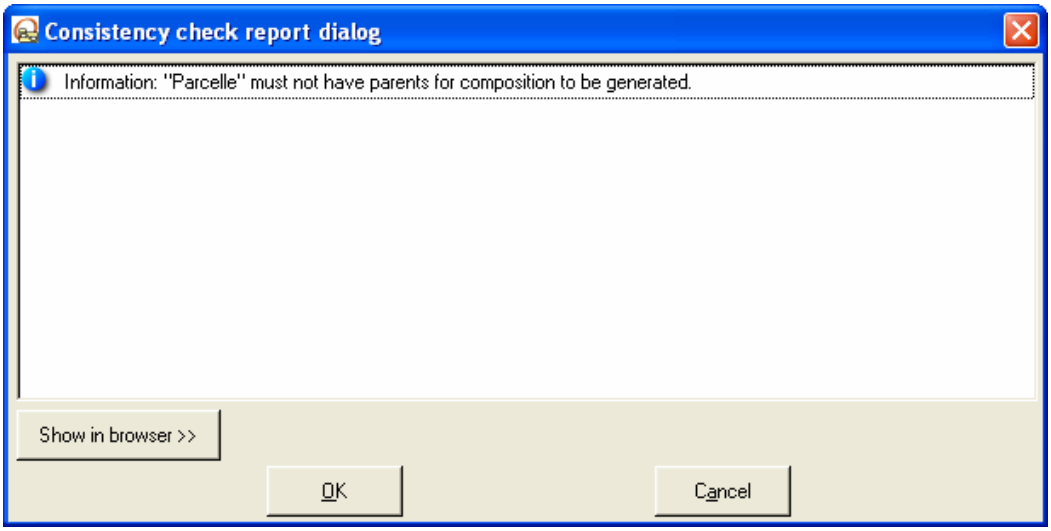


Figure 170 Affichage occasionné par la présence dans le modèle d'un ou de plusieurs motifs de la figure 171.

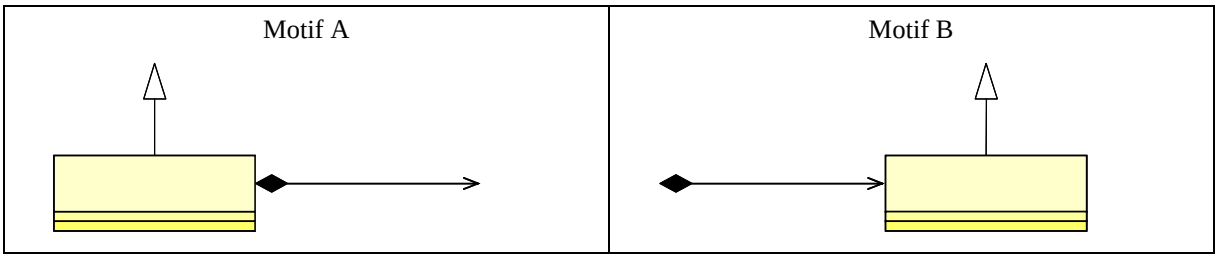


Figure 171 Motifs provocant l'affichage du message d'avertissement.

IV EN RESUME

Le générateur de code SQLDesigner applique l'approche MDA car il crée automatiquement un modèle intermédiaire, appelé Modèle Physique, sur lequel le concepteur a encore la possibilité d'ajouter des spécifications détaillant l'implémentation de la base de données et ce en fonction du SGBDR[126] cible.

Au final, le générateur de code SQLDesigner projette le Modèle Physique obtenu en code SQL suivant un certain nombre de règles qui interprètent les spécifications du langage SQL présentes dans le métamodèle sous forme de stéréotypes ou de valeurs marquées.

Dans le contexte de la méthode *Continuous Integration Unified Process*, ce générateur de code présente quelques inconvénients qui nous ont amené à concevoir et à implémenter des transformations PSM/PSM pour les « contourner » (cf. Chapitre 5-IV).

[126] Les SGBDR disponibles dans Objecteering sont : Oracle, Sybase, SQL Server.

Annexe VI. Fondements théoriques sur le prototypage

Avertissement : Le texte de cet annexe est reproduit in extenso du mémoire pour l'obtention du grade de maître ès sciences de Louis-Etienne Guimond (Guimond, 2005).

Bien qu'il existe plusieurs formes de prototypage, nous ne nous référons qu'à trois d'entres elles dans le cadre de nos travaux. Il s'agit du prototypage évolutif, du prototypage jetable et du maquettage.

Sommaire détaillé

I PROTOTYPAGE EVOLUTIF

Des trois démarches que nous définissons, le prototypage évolutif est la forme la plus évoluée. Il consiste à développer un prototype tout au long d'un projet, en plusieurs itérations, jusqu'à ce qu'il devienne le système final. Cela permet, lors de chacune des itérations, de présenter un livrable aux utilisateurs, qu'ils peuvent explorer, valider et commenter en vue de l'améliorer et de le rendre plus satisfaisant par rapport à leurs attentes. Ainsi, d'itérations en itérations, on obtient une version finale, complète et fidèle par rapport aux attentes formulées et qui devrait répondre de façon satisfaisante aux besoins des utilisateurs. Lorsque ce niveau d'exhaustivité est atteint, l'analyse des besoins peut être considérée complète et les derniers développements nécessaires pour rendre le prototype pleinement fonctionnel peuvent être réalisés. Ces développements peuvent concerner la programmation des fonctionnalités qui auraient été simulées jusque là et l'intégration complète des données dans le système. Cette façon de faire, en deux temps distincts, soit l'analyse itérative suivie du développement, rappelle le modèle de développement itératif. Une autre façon de faire consiste à effectuer le développement en même temps que le prototypage, c'est-à-dire que le prototype est en fait une partie fonctionnelle du système. Ainsi, à chaque itération, les utilisateurs ont en main la partie développée du système final. Ils peuvent dès lors opérer et évaluer cette portion du système. Cette démarche, qui intègre l'analyse et le développement autour d'un prototype en évolution, rappelle davantage le modèle de développement évolutif. Le point essentiel qui distingue le prototypage évolutif des autres formes est sa réutilisation complète dans le système final.

II PROTOTYPAGE JETABLE

Par opposition au prototype évolutif, le prototype jetable n'est pas réutilisé dans le système final. Son principal rôle est d'assister l'analyse des besoins en proposant aux utilisateurs une représentation statique ou peu interactive du système final. Tout au long de l'analyse de besoins, le prototype jetable est amélioré en réponse aux commentaires et aux besoins formulés par les utilisateurs. Ainsi, la dernière version du prototype jetable devrait représenter le plus fidèlement possible les spécifications du système final. Un avantage du prototypage jetable est qu'il est facile et rapide à réaliser. L'effort demandé pour produire un prototype jetable peut cependant sembler vain puisqu'il n'en reste rien lors du développement. Malgré cela, cette approche est bien appropriée pour réaliser l'analyse des besoins puisqu'elle permet à peu de coûts et en peu de temps de présenter un aperçu du système final aux utilisateurs. Il est ainsi possible de mesurer la satisfaction des utilisateurs par rapport à la solution finale.

III MAQUETTAGE

À un niveau de fidélité moindre, on retrouve le maquettage qui permet la représentation la plus sommaire par rapport au système final. En fait, l'objectif du maquettage n'est pas d'avoir une représentation complète du système final. Il s'agit plutôt de proposer aux utilisateurs une « traduction » des besoins qu'ils ont formulés en termes de possibilités offertes par la solution. Il sert également à vérifier que la problématique des utilisateurs est bien comprise et à la positionner par rapport à la solution proposée. Aussi, une maquette n'a pas à être exhaustive et fidèle quant aux fonctionnalités qui se retrouveront dans le système final. Elle doit se contenter d'illustrer ces fonctionnalités pour s'assurer que les utilisateurs en ont une bonne compréhension. En ce sens, une maquette est beaucoup plus un moyen de communication qu'un résultat en soi. Aussi, lorsque son rôle est accompli et que les utilisateurs sont en mesure d'énoncer clairement leurs attentes, et que celles-ci peuvent être facilement traduites en spécifications du système final, le maquettage prend fin pour être remplacé par le développement ou une forme de prototypage plus évoluée. Dans le déroulement d'un projet de développement, le maquettage peut jouer un rôle principalement au début de l'analyse des besoins, alors que les utilisateurs doivent apprendre les concepts et les possibilités de la solution finale. À ce stade, les besoins qu'ils sont en mesure d'énoncer sont assez généraux et peuvent être facilement représentés dans une maquette. Pour sa réalisation, le maquettage emprunte beaucoup au prototypage jetable, ce qui le rend très rapide à réaliser. Il nécessite très peu ou pas de programmation des fonctions du système et bien sûr, il n'intègre par de données réelles.

Le Tableau 11 présente une brève synthèse de certaines caractéristiques des différentes formes de prototypage que nous avons définies. Pour chacune des caractéristiques, nous donnons une cote d'appréciation (Faible à Élevé) basée sur une échelle relative de classement des formes de prototypage.

	Prototypage évolutif	Prototypage jetable	Maquettage
Rapidité et coût de réalisation	Élevé	Moyen	Faible
Effort de programmation demandé	Élevé	Moyen	Faible
Apport pour l'analyse des besoins généraux	Moyen	Moyen	Élevé
Apport pour l'analyse des besoins détaillés	Élevé	Moyen	Faible
Exhaustivité par rapport au système final	Élevée	Moyenne	Faible
Niveau de réutilisation pour le développement	Élevé	Nul	Nul

Tableau 11 Synthèse comparative des formes de prototypage.

Abstract :

At the crossroad of the fields of *Geographic Information* and *Computing*, this research which was carried out comes under the "*Geomatic*" field. It deals with the contribution of the model engineering to the design and the development of *Geographic Information Systems* (GIS). The geomatic application field, a weak capitalization of knowledge during the development process and a poor quality of the captured knowledge during the analysis have led to fix as an aim to *develop a supporting tool for the design of Geographic Information Systems* adapted to *a development process enabling fast prototyping during the analysis phase* and ensuring the *knowledge capitalization*. The dichotomy Geographic Information/Computing required to mobilize knowledge belonging to these two fields. It gives the structure to the research content.

The first contribution concerns the GIS modeling. After a quick overview of the background bibliography on the methods and the formalisms used for the design of the GIS, the need to compare the spatial and temporal properties used in these methods and these formalisms has resulted in a terminological study realized with the Set Theory. This study resulted in a general taxonomy of the used terminology. The conceptual proximity between this taxonomy and the UML class diagrams enabled to define a method of derivation of this taxonomy in order to obtain a *GIS metamodel* and a *GIS Design Pattern*. The latest will be automatically generated by the transformations defined in model engineering. The terminological study also enabled to specify the relationships between the thematic concepts, the spatial and temporal properties, etc. These relationships contributed to specify the model transformations of the second contribution.

The second contribution concerns the model engineering. The objective related to the *development process enabling fast prototyping during the analysis phase* has led to define the method *Continuous Integration Unified Process*, which superimposes a fast prototyping cycle in the analysis phase of the *Unified Process* method. The objective of *knowledge capitalization* required to design a generalization of the MDA approach called *Software Development Process Approach* (SDPA). It is based on the statement that the knowledge capitalization is an issue which is met during the whole development process. A multimodel artifact, called *Software Development Process Model* (SDPM), has been developed to solve this problem. It associates a sub-model to each of the development cycle phases. A diffusion transformation based on the cloning of the concepts enables to transfer, from sub-model to sub-model, the concepts from the analysis sub-model up to the implementation sub-models. A retrodiffusion transformation enables to achieve the reverse transfer. The coherence of the SDPM is ensured by an traceability link architecture which connect all the concepts to their clone and by pre-treatments and/or post-treatments to the transformations of diffusion and of retrodiffusion. Four geomatic transformations enable on the one hand, the capture the spatial and/or temporal properties of the thematic concepts and, on the other hand, the conversion of these properties into UML model elements which can be used by the code generators of the case tool. Finally, three transformations which were carried out on the SQL implementation model fit the diffused model to the code generator of the case tool.

The *Continuous Integration Unified Process* method and the *Software Development Process Approach* has been implemented in the Objecteering Case Tool.

Keywords :

Knowledge capitalization, Model-Driven Architecture, Software Development Process, Unified Process, Geographic Information System, Languages and Formalisms of description of Geographic Information Systems

Résumé :

À la conjonction de l'*Information Géographique* et de l'*Informatique*, la recherche menée relève du domaine *Géomatique*. Elle traite de l'apport de l'*ingénierie des modèles* à la conception et au développement de *Systèmes d'Information Géographique* (SIG). Le domaine d'application géomatique, une capitalisation des connaissances déficiente au cours du processus de développement et une qualité des connaissances capturées en séance d'analyse insuffisante ont conduit à se fixer comme objectif de réaliser un *outil d'aide à la conception de Systèmes d'Information Géographique* adapté à un *processus de développement permettant le prototypage rapide en séance d'analyse* et assurant la *capitalisation des connaissances*. La dichotomie Information Géographique / Informatique a nécessité de mobiliser des connaissances de ces deux domaines et structure le contenu de la recherche.

La première contribution porte sur la modélisation des SIG. Suite à un balayage rapide de la bibliographie sur les méthodes et formalismes de conception de SIG existants, le besoin de comparer les propriétés spatiales et temporelles mise en œuvre par les méthodes et les formalismes a conduit à entreprendre une étude terminologique s'appuyant sur la théorie des ensembles. Cette étude a donné lieu à une taxinomie générale de la terminologie utilisée. La proximité conceptuelle entre cette taxinomie et les diagrammes de classes UML a permis de définir une méthode de dérivation de cette taxinomie pour obtenir un métamodèle SIG et un Patron de Conception SIG. Ce dernier sera généré automatiquement par les transformations définies en ingénierie des modèles. L'étude terminologique a aussi permis d'identifier les relations entre les concepts thématiques, les propriétés spatiales et temporelles, etc. Ces relations ont contribué à définir les transformations de modèles de la seconde contribution.

La seconde contribution relève de l'ingénierie des modèles. L'objectif relatif au *processus de développement permettant le prototypage rapide en séance d'analyse* a conduit à définir la méthode *Continuous Integration Unified Process* qui superpose un cycle de prototypage rapide en phase d'analyse de la méthode Unified Process. L'objectif de *capitalisation des connaissances* a nécessité de concevoir une généralisation de l'approche MDA appelée *Software Development Process Approach* (SDPA) fondée sur le constat que la capitalisation des connaissances est une problématique qui se pose au cours de tout le processus de développement. Un artefact multimodèle, appelé *Software Development Process Model* (SDPM), a été conçu pour résoudre ce problème. Il associe un sous-modèle à chacune des phases du cycle de développement. Une transformation de diffusion fondée sur le clonage des concepts permet de transférer, de sous-modèle en sous-modèle, les concepts du sous-modèle d'analyse jusqu'au(x) sous-modèle(s) d'implémentation. Une transformation de rétrodiffusion permet le transfert inverse. La cohérence du SDPM est assurée par une architecture de liens de traçabilité qui relie tout concept à son clone et par des prétraitements et/ou post-traitements aux transformations de diffusion et de rétrodiffusion. Quatre transformations de nature géomatique permettent d'une part, la saisie des propriétés spatiales et/ou temporelles des entités référencées sous forme d'annotations et, d'autre part, la conversion de ces annotations en éléments de modélisation UML exploitables par les générateurs de code de l'atelier de génie logiciel. Enfin, les trois transformations effectuées sur le modèle d'implémentation SQL adaptent le modèle diffusé au générateur de code de l'atelier de génie logiciel utilisé.

La méthode *Continuous Integration Unified Process* et la démarche *Software Development Process Approach* ont été instrumentées au sein de l'atelier de génie logiciel Objecteering.

Mots clés :

Capitalisation des connaissances, Ingénierie des modèles, Méthode de conduite de projet, Processus Unifié, Système d'Information Géographique, Langage et formalisme de description des Systèmes d'Information Géographique