Ingénierie des systèmes d'information : bases de données, bases de connaissances et méthodes de conception

Dominique Rieu

LSR-IMAG

Mémoire d'habilitation à diriger les recherches

spécialité
Informatique

présenté à
l'Institut National Polytechnique de Grenoble

par

Dominique RIEU

le 8 Décembre 1999

Ingénierie des Systèmes d'Information

Bases de Données, Bases de Connaissances

et Méthodes de Conception

Composition du jury :
Michel Adiba
Claude Chrisment
Jacques Courtin
Jean-Pierre Giraudin
Paul Jacquet (rapporteur)
Michel Léonard (rapporteur)
Colette Rolland (rapporteur)

Résumé

Ce mémoire est le reflet de 17 ans d'activités de recherche liées aux modèles et aux techniques orientés objet mises en œuvre en base de données, en représentation des connaissances puis plus récemment en système d'information.

J'ai participé à des projets dont l'objectif commun était de contribuer à l'évolution des SGBD et des SRC pour une meilleure prise en compte des informations complexes et évolutives liées aux applications d'ingénierie (la CAO, l'ingénierie des méthodes, les systèmes d'information produit, etc.). Les principaux apports de ces projets ont été de montrer que les approches « tout objet » constituent un précieux outil de représentation des connaissances :

- permettant d'exprimer la sémantique des systèmes et de leurs concepts,
- facilitant la mise en œuvre de mécanismes d'évolution,
- favorisant l'élaboration d'environnements adaptables à des besoins particuliers.

L'approche objet offre de fait la possibilité de réutiliser des classes. Cependant, il est clair qu'aujourd'hui un concepteur ne peut plus spécifier et implanter un système en partant d'une granularité aussi fine. Il ne s'agit plus de réutiliser une classe ou un mécanisme de bas niveau tel que l'instanciation mais de favoriser la réutilisation d'architectures de granularité plus importante (ensemble de classes, de types, etc.) et détenant en propre leurs propres mécanismes d'utilisation.

La réutilisation à tous les stades du développement des systèmes d'information est devenue depuis 1997 mon principal thème de recherche. L'accent est particulièrement mis sur les approches à base de patrons (pattern).

Mots clés : base de données, représentation de connaissances, système d'information, modèle à objet, réutilisation, patron d'ingénierie.

Sommaire

Chapitre 1 : le fil conducteur

Chapitre 2 : Des Bases de Données aux Bases de Connaissances

Chapitre 3 : Réutilisation dans l'ingénierie des SI

Chapitre 4 : Perspectives

Bibliographie

Chapitre 1 : Le fil conducteur

1.1. Bases de Données pour la CAO

En 1982, mon projet de DEA s'intitulait “Bases de Données et nouvelles applications”. Dans l’environnement Bases de Données, cela concernait d’une part les applications multimédias (son, image, etc.) et d’autre part les applications "constructives" en particulier celles d'ingénierie telles que la Conception Assistée par Ordinateur. Très rapidement, je me suis consacrée aux applications de CAO qui restent aujourd’hui un défi pour les recherches en représentation de connaissances (BD, IA, SI, etc.). Un numéro spécial de la revue l’Objet (Représentation Objet en Conception), sorti en Septembre 1998 [Objet 98], montre que la recherche française reste active sur ce thème. Plusieurs projets européens à l’initiative de l’AIT (Advanced Information Technology Design and Manufacturing) témoignent également de cette effervescence [Waite 97].

La représentation, la gestion et l’intégration des données techniques issues des processus de développement (conception et industrialisation) des produits a été et reste encore mon domaine d’application privilégié. Il s'agit, en particulier, de mettre en œuvre des modèles de données et de connaissances à sémantiques fortes permettant une gestion intégrée de produits multi-métiers évoluant d'un cahier des charges au produit conçu et réalisé. Le modèle de connaissances est alors le support du cycle de vie des produits : il maintient la cohérence globale du produit et "présente à chacun des intervenants une vue correspondant à son métier tout en gérant la cohérence et les liens entre les intervenants" [Tollenaere 95].

Durant ma thèse, j’ai spécifié et réalisé un SGBD dédié aux applications de CAO (CADB[1]) [Rieu 85]. En 1985, CADB présentait des fonctionnalités originales permettant par exemple une conception aussi bien ascendante que descendante d'objets composites. Par contre, il n'offrait que peu de fonctionnalités adaptées à des évolutions cohérentes et assistées d'objets en cours d'élaboration. Ces fonctionnalités, fondamentales dans le cadre d'applications CAO, nécessitaient d'intégrer de nouveaux mécanismes permettant par exemple le contrôle automatique de la cohérence et le calcul des effets de bord lors de la mise à jour des informations. Ces extensions ont été spécifiées en utilisant des concepts et des techniques inspirés de l'Intelligence Artificielle [Nguyen 87a]. Des connaissances expertes mises en

[1] CADB : Computer Aided Design Data Base.

oeuvre par des règles heuristiques avaient pour rôle de gérer des objets incomplets et incohérents, de contrôler la légalité des opérations de mises à jour et de répercuter les mises à jour de données interdépendantes[2].

1.2. Systèmes de Représentation de Connaissances à Objet

En 1988 l'étude que nous avions abordée sur la gestion des évolutions des données a contribué à la définition du projet de recherche INRIA SHERPA [Escamilla 90]. Ce projet rassemblait deux équipes travaillant respectivement dans les domaines des bases de données et de l'intelligence artificielle, afin de mener une recherche commune au sein d'un concept unificateur, l'objet. L'objectif plus général de SHERPA était de fournir un support à des applications dont les exigences opérationnelles n'étaient pas entièrement satisfaites par les SGBD ou les Systèmes de Représentation de Connaissances (SRC) de l'époque [Favier 90].

Les travaux menés au sein de notre équipe nous ont permis de spécifier et d'expérimenter un Système de Représentation de Connaissances à Objet (SHOOD[3]) destiné aux applications à forte évolution (dynamique des valeurs, des structures et des modèles) [Rieu 92a]. L'objectif à terme était de développer une plate-forme d'intégration par les données dédiée aux applications de CAO. Le système SHOOD, cœur de la plate-forme, était destiné à l'élaboration de bases de connaissances communes et partagées par les outils intervenant lors des processus de conception et d'industrialisation des produits.

Des travaux similaires (par exemple le projet IIICAD de l'Université de Tokyo) ont montré la nécessité de représenter les connaissances sur les produits mais aussi sur leur processus de conception. La modélisation simultanée des produits, de leur processus de conception et des ressources intervenant lors de ce processus, nécessite la mise en oeuvre de puissants formalismes de représentation de connaissances. L'une des solutions les plus prometteuses semblait être celle des systèmes de représentations de connaissances dits hybrides qui offraient des formalismes complémentaires : en général un modèle à objet couplé à d'autres formalismes en particulier celui des règles de production.

C'est la démarche que nous avons suivie dans SHOOD qui combinait un modèle à objet issu :

- des Langages Orientés Objet (LOO) avec la notion de métacircularité inspirée de Objvlisp [Cointe 87], des concepts de fonctions génériques de CLOS[4] [Bobrow 88], etc.

[2] CADB devint alors le sigle de Computer Aided Design **Deductive** Data Base.
[3] SHerpa Object Oriented Dynamic
[4] Common Lisp Object System

- et des Systèmes de Représentation de Connaissances à Objet (SRCO) [Rechenman 88, Dugertil 88, Carré 90, Marino 93] avec des notions de point de vue, de classification dynamique, etc.

Ce modèle à objet était couplé à un formalisme de règles basé sur le concept de règles événementielles (Evénement-Condition-Action) issu des Bases de Données Actives [Dayal 88, Collet 94, Bounaas 95].

Durant une dizaine d'années, j'ai donc participé à des projets (CADB et Shood) dont l'objectif commun était de contribuer à l'évolution des SGBD et des SRC pour une meilleure prise en compte des informations complexes et évolutives liées aux produits en cours de développement. A ce titre, ces projets répondirent partiellement à cet objectif ; par contre nous manquions fortement de méthodes permettant l'extraction puis le transfert des connaissances vers nos systèmes de représentation de connaissances. Dans les applications de CAO, cet aspect est devenu crucial lorsqu'on a commencé à s'intéresser non plus à la simple expression des connaissances structurelles et comportementales des objets de conception mais aussi à celles inhérentes aux stratégies de résolution des problèmes de conception, en d'autres termes à la modélisation et l'exécution des processus même de conception. La modélisation des processus par des règles événementielles a certainement été le plus gros problème des utilisateurs de notre système. Pourtant il s'agissait d'utilisateurs "avertis", en l'occurrence des concepteurs de métiers (mécaniciens et électrotechniciens), bien au fait des techniques de représentation de connaissances, et en particulier, connaissant les modèles à objet et certains mécanismes de résolution de problèmes [Rieu 94].

Le développement de ce que l'on appellerait quelques années plus tard les Systèmes d'Information Produit (SIP) [Randoing 95] se faisait sans aide méthodologique. Aujourd'hui, mes activités sur ce thème ont pour cadre l'élaboration d'une méthode de développement pour les Systèmes d'Information Produit (Projet POSEIDON[5]) [Cauvet 98, Gzara 99].

Commentaire :

1.3. Ingénierie des Systèmes d'Information

Depuis 1995, mon objectif est de contribuer aux évolutions des méthodes de développement des systèmes d'information et plus particulièrement à celles des Méthodes d'Analyse et de Conception Orientées Objet (OMT [Rumbaugh 95], UML [Booch 97b, Booch 99], etc.). L'accent est particulièrement mis sur la réutilisation à tous les stades de l'ingénierie des systèmes.

[5] Patrons d'Objet pour les Systèmes de gestion et d'Echanges Industriels de DONnées.

Traditionnellement, l'ingénierie des systèmes s'organise par la mise en place, dans le cadre d'une méthode (MERISE, OMT, UML, etc.), d'une succession de modèles (Entité-Association, Objet, réseau de Petri, graphe d'états, modèle de flux, etc.) afin de favoriser un continuum de la définition des besoins des clients jusqu'au système développé et exploité. Cette approche conduit à analyser, concevoir et implanter des systèmes spécifiques. Récemment, une nouvelle approche basée sur la réutilisation de composants commence à émerger. Initialement cantonnée dans les étapes d'implantation, la réutilisation s'impose de plus en plus dans les étapes en amont (expression des besoins, analyse, conception). On a vu ainsi apparaître des bibliothèques (toolkits), des frameworks (cadriciels), des patrons d'ingénierie (patterns), des composants ou objets métiers, etc. [Front 99b, Rieu 99].

D'une manière générale, la réutilisation lors du développement des systèmes orientés objet vise trois objectifs : diminuer les coûts de développement et de maintenance, réduire les délais et améliorer la qualité du logiciel. La réutilisation systématique appliquée lors des différentes étapes du développement logiciel (expression des besoins, analyse, conception, implantation) a un enjeu supplémentaire, celui d'une meilleure traçabilité des transformations des objets au sein de processus de développement « sans couture » (seamless) [Reenskaugh 92, Barbier 94, Corriveau 96].

L'une des voies les plus pertinentes semble aujourd'hui celle des patrons d'ingénierie. L'idée du concept de patron consiste à capitaliser un problème récurrent d'un domaine et sa solution de manière à faciliter la réutilisation et l'adaptation de cette solution lors d'une nouvelle occurrence du problème. Dans les développements orientés objet, les patrons sont généralement classifiés en fonction de l'étape d'ingénierie à laquelle ils s'adressent : on parle de patrons d'analyse, de conception et d'implantation. Les patrons d'analyse [Rolland 93, Coad 96, Fowler 97b] ont pour objectif d'aider le concepteur d'un système dans la construction de modèles représentant au mieux les besoins du système et les connaissances du domaine. Par exemple, il est possible d'utiliser un patron général de ressources pour modéliser aussi bien un problème de ressources humaines qui comprend l'affectation de personnes à des activités, qu'un problème de gestion de ressources matérielles réalisant l'affectation de machines dans un contexte de production [Bounaas 97a]. Les patrons de conception (design patterns) [Gamma 95, Buschmann 96] identifient, nomment et abstraient des thèmes communs du domaine de la conception orientée objet. Ils capturent l'expérience et la connaissance liées à la conception en identifiant les objets, leurs collaborations et la distribution des responsabilités.

Nous abordons le thème de la réutilisation dans les MCOO d'un point de vue théorique et d'un point de vue pratique. Les aspects théoriques concernent deux problématiques.

La première, directement liée aux patrons orientés objet, a pour enjeu d'affiner et de formaliser le concept de patron, de proposer et de formaliser des opérateurs d'utilisation et de composition de patrons, de disposer de représentations standard pour intégrer et rechercher des patrons dans des bibliothèques, etc. [Rieu 99]. Cette étude s'inscrit dans le cadre de notre nouveau projet PRIS[6] dont la perspective est de développer et d'expérimenter un environnement d'aide au développement de systèmes d'information facilitant la spécification, l'organisation et l'usage de patrons.

La deuxième problématique a pour enjeu de répondre du moins partiellement à une des principales difficultés en modélisation : la gestion de la coexistence des spécifications statique et dynamique des objets. De grands efforts restent aujourd'hui à faire pour assurer la cohérence globale des modèles statiques et dynamiques. Nous nous attachons plus particulièrement à l'intégration des diagrammes de classes et des diagrammes d'états. Les diagrammes d'états et en particulier les statecharts [Harel 87, Harel 96] ont rapidement été intégrés dans les MCOO. Force est pourtant de reconnaître que leur utilisation reste aujourd'hui marginale lors de l'analyse et de la conception des systèmes d'information. Plusieurs raisons peuvent être invoquées.

- La spécification des statecharts est réservée à des spécialistes. La plupart des concepteurs trouvent plus simple et plus naturel d'exprimer la dynamique globale du système (modèle de traitement de Merise, diagrammes de séquences et de collaboration d'UML, etc.).
- Les statecharts sont peu réutilisables. Un statechart décrit le comportement des objets d'une classe et est rarement réutilisable pour décrire celui des objets d'une autre classe.
- Les spécifications des statecharts sont peu exploitables. En particulier l'intégration des dimensions statique et dynamique, souvent jugée difficile, est repoussée au moment de l'implantation et n'est donc pas conceptualisée.

L'enjeu du modèle NCR[7] [SaintMarcel 98, SaintMarcel 99a], basé sur des formalismes éprouvés tels que les diagrammes de classes et les diagrammes d'états, est de répondre en partie à ces problèmes. L'accent est mis sur la spécification et la capitalisation de comportements abstraits (par exemple un comportement général de ressource, une évolution de produit CAO, etc.).

D'un point de vue pratique nous nous sommes intéressés et nous continuons à nous intéresser à développer des patrons généraux, mais aussi des patrons spécifiques pour des domaines particuliers : l'ingénierie des modèles au travers du projet *M*ENINGE[8] [Bounaas 97b], et des Systèmes d'Information Produit (SIP) au travers du projet POSEIDON [Rieu 97a,

[6] Pattern & Reuse for Information Systems
[7] Notion, Comportement, Rôle
[8] MEta-modélisatioN pour les projets d'INGEnierie

Cauvet 98]. Ces deux projets s'appuient sur des techniques de réutilisation similaires mais ont des objectifs différents.

L'objectif de MENINGE était de développer un environnement de méta-modélisation [Bounaas 97a, Rieu97b]. Pour cela, nous nous plaçons dans un cadre d'ingénierie des modèles qui consiste à appliquer les techniques d'ingénierie à ces systèmes particuliers que sont les modèles. Une attention particulière est portée sur la réutilisation des modèles. En effet, la plupart des "nouveaux modèles" sont souvent des extensions ou des adaptations de modèles existants (pensez par exemple à toutes les adaptations de modèles de base tels que les machines à états finis, le modèle Entité/Association, etc.). Il s'agit donc d'offrir des fonctionnalités permettant de spécifier et d'implanter des modèles par réutilisation partielle des concepts d'un autre modèle [Bounaas 97b]. Un concept est défini par un patron correspondant à l'ensemble des types nécessaires à sa définition. Un prototype nous a permis de valider cette approche au travers des modèles de Merise/2 [Panet 94] ainsi que d'un modèle de processus CAO [Harani 97].

L'objectif de POSEIDON est de définir un cadre méthodologique adapté à l'ingénierie des Systèmes d'Information Produit (SIP) des entreprises industrielles. Un SIP supporte toute la gestion de l'information technique : la base documentaire qui accompagne le développement des produits, les fichiers d'échanges avec les autres systèmes (outils de calcul, de simulation, etc.), l'ensemble des tâches du processus de développement et leurs répartitions entre les différents acteurs. L'ingénierie des SIP pose actuellement de nombreuses difficultés d'ordre méthodologique : mise en accord ambiguë entre utilisateurs et informaticiens, absence de continuum de transformations entre le dossier d'expression des besoins et la spécification puis l'implantation du SIP, développement trop lent du système, etc. [Rieu 97a]. La méthode proposée dans le cadre du projet POSEIDON [Gzara 99] est axée sur la réutilisation de patrons tout au long du processus de développement des SIP : de l'expression des besoins à la maintenance évolutive, en passant par la conception et l'implantation. L'accent est particulièrement mis sur la définition de patrons "métiers" qui mettent en évidence des formes de modélisation essentielles : la modélisation des produits et de leurs bases documentaires, la modélisation des processus, des acteurs et de leurs droits, etc. [Cauvet 98].

Tous ces projets de recherche sont des travaux d'équipe qui n'auraient pu exister sans les collègues et les jeunes chercheurs qui y ont participé :

- CADB, 1983 – 1987 :
  - M. Adiba, professeur à l'UJF alors responsable du groupe Bases de Données du LGI-IMAG.
  - Nguyen G.T, directeur de recherches à l'INRIA.

- En thèse : M.C. Fauvet, actuellement maître de conférences à l'UJF.
- En DEA : K. Nader, et J.P. Chevallet actuellement maître de conférences à l'UPMF.

➢ SHOOD, 1988 – 1994 :

- F. Rechenman, directeur de recherches à l'INRIA, alors responsable du projet INRIA Sherpa.
- Nguyen G.T, alors responsable de l'équipe Bases de Connaissances Dynamiques du LGI.
- En thèse : F. Bounaas, J. Escamilla actuellement enseignant-chercheur à l'Université de Mexico et C. Djeraba actuellement maître de conférences à l'Université de Nantes.
- En mémoire CNAM : V. Favier et M.P. Liotard.
- En DEA : P. Jean, J.F. Millasseau, H. Valéro et P. Guinet.

➢ *M*eninge, 1995 – 1997 :

- Ph. Morat, maître de conférences à l'UJF.
- F. Bounaas, alors post-doctorant.
- En mémoire CNAM : C. Martin et P. Cailloce.
- En DEA : D. Tamzalit actuellement en thèse à l'EERIE à Nîmes.

NCR, 1996 - …

- Ph. Morat, maître de conférences à l'UJF.
- En thèse : C. Saint-Marcel, actuellement ATER à l'UPMF.

POSEIDON, 1997 - ….

- Jean-Pierre Giraudin, professeur à l'UPMF et responsable du pôle ingénierie des SI de l'équipe STORM du LSR.
- En thèse : L. Gzara.
- En DEA : H. Randriamparany.

PRIS, 1999 - :

- Jean-Pierre Giraudin, responsable du projet.
- M. Chabre-Peccoud, maître de conférences à l'UJF.
- A. Front-Conte, maître de conférences à l'UPMF.
- En thèse : C. Saint-Marcel et Lilia Gzara.
- En mémoire CNAM : B. Barthez.

Bon nombre de ces travaux ont nécessité des compétences pluridisciplinaires. De nombreuses collaborations avec des collègues de disciplines mécanique, électrotechnique et sociologique se sont révélées particulièrement fructueuses.

- M. Tollenaere, professeur à l'ENSGI et Jean Bigeon, directeur de recherche au CNRS (Laboratoire d'Electrotechnique de Grenoble) furent les principaux moteurs de l'utilisation de Shood dans les applications de conception de produits mécaniques et électrotechniques. Actuellement M. Tollenaere est responsable du projet POSEIDON dans le cadre du réseau PROSPER du CNRS.

- J.L Guffond et G. Leconte sont tous deux chercheurs au CNRS en sociologie (laboratoire CRISTO). Ils ont très activement participé à l'intégration de Shood comme outil de prototypage de systèmes de bases de connaissances dans des entreprises et sont aujourd'hui d'une aide précieuse pour les audits de terrain du projet POSEIDON.

Je tiens ici à signaler le rôle fondamental de l'Institut de la Production Industrielle (anciennement Groupement Scientifique Interdisciplinaire en Productique) pour accueillir et favoriser des travaux interdisciplinaires.

La suite de ce mémoire est organisée en deux chapitres.

Le chapitre 2 intitulé "Des Bases de Données aux Bases de Connaissances" décrit l'apport du concept d'objet dans le rapprochement de deux disciplines : les Bases de Données et les Systèmes de Représentation des Connaissances (cf. §2.1.1). Ce rapprochement a permis une meilleure prise en compte des besoins des applications d'ingénierie telle que la CAO (cf. §2.1.2) et l'ingénierie des méthodes (cf. §2.1.3) en favorisant l'expression et la gestion des évolutions dynamiques d'objets tout en renforçant le pouvoir déclaratif des modèles à objet traditionnels. Mes travaux sur ce thème correspondent :

- aux projets CADB (cf. §2.2) et Shood (cf. §2.3), dédiés aux applications de CAO (mécanique, électrotechnique et architecturale),
- au projet *M*eninge (cf. §2.4), dédié à l'ingénierie des méthodes et en particulier à celle des modèles offerts par les méthodes d'analyse et de conception des Systèmes d'Information.

Le chapitre 3 intitulé "La Réutilisation dans l'Ingénierie des Systèmes d'Information" met l'accent sur deux des principaux enjeux des Méthodes d'Analyse et de Conception Orientées Objet : garantir la maîtrise des activités de développement des systèmes et offrir une gestion efficace de composants réutilisables (§3.1). Ces deux objectifs, étroitement liés, nécessitent de mettre en œuvre de nouveaux modèles de composants et de nouvelles démarches de

développement facilitant la réutilisation dans les étapes en amont du développement des systèmes : expression des besoins, analyse et conception. Dans le cadre de ce mémoire, nous mettrons l'accent sur deux travaux de l'équipe :

le modèle NCR, actuellement en cours de spécification dans le cadre de la thèse de C. Saint Marcel (§3.2) ;

les approches à base de patrons (§3.3). Plusieurs projets de l'équipe (SCalP [Front 97], *M*eninge et POSEIDON) ont abordé ce thème. Nous nous contenterons ici de proposer des techniques issues des expériences de ces divers projets.

Le dernier chapitre tient lieu de conclusion. Il présente les grands axes du projet PRIS qui constituera l'essentiel des activités de la toute nouvelle équipe SIGMA du laboratoire LSR-IMAG dans les prochains mois.

Chapitre 2 : Des Bases de Données aux Bases de Connaissances

2.1. Introduction

2.1.1. L'objet en BD

La recherche en informatique a toujours subi des modes plus ou moins durables. Une des plus spectaculaires fut certainement celle de l'Objet. J'ai eu la chance de démarrer mes activités de recherche avec elle et de voir cette vague déferlante gagner peu à peu toutes nos disciplines informatiques. Les Langages Orientés Objet (LOO), les Systèmes de Gestion de Bases de Données Orientés Objet (SGBDOO), les Systèmes de Représentation de Connaissances à Objet (SRCO), les Méthodes de Conception Orientées Objet (MCO_O) sont devenus des réalités industrielles. Il n'existe pas aujourd'hui un mais des modèles à objet adaptés à différents domaines et à différents objectifs.

Les LOO et les SRCO sont sans conteste les précurseurs. Ils ont, dès les années 1985, influencé la recherche en bases de données. L'influence des LOO sur les SGBDOO avait de multiples raisons aux noms attractifs : réutilisation, abstraction, objets composites (ou complexes), etc. Plus précisément, les travaux entrepris sur les SGBDOO avaient pour objectifs de répondre aux faiblesses des SGBD relationnels. Rappelons rapidement ces faiblesses aujourd'hui bien connues [Bouzeghoub 95].

Tout d'abord les SGBD relationnels avaient été conçus pour des applications de gestion traditionnelles (gestion des stocks, du personnel, etc.) nécessitant des types de données classiques (entier, caractère, etc.) et des modèles de données simples (tables relationnelles). Les nouvelles applications telles que la CAO et la bureautique nécessitaient la prise en compte de types de données moins classiques (graphique, texte, etc.) et des modèles de données plus riches offrant en particulier des structures de données hiérarchisées.

D'autre part, le couplage des langages de requête de nature ensembliste tels que SQL et des langages de programmation procéduraux a toujours été le point faible des SGBD relationnels.

Toute requête quelque peu complexe, nécessitant par exemple une manipulation itérative sur un ensemble de n-uplets, est réalisée par un programme intégrant des requêtes SQL. Cette intégration génère des dysfonctionnements résolus de manière peu élégante par des conversions de types et des manipulations de curseurs.

L'approche orientée objet répondait à ces problèmes en offrant des modèles permettant la prise en compte d'objets complexes et un langage de programmation de bases de données unique. Finalement, mais je ne pense pas qu'à l'époque l'enjeu était le même qu'aujourd'hui, la technologie objet offrait des possibilités de partage et de réutilisation du code des applications. Les concepts offerts par les LOO [Masini 89] et les SGBDOO [Delobel 91] sont rapidement devenus très similaires, permettant ainsi d'éviter toute incompatibilité conceptuelle et donc structurelle. Un parallèle peut être fait entre les différentes écoles des LOO et celles des SGBDOO. L'école scandinave, dont le leader a été Simula [Dahl 66] et les représentants les plus typiques C++ [Stroustrup 89] et Eiffel [Meyer 88], a fortement influencé dans les années 85-90 toute une génération de SGBDOO à typage statique. C'était par exemple le cas de Ontos [Andrews 90] et de notre SGBD national O2 [Velez 89]. L'école Smalltalk [ParcPlace 90] a, par contre, fait des adeptes chez les partisans du typage dynamique et du "tout objet", c'était par exemple le cas des SGBD Orion [Kim 90] et Gemstone [Copeland 84].

Il est intéressant de noter, qu'à la fin des années 1980, l'approche SGBDOO se basait sur des concepts et des langages mis au point pour les langages de programmation, alors que la demande et la recherche en BD s'étaient simultanément orientées vers les techniques de l'IA comme en témoignent les systèmes de bases de données expertes (Expert Database Systems ou EDS) [Nguyen 87a] et les SGBD déductifs [Kuper 84, Kiernan 89]. Un des résultats des travaux entrepris dans les SGBD déductifs a été de jeter un pont vers l'Intelligence Artificielle. Il s'agissait d'une part d'étendre les fonctionnalités des SGBD à l'aide de mécanismes d'inférence et d'autre part de permettre l'accès à de grandes bases de faits depuis des systèmes de déduction automatique. L'appellation "base de connaissances" était utilisée en BD et en IA avec des acceptions différentes, mais qui révèlent un besoin commun : celui **de renforcer le pouvoir déclaratif** des modèles de données ou de connaissances. C'est ainsi que l'ajout de capacités déductives à un SGBD n'a pu en effet se faire qu'en renforçant le modèle de données par de nouveaux formalismes, la plupart du temps basés sur la logique. De même, les Systèmes de Représentation de Connaissances à Objet, développés en IA, offraient des formalismes très déclaratifs, permettant une représentation des connaissances indépendantes de leur exploitation.

Ce besoin de déclarativité est fondamental en IA mais aussi en BD. Pourtant, les travaux sur les SGBDOO l'ont en partie occulté en mettant l'accent sur les aspects structurels et procéduraux. C'est ainsi que contrairement aux SGBD relationnels, la plupart des SGBDOO ont, dans un premier temps, oublié les travaux antérieurs sur la gestion et l'expression des schémas de vues, des contraintes d'intégrité, des triggers, etc. En particulier, ils n'offraient plus d'expression déclarative des contraintes et donc pas de mécanismes généraux de gestion de l'intégrité. Les contraintes étaient, en effet, "codées" dans les méthodes et étaient explicitement invoquées.

Bon nombre de travaux furent alors entrepris pour enrichir les concepts issus des LOO d'un certain pouvoir déclaratif. Les travaux menés dans le cadre des mécanismes d'évolution d'objets [Banerjee 87, Nguyen 89a, Nguyen 89b], de la prise en compte des versions d'objets composites [Kim 88, Fauvet 88, Katz 90], des vues (d'objets et de schémas) [Heiler 90, Abiteboul 91, Scholl 91, Bertino 92, Rieu 92b, Ra 94, Souza 94] ou celles des objets actifs (Bases de Données Actives) [Dayal 88, Collet 94] sont en ce sens significatifs. Une autre solution consistait à restaurer le pont entrepris par les SGBD déductifs entre les BD et l'IA. Cette approche est celle actuellement suivie par les travaux proposant d'adjoindre aux SGBDOO des capacités déductives [Falcone 99]. C'est l'axe de recherche que nous avions initialisé dans le projet CADB et qui a été renforcé dans les projet SHOOD ET *M*eninge. Ces trois projets sont respectivement décrits aux paragraphes 2, 3 et 4.

Les principaux objectifs de ces projets étaient d'une part de **renforcer la déclarativité des modèles à objet** et d'autre part de proposer des mécanismes permettant des **évolutions dynamiques d'objets**. Il s'agissait également d'offrir des **environnements adaptables** en fonction des domaines d'application visés. Pour atteindre de tels objectifs, les projets SHOOD et *M*eninge ont mis l'accent sur les approches "tout objet" offrant une représentation explicite de la sémantique des systèmes de représentation de connaissances proposés (les systèmes SHOOD et *M*eninge) et de celle des systèmes applicatifs construits à partir de ceux-ci.

Ces types d'extension sont fondamentaux pour représenter et gérer au mieux des objets complexes à fortes évolutions. De tels objets sont fréquents dans les applications d'ingénierie où ils représentent les produits et les processus d'ingénierie. C'est, en particulier, le cas de la CAO (cf. §2.1.2) qui constitua un champ d'expérimentation privilégié des projets CADB et SHOOD. C'est également le cas de l'ingénierie des méthodes (cf. §2.1.3), domaine de prédilection du projet *M*eninge.

2.1.2. La Conception Assistée par Ordinateur

2.1.2.1. Les environnements de CAO

La nécessité de disposer de systèmes supportant la globalité des cycles de vie des produits industriels et donc en particulier de leur processus de développement n'est plus aujourd'hui à démontrer. Une solution communément admise consiste en l'élaboration d'environnements intégrés facilitant la cohabitation et la coopération des différents outils utilisés lors de l'élaboration des produits[9]. L'intégration peut être envisagée sous différents aspects. Au niveau physique, il s'agit d'offrir une plate-forme logicielle offrant des mécanismes opératoires uniformes (par exemple un système d'exploitation unique). Bien que nécessaire, ce type d'intégration ne permet aucune assistance ni contrôle dans l'activité de production. Les différents outils sont tout au plus compatibles.

En CAO, les premiers environnements intégrés de conception réalisaient une intégration dite par les traitements consistant à mettre en place des enchaînements de traitements la plupart du temps prédéfinis. Toute transmission des données d'un outil à un autre nécessitait de générer de nouvelles interfaces de conversion de données. Cette approche imposait une démarche figée de production et donc une trop grande rigidité des processus de développement. L'apparition de formats standardisés de transmission de données (IGES, SET, etc. [Grabowiski 90]) a partiellement résolu ce problème en fournissant des structures de données dites neutres, communes et acceptées par les différents outils. Cependant les modèles de produits étant constitués d'éléments de trop bas niveaux, la représentation des données restait sémantiquement très pauvre. Tout partage d'informations par fichier neutre est une source potentielle de perte d'information.

Les travaux entrepris dans le cadre des environnements intégrés de conception se sont alors focalisés sur l'intégration par les données (cf. §2.1.2.2) puis plus récemment sur l'intégration par les processus (cf §2.1.2.3).

2.1.2.2. Intégration par les données.

L'approche dite d'intégration par les données a fait un moment l'unanimité. Elle consiste à mettre en œuvre des systèmes de gestion de données offrant une base de données commune et partagée par les différents outils logiciels participant à l'élaboration du produit. Chaque traitement accède aux informations qui lui sont nécessaires et alimente la base par de

[9] CIM (Computer Integrated Manufacturing) est une expression caractérisant "l'intégration en entreprise de tous les éléments impliqués dans la production par des moyens informatiques" [Doumeingt 90]. Ces environnements présentent de fortes analogies avec ceux destinés à la production des logiciels [Lonchamp 92].

nouvelles informations. Les avantages d'une intégration par les données sont évidents : rapidité et sûreté de la transmission, souplesse en cas d'intégration d'un nouvel outil, une modélisation unique des données, etc. Cependant, cette approche s'est rapidement heurtée à la difficulté de disposer de systèmes de gestion de données suffisamment puissants pour prendre en compte la modélisation et la gestion d'objets composites, à représentations multiples et fortement évolutifs.

En effet, un produit en cours de conception n'est informationnellement stabilisé qu'à l'issue d'une série de transformations et de choix. La première de ces transformations aboutit à la modélisation du cahier des charges, la dernière à celle de l'objet conçu. Chaque étape permet de concrétiser, d'affiner ou de préciser les connaissances détenues sur l'objet. Les informations relatives aux produits en cours de conception sont :

- de natures différentes. Un produit admet plusieurs représentations correspondant à des niveaux morphologiques différents et certaines étapes de conception ont pour rôle d'autoriser le passage d'un niveau morphologique à un autre. C'est par exemple le cas, en conception mécanique, de l'étape de satisfaction d'une fonction par des éléments structurels (pièces de catalogues, composants, etc.). Une fonction est la plupart du temps réalisée par l'assemblage de plusieurs éléments structurels. Inversement un élément structurel intervient dans la réalisation de plusieurs fonctions.
- de niveaux d'abstraction différents. Chaque représentation de l'objet est perçue à des niveaux d'abstraction différents et certaines étapes de conception ont pour rôle de passer d'un niveau d'abstraction à un autre moins élevé. C'est le cas des étapes de décomposition (par exemple d'une fonction en sous-fonctions). Un ensemble d'informations de même nature (par exemple fonctionnelle ou structurelle) est généralement représenté par un graphe de décomposition décrivant les différentes nomenclatures (fonctionnelle, structurelle, logistique) du produit. Bien entendu, ces différents graphes ne sont pas isomorphes.
- de niveaux de précision différents. Toute étape de conception produit de nouvelles informations permettant ainsi de compléter une ou plusieurs représentations de l'objet. C'est souvent le cas des étapes de simulation d'objets mécaniques dont les résultats permettent de préciser le comportement des éléments fonctionnels et structurels.

La prise en compte d'objets multi-représentés et évolutifs nécessite donc de représenter et de gérer **l'évolution simultanée de ses différentes représentations**. Il s'agit de maintenir des liens de composition intra-représentation dont la sémantique et les propriétés sont variables d'une représentation à l'autre (propriété de variance, d'interchangeabilité, etc.) mais aussi des relations de dépendance inter-représentations ("est réalisé par" entre les nomenclatures fonctionnelle et structurelle, "est maintenu par" entre la nomenclature structurelle et la

nomenclature logistique, etc.). Il s'agit également de maintenir les règles de dépendance, de calcul, de compatibilité entre les contenus informationnels de ses différentes représentations. Le maintien de telles règles est particulièrement difficile dans le cadre d'objets évoluant en valeur mais aussi en structure.

Dans les années 80, de nombreux travaux ont eu pour objectif l'élaboration de "modèles de données à sémantique forte" par opposition "aux modèles de données neutres" évoqués précédemment, ces modèles s'inspiraient en grande partie des concepts issus des systèmes orientés objet [Cholvy 83]. Les travaux entrepris dans le cadre de CADB relèvent de cette mouvance (cf. §2.2).

Si les modèles de produits ont d'ores et déjà fait de gros progrès, la mise en place de structures de données communes s'est rapidement révélée insuffisante en ne permettant ni le contrôle, ni la coordination des différents outils au sein des processus de développement. Différentes approches ont alors été proposées, différents termes employés : on a parlé d'intégration par les connaissances, d'intégration du contrôle, d'intégration par les procédés ou les processus, etc.

2.1.2.3. Intégration par les processus

Il s'agit d'offrir des modèles et des fonctionnalités permettant la modélisation et l'évolution des produits mais aussi :

la modélisation et l'exécution des traitements (méthodes de calcul, outils logiciels...) intervenants sur ces produits,

la modélisation et le pilotage des processus de développement, c'est à dire l'utilisation par des agents humains des traitements pour avancer dans l'activité de production [Trichon 91].

La représentation des processus de développement, bien que fondamentale pour un réel contrôle des activités d'ingénierie, soulève d'énormes difficultés liées à la complexité et à la diversité des processus. En effet, sauf cas de conception extrêmement routinière, il existe rarement une bonne stratégie de résolution de problèmes de conception. Par exemple en mécanique, pour des conceptions non innovantes où la structure de l'objet est définie a priori, il est possible d'effectuer immédiatement une simulation de l'objet. Par contre, si la structure de l'objet n'est pas connue, une décomposition fonctionnelle par un technologue est un préambule indispensable à toute simulation. La représentation et la gestion des processus de développement nécessitent également de prendre en compte les différents rôles des acteurs intervenant durant ces processus. Il s'agit en particulier d'offrir des systèmes multi-vues où chaque acteur détient sa propre perception du produit en cours d'élaboration. Un produit

mécanique est, par exemple, perçu différemment par le technologue, l'expert en simulation et l'expert en cotation.

La modélisation simultanée des produits, de leur processus de conception et donc implicitement des outils et des acteurs intervenant lors de ce processus a été un des objectifs du projet SHOOD (cf. §2.3). De grands progrès restent encore à faire dans le domaine de la représentation des produits et des processus de conception. Les modèles de processus n'en sont qu'à leur début et leur couplage avec les modèles de produits reste encore problématique [Harani 97].

2.1.2.4. Outils industriels orientés objet

Il existe aujourd'hui de nombreux outils orientés objet dédiés à la CAO [Trousse 97]. Certains sont destinés aux concepteurs d'outils de CAO, c'est par exemple le cas des environnements ANAXAGORE [Trousse 89], CASCADE de Matra Datadivision et MTEL de Caroline Informatique. Citons, dans la même approche, le projet Esprit DEKLARE (Ilog, Peugeot, etc.) [Vargas 95] dont l'objectif est d'offrir un environnement de développement d'applications d'aide à la conception intégrant des structures à objet, des représentations géométriques et des mécanismes de propagation de contraintes. Son langage de modélisation des connaissances s'appuie sur un modèle de produit et un modèle de processus. D'autres outils, destinés aux concepteurs de produits, sont généralement dédiés à un domaine d'applications. C'est par exemple le cas d'ARCHIX (Régie Renault 1990) dont l'objectif est d'aider la conception d'un véhicule en phase d'avant-projet et d'EXPORT (1991) qui offre un environnement pour la conception des avant-ports.

La problématique d'intégration des produits et des processus de CAO est très similaire à celle apparue dans les années 90 dans les domaines des Systèmes d'Information [Rolland 95, Roland 97, Rolland 97] et du Génie Logiciel [Oquendo 90, Lonchamp 92, Oivo 92, Melo 93, Finkelstein 94] où l'on parle de plus en plus d'ingénierie des méthodes, d'ingénierie des processus, de (méta) modèles de processus, etc.

2.1.3. Ingénierie des Méthodes

Les méthodes d'analyse et de conception des systèmes d'information et/ou de logiciels (Merise, OMT, UML, etc.) proposent :

- un ensemble de modèles (formalismes) permettant de représenter les systèmes sous différents aspects (statique, dynamique, fonctionnel) et à différents niveaux d'abstraction (expression des besoins, analyse, conception). Le terme modèle de produits (cf. Figure 2-1) est généralement utilisé pour désigner un formalisme (ex : le modèle E/A), permettant

de décrire un produit (ex : le modèle E/A d'un système d'information bancaire). Dans les méthodes d'analyse et de conception, les modèles de produits proposent des formalismes graphiques généralement bien spécifiés.

- une (des) démarche(s) ou processus permettant de guider les activités de développement. Elles sont constituées d'une succession d'étapes permettant en particulier de passer d'une modélisation du système cible à une autre (d'un produit à un autre au sens de la Figure 2- 1). Ces démarches sont souvent décrites de manière trop imprécise. Pensez par exemple à l'une des premières phases communes à de nombreuses méthodes orientées objet : "Identifier les objets". D'autres démarches sont par contre trop rigides. C'est le cas de Merise dont la démarche est constituée d'une longue succession d'étapes inadaptée aux petites applications. Cette rigidité est d'ailleurs la source de très nombreuses adaptations de la méthode en fonction de la taille mais aussi de la nature des applications.

De nombreux travaux ont aujourd'hui pour objectif de modéliser et d'exécuter les démarches de développement des SI. La représentation explicite des démarches de développement produit des modèles de processus (cf. Figure 2- 1) qui facilitent le contrôle des activités de développement des SI et éventuellement l'automatisation de certaines tâches [Rolland 97]. Elle permet d'expliciter les liens de cohérence ou de déduction entre les modèles de produits. Elle est également une source potentielle d'évolution et d'adaptation des méthodes [Rolland 98]. Chaque méthode offre des modèles de produits et éventuellement de processus. Ces modèles sont utilisés par les ingénieurs d'applications au travers d'outils CASE (Computer Aided Software Engineering).

Changeons à présent l'enjeu du développement. L'objectif n'est plus de développer des applications (SI, logiciel, etc.) mais des méthodes, on parle alors d'ingénierie des méthodes. Quel que soit le domaine d'ingénierie, la modélisation reste une des techniques les plus éprouvées pour spécifier des systèmes. L'appliquer à ces systèmes particuliers que sont les modèles nécessite de disposer d'un plan de modélisation supplémentaire : on parle de méta-modélisation, autrement dit de modélisation de modèles.

Un méta-modèle (de produits ou de processus), par exemple *M*eninge sur la Figure 2- 1, propose un ensemble de concepts permettant à l'ingénieur des méthodes de décrire des modèles (de produits ou de processus). La Figure 2- 1 illustre les trois niveaux offerts par la plupart des méta-outils (MetaEdit+ [Smolander 91], GraphOR [Morejon 91], GraphTalk [Parallax 94], MetaGen [Blain 94], Tramis [Hainault 92], Mentor [Grosz 96], etc.).

Le lien "est décrit par", qu'il s'agisse d'un acte de modélisation ou de méta-modélisation, dépend bien entendu des règles « d'instanciation » du modèle ou du méta-modèle utilisé. Les définitions proposées sont inspirées de [Grosz 96].

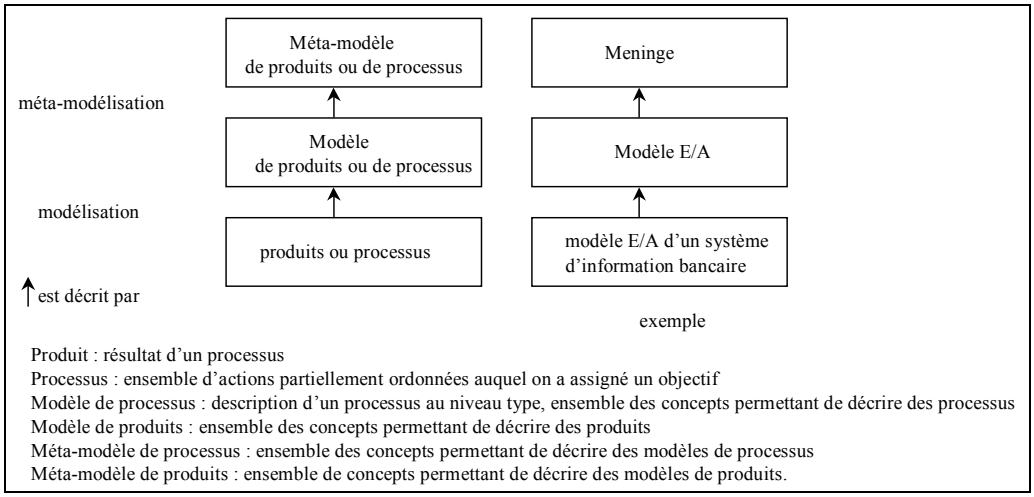


Figure 2- 1: Trois niveaux de modélisation

Ces différents niveaux de modélisation sont aujourd'hui classiques [Bézivin 99]. L'OMG (Object Management Group), par exemple, a proposé UML comme langage de modélisation unifié, mais aussi MOF (Meta-Object Facility) comme méta-langage de modélisation [OMG/MOF 97]. MOF offre un sous-ensemble des concepts d'UML (correspondant au paquetage CORE) et s'auto-décrit. Il est utilisé pour représenter UML mais aussi d'autres formalismes de modélisation. C'est par exemple le cas de MCC (Modèle Corba de Composants) ou de UML_FBO (UML for Business Objects) [Bézivin 99].

L'originalité du projet *M*eninge était d'utiliser les trois niveaux d'objets offerts par les systèmes à objet réflexifs (méta-classe, classe, instance) pour proposer un environnement uniforme où tout modèle (par exemple le modèle E/A), mais aussi tout produit (par exemple le modèle E/A d'un SI bancaire) serait exprimé comme une extension du méta-modèle.

Par la suite, nous donnons une rapide description des projets CADB (§2.2) et SHOOD (§2.3) dédiés aux applications de CAO, puis du projet *M*eninge (§2.4) dédié à l'ingénierie des modèles.

2.2. L'expérience CADB

Ce projet a été mené dans le cadre de l'équipe Base de Données du LGI, dirigée par Michel Adiba.

2.2.1. Un système de gestion de bases de données

La gestion de données techniques occupe depuis longtemps une place de choix dans mes travaux [Rieu 85, Rieu 86c, Nguyen 92, Rieu 94, Rieu97a, Gzara99]. Une première expérience, menée dans le cadre de la conception de circuits VLSI, a abouti à l'élaboration de CADB, un système de gestion de bases de données pour les applications CFAO.

Ce projet est parti du constat des lacunes des SGBD traditionnels, en particulier les SGBD relationnels, pour prendre en compte la sémantique et l'évolutivité des applications CAO. La nécessité, par exemple, de prendre en compte des objets composites, de modéliser dans la base les outils ad hoc et d'offrir des mécanismes permettant une conception aussi bien ascendante que descendante nous a conduit à proposer un modèle et des fonctionnalités qui se sont révélés très proches de ceux offerts par les systèmes de représentation de connaissances à objet issus de l'Intelligence Artificielle et en particulier par les systèmes à base de frames [Rechenmam 88].

Le langage de description et de manipulation de CADB [Rieu 85] introduisait par exemple le concept d'ensemble d'objets, similaires à celui de frames, organisés en une hiérarchie de spécialisation (cf. Figure 2- 2: les ensembles Segment, Triangle et Triangle-rectangle-isocèle). Les programmes ad hoc étaient modélisés par des fonctions (par exemple les fonctions translation-vertical et égal) similaires au concept de méthode. La structure d'un ensemble était définie par un ensemble de caractéristiques similaires au concept d'attribut. Un attribut pouvait prendre ses valeurs dans un ensemble d'objets, ce qui permettait la prise en compte des objets composites. Sur chaque attribut, il était possible d'exprimer des contraintes d'intégrité et des règles de déduction très proches de certaines facettes procédurales des systèmes à base de frames.

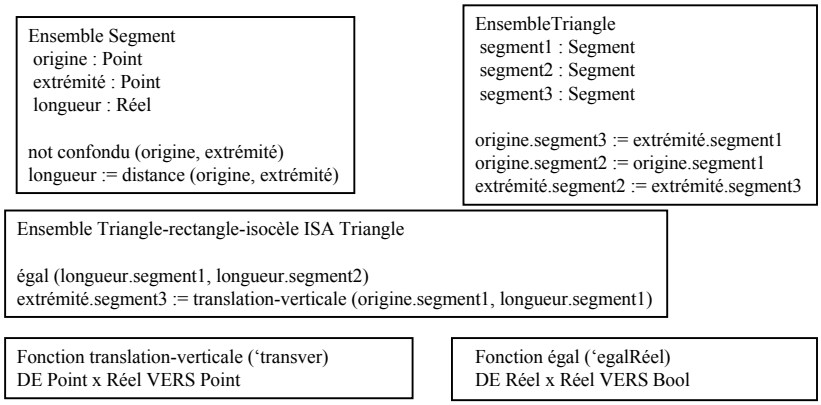


Figure 2- 2: Quelques exemples d'ensembles et de fonctions CADB

Cette étude, menée dans le cadre du projet TIGRE conjointement avec le CNET de Meylan (Département Recherche en Conception Assistée), a donc permis de spécifier un modèle de base de données mais aussi des fonctionnalités adaptées aux applications CAO.

Le pivot du système CADB était un outil intégré de description et de manipulation des objets CAO. Un environnement de conception construit autour de ce noyau intégrait les concepts d'alternative, de version et de représentation multiple d'objet en cours d'élaboration. Cet environnement offrait déjà un certain nombre de fonctionnalités permettant une réelle gestion de projets et d'échanges de données entre concepteurs. Un prototype a été utilisé comme support de différentes expérimentations dans le domaine de la conception architecturale et dans celui des circuits VLSI.

Un travail important, réalisé dans le cadre de la thèse de Marie-Christine Fauvet, a abouti à la spécification et l'implantation d'un modèle de versions et de transactions permettant à un groupe de concepteurs de coopérer en vue de l'élaboration d'un même produit [Fauvet 87, Fauvet 88].

2.2.2. Des BD vers l'Intelligence Artificielle

En 1986, le bilan de l'expérience CADB pouvait se résumer en trois points.

- L'approche "objet" semblait adéquate pour la prise en compte d'objets complexes et multi-représentés.
- CADB n'offrait pas de fonctionnalités permettant une évolution fine et assistée des valeurs et des structures des produits en cours d'élaboration.
- CADB n'était que difficilement évolutif. La plupart des concepts du modèle étaient codés. Tout nouveau concept nécessitait donc une révision complète du prototype.

Cette dernière lacune était gênante à plusieurs niveaux. Tout d'abord en terme de prototype de recherche, nous n'étions jamais sûrs d'introduire le "bon" concept. Sur un plan plus applicatif, nous ne voulions pas offrir un système dédié à un type d'applications donné. Prendre en compte un nouveau métier avec ses besoins spécifiques, nécessitait donc la génération d'un nouveau CADB avec des concepts légèrement différents. Il s'agissait donc de rendre évolutif ce que quelques années plus tard nous appellerions le méta-modèle.

Le second point, l'évolution des valeurs et des structures d'objets, nous a occupé très longtemps (projet SHOOD). L'idée consistait à développer des fonctionnalités permettant une évolution cohérente et assistée d'objets en cours d'élaboration. En effet, l'objet en cours de conception ne peut être stabilisé qu'à l'issue de son processus d'élaboration, processus durant lequel il va passer par différents stades d'incomplétude et d'incohérence temporaires [Rieu 86b, Rieu 86c]. CADB ne permettait à ce niveau qu'une gestion triviale d'objets incomplets et incohérents : un attribut d'objet pouvait ne pas être instancié, de même qu'une contrainte pouvait être définie comme faible et donc temporairement violable. Aucune assistance n'était offerte à l'utilisateur pour le guider dans son activité de conception.

Ces fonctionnalités, fondamentales dans le cadre d'applications de conception, nous ont poussé à étendre CADB en un Système de Bases de Données Expert offrant non seulement une modélisation adéquate des objets manipulés mais aussi des connaissances expertes utilisées pour le contrôle automatique de la cohérence [Rieu 87a] et le calcul des effets de bord lors de la mise à jour des informations [Nguyen 87b].

Ce travail mené en collaboration avec des étudiants en DEA et Nguyen Gia Toan, directeur de recherches à l'INRIA, mettait en oeuvre des techniques tirant parti des fonctions bien connues des SGBD et des techniques inspirées de l'Intelligence Artificielle. Le résultat s'est soldé par un nouveau prototype de CADB reposant sur deux composants :

- une base de données contenant les diverses représentations des objets, par exemple logique, fonctionnelle et électrique pour des circuits VLSI.
- une base déductive contenant des règles opératoires concernant la modification des structures des objets, les règles de conception propres à l'application et des heuristiques concernant les contrôles automatiques de cohérence [Rieu 87b].

Ces deux composants permettaient la détection automatique des conséquences des mises à jour sur l'intégrité des informations. Ils étaient utilisés comme garde-fous afin de prévenir les conséquences inopportunes d'actions erronées. Cette étude a débouché sur la définition et

l'expérimentation d'un modèle d'objets dynamiques. Son originalité tient à la répercussion automatique et cohérente des modifications demandées par un usager sur les données interdépendantes.

Une méthode de contrôle de cohérence à deux phases (certification puis validation) et de contrôle de la légalité des opérations de mise à jour a été définie [Nguyen 88c]. La propagation et le contrôle des modifications de schémas sur les définitions de données et sur les instances d'objets a fait l'objet d'une étude postérieure qui a été poursuivie dans le cadre du projet Shood.

2.3. Systèmes de Représentation de Connaissances : SHOOD

Ce travail a été mené dans le cadre d'un projet INRIA-IMAG dirigé par François Rechenman et dans l'équipe Bases de Connaissance du LGI dirigée par Nguyen Gia Toan.

En 1988, l'étude abordée sur la dynamique des bases de données dans le cadre du système CADB a abouti à la définition du projet SHERPA[10] dont le thème majeur était la dynamique dans les bases de connaissances à objet. Ce projet réunissait deux équipes, l'une dirigée par F. Rechenman et appartenant au laboratoire ARTEMIS, l'autre dirigée par G.T Nguyen, appartenait au laboratoire LGI.

La motivation première de ce projet était de rassembler deux équipes travaillant respectivement dans les domaines des bases de données et de l'intelligence artificielle. Cette fusion, devenue possible grâce aux progrès effectués dans les domaines des bases de données et des systèmes de représentation de connaissances, s'est organisée autour du concept unificateur d'objet. L'expérience CADB nous avait montré l'intérêt des concepts d'objets pour les applications CAO. L'objectif plus général de SHERPA était de fournir un support à des applications dont les exigences opérationnelles n'étaient pas entièrement satisfaites par les SGBD ou les systèmes de Représentation de Connaissances de l'époque.

Les travaux menés au sein de l'équipe se sont articulés autour de la spécification et l'expérimentation d'un modèle de représentation des connaissances à objet (SHOOD). Nos objectifs étaient alors :

- de renforcer le pouvoir déclaratif des modèles à objet traditionnels. A ce titre nous nous sommes particulièrement intéressés à la modélisation des relations sémantiques inter-

10 SHERPA : projet IMAG-INRIA créé en janvier 1988 et accepté par le comité d'évaluation des projets de l'INRIA en juin 1988.

objets et des représentations multiples des objets. Les concepts de relations et de représentations multiples sont fondamentaux dans les applications de conception pour prendre en compte des sémantiques particulières de liens de composition et l'évolution morphologique des objets en cours de conception. Cette étude a fait l'objet de la thèse de Chabane Djeraba [Djeraba 93].

- de poursuivre l'étude abordée, dans CADB, sur la dynamique des données. Les aspects dynamiques ont été abordés suivant trois aspects : la *dynamique des instances* (cf. §2.3.1) consistant en particulier à autoriser l'évolution des classes de rattachement des instances, la *dynamique des schémas*[11] (cf. §2.3.2) permettant l'évolution des classes et des graphes de classes et la *dynamique du modèle de connaissances* (cf. §2.3.3) offrant par exemple la possibilité de définir des sémantiques particulières d'association. L'étude des mécanismes d'évolution a fait l'objet de la thèse de Fethi Bounaas [Bounaas 95].

Par la suite seules les études portant sur la dynamique des données seront abordées.

2.3.1. Dynamique des instances

L'appartenance d'une instance à une classe dépend des valeurs de ses attributs. Quand ces dernières sont modifiées, les classes de rattachement sont susceptibles d'évoluer. De plus, l'étude de cette évolution doit prendre en compte le fait que, dans un contexte d'applications constructives telles que la CAO, ces instances peuvent modéliser des objets en cours d'élaboration : elles sont alors incomplètes, temporairement incohérentes, multi-représentées et dépendantes fonctionnellement ou existentiellement d'autres instances.

Notre objectif était donc ici d'améliorer et de généraliser les travaux effectués dans CADB sur les évolutions dynamiques d'objets incomplets et incohérents. En particulier, CADB gérait l'évolution d'une instance dans sa classe, dans Shood l'idée était de prendre en compte l'évolution dynamique de l'objet dans un graphe de classes.

Plus particulièrement les travaux entrepris dans cet axe nous ont poussés à remettre en cause un certain nombre d'hypothèses concernant l'instanciation.

- Pour supporter la multi-représentation des objets, nous avons autorisé une instance à appartenir à plusieurs classes.

[11] Un schéma est constitué des classes et des relations entre elles (association, héritage, etc.).

- Pour supporter des objets incomplets et incohérents, nous avons autorisé les instances à ne plus être des moulages parfaits des entités conceptuelles qui les décrivent (violation possible de contraintes dites faibles, non valuation de certains attributs, etc.).

Le concept d'instanciation par moulage parfait a été remplacé par celui de multi-instanciation par moulage souple.

Un mécanisme de classification s'appuyant sur ce principe a été spécifié dans le cadre de la thèse de Fethi Bounaas [Rieu 91, Bounaas 95] et réalisé dans le cadre du mémoire CNAM de Marie-Pierre Liotard [Liotard 93]. Les mécanismes de classification ont pour objectif de chercher la classe d'appartenance d'un objet en fonction des informations connues sur cet objet [Haton 91, Marino 91, Girard 95]. Le processus d'appariement est généralement régi par les invariants d'instanciation du système ; une classe est sûre pour un objet si celui-ci est complet et cohérent dans cette classe (si par exemple tous les attributs sont valués et toutes les contraintes sont respectées), une classe est possible si la comparaison entre la classe et l'objet ne peut être complète mais qu'aucune contradiction n'est apparue, une classe est impossible si les invariants d'instanciation sont violés (l'objet viole une contrainte, détient une propriété non définie dans la classe, etc.).

Le mécanisme proposé dans Shood avait la particularité d'une part de reposer sur la multi-instanciation par moulage souple et d'autre part d'utiliser les inférences définies dans les classes afin de compléter les informations disponibles sur les objets à classifier. Tout objet à classifier peut ainsi être complété de différentes manières dans toutes les classes où son appariement est possible. Des degrés d'incomplétude et d'incohérence mesurant la "distance" aux classes permettent ensuite de choisir la ou les classes de rattachement.

Appliqué aux objets CAO, ce mécanisme permet de raisonner sur le produit à concevoir à un certain niveau de résolution afin de dériver des représentations plus complètes et donc plus proches de l'objet conçu. Ce mécanisme était également utilisé pour réaliser des fonctionnalités propres au système telles que la liaison dynamique (cf. §2.3.3.2).

2.3.2. Dynamique des schémas,

Il s'agissait ici d'autoriser les manipulations sur des classes (par exemple de rajouter ou de supprimer un attribut) ou sur des graphes de classes (par exemple de rajouter ou de supprimer un lien de spécialisation) [Nguyen 88b, Nguyen 89b]. De très nombreux travaux de recherche aussi bien en BD qu'en IA ont mis en évidence trois grandes approches [Bounaas 95, Nguyen 97] :

- La réorganisation

Les techniques mises en œuvre font appel à des algorithmes spécifiques [Lieberherr 89, Casais 89] ou généraux, par exemple des mécanismes de classification de classes [Napoli 92], permettant de réorganiser des hiérarchies de classes.

- La correction

Cette approche [Nguyen 89b] consiste à proposer une taxonomie d'opérations de manipulation de schémas et d'invariants à préserver à l'issue de chaque manipulation. Une stratégie de propagation des modifications permet de répercuter les mises à jour sur les instances. Aucune trace de l'état des classes avant manipulation n'est préservée. Cette approche a été mise en œuvre dans de nombreux SGBDOO : Orion [Banerjee 87], GemStone [Penney 87], OTGEN [Lerner 90] et O2 [Zicari 91, Delcourt 92].

- Le versionnement

Il s'agit ici de conserver l'état des classes et parfois plus généralement d'un graphe de classes avant manipulation. C'est le cas d'Orion qui autorise le versionnement des instances, des classes et des schémas. Cette approche est particulièrement suivie dans les environnements multi-concepteurs comme par exemple dans les ateliers de génie logiciel [Ahmed-Nacer 94] et dans les systèmes de gestion de données techniques [Randoing 95].

Ces trois techniques présentent l'inconvénient commun de n'offrir qu'une seule stratégie d'évolution de schémas. Dans le cadre du projet SHOOD nous souhaitions rendre déclarative ces stratégies de manière à faciliter leur adaptation en fonction des besoins des applications. Prenons l'exemple d'une suppression de classe comportant des instances. Plusieurs stratégies sont possibles : interdire cette suppression, supprimer les instances, reclassifier les instances (par exemple dans la super-classe). Dans sa thèse, F. Bounaas a particulièrement mis l'accent sur l'utilisation de règles actives Evénement-Condition-Action pour gérer une évolution déclarative des évolutions de schémas. L'expression de règles d'évolution de schémas a été grandement facilitée par l'approche " tout objet " de SHOOD (cf. §2.3.3.3).

2.3.3. Dynamique du modèle

Il s'agissait ici d'autoriser une évolution des concepts même du modèle par exemple la possibilité de rajouter dynamiquement un nouveau type de lien de composition ou de dépendance. Cet objectif nous a conduit à faire de SHOOD un système réflexif. Ce choix garantissait l'évolution du modèle de représentation de connaissances et permettait d'uniformiser le traitement et la gestion de la dynamique des schémas et du modèle lui-même. Pour cela, le modèle suit l'approche " tout objet " comme ObjVlisp [Cointe 87] ; un objet peut être une connaissance élémentaire (" Dupond "), une connaissance générique (" toute personne a une tête "), une méthode (" la différence entre deux entiers "), une inférence (" l'âge d'une personne est calculé par la différence entre l'année courante et l'année de sa

naissance "). Le modèle est réflexif : il possède la propriété de se décrire lui-même. Tous les concepts tels que les classes, les attributs, les méthodes sont décrits et implantés à partir d'un noyau minimal de modélisation constituant l'amorce du système.

Les classes `Objet`, `Méta` et `Méta-attribut` constituent les principales classes du noyau de modélisation (cf. Figure 2- 3). La classe `Objet` est la racine du graphe d'héritage, elle définit l'attribut `instance_de` caractérisant l'appartenance des objets (instances, classes et méta-classes) à leur classe. Les informations nécessaires pour la génération automatique des classes du système sont contenues dans la définition d'une méta-classe (`Méta`) détenant les attributs minimaux caractérisant les classes : `nom_classe`, `super`, `sous`, `attributs` et `instance_de`. Les attributs sont représentés par des classes, instances de la méta-classe `Méta_attribut`. La classe `Méta_attribut` définit donc les attributs nécessaires à la création d'un attribut tels que `nom_attribut`, `domaine` et `constructeur` (un, ens, etc.).

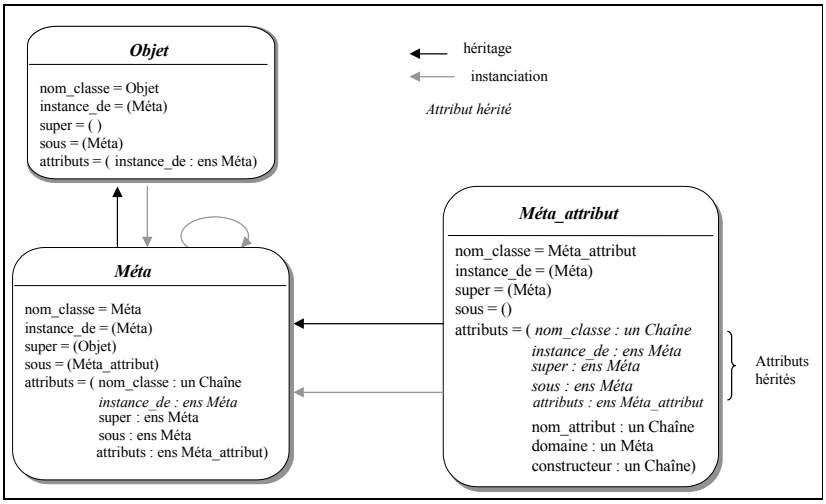


Figure 2- 3 : Les classes Objet, Méta et Méta_attribut

L'ensemble des classes et des méta-classes constitutives de l'amorce sont organisées en un graphe d'instanciation et d'héritage qui constituent le minimum de connaissances pour que le modèle " fonctionne tout seul ". La Figure 2- 4 présente le noyau de modélisation minimal. Dans SHOOD, comme dans ObjVlisp, un objet est une classe s'il hérite de la classe `Objet` et un objet est une méta-classe s'il hérite de la classe `Méta`.

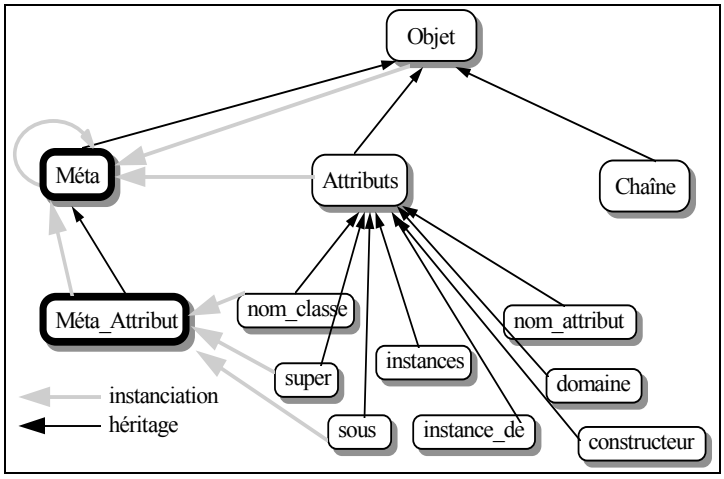


Figure 2- 4 : Le graphe d'héritage et d'instanciation minimal

Les méta-classes permettent de définir des modèles de classes. Dans certains systèmes tels que SmallTalk, la génération des méta-classes est implicite ; le concepteur n'a pas accès à ces modèles et ne peut donc pas les adapter à son problème. D'autres systèmes autorisent l'accès au méta-modèle et aux mécanismes sous-jacents ; dans ce cas, le système étant totalement "ouvert ", le concepteur peut étendre et adapter soit le modèle de base, soit les mécanismes associés tels que l'héritage ou l'instanciation.

C'est l'approche que nous avons suivie dans SHOOD où tout concept a été décrit comme une extension du noyau. Les méta-classes sont ainsi utilisées pour étendre le modèle ou pour l'adapter aux besoins d'une application. Le noyau minimal de SHOOD a ainsi été étendu pour supporter les relations sémantiques (cf. § 2.3.3.1), les méthodes (cf. §2.3.3.2) et les règles actives (cf. §2.3.3.3). La Figure 2- 5 illustre les trois plans de modélisation de SHOOD.

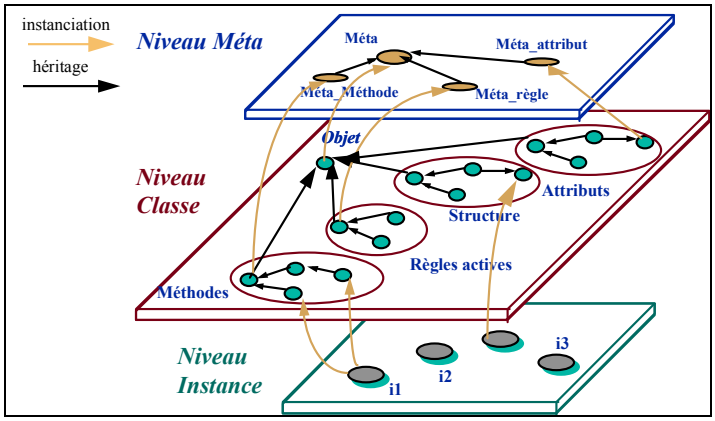


Figure 2- 5 : Les trois niveaux du modèle SHOOD

2.3.3.1. Les relations sémantiques

Dans un modèle orienté objet, la notion d'attribut permet de définir la structure d'un objet ; les attributs ont généralement la même sémantique. Il est parfois nécessaire d'exprimer une sémantique particulière dénotant par exemple une dépendance existentielle ou une règle de composition entre un objet et ses valeurs d'attributs. Dans ce cas, l'attribut est perçu comme une relation sémantique. Le modèle SHOOD propose de modéliser les attributs par des classes instances d'une méta-classe particulière (`Méta_attribut`). Suivant la sémantique à exprimer, la classe modélisant l'attribut est instance d'une sous-classe de `Méta_attribut` modélisant la sémantique appropriée [Djeraba 93]. La Figure 2- 6 illustre une dépendance existentielle entre les classes *Voiture* et *Moteur* exprimée par l'attribut " moteur ". Cet attribut est modélisé par une classe instance de la méta-classe `Méta_att_existentiel`.

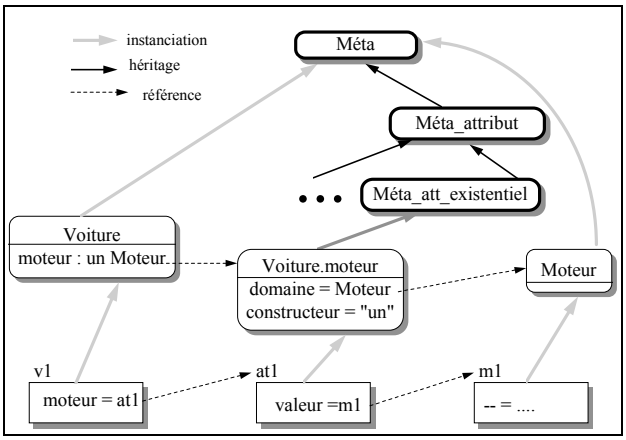


Figure 2- 6 : Modélisation des dépendances existentielles dans SHOOD

La représentation des attributs par des classes permet de réutiliser les mécanisme inhérents aux classes (héritage et instanciation) et d'accroître la sémantique d'un attribut. C'est ainsi que la redéfinition d'un attribut se traduit par une spécialisation de la classe modélisant cet attribut. De même, donner une valeur à un attribut revient à instancier la classe le représentant.

2.3.3.2. Les méthodes

Une approche similaire nous a conduit à réifier les méthodes par des classes. La surcharge d'une méthode se traduit alors par une spécialisation de la classe modélisant cette méthode. De même, exécuter une méthode revient à instancier la classe la représentant. Finalement, la sélection de la bonne méthode à exécuter en fonction des types des arguments est obtenue en classifiant l'instance modélisant la méthode dans un graphe de méthodes de même sémantique. Le mécanisme de classification (cf. § 2.3.1) a donc été utilisé pour réaliser la liaison dynamique.

La Figure 2- 7 illustre en partie gauche un graphe de méthodes. Les méthodes sont organisées en fonction du nombre et du type de leur paramètres. On suppose ici que la classe Entier est une sous-classe de la classe Réel. La partie droite illustre le mécanisme de sélection des méthodes. L'instance (`différence(3,2)`) modélisant l'appel d'une méthode est classifiée dans le graphe de méthodes. Le mécanisme de classification sélectionne l'ensemble des méthodes compatibles avec les paramètres effectifs puis retient la classe méthode la plus spécialisée et donc la plus adéquate pour effectuer le calcul.

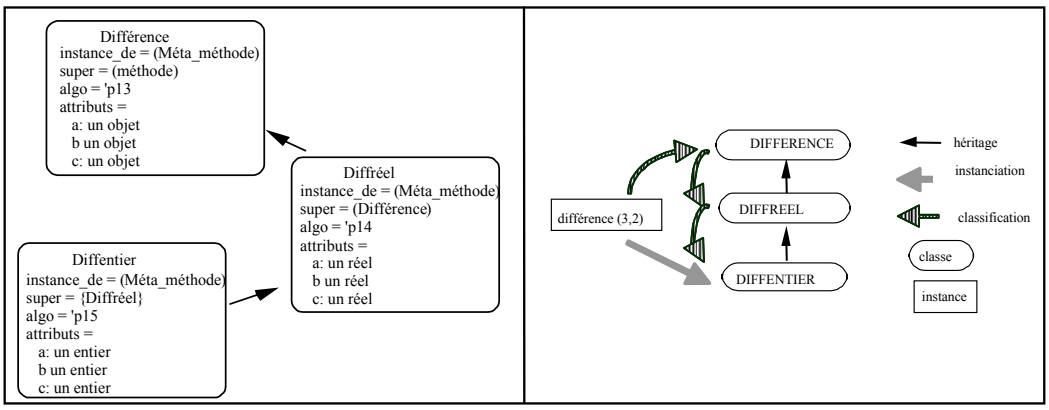


Figure 2- 7 : Graphe et sélection des méthodes

2.3.3.3. Les règles actives

Le modèle SHOOD propose également un modèle de règles actives permettant aux concepteurs d'applications de modéliser des contraintes et des comportements réactifs et aux concepteurs du système d'exprimer de manière déclarative l'évolution des objets. Et pour " appliquer la réflexivité à fond ", ce mécanisme de règles est utilisé pour gérer l'évolution des règles elles-mêmes, les règles étant modélisées par des classes instances d'une méta-classe `Méta_règle`. Les règles sont organisées dans des bases de règles, qui permettent de hiérarchiser et de structurer le comportement actif des objets. La règle " Règle_suppression " (cf. Figure 2- 8) exprime que la suppression d'une base de règles entraîne la suppression des règles de cette base.

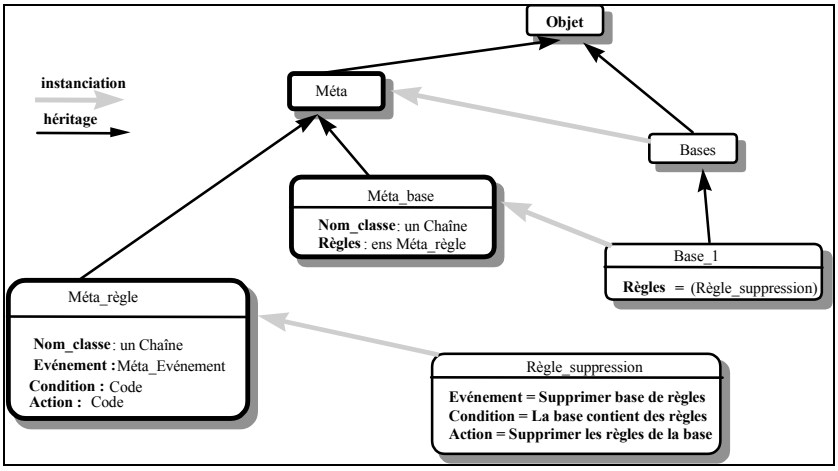


Figure 2- 8 : Un exemple de règle

2.3.4. Bilan

En 1993, SHOOD proposait un modèle objet basé sur les concepts de classes, méta-classes, spécialisation multiple, multi-instanciation, attributs relations (composition, dépendances...), fonctions génériques et règles événementielles. Il avait été expérimenté par des concepteurs de métier bien au fait des techniques de représentation de connaissances. De nombreux travaux interdisciplinaires nous ont permis d'acquérir une démarche pragmatique pour capitaliser, transférer et modéliser des produits en cours d'élaboration. Ces expérimentations ont porté sur des exemples de conception mécanique (par exemple la conception d'une fixation de surf) et électrotechnique (par exemple la conception de machines asynchrones).

Le bilan de cette expérience de cinq ans peut se résumer en deux points.

- Les modèles à présent classiques offerts par les SRCOO et les SGBDOO (règles, objets, relations, etc.) se sont révélés adéquats pour prendre en compte les connaissances structurelles et comportementales des produits en cours d'élaboration. Ils sont par contre insuffisants pour la prise en compte des processus d'ingénierie. Nos concepteurs ont eu d'énormes difficultés pour représenter leurs processus de conception et ceci pour deux raisons. La première est liée à l'absence d'une vision globale du modèle du processus représenté par une multitude de règles et de méthodes. La deuxième est liée à la nécessité de comprendre les mécanismes de résolution du système de représentation de connaissances (classification, unification, etc.). La nécessité de disposer de modèles de produits et de processus de haut niveau d'abstraction nous a poussé quelques années plus tard à développer un cadre méthodologique adapté à l'ingénierie des Systèmes d'Information Produit des entreprises industrielles. Le projet POSEIDON tire parti des

techniques de modélisation et de réutilisation développées dans le domaine des SI. Il sera décrit au chapitre suivant.

- Le plus grand succès du projet SHOOD est d'avoir poussé à fond et avec succès l'approche du " tout objet ". En effet, les principes inhérents aux systèmes à objet réflexifs se sont révélés pertinents pour adapter le système à des domaines d'ingénierie aussi divers que la conception mécanique, électronique et architecturale. Nos différents utilisateurs avaient des versions du système relativement différentes : relations sémantiques spécifiques, propagation personnalisée des opérations d'évolution des schémas, etc. Cette facilité d'adaptation nous a conduit à expérimenter cette approche dans un domaine d'ingénierie particulier, celui des méthodes (cf. §2.4).

2.4. Environnement de méta-modélisation : *M*eninge

Ce travail a été mené dans le cadre du laboratoire LSR de l'IMAG en co-responsabilité avec Ph. Morat.

L'objectif du projet *M*eninge concernait la spécification et le développement d'un environnement permettant aux organisations de spécifier, d'implanter puis d'utiliser des modèles et des méthodes. Nous nous placions donc dans un cadre d'ingénierie des modèles qui consiste à appliquer les techniques d'ingénierie à ces systèmes particuliers que sont les méthodes [Freire 95, Saeki 95, Plihon 96, Kelly 96, Revault 95, Parallax 94]. La principale approche utilisée est celle de la méta-modélisation, c'est à dire de la modélisation des modèles (cf. §2.1.3).

L'objectif initial était d'utiliser la flexibilité des systèmes à objet réflexifs pour proposer un outil de méta-modélisation uniforme où tout nouveau modèle serait décrit comme une extension du méta-modèle (le formalisme de méta-modélisation, cf. §2.1.3). Le méta-modèle *M*eninge s'inspire de la notion de type de VDM (Vienna Development Method), des concepts issus des langages orientés objet (Eiffel, Smalltalk) et des systèmes réflexifs (Objvlisp, SHOOD). *M*eninge [Bounaas 97b] offre des représentations graphiques et textuelles qui peuvent être utilisées conjointement lors de la conception des modèles permettant ainsi de visualiser, d'utiliser ou d'étendre un modèle à partir de l'une ou l'autre de ces représentations.

Ce travail a été spécifié et réalisé en collaboration avec Philippe Morat (Maître de Conférences UJF), Fethi Bounaas (post-doctorant), Dalila Tamzalit [Tamzalit 96] dans le cadre de son DEA, Christine Martin [Martin 97] et Patrice Cailloce [Cailloce 97] dans le cadre de leur mémoire CNAM. Nous nous contenterons ici de donner les principes généraux

d'un environnement de méta-modélisation bâti sur un modèle à objet réflexif (cf. §2.4.1), de préciser davantage le niveau méta-modélisation de *M*eninge (cf. §2.4.2, 2.4.3, 2.4.4), puis de tirer quelques conclusions de cette approche (cf. §2.4.5).

2.4.1. Modélisation, méta-modélisation et réflexivité

On emploie le terme modèle pour désigner un formalisme. Par exemple le modèle E/A, le modèle hors-contexte, etc. Un acte de modélisation consiste à décrire un système, soit B dans un formalisme (un modèle), soit A. C'est ainsi que les aspects structurels d'un système d'information bancaire peuvent être modélisés par un diagramme E/A (Figure 2- 9), la syntaxe d'un langage de programmation peut être exprimée par une grammaire hors-contexte, etc. Le résultat d'un acte de modélisation de B dans le formalisme (le modèle) A est appelé le modèle A de B (ou modèle de B en A). Par exemple on parle du modèle E/A du système d'information bancaire, du modèle hors-contexte du langage de programmation (en fait on dit plutôt grammaire hors-contexte du langage).

On emploie le terme méta-modèle pour désigner un formalisme permettant de décrire des modèles. Un méta-modèle est donc un cas particulier de modèle utilisé pour modéliser des modèles. C'est par exemple le cas de *M*eninge. Un acte de modélisation (on parle de méta-modélisation) avec *M*eninge consiste donc à décrire un modèle A dans le formalisme de *M*eninge. Le résultat est le modèle *M*eninge du modèle A on dit aussi le méta-modèle de A en *M*eninge. Par exemple, la modélisation en *M*eninge du modèle E/A produit un méta-modèle de l'E/A en *M*eninge (cf. Figure 2- 9). De même, la modélisation en *M*eninge du modèle H-C produit un méta-modèle du H-C en *M*eninge.

*M*eninge étant lui même un modèle on peut le modéliser en *M*eninge. Le résultat d'un tel acte de méta-modélisation produit le modèle *M*eninge du modèle *M*eninge (ou méta-modèle de *M*eninge en *M*eninge) (cf. Figure 2- 9).

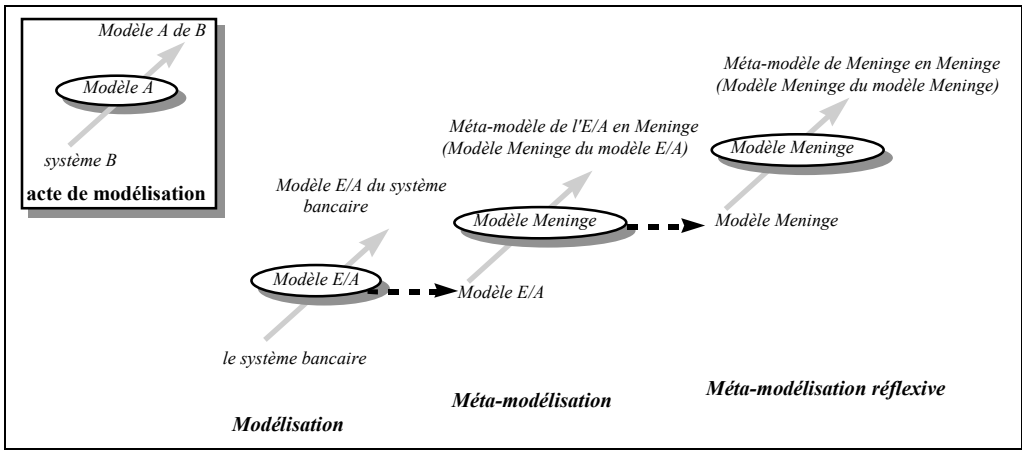


Figure 2- 9 : Niveaux de modélisation

2.4.2. Environnement d'ingénierie de modèles

L'environnement d'ingénierie de modèles proposé offre un méta-modèle doté de deux représentations, une graphique et une textuelle. Ces représentations peuvent être utilisées conjointement pour développer des modèles. La propriété de réflexivité a permis de générer le méta-modèle à partir d'un noyau. La Figure 2- 10 décrit un enchaînement d'utilisations de l'environnement commençant par la génération du méta-modèle à partir de l'amorce.

- Le méta-modèle de *M*eninge, résultat d'un acte de modélisation utilisant l'amorce de *M*eninge, produit une extension du noyau.
- Le méta-modèle de l'E/A, résultat d'un acte de modélisation utilisant le méta-modèle de *M*eninge, produit une extension du méta-modèle de *M*eninge.
- Le modèle E/A du système bancaire, résultat d'un acte de modélisation utilisant le méta-modèle de l'E/A, produit une extension du méta-modèle de l'E/A.
- Finalement, un diagramme d'instances du système bancaire, résultat d'un acte de modélisation utilisant le modèle E/A du système bancaire, produit une extension du modèle E/A du système bancaire.

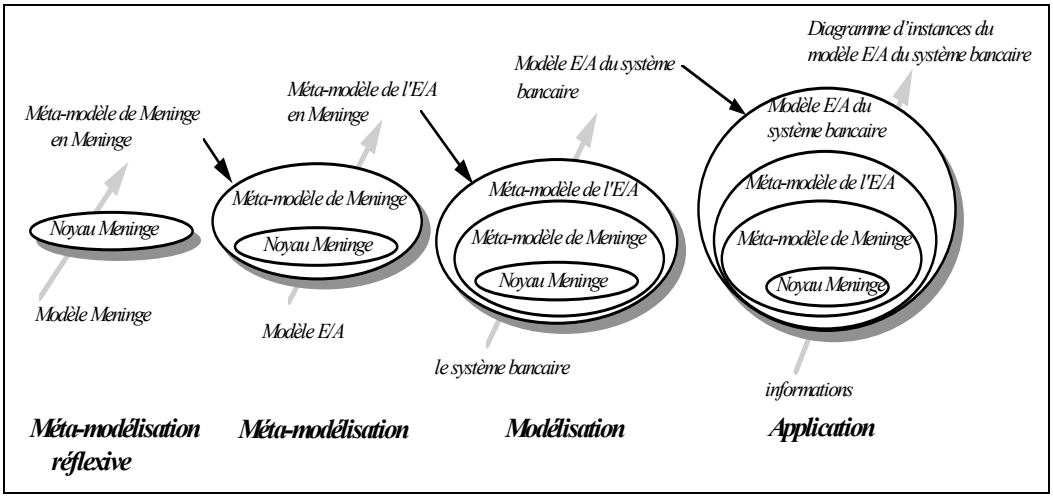


Figure 2- 10 : Environnement uniforme et extensible

*M*eninge offre donc trois niveaux de modélisation : un niveau méta-modélisation destiné à l'ingénierie des modèles des méthodes, un niveau modélisation destiné à celui des systèmes applicatifs et enfin un niveau application destiné à celui des bases d'informations des systèmes applicatifs.

La Figure 2- 11 illustre par exemple la méta-modélisation du modèle E/A, puis (niveau modélisation) l'utilisation du méta-modèle obtenu pour produire une spécification du modèle E/A d'un système bancaire. Cette spécification est exprimée dans la représentation graphique de *M*eninge. Finalement (niveau application) le modèle E/A du système bancaire permet de générer une base d'informations de l'application.

Le niveau méta-modélisation (partie gauche de la Figure 2- 11) est destiné à l'ingénierie des modèles. Celle-ci se déroule en deux étapes :

- l'étape de conceptualisation qui concerne l'identification des concepts du modèle. Nous pourrions ici parler d'"objets métiers", le métier étant celui du concepteur de modèles (cf. §2.4.3).
- l'étape de spécification dont l'objectif est d'implanter par spécification incrémentale les concepts déterminés préalablement (cf. §2.4.4).

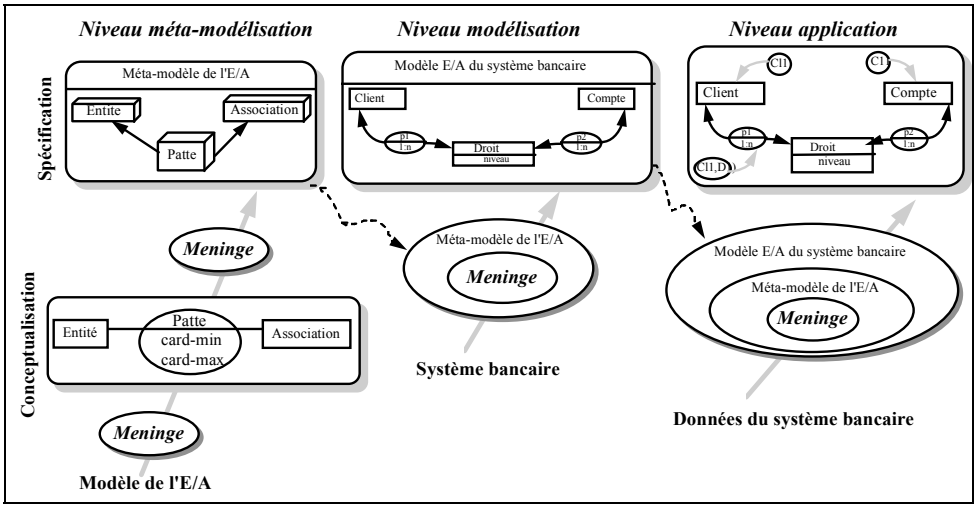


Figure 2- 11 : Trois niveaux de modélisation

2.4.3. Conceptualisation

La conceptualisation d'un modèle est essentiellement une étape de dialogue, de compréhension et de mise en accord dont l'objectif est d'identifier l'ensemble des concepts d'un modèle et leurs associations sans se soucier de leur implantation. Elle correspond à une étude d'expression des besoins produisant des méta-modèles similaires à ceux offerts dans la plupart des manuels des méthodes de conception où les différents modèles sont décrits par instanciation d'un méta-modèle graphique [Mega 95, Booch 97b, Oussalah 97, Booch 99]. La conceptualisation d'un modèle peut être réalisée en utilisant n'importe quel méta-modèle graphique. La Figure 2- 12 illustre par exemple les principaux concepts du modèle E/A (entité, association, patte, propriété, occurrence d'entité, occurrence d'association et occurrence de patte) en OMT (partie gauche) et en *M*eninge (partie droite). Nous préconisons cependant l'utilisation de *M*eninge afin d'automatiser en partie le passage vers l'étape de spécification.

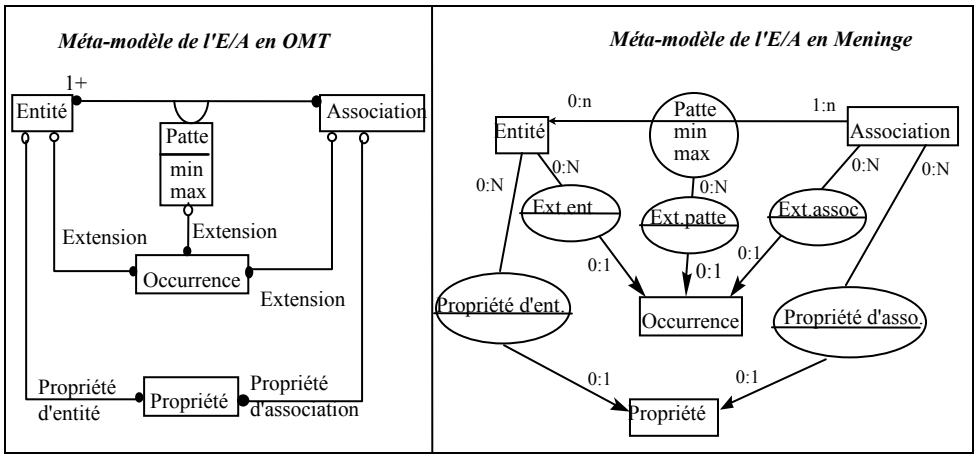


Figure 2- 12: : Etape de conceptualisation

2.4.4. Spécification

L'étape de spécification a ensuite pour objectif d'implanter par spécifications incrémentales les concepts déterminés préalablement. L'idée est ici de favoriser la réutilisation des concepts et donc des mécanismes existants.

Intuitivement les concepts d'Entité et d'Association constituent des réutilisations du concept de `Composé` (proche de la notion de classe) présent dans *M*eninge. De même le concept de Patte est une réutilisation de celui de `Lien` de *M*eninge, le concept de Propriété réutilise celui de `Paramètre`, etc.

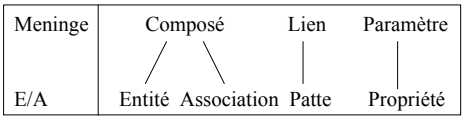


Figure 2- 13 : Réutilisation de concepts

La difficulté consiste à choisir les concepts les "plus proches " c'est à dire ceux favorisant au maximum la réutilisation des types et donc des mécanismes associés. La technique de réutilisation est ici tout à fait similaire à celle présentée au chapitre précédent (projet SHOOD) où nous avons vu que la réification des méthodes ou des attributs par des classes permet, par exemple, de réutiliser les mécanismes inhérents aux classes (spécialisation, instanciation...) pour les adapter à la surcharge des méthodes (ou à l'affinage des attributs).

Dans *M*eninge, les méta-classes sont utilisées pour définir de nouveaux concepts. La méta-classe `Méta-Entité` détient, par exemple, les informations structurelles et comportementales de toute entité : un nom, des propriétés, une clé définie par un sous-ensemble des propriétés,

etc. A chaque concept est également associé une classe détenant les propriétés communes aux occurrences (instances) de ce concept. La classe `Entité` comporte, par exemple, l'invariant d'unicité de la valeur de clé qui devra être vérifié lors de chaque création d'occurrences d'entités.

Chaque niveau de spécification, même partiel, est simulable. En effet le modèle de spécification correspond au modèle d'implantation : il s'agit de *M*eninge ou d'un modèle défini comme une de ses extensions. Tout nouveau concept " hérite " donc des propriétés et des mécanismes des concepts réutilisés. Ceux-ci peuvent à tout moment être valués ou exécutés même si les propriétés et donc les mécanismes spécifiques n'ont pas encore été spécifiés et implantés. Les mécanismes d'instanciation d'`Entité` et d'`Association` peuvent, par exemple, réutiliser celui du type `Composé` de *M*eninge. C'est ainsi que la classe `Entité` définie comme une sous-classe de la classe `Composé`, hérite en particulier des opérations lui permettant de générer des occurrences d'entités même si l'invariant d'unicité des clés n'a pas encore été exprimé. La Figure 2- 14 montre une vision simplifiée des liens entre les classes définissant le concept d'`Entité` et celles définissant celui de `Composé`.

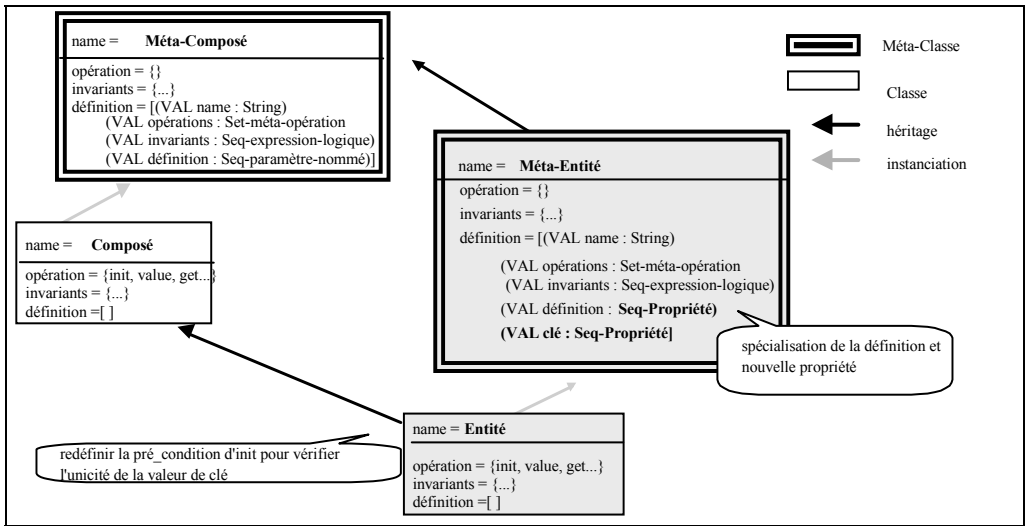


Figure 2- 14: Vision simplifiée de la définition d'un nouveau concept

2.4.5. Bilan : Vers les patrons d'ingénierie

Dans un environnement de méta-modélisation, l'utilisation d'un système objet réflexif a pour principal avantage de faciliter la cohabitation des différents modèles intervenant lors de la conception des systèmes : ces modèles s'expriment comme des extensions du même méta-modèle [Rieu 97b]. Cette uniformité de représentation offre d'autres avantages.

- Simulation des modèles en cours de spécification.

Quel que soit son niveau de spécification un modèle est défini comme une extension du méta-modèle bénéficiant ainsi de ses mécanismes opératoires (instanciation, héritage, etc.) qui sont ensuite partiellement ou totalement redéfinis. Il est donc possible à tout moment de simuler un modèle en cours de spécification permettant ainsi de tester les concepts en cours d'élaboration.

- Gestion des représentations multiples d'un même modèle.

Plusieurs représentations, par exemple graphique et textuelle dans le cas de *M*eninge, peuvent être utilisées conjointement lors de la conception des modèles permettant ainsi de visualiser, d'utiliser ou d'étendre un modèle à partir de l'une ou l'autre de ces représentations. Certains modèles admettent eux-aussi une représentation graphique ; c'est par exemple le cas des modèles offerts par les méthodes d'analyse et de conception des Systèmes d'Information (SI). D'autres privilégient les représentations textuelles ; c'est par exemple le cas des modèles offerts par les langages de programmation. Lors de la conception de ces modèles, il est possible de définir leurs représentations par redéfinition de celles du méta-modèle.

- Réutilisation partielle de modèles préalablement conçus.

Un nouveau modèle est souvent une extension ou une adaptation de modèles existants (c'est d'ailleurs le cas de *M*eninge). Il nous a donc semblé judicieux, lors de la spécification d'un modèle, de faciliter la réutilisation des concepts des modèles préalablement conçus. Cependant, la plupart des concepts d'un modèle, par exemple ceux d'`Entité` et d'`Association` du modèle E/A, ceux de `Composé` et de `Lien` de *M*eninge ne peuvent pas être définis par un seul type (classe) du méta-modèle. La réutilisation d'un concept A pour définir un nouveau concept B nécessite donc de lier par des relations d'héritage, de référence, d'instanciation, etc. l'ensemble des types de A aux différents types de B. La classe `Méta-Entité` est, par exemple, sous-classe de la classe `Méta-Composé`, la classe `Entité` est une sous-classe de la classe `Composé`, etc.

Très rapidement nous nous sommes aperçus que les différents concepts représentés en *M*eninge correspondaient à des configurations de types que nous appliquions systématiquement. De plus, les liens entre la configuration de types d'un concept et celle des concepts le réutilisant obéissaient eux aussi à des sortes de schémas types. D'où l'idée de définir des "patrons de types" utilisés comme un tout représentant une configuration de types détenant la sémantique d'un concept d'un modèle. Intuitivement dans un cadre de modélisation de modèles nous pouvons parler du Patron-entité pour modéliser l'ensemble de

types nécessaires pour modéliser le concept d'Entité du modèle E/A, du Patron-link pour signifier l'ensemble des types représentant le concept de lien de *M*eninge.

Dans *M*eninge, un patron [Rieu 97b] est modélisé par une configuration (ou composition) nommée de types liés par des relations en particulier d'instanciation et d'héritage. Définir un patron revient donc à décrire une composition réglementée de types. Pour chaque patron, il s'agit également de fournir des " règles " d'utilisation de ce patron. En particulier, dans un monde objet, ces règles doivent permettre d'instancier ce patron pour générer ses représentants, de le spécialiser pour élaborer des patrons plus spécifiques, de le composer avec d'autres patrons pour proposer des patrons d'une granularité encore plus importante. Les travaux entrepris dans *M*eninge ont en particulier permis la définition d'un ensemble de patrons modélisant la sémantique des principaux concepts de *M*eninge. Réutiliser un concept revient alors à spécialiser ou à instancier l'un de ces patrons permettant ainsi de disposer d'un grain de réutilisation supérieur à celui de type (classe) [Bounaas 97b].

Cette notion de patron en tant qu'ensemble de types nécessaires à la définition d'un concept a été systématisée et appliquée à notre propre modèle. La validation de cette approche a été faite au travers des modèles de Merise ainsi que d'un modèle de processus CAO [Harani 97].

2.5. Conclusion

Je suis aujourd'hui persuadée que la technologie objet reste une des voies les plus pertinentes pour prendre en compte les besoins des applications d'ingénierie (la CAO, l'ingénierie des méthodes, etc.). En particulier les approches "tout objet" que nous avons adoptées dans les projets SHOOD et *M*eninge ne sont pas uniquement satisfaisantes intellectuellement, elles constituent un précieux outil de représentation des connaissances :

- permettant une expression déclarative de la sémantique des systèmes et de leurs concepts,
- facilitant la mise en œuvre de mécanismes d'évolution,
- favorisant l'élaboration d'environnements adaptables à des besoins particuliers.

Tout système objet offre de fait la possibilité de réutiliser des classes. Les systèmes objet réflexifs, en particulier ceux dotés d'un niveau méta-classes, ont poussé au maximum cette faculté. La réutilisation est étendue à la plupart des classes (méta-classes) du système permettant ainsi de réutiliser et d'adapter les mécanismes internes du système[12].

[12] Malheureusement, de tels systèmes restent aujourd'hui du domaine de la recherche. Dans la plupart des langages de programmation orientés objet, les classes ne sont pas des objets. En particulier, elles ne sont pas instances de classes et ne sont pas modélisées. Smalltalk80 constitue une des rares exceptions. Cependant, dans Smalltalk le plan des méta-classes est isomorphe à celui des classes et n'est pas directement manipulable.

Cependant, il est clair qu'aujourd'hui un concepteur ne peut plus se permettre de spécifier et d'implanter un système en partant d'une granularité aussi fine. Il ne s'agit plus de réutiliser une classe ou un mécanisme de bas niveau tel que l'instanciation mais de favoriser la réutilisation d'architecture de granularité plus importante (ensemble de classes, de types, etc.) et détenant en propre leur mécanismes d'utilisation (l'instanciation, l'héritage). Il s'agit également de favoriser la réutilisation tout au long du cycle de développement des systèmes. L'expérience acquise dans *M*eninge nous a prouvé que la réutilisation peut se faire au niveau des "objets métiers", en l'occurrence dans notre cas au niveau des concepts des modèles.

La réutilisation à tous les stades du développement des systèmes est devenue depuis 1997 le principal thème de travail du pôle ingénierie des SI de l'équipe STORM. Elle fait l'objet du chapitre 3.

2.6. Articles sur les projets CADB, SHOOD et *M*eninge

Nous énumérons ci-dessous quelques articles concernant ces travaux.

- [Rieu 86b] - D. RIEU, G.T. NGUYEN
  Semantics of CAD Objects for Generalized Databases
  ACM/IEEE Design Automation Conference, Las Vegas, 1986
  Cet article présente les fonctionnalités d'un SGBD dans un environnement de CAO et décrit les principes de base du système CADB.

- [Nguyen 88c] - G.T. NGUYEN, D. RIEU
  Heuristic Consistency Control on Dynamic Database Objects.
  IFIP Conference " The role of Artificial Intelligence in DB and IS ", Guangzhou, 1988.
  Cet article décrit plus précisément le mécanisme de contrôle des évolutions d'objets en cours de conception. Ce mécanisme tire profit de techniques issues de l'Intelligence Artificielle.

- [Nguyen 89b] - G.T. NGUYEN, D. RIEU
  Schema evolution in object-oriented database system
  Revue : data Knowledge Engineering, North-Holland, vol. 4, 1989
  Cet article constitue un état de l'art des fonctionnalités offertes par les SGBD Orientés Objet en terme d'évolution de schéma et de propagation des mises à jour.

- [Rieu 91] – D. Rieu et al.
  Instanciation multiple et classification d'Objets
  Journées Bases de Données Avancées, Lyon, 1991
  Cet article propose de remettre en cause les principes de base de l'instanciation de manière à permettre à un objet d'appartenir à plusieurs classes, chacune d'elle modélisant un point de vue (une représentation) de l'objet. Un mécanisme de classification basé sur la multi-instanciation autorise l'objet à évoluer dans chacune de ses représentations.

- [Nguyen 92]- G.T. NGUYEN, D. RIEU
  SHOOD : a Design Object Model
  AID Conference, Pittsburgh, 1992
  Cet article décrit les concepts de base du système SHOOD.

- [Rieu 94] - D. RIEU, D. CONSTANT, G.T. NGUYEN, M. TOLLENAERE
  Conception intégrée des liaisons dynamiques
  Conf. Internationale sur la modélisation et la reconnaissance de caractéristiques dans les systèmes de CFAO avancés, IFIP, 1994
  Cet article décrit l'utilisation de SHOOD pour le génie mécanique. Il insiste plus particulièrement sur la description et l'évolution simultanée des représentations fonctionnelles et structurelles du produit en cours de conception.

- [Bounaas 97a] - F.Bounaas, M. Chabre, P-Y Cunin, J.P. Giraudin, P. Morat, D. Rieu
  Objets et méta-modélisation
  Chapitre 5 de l'ouvrage collectif : Ingénierie objet : concepts et techniques, Editeur C. Oussalah, InterEditions, avril, 1997.
  Ce chapitre de livre est un état de l'art et une illustration des différentes acceptions du terme méta-modélisation dans les systèmes objet.

- [Bounaas 97b] - F. BOUNAAS, D. RIEU
  Réutilisation pour l'ingénierie des modèles
  First worshop on the Many Facets of Process Engineering, Gammarth,Tunisie, Septembre, 1997.
  Cet article montre l'intérêt des systèmes objet réflexifs pour l'ingénierie des modèles et propose des patrons d'ingénierie permettant la réutilisation d'ensembles réglementés de types.

Chapitre 3 : Réutilisation dans l'ingénierie des SI

3.1. Introduction

Depuis les années 1960, l'ingénierie des systèmes d'information est régie par des méthodes offrant une succession de modèles afin de favoriser un continuum de la définition des besoins des clients jusqu'au système développé et exploité. Les approches ont constamment évolué, depuis les premières méthodes dites d'analyse (années 1960) jusqu'aux méthodes systémiques (années 1980) en passant par les méthodes cartésiennes (années 1970). De nos jours, les méthodes à objet ont pour enjeu de participer à l'industrialisation du développement des logiciels. Comme dans la plupart des domaines d'ingénierie, le passage de l'artisanat à l'industrie vise à réduire les coûts et les délais des développements tout en améliorant la qualité des produits.

Ces objectifs ne peuvent être atteints qu'au prix de deux principales évolutions qui constituent des facteurs clés dans les développements des systèmes et particulièrement dans celui des systèmes à objet :

- la maîtrise voire l'automatisation des activités de développement,
- une gestion efficace de composants réutilisables.

On a tout d'abord cru (ou fait semblant de croire) que l'approche objet offrait de fait ces potentialités.

3.1.1. Maîtrise des développements

L'utilisation du paradigme objet tout au long du cycle d'ingénierie semblait offrir de fait des développements " sans couture " favorisant le raffinement des spécifications. Il ne s'agissait plus par exemple de transformer un diagramme Entité/Association en un schéma relationnel mais bien de raffiner un diagramme de classes modélisant des objets métiers de l'utilisateur en un diagramme de classes représentant les objets techniques des systèmes logiciels. Cette approche est apparue clairement dans OMT [Rumbaugh 91] et OOSE

[Jacobson 92] et a été complètement adoptée dans les méthodes à objet dites de deuxième génération telles que Syntropy [Cook 94], Catalysis [D'Souza 96, D'Souza 98] et bien sûr UML au travers de la démarche Objectory [Rational 97].

Cependant, force est aujourd'hui de reconnaître que la représentation des systèmes dans des formalismes communs et adaptables aux différentes étapes du développement facilite les développements sans couture mais ne les garantit pas [Barbier 99]. On est loin des illusions et de l'engouement provoqués par les premiers articles sur les développements orientés objet. Citons, entre autre, J. Rumbaugh [Rumbaugh 91] qui en 1991 affirmait : " Because an object-oriented approach defines a set of problem-oriented objects early in the project and continues to use and extend these objets throughout the development cycle, the separation of life cycle phases is much less distinct. (...) The process is seamless because there are no discontinuities in wich a notation at one phase is replaced by a different notation at another phase ".

Quelques années plus tard, C. Cook et J. Daniels proposent la méthode Syntropy [Cook 94] et expriment leurs doutes : "We should emphasise that the formal relationship between specification and implementation models is far from staightforward", et pourtant leur méthode offre une démarche de développement autrement plus rigoureuse que ne l'était celle d'OMT.

Nous proposons ci-dessous quelques exemples de transformations ou de raffinements de modèles, destinés à illustrer cette difficulté et à montrer que l'adoption de formalismes communs entre les différentes étapes du cycle de développement ne permet pas de faire l'économie de l'expression des règles de cohérence et de transformation entre les différents niveaux de représentation des systèmes.

Les cas d'utilisation et les diagrammes de séquences sont utilisés en UML dès l'expression des besoins pour représenter les principales activités du système ainsi que les flux d'informations entre les acteurs et le système cible (cf Figure 3- 1). Ces diagrammes peuvent être graduellement raffinés pour exprimer des demandes de services entre objets. C'est ainsi que les flux d'informations de la Figure 3- 1 (numéro compte et montant) sont représentés dans la Figure 3- 2 par des paramètres (numc et m) des demandes de services entre objets. De même, les activités du système ont été décomposées de manière à mettre en évidence les demandes de services entre objets.

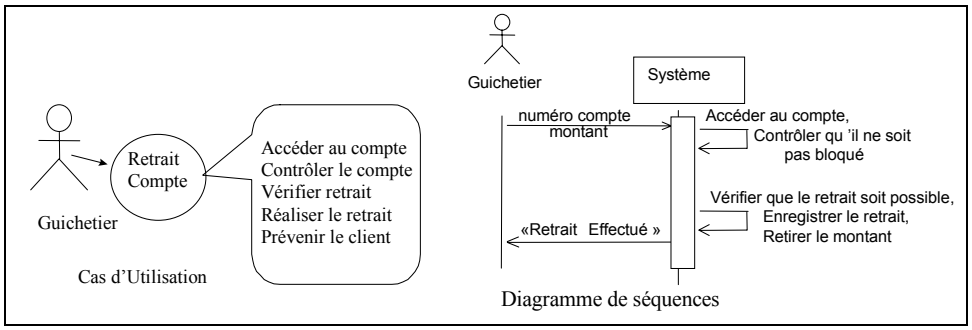


Figure 3- 1 : Diagrammes d'expression des besoins

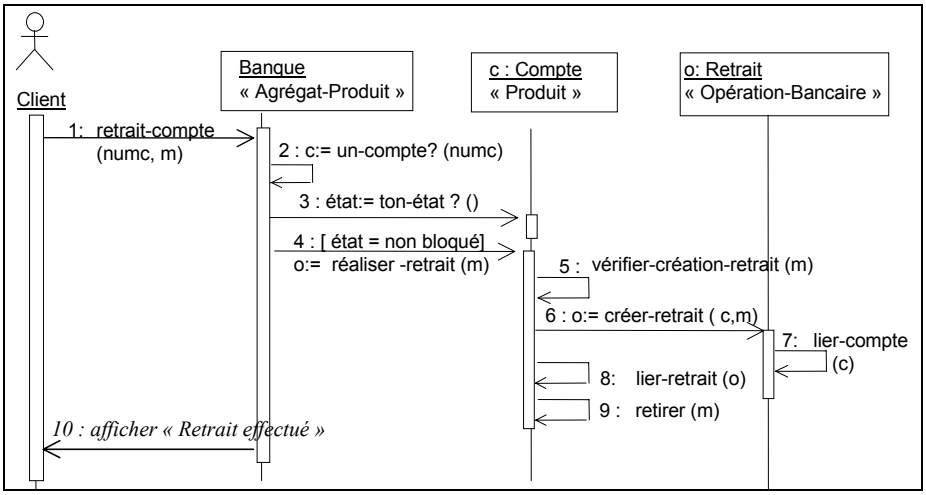


Figure 3- 2 : Diagramme de séquences « retrait compte »

Des transformations sont également utilisées pour raffiner les diagrammes de classes. La Figure 3- 3 issue d'un article de P. Desfray [Desfray 98] et reprise par F. Barbier [Barbier 99] montre, par exemple, la transformation d'une association exprimée dans un modèle d'analyse, en un attribut de type collection dans le modèle de conception correspondant.

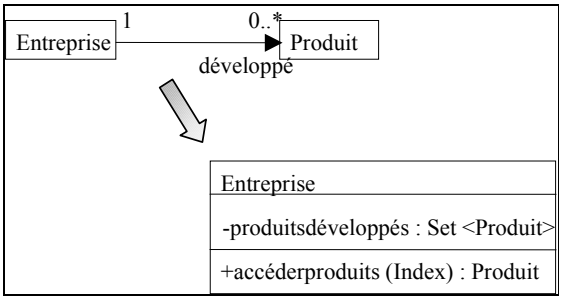


Figure 3- 3: Transformation de l'objet Entreprise

Les deux cas précédents illustrent des raffinements de modèles. Il s'agit également de maintenir la cohérence entre des modèles de même niveau d'abstraction. La Figure 3- 4 montre un diagramme de classes conforme au diagramme de séquences de la Figure 3- 2. Les demandes de services entre objets sont transformées en méthodes publiques (par exemple la méthode ton-état ?() réalise la demande de service de même label dans le diagramme de séquences), les accès entre objets (par exemple une banque à ses comptes) sont matérialisés par des associations entre objets et des méthodes privées de consultation (par exemple la méthode un-compte ?()), etc.

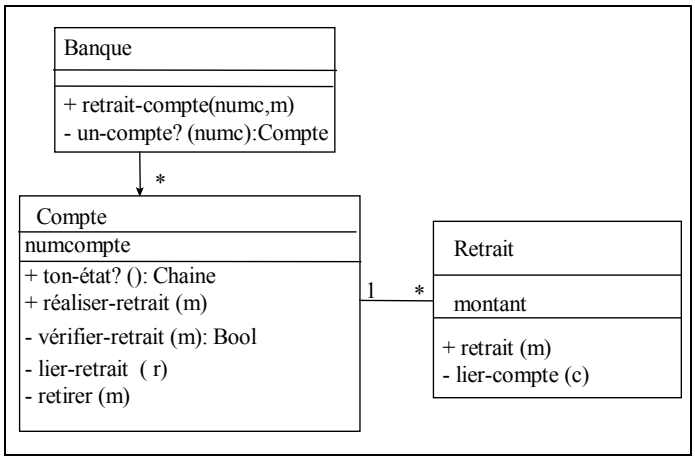


Figure 3- 4 : Diagrammes de classes pour la fonction « retrait-compte »

Ces trois exemples montrent bien que l'approche objet n'a en rien simplifié les processus de développement. En particulier, les transformations inter-modèles ne sont pas plus simples ni plus naturelles qu'elles ne l'étaient dans des méthodes de développement traditionnelles telles que Merise. Pour assurer une évolution continue des objets et donc garantir une traçabilité des activités de développement, différentes voies de recherche sont actuellement explorées. Une solution consiste à représenter explicitement les évolutions des objets par des règles, des modèles de processus, voire des méta-modèles [Grosz 96, Rolland 98].

Cette approche se heurte à la difficulté d'expliciter et de raffiner les démarches de développement proposées aujourd'hui dans la littérature. Remarquons, par exemple, que les ouvrages dédiés aux méthodes à objet de seconde génération proposent rarement une formalisation de leurs démarches de développement.

3.1.2. Réutilisation de composants

Toute véritable industrialisation se caractérise par la conception et le développement d'un ensemble de produits à partir d'éléments réutilisables. Ceci est désormais vrai dans plusieurs

domaines industriels, par exemple la construction automobile ou l'électronique. Dans le secteur du logiciel, il existe depuis fort longtemps des bibliothèques de programmes de calculs scientifiques, statistiques, etc. Dans le domaine de l'orienté objet, les informaticiens des banques ont été les premiers à appliquer ces principes de réutilisation, en raison de la centralisation et de l'homogénéité des systèmes d'information bancaires. Ainsi ont-ils été des pionniers dans l'usage professionnel d'un langage comme SmallTalk. Ce besoin de réutilisation est l'une des bases du succès de l'approche objet des dernières années. Les langages à objet SmallTalk, Eiffel et C++ ont les premiers répondus à ce besoin en mettant à la disposition du développeur des bibliothèques de composants logiciels. Ces bibliothèques constituent aujourd'hui de véritables boîtes à outils couramment utilisées et intégrées aux outils de développement. C'est par exemple le cas des systèmes de composants logiciels tels que COM+ et EJB (Enterprise Java Beans).

Cependant, le plus souvent, de telles bibliothèques n'apparaissent qu'au niveau du codage en vue d'améliorer la productivité des développeurs. De nombreux progrès restent à faire pour intégrer la réutilisation dans tout le processus de développement des applications. Les tendances actuelles consistent à favoriser la réutilisation de résultats d'expression des besoins, d'analyse et de conception. L'objectif commun est de permettre aux concepteurs de travailler à un niveau d'abstraction plus élevé que celui des classes logicielles (Figure 3- 5).

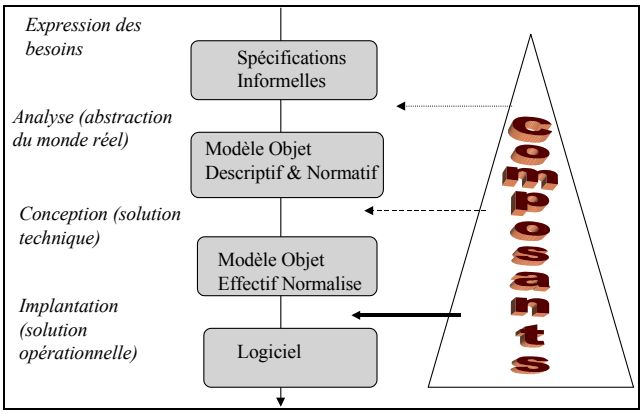


Figure 3- 5 : Réutilisation de composants dans les étapes amonts du développement [Nerson92]

De nombreux travaux en génie logiciel, en système d'information et en intelligence artificielle vont dans ce sens sans qu'une réelle harmonisation soit actuellement envisageable. Une grande variété de composants réutilisables a été proposée : les objets métiers [Andro 98], les modèles de domaines génériques [Maiden 92, Maiden 93], les patrons de conception (design patterns) [Gamma 95], les patrons d'analyse (analysis patterns) [Coad 96], les

ontologies [Gruber 93], les frameworks [Johnson 92, Revault 96], etc. Dans tous les cas, un composant est vu comme une solution testée et acceptée pour résoudre un problème fréquemment rencontré lors du développement de logiciels.

Si les propositions concernant les types de composants sont nombreuses, les techniques de développement à base de composants restent encore embryonnaires. Il s'agit de remplacer les démarches traditionnelles de développement par deux types de processus complémentaires.

- Des processus de développement POUR la réutilisation dont l'objectif est d'identifier, de spécifier et de développer des composants réutilisables.
- Des processus de développement PAR la réutilisation dont l'objectif est d'utiliser les composants réutilisables pour spécifier et implanter un système d'information. Il s'agit ici d'utiliser tout type de composants : les objets métiers, les patrons, les frameworks, etc.

Deux études bibliographiques récentes nous ont permis de faire le point sur les techniques de réutilisation dans l'ingénierie des SI. La première [Front 99b] est centrée sur le concept de patron dans les développement orientés objet, la seconde [Cauvet 00], plus générale, décrit plusieurs modèles de composants ainsi que quelques techniques utilisées pour mettre en œuvre les développements par et pour la réutilisation.

Nous résumons ces études par une comparaison de quatre modèles de composants réutilisables (§ 3.1.2.1) et des techniques d'identification des composants (§ 3.1.2.2). La dernière section (§ 3.1.2.3) se focalise sur les outils de développement à base de patrons. Un chapitre d'un livre collectif [Cauvet 00] développe les environnements centrés sur d'autres types de composants.

3.1.2.1. Différents types de composants : critères de comparaison

Plusieurs critères permettent de comparer les très nombreux modèles de composants proposés aujourd'hui dans la littérature. Nous en illustrons certains en les appliquant à quatre types de composants : les patrons, les frameworks (ou cadriciels [Booch 97a], canevas [Revault 96]), les ontologies et les modèles de domaines génériques. Nous insisterons plus particulièrement sur les frameworks et les patrons qui constituent aujourd'hui des types de composants particulièrement étudiés, référencés et utilisés.

- *Leur nature.*

  Les composants sont de nature conceptuelle ou logicielle.

Parmi les quatre types de composants comparés, seuls les frameworks[13] sont des logiciels exécutables. Les patrons représentent des connaissances et des expériences concernant des problèmes logiciels et leurs solutions [Appleton 97]. Ces solutions sont sous la forme de spécifications semi-formelles exprimées dans un ou plusieurs modèles offerts par les méthodes à objet (OMT, UML, OOSE, etc.). Dans le domaine de l'intelligence artificielle, une ontologie peut être vue comme un référentiel formalisé des connaissances partageables (ou réutilisables) entre plusieurs systèmes à base de connaissances. Une ontologie associe des définitions en langue pseudo naturelle des éléments de l'univers du discours et des axiomes formels qui contraignent l'interprétation et l'utilisation de ces termes. C'est ainsi que T.R. Gruber [Gruber 93] utilise le format KIF (Knowledge Interchange Format) [Genesereth 92][14] pour exprimer une ontologie dédiée au partage de références bibliographiques. La représentation des modèles de domaines génériques est graphique, elle est basée sur un méta-modèle intégrant les notions de transition d'état et de structure d'objet.

- *Le degré de généralité des composants*

Les composants peuvent être mono-domaines, s'ils répondent à des problèmes spécifiques d'un domaine d'applications. Ils sont multi-domaines si le problème traité est fréquent dans de nombreux domaines d'applications.

Certains frameworks, dits verticaux, sont spécifiques à un domaine d'application comme par exemple la comptabilité [Keefer 94]. D'autres, dits horizontaux, sont plus généraux, c'est par exemple le cas des frameworks permettant la réalisation d'éditeurs graphiques (HotDraw [Johnson 92], Editalk [Voyer 92]). On retrouve cette différence au niveau des patrons. Certains patrons d'analyse sont dédiés à un domaine particulier (les systèmes d'information bancaires, les réseaux électriques, etc.) alors que d'autres adressent des problèmes plus généraux. Par exemple, il est possible d'utiliser un patron général de gestion de ressources pour représenter aussi bien un problème de ressources humaines qui comprend l'affectation des personnes à des activités, qu'un problème de gestion de ressources matérielles réalisant l'affectation des machines dans un contexte de production [Bounaas 97a].

Les modèles de domaines génériques sont multi-domaines. N. Maiden [Maiden 92, Maiden 93] proposent 32 modèles génériques considérés comme « récurrents » dans de nombreux domaines d'applications. Par exemple le modèle générique « conteneur de ressources »

13" A framework is a set of classes that embodies an abstract design for solutions to a family of related problems ". R. Johnson
" Un framework est une architecture réutilisable qui fournit la structure générique et le comportement d'une famille d'abstractions logicielles, dans un contexte qui spécifie leurs collaborations et leur utilisation à l'intérieur d'un domaine donné." B. Appleton

14 KIF est une notation préfixée pour le calcul de prédicats indépendante d'un langage de programmation ou d'un langage de données particulier. Dans KIF, une ontologie définit un ensemble de classes, de relations, de fonctions et de constantes du domaine du discours et inclut une axiomatisation contraignant l'interprétation.

correspond aussi bien à une bibliothèque (un conteneur de livres) qu'à un entrepôt (un conteneur d'articles). En ce sens, les modèles de domaines génériques sont proches des patrons généraux d'analyse. Une ontologie correspondant à un référentiel de domaine est bien entendue mono-domaine.

- *La granularité des composants*

La granularité des composants est souvent mesurée en nombre de classes. Elle peut être fixe ou variable.

Les patrons de conception d'E. Gamma ou ceux d'analyse de P. Coad proposent des diagrammes constitués de deux à six classes. La plupart des frameworks proposent au contraire des architectures logicielles complètes. Prenons par exemple le framework MVC de Smalltalk 80 dédié au développement d'interfaces utilisateur. Le noyau de MVC composé des trois classes abstraites `Model`, `View` et `Controller` réalise les trois principaux concepts du framework. Le framework complet est constitué d'une très grande variété de classes facilitant la construction d'interfaces plus ou moins spécifiques. Les patrons sont des architectures plus petites que les frameworks [Gamma 95], un framework pouvant en effet contenir plusieurs patrons. Les patrons sont alors des micro-architectures et les frameworks des macro-architectures [Booch 97a]. Les modèles de domaines génériques ont une granularité similaire à celle des patrons d'analyse. Par contre, la granularité des ontologies est très variable. Elle dépend de la complexité du domaine représenté (nombre d'objets métiers, de relations, etc.).

- *La portée des composants*

La portée d'un composant est évaluée en fonction de l'étape d'ingénierie à laquelle le composant s'adresse.

L'utilisation d'un framework se fait lors des étapes d'analyse (dans le cas de frameworks verticaux) ou lors de la conception architecturale (dans le cas des frameworks horizontaux) des systèmes. On parle également de patrons d'analyse [Coad 92, Coad 96, Folwer 97b], de patrons de conception [Gamma 95] et de patrons d'implantation [Coplien 92]. Les patrons et les frameworks ont donc des portées différentes et sont intuitivement « typés » en fonction de leur portée. L'utilisation des ontologies et des modèles de domaines se fait lors de l'étape d'analyse.

- *Les formes de liens entre composants*

Quelle que soit la granularité des composants réutilisables il est extrêmement rare de ne pas avoir à en sélectionner puis à en utiliser plusieurs lors du développement des applications. Peu de techniques existent aujourd'hui pour d'une part organiser les bibliothèques de composants afin d'en faciliter la sélection et d'autre part assembler les composants pour obtenir des architectures plus importantes.

Les patrons de conception d'E. Gamma sont essentiellement liés par des liens d'utilisation mettant en évidence, pour chaque patron, les patrons utilisés avec ou par celui-ci. Les modèles de domaines de N. Maiden sont d'une part classifiés en fonction de leur degré de généralité, d'autre part structurés hiérarchiquement et agrégés. Les ontologies pouvant être assimilées à des référentiels pour des domaines d'applications particuliers, nous pouvons supposer qu'elles peuvent être classifiées en fonction du domaine auquel elles sont destinées.

Enfin, nous connaissons aujourd'hui encore peu de techniques pour assembler les frameworks. Notons cependant que quelques outils d'aide à l'adaptation de frameworks offrent des mécanismes de composition. La composition s'exprime généralement par un langage de composition visuelle [De Mey 94] ou par un langage de script [Staldemann 89].

3.1.2.2. Ingénierie des composants réutilisables

Il n'existe pas aujourd'hui de méthodes spécifiquement dédiées à l'identification, la spécification, l'implantation et l'organisation de composants. En revanche, nous disposons d'un ensemble de techniques allant dans ce sens. F. Semmak [Semmak 98] a identifié deux grandes approches d'identification de composants que nous avons reprises et illustrées dans un chapitre d'ouvrage collectif [Cauvet 00] (cf. Figure 3- 6). Nous résumons ces deux approches ci-dessous.

Identification de composants réutilisables			
Rétro-ingénierie		Analyse de domaine	
Analyse de similarité	Recherche par analogie	Démarche statique	Démarche dynamique
[Castano 94] [Spanoudakis 96]	[Falkerhainer 89] [Maiden 92]	[Kang 90]	[O 'Connor 94]

Figure 3- 6 : Quelques techniques d'identification de composants réutilisables.

- L'approche par **rétro-ingénierie** vise à abstraire des éléments réutilisables à partir de l'analyse de produits existants. Cette approche, essentiellement ascendante, permet d'évaluer les similarités et les différences entre produits. Deux techniques de base sont principalement utilisées : l'*analyse de similarités* [Castano 94, Spanoudakis 96] et la *recherche par analogie* [Falkerhainer 89, Gentner 83, Holyoak 89].

  Si l'objectif de ces deux techniques est sensiblement le même, le degré de généralité des composants identifiés est plus grand dans les techniques de recherche par analogie. En effet la recherche par analogie, contrairement aux techniques d'analyse de similarités, évalue les ressemblances sémantiques entre éléments (par exemple la nature des liens entre objets, leur organisation au sein des système, etc.) plutôt que leurs similitudes syntaxiques qui rendent nécessaire une terminologie commune et donc le plus souvent mono-domaine.

- L'approche par **analyse de domaine** a également pour objectif d'identifier les objets, les opérations, les associations communs à une classe de systèmes similaires [Neighbors 89, Kang 90] dans le but d'élaborer un **modèle de domaine**. Deux types de démarches, qualifiées de *statique* et de *dynamique* [Cauvet 00], sont actuellement proposées [Wartik 92, Semmak 98].

  Les démarches statiques partent d'une famille de systèmes de fonctionnalités similaires (gestion des comptes bancaires, gestion de stock, systèmes de réservation, etc.) dans l'objectif de produire un modèle de domaine exprimant les ressemblances et les différences entre les systèmes. C'est par exemple l'approche préconisée dans la méthode FODA (Feature Domain Abalysis) développée au SEI [Kang 90]. Un modèle de domaine obtenu à l'issue de l'application de la démarche FODA a pour particularité de proposer un sous-modèle dit de « features » mettant en évidence sous la forme de hiérarchie les caractéristiques qui discriminent les systèmes de même domaine.

  Les *démarches dynamiques* considèrent un domaine comme un ensemble de problèmes pour lesquels différentes solutions sont envisageables. Les modèles de domaine obtenus sont « orientés problèmes » et non plus solutions [Grosz 96]. Par exemple le domaine des réservations de places d'avions existe à travers des besoins de satisfaction des attentes des clients, d'optimisation des réservations, etc.

  Contrairement à l'approche statique, les méthodes d'analyse de domaine dynamiques rompent avec l'ingénierie traditionnelle des systèmes en mettant en évidence la définition des problèmes. C'est par exemple le cas de la méthode Synthesis [O'Connor 94] où les modèles de domaine sont spécifiés par des processus de décision qui guident les concepteurs d'application dans l'expression de leurs besoins et dans la construction d'applications les satisfaisant.

3.1.2.3. Ingénierie des systèmes par réutilisation

Des travaux de plus en plus nombreux centrent leurs efforts sur la recherche de méthodes adaptées au développement de logiciels par réutilisation. C'est le cas du consortium européen REBOOT [Karlsson 95, Morel 96] et des travaux autours de la méthode Catalysis [D'Souza 98]. Il privilégient la mise en œuvre de nouveaux environnements de développement fournissant des fonctionnalités concernant :

- la gestion de catalogues de composants,
- la spécification des systèmes,
- l'implantation par génération de code,

- la validation de spécifications existantes,
- la maintenance des systèmes.

Dans le cadre de notre équipe nous nous sommes particulièrement intéressés aux environnements basés sur le concept de patrons. Il s'agit pour la plupart de prototypes de recherche même si des ateliers commercialisés tels que Objecteering de la société Softeam ou Blueprint Technologies commencent également à intégrer l'usage des patrons de conception en particulier ceux d'E. Gamma [Gamma 95].

Les fonctionnalités attendues d'un environnement de développement par réutilisation centré patrons commencent à être bien définies. Nous les développons ci-dessous et nous citons quelques outils les prenant partiellement ou totalement en compte.

- *Gestion des catalogues de patrons*

Des relations inter-patrons devraient permettre d'organiser les catalogues de patrons existants dans le but d'en faciliter leur sélection. Dans [Front 99b] nous avons mis en évidence un certain nombre de ces relations : l'alternative met en correspondance des patrons d'intentions identiques, le raffinement correspond à une spécialisation des intentions… Ces relations seront présentées dans la section 3.3.

A notre connaissance peu de travaux se sont intéressés à ce problème d'autant plus difficile qu'il s'agit d'organiser des patrons en fonction de l'intention des patrons qui est la plupart du temps décrite de manière informelle. Citons toutefois l'outil de F. Budinsky [Budinsky 96] qui propose une organisation des patrons de conception d'E. Gamma sous la forme de pages HTML. Le concepteur peut naviguer pour trouver le patron correspondant à son problème.

- *Analyse et conception à partir de patrons*

Une utilisation efficace de catalogues de patrons lors de la spécification des systèmes nécessite de disposer d'opérateurs de *sélection* de patrons en fonction des problèmes à résoudre, mais aussi d'*instanciation* des solutions génériques offertes par les patrons et finalement de *composition* d'instances de patrons pour générer une spécification complète du système [Rieu 99].

La plupart des outils proposés se sont surtout intéressés à l'instanciation des solutions génériques offertes par les patrons. Pour parler d'instanciation et non pas simplement d'imitation (d'application, d'utilisation, etc.), l'outil doit offrir une représentation explicite généralement sous la forme de classes ou de méta-classes des solutions offertes par les

patrons. Dans [Borne 99] trois niveaux liés à l'instanciation des patrons sont clairement identifiés :

- la méta-représentation des informations d'un patron définit un méta-modèle de patrons,
- la représentation abstraite d'un patron définit un modèle général, générique ou prototypique instanciant un méta-modèle,
- la représentation concrète d'un patron correspond à une utilisation d'un patron. Elle définit un modèle concret par instanciation du méta-modèle et spécialisation du modèle général (paramétrage du modèle générique ou clonage du modèle prototypique).

Une forte analogie peut être faite avec les notions de méta-classe, classe abstraite et classe concrète (cf. par exemple les systèmes Shood et *M*eninge au chapitre 2) où :

- la classe concrète est une sous-classe de la classe abstraite
- les classes abstraite et concrète sont des instances de la méta-classe.

Quelques outils offrent ces trois niveaux de modélisation indispensables à une réelle instanciation [Pagel 96, Meijler 97]. La Figure 3- 7 illustre ces trois niveaux présents dans l'outil FACE [Meijler 97].

Le méta-modèle (constitué de méta-classes) et le modèle général constituent une réification de la solution offerte par le patron Etat de Gamma [Gamma 95]. Un lecteur non familiarisé avec les « design patterns » d'E. Gamma peut préalablement se reporter à la section 3.3.1.3 qui reprend en partie le patron Etat. Le modèle concret correspond à l'exemple du protocole TCP [Gamma 95]. Pour ne pas alourdir la figure, seuls les liens d'instanciation ont été représentés. Des liens d'héritage unissent également les classes du modèle général et celles du modèle concret. C'est ainsi que la classe EtatTCP hérite de la classe PrimalAbstrState, le classe ConnexionTCP hérite de la classe PrimalContexte, etc.

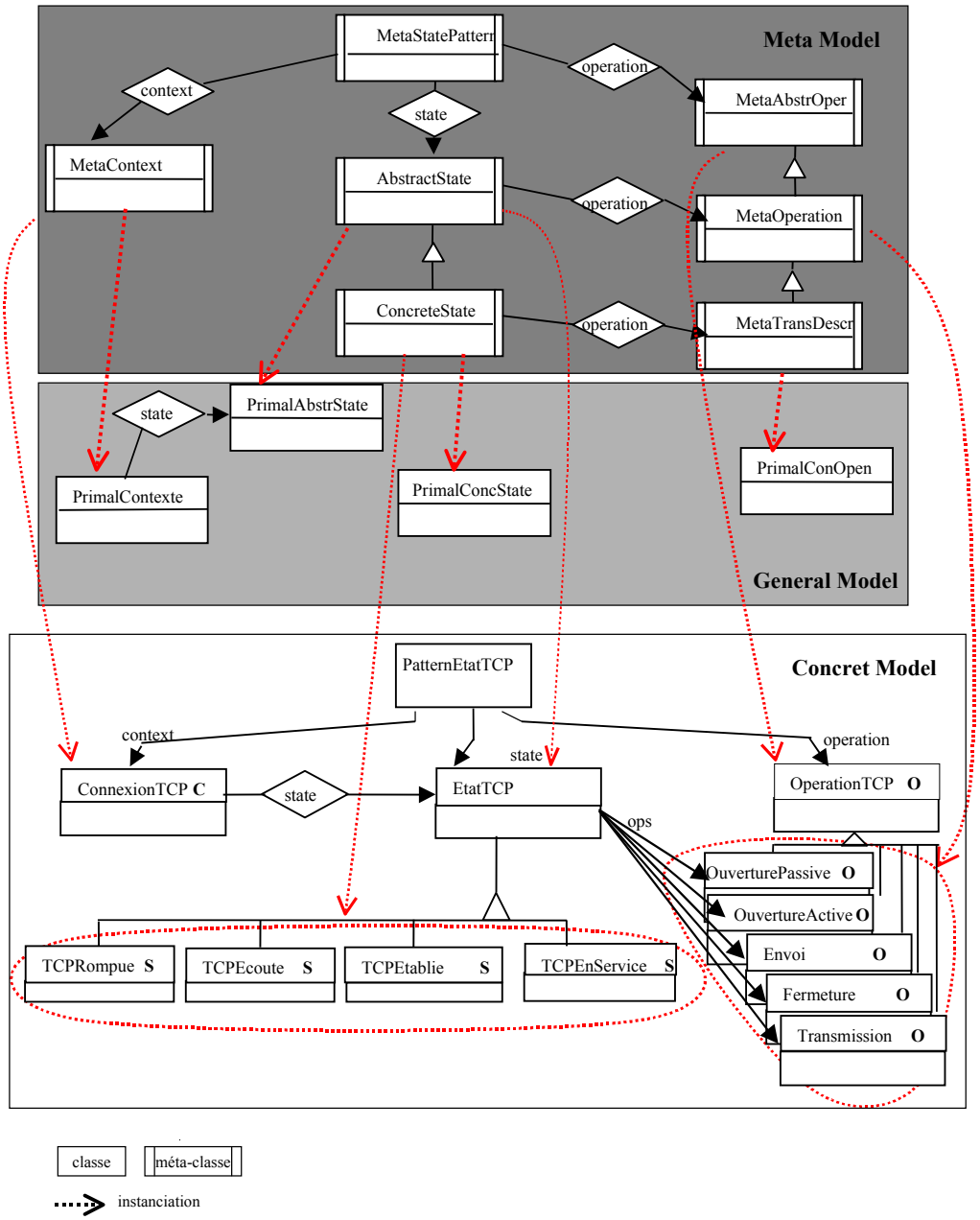


Figure 3- 7 : Trois niveaux de représentation

- *la génération de code.*

Les patrons offrent des solutions génériques et conceptuelles dont l'instanciation (l'adaptation) fournit une spécification semi-formelle des applications. Les patrons peuvent également être utilisés dans les étapes d'implantation. Plusieurs approches de génération de code à partir de patrons ont été proposées [Borne 99, Sunyé 99]. Il peut s'agir d'étendre un langage de programmation par de nouvelles primitives permettant de décrire et d'instancier

des patrons, c'est par exemple le cas de l'outil LayOM (Layered Object Model) [Bosch 96]. Une autre approche, qualifiée de « moulinette » dans [Borne 99] consiste à assimiler un patron à un composant générique à qui le concepteur fournit des valeurs [Soukup 95].

La plupart du temps les outils proposent une implantation unique d'un patron ce qui semble contradictoire avec le concept même de patron sensé offrir une solution conceptuelle et donc implantable de différentes manières. Citons toutefois les outils développés par G. Sunyé [Sunyé 99] et F. Budinsky [Budinsky 96] qui prennent en compte les différentes implantations possibles d'un patron.

- *la validation des spécifications*

Il s'agit à partir de modèles ou d'implantations existants, d'identifier des schémas correspondant aux solutions génériques offertes par les patrons. L'objectif est de proposer des restructurations des modèles afin d'améliorer les spécifications des systèmes. Quelques outils vont dans ce sens [Eden 97, Krämer 96]. Le système Pat [Krämer 96] recherche par exemple des instances de patrons dans des fichiers C++.

- *La traçabilité des développements*

L'objectif est ici de conserver les traces d'utilisation des patrons tant au niveau des spécifications que du code de l'application de manière à faciliter la maintenance des systèmes.

Au niveau des spécifications, la réification des patrons permet de mettre en œuvre un « réel » mécanisme d'instanciation et a également pour avantage de conserver les traces d'utilisation des patrons dans les modèles d'applications.

Au niveau de la génération de code, les outils proposant d'étendre les langages permettent de conserver le lien entre l'instance d'un patron et son implantation [Borne 99]. Au contraire, les outils de type « moulinette » doivent se doter de mécanismes supplémentaires d'étiquetage du code pour maintenir ce lien.

Les travaux destinés à garantir une meilleure maîtrise des processus de développement et ceux destinés à favoriser la réutilisation de composants sont indéniablement complémentaires. C'est ainsi que des techniques de modélisation de processus peuvent être utilisées pour décrire les démarches de développement par ou pour la réutilisation. Inversement, des processus de développement peuvent être décrits sous la forme de patrons de processus (process patterns). C'est par exemple le cas de la démarche préconisée dans Catalysis [D'Souza 98] et dans SCalP [Front 97]. L'utilisation des techniques à base de patrons pour décrire des processus ou des fragments de processus nous semble une technique de formalisation des démarches de développement particulièrement prometteuse. Elle permet une formalisation des processus en fonction des buts à atteindre (exprimés dans l'intention des patrons) et nécessite l'expression

de liens inter-patrons exprimant par exemple un ordre d'application entre eux. C'est l'approche prise dans SClaP où le lien d'utilisation (cf. §3.3.2.2) permet une décomposition du problème de modélisation à résoudre.

3.1.3. Activités de recherche de l'équipe

Depuis 1997, mes activités de recherche se sont focalisées sur les approches à base de composants et plus particulièrement sur les patrons d'ingénierie. Dans le cadre du pôle ingénierie des SI de l'équipe STORM, différents travaux de recherche ont été entrepris en ce sens. Ils sont basés sur la conviction que ***l'usage systématique de composants tout au long du cycle de développement permet de combiner réutilisation de composants et maîtrise (et donc traçabilité) des procédés de développement.*** Deux types de travaux sont actuellement en cours dans notre équipe. Les premiers adressent les aspects dynamiques des systèmes, les autres, plus généraux, concernent les patrons d'ingénierie.

3.1.3.1. Réutilisation des aspects dynamiques : SCalP et NCR

La première catégorie s'attache à développer des techniques, des modèles et des démarches facilitant la spécification et la réutilisation des aspects dynamiques des systèmes d'information. La thèse d'Agnès Front [Front 97] a initialisé cet axe en proposant un langage de patrons[15] permettant l'analyse, la conception et l'implantation des aspects réactifs des systèmes d'information. L'objectif consistait à utiliser l'approche des patrons pour représenter une démarche de spécification et d'implantation des situations comportementales. Cette démarche basée sur l'utilisation successive de six patrons corrélés facilite une spécification incrémentale aisément traduisible dans une base de données active par des méthodes de classes et de règles actives. Le langage SCalP garantit la traçabilité des spécifications comportementales et non leur réutilisation [Front 99a]. En effet, il ne s'agit pas de réutiliser des spécifications dynamiques mais le processus permettant de les spécifier.

La thèse de Christophe Saint-Marcel [SaintMarcel 99c] s'inscrit dans le même axe de recherche en mettant l'accent sur la réutilisation de comportements types d'objets (par exemple les différents comportements possibles de ressources : multi-exemplaires, limitées, consommables, etc.). La section 3.2 présente les concepts de base du modèle NCR (Notion Rôle Comportement) dont les objectifs sont :

- d'assurer une meilleure traçabilité des activités de spécification et d'implantation des aspects dynamiques des SI,

[15] Un langage de patrons est « une collection structurée de patrons construits l'un sur l'autre pour transformer les besoins et les contraintes dans une architecture » [Coplien 96].

- de faciliter la réutilisation des spécifications dynamiques exprimées sous la forme de statecharts [Saint Marcel 99b].

3.1.3.2. Patrons d'ingénierie : SCalP, Meninge et POSEIDON

La deuxième catégorie de travaux de l'équipe s'attache à développer :

- des patrons d'analyse dédiés aux domaines étudiés : les situations comportementales pour SCalP, la méta-modélisation pour *M*eninge (cf. §2.4) et les Systèmes d'Information Produit[16] (SIP) pour POSEIDON. Nous parlerons de patrons métiers.
- des patrons de conception dépendants d'une technologie cible : les SGBD actifs pour SCalP, les systèmes à objet réflexifs pour *M*eninge, les SGDT (Systèmes de Gestion de Données Techniques) [Randoing 95] pour POSEIDON.

Nous présentons brièvement ces deux niveaux de patrons au travers du projet POSEIDON [Rieu 97a, Cauvet 98, Gzara 99] dont l'objectif est de définir un cadre méthodologique pour le développement des Systèmes d'Information Produit (SIP). Comme pour la conception de systèmes d'information de gestion [Espinasse 97], la conception de SIP est appréhendée selon deux grands niveaux de préoccupation. Le niveau organisationnel conduisant à spécifier le système d'information organisationnel (SIO) auquel sont associés des patrons métiers. Le niveau technique (informatique) concernant à spécifier la partie de ce SIO qui donne lieu à informatisation, c'est le système d'information informatisé (SII) qui conduira à spécifier et réutiliser des patrons de conception.

- Au niveau organisationnel, les **patrons métiers** prennent en compte deux formes essentielles de modélisation : la modélisation des produits et la modélisation des processus [Harani 97]. Concernant la modélisation des produits, il s'agit d'utiliser et d'adapter les solutions de modélisation relevant de l'ingénierie des systèmes d'information. Des adaptations sont nécessaires pour prendre en compte l'évolution des produits à travers des changements (notion de versions, d'alternatives, de révisions de produits). Concernant la modélisation des processus nous nous inspirons des travaux relevant du génie industriel [Vernadat 94, Ladet 95] mais aussi de travaux réalisés dans le domaine de l'ingénierie des processus en systèmes d'information [Rolland 96].
- Au niveau technique, la définition des **patrons de conception** est fortement liée à une technologie cible en l'occurrence les SGDT qui sont à la base du développement des SIP. La généricité d'une modélisation à base de patrons provient par contre de son

[16] L'objet d'un SIP est de gérer l'ensemble du patrimoine informationnel du produit tout au long de son cycle de vie, depuis sa première évocation en stratégie d'entreprise jusqu'à sa destruction. En d'autres termes, il s'agit de l'ensemble des informations techniques nécessaires à la conception, à la production, à la commercialisation et au soutien logistique du produit.

indépendance vis-à-vis d'un logiciel particulier. Il s'agit donc de prendre en compte les concepts et mécanismes communs aux SGDT. Les SGDT utilisent très fréquemment des modèles de bases de données et des modèles de « workflows » qui peuvent être intégrés dans le cadre méthodologique proposé afin de prendre en compte l'implantation des modèles de produits et de processus.

La définition et la spécification de patrons de conception associés aux SIP s'appuient sur les fonctionalités proposées par la plupart des SGDT [Rieu 97a] et les catalogues de patrons de conception déjà disponibles en génie logiciel [Gamma 95]. En ce qui concerne les patrons métiers, intervenant principalement en expression des besoins et dans la définition de spécifications fonctionnelles du SIP, il s'agit de les identifier à partir d'une analyse de domaine (cf. §3.1.2.2). Ce travail fait l'objet de la thèse de L. Gzara. Cette thèse est co-encadrée avec Michel Tollenaere, professeur à l'Ecole de Génie Industriel de l'INP de Grenoble.

Cette approche consistant à développer des systèmes par utilisation de patrons de différents niveaux d'abstraction se heurte à de nombreuses difficultés. En particulier, les démarches de développement à base de patrons n'en sont qu'à leur début (cf. § 3.1.2.3). On ne sait pas ou peu organiser, composer, spécialiser, instancier des composants conceptuels tels que les patrons d'analyse et de conception. C'est ce dernier point qui nous semble aujourd'hui fondamental pour systématiser l'usage des bibliothèques de composants tout au long des processus de développement. Les (bons) composants réutilisables sont extrêmement difficiles à identifier, spécifier et implanter[17]. Même le meilleur des composants ne sera jamais utilisé si l'on ne donne pas les moyens :

- de le sélectionner en fonction du problème à résoudre,
- de l'adapter en fonction du contexte,
- de le composer à d'autres pour résoudre un problème plus global.

Les différents projets SCALP, *M*eninge puis POSEIDON nous ont permis d'acquérir une expérience pragmatique concernant :

- les opérations permettant d'adapter et de composer des adaptations de patrons,
- les relations qui nous semblent aujourd'hui fondamentales pour organiser des bibliothèques de patrons et faciliter leur sélection.

[17] " Reusable objet class are like poems – it is easy to talk about them, but it is hard to write a good one " [Nierstrasz 92]

La suite de ce chapitre est destinée à illustrer les deux types de travaux de recherche de l'équipe. La section 3.2 présente les concepts de base du modèle NCR. La spécification du modèle et sa réalisation sont décrites dans le mémoire de thèse de C. Saint-Marcel [SaintMarcel 99c]. La section 3.3 constitue un rappel du concept de patron orienté objet et donne trois exemples (cf. §3.3.1) utilisés pour illustrer les opérations et les relations (cf. §3.3.2).

3.2. Le modèle Notion – Comportement –Rôle

Ce travail fait l'objet de la thèse de C. Saint-Marcel, co-encadrée avec Ph. Morat.
Il a été mené dans le cadre des équipes Storm et Adèle du LSR.

Contrairement aux LOO, aux SRCOO et aux SGBDOO, les Méthodes de Conception Orientées Objet n'ont pu se contenter d'une représentation unique des systèmes. Comme chez leurs précurseurs (méthodes cartésiennes et systémiques) de nombreux modèles se sont révélés nécessaires pour représenter les systèmes sous différents aspects (statiques, dynamiques et fonctionnels) et à différents niveaux d'abstraction. La première moitié des années 1990 a vu fleurir une cinquantaine de méthodes à objet [Muller 97], pour finalement aboutir à un consensus sous la forme d'une proposition unique [Booch 99]. Si UML reprend l'ensemble des modèles de produits ayant fait leurs preuves ses dernières années, force est de reconnaître que les processus de développement ne sont guère plus explicités et encore moins formalisés. Les modèles de processus restent encore du domaine de la recherche et aucun atelier de développement n'intègre encore une aide efficace en terme de gestion de la cohérence inter-modèles, de transformation d'un modèle à un autre, ni même de raffinement de modèles.

Les travaux entrepris dans NCR concernent cette problématique en se focalisant sur les aspects comportementaux des méthodes de conception. Deux modèles ayant fait leur preuve sont privilégiés : les diagrammes de classes et les diagrammes d'états. Si on sait bien aujourd'hui utiliser et réutiliser les premiers, les seconds sont souvent sous-employés lors de l'ingénierie des SI. Plusieurs raisons détaillées et illustrées dans [SaintMarcel 98] peuvent être citées. Intuitivement et pour résumer nous pouvons dire que « le prix à payer est trop cher ». En effet, l'expression complète et cohérente d'un statechart est un exercice de modélisation :

- **difficile** dès lors qu'il s'agit d'exprimer l'évolution d'objets complexes et co-occurrents,
- **peu réutilisable** dans une modélisation objet où un statechart décrit le comportement des objets d'une seule classe[18],

[18] Dans la plupart des méthodes objets une classe est définie par des propriétés statiques (les attributs et les associations), fonctionnelles (les méthodes) et comportementales (les états et les transitions). Les propriétés

- **peu exploitable** au niveau des spécifications car l'intégration des propriétés statiques et dynamiques, souvent jugée difficile, est repoussée lors des étapes d'implantation.

En partant de ce constat, les travaux dans NCR ont pour objectif de proposer un environnement de modélisation en trois dimensions : structurelle, comportementale et phénoménale[19]. Les deux premières sont destinées à l'ingénierie de composants réutilisables, la troisième est dédiée aux développements de SI par réutilisation.

3.2.1. Composants structurels et comportementaux

Les dimensions structurelle et comportementale sont dédiées à l'identification, la spécification et l'organisation de composants structurels et comportementaux de différents niveaux d'abstraction. Les composants structurels, appelés *notions*, spécifient les propriétés statiques et fonctionnelles des objets métiers et logiciels des organisations indépendamment des comportements qu'ils *pourront* adopter au sein des applications. Inversement les composants comportementaux offrent une spécification de comportements abstraits (pensez par exemple à tous les comportements types de ressources) possibles dans cette organisation indépendamment des objets *pouvant* les adopter au sein des applications. L'idée de base consiste donc, au sein d'une organisation, à favoriser une dissociation complète des propriétés structurelles et comportementales des objets. Un comportement n'exprime plus l'évolution des objets d'une classe mais une évolution potentielle d'un grand nombre d'objets de l'organisation.

Les **notions** peuvent être déterminées classiquement par une analyse de domaine (cf. §3.1.2.3), le modèle de domaine résultat est représenté en NCR par un diagramme de classes mettant en évidence les propriétés statiques et fonctionnelles communes et partagées par les objets communs aux organisations de même domaine. Pour des bibliothèques, des exemples classiques de notions sont les documents, les ouvrages, les livres, les abonnés, etc. Dans le cadre d'une organisation, la dimension structurelle est à affiner en fonction de la spécificité des objets de l'entreprise. Pour les « merisiens », il s'agit de prendre en compte les règles de gestion[20] c'est-à-dire les invariants ou contraintes portant sur les objets métiers de l'organisation. Une bibliothèque pourra par exemple imposer un nombre maximal

---

comportementales sont comme les autres héritées par les sous-classes. Par contre le mécanisme d'héritage reste généralement mal défini.

[19] Phénoménal : (Philos.) Qui concerne les apparences des choses : le monde phénoménal s'oppose au monde nouménal des choses en soi.

[20] Dans la méthode Merise, les règles de gestion sont la « traduction conceptuelle des objectifs et des contraintes acceptées par l'entreprise »… « Leur origine est soit externe à l'entreprise (lois, règlements, etc.), soit interne à l'entreprise (règlements intérieurs, choix de gestion, etc.) » [Collongues 86]

d'exemplaires par ouvrage restreignant ainsi la cardinalité de l'association entre un ouvrage et ses différents exemplaires.

Les **comportements** correspondent à des évolutions abstraites et réutilisables d'objets modélisées par des statecharts[21]. Dans les systèmes d'information, des exemples classiques de comportements réutilisables sont les cycles de conception et de maintenance des produits, d'occupation des ressources, etc. La dimension comportementale capitalise donc des comportements très généraux adaptables à tout domaine par exemple les différents comportements de ressources. L'identification des comportements relève donc plus d'une analyse de situations comportementales types inter-domaines, contrairement aux notions qui correspondent aux propriétés structurelles d'objets de domaine. Ici aussi une organisation utilisant NCR devra étendre ou affiner ces évolutions types en fonction des contraintes de l'organisation, par exemple en fonction de politiques particulières de gestion de ressources.

Un comportement est défini dans NCR comme « une représentation des évolutions multiples d'un objet selon une ou plusieurs caractéristiques identifiables et calculables » appelées caractéristiques d'évolution [SaintMarcel 98].

La Figure 3-8 illustre la représentation d'un comportement de « Ressource Limitée ». Chaque comportement est associé à un ensemble de caractéristiques (ECE = Ensemble des Caractéristiques d'Evolution) dont les valeurs caractérisent les états. Dans NCR, un objet qui veut rester dans un état doit vérifier l'invariant de cet état. C'est ainsi qu'un objet adhérant au comportement RessourceLimitée doit vérifier nbOccurences = MIN pour se rendre Non Disponible.

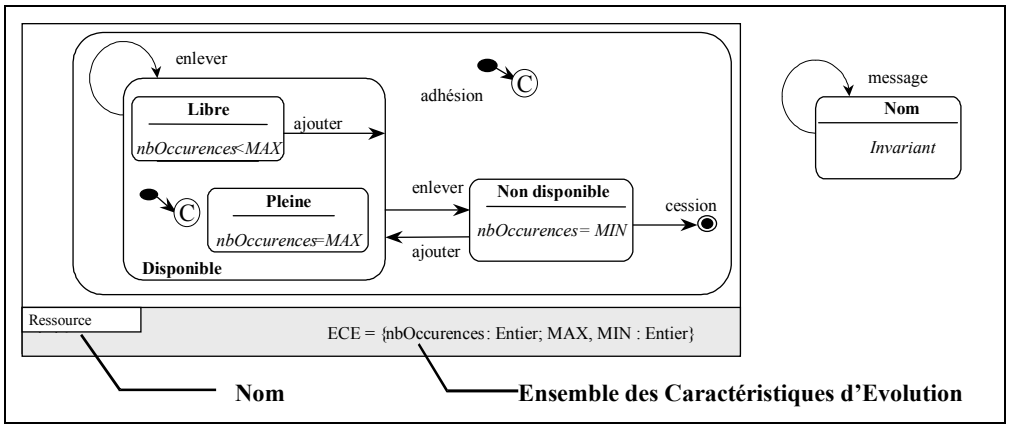


Figure 3- 8 : Comportement « Ressource Limitée »

[21] La sémantique des statecharts a été modifiée pour prendre en compte ce caractère abstrait et réutilisable. Pour plus de détails voir [SaintMarcel 99c].

3.2.2. La dimension phénoménale

La troisième dimension est destinée au développement par réutilisation des systèmes d'information. L'objectif est ici de **réutiliser, d'adapter et de fusionner** les spécifications détenues dans les deux autres dimensions. La spécification d'une application est obtenue en sélectionnant des notions correspondant aux objets métiers (lors de l'analyse) ou logiciel (lors de la conception) intervenant dans l'application et à les doter d'un comportement décrivant leur évolution au sein de cette application.

L'idée de base consiste donc à animer des composants structurels (les notions) par des composants comportementaux (les comportements) pour obtenir des modèles complets d'objets, appelés *rôles*. Le rôle sert ainsi de " charnière " conceptuelle entre les dimensions structurelle et comportementale : en général un rôle anime une et une seule notion selon un unique comportement qu'il concrétise. C'est ainsi que dans une application de gestion de prêts d'une bibliothèque le rôle « livre empruntable » réalise l'animation de la notion « livre » en la dotant d'un comportement de « ressource mono-exemplaire » (cf. Figure 3- 9) .

Le rôle est le seul générateur d'instances dans NCR, c'est à lui que revient la charge de faire évoluer les objets en contrôlant leur comportement. Il concrétise les propriétés de son comportement en utilisant celles de sa notion. C'est ainsi que la transition « adhésion » du comportement « ressource mono-exemplaire » est réalisée par la méthode enRayon de la notion Livre (cf. Figure 3- 9). Un rôle peut admettre des propriétés spécifiques qui ne sont issues ni de sa notion ni de son comportement. Il s'agit de propriétés liées à la mis en œuvre de l'application. Dans un système d'information, elles correspondent la plupart du temps à l'expression formalisée de règles d'organisation[22].

[22] Dans le méthode Merise, une règle d'organisation traduit l'organisation mise en place pour atteindre les objectifs et les contraintes fixées par les règles de gestion.

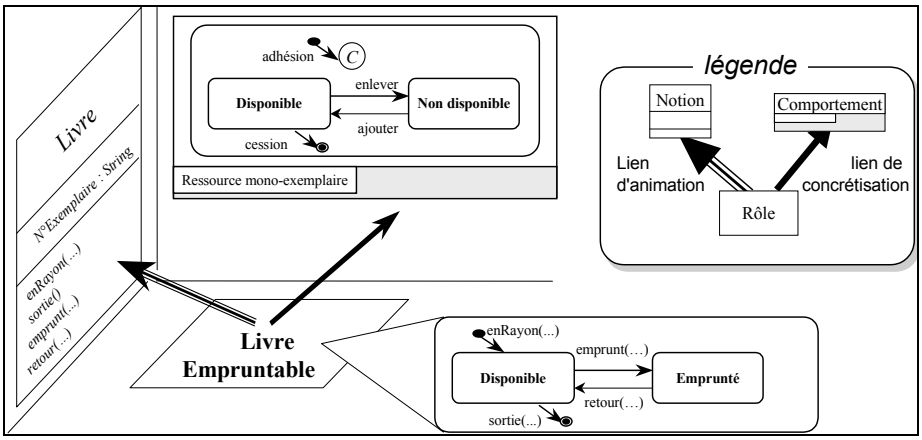


Figure 3- 9 : Trois paradigmes de modélisation

La notion Livre, qui fait partie de la dimension structurelle, donne une spécification générale d'un livre que l'on souhaite exploiter dans le cadre de la gestion des prêts d'une bibliothèque. La dimension comportementale fournit un comportement Ressource mono-exemplaire. La correspondance entre la notion Livre et le comportement Ressource mono-exemplaire qu'on veut lui donner est établie par le rôle Livre Empruntable[23]. Celui-ci assure par exemple le renommage de l'état Non Disponible, la fusion des méthodes emprunt() et retour() respectivement avec les transitions enlever et ajouter. Nous pouvons remarquer qu'à la création d'une instance de rôle, l'état d'entrée par défaut est Disponible. Il s'agit là d'une restriction par rapport au choix laissé dans le comportement (marqué par un ©).

3.2.3. Bilan

L'originalité du modèle NCR est liée à la « priorité » qui est donnée aux modèles. La structure n'est plus favorisée par rapport au comportement, chacun étant dissocié et réutilisé indépendamment. Cependant, on ne force pas à une approche comportementale systématique, on la favorise seulement. Le concepteur n'a donc pas à décrire de manière exhaustive le comportement d'un objet mais seulement les parties qu'il veut contrôler : sur l'exemple de la Figure 3- 9 est spécifié le comportement du livre dans le processus de prêt et volontairement ignoré celui lié aux processus de réservation et de maintenance.

Pour faciliter la réutilisation, tant des notions que des comportements, les dimensions structurelle et comportementale sont organisées en domaines où chaque domaine détient des composants réutilisables de même nature (structurelle ou comportementale) ; par exemple le domaine des documents dans les notions, celui des ressources dans les comportements, etc. Les liens intra-domaine correspondent à des liens d'héritage.

[23] NCR permet d'exprimer graphiquement et textuellement les rôles.

Dans un domaine comportemental, des comportements très généraux (inter-domaines) cohabitent avec des comportements plus spécifiques liés à un métier ou à une organisation mais indépendants des applications. La dimension comportementale représente ainsi des évolutions *possibles* d'objets. La dimension de rôles décrit au contraire des évolutions effectives d'objets au sein des applications. Ce modèle établit donc clairement le distinguo entre les évolutions possibles (comportements) et les évolutions effectives (rôles).

Des fragments des processus de développement des applications et de maintenance des dimensions structurelle et comportementale sont décrits par un ensemble de patrons corrélés [SaintMarcel 99c].

3.3. Organisation et utilisation des patrons

Ce travail est le résultat d'une réflexion de groupe menée à partir de nos différents projets : ScalP, Meninge et POSEIDON. Il a fait l'objet de deux publications récentes [Front 99b, Rieu 99] rédigées en collaboration avec J.P. Giraudin, A. Front-Conte et C. Saint-Marcel.

3.3.1. Patrons orientés objet et exemples

Cette section est destinée à introduire le concept de patrons et à proposer trois exemples qui serviront à illustrer concrètement les opérations et les relations introduites au §3.3.2.

Le principe de base du concept de patron consiste à capitaliser un problème récurrent d'un domaine et sa solution de manière à faciliter la réutilisation et l'adaptation de cette solution lors d'une nouvelle occurrence du problème. Un patron constitue donc une base de savoir-faire :

- permettant d'identifier le problème à résoudre,
- proposant une solution correcte et si possible consensuelle pour y répondre,
- offrant les moyens d'adapter cette solution à un contexte spécifique.

Les premiers patrons orientés objet ont été présentés par K. Beck et W. Cunninghan [Beck 87] lors de la conférence OOPSLA de 1987. Il s'agissait d'une première adaptation du langage de patrons d'Alexander[24] à la conception et à la programmation objet. Quelques années plus tard, dans le cadre de l'ingénierie des systèmes d'information, P. Coad [Coad 92] propose de faciliter l'analyse d'un système en identifiant les besoins selon sept patrons pré-établis. Il définit un patron orienté objet comme une abstraction d'un doublet, d'un triplet ou d'un ensemble de classes qui peut être réutilisé encore et encore pour le développement d'applications. Un premier catalogue plus spécifiquement destiné à la conception des logiciels a été élaboré par E. Gamma en 1991 à Zurich lors des travaux de son Ph.D. Ces travaux ont été suivis en 1995 d'un premier ouvrage collectif [Gamma 95] développant plus largement cette approche. Depuis, de nombreux ouvrages et articles ont été publiés par G. Booch, R. Helm, E. Gamma, P. Coad, J.O. Coplien, F. Buschmann, M. Fowler, etc. La première conférence internationale dédiée à la notion de patrons de conception date de 1994 (PLoP), la première conférence européenne de 1996 (EuroPLoP). En France, un numéro spécial de la revue l'Objet [Objet 99] lui a été consacré en 1999.

Les patrons orientés objet sont comparables aux patrons dédiés à la confection vestimentaire ou à l'architecture. Ils facilitent le processus d'ingénierie des systèmes à objet en proposant des artefacts prédéfinis, adaptables à des problèmes similaires et à des

[24] Le terme patron de conception (*design pattern*) a été introduit, dans les années 1970, dans le domaine de l'architecture des bâtiments par C. Alexander [Alexander 77, Alexander 79].

technologies différentes d'implantation. Ils sont généralement classifiés en fonction de l'étape d'ingénierie à laquelle ils s'adressent.

- Les **patrons d'analyse** [Rolland 93, Coad 92, Fowler 97b, Cauvet 98, Gzara 99] aident le concepteur dans la construction de modèles représentant au mieux les besoins de son système.
- Les **patrons de conception** (design patterns) identifient, nomment et abstraient des thèmes communs du domaine de la conception objet qu'il s'agisse de conception détaillée [Gamma 95], globale [Buschmann 96] ou liée à des architectures particulières [Jézéquel 99], etc.
- Les **patrons d'implantation,** aussi appelés idiomes [Coplien 92], sont généralement spécifiques à un langage. Ils décrivent comment implanter dans un langage particulier certaines caractéristiques généralement absentes (par exemple comment conserver le lien dynamique lors d'une déclaration de variable en C++ [Pree 98]).

Tout patron doit présenter ses trois principaux éléments (le problème, le contexte et la solution) dans un **formalisme de représentation**. Plusieurs formalismes[25] peuvent être utilisés : la "Portland Form[26]", le formalisme de P. Coad [Coad 96] ou le formalisme du groupe des quatre [Gamma 95] que nous appellerons par la suite formalisme de Gamma. Ces différents formalismes se distinguent essentiellement par le nombre de rubriques plus ou moins détaillées qu'ils proposent. Ainsi, le formalisme de P. Coad est composé de 7 rubriques alors que celui de Gamma en comporte 13.

Afin de simplifier la compréhension des patrons que nous présentons dans la suite, nous adaptons le formalisme de Gamma en ne retenant que les 5 rubriques que nous jugeons essentielles : nom, intention, motivation, solution, implantation du patron.

Nom du patron	
Intention	Le problème auquel le patron s'adresse, son point fort.
Motivation	Un scénario (un exemple) d'application du patron décrit de manière textuelle et éventuellement graphique.
Solution	La solution semi-formelle proposée par le patron, exprimée à l'aide de diagrammes UML décrivant en particulier les participants du patron ainsi que leurs collaborations.
Implantation	Des astuces, des conseils et des techniques utilisables pour implanter le patron.

Nous utilisons UML pour présenter des patrons (rubrique Solution) qui pour la plupart sont décrits dans la littérature avec d'autres formalismes. Les solutions proposées pour ces patrons ont parfois été complétées ou au contraire simplifiées. D'autre part, l'intention est ici prise au sens large : elle définit aussi bien le problème du patron que son contexte d'utilisation, deux rubriques qui sont actuellement distinguées dans les formalismes de P. Coad ou d'E. Gamma.

[25] Bien que très utilisé, le terme « formalisme » est abusif. Il s'agit plutôt de formulaire de représentation.
[26] http://www.c2.com:80/ppr/about/portland.html

3.3.1.1. Patron d'analyse Nouvelle-Opération-Bancaire

Ce patron a pour origine le cours " UML 1.1 " donné par la société Arche-SQL à Nanterre en Mars 1998. Il a été spécifié et utilisé par des étudiants du département d'informatique de l'IUT2 de Grenoble [Front 99b].

Patron Nouvelle-Opération-Bancaire

Intention : Ce patron permet de traiter uniformément toute demande de prise en compte d'une nouvelle opération bancaire.

Motivation : La prise en compte d'une nouvelle opération bancaire doit être traitée de manière uniforme pour faciliter la maintenance et l'évolution du système. La Figure 3- 10 propose en partie gauche un cas d'utilisation spécifique traitant la création d'un nouveau compte et en partie droite un cas d'utilisation générique. Notons dès ce niveau la similitude avec le cas d'utilisation traitant le retrait d'une somme d'un compte (cf. Figure 3- 1). Une analyse plus détaillée confirme cette similitude. Elle est illustrée par les diagrammes de séquences détaillés (cf. Figure 3- 2 et Figure 3- 11 mettant en évidence les collaborations entre objets.

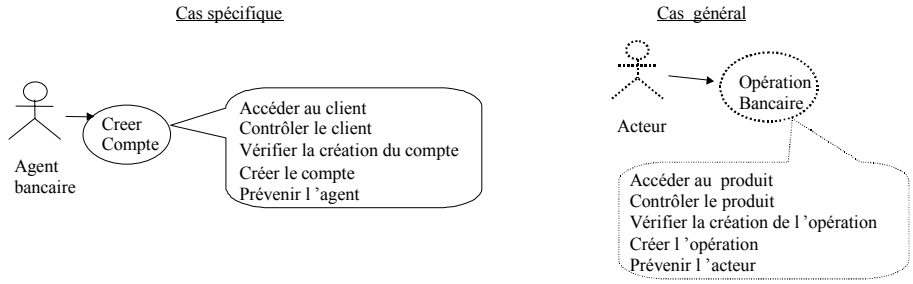


Figure 3- 10 : Cas d'utilisation des opérations bancaires

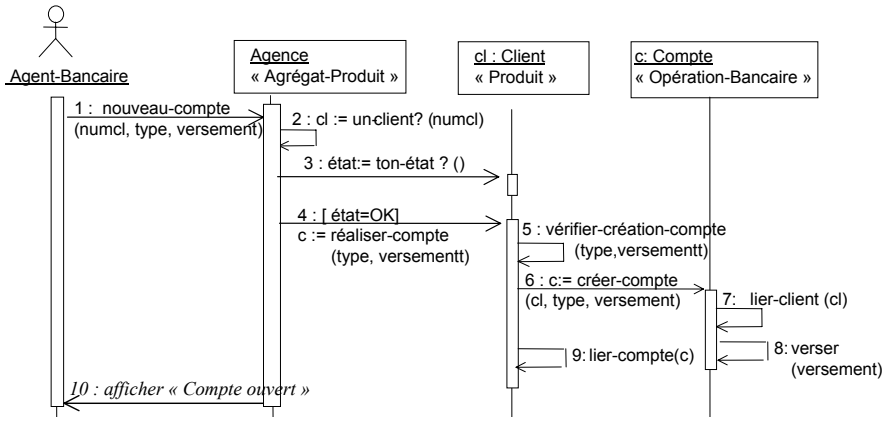


Figure 3- 11 : Diagramme de séquences spécifiques

Solution : La classe Agrégat-Produit détient une méthode permettant la réalisation de la demande d'opération ; elle connaît et contrôle ses produits. La classe Produit connaît et contrôle la création de ses opérations bancaires. La classe Opération-Bancaire maintient un lien vers son produit.

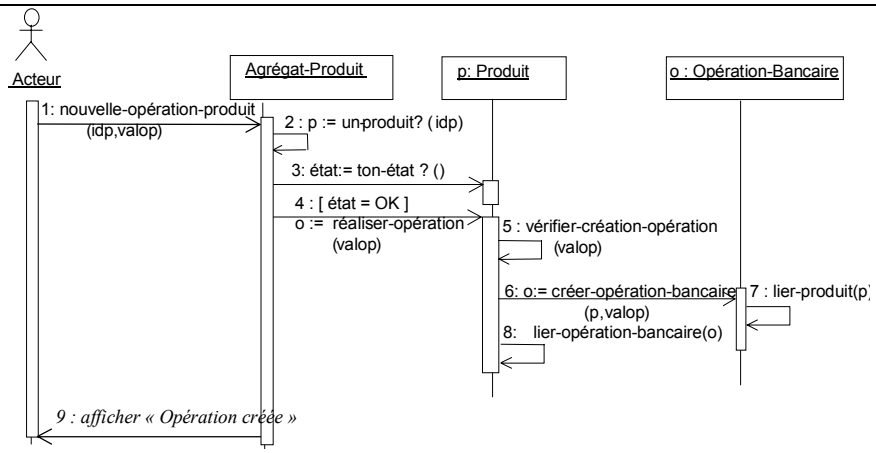


Figure 3- *12: Diagramme de séquence général pour une nouvelle opération bancaire*

La Figure 3- 12 illustre par un diagramme de séquence général, les collaborations entre les trois classes pour la prise en compte d'une nouvelle opération bancaire. Le diagramme de classes de la Figure 3- 13 précise les associations entre ces classes et leurs méthodes déduites du diagramme de séquence. Dans cette figure comme dans les suivantes, nous faisons le choix pratique d'utiliser le symbolisme graphique du paquetage pour englober le diagramme de classes d'un patron alors qu'en UML, ce symbolisme représente un framework, un sous-modèle, un sous-système, etc. Le paquetage sera en traits pointillés pour un patron et en trait plein pour un modèle issu d'un patron.

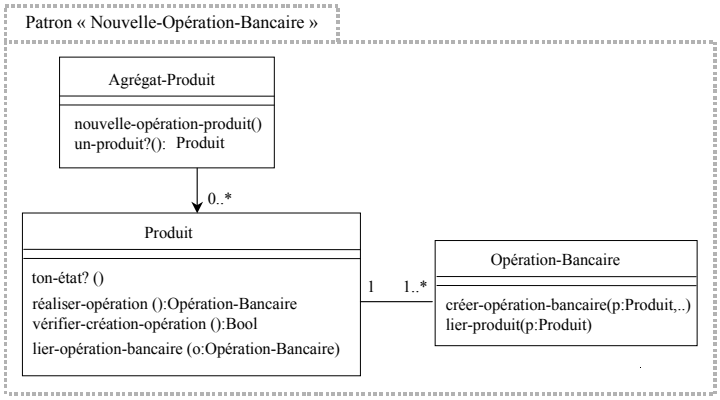


Figure 3- 13: Diagramme de classes pour une nouvelle opération bancaire

3.3.1.2. Patron d'analyse Rôle

Patron Rôle [Coad 92]

Intention : A un moment donné, un objet peut jouer plusieurs rôles. Le patron Rôle donne la possibilité à l'objet d'adhérer et/ou de perdre ces rôles.

Motivation : Le patron Rôle est utilisé lorsqu'un objet a des propriétés variables selon le rôle joué. Un livre a par exemple des propriétés intrinsèques telles que le numéro d'exemplaire, le N° ISBN, le nombre de pages, etc. Dans un contexte particulier, par exemple une bibliothèque, il pourra jouer plusieurs rôles fortement dépendants des processus métiers de la bibliothèque : l'emprunt, la réservation et la maintenance.

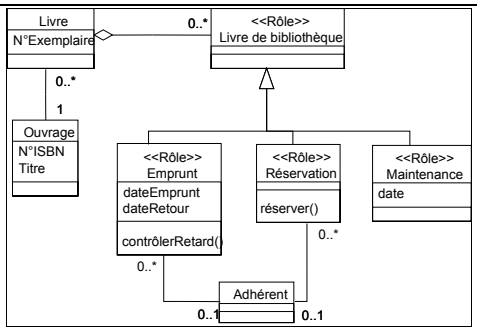


Figure 3- 14 : Différents rôles d'un livre de bibliothèque

Solution : La classe Acteur est associée à une classe Rôle abstraite. Chaque acteur est lié à des instances des classes Rôles concrètes (Rôle1, Rôle2, etc.), chacune d'elle représentant un de ses rôles. Cette approche a l'avantage d'être plus concise et flexible que l'utilisation de l'héritage multiple.

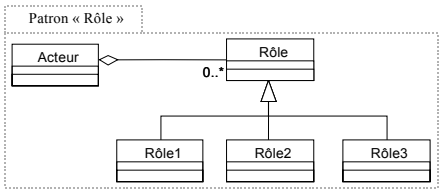


Figure 3- 15 : Diagramme de classes du patron Rôle

3.3.1.3. Patron de conception Etat

Patron Etat [Gamma 95]

Intention : Le patron Etat permet à un objet de modifier son comportement quand son état interne change.

Motivation : Dans le cadre du processus d'emprunt des livres d'une bibliothèque, chaque livre passe par différents états. Son comportement est lié à son état courant. C'est ainsi qu'un retour n'a de sens que si le livre est dans l'état emprunté. Le patron Etat permet de gérer une évolution simple d'objet qui peut être décrite par un diagramme de transitions d'états où chaque état est modélisé par une classe. L'objet à un instant donné ne peut être que dans un unique état (état courant). La classe abstraite Emprunt (cf. Figure 3- 16) représente les différents états du livre. Elle définit une interface commune à toutes ses sous-classes représentant les états opérationnels : Emprunté et Disponible. Les sous-classes de Emprunt implantent les comportements spécifiques.

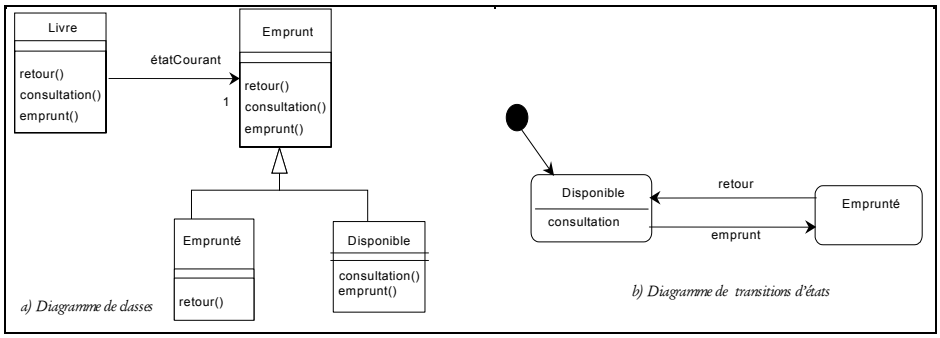


Figure 3- 16 : Un état est modélisé par une classe

Solution : La classe Contexte définit l'interface pour les clients ; les états sont donc transparents pour l'utilisateur. Chaque contexte est lié à une instance qui définit son état courant. La classe Etat spécifie l'interface commune de toutes ses sous-classes; il s'agit d'une classe abstraite. Les sous-classes concrètes de la classe Etat implantent le comportement spécifique associé à un état du contexte.

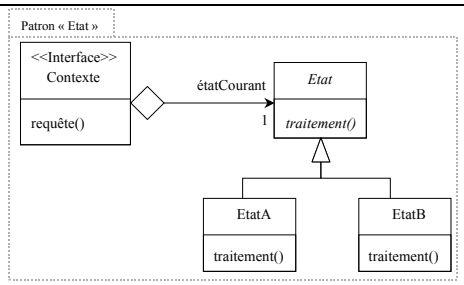


Figure 3- 17 : Structure du patron Etat

Les interactions peuvent être décrites à l'aide d'un diagramme de séquence : (1) le contexte reçoit une requête, (2) il la délègue à l'état courant ; le contexte est passé en argument à l'état qui traite la requête ; cet état peut ainsi accéder aux données du contexte et réaliser le traitement ; (3) s'il y a lieu, l'état notifie au contexte le changement d'état en passant en paramètre le nouvel état courant.

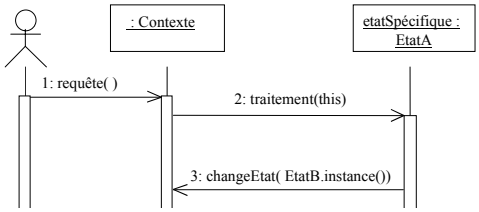


Figure 3- 18 : Dynamique du patron Etat

La méthode changeEtat() n'est visible que par les états du contexte. La méthode instance() d'EtatB est une méthode statique qui permet l'obtention d'un singleton partagé de la classe EtatB.

Implantation : L'implantation du patron Etat dépend des besoins : on doit se demander qui implante les transitions d'états. Il peut s'agir du contexte dans une solution centralisée ou des sous-classes d'Etat dans une solution distribuée. La deuxième solution est plus flexible même si elle oblige chaque état à connaître ses successeurs en créant ainsi une dépendance d'implantation.

3.3.2. Opérations et relations sur les patrons

La question délicate posée ici est :

Comment utiliser, transformer et agencer des patrons pour décrire et réaliser des systèmes mieux construits en augmentant le taux de réutilisation ?

Même si cette question est actuellement du domaine de la recherche, les expériences menées au niveau des différents projets du groupe ont mis en évidence des techniques encore assez " frustres " mais qui offrent déjà de bonnes potentialités [Front 99b, Rieu 99].

Ces techniques sont introduites en deux parties. La première partie (§3.3.2.1) concerne **l'ingénierie des systèmes** basée sur la réutilisation de patrons ; dans ce cadre, nous décrivons les opérations d'imitation de patrons et d'intégration d'imitations de patrons.

La deuxième partie (§3.3.2.2) s'intéresse à un problème plus délicat : **l'ingénierie des patrons**. Nous abordons ce problème en décrivant trois relations inter-patrons : l'alternative, le raffinement/extension et l'utilisation. Ces relations sont destinées à organiser des bibliothèques de patrons afin de faciliter leur sélection lors du développement des systèmes.

3.3.2.1. Réutilisation de patrons pour l'ingénierie des systèmes

- *Imitation de patrons*

L'application la plus simple des patrons est certainement du type " imitation " d'une solution à un problème analogue. On retrouve cette opération de base brièvement évoquée dans l'article de P. Coad [Coad 92] introduisant sept patrons généraux d'analyse et de conception et appliquant sommairement ces patrons au domaine de la vente. Un patron n'est pas directement instancié au sens de l'instanciation objet. Le concepteur doit donc dupliquer sa solution, puis l'adapter à un contexte donné.

Imiter un patron *= dupliquer + adapter la solution du patron*
Adapter un duplicata de patron *= (renommer | redéfinir | ajouter | supprimer)**
des propriétés statiques (attributs), dynamiques (méthodes) et relationnelles (associations)

Il est facile d'imaginer un concepteur désirant spécifier la partie d'un système bancaire concernant la création d'un nouveau compte pour un client. En supposant que dans une bibliothèque de patrons, il dispose du patron Nouvelle-Opération-Bancaire (cf. §3.3.1.1), il va réaliser une opération de duplication de ce patron suivie d'une adaptation concrète des 3 classes pour obtenir le modèle " Nouveau Compte " (cf. Figure 3- 19 (a)).

On peut remarquer que cette adaptation se résume à renommer des classes et des propriétés. Une telle adaptation peut se poursuivre par la redéfinition, l'ajout ou la suppression de propriétés. C'est par exemple le cas de la méthode verser() qui est ajoutée à la classe Compte.

Le diagramme de la Figure 3- 19 (b) décrit également le modèle " Retrait-Compte ", résultat d'une nouvelle duplication suivie d'une adaptation du même patron. Ces deux imitations du même patron mettent en évidence que cette première opération naturelle sur les patrons n'est ni une instanciation, ni une spécialisation, mais essentiellement une différenciation dirigée par le contexte. Certains outils de spécification introduisent néanmoins des opérations d'instanciation (cf. §3.1.2.3) basées sur une réification des solutions génériques des patrons [Meijler 97].

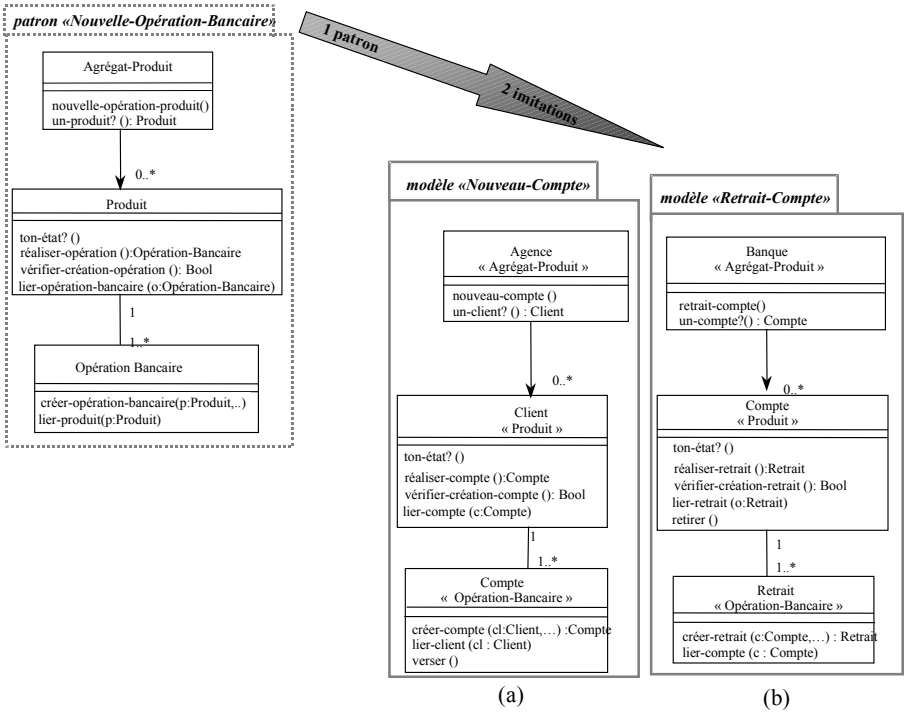


Figure 3- 19 : Imitations d'un patron

- *Intégration d'imitations de patrons*

Le résultat de l'imitation de plusieurs patrons ou de plusieurs imitations du même patron correspond donc à plusieurs sous-modèles qu'il s'agit maintenant d'intégrer afin de disposer d'un modèle cohérent du système à concevoir. Cette situation est analogue à celle de la conception de bases de données basée sur des techniques d'intégration de vues [Heiler 90, Rieu 92b, Souza 94, Ra 94]. Le concepteur se retrouve dans une situation classique en modélisation avec en particulier des problèmes de synonymies et d'homonymies de classes et de propriétés ainsi que de cohésion du modèle résultant. L'union constitue une technique de base d'intégration de modèles.

Intégrer des imitations *= (unir des classes)**

L'union consiste à remplacer deux classes par une seule. Cette nouvelle classe regroupe alors l'union des propriétés des deux classes initiales. Par exemple, dans le cadre de l'application bancaire, l'union des deux classes " Compte " regroupe les propriétés issues des

deux adaptations précédentes. Naturellement cette opération d'union doit tenir compte des conflits de noms et des polysémies sur les propriétés.

Remarquons qu'il s'agit bien ici d'une **union de rôles**[27], les rôles joués par un compte dans deux cas d'utilisation : le retrait où le compte joue le rôle de Produit et la création de compte où il joue celui d'Opération-Bancaire. Une union de rôles peut également être réalisée en utilisant le patron Rôle de P. Coad : la classe Compte est alors l'Acteur lié à ses différentes classes Rôle (cf. §3.3.1.2). Cette solution n'est pas satisfaisante et ne ferait qu'alourdir le schéma global. En effet, le patron Rôle est utilisé pour des objets changeant de rôles au cours de leur cycle de vie.

Intégrer des imitations = *(unir les rôles des classes)**

Après cette opération d'union dont l'objectif essentiel est de supprimer des redondances explicites ou implicites d'une modélisation, il est nécessaire de lier des éléments du modèle obtenu un peu comme le maçon qui assemble avec du mortier deux éléments préfabriqués. Une première approche consisterait à lier des éléments des modèles issus de différents processus de modélisation par des associations. Par exemple, le modèle de l'application bancaire peut être complété par une association de composition entre les classes Banque et Agence (cf. Figure 3- 20).

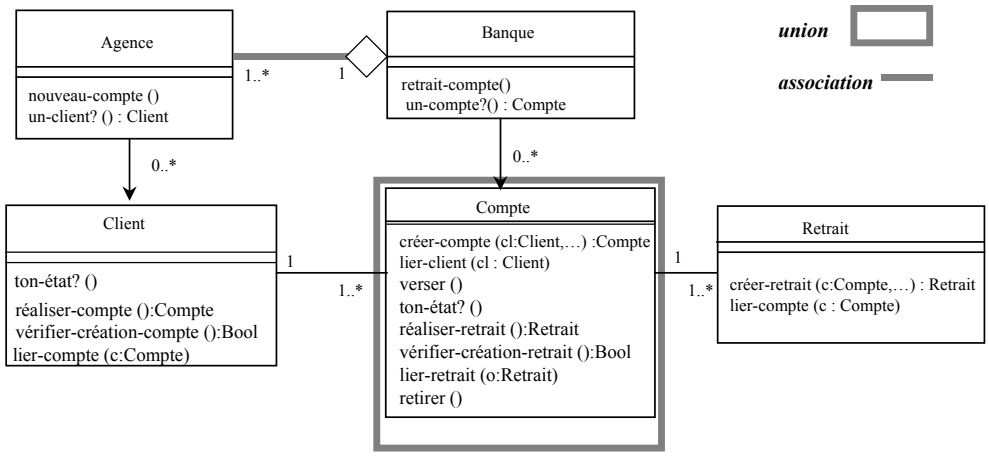


Figure 3- 20 : Intégration de deux imitations de patrons

Cependant, un concepteur expérimenté n'introduira pas une association sans prendre des précautions. Il réalisera une autre combinaison d'opérations sur des patrons en s'appuyant cette fois sur le patron Collection [Coad 92]. Ce patron permet de modéliser les propriétés et

[27] la notion de rôle est à prendre ici dans le sens intuitif du terme. Il s'agit d'une manière de faire le tri parmi l'ensemble de caractéristiques et des comportements possibles d'un objet.

les relations entre des objets d'une classe et des ensembles d'objets de cette classe : des personnes (membres) et des groupes de personnes (collections), des produits (membres) et des familles de produits (collections), etc. Ce patron est à appliquer lorsque des propriétés structurelles et/ou comportementales d'un objet « Collection » sont étroitement liées aux propriétés structurelles et/ou comportementales de ses différents objets « Membre ». Dans le cadre d'une banque (la collection) et de ses agences (les membres), ce patron est à adapter au contexte de la contrainte existentielle : une agence est impérativement rattachée à une et une seule banque ; la création et la suppression d'une agence sont réalisées à partir de la banque correspondante ; la déconnexion d'une agence d'une banque pour une connexion à une autre banque n'est pas autorisée.

Le diagramme de classes de la Figure 3- 20 aurait donc pu être obtenu comme l'union de trois modèles (cf. Figure 3- 21) : deux modèles issus de la double imitation du patron Nouvelle-Opération-Bancaire et un modèle issu du patron Collection [Coad 92].

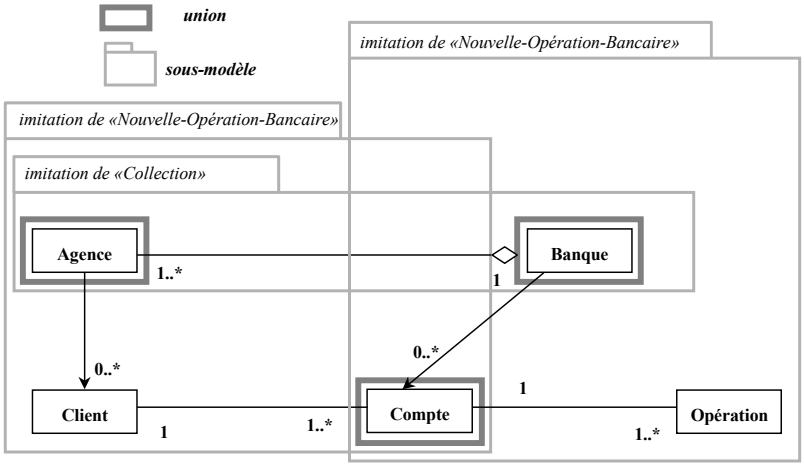


Figure 3- 21 : Intégration de trois imitations de patrons

Dans la mesure où les catalogues de patrons seraient extensibles, une question se pose.

La structure générale du modèle d'un système d'information peut-elle être construite exclusivement par imitation et union de patrons sans ajout de classes ou de liaisons originales ?

Il est tout d'abord évident que l'union de rôles ne se résume pas toujours au regroupement dans une même classe des propriétés de l'objet dans ses différents rôles. Une union peut nécessiter par exemple l'ajout d'un modèle résultat d'une imitation du patron Rôle. Dans ce cas une nouvelle classe est introduite. Mais il ne s'agit là que de l'application d'une technique objet : l'ajout de cette classe n'introduit aucune propriété sémantique.

Une deuxième réponse à cette question est en partie liée à la complétude des langages de patrons utilisés. Dans l'exemple précédent, si nous ne disposions pas du patron Collection, le modèle " Agence-Banque " devrait être spécifié fonctionnellement et structurellement sans s'appuyer sur une solution générique existante. Par contre, lors de la spécification de ce modèle, le concepteur expérimenté ferait naturellement une analogie avec des situations similaires (modèles Service-Société, Exemplaire-Livre, etc.) rencontrées lors de conceptions antérieures et reproduirait une solution très proche de celles qu'il a précédemment spécifiées. Dans une optique d'ingénierie de patrons (cf. §3.3.2.2.) une abstraction du modèle " Agence-Banque " pourrait alors être proposée puis documentée afin d'enrichir le catalogue de patrons.

La Figure 3- 22 illustre l'utilisation d'une part des opérations d'imitation et d'union lors du développement d'un système d'information bancaire et d'autre part l'opération d'abstraction inverse de l'imitation permettant de produire un nouveau patron.

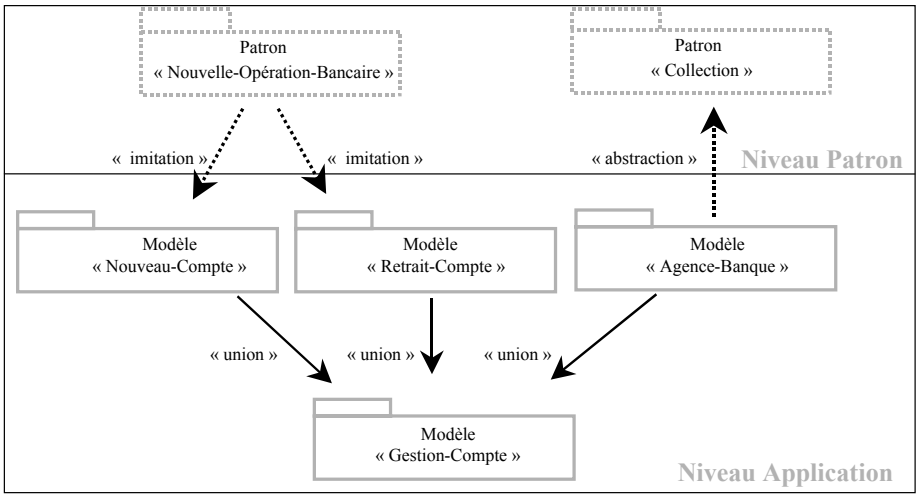


Figure 3- 22 : Spécification de système

Remarquons finalement que le niveau de granularité des patrons d'ingénierie est tout à fait arbitraire. Il est donc tout à fait possible dans des domaines d'applications bien connus de spécifier des patrons dont l'intention recouvre des domaines fonctionnels correspondant à des processus métiers. Dans le cas de la gestion bancaire, un patron générique Gestion-des-Comptes est tout à fait réalisable.

3.3.2.2. Réutilisation de patrons pour l'ingénierie des patrons

L'ingénierie de nouveaux patrons se fait rarement ex nihilo. Elle fait appel le plus souvent à des connaissances empiriques pour apporter une solution consensuelle à un problème donné. Ces connaissances peuvent ou non être formalisées sous la forme de patrons. Mais il est rare aujourd'hui qu'un problème ne puisse être résolu, du moins en partie, par des patrons existants. C'est pourquoi nous nous intéressons par la suite aux seuls patrons issus d'abstractions sur d'autres patrons. Il est cependant évident que bon nombre de patrons sont actuellement issus d'une analyse de domaine et de systèmes existants. Les opérations définies dans cette section doivent permettre de positionner à la fois le **problème** (rubrique Intention) du patron par rapport aux problèmes existants et la **solution** (rubrique Solution) proposée par rapport aux solutions existantes.

L'ingénierie de patrons est principalement une extraction de nouveaux problèmes et une découverte de nouvelles solutions.

Si les opérations définies précédemment (cf. §3.3.2.1) donnent une première piste pour spécifier les relations entre les solutions des patrons, elles n'adressent pas les relations entre intentions. Or une classification des patrons peut être obtenue à l'aide d'opérateurs de comparaison des intentions et solutions proposées. Cette classification doit permettre le positionnement de nouveaux patrons par rapport à des patrons existants. Une telle organisation peut être construite à partir de relations de dépendance entre patrons : **alternative**, **extension** / **raffinement** et **utilisation** de patrons.

- *Alternative*

L'alternative est une nouvelle proposition de solution pour un patron existant.

Un patron X est une alternative d'un patron Y, si X a la même intention que Y mais propose une solution différente.

L'existence de l'alternative pourrait être remise en cause par le principe même des patrons : un patron propose une solution consensuelle et éprouvée pour un problème récurrent. Cependant, rien aujourd'hui ne permet de mesurer la qualité d'un patron. Actuellement, le statut de patron s'obtient d'une part parce que le problème posé est rencontré fréquemment et aussi parce que la solution donnée est judicieuse dans de nombreux cas [Gamma 95]. Les patrons n'ont pas de caractère définitif ; ils peuvent évoluer avec le temps ou disparaître. Il est donc judicieux d'utiliser l'alternative, d'autant plus qu'il existe rarement une solution unique à un problème donné. Tout patron peut apporter une alternative en proposant une nouvelle solution de conception ou d'implantation. Par exemple, M. Fowler [Fowler 97a] propose des **alternatives conceptuelles** au patron Rôle défini par P. Coad. De

même, l'article « State Patterns » [Dyson 96] présente sept patrons perçus comme des **alternatives d'implantation** du patron Etat.

La distinction entre alternative conceptuelle et alternative d'implantation n'est pas toujours évidente. On doit se demander si la nouvelle proposition remet ou non en cause la solution conceptuelle proposée par l'autre patron. Par exemple les patrons de P. Dyson ne remettent pas en cause la solution conceptuelle du patron Etat de Gamma, solution qui peut se résumer à la représentation d'un état par une classe. Nous donnons ci-dessous les intentions et les solutions offertes par trois des patrons Etat de P. Dyson qui constituent des alternatives d'implantation au patron Etat présenté au §3.3.1.3.

Nom	**Intention**	**Solution**
Etat donnée	Où placer les données membres ?	Si une donnée membre est utilisée dans un unique état, alors la placer dans cet état. On peut créer artificiellement une super-classe qui regroupe les données liées à plusieurs états. Si la donnée est indépendante des états, la placer dans l'objet "Contexte" (cf. Figure 3- 17).
Etat exposé	Comment limiter le nombre de méthodes spécifiques à un petit nombre des états de l'objet ?	Exposer directement l'état aux utilisateurs qui peuvent lui adresser directement leurs requêtes. Les états sont visibles pour l'utilisateur.
Transitions dirigées par les états	Comment réaliser les transitions ?	L'état assure lui-même la transition de lui-même vers un autre état. L'atomicité de la transition est donc garantie, l'objet Contexte n'en assurant plus le franchissement.

- *Extension & raffinement*

L'extension est une généralisation de l'intention d'un patron. Inversement, le raffinement est une restriction de l'intention d'un patron

Un patron X est une extension d'un patron Y, si X peut résoudre les problèmes posés par Y. Réciproquement, on dit aussi que Y est un raffinement de X.

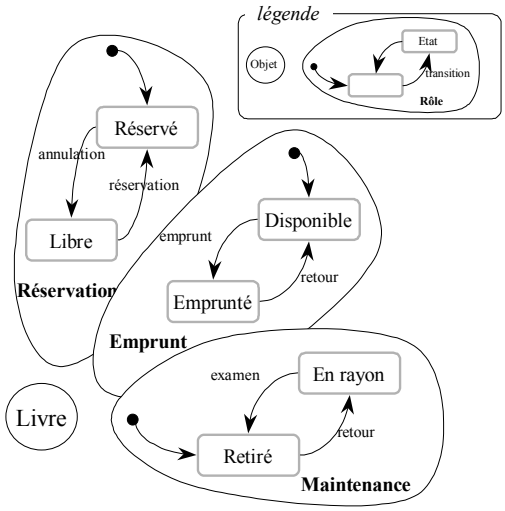
Nous illustrons ces deux opérateurs réciproques à partir des patrons Rôle de P. Coad et Etat de E. Gamma. Le patron Rôle permet de représenter les différents rôles d'un objet, l'évolution est limitée à la perte ou le gain de rôles par l'objet. Le patron Etat permet la gestion de l'évolution d'objets à l'aide de classes états mais ne prend en compte que des évolutions non concurrentes d'objets. C'est pourquoi nous proposons une extension de ces deux patrons sous

la forme d'un nouveau patron nommé Rôles-Dynamiques-d'Objet dont l'intention est de représenter et de gérer l'évolution des objets dans chacun de leurs rôles.

Patron Rôles-Dynamiques-d'Objet [SaintMarcel 99b]

Intention : Ce patron permet de représenter des objets multi-rôles pouvant adhérer ou perdre des rôles au cours de leur cycle de vie. Il établit la différence entre les rôles statiques et les rôles dynamiques. Dans le premier cas, on ne s'intéresse pas à l'évolution de l'objet dans le rôle (patron de P. Coad) alors que dans le second cas, le comportement de l'objet dépend à la fois du rôle joué et de l'état courant dans ce rôle.

Motivation : Dans certains cas, un objet intervient et évolue dans plusieurs processus. Il admet alors des comportements mais aussi des **évolutions différentes** en fonction du **rôle** joué dans ces processus. C'est ainsi qu'un objet livre d'un système d'information d'une bibliothèque joue un rôle et a une évolution différente dans les processus de prêt, de réservation et de maintenance.



Solution : La solution consiste à réaliser l'intégration (cf. § 3.3.2.1) des solutions offertes par les patrons Etat et Rôle. La classe RôleA correspond à un rôle statique dont la représentation de l'évolution des instances n'est pas requise ; la classe RôleB correspond à un rôle dynamique dont la représentation de l'évolution des instances est significative pour l'application. La classe RôleB regroupe l'union des propriétés d'une classe Rôle concrète du patron Rôle et d'une classe Contexte du patron Etat.

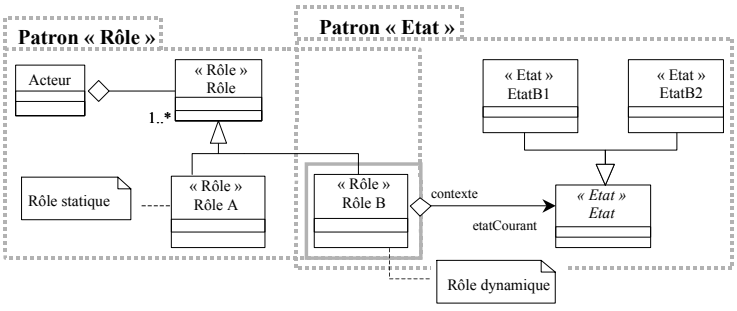


Figure 3- 23: Le patron " Rôles-Dynamiques-d'Objet "

La Figure 3- 24 reprend l'ensemble des patrons présentés ou référencés dans cette section et qui traitent du comportement d'objets pour les organiser dans un graphe de dépendances.

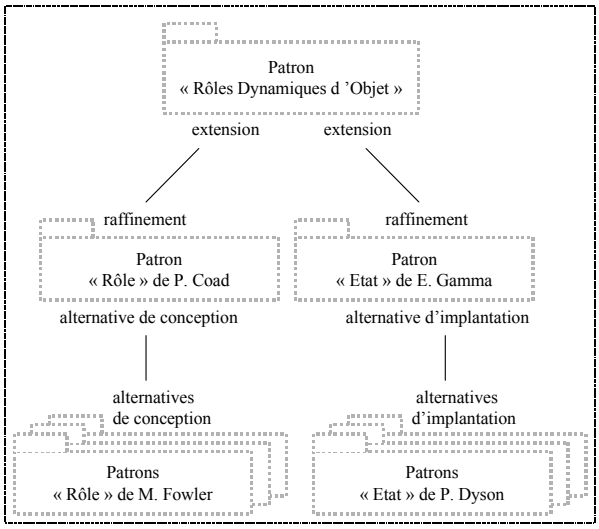


Figure 3- 24 : Patrons de comportement d'objets

Remarquons que le raffinement restreint l'intention du patron. Aucune hypothèse n'est faite sur la solution. Celle-ci peut être complètement différente de la solution initiale. Par analogie avec les concepts objets, on peut comparer ce type de raffinement à une surcharge de la solution du patron initial. Dans d'autres cas la solution du patron raffiné est obtenue à partir du patron qu'il raffine en utilisant les opérateurs introduits dans la section précédente : imitation, abstraction, intégration. Dans l'exemple précédent des patrons de comportement d'objets, le patron Rôles-Dynamiques-d'Objet est une **extension par intégration** des patrons Etat d'E. Gamma et Rôle de P. Coad (cf. Figure 3- 23). Le **raffinement par imitation** est largement utilisé lorsqu'on décide de donner le statut de patron à une imitation de patron existant. Supposons par exemple que l'on souhaite élever au rang de patron l'opération de création de compte (respectivement de retrait sur un compte). Le modèle Nouveau-Compte (Retrait-Compte) est alors repris, documenté par des rubriques intention, motivation, etc. afin de l'élever au rang de patron. Il constitue un raffinement par imitation du patron Nouvelle-Opération-Bancaire. Inversement l'**extension par abstraction** permet d'établir une abstraction d'un modèle existant afin de l'élever au rang de patron. C'est ainsi que le patron Collection, non présent dans le catalogue, peut être obtenu comme une extension par abstraction du patron Agence-Banque.

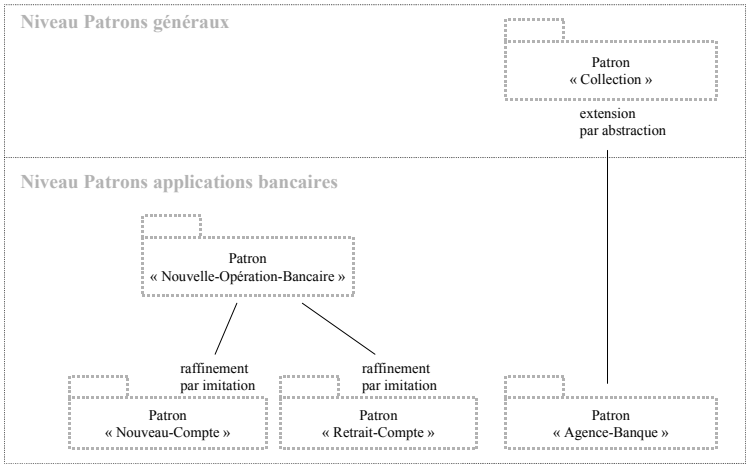


Figure 3- 25 : Patrons de gestion bancaire

- *Utilisation*

Un patron transforme un problème soit en solution soit en de nouveaux problèmes. Il utilise d'autres patrons lorsque ces derniers peuvent résoudre en partie ses problèmes.

Un patron X utilise un patron Y, si les problèmes posés par X peuvent être résolus en partie ou en totalité par Y.

Fréquemment, le problème posé par un patron ne peut être positionné simplement par rapport à un seul autre. La solution est alors donnée en décomposant le problème de manière à identifier des patrons existants. Le patron proposé utilise alors en quelque sorte d'autres patrons, ce qui est d'ailleurs une garantie de robustesse de la solution proposée. On retrouve cette idée dans le formalisme de Gamma où chaque patron déclare les patrons qui lui sont apparentés (Related Patterns), i.e. les patrons utilisés avec (ou par) celui-ci.

Historiquement, l'idée de représenter les relations d'utilisation entre patrons a été introduite par les langages de patrons [Beck 87, Coplien 96b, Fowler 97b, Front 99a]. Un langage de patrons est alors constitué de plusieurs patrons corrélés et souvent hiérarchisés pour guider le concepteur dans une démarche descendante partant de l'analyse globale d'une situation jusqu'à la conception détaillée d'une solution. Le langage SCalP [Front 97] représente une très bonne illustration du lien d'utilisation.

3.3.3. Bilan

Les patrons orientés objet peuvent être vus comme des mini-systèmes complets offrant un bon équilibre entre spécification et implantation, permettant de réduire la complexité des systèmes à objet et favorisant la réutilisation lors des différentes étapes de développement. De

même que le choix d'un langage de programmation et de ses concepts fondamentaux influencent fortement la construction et la structuration d'un logiciel, il est certain que la connaissance et l'usage de catalogues de patrons vont jouer un rôle important dans la manière d'appréhender, de formuler et de structurer les systèmes. Un catalogue de patrons dans une même équipe d'ingénierie permet de disposer d'un vocabulaire commun et d'un éventail de solutions consensuelles. Il offre, par analogie avec un langage de programmation, les termes d'un langage d'analyse et de conception. Il s'agit aujourd'hui d'offrir une « grammaire » donnant les bons usages d'utilisation et de combinaison de ces termes de manière à modéliser des systèmes dans leur globalité. Les opérations et les relations illustrées dans ce chapitre constituent une première approche destinée à identifier des opérateurs de base pour l'ingénierie des patrons (« for reuse ») et pour celle des systèmes à base de patrons (« by reuse »).

Il est certain que les prochains langages de patrons devront introduire des opérations et des relations inter-patrons de plus en plus formalisées afin de faciliter d'une part la tâche du concepteur et d'autre part d'assurer la traçabilité de la conception. De nouveaux ateliers de développement commencent à apparaître [Borne 99, Pagel 96]. Ils ont pour vocation d'automatiser la génération de code [Budinsky 96], d'offrir une aide à la conception [Meijer 96] ou de valider une conception [Krämer 96]. Ces travaux de recherche proposent généralement des représentations explicites des solutions génériques offertes par les patrons et une formalisation plus ou moins poussée du mécanisme d'instanciation (et non plus simplement d'imitation). De grands efforts restent aujourd'hui à faire pour mettre en œuvre efficacement des catalogues de patrons afin de proposer d'autres mécanismes, en particulier de sélection, de composition (et non plus simplement d'intégration) et d'héritage (et non plus simplement d'alternative et de raffinement).

Quelque soient les résultats de ces recherches, la notion de patron est déjà d'un intérêt pédagogique fondamental. Les patrons de conception orientée objet [Gamma 95, Buschmann 96, etc.] constituent une base solide pour l'enseignement de la conception orientée objet tout à fait similaire à celle offerte par les algorithmes classiques dans l'enseignement de la programmation.

3.4. Articles sur la réutilisation

Nous énumérons ci-dessous quelques articles révélateurs de ces travaux.

- [SaintMarcel 99a] – C. SAINT-MARCEL, Ph. MORAT, D. RIEU

Notion, Comportement, Rôle : trois concepts de modélisation pour une meilleure réutilisation des comportements,

2nd worshop on the Many Facets of Process Engineering, Gammarth,Tunisie, mai 1999.

Cet article présente les principaux concepts du modèle Notion-Comportement-Rôle.

- [Cauvet 98] – C. CAUVET, D. RIEU, B. ESPINASSE, J.P. GIRAUDIN, M. TOLLENAERE

Ingénierie des systèmes d'information produit : une approche méthodologique centrée réutilisation de patrons,

Congrès INFORSID, Montpellier, 1998.

Cet article montre l'usage des patrons pour le développement des SIP.

- [Gzara 99] – L. GZARA, D. RIEU, M. TOLLENAERE

Un référentiel générique de données techniques à des fins de réutilisation en ingénierie des systèmes d'information produit,

3ème congrès international de génie industriel : l'intégration des ressources humaines et technologiques : le défi, mai 1999, Montréal, Canada.

Cet article propose un référentiel de données pour les SIP. Ce référentiel, issu d'une analyse de domaine, a été utilisé pour spécifier des patrons de domaine.

- [Rieu 99] – D. RIEU, J.P. GIRAUDIN, A. FRONT-CONTE, C. SAINT-MARCEL

Des opérations pour les patrons de conception,

Congrès INFORSID, La Garde, 1999.

Cet article présente les opérations sur les patrons et les relations inter-patrons issues des différents projets de l'équipe.

- [Front 99b] – A. FRONT, J.P. GIRAUDIN, D. RIEU, C. SAINT-MARCEL

Réutilisation et patrons d'ingénierie,

dans l'ouvrage Génie Objet : analyse et conception de l'évolution, C. Oussalah, HERMES 1999.

Ce chapitre d'ouvrage constitue un état de l'art sur les approches à base de patrons.

- [Cauvet 00] – C. CAUVET, D. RIEU, A. FRONT, P. RAMADOUR

La réutilisation dans l'ingénierie des systèmes d'information

Dans l'ouvrage collectif Conception des SI, C. Rosenthal-Sabroux et C. Cauvet, HERMES 2000

Ce chapitre d'ouvrage constitue un état de l'art sur les modèles de composants, les techniques d'ingénierie des composants et celles d'ingénierie des SI par réutilisation.

Chapitre 4 : Perspectives

Tout projet de recherche devrait être défini (au moins a posteriori) comme un patron : un nom, une intention, une motivation, une solution, les conséquences de l'application de cette solution, les projets proches ou complémentaires… Mon propos n'est pas ici de débattre de généralités sur la recherche mais de tenter de décliner cette approche pour l'*ingénierie des systèmes d'information*, domaine qui constitue aujourd'hui mon cadre de référence, celui où j'encadre des jeunes chercheurs et je propose des thèmes de recherches.

Avant de préciser les orientations de ma thématique de recherche actuelle à savoir la *réutilisation dans l'ingénierie des SI* (§4.2*)*, je souhaite débattre de quelques points relatifs à nos démarches de recherche dans le domaine de l'*ingénierie des SI (§4.1)*.

4.1. Intention et Solution d'un projet de recherche en SI

4.1.1. L'intention d'une recherche en SI

De manière très générale, les travaux de recherche en systèmes d'information ont pour intention :

- de prendre en compte de nouveaux besoins ou de nouvelles techniques,
- d'améliorer la qualité des produits et la performance des processus utilisés pour les élaborer.

C'est ainsi que les méthodes à objet ont été définies pour prendre en compte la technologie objet et mettre en œuvre des architectures logicielles plus faciles à maintenir et à étendre.

Aujourd'hui, les travaux sur la réutilisation dans les SI ont pour objectifs :

- de répondre aux besoins actuels de développement,
- d'améliorer nos démarches de développement et ainsi garantir une meilleure traçabilité des activités d'ingénierie.

Tout projet en ingénierie des SI doit clairement être motivé par :
- des nouveaux besoins ou des nouvelles techniques à prendre en compte,
- des lacunes de modèles et de démarches existantes.

4.1.2. Solutions d'une recherche en SI

Les recherches en SI visent à proposer, spécifier, mettre en œuvre et expérimenter des modèles, des démarches et des outils dédiés aux activités d'ingénierie.

Les formalismes utilisés pour représenter les systèmes d'information sous différents aspects et à différents stades de leur cycle de développement sont basés sur un très petit nombre de modèles anciens et bien formalisés [Giraudin 00]. Citons entre autre le modèle Entité-Association de P. Chen [Chen 76], les modèles de flux de messages [Gane 79], les réseaux de Petri [Peterson 77] et les automates à états finis [Hopcroft 79].

Ces modèles ont été adaptés pour prendre en compte de nouveaux besoins et de nouvelles techniques. De telles adaptations sont inévitables et continueront à se faire, malgré les standards : « The UML provide a standard language for writing software blueprints, but it is not possible for one closed language to ever be sufficient to express all possible nuances of all model accross all domain across all time » [Booch 99].

Aujourd'hui, ces adaptations doivent être **formalisées** afin de ne plus tomber dans les problèmes rencontrés pendant des années avec les modèles objets : une multitude de propositions difficiles à évaluer et à comparer. Remarquons d'ailleurs que nous reproduisons aujourd'hui la même erreur avec les modèles de composants. La tentative réalisée dans la section 3.1.2.1 montre bien la difficulté de comparer des propositions non formalisées. En d'autre termes, il n'y a pas de raison en SI de manipuler des modèles « semi-formels » sous prétexte qu'ils sont graphiques et destinés à être compris par des utilisateurs (incompétents !).

La **méta-modélisation** (au sens modélisation de modèles cf. §2.1.3) constitue aujourd'hui une technique simple et efficace pour spécifier nos modèles, il suffit d'appliquer les contraintes que l'on impose aux utilisateurs de nos modèles. Cependant, la méta-modélisation n'a de sens que si l'on dispose d'un méta-modèle formalisé et adapté aux modèles à décrire.

Les travaux de l'OMG sont aujourd'hui suffisamment matures pour que nous disposions d'un nombre suffisant de modèles objets normalisés et consensuels permettant de représenter les SI mais aussi ces systèmes particuliers que sont les modèles, voire ces modèles particuliers que sont les méta-modèles. UML propose par exemple quelques mécanismes (les stéréotypes,

les « tagged values », les profils, etc.) permettant d'étendre et donc d'adapter le méta-modèle d'UML en fonction du domaine d'application visé et de la technologie sous-jacente utilisée. Ces mécanismes sont encore peu ou mal utilisés car difficiles à contrôler. Les extensions restent la plupart du temps syntaxiques, rendant ainsi vains les efforts qui ont été fait pour formaliser UML.

Je pense qu'actuellement une bonne solution consiste à :

- utiliser les « parties stables » du formalisme des diagrammes de classes (éviter par exemple l'usage de la composition, n'utiliser l'héritage que pour classifier les concepts, etc.)
- compléter les spécifications par des contraintes exprimées en OCL.

Une autre solution consiste à utiliser MOF (Méta Object Facility) comme méta-langage de modélisation [OMG/MOF 97]. MOF offre un sous-ensemble des concepts d'UML (cf. §2.1.3).

La même nécessité peut être évoquée pour les démarches de développement. La formalisation des processus de développement est bien entendue plus difficile que celle des modèles. Non pas que qu'il soit impossible d'appliquer à nos démarches les mêmes techniques de méta-modélisation, mais bien parce qu'il est extrêmement difficile de définir (même de manière informelle) une démarche complète de développement. La plupart du temps, seuls des fragments de processus sont suffisamment clarifiés et ont été suffisamment utilisés pour pouvoir être spécifiés.

Une voie qui nous semble particulièrement prometteuse est d'utiliser des patrons dont l'intention correspond à un problème de modélisation, de transformation de modèles produits, etc. C'est la voie suivie par Catalysis, c'est également l'approche que nous avons prise dans le projet NCR. Les patrons d'E. Gamma doivent d'ailleurs être perçus en ce sens, ils ne prétendent pas couvrir tout le cycle de conception des systèmes objets mais uniquement traiter quelques problèmes récurrents.

Toute proposition de modèles et/ou de démarches doit clairement être spécifiée. Utilisez la méta-modélisation si les solutions sont suffisamment matures, contentez vous d'un recueil pédagogique de conseils (par exemple sous la forme de patrons) dans le cas contraire.

4.2. Réutilisation dans l'ingénierie des SI : le projet PRIS

PRIS signifie *Pattern* & *Reuse* in Information Systems : *pattern* car c'est actuellement notre thème de recherche privilégié et *reuse* car il est évident que l'on ne peut pas ignorer les autres modèles de composants.

Ce projet est actuellement en cours de définition dans l'équipe SIGMA du laboratoire LSR dirigée par Jean-Pierre Giraudin.

4.2.1. Contexte du projet

De nombreuses enquêtes montrent que la pratique actuelle de l'ingénierie à base de composants repose sur des méthodes et des outils empiriques et qu'il existe aujourd'hui un réel besoin de rendre cette réutilisation plus systématique dans l'ingénierie des SI [Cauvet 00]. Le projet PRIS s'inscrit dans ce cadre.

Plus particulièrement, l'intention du projet PRIS concerne la spécification et la mise en œuvre d'environnements de développement par réutilisation. Dans un premier temps, l'accent sera particulièrement mis sur les approches à base de patrons.

Un environnement centré réutilisation de patrons nécessite le développement :

- d'un **système de réutilisation** centré sur l'identification, la spécification et l'organisation de patrons. L'objectif d'un tel système est de produire des bibliothèques organisées de patrons formalisés. La spécification des patrons ainsi que leur organisation doivent faciliter la mise en œuvre des opérations de base du système par réutilisation (sélection, adaptation, intégration).
- d'un **système par réutilisation** centré sur la sélection, l'adaptation et l'intégration des patrons. L'objectif d'un tel système est d'intégrer de manière systématique la réutilisation de patrons métiers et techniques dans le développement de SI.

Comme dans toute recherche en ingénierie nos axes de recherche sont à décliner en terme de modèles de patrons et de processus de développement. L'accent sera prioritairement mis sur les modèles de patrons (§4.2.2).

4.2.2. Modèles de patrons

Les formulaires utilisés aujourd'hui pour documenter des patrons ne permettent pas d'envisager une automatisation, ni même une réelle assistance des activités d'ingénierie des SI. Notre premier objectif sera de proposer une meilleure formalisation des patrons et des liens inter-patrons afin de permettre la spécification et la mise en œuvre des opérations que nous avons mis en évidence de manière empirique dans nos différents projets. La section 3.3.2 constitue donc la motivation de ce thème de recherche.

Plusieurs problèmes devront être abordés.

1. **Comment structurer les informations ?**

Les informations utiles pour sélectionner, adapter, composer et organiser des patrons sont **disséminées** dans de nombreuses rubriques. Citons à titre d'illustration certaines rubriques des patrons de conception d'E. Gamma.

- La rubrique « motivation » donne un exemple concret montrant l'utilité du patron et son application. En terme de *sélection*, cette rubrique permet au concepteur d'établir une analogie entre son problème et celui de l'exemple. En terme d'*adaptation*, elle est au moins aussi utile que la solution générale offerte par le patron.

- La rubrique « implantation » donne des astuces, des conseils et des techniques pour implanter le patron. Elle est donc fondamentale pour implanter la solution conceptuelle en fonction de divers compromis d'implantation (appelés *alternatives* d'implantation dans la section 3.3.2.2). Malheureusement elle intègre également des *raffinements* de patrons. C'est par exemple le cas du patron « composite » qui offre une modélisation générique des relations liant des objets composites (des figures composites, des répertoires, des produits, etc.) et des objets feuilles (des figures de base, des fichiers, des pièces de catalogues, etc). Le partage ou non de composants est vu comme un compromis d'implantation alors qu'il correspond à notre sens à un nouveau patron dont l'intention restreint celle du patron composite.

- La rubrique « voir aussi » indique les patrons susceptibles d'être utilisés avec celui décrit. Elle facilite ainsi l'*intégration* de patrons. De plus, de part le fait qu'elle indique les patrons dont l'intention est proche de celle du patron décrit, cette rubrique est également utile pour la *sélection* des patrons.

En premier lieu, il s'agit de réaliser une analyse critique des différentes rubriques proposées par les « formalismes » à base de patrons. L'idée consiste à voir une bibliothèque de patrons comme un système à modéliser.

- Une nouvelle structuration des patrons devra être proposée afin de regrouper les informations destinées au même usage.
- Des schémas conceptuels décrivant les relations inter-patrons devront donner la cartographie des bibliothèques de patrons.

2. Comment formaliser les informations ?

Les différentes rubriques des patrons sont souvent intégrées sous la forme de texte aux outils de mise en œuvre ; leur manque de formalisation ne permet pas d'automatiser leur usage. Seule la rubrique « solution » a fait l'objet de formalisation. La plupart des travaux proposent une représentation des solutions générales sous la forme de classes et de méta-classes permettant ainsi de tirer profit des mécanismes de base de l'approche objet (§ 3.1.2.3). Nous pensons que cette approche est à approfondir et à généraliser aux autres rubriques.

Il s'agit aujourd'hui de formaliser :

- la solution générale offerte par les patrons de manière à contrôler les mécanismes d'adaptation et d'intégration,

- les liens inter-patrons de manière à organiser des bibliothèques,

- le problème abordé par ce patron de manière à faciliter sa sélection.

Outre la mise à plat et la restructuration des formalismes à base de patrons, l'objectif sera donc de proposer des modèles de représentation des problèmes, des solutions et des liens inter-patrons permettant la spécification et la mise en œuvre d'opérations de sélection, d'adaptation et d'intégration.

Ces modèles pourront s'inspirer de techniques de représentation bien connues mais rarement utilisées dans la modélisation conceptuelle des SI. C'est ainsi que les notions de généricité et/ou de méta-classe nous semblent fondamentales pour exprimer les solutions mais aussi des problèmes génériques. Elles sont présentes dans les modèles d'UML mais leur pouvoir d'expression reste aujourd'hui plus faible que dans les langages de programmation ou dans les systèmes de représentation de connaissances (cf. les travaux sur Shood et Meninge aux section 2.3 et 2.4).

Plus généralement et dépassant le simple cadre des modèles de patrons, l'objectif est donc de mener une réflexion de fond sur l'expression de connaissances génériques dans les modèles conceptuels. Le concept de généricité n'est pas à prendre ici dans le sens restrictif des langages de programmation. Il s'agit de proposer d'autres techniques dédiées à l'abstraction et à la réutilisation. Les travaux entrepris dans NCR (§ 3.2) vont en ce sens en proposant une évolution de la sémantique des diagrammes d'états d'UML afin de spécifier des comportements abstraits réutilisables.

Il s'agit de proposer des modèles et des langages adaptés à la représentation conceptuelle des connaissances génériques.

De plus, nous sommes aujourd'hui convaincus de la nécessité de conserver et de maintenir la cohérence entre différentes représentations des patrons. La langue naturelle devra cohabiter avec des représentations semi-formelles et formelles afin de conserver aux patrons leur caractère pédagogique tout en facilitant leur adaptation et la testabilité de ces adaptations.

Les travaux sur la modélisation suivant plusieurs formalismes ont été initialisés dans l'équipe par J. Freire [Freire 97] et S. Dupuy [Dupuy 97]. Ces travaux constituent des points de départ précieux pour ce projet.

Il s'agit de maintenir la cohérence entre les représentations textuelles, graphiques et formelles des problèmes, solutions et relations inter-patrons.

4.2.3 Ouverture

Le premier objectif du projet PRIS est donc de proposer de nouveaux modèles adaptés à la définition, à l'organisation et à l'utilisation des patrons. Des bibliothèques de patrons formalisées seront associées à des opérations de sélection, d'adaptation et de composition. Ces bibliothèques et ces opérations permettront de définir des environnements centrés sur la manipulation de composants. Néanmoins, nous avons conscience que ce type d'approche bien qu'incontournable reste insuffisante au niveau de l'aide fournie durant le processus de réutilisation.

Un environnement de développement par réutilisation doit également supporter le processus de réutilisation. Un tel processus est nécessaire pour **guider** la sélection de composants, leur adaptation et leur composition. Cette approche consistant à mettre en œuvre des processus de développement entièrement basés sur la réutilisation de bibliothèques de patrons métiers et techniques constituera le deuxième axe de travail du projet.

Bibliographie

Abiteboul 91 S.ABITEBOUL, A. BONNER - Objects and Views, In proc. ACM -Sigmod, pp238-247, 1991.

Acm 96 ACM – Software Patterns, Communications of the ACM, 39 (10), 1996.

Ahmed-Nacer 94 M. AHMED-NACER - Un modèle de gestion et d'évolution de schéma, Thèse de l'INPG de Grenoble, mars 1994.

Alexander 77 C. ALEXANDER, S. ISHIKAWA et al. - A Pattern Language, Oxford University Press, New York, NY, 1977.

Alexander 79 C. ALEXANDER - The Timeless Way of Building, Oxford University Press, New York, NY, 1979.

Andrews 90 T. ANDREWS, C. HARRIS, J. DUHL - The Ontos Object Database, Rapport Technique, Ontologic Inc., Burlington, MA 01803, 1990.

Andro 98 T. ANDRO, J.M. CHAUVET – Objets métiers, Eyrolles 1998.

Appleton 97 B. APPLETON - Patterns and Software, Essential Concept and Terminology,

http://www.enteract.com/~bradapp/docs/patterns-intro.html, 1997.

Arbouy 94 S. ARBOUY, A. BEZOS & al. *STEP : Concepts fondamentaux*. AFNOR 1994.

Asplund 73 K. ASPLUND, G. SWIFT - Downstream Water Volume,

http://www.designmatrix.com/pl/ecopl/dswv.html, 1973.

Badouel 93 D. BADOUEL, A. KHALED - La programmation C et C++, Hermès, 1993.

Bailin 89 S.C. BAILIN S.C. et J.M. MOORE, The KAPTUR Environment : An Operations Concept, Report prepared for NASA Goddard Space Flight Center, CTA Incorporated, Tockville, 1989.

Banerjee 87 J. BANERJEE, W. KIM, K.J. KIM, H. KORTH - Semantics and implementations of schema evolution in object-oriented databases, Proc. ACM SIGMOD, Conference. San Fransisco, mai 1987.

Barbier 94 F. BARBIER - Traceability in object-oriented software life cycle, TOOLS EUROPE 94, Versailles, mars 1994.

Barbier 99 F. BARBIER, H. BRIAND, B. DANO - Evolutions d'objets au cours du processus de développement logiciel, dans l'ouvrage Génie Objet : Analyse et Conception de l'évolution, C. Oussalah, HERMES, 1999.

Beck 87 K. BECK, W. CUNNIMGHAM - Using Pattern Languages for Object-Oriented Programs, Proceedings of OOPSLA-87 (1987).

Beck 94 K. BECK, R. JOHNSON - Patterns Generate Architectures, Proceedings of ECOOP'94 (1994).

Bertino 92 E. BERTINO - A view mechanism for Object-Oriented Database, 3rd International Conf. on Extending Database Technology, 1992.

Bezivin 99 J. BEZIVIN - Modèle et méta-modèles, comme entités de première classe, dans le processus de développement du logiciel, Congrès INFORSID, La Garde 1999,

Blain 94 G. BLAIN, N. REVAULT, H.A. SAHRAOUI, J.F. PERROT - A Meta Modelization Technique, OOPSLA 94, Worshop on AI and OO Software Engineering, Portland, octobre 1994.

Bobrow 88 D.G. BOBROW, L.G. DEMICHEL & al. – Common Lisp Object System Specification, X3J13 Document 88-002R, juin 1988.

Booch 97a G. BOOCH - Des solutions objets, Gérer les projets orientés objets, International Thomson Publishing, 1997.

Booch 97b G. BOOCH, I. JACOBSON, and al. - Unified Modeling Language, version 1.1, 1997, Rational Software Corporation, http://www.rational.com.

Booch 99 G. BOOCH, I. JACOBSON, J. RUMBAUGH - The Unified Modeling Language User Guide, Editor Addison Wesley, 1999.

Borne 99 I. BORNE, N. REVAULT - Comparaison d'outils de mise en œuvre de design patterns ", Numéro spécial de la revue L'OBJET : Patrons Orientés Objet, Hermès, 1999.

Bosch 96 J. BOSCH - Language support for design patterns, in Tools Europe'96, Prentice Hall, 1996

Bounaas 95 F. BOUNAAS – Gestion de l'évolution dans les bases de connaissances : une approche par les règles, thèse de doctorat en informatique de l'INPG, Grenoble, octobre 1995.

Bounaas 97a F. BOUNAAS, M. CHABRE-PECCOUD, P-Y CUNIN, J.P. GIRAUDIN, P. MORAT, D. RIEU - Chapitre 5 : Objets et Méta-modélisation, Dans l'ouvrage collectif : Ingénierie objet : concepts et techniques, Editeur C. Oussalah, InterEditions, avril, 1997.

Bounaas 97b F. BOUNAAS, D. RIEU - Réutilisation pour l'ingénierie des modèles, First worshop on the Many Facets of Process Engineering, Gammarth,Tunisie, septembre, 1997.

Bouzeghoub 95 M. BOUZEGHOUB, G. GARDARIN, P. VALDURIEZ - DU C++ À MERISE OBJET, Editions Eyrolles, 1995

Budinsky 96 F.J. BUDINSKY, M.A. FINNIE, J.M. VLISSIDES, P.S. YU - Automatic Code Generation from Design Patterns, IBM Systems Journal, vol. 35, n°2, 1996.

Buschmann 96 F. BUSCHMANN, R. MEUNIER, H. ROHNERT, P. SOMMERLAD, M. STAL – Pattern-Oriented Software Architecture, A System of Patterns, Wiley and Sons Ltd., (1996).

Cailloce 97 P. CAILLOCE - Modélisation graphique dans Meninge : conception, intégration et interprétation, Mémoire d'ingénieur CNAM, Grenoble, mars 1997.

Cargill 92 T. CARGILL - C++ Programming Style, Addison-Wesley, 1992.

Carré 89 B. CARRE - Méthodologie Orientée Objet pour le représentation des connaissances concept de point de vue, de représentation multiple et évolutive d'objet thèse troisième cycle, Univ. de Lille, 1989.

Carré 90 B. CARRE, L. DEKKER, J.M. GEIB – Multiple and evolutive representation in the Rome Language, in Proc. TOOL2, Paris, 1990.

Casais 89 E. CASAIS – Managing Evolution in Object Oriented Environments : an Algorithmic approach, PhD Thesis, University of Geneva, 1991.

Castano 94 S. CASTANO, V. DE ANTONELLIS, C. FRANCALANCI, B. PERNICI B., A reusability-based comparison of requirement specification methodologies, in Methods and Associated Tools for the Information Systems Life Cycle (A-55), A. A. Verrijn-Stuart and T. W. Olle (Editors), Elsevier Science B. V. (North-Holland), 1994.

Cauvet 98 C. CAUVET, D. RIEU, B. ESPINASSE, J.P GIRAUDIN, M. TOLLENAERE - Ingénierie des systèmes d'information produit : une approche méthodologique centrée réutilisation de patrons, Congrès INFORSID, Montpellier, 1998.

Cauvet 00 C. CAUVET, D. RIEU, A. FRONT-CONTE, P. RAMADOUR – Réutilisation dans l'ingénierie des systèmes d'informations, dans l'ouvrage collectif Conception des SI, C. Rosenthal-Sabroux et C. Cauvet, HERMES 2000.

Chen 76 P.P.S CHEN – The Entity-Relationship Model, toward a unified view of data, ACM transactions on Database Systems, vol 1, n°1, 1976

Chikhi 99 Y. CHIKHI, F. BOUNAAS, J. MOUSTAFIADES - Patterns pour la conception de réseaux électriques ", Numéro spécial de la revue L'OBJET : Patrons Orientés Objet, Hermès, 1999.

Cholvy 83 L. CHOLVY - Structuration et intégrité des informations dans les BD pour la CAO. Définition d'un modèle de données et réalisation d'une maquette, Thèse de l'ENSA, Toulouse, 1983.

Coad 92 P. COAD - Object-Oriented Patterns, Communication of ACM, vol. 35, n°9, 1992.

Coad 96 P. COAD et al. – Object Models – Strategies, Patterns and Applications, Yourdon Press Computing Series, (1996).

Cointe 87 P. COINTE – Metaclass are first class : the objvlisp Model, Proceedings OOPSLA, Orlando, 1987.

Collet 94 C.COLLET, P. HABRAKEN, T.COUPAYE, M. ADIBA – Active Rules for the software engineering platform GOODSTEP, in Proc. Of the 2nd International Workshop on Database and Software Engineering, Sorrento, Italy, mai 1994.

Collongues 86 A. COLLONGUES, J. HUGUES, B. LAROCHE – Merise, 1. Méthode de conception, Dunod Informatique, 1986.

Cook 94 S. COOK and J. DANIELS - designing object systems : object oriented modelling with SYNTROPY, Prentice Hall New York 1994.

Copeland 84 G. COPELAND, D. MAIER - Making Smalltalk a Database System, International Conference ACM SIGMOD, juin 1984.

Coplien 92 J.O. COPLIEN - Advanced C++ Programming Styles and Idioms. Addison-Wesley, 1992.

Coplien 96a J.O. COPLIEN - Software design patterns : common questions and answers. ftp://st.cs.uiuc.edu/pub/patterns/papers/PatQandA.ps.

Coplien 96b J.O. COPLIEN - Patterns, The Patterns White Paper. ISBN 1-884842-50-X, 1996.

Corriveau 96 J.P CORRIVEAU - Traceability Process for large OO Projects, IEEE Computer, septembre 1996.

Dahl 66 O. DAHL, K. NYGAARD - Simula - an Algol-based Simulation Language, Communications of the ACM, 1966.

Dayal 88 U. DAYAL, M. HSU, P. GRAY. – Rules are objects too : a knowledge model for an active OODBMS, In proc Advances in Object-Oriented Database System, september 88.

Dekker 92 L. DEKKER, B.CARRE - Multiple and dynamic Representaion of frames with points of views in FROME, conf. Représentation Par Objet, La grande Motte, juin 92

Delcourt 92 C. DELCOURT – Evolution de schémas dans un système de bases de données orienté-objet, thèse de l'Universite d'Orsay, 1992.

Delobel 91 C. DELOBEL, C. LECLUSE, P. RICHARD - Bases de Données : des Systèmes Relationnels aux Systèmes à Objets, InterEditions, 1991.

De Mey 94 V. DE MEY – Visual Composition of Software Composition, Thèse de l'Université de Genève, 1994

Demichel 87 L.G DEMICHIEL et al. - The Common Lisp Object System : an overview, proc. ECCOP, Paris, 1987

Desfray 98 P. DESFRAY – Automation of design patterns : concepts, tools and practices, actes du congrès « UML'98, Beyond the notation », Mulhouse, 3-4 juin 98.

Djeraba 93 C. DJERABA – Quelques liens sémantiques dans un systèmes à Bases de Connaissances, Doctorat de l'Université C. Bernard, Lyon, 1993.

D'Souza 96 F. D'SOUZA - Refine : A Pattern Language for Catalysis, Proceedings of PloP'96 (1996).

D'Souza 98 F. D'SOUZA, A. C. WILLS - Objects, Components and Frameworks with UML : the CATALYSIS approach, Editions Addison-Wesley, 1998.

Dugertil 88 P. DUGERDIL – Contribution à l'étude de la représentation des connaissances fondée sur les objets. Le langage ObjLog, Thèse de doctorat de l'Univ. d'Aix-Marseille, 1988.

Dupuy 97 S. DUPUY, Y. LEDRU, M. CHABRE-PECCOUD, Integrating OMT and Object-Z, workshop on « Making Object Oriented Methods More Rigourous », London, June 1997.

Doumeingts 90 G. DOUMEINGTS - Méthodes pour cencevoir et spécifier les systèmes de production. Actes CIM 90, Bordeaux, juin 1990.

Dyson 96 P. DYSON, B. ANDERSON - State Patterns, Proceedings of EuroPLoP'96 (1996).

Eden 97 A. EDEN, J. GIL, A. YEHUDAI – Automating the application of design patterns, JOOP, 1997.

Escamilla 90 J. ESCAMILLA, V. FAVIER, P. JEAN, NGUYEN G.T, D. RIEU - Représentation de Connaissances Dynamiques dans SHERPA, congrès INFORSID 90, Biarritz, Mai 1990, Egalement Rapport de Recherche INRIA, n° 1208, avril 1990

Escamilla 93 J. ESCAMILLA - SHOOD : un modèle méta-circulaire de représentation de connaissances, Doctorat de l'INPG, Grenoble, septembre 1993

Espinasse 97 B. ESPINASSE, D. NANCI - Merise et l'approche orientée objet : du couplage avec OMT à une troisième génération, Revue Ingénierie des systèmes d'information, Hermes, Vol 5, N° 4, Octobre 1997.

Falcone 99 P. FALCONE SAMPAIO, N. W. PATON - A deductive extension for ODMG compliant object databases, revue l'OBJET, "Best of OOIS'98, volume 5 n° 1/1999.

Falkenhainer 89 B. FALKENHAINER, K. FORBUS, D. GENTNER - The Structure-Mapping Engine: Algorithm and Examples, Artificial Intelligence 41, pp 1-63, 1989.

Fauvet 87 M.C. FAUVET, D. RIEU - CADB: un système de gestion de bases de données et de connaissances pour la CAO, Proc. MICAD'87. Paris, février 1987.

Fauvet 88 M.C. FAUVET - Un SGBD pour la CAO dans un environnement partagé, Doctorat de l'Université Joseph Fourier, Grenoble, décembre 1988

Favier 90 V. FAVIER, D. RIEU - Dynamique dans les Bases de Connaissances Projet Sherpa, Revue " Modèles et Bases de Données ", février 1990.

Ferber 84 J. FERBER – Quelques aspects du caractère réflexif du langage Mering, Proc. JLOO, Brest, 1984.

Finkelstein 94 A. FINKELSTEIN, J. KRAMER and al. - Software Process Modelling and Technology, John Wiley, 1994.

Fowler 97a Fowler M., " Role Patterns ", *Proceedings of PLoP'97* (1997).

Fowler 97b M. FOWLER – Analysis Patterns – Reusable Object Models, Addison-Wesley, (1997).

Freire 95 J. C. FREIRE JUNIOR - Analyse de la puissance d'expression des modèles orientés objet.Proc. INFORSID 95, Grenoble, Mai 95.

Freire 97 J. FREIRE JUNIOR- Ingénierie des SI : une approche de méta-modélisation et de multi-modélisation, Thèse de doctorat, UJF, Grenoble, Juillet 1997.

Front 97 A. FRONT - Développement de systèmes d'information à l'aide de patrons, Application aux bases de données actives, Thèse de l'Université Joseph Fourier, Grenoble, 1997.

Front 99a A. FRONT-CONTE - SCalP : un langage de patrons pour exprimer les aspects réactifs des systèmes d'information, Numéro spécial de la revue L'OBJET : Patrons Orientés Objet, Hermès, 1999.

Front 99b A. FRONT-CONTE, J.P. GIRAUDIN, D. RIEU, C. SAINT-MARCEL - Réutilisation et patrons d'ingénierie, dans l'ouvrage Génie Objet : Analyse et Conception de l'évolution, C.Oussalah, HERMES 1999.

Gamma 95 E. GAMMA, R. HELM, R. JOHNSON, J. VLISSIDES – Design Patterns, Elements of reusable Object-Oriented Software, Addison-Wesley Publishing Company (1995).

Gane 79 C. GANE, T. SARSON – Structured Systems Analysis, Prentice Hall, 1979.

Genesereth 92 M.R. GENESERETH, R.E. FIKES - Knowedge Interchange Format, Version 3.0, Reference Manual, Technical Report Logic-92-1, Computer Science Department, Stanford University.

Gentner 83 D. GENTNER - Structure Mapping: A Theoretical Framework for Analogy, Cognitive Science 5, 1983.

Girard 95 P. GIRARD - Gestion hypothétique d'objets complexes, thèse UJF, Grenoble, octobre 1991.

Giraudin 00 J.P. GIRAUDIN, M. CHABRE, D. RIEU, C. SAINT MARCEL – Modèles de spécification pour l'ingénierie des SI, dans l'ouvrage collectif Conception des SI, C. Rosenthal-Sabroux et C. Cauvet, HERMES 2000.

Godin 95 R. GODIN, G.W. MINEAU, R. MISSAOUI - Incremental structuring of knowledge bases. In Proceedings of the International Knowledge Retrieval, Use, and Storage for Efficiency Symposium (KRUSE'95), Santa Cruz: Springer-Verlag's Lecture Notes in Artificial Intelligence, pp. 179-198.

Goldberg 83 A. GOLDBERG, D. ROBSON - SmallTalk-80 : the language and its implementation, Addison-Wesley, Publishing Company, 1983.

Goldberg 84 A. GOLDBERG - SmallTalk-80, The Interactive Programming Environment, Addison-Wesley, 1984.

Grabowski 90 H. GRABOWSKI, H. SCHAFER, S. BRIGE - CAD/CAM Integration. Actes congrès CIM 90, Bordeaux, juin 1990.

Grosz 96 G. GROSZ, S. SI-SAID, C. ROLLAND - MENTOR : un environnement pour l'ingénierie des méthodes et des besoins, congrès INFORSID, Bordeaux, juin 1996.

Gruber 93 T.R. GRUBER – Toward principles for the design of ontologies used for knowledge sharing, in Formal Ontology in Conceptual Analysis and Knowledge Representation, edited by N. Guarino, Kluwer Academic Publishers, 1993.

Gzara 99 L. GZARA, D. RIEU, M. TOLLENAERE - Un référentiel générique de données techniques à des fins de réutilisation en ingénierie des systèmes d'information produit, 3ème congrès international de génie industriel : l'intégration des ressources humaines et technologiques : le défi, mai 1999, Montréal, Canada.

Hainault 92 J.L HAINAUT and al. - TRAMIS : a Transformation-Based Database CASE Tool, in Proceeding of the 5th International Conference on Software Engineering and its applications, Toulouse, décembre 1992.

Harandi 93 M. HARANDI - The Role Of Analogy In Software Reuse, ACM-SAC '93/2/93/IN, USA.

Harani 97 Y. HARANI - Une approche multi-modèles pour la capitalisation des connaissances dans le domaine de la conception, thèse de doctorat de l'Université de Metz, novembre 1997.

Harel 87 D. HAREL – Statecharts : a visual formalism for complex systems, Science of Computer Programming pp. 231-274 – 1987.

Harel 96 D. HAREL, E. GERY - Executable object modeling with statecharts, Proceedings 18 th Int. Conf. Soft. Eng. IEEE Press 1996 pp.246-257.

Haton 91 J.P. HATON, N. BOUZID, F. CHARPILLET & al. - Le raisonnement en intelligence artificielle, Interéditions, IIA, 1991.

Heiler 90 S.HEILER, S, ZDONIK, Object Views : Extending the vision, in IEEE Data Engineering Conference, Los Angeles, 1990.

Heitz 87 M. HEITZ - HOOD : une Méthode de Conception Hiérarchisée Orientée Objets pour le Développement de Gros Logiciels Techniques et Temps-réels, Revue Bigre n°57, 1987.

Hill 93 D. R. C. HILL - Analyse Orientée Objets & modélisation par simulation, éditions Addison-Wesley, 1993.

Holyoak 89 K. HOLYOAK, P. THAGARD - Analogical Mapping by Constraint Satisfaction, Cognitive Science, pp 295-355, 1989.

Hopcroft 79 J.E. HOPCROFT, J.D. ULLMAN – Introduction to automata theory, languages and computation, Addison Wesley, 1979.

Jacobson 92 I. JACOBSON, M. CHRISTERSON, P. JONSSON, G. OVERGAARD, Object-Oriented Software Engineering : A Use Case Driver Approac, Addison Wesley, 1992.

Jaczynski 99 M. JACZYNSKI, B. TROUSSE - Patrons de conception dans la modélisation d'une plate forme à objets pour le raisonnement à partir de cas, Numéro spécial de la revue L'OBJET : Patrons Orientés Objet, Hermès, 1999.

Jaunin 96 B. JAUNIN - Les patrons et les langages de patrons, Université de Nantes, http://www.unantes.univ-nantes.fr/~jaunin/patterns, 1996.

Jézéquel 99 J-M. JEZEQUEL, S. LORCY, N. PLOUZEAU - Un patron pour la gestion de la qualité de service d'applications réparties, Numéro spécial de la revue L'OBJET : Patrons Orientés Objet, Hermès, 1999.

Johnson 88 R.E. JOHNSON, B. FOOTE - Designing reusable classes, Journal of Object-Oriented Programming, vol. 1, n°2, 1988.

Johnson 92 R.E. JOHNSON - Documenting frameworks using patterns, in Proceedings of OOPSLA'92 (1992), ACM Press.

Kang 90 K.C. KANG, S.G. COHEN, J.A. HESS, W.E. NOVAK - Feature-Oriented Domain Analysis (FODA), Feasible study. Carnegie-Mellon University, Pittsburg, PA, Technical Repport, 1990.

Karlsson 95 E.A. KARLSON - Software Reuse : A Holistic Approach, John Wiley & Sons, Wiley Series in Software Based Systems, 1995.

Katz 90 R.H. KATZ - Toward a unified framework fot version modelling in engineering databases, In ACM Computing Surveys, Vol.22, n°4, 375-408, 1990.

Keefer 94 P. KEEFER – The accounts framework, Master Thesis, University of Illinois, 1994.

Kelly 96 S. KELLY, K. LYYTINEN, M. ROSSI - MetaEdit+ A Fully Configurable Multi-user and Multi-Tool CASE and CAME Environment, in Proc. 8th International Conference CAISE'96,

Kiernan 89 J. KIERNAN, C. DE MAINDREVILLE, E . SIMON - The design and Implementation of an Extensible Deductive Database System, SIGMOD Records, septembre 1989.

Kim 88 W. KIM, J. BANERJEE and all. - Composite Object Support in an Object-Oriented Database System, International Conference ACM Sigmod, 1988.

Kim 90 W. KIM, F. GARZA ; N. BALLOU and all. - Architecture of the Orion next-generation Database System, IEEE Transaction on Knowledge and Data Engineering, mars 1990.

Krämer 96 C. KRAMER, L. PRECHELT - Design recovery by automated search for structural design patterns in objet-oriented software. In Working Conference on Reverse Engineering (WCRE 96), IEEE CS Press 1996.

Kuper 84 G. KUPER, M. VARDI - A new approach to Database Logic, International Conference PODS, 1984.

Ladet 95 P. LADET, F. VERNADAT - The dimensions of Integrated Manufacturing Systems Engineering. Integrated Manufacturing Systems Engineering, Ed. P. Ladet, F. Vernadat, Chapman & Hall, 1995.

Lea 97 D. LEA - Patterns Discussion FAQ, http://g.oswego.edu/pd-FAQ.html, 1997.

Legrand 97 A. LE GRAND - Spécialisation de cycle de vie d'objet : une introduction. Journées BDA'97, Grenoble, septembre 1997

Lerner 90 B.S. LERNER, A.N. HABERMANN – Beyond schema evolution to database reorganisation, in proc ECOOP, octobre 1990.

Lieberherr 89 K.J. LIEBERHERR, I. HOLLAND – Formulations and benefits of the law of Demeter. SIGPLAN Notices ACM, vol 24, N°3, mars 89.

Liotard 93 M.P. LIOTARD - Mécanisme de classification pour un système de représentation de connaissances, mémoire d'ingénieur CNAM, Grenoble, mars 1993.

Lonchamp 92 J. LONCHAMP, C. GODART, J.C. DERNIAME - Les environnements intégrés de production de logiciel, dans TSI, vol.11, n. 1/1992, p. 31 à 95.

Maiden 92 N. MAIDEN, A. SUTCLIFFE – Exploiting Reusable Specification Trough Analogy, Communications of the ACM, 34(5), avril 1992.

Maiden 93 N. MAIDEN, A. SUTCLIFFE - The Domain Theory: Object System Definition, Nature Report CU-93-OOA, 1993.

Marino 91 O. MARINO - Classification d'objets composites dans un système de représentation de connaissances multi-points de vue, 8ième congrès RFIA, Lyon, novembre 1991.

Marino 93 O. MARINO – Raisonnement classificatoire dans une représentation à objets multi-points de vue, Thèse UJF, Grenoble, octobre 93.

Martin 97 C. MARTIN - Modélisation des opérations dans Meninge : conception, réification, exécution, Mémoire d'ingénieur CNAM, Grenoble, mars 1997.

Masini 89 G. MASINI, A. NAPOLI, D. COLNET, and all - Les langages à Objets, Interéditions, 1989.

Mattson 95 M. MATTSON - Object-Oriented Frameworks, A survey of methodological issues, Thèse Department of Computer Science, Lund University, Coden, 1995.

Mega 95 MEGA 4.1 – Mega International, Janvier 95.

Meijer 96 M. MEIJER - Tool support for object-oriented design patterns. Masters's thesis INF-SCR-96-238, Utrecht University, The Netherlands, 1996.

Meijler 97 T.D. MEIJLER, S. DEMEYER, R. ENGEL - Making design patterns explicit in face ", Proceedings of European Software Engineering Conference, ESEC/FSE 97 (1997), pp. 94-110.

Melo 93 W. LOUZADA MARTINS MELO - TEMPO: un environnement de développement logiciel centrés procédés de fabrication, Doctorat de l'Univ. Joseph Fourier, Grenoble, octobre 93

Meyer 87 B. MEYER, J-M. NELSON, M. MATSUO - Eiffel : Object-Oriented Design for Software Engineering ", Proceedings of ESOP'87 (1987), pp. 327-245.

Meyer 88 B. MEYER - Object-Oriented Software Construction. Prentice Hall, 1988.

Meyer 92 B. MEYER - Design by contract, IEEE Computer, vol. 25, n°10, pp. 40-51, 1992.

Mondemé 96 L. MONDEME - Informatique de gestion : les sept couches de la réutilisation, Revue Génie Logiciel, n°42, pp. 40-44, 1996.

Morejon 91 J. MOREJON and al. - GraphOR : a Meta Design TOOL; Entity-Relationship Approach, the core of conceptual modeling, North Holland, 1991.

Morel 96 J-M. MOREL - Expériences de réutilisation avec la méthode REBOOT, Revue Génie Logiciel, n°42, pp. 45-50, 1996.

Muller 97 P.A. MULLER – Modélisation objet avec UML, Editions Eyrolles, Avril 1997

Napoli 92 A. NAPOLI – Représentations à objets et raisonnement par classification en Intelligence Artificielle, Thèse de Doctorat d'Etat, Université de Nancy 1, janvier 1992.

Neighbors 89 J.M. NEIGHBORS - DRACO : A method for engineering reusable systems, in Software Reusability, Volume 1: concepts and models, T.J. Biggerstaff, A.J. Perlis (Eds), Addison-Wesley, 1989.

Nerson 92 J.M. NERSON – Applying object-oriented analysis and design, communications of the ACM, vol. 35 n°9, septembre 1992.

Nguyen 87a G.T. NGUYEN, D. RIEU - Expert database support for consistent dynamic objects, 13th International Conference on Very Large Data Bases, Brighton (England), septembre 1987.

Nguyen 87b G.T. NGUYEN, RIEU D - Manipulation d'objets dynamiques dans les bases de données, Journées AFCET "Des bases de données aux bases de connaissances", Sophia Antipolis, septembre 1987.

Nguyen 88a G.T NGUYEN, D.RIEU - Dynamic schemas for engineering databases, Proc. 4th International Conference on Systems Research, Informatics and Cybernetics. Baden-Baden (RFA), août 1988.

Nguyen 88b G.T NGUYEN, D.RIEU - Schema evolution in object-oriented database systems. "Data & Knowledge Engineering", North Holland, Vol 4, n° 1, juillet 1989.

Nguyen 88c G.T. NGUYEN, D. RIEU - Heuristic control on dynamic database object, IFIP Conference "The role of artificial intelligence in databases and information systems", Guangzhou (R.P Chine), juillet 1988.

Nguyen 89a G.T NGUYEN, D. RIEU - Schema change propagation in object-oriented databases, XIth World Computer Congress, IFIP Congress '89. San Francisco (USA), août 1989.

Nguyen 89b G.T. NGUYEN, D. RIEU - Schema evolution in object-oriented database systems, Data & Knowledge Engineering, North Holland. Vol 4, n° 1, juillet 1989.

Nguyen 92 G.T. NGUYEN, D. RIEU - Shood : a design object model, proc. 2nd Intl. Conf. Artificial Intelligence in Design, Carnegie Mellon Univ., Pittsburgh, juin 1992.

Nguyen 97 G.T. NGUYEN – chapitre 6. Objets et Evolution dans l'ouvrage collectif Ingénierie Objet, Concepts et Techniques, C. Oussalah, InterEditions, 1997.

Objet 98 Les Représentations par objets en conception – Numéro spécial de la revue l' OBJET, coordonnateurs : M. Oussalah, B. Trousse, éditions Hermes, 1998.

Objet 99 Patrons Orientés Objet - Numéro spécial de la revue l' OBJET, coordonnateurs : D. Rieu, J.P. Giraudin, éditions Hermes, 1999.

O'Connor 94 J. O'CONNOR , C. MANSOUR, J. TURNER-HARRIS, G.H. CAMPBELL - Reuse in Command-and-Control Systems, IEEE Software, September 1994.

OMG/MOF 97 OMG/MOF – Meta Object Facility (MOF) Specification. AD/97-08-14, Object Management Group, Septembre 97.

Oquendo 90 F. OQUENDO, J.P. ZUCKE - Support for software tool integration and process-centered software engineering environments, 3ième journées du Génie Logiciel et ses applications, décembre 1990

Oivo 92 M. OIVO, V.R. BASILI - Representing software engineering models : the TAME goal oriented approach, IEEE Transactions on Software Engineering, 1992

Oussalah 97 M. OUSSALAH - Ingénierie Objet – Concepts, techniques et méthodes, InterEditions, 1997.

Pagel 96 G.U. PAGEL et M. WINTER - Towards patterns-based tools, Proceedings of EuroPloP'96 (1996).

Panet 94 G. PANET, R. LETOUCHE - MERISE/2, Modèles et techniques Merise avancés, Les éditions d'organisation, 1994.

Parallax 94 GrahTalk 2.5 - Référence manual, Parallax Software technologies, 1994.

ParcPlace 90 ParcPlace Systems, Mountain View, CA – ObjectWorksSmalltalk Release 4 Users Guide, 1990.

Penney 87 D.J. PENNEY, J. STEIN - Class Modification in the Gemstone OODBMS, In Proc. OOPSLA, Orlando, Florida, octobre 1987.

Peterson 77 J.L. Peterson – Petri Nets, ACM Computing Surveys, vol9, n°3, 1977.

Plihon 96 V. PLIHON - Un environnement intégré pour l'ingénierie des méthodes, Thèse de doctorat Paris 1, 1996.

Pree 98 W. PREE - *Design Patterns et Architectures Logicielles*, Traduction de M. Lhomme, Vuibert, 1998.

Ra 94 Y.G. RA, H.A. KUNO, E.A. RUDENSTEINER, A flexible Object-Oriented Database Model and Implementation for capacity-Augmenting Views, University of Michigan, Dept of Electrical Engineering and Computer Science, Rapport CSE-TR-215-94, 1994

Randoing 95 J.M. RANDOING - Les SGDT, Editions Hermes, 1995.

Rational 97 Rational Software Corporation, Rational Objectory Process 4.1, Technical Report, 1997.

Rechenman 88 F. RECHENMAN – Shirka : un système de gestion de base de connaissances centrées objet, Rapport technique INRIA, 1988.

Reenskaugh 92 T. REENSKAUGH and al. - OORASS: seamless support for the creation and maintenance of object oriented systems, Journal of Object Oriented Programming, 1992.

Revault 95 N. REVAULT, H.A. SAHRAOUI, G. BLAIN, J.F PERROT - A Metamodeling Technique : The METAGEN system, TOOLS Europe'95, Versailles, 1995, 127-139.

Revault 96 N. REVAULT – Principes de méta-modélisation pour l'utilisation de canevas d'applications à objets (MetaGen et les frameworks), Thèse de l'Université Paris VI, 1996.

Rieu 85 D. RIEU - Modèle et Fonctionnalités d'un SGBD pour les applications CAO, Journées Bases de Données Avancées, mars 85.

Rieu 86a D. RIEU - Nature, Etat et Dynamicité de l'objet CAO. Journées Bases de Données Avancées, Giens, avril 1986.

Rieu 86b D. RIEU, G.T. NGUYEN - Semantics of CAD objects for Generalized Databases, 23rd ACM-IEEE Design Automation Conference, Las Vegas (USA), juillet 1986.

Rieu 86c D. RIEU, G.T. NGUYEN - Modélisation d'objets en CAO : versions, représentations et implantation, Revue "Modèles et Bases de Données", n° 5. AFCET Informatique, décembre 1986.

Rieu 87a D. RIEU, G.T. NGUYEN - Contrôle automatique de cohérence sur des objets dynamiques, Journées INRIA Bases de Données Avancées. Port Camargue, mai 1987.

Rieu 87b D. RIEU, G.T. NGUYEN - Knowledge base support for dynamic objects, International Conference on Data and Knowledge Systems for Manufacturing and Engineering, Hartford (Connecticut), octobre 1987.

Rieu 91 D. RIEU et al - Instanciation multiple et classification d'objets, 7ieme journées BDA, septembre 91

Rieu 92a D. RIEU, G.T NGUYEN, J. ESCAMILLA - SHOOD : an object model for design applications, Proc. 7th Intl. Conf. Artificial Intelligence, Expert Systems & Natural Language, Avignon (F), juin 1992.

Rieu 92b D. RIEU, G.T. NGUYEN - Object views for engineering databases, 3rd. conf. Data & Knowledge Systems for Manufacturing & Engineering, Lyon,1992

Rieu 94 D. RIEU, D. CONSTANT, G.T. NGUYEN, M. TOLLENAERE - Conception intégrée de liaisons mécaniques, Conf. Internationale sur la Modélisation et la Reconnaissance de Caractéristiques dans les Systèmes de CFAO Avancés, IFIP, mai 94.

Rieu 97a D. RIEU, M. TOLLENAERE, J.P. GIRAUDIN, F. BOUNAAS - Patrons d'Objets pour les SGDT : le projet POSEIDON, 2ème congrès de Génie Industriel, Albi, 1997.

Rieu 97b D. RIEU, F. BOUNAAS, P. MORAT, D. TAMZALIT - La métacircularité au service de la métamodélisation, Congrès INFORSID, Toulouse, juin 1997.

Rieu 99 D. RIEU, J.P. GIRAUDIN, A. CONTE-FRONT, C. SAINT-MARCEL - Des opérations pour les patrons de conception, Congrès INFORSID 1999.

Robinson 92 P.J. ROBINSON - Hierarchical Object-Oriented Design, Prentice-Hall, 1992.

Roland 97 D. ROLAND, J.L HAINAUT - Database Engineering Process Modelling, Proceeding of the first International WorkShop on Many Facets of Process Engineering, Gammarth, Tunisie, septembre 1997.

Rolland 93 C. ROLLAND – Adapter les Méthodes à l'Objet : Challenges et Embûches. Journées Méthodes d'Analyse et de Conception Orientées Objet des Systèmes d'Information, AFCET, Paris, (1993).

Rolland 95 C. ROLLAND, S. SOUVEYET, M. MORENO - An approach For Definig Ways-Of-Working, Information Systems, Vol 20, n° 4, 1995.

Rolland 96 C. ROLLAND - L'ingénierie des processus de développement de systèmes : un cadre de référence. Revue Ingénierie des Systèmes d'Information, Hermes,vol 4, n°6, 1996.

Rolland 97 C. ROLLAND - A primer for method engineering, congrès INFORSID, Toulouse, juin 1997.

Rolland 98 C. ROLLAND - A comprehensive View of Process Engineering, Proceedings of the 10th International Conference CAISE 98, Pise, Italie, juin 1998.

Rumbaugh91 J. RUMBAUGH, M. BLAHA, WW. PREMERLANI and al. - Object-Oriented modeling and design, Prentice Hall, 1991.

Rumbaugh 95 J. RUMBAUGH and al. - OMT : Modélisation et Conception Orientées Objet, Editions masson, Paris 1995.

Saeki 95 M.SAEKI - Object-Oriented Meta Modelling. Proceedings of Object-Oriented and Entity-Relationship Modeling, Australia 95, LNCS 1021.

SaintMarcel 98 C. SAINT MARCEL - Modélisation du comportement, une approche par les rôles, Congrès Inforsid, mai 1998.

SaintMarcel 99a C. SAINT MARCEL, P. MORAT, D. RIEU - Notion, Comportement, Rôle : trois concepts de modélisation pour une meilleure réutilisation des comportements, 2nd worshop on the Many Facets of Process Engineering, Gammarth,Tunisie, mai 1999.

SaintMarcel 99b C. SAINT MARCEL, D. RIEU, P. MORAT – Patron « Rôles Dynamiques d'Objet », 2nd worshop on the Many Facets of Process Engineering, Gammarth,Tunisie, mai 1999.

SaintMarcel 99c C. SAINT MARCEL – Tracabilité et Réutilisation des spécifications comportementales d'objets, Thèse de l'INPG de Grenoble, 1999.

Scholl 91 M. SCHOLL and al. - Updatable Views in Object-Oriented Databases, in proc. 2nd DOOD conf., Allemagne, décembre 94

Semmak 98 F. SEMMAK - Réutilisation de composants de domaine dans la conception des systèmes d'information, Thèse de l'Université Paris I, France, 1998.

Smolander 91 K. SMOLANDER, K. LYYTINEN and al. - Meta-Edit+ : A flexible graphical environment for methodology modelling, Advanced Information Systems Engineering, LNCS #498, Springer Verlag, 1991

Sunyé 99 G. SUNYE – PatternGen, un outil de mise en œuvre de patterns de conception, Thèse de l'Université Paris VI, 1999.

Soukup 95 J. SOUKUP – Implementing Patterns, In J. Coplien et D. Schmidt, Pattern Languages of Program Design, P. 395-412, Addison-Wesley, 1995.

Souza 94 C. SOUZA DOS SANTOS - Design and Implementation of an Object-Oriented View Mechanism, Journées BDA, 1994

Spanoudakis 96 K. SPANOUDAKIS, P. CONSTANPOPOULOS - Elaborating analogies from conceptual models, Internation Journal of Intelligent Systems, Vol. 11, No. 11, novembre 1996.

Staldelmann 89 M. STALDELMANN, G. KAPPEL, J. VITEK – ITHACA Visual Scripting Toll : a first implementation based on the unix shell scripting model, ITHACA report CUI.89.E4, Université de Genève, 1989.

Stroustrup 89 B. STROUSTRUP – Le langage C++, InterEditions 1989.

Tamzalit 96 D. TAMZALIT - Meninge : un noyau de MEta-modélisatioN pour les projets d'INGEnierie, DEA de Génie Industriel, INPG, Grenoble, juin 1996.

Tollenaere 95 M. TOLLENAERE - Modélisation objet pour la CFAO : modèle de conception, projet SHOOD, Rapport de fin de contrat 1995, Région Rhône-Alpes, Pôle Productique.

Trichon 91 F.TRICHON - Modélisation du processus de conception des machines électriques : le système expert DAMOCLES, Doctorat de l'INPG, spécialité électrique, mars 91

Trousse 89 B. TROUSSE - Coopération entre systèmes à base de connaissances et outils de CAO : l'environnement multi-agant ANAXAGORE, Thèse de doctorat de l'Université de Nice-Sophia Antipolis, décembre 1989.

Trousse 97 B. TROUSSE - Objets et CAO. Chapitre 9 de l'ouvrage collectif Ingénierie Objet, InterEditions 1997.

Vargas 95 C. VARGAS - Modélisation du processus de conception en ingénierie mécaniques. Mise en œuvre basée sur la propagaration de contraintes. Application à la conception d'une culasse automobile. Thèse de doctorat ENS de Cacahn, 1995.

Vayda 95 T. P. VAYDA - Lessons From the Battlefield, OOPSLA95

Velez 89 F. VELEZ, G. BERNARD, V. DARNIS - The O2 Object Manager : an Overview, International Conference VLDB, 1989.

Vernadat 94 F. VERNADAT - Future R&D Directions for CIM Deployment,, European Workshop on Integrated Manufacturing Systems Engineering, Grenoble, France, dec 12-14, 1994, pp. 3-6.

Voyer 92 P. VOYER – Editalk, Documentation Technique, Rapport interne Ackia, 1992.

Waite 97 E.J. WAITE - AIT: Advanced Information Technology for Design and Manufacture, Proc. Of ICEIMT'97, International Conference on Entreprise Integration and Modeling Technology, Eds. K. Kosanke, J.G. Nell, 1997.

Walden 94 K. WALDEN et J.M. NERSON - Seamless Object-Oriented Software Architecture, Analysis and Design of Reliable Systems, Prentice Hall, ISBN 0-13-031303-3, 1994.

Wartik 92 S. WARTIK et R. PIETRO-DIAZ - Criteria for comparing Domain Analysis Approaches, International Journal of Software Engineering and Knowledge Engineering, pp 403-431, 1992.

Weinand 88 A. WEINAND, E. GAMMA et R. MARTY - ET++ - An object-oriented application framework in C++ ", Proceedings of the Object-Oriented Programming Systems, Languages, and Applications Conference (1998), pages 46-57, ACM Press.

Wirfs-Brock 90 R. WIRFS-BROCK, B. WILKERSON et L. WIENER - Designing Object Oriented Software, Prentice-Hall, 1990.

Zicari 91 R. ZICARI - A framework for schema updates in an object oriented database system, in proc. Of the 7th Conf. On Data Engineering, Japan, 1991.