N° d'ordre D.U. 1797 Année 2007
EDSPIC n°390

Thèse

Un Système de Contraintes d'Intégrité OCL pour les Bases de Données Spatiales
-
Application à un Système d'Information pour l'Epandage Agricole

Présentée à l'Université Blaise Pascal

Pour obtenir
Le diplôme de doctorat

Spécialité
Informatique

École doctorale Sciences Pour l'Ingénieur (SPI) de Clermont-Ferrand

Par
Magali Duboisset

Soutenue le 3 décembre 2007 devant la Commission d'examen

Jury MM.

Rapporteur	**Christophe Claramunt**	Professeur (Ecole Navale, Brest)
Rapporteur	**Robert Laurini**	Professeur (INSA, Lyon)
Examinateur	**Ahmed Lbath**	Professeur (Univ. Joseph Fourier, Grenoble)
Co-Directeur	**François Pinet**	Chargé de recherche (Cemagref, Clermont-Ferrand)
Examinateur	**Alain Quilliot**	Professeur (Univ. Blaise Pascal, Clermont-Ferrand)
Directeur	**Michel Schneider**	Professeur (Univ. Blaise Pascal, Clermont-Ferrand)

Remerciements

Acte final, dernière scène : rédaction des remerciements. C'est ici que s'achèvent trois années passées au sein de l'UR TSCF / UMR TETIS du Cemagref de Clermont-Ferrand. Je remercie MM. Emmanuel Hugo et Didier Méchineau de m'y avoir accueillie et offert ainsi un cadre idéal pour mener à bien mes travaux, les valoriser et communiquer. L'étroite collaboration avec le LIMOS m'a permis de rester en contact avec le monde de l'enseignement durant ces trois ans ; merci à son directeur M. Alain Quilliot d'avoir permis cette collaboration, ainsi que d'avoir présidé mon jury de soutenance.
Je souhaite également remercier le Conseil Régional d'Auvergne et le Cemagref sans les financements desquels cette aventure n'aurait été réalisable.
Merci à MM. Christophe Claramunt et Robert Laurini, rapporteurs de ce mémoire de thèse, et à M. Ahmed Lbath, examinateur, pour l'intérêt qu'ils ont porté à mes travaux. Leur contribution m'a ouvert de nouvelles perspectives quant à l'avenir de mes recherches.

Je tiens à remercier tout particulièrement M. Michel Schneider, d'une part, pour m'avoir informée de l'ouverture de cette thèse, d'autre part, pour l'avoir dirigée ensuite. Ses conseils et les directions qu'il m'a suggéré de suivre m'ont beaucoup aidée.
Merci également à M. François Pinet qui m'a encadrée tout au long de ce doctorat. Son soutien dans un moment difficile de ma vie m'a beaucoup touchée. Ces remerciements s'adressent aussi à Myoung-Ah Kang, pour les réunions de travail et les publications que nous avons pu faire ensemble. Ce fut un plaisir de travailler avec de jeunes chercheurs dynamiques.
Jeune est aussi l'esprit de l'équipe à laquelle j'ai appartenu durant ces années de labeur : l'équipe COPAIN. L'ambiance joviale qui y règne fait de chaque réunion un moment très attendu. Je remercie M. Frédéric Vigier pour son soutien, ainsi que M. Jean-Pierre Chanet et tous les membres de l'équipe COPAIN.

Et qu'aurait été cette thèse sans mon homologue masculin, Cédric Tessier, avec qui j'ai partagé un bureau et bien des crises : les petites crises "d'angoisse" connues de tout thésard, mais surtout les crises de rires. De ces trois ans de cohabitation sont nés deux docteurs et une véritable amitié. Cédric, merci pour tout.
Les couloirs du Cemagref regorgent de personnes de grande valeur : parmi elles, Claire Calchera, Laure Moiroux-Arvis, Christophe Debain, les membres de la chorale du Cemagref… Et d'autres n'ont fait hélas que passer dans ces murs : Thomas Brun, Laurent Dufy… Merci à tous d'avoir égayé les déjeuners à la cantine et autres "galettes des rois" du Cemagref.

Enfin, un immense merci à tous mes proches. Ils ont tous, directement ou non, participé à l'aboutissement de ces trois années de thèse.
A mes grands-parents, Pierre et Edith Peinet, à mes parents, j'espère qu'ils sont fiers de m'avoir pour fille comme je suis heureuse de les avoir pour parents. A mon frère Christophe et Cécile, ainsi qu'à mon futur neveu / ma future nièce – l'attente de ton arrivée me semble interminable !
Merci à Christophe, a.k.a. K, qui m'a supportée, dans tous les sens du terme ces derniers mois… Merci de m'avoir ouvert les yeux sur l'importance des tableaux récapitulatifs, du punk et du vin de Bourgogne.
Merci à Caroline, sans qui je n'aurais peut-être jamais intégré les bancs de l'IUT d'informatique. Merci de m'avoir aidée à trouver ma voie – et de m'avoir appris à nager…
Merci à Céléna, ou le dynamisme personnifié. Merci d'être toujours là quand il est temps de se changer les idées.
Merci à Stèph R., à qui je souhaite beaucoup de réussite dans sa nouvelle voie. Dommage que nous n'ayons pas terminé l'aventure ensemble, les cours de l'école doctorale étaient plus sympas avec toi.
Merci à Stèph B. d'avoir partagé mes pauses cafés et de m'avoir recueillie quand je n'avais plus de toit.
Un grand merci également à Nico a.k.a. S'rin, à Titi, à Jeeb, mon binôme à l'ISIMA, et à Marie-Marthe a.k.a. m&m's.

Merci à vous qui lisez ces quelques pages de les faire vivre.
Je referme ici ce mémoire et trois années de ma vie, en espérant que cela débouche sur d'autres expériences tout aussi passionnantes et enrichissantes.

Glossaire

9 I.M.	**9 I**ntersection **M**atrix. Modèle de relations topologiques entre objets spatiaux simples.
ADEME	**A**gence **D**e l'**E**nvironnement et de la **M**aîtrise de l'**E**nergie.
A.G.L.	**A**telier de **G**énie **L**ogiciel. (**CASE**, **C**omputer **A**ided **S**oftware **E**nvironment) Ensemble cohérent d'outils informatiques formant un environnement d'aide à la conception, au développement et à la mise au point de logiciels d'application spécialisés.
Application géomatique	Application s'adressant à une classe de problèmes de gestion de données spatiales orientées utilisateur final non expert.
Base de données	Collection de données inter reliées, stockées dans des relations (ou tables).
C.B.M.	**C**alculus **B**ased **M**ethod. Modèle de relations topologiques entre objets spatiaux simples.
Cemagref	Etablissement public de recherche sur l'ingénierie de l'agriculture et de l'environnement.
Contrainte d'intégrité	Règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la base de données.
DDAF	**D**irection **D**épartementale de l'**A**griculture et de la **F**orêt.
DDASS	**D**irection **D**épartementale des **A**ffaires **S**anitaires et **S**ociales.
DDE	**D**irection **D**épartementale de l'**E**quipement.
DDSV	**D**irection **D**épartementale des **S**ervices **V**étérinaires.
DIREN	**DI**rection **R**égionale de l'**EN**vironnement.
DRIRE	**D**irection **R**égionale de l'**I**ndustrie, de la **R**echerche et de l'**E**nvironnement.
D.T.D.	**D**ocument **T**ype **D**efinition. Document permettant de décrire la structure d'un document SGML ou XML.

ISIMA **I**nstitut **S**upérieur d'**I**nformatique, de **M**odélisation et de leurs **A**pplications.

LIMOS **L**aboratoire de recherche d'**I**nformatique, de **M**odélisation et d'**O**ptimisation des **S**ystèmes.

MADS **M**odel for **A**pplication **D**ata with **S**patio-temporal features.
Formalisme orienté-objet dédié pour la modélisation de systèmes d'information géographiques

O.C.L. **O**bject **C**onstraint **L**anguage.
Langage textuel de contraintes associé à UML.

O.G.C. **O**pen **G**eospatial **C**onsortium.
Organisation créée pour régler le problème d'interopérabilité entre les systèmes qui traitent des données géospatiales, élaborant des spécifications d'interface disponibles au grand public.

O.M.G. **O**bject **M**anagement **G**roup.
Association de professionnel de l'informatique orientée objet ayant notamment défini la norme CORBA.

R.C.C. **R**egion **C**onnection **C**alculus.
Modèle de relations topologiques entre objets spatiaux simples.

S.G.B.D. **S**ystème de **G**estion de **B**ase de **D**onnées.
Ensemble des programmes assurant la structuration, le stockage, la mise à jour et la recherche des données dans une base.

S.I.G. **S**ystème d'**I**nformation **G**éographique.
Outil destiné à l'édition, la modélisation et la visualisation de données à référence spatiale.

SIGEMO **S**ystème **I**nformatisé de **G**estion des **E**pandages de **M**atières **O**rganiques.
Projet informatique visant à enregistrer et restituer par internet l'ensemble des données relatives à l'épandage de matières organiques.

S.P.I. Ecole Doctorale **S**ciences **P**our l'**I**ngénieur de l'Université Blaise Pascal.

S.Q.L. **S**tructured **Q**uery **L**anguage.
Langage de définition, de manipulation et de contrôle de données pour les bases de données relationnelles.

T.R.C.R. **T**opological **R**elationships on **C**omposite **R**egions.
Modèle de relations topologiques entre objets spatiaux composites.

T.S.C.F. Unité **T**echnologies, **S**ystèmes d'information pour les agro-systèmes du Cemagref de **C**lermont-**F**errand.

U.M.L. **U**nified **M**odeling **L**anguage.
Langage graphique de modélisation de système d'information.

X.M.I. **X**ML **M**etadata **I**nterchange.
Spécification OMG exprimée sous forme de DTD (suivant les versions) et de schémas XML accompagnés de leurs règles de production, servant à l'échange des modèles et des métamodèles sous forme de documents XML.

X.M.L. E**X**tensible **M**arkup **L**anguage.
Standard de description de données défini par le W3C.

W3C **W**orld **W**ide **W**eb **C**onsortium.
Organisme officiellement chargé de la normalisation de tout ce qui concerne le Web, et, en particulier, des évolutions du langage HTML.

ZNIEFF **Z**one **N**aturelle d'**I**ntérêt **E**cologique **F**aunistique et **F**loristique.
Zone définissant un périmètre d'intérêt écologique.

Table des figures et illustrations

Table des matières

Chapitre 1
Introduction

1.1 Contexte

Le contrôle des contraintes d'intégrité permet d'assurer la qualité des données et la cohérence d'un système. Les données qui ne respectent pas ces contraintes dans le système sont erronées ; il est donc important d'ajouter au modèle de système d'information la spécification de ces contraintes. Les auteurs de [ABI00] précisent que cette spécification offre un cadre pour ajouter une sémantique au modèle de base de données : "on incorpore des contraintes d'intégrité (en anglais *integrity constraints*), c'est-à-dire des propriétés supposées être satisfaites par toutes les instances d'un schéma de base de données."

Par nature, les systèmes d'information (SI) environnementaux manipulent souvent des données géoréférencées. L'objectif de nos travaux était de fournir des méthodes pour modéliser (de manière expressive) les contraintes d'intégrité des bases de données spatiales, ainsi que mettre en place des outils pour les contrôler. La modélisation des contraintes devait pouvoir s'intégrer facilement aux formalismes utilisés pour la conception de système d'information, tels qu'UML. Dans cette thèse, le champ d'application des travaux s'est placé dans le cadre d'un système d'information agricole pour la gestion des épandages.

Dans la littérature, il a été proposé des solutions qui peuvent être employées pour modéliser les contraintes d'intégrité des bases de données spatiales (formalismes orientés-objet pour les SIG [PAR98] [PAR06] [BOR99] [PAN94] [PAN96] [PIN02] ou outils à base d'interface graphique [SER00] [UBE97] [COC04]). Ces propositions fournissent d'excellents résultats pour l'expression de contraintes d'un niveau de complexité moyen, en privilègiant les représentations graphiques dans un objectif de dialogue avec les utilisateurs finaux. Dans nos travaux, nous avons recherché un formalisme plus expressif, pouvant à la fois exprimer des contraintes sur les données alphanumériques et sur les données spatiales. Utiliser un seul formalisme pour ces deux types de données permet ainsi de modéliser sous une même forme toutes les contraintes d'intégrité d'une base de données.

Des langages ou des méthodes offrent la possibilité d'exprimer de manière déclarative et expressive des contraintes d'intégrité sur des données alphanumériques, comme OCL, le Langage de Contrainte Objet associé à UML [DEM99] [DEM01] [OMG05] ou la méthode B [ABR96] [MAR01]. Néanmoins, aucun de ces deux langages ne permettaient de modéliser simplement les contraintes spatiales.

Dans cette thèse, nous nous focalisons sur l'étude des régions simples et composites, c'est-à-dire faites de l'union de plusieurs régions simples. Afin d'exprimer des contraintes complexes sur ce type d'objets spatiaux, nous avons proposé des extensions spécifiques du langage de contraintes OCL. Un des atouts de ce langage est qu'il est dédié à la spécification de contraintes ; de plus il est étroitement lié à UML, et il est supporté par différents générateurs automatiques de code [KLA05]. Afin d'offrir une solution opérationnelle, nous avons étendu un outil de ce type, OCL2SQL, pour générer automatiquement le code de vérification des contraintes exprimées avec les extensions spatiales d'OCL proposées.

Les extensions spatiales d'OCL ont été expérimentées dans le cadre du projet SIGEMO au Cemagref (Système d'Information sur la Gestion des Epandages de Matières Organiques). Ce projet répond à une nécessité de suivi fiabilisé des pratiques d'épandage de matières organiques en France [SOU03]. Mis en place en 2006, le système d'information national SIGEMO permet à différents acteurs de saisir, transmettre et analyser des projets d'épandage par internet. Toutes les informations sont centralisées dans la base de données nationale SIGEMO. Dans la phase de démarrage du projet, différentes données issues de sources externes ont dû être intégrées dans la base de données nationale (par exemple des parcelles d'épandage déjà disponibles en version électronique avant SIGEMO).

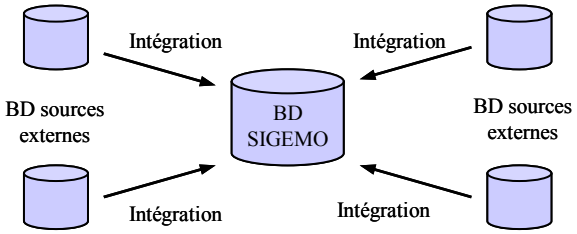


Figure 1. Processus d'intégration

Ainsi, nous avons appliqué nos propositions pour modéliser et vérifier les contraintes spatiales des données de SIGEMO ; aussi bien celles intégrées dans la base de données SIGEMO depuis d'autres sources de données, que celles saisies directement par les utilisateurs via l'application SIGEMO (via Internet). Dans cette optique, nos travaux ont ainsi visé à offrir, d'une part, des moyens pour spécifier les contraintes alphanumériques et spatiales que le système doit satisfaire, et d'autre part, des outils pour tester effectivement ces contraintes sur la base.

Nous présentons maintenant quelques définitions de base.

Dans [COC97], l'auteur propose une taxonomie des contraintes d'intégrité spatiales. Cette classification distingue 3 catégories de contraintes spatiales :

- les **contraintes topologiques**. Elles portent sur la topologie des objets géographiques, c'est-à-dire leurs propriétés géométriques et les relations spatiales qui peuvent exister entre ces objets. Elles peuvent dépendre de la structure des objets, notamment pour les composants d'une partition qui sont tous disjoints ou adjacents 2 à 2.
- les **contraintes d'intégrité sémantique**. Elles se rapportent à la signification des objets géographiques, comme dans la contrainte qui spécifie que "un bâtiment ne peut pas être situé dans un lac". Elles portent sur la nature des objets dans le monde réel.
- les **contraintes d'intégrité définies par l'utilisateur**. Elles ne sont pas nécessairement basées sur la sémantique. On peut les assimiler aux règles de gestion des systèmes alphanumériques. Par exemple, si la législation impose que les stations d'épuration ne soient pas dans certaines zones, alors une règle définie par l'utilisateur empêchera l'insertion de stations d'épuration dans ces zones particulières.

Dans [SER00], les auteurs introduisent la notion de contraintes topo-sémantiques, qui relèvent de la signification des objets du monde réel et des relations qu'ils ont avec les autres objets.

Pour différents types d'objets, notamment ceux ayant plusieurs niveaux de représentation, [ROD04] passe en revue les travaux sur les incohérences dans les bases de données spatiales. Par **qualité des données**, on entend l'adéquation des données avec certains critères [DEV04]. Dans son mémoire de thèse, [PUR00] identifie 7 éléments de qualité de données spatiales :

- la généalogie, "l'histoire des données", traite de la provenance des données, et des moyens par lesquels elles ont été produites.
- la précision géométrique, ou géographique, évalue le degré de conformité des données par rapport au monde réel en termes de position.
- la précision sémantique correspond à la précision géométrique pour les attributs non-spatiaux ; elle définit les différences entre les valeurs des attributs et leurs valeurs réelles.
- l'exhaustivité contient toutes les informations sur les moyens et règles utilisés lors de l'acquisition des données, comme par exemple les seuils utilisés, et les relations entre les objets du système et les objets du monde réel.
- la cohérence logique se rapporte à la structure des données de la base. Par exemple, certains attributs appartiennent à un domaine particulier (latitude comprise entre 0 et 90°, etc.).
- la cohérence sémantique décrit la qualité avec laquelle les objets sont représentés par le modèle utilisé. Elle concerne essentiellement la pertinence et la signification des objets par la mesure de la "distance sémantique" entre les objets réels et leurs représentations.
- la précision temporelle traite des informations sur l'aspect temporel des données : leurs dates d'origine, leurs fréquences de mise à jour, leurs validité.

Les travaux de cette thèse traitent des problèmes de cohérence logique et de cohérence sémantique.

Les sources d'incohérence sémantique sont multiples. Par exemple, dans un SIG, une maison peut être située dans un lac pour différentes raisons : suite à une erreur de saisie, ou suite à une évolution temporelle non-enregistrée (si la rive du lac a été modifiée), ou encore suite à un problème de cohérence logique (le système ne permet pas d'indiquer que la maison est sur pilotis), ou enfin suite à un problème d'exhaustivité (une île a été oubliée).

1.2 Exemples de contraintes d'intégrité

Les exemples suivants illustrent quelques-uns des cas de contraintes d'intégrité vues avec les experts-métier et qu'il était intéressant de pouvoir exprimer sur la base de données de SIGEMO.

Exemple 1.

"Les projets d'épandage ne devraient pas prévoir d'épandage en zones ZNIEFF 1."

Certaines zones naturelles ne devraient recevoir aucun produit d'épandage, notamment certains types de zones ZNIEFF (Zones Naturelles d'Intérêt Ecologique Faunistique et Floristique). Sur la Figure 2, deux zones ZNIEFF intersectent des parcelles d'un périmètre d'épandage *PE1*. Un périmètre d'épandage est un ensemble de parcelles d'épandages. Seules les zones ZNIEFF de type 1 sont concernées par la contrainte ci-dessus. Sur cet exemple, seule la zone d'identifiant *ZNIEFF_B* viole la contrainte.

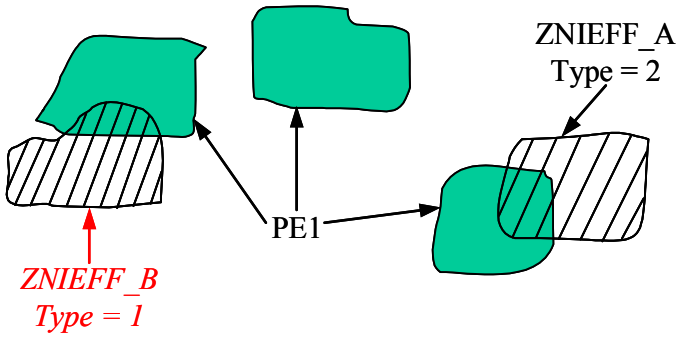


Figure 2. Une zone ZNIEFF de type 1 violant la contrainte d'intégrité de l'Exemple 1.

Les propositions faites dans cette thèse vont permettre de spécifier des contraintes spatiales dépendant de la valeur d'un attribut, comme c'est le cas ici avec l'attribut *Type* de la classe *ZNIEFF*. □

Exemple 2.

"Les parcelles d'épandage d'un périmètre d'épandage sont contenues dans le département qui lui est associé dans la base ou dans un des départements limitrophes de celui-ci."

La Figure 3 présente une situation qui vérifie cette contrainte. Cette extension aux départements limitrophes assouplit la gestion des projets d'épandage. En effet, en pratique, un périmètre d'épandage peut porté sur le Puy-de-Dôme mais comporté quelques parcelles dans des départements alentour. Ainsi, son dossier ne sera suivi que par les administrations du département principal auquel il est rattaché.

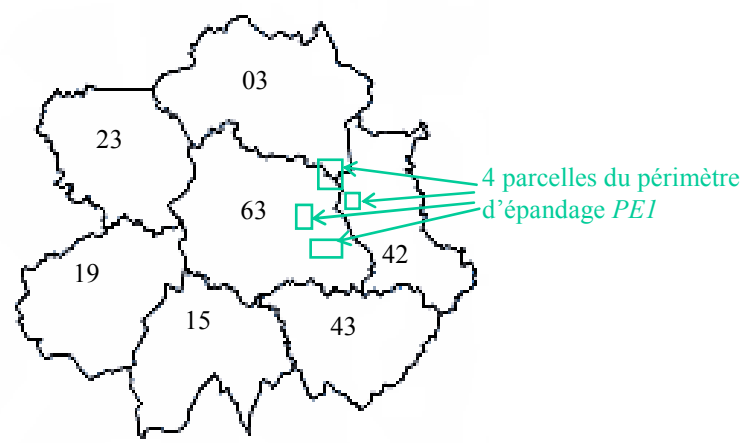


Figure 3. Un plan d'épandage rattaché au Puy-de-Dôme (63), localisé dans le 63 et ses départements limitrophes

Les langages de contraintes proposés dans cette thèse autoriseront la spécification de contraintes spatiales dépendant d'une autre relation spatiale, comme c'est le cas sur cet exemple. En effet, ici, un département contient une parcelle d'un périmètre d'épandage, si ce département est adjacent au département de rattachement de ce périmètre d'épandage. □

1.3 Plan de thèse

Nous allons proposer dans ce mémoire des langages de contraintes permettant d'exprimer des relations topologiques entre objets spatiaux de type région. Ces langages sont des extensions d'OCL qui est le langage de contraintes "standard" associé à UML. Ils sont basés sur l'intégration dans OCL de relations topologiques issues de différents modèles. Pour les relations entre deux régions simples, nous avons intégré les relations des deux modèles les plus utilisés : 9IM (9-Intersection Model) et CBM (Calculus-Based Method). Une région composite est une région faite de plusieurs régions simples (ses composants). Pour les relations entre deux régions composites, nous proposons, d'une part, d'utiliser les opérations sur les collections d'OCL afin d'accéder aux composants des régions pour leur appliquer les relations définies entre deux objets simples ; d'autre part, nous introduisons des langages offrant de nouvelles opérations pour spécifier des relations entre régions composites.

Nos propositions de langages de contraintes spatiales sont intégrées dans un outil de génération de code : OCL2SQL. Ce générateur produit, à partir de contraintes spécifiées au niveau conceptuel, les mécanismes de vérification correspondants.

Ce mémoire s'organise comme suit :

Partie 2. Cette partie réalise un état de l'art : elle introduit tout d'abord les modèles pour la spécification de relations topologiques entre objets simples, puis entre objets complexes ou composites ; ensuite elle présente les différents formalismes dédiés à la modélisation de Systèmes d'Information Géographiques ; enfin, elle recense les divers moyens pouvant être utilisés ou adaptés pour spécifier des contraintes spatiales.

Partie 3. Cette partie présente les extensions spatiales d'OCL que nous proposons. Dans une première section, seuls les types spatiaux sont intégrés dans OCL et seules les contraintes sur les types des objets sont possibles. Ensuite, dans une deuxième section, les langages proposés fournissent des opérations pour exprimer les relations entre deux géométries simples. Enfin, la troisième section s'intéresse plus particulièrement aux solutions proposées pour la spécification de contraintes impliquant des objets complexes ou composites, c'est-à-dire des objets géographiques composés de plusieurs objets simples.

Partie 4. Dans cette partie, nos différentes propositions sont comparées en termes d'expressivité d'une part entre elles, et d'autre part avec des modèles particulièrement expressifs pour décrire des relations pour les régions composites.

Partie 5. Cette partie concerne l'implémentation de nos propositions. Cette implémentation a été effectuée par l'intermédiaire d'un outil open source, OCL2SQL, qui, à partir de contraintes alphanumériques spécifiées en OCL, génère le code de vérification des contraintes en SQL pour Oracle. Les langages de contraintes spatiaux proposés en Partie 4 sont implémentés dans l'extension de cet outil au spatial.

Partie 6. Cette partie décrit les expérimentations que nous avons effectuées sur le projet SIGEMO au Cemagref. Ce projet permet la gestion via internet des projets d'épandage de matières organiques en France. Les contraintes spécifiées grâce à nos propositions de langages sont présentées dans cette partie ; elles ont été testées sur la base de données de SIGEMO grâce aux requêtes SQL générées par OCL2SQL. Cette partie recense différentes contraintes alphanumériques et spatiales portant sur SIGEMO.

Partie 7. Cette partie concerne les conclusions tirées de nos travaux et les perspectives qu'elles suscitent.

Annexe. Cette annexe fournit les 16 matrices des 9-Intersections représentant des relations entre régions composites et, pour chacune d'entre elles, l'équivalence avec l'une de nos propositions de langage de contraintes.

Chapitre 2
Etat de l'art

Dans cette partie, les 3 éléments nécessaires pour la compréhension de notre proposition d'un langage de contraintes spatiales (Section 3) sont présentés. Cet état de l'art consiste en 3 parties : la présentation des modèles pour la spécification de relations topologiques (sections 2.1 et 2.2), celle des formalismes orientés-objet pour les SIG (Section 2.3), et enfin celle des langages de contraintes (Section 2.4).

2.1 Modèles de relations topologiques entre objets simples

Nous décrivons dans cette partie les principaux modèles de relations topologiques entre objets spatiaux simples (point, ligne, polygone). Nous nous intéressons aux modèles débouchant sur des implantations dans les bases de données spatiales.

2.1.1 9IM

Tel que défini dans [EGE92], l'intérieur, la frontière et l'extérieur d'un objet spatial O sont notés respectivement $O°$, δO et O^-. La Figure 4 illustre les intérieurs, frontières et extérieurs pour les trois types d'objets simples point, ligne et région.

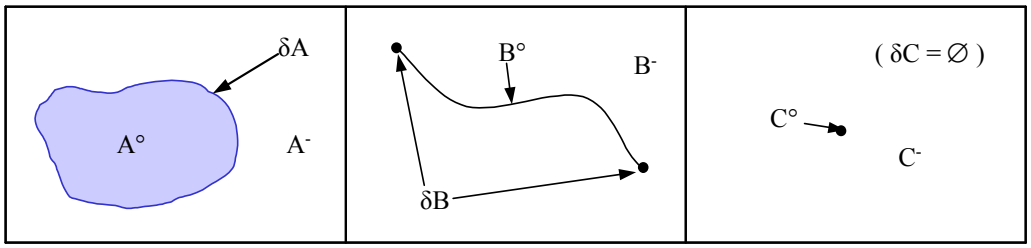


Figure 4. Intérieur, frontière et extérieur d'une région *A*, d'une ligne *B* et d'un point *C*

Dans [EGE92], le modèle des 9 intersections (9IM) fournit une méthodologie pour formaliser et identifier les relations topologiques entre 2 objets spatiaux simples (de type point, ligne ou région). Comme indiqué

dans [ORA05], des relations topologiques identifiées par 9IM ont été intégrées dans le SQL d'Oracle Spatial.

Avec 9IM, chaque relation topologique entre 2 objets spatiaux est représentée par une matrice. Cette matrice décrit l'intersection des frontières, des intérieurs et des extérieurs des 2 objets spatiaux (voir le modèle Figure 5). Il est à noter que 9IM peut être considéré comme une extension du modèle des 4-intersections [EGE93] [EGE94], 9IM ajoutant la notion d'extérieur des objets.

$$M = \begin{array}{|c|c|c|} \hline A^\circ \cap B^\circ \neq \varnothing & A^\circ \cap \delta B \neq \varnothing & A^\circ \cap B^- \neq \varnothing \\ \hline \delta A \cap B^\circ \neq \varnothing & \delta A \cap \delta B \neq \varnothing & \delta A \cap B^- \neq \varnothing \\ \hline A^- \cap B^\circ \neq \varnothing & A^- \cap \delta B \neq \varnothing & A^- \cap B^- \neq \varnothing \\ \hline \end{array}$$

Figure 5. Patron de matrice des 9-intersections

En théorie, on a donc 2^9 soit 512 matrices. Cependant, en fonction du type d'objet auquel on les applique, on peut supprimer des matrices qui correspondent à des cas impossibles à dessiner.

Exemple 3.
La matrice suivante représente un cas qui ne peut pas être dessiné entre deux régions simples sans trou *A* et *B*. Les coefficients nuls de la matrice correspondent aux intersections vides, les non-nuls aux intersections non-vides.

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

i) L'extérieur de A intersecte tout B, *ii)* l'extérieur de B intersecte tout A, *iii)* l'intérieur de A doit intersecter l'intérieur de B, *iv)* les autres éléments restent disjoints. Cette scène ne peut être dessinée dans un espace 2-D pour deux régions, par conséquent, elle ne correspond pas à une relation topologique acceptable. □

Il reste par exemple 8 matrices pour les relations topologiques entre 2 régions simples comme illustrées dans la Figure 6 [EGE92].

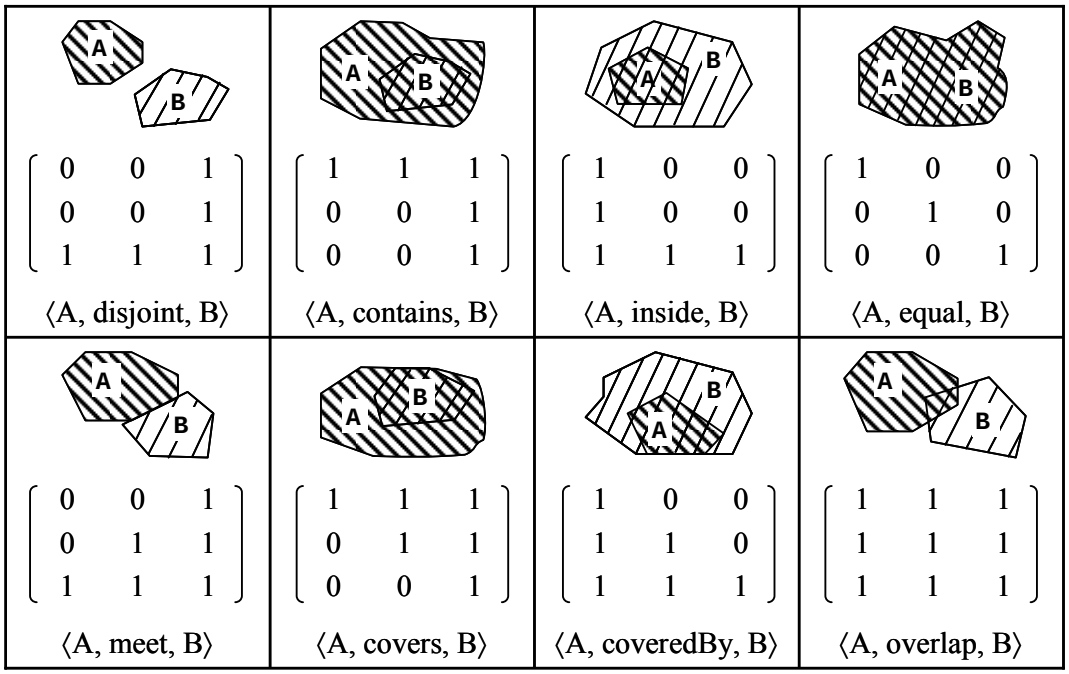


Figure 6. 8 relations topologiques possibles entre deux régions simples [EGE92]

2.1.2 CBM

Calculus-Based Method (CBM) distingue 5 relations topologiques de base entre 2 géométries simples, dont seulement 4 applicables entre 2 régions simples :

Définition 1. les 5 relations topologiques de CBM [CLE93]

Soient *A* et *B* deux géométries. *dim(A)* est la dimension de *A*.

(1) *touch* (appliqué à région/région, ligne/ligne, ligne/région, point/région, point/ligne) :

$\langle A, touch, B\rangle \Leftrightarrow (A^\circ \cap B^\circ = \emptyset) \wedge (A \cap B \neq \emptyset)$

(2) *in* (peut toujours être appliqué) :

$\langle A, in, B\rangle \Leftrightarrow (A \cap B = A) \wedge (A^\circ \cap B^\circ \neq \emptyset)$

(3) *overlap* (appliqué à région/région, ligne/ligne) :

$\langle A, overlap, B\rangle \Leftrightarrow (dim(A^\circ) = dim(B^\circ) = dim(A^\circ \cap B^\circ))$

$\wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$

(4) *disjoint* (peut toujours être appliqué) :

$\langle A, disjoint, B\rangle \Leftrightarrow A \cap B = \emptyset$

(5) *cross* (appliqué à ligne/ligne, ligne/région) :

$\langle A, cross, B\rangle \Leftrightarrow (dim(A^\circ \cap B^\circ) = max(dim(A), dim(B))-1)$

$\wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$ □

Des opérations peuvent aussi être utilisées sur les objets pour retourner leur frontière. En effet, il est défini dans CBM, une opération qui extrait la frontière d'une région et des opérations qui extraient les points des frontières d'une ligne. Ainsi, en combinant les 5 relations de la Définition 1 avec ces opérations, il devient possible d'exprimer davantage de relations topologiques. On peut notamment différencier les relations topologiques en fonction de la dimension de l'intersection des frontières des deux objets simples ; ainsi, 9 relations topologiques deviennent exprimables entre 2 régions simples (voir Figure 7).

Les 5 relations topologiques et les opérations de CBM sont par exemple supportées par PostGIS.

Si on considère les relations entre deux régions simples, 9IM distingue plus de relations topologiques que celles présentées dans la Définition 1 (relations 1 à 4 - c'est-à-dire les relations de base de CBM pouvant s'appliquer entre deux régions). Néanmoins, puisqu'on peut aussi considérer des opérations d'extraction de frontières dans CBM (en plus des relations de base), une comparaison entre 9IM et CBM a montré la supériorité d'expressivité offerte par CBM [CLE94] (en d'autres termes, davantage de relations sont distinguées par CBM). Par exemple, les relations 4 et 5 de la Figure 7 (CBM) correspondent à une seule relation dans la Figure 6 (la relation *meet* de 9IM). Néanmoins, si l'on indique dans les matrices des 9-intersections, la dimension de l'intersection entre les deux objets [CLE93], les deux modèles deviennent équivalents. On appelle 9IM étendu le modèle permettant de spécifier la dimension (c'est-à-dire 0-D, 1-D, 2-D) dans les matrices 9-intersections à la place des valeurs booléennes.

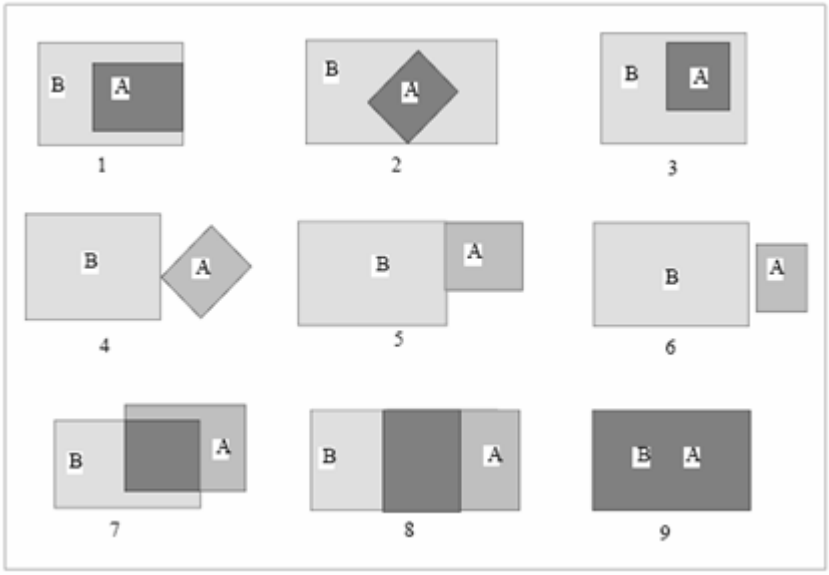


Figure 7. Relations topologiques issues de CBM entre deux régions simples

2.1.3 RCC

Le modèle Region Connection Calculus (RCC) décrit dans [RAN92] [COH97] est basé sur des relations topologiques entre deux régions. Partant d'une relation de connexion C(x,y) signifiant que les deux régions x et y partagent un point, un ensemble d'autres relations sont définies (par exemple "est déconnecté de", "est une partie de", "recouvre",…). Différentes autres fonctions sont aussi associées au formalisme, par exemple des fonctions retournant l'intersection de deux régions, leur différence, etc. Une étude sur l'implémentation de RCC est présentée dans [REN97].

RCC permet de distinguer 8 relations topologiques entre deux régions simples, qui sont en fait exactement les mêmes que celles identifiées par 9IM (voir Figure 6).

2.2 Modèles pour la spécification de relations topologiques entre régions complexes et composites

Nous décrivons dans cette partie les principaux modèles de relations topologiques entre régions complexes ou composites (c'est-à-dire certaines collections de régions simples). Nous nous focalisons sur les modèles basés sur 9IM et CBM qui sont plus répandus que RCC. Cependant, il existe également des modèles de spécification de relations topologiques entre régions complexes basés sur RCC, comme TCC [PRA01].

2.2.1 9IM appliqué aux régions complexes

Les travaux de [BEH01] proposent d'utiliser 9IM pour décrire les relations topologiques entre des régions dites complexes.

Définition 2. Région complexe [BEH01]
Soit $\{F_1 , ... , F_n\}$ un ensemble de régions simples pouvant comporter des trous.
L'ensemble $F = \bigcup_{i=1..n} F_i$ est une région complexe ssi :
(i) $\forall\ 1 \le i < j \le n : F_i^\circ \cap F_j^\circ = \emptyset$
(ii) $\forall\ 1 \le i < j \le n : \delta F_i \cap \delta F_j = \emptyset \vee |\ \delta F_i \cap \delta F_j\ |$ est fini. □

La Figure 8 présente des exemples de régions complexes.

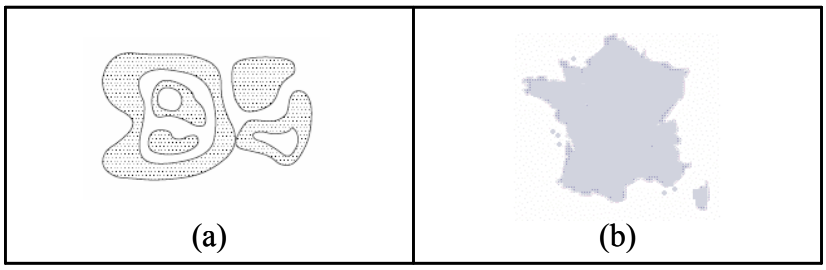


Figure 8. Exemples de régions complexes

Dans [BEH01], il est proposé d'utiliser une matrice 9IM pour décrire les relations entre deux objets complexes. Intuitivement parlant, une région complexe est composée de plusieurs sous-régions ; par exemple, cinq sous-régions sont présentées sur la Figure 8 (a). Ainsi, l'intérieur d'une région complexe correspond en fait à l'union des intérieurs de toutes ses sous-régions. De la même façon, la frontière d'une région complexe correspond à l'union des frontières de toutes ses sous-régions. L'extérieur d'une région complexe est l'intersection de tous les extérieurs de ses sous-régions.

Considérons par exemple la matrice suivante :

$$R = \begin{array}{c} \\ A^\circ \\ \delta A \\ A^- \end{array} \begin{array}{c} \begin{array}{ccc} B^\circ & \delta B & B^- \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}$$

Cette matrice ne correspond à aucune relation topologique entre régions simples (voir Figure 6). Par contre, elle peut correspondre à une relation topologique entre deux régions complexes comme illustré sur la Figure 9.

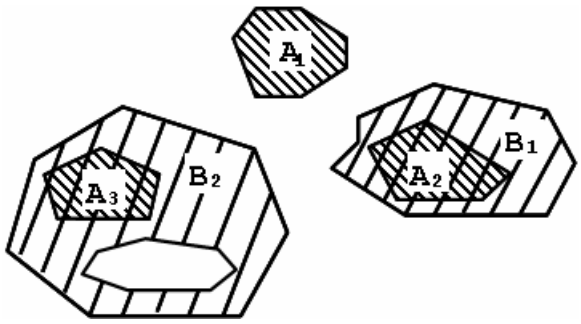


Figure 9. Exemple de scène correspondant à la matrice *R*

Sur la Figure 9, la région complexe *A* est composé de 3 sous-régions (A_1, A_2, A_3) et la région complexe *B* est composé de 2 sous-régions (B_1 et B_2). Selon la matrice *R*, les intérieurs de *A* et *B* s'intersectent, comme l'indique la valeur du coefficient en première ligne, première colonne ; c'est le cas dans l'exemple puisque l'intérieur d'une sous-région de *A* intersecte l'intérieur d'au moins une sous-région de *B*. Dans la matrice *R*, la frontière de *A* intersecte l'intérieur de *B*; c'est le cas dans la Figure 9 puisque la frontière d'une des sous-régions de *A* intersecte l'intérieur d'au moins une sous-région de *B*.

Dans [SCH02b] [SCH06], les auteurs précisent que leur définition des objets complexes est assez similaire à celle donnée dans [GUT95]. Ainsi, l'algèbre "ROSE", présentée dans [GUT95], pourrait être utilisée comme implémentation du modèle de [SCH06].

2.2.2 TRCR (niveau gros-grain et niveau détaillé)

Dans leur modèle TRCR (Topological Relationship for Composite Regions), les auteurs de [CLE95] présentent des relations topologiques entre deux régions composites. Ils définissent ces relations à partir de celles de CBM (rappelées à la Définition 1).

Définition 3. Région composite [CLE95]
Une région composite est un sous-ensemble fermé A de R^2 tel que, si l'on définit les composants $A_1 \ldots A_n$ de A comme la fermeture des composants correspondants de A° (l'intérieur de A) alors :

- chaque A_i est une région simple, c'est-à-dire un ensemble fermé de points 2D connexe non-vide et sans trou
- $\forall\, i \neq j$, $A_i° \cap A_j° = \varnothing$ et
- $\delta A_i \cap \delta A_j = \varnothing$ ou égal à un nombre fini de points $\{p_1, ..p_k\}$ □

Par exemple, la Figure 10 présente une région composite faite de 4 composants. Deux d'entre eux, A_1 et A_2, partagent un point de leur frontière.

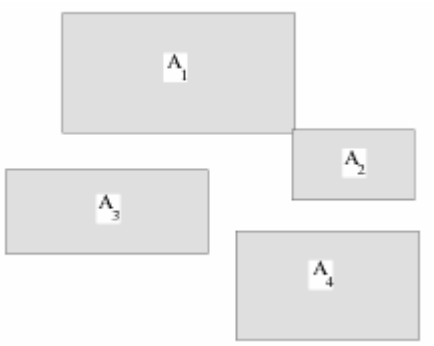


Figure 10. Une région composite faite de 4 composants

TRCR intègre deux niveaux de raffinement pour décrire les relations spatiales (gros-grain et détaillé).

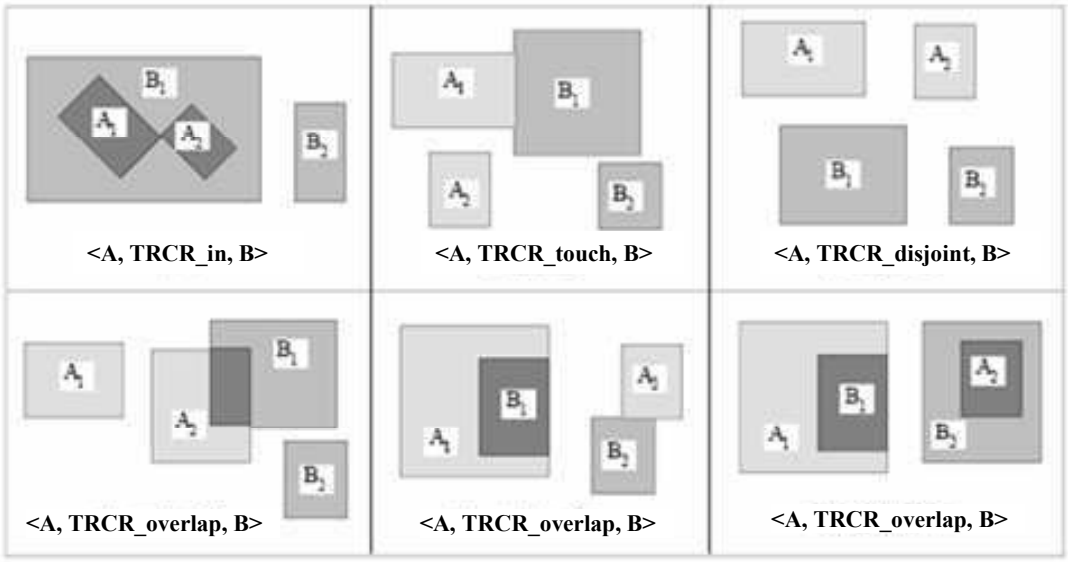


Figure 11. Exemple de relations topologiques issues de TRCR entre régions composites.

Au niveau gros-grain, TRCR fournit 4 relations topologiques définies à partir des relations de CBM (voir Tableau 1). En effet, dans le Tableau 1, *touch*, *disjoint*, *in* et *overlap* correspondent aux relations de la Définition 1. Les relations issues de TRCR sont illustrées sur la Figure 11.

Tableau 1. Sémantique des 4 relations de TRCR (niveau gros-grain) entre deux régions composites [CLE95]

Opérateurs TRCR	Sémantique sous forme logique de TRCR
⟨A, TRCR_touch, B⟩	∃i∈1..n, ∃j∈1..m | ∀r∈1..n, ∀s∈1..m, ⟨Ai, touch, Bj⟩ ∧ (⟨Ar, disjoint, Bs⟩ ∨ ⟨Ar, touch, Bs⟩)
⟨A, TRCR_in, B⟩	∀i∈1..n, ∃j∈1..m | ⟨Ai, in, Bj⟩
⟨A, TRCR_disjoint, B⟩	∀i∈1..n, ∀j∈1..m | ⟨Ai, disjoint, Bj⟩
⟨A, TRCR_overlap, B⟩	¬⟨A, TRCR_touch, B⟩ ∧ ¬⟨A, TRCR_in, B⟩ ∧ ¬⟨A, TRCR_disjoint, B⟩

Le niveau détaillé de TRCR permet de considérer une matrice *n*x*m* où *n* est le nombre de composants de la région composite *A*, et *m* est le nombre de composants de la région composite *B*. La matrice présente l'ensemble des relations qui existent entre les composants de *A* et les composants de *B*. Les nombres de composants de *A* et de *B* sont nécessaires pour écrire de telles matrices.

Plus précisément, une scène topologique est représentée au moyen d'une matrice dans laquelle l'élément en position (*i*, *j*) donne la relation qui existe entre la région (le composant) de la i-ème ligne(A_i) et la ré-

gion (le composant) de la j-ème colonne (B_j). Cette relation est une relation CBM et peut donc être : *O* (*overlap*), *M* (*meet*), *D* (*disjoint*), *I* (*in*) ou *J* (l'inverse de I, i.e. *contains*). La Figure 12 et le Tableau 2 exemplifient ce type de matrice.

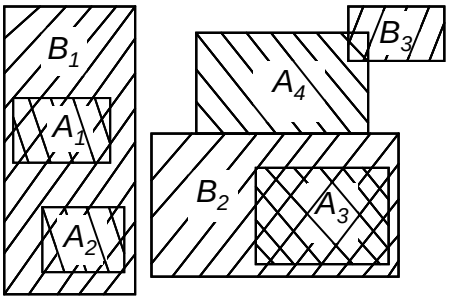


Figure 12. Une configuration topologique entre deux régions composites *A* et *B*

Tableau 2. La matrice topologique *nxm* pour les régions composites de la Figure 12 (*O=overlap*, *M=meet*, *D=disjoint*, *I=in*)

	B_1	B_2	B_3
A_1	I	D	D
A_2	I	D	D
A_3	D	I	D
A_4	D	M	O

□

2.2.3 Adverbes

Dans [CLA00], l'auteur propose de définir des relations topologiques entre des unions de régions à l'aide d'adverbes qualitatifs ; ceci permet par exemple de préciser qu'une union de régions contient "occasionnellement" une autre union de régions, ou bien encore, "fréquemment", "jamais", etc.

Définition 4. Union de Régions

Une union de régions UR est un ensemble fini de régions simples $R_1, \ldots, R_n$ de R^2 tel que:

1. $R_i^\circ \cap R_j^\circ = \emptyset, \forall\, i \neq j$
2. $\delta R_i \cap \delta R_j = \emptyset, \forall\, i \neq j$

La sémantique de chaque adverbe est présentée dans le Tableau 3. *relation_topo* est une des 8 relations topologiques de 9IM entre 2 régions simples (*inside*, *contains*, *covers*, *coveredBy*, *disjoint*, *equal*, *overlap*, *meet*).

mostly - A mostly topo_relationship B $\forall j \in 1..m, \exists i \in 1..n \mid \langle Ai, topo\_relationship, Bj \rangle$
mostly$_{rev}$ - A mostly$_{rev}$ topo_relationship B $\forall i \in 1..n, \exists j \in 1..m \mid \langle Ai, topo\_relationship, Bj \rangle$
completely - A completely topo_relationship B $(\forall j \in 1..m, \exists i \in 1..n \mid \langle Ai, topo\_relationship, Bj \rangle \wedge$ $(\forall i \in 1..n, \exists j \in 1..m \mid \langle Bj, topo\_relationshiprev, Ai \rangle)$
partially – A partially topo_relationship B $\exists i \in 1..n, \exists j \in 1..m \mid \langle Ai, topo\_relationship, Bj \rangle \wedge$ $(\forall r \in 1..n, \forall s \in 1..m \mid \langle Ar, disjoint, Bs \rangle$ $\vee \langle Ar, topo\_relationship, Bs \rangle)$
occasionally - A occasionally topo_relationship B $\exists i \in 1..n, \exists j \in 1..m \mid \langle Ai, topo\_relationship, Bj \rangle$
entirely - A entirely topo_relationship B $\forall i \in 1..n, \forall j \in 1..m, \mid \langle Ai, topo\_relationship, Bj \rangle$ $\wedge \langle Bj, topo\_relationshiprev, Ai \rangle$
never - A never topo_relationship B $\forall i \in 1..n, \forall j \in 1..m, \mid \neg\langle Ai, topo\_relationship, Bj \rangle$

Tableau 3. Sémantique des 7 adverbes [CLA00]

Ces adverbes sont illustrés avec la relation *meet* sur deux unions de régions UR_1 et UR_2 dans la Figure 13. Une union de région UR_a est composée de *n* régions simples UR_{a1}, … UR_{an}. On notera que "UR_1 *mostly meet* UR_2" est équivalent à "UR_2 *mostly*$_{rev}$ *meet* UR_1".

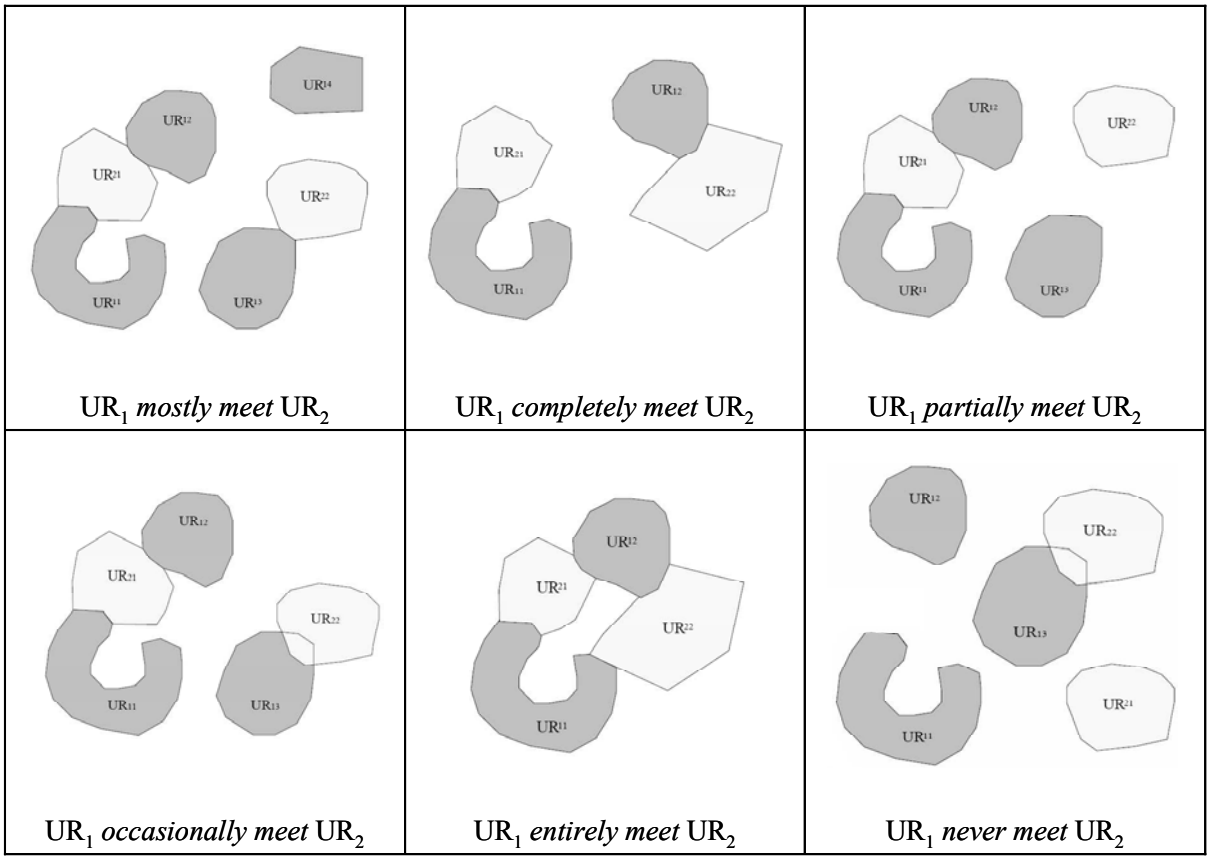


Figure 13. Les adverbes appliqués à la relation *meet* [CLA00]

2.3 Formalismes orientés-objets pour les SIG

Il existe plusieurs formalismes orientés-objets dédiés pour la modélisation de systèmes d'information géographiques : MADS, OMT-G, CONGOO, Perceptory, OMEGA**,** etc. Ces derniers intègrent entre autres des moyens pour décrire les différents types géographiques dans les modèles, et les relations spatiales entre les objets modélisés. Ces formalismes pourraient donc être utilisés pour représenter des contraintes spatiales. Nous réalisons un résumé des propositions de chaque formalisme sur ces deux aspects. Des comparatifs détaillés peuvent être trouvés dans [MIR06] [PIN02] [FRI01] [CLA97] [LBA97].

2.3.1 MADS

MADS (Model for Application Data with Spatio-temporal features) permet de décrire de façon détaillée des modèles conceptuels de données spatio-temporelles [PAR98]. Dans MADS, ce sont les attributs des classes d'objets qui peuvent avoir un type spatial tel que surface, point, ligne, ensemble d'objets, etc. [PAR98] [PAR99] [PAR99b] [PAR04] [PAR06]. Ce type spatial est représenté dans les diagrammes par un pictogramme. Comme décrit dans [PAR06], différentes contraintes topologiques entre les objets spa-

tiaux sont exprimables graphiquement sur les diagrammes ("*disjoint*", "*touch*", "*overlap*", "*cross*", "*within*", "*equal*"…).

La Figure 14 modélise une classe spatiale *Route*. Le pictogramme en haut à droite de cette classe implique que sa représentation graphique soit une ligne. Deux routes peuvent se "croiser". Cette contrainte topologique est modélisée par l'association réflexive *Croise* sur la classe *Route* annotée du pictogramme "*TopoCross*".

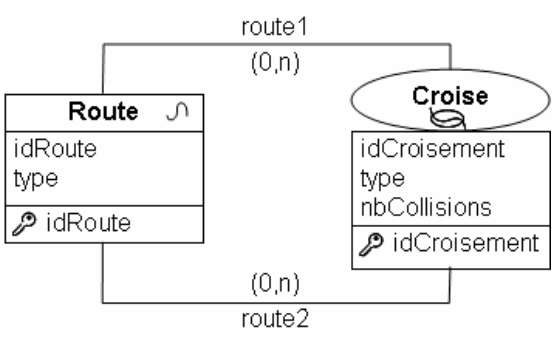


Figure 14. Schéma MADS d'une classe spatiale et spécification d'une contrainte topologique "*TopoCross*" réflexive sur cette classe [PAR06]

2.3.2 OMT-G

OMT-G se base sur OMT. Il permet de modéliser par des pictogrammes des classes dites géoréférencées. Chaque objet d'une classe géoréférencée représente des objets géographiques discrets (point, ligne, polygone, nœud, ligne unidirectionnelle ou ligne bidirectionnelle,…) ou des objets géographiques continus (réseau de triangles irréguliers, isolignes,...). OMT-G donne la possibilité d'exprimer dans les diagrammes, sous la forme de relations entre classes des contraintes spatiales prédéfinies (contenir, couvrir, traverser, être devant, être à gauche de, être à droite de,…) [BOR99] [BOR01]. Dans la Figure 15, les pictogrammes présents dans les classes indiquent qu'une commune est définie par un polygone et qu'une mairie est représentée par un point. La relation représentée par un trait pointillé entre les deux classes signifie qu'une commune contient une mairie.

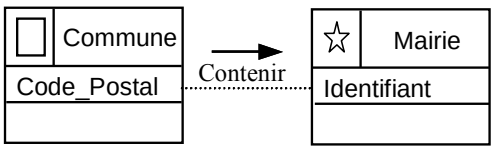


Figure 15. Exemple de relation topologique entre deux classes spatiales avec le formalisme OMT-G

2.3.3 CONGOO

CONGOO est un formalisme associé à la méthode MECOSIG (Méthode de Conception des Systèmes d'Information Géographique) qui est historiquement la première proposition de méthode orientée objets pour les SIG [PAN94] [PAN96]. MECOSIG repose sur OOA [COA91]. CONGOO signifie CONception Géographique Orientée Objets (CONGOO).

CONGOO permet de décrire les objets sous la forme : 1) d'OGS - Objet Géographique Simple, c'est-à-dire un point, une ligne, un polygone ou une image, 2) d'OGC - Objet Géographique Composé, c'est-à-dire une agrégation d'OGS de même type (exemple une agrégation de polygones), 3) d'OGX - Objet Géographique Complexe, qui est une agrégation d'objets de types différents. Par ailleurs en ce qui concerne les relations entre objets, CONGOO permet de décrire sur les modèles deux types de relations topologiques de base ; le voisinage (intersection de frontières) et la superposition (intersection des objets).

2.3.4 Perceptory

Perceptory est le nom d'un outil implantant une extension d'UML pour le spatial [BED99] [BED04]. Dans Perceptory, les types spatiaux sont associés aux classes d'objets et sont définis à l'aide de pictogrammes. Les pictogrammes peuvent être combinés afin de produire des types alternatifs, complexes, etc. On notera comme type spatial représentable avec Perceptory, les géométries suivantes :

- simple : les objets de classes sont associés à une et une seule primitive géométrique (point, ligne ou polygone),
- alternative : les objets de classes possèdent l'une ou l'autre des géométries proposées (deux géométries ou plus de façon mutuellement exclusive),
- multiple : chaque objet de classe possède autant de géométries qu'il y a de pictogrammes spécifiés pour la classe (c'est-à-dire deux géométries ou plus) mais dont une seule est habituellement utilisée à la fois dans une représentation cartographique ou une requête spatiale,
- complexe : chaque objet de classe est associé en même temps à deux ou plusieurs géométries.

L'exemple de la Figure 16 modélise une classe représentant les exploitations agricoles. Chaque instance d'exploitation agricole a une géométrie complexe qui peut être composée d'une ou plusieurs régions. Perceptory n'intègre pas spécifiquement de moyen pour formaliser les relations spatiales dans les diagrammes UML.

C'est sur le formalisme de Perceptory que reposent le profil UML-SIG et l'approche dirigée par les modèles pour la géomatique proposés par [MIR06].

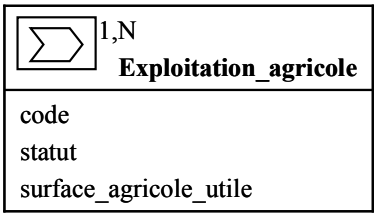


Figure 16. Modèle de classe Perceptory

2.3.5 OMEGA

La méthode basée sur UML nommée OMEGA pour "Object Modeling for End-user Geographical Applications" permet de définir quatre types de modèles : statiques, dynamiques, fonctionnels et plates-formes matérielles d'implémentation. Un formalisme spécifique est proposé pour modéliser les données géographiques dans les modèles statiques. Ainsi, selon les pictogrammes qui sont associés aux classes, trois types de références spatiales peuvent être associés aux objets d'une classe :

- géométrie simple (Point, Polyligne ou Polygone),
- géométrie complexe ; les objets de la classe sont associés à deux géométries simples,
- une géométrie alternative ; chaque objet de la classe est associé à une seule géométrie simple mais le type de cette géométrie doit être choisi entre deux types définis pour la classe.

OMEGA permet de représenter les contraintes topologiques de la même façon que CONGOO. L'Atelier de Génie Logiciel AIGLE supporte OMEGA et une extension de la méthode pour la modélisation d'applications géomatiques communicantes existe (T-OMEGA) [PIN02].

2.3.6 Bilan

L'aspect visuel de ces formalismes leur confère une simplicité d'utilisation et de compréhension. Leur utilisation principale n'est cependant pas dédiée à l'expression de contraintes. En effet, ceux qui fournissent des moyens de spécification de relations entre objets géographiques le font généralement via des relations entre classes dans les diagrammes.

Cette représentation graphique est très bien adaptée pour un certain niveau d'utilisation : elle permet d'exprimer les relations entre les entités spatiales de manière simple et lisible ce qui facilite son appropriation par les utilisateurs. Cependant, les langages graphiques peuvent être limités pour la spécification de contraintes très précises ou plus fines. Par exemple, on ne peut pas émettre de condition sur la relation to-

pologique exprimée en Figure 15 (formalisme OMT-G). Or il pourrait être intéressant de lier le nombre de mairies à la population dans certaines grandes agglomérations ayant plusieurs arrondissements.

Des outils très intéressants supportant ces formalismes existent. Néanmoins, le but premier de ces outils n'étant pas la modélisation de contraintes d'intégrité, ils ne proposent pas de générer à partir des modèles des mécanismes de contrôle d'intégrité des relations spatiales dans les bases de données.

Aussi, les formalismes présentés dans cette section permettent de décrire des types géométriques par des pictogrammes. Là encore, cette représentation permet une spécification simple et intuitive. Elle pourrait néanmoins être complétée afin de permettre d'exprimer des cas complexes, par exemple que le type géométrique des objets d'une classe dépend de la valeur d'un ou plusieurs attributs des objets. En effet, la représentation géographique d'une classe spatiale peut dépendre de la valeur d'un de ses attributs, notamment pour les communes dont la représentation peut être un point ou un polygone suivant sa population. On ne peut pas distinguer cette nuance avec les pictogrammes qui spécifieront seulement que la représentation de la commune est soit un point soit un polygone sans préciser sous quelles conditions.

Nous allons voir dans la section suivante les différents moyens existants pour spécifier des contraintes topologiques. L'un des objectifs de nos travaux est de proposer un moyen relativement simple pour spécifier des contraintes topologiques dès le niveau conceptuel d'une application, c'est-à-dire sans se soucier des outils utilisés.

2.4 Méthodes et outils pour la spécification de contraintes spatiales

Cette section présente les différentes solutions pour exprimer les contraintes d'intégrité topologiques ou spatiales.

Des outils autonomes ont été proposés durant la dernière décennie pour modéliser et contrôler l'intégrité spatiale. Ils peuvent être intégrés dans un système d'information géographique pour en gérer l'intégrité. Dans [SER00], les auteurs ont développé et utilisent une interface visuelle pour spécifier des contraintes topologiques entre les entités d'un système d'information géographique comme illustré en Figure 17. Le modèle utilisé dans cette interface permet de définir les 81 relations issues de 9IM étendu entre 2 objets [CLE94b] (9IM a été introduit en section 2.2.1). Certaines relations ont été regroupées en sous-ensembles nommés afin de simplifier l'utilisation de cette interface [UBE97]. Ces sous-ensembles partagent des caractéristiques communes. Par exemple, le sous-ensemble de relations "contient" entre deux régions re-

groupe 4 matrices des 9I qui diffèrent suivant que les frontières des deux régions s'intersectent ou non. Ces 4 matrices sont données dans la Figure 18.

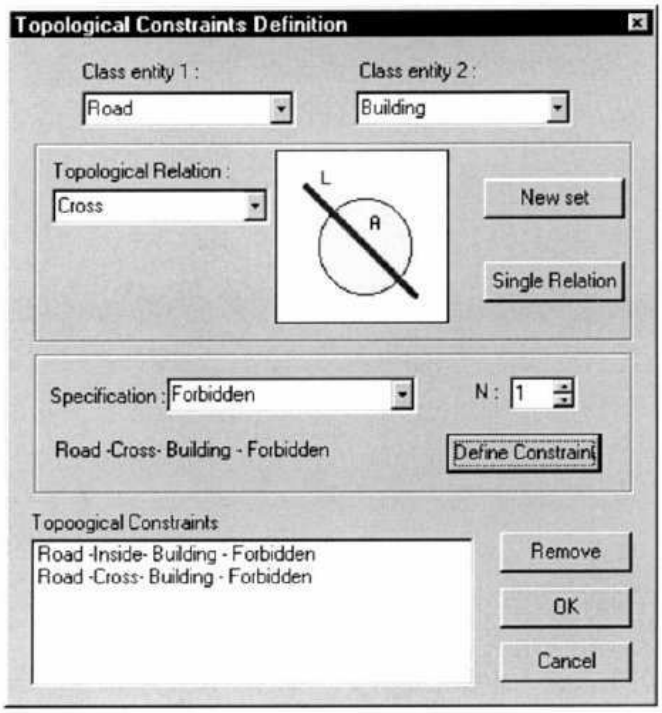


Figure 17. Interface de définition de contraintes topologiques [SER00]

Avec l'interface proposée, le cheminement pour spécifier une contrainte est le suivant :
(1) sélectionner la classe d'objets géographiques 1 dans le cadre "Class entity 1",
(2) sélectionner la classe d'objets géographiques 2 dans le cadre "Class entity 2",
(3) sélectionner le sous-ensemble de relations ou la relation topologique précise dans le cadre "Topological Relation",
(4) sélectionner la spécification dans la moitié inférieure de la fenêtre. Celle-ci peut être "interdit" comme sur l'exemple de la Figure 17, ou encore "exactement *N* fois" ou "au moins *N* fois" ou "au plus *N* fois".

Les contraintes basées sur la relation "disjoint" ne sont pas permises. Elles génèreraient un trop grand nombre d'incohérences et feraient exploser les méthodes de recherche d'erreur. Cette interface ne permet de spécifier que des relations "*N-N*", c'est-à-dire des contraintes qui doivent être vérifiées pour toutes les instances des deux classes géographiques considérées.

Sur la Figure 17, deux contraintes ont été définies entre *Route* (*Road*) et *Bâtiment* (*Building*). La situation où une route est dans un bâtiment est interdite (*Road-Inside-Building-Forbidden*).Autrement dit, le sous-

ensemble de relations "dans" est interdit entre *route* et *bâtiment*. De même, le sous-ensemble de relations "croise" est interdit entre *route* et *bâtiment* (*Road-Inside-Building-Forbidden*).

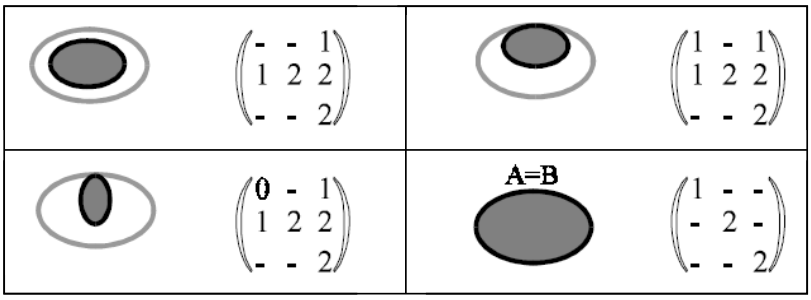


Figure 18. Sous-ensemble de relations "contient"

Dans [COC98] [COC01], l'auteur présente un outil de principe similaire. La principale différence avec la proposition de [SER00] réside dans le fait que les contraintes peuvent aussi être spécifiées en fonction de la valeur des attributs. Par exemple, dans la Figure 19, l'utilisateur entre une contrainte topologique qui dépend d'une condition sur la valeur d'un attribut : une canalisation (*pipe*) ne doit pas intersecter une valve (*ButterflyValve*) si le diamètre de cette canalisation est de plus de 14 centimètres.

Ces interfaces permettent de spécifier de façon simple des contraintes topologiques dans les systèmes d'information géographique. La proposition de [COC98] introduit des possibilités de spécification de contraintes régies par la valeur des attributs. Cependant, certaines contraintes plus complexes ne sont pas gérées par cet outil. Par exemple, "le nombre d'objets intersectant un objet de la classe *A* est égal à 3" ; ou bien, "si un objet de la classe *A* contient spatialement un objet de la classe *B*, alors l'objet *A* doit aussi être adjacent à un objet de la classe *C*".

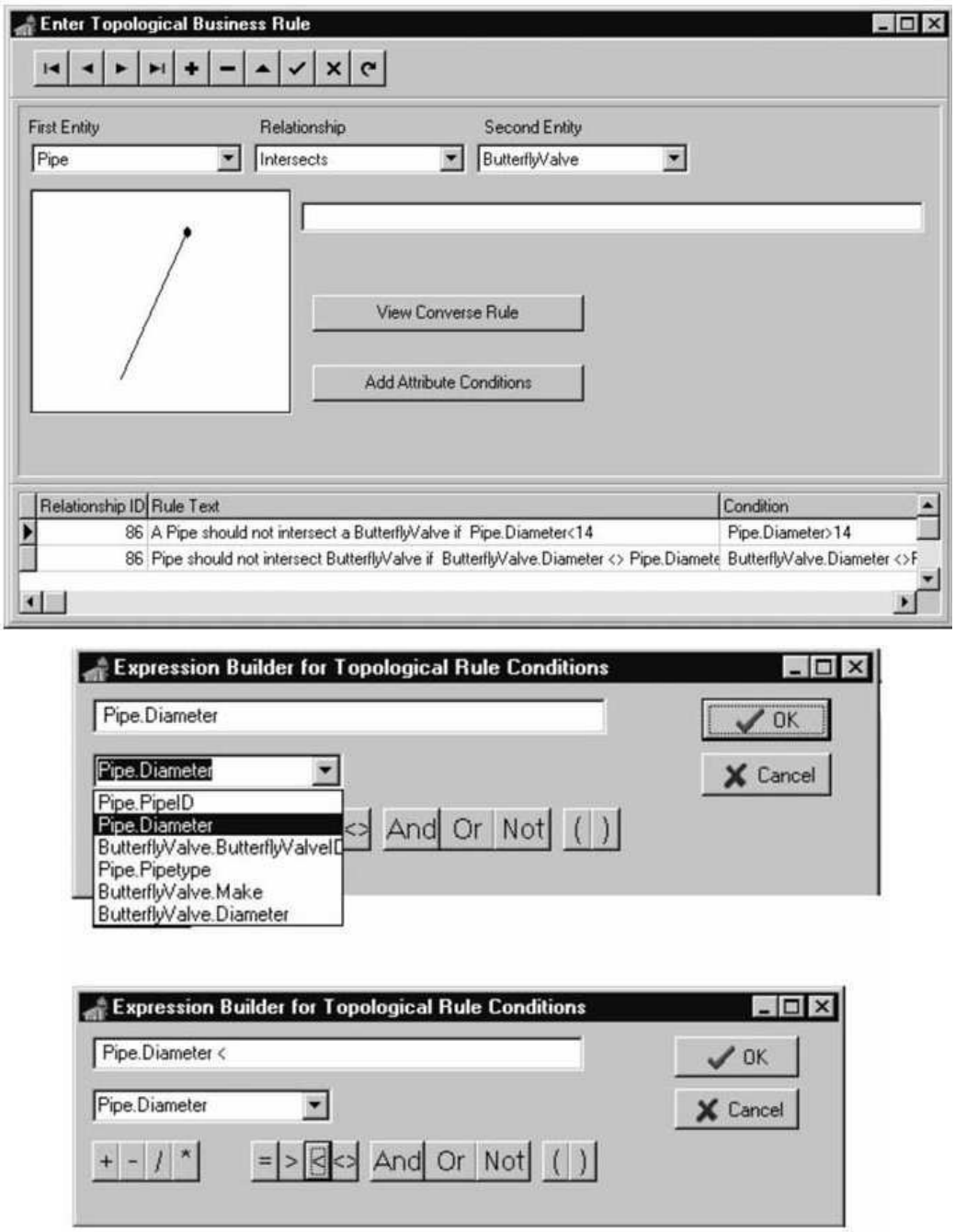


Figure 19. Spécification d'une règle de gestion topologique [COC98]

2.5 Formalismes et langages expressifs pour la spécification de contraintes

Nous présentons ici les formalismes et les langages particulièrement expressifs supportés par des outils pour spécifier des contraintes d'intégrité.

2.5.1 Formalismes

A un autre niveau, pour définir des contraintes d'intégrité expressives, l'utilisation de "triggers" peut être envisagée [BAU95]. Ces triggers par exemple écrits en SQL (étendu au spatial) se déclenchent lors d'une modification de la base de données pour vérifier une contrainte. Cependant, l'écriture directe des triggers reste lourde. SQL est un langage de requête et non un langage déclaratif de contraintes. De plus, les triggers portent sur le schéma physique des bases. Ils ne peuvent donc pas être écrits dès le niveau conceptuel, à moins d'envisager l'utilisation d'un outil de plus haut niveau pour les générer à partir d'une spécification conceptuelle. Dans [BEL00], les auteurs proposent un système qui utilise les contraintes d'intégrité spatiales pour diriger les modifications faites par l'utilisateur. Ce système a été implémenté pour une catégorie de contraintes appelées les STIC (*Simple Topological Integrity Constraints*). Cette implémentation demande une bonne connaissance du système, elle n'est pas orientée utilisateur.

La méthode B est une méthode formelle de développement du logiciel. Elle permet de spécifier des contraintes sur le comportement des logiciels. Il s'agit d'un formalisme déclaratif ; elle permet de "parler du quoi, pas du comment" [ABR96]. Elle ne couvre pas les aspects initiaux de la programmation, c'est-à-dire le cahier des charges et la description des besoins. Bien qu'elle soit applicable à des schémas conceptuels de bases de données, l'écriture de contraintes en B reste dédiée à des utilisateurs confirmés. Des outils visuels permettent quant à eux d'exprimer des contraintes, mais leur expressivité peut se montrer limitée. Aussi, B reste relativement déconnecté des langages usuels (notamment UML) et des Ateliers de Génie de Logiciel utilisés aujourd'hui dans la conception de base de données et de systèmes d'information. Aucune extension de B n'a été proposée pour le spatial.

2.5.2 Langages

OCL (Object Constraint Language) est un langage formel pour l'expression de contraintes appliquées à des diagrammes UML [OMG05] [WAR99]. OCL fait partie intégrante d'UML depuis la version 1.1 de 1999. OCL est un langage déclaratif : il spécifie les conditions qui doivent être satisfaites. C'est un langage sans effet de bord : l'évaluation d'une contrainte ne modifie pas l'état du système. OCL a été conçu pour être à la portée des ingénieurs concepteurs de systèmes d'information.

OCL a été développé par un groupe de scientifique d'IBM autour de 1995 au cours d'un projet de modélisation d'entreprise. Le langage a été influencé par Syntropy, un langage orienté-objet [COO94]. Un nombre grandissant de concepteurs de systèmes d'information utilise ce langage de contraintes en complément des diagrammes de classes d'UML pour exprimer les invariants des modèles par exemple.

Dans [MAN99] et [AKE01], les possibilités d'OCL sont étudiées pour exprimer des requêtes ; dans ces articles, l'expressivité d'OCL en temps que langage de requête est alors étudiée. Les auteurs montrent qu'en intégrant la notion de tuples à OCL, le langage devient équivalent au calcul relationnel. Un nombre grandissant d'outils supportant OCL existent [KLA05]. On trouve aujourd'hui des plug-ins OCL pour Eclipse, Rational Rose, ArgoUML… Dans le contexte des bases de données, un avantage important apporté par OCL est que les contraintes sont exprimées au niveau conceptuel (sur le schéma conceptuel), ce qui simplifie grandement leur expression. De toutes premières idées d'extension d'OCL pour exprimer des contraintes d'intégrité spatiales ont été introduites dans [PIN02].

Notre proposition étant basée sur OCL, nous introduisons dans la partie suivante les principaux concepts de ce langage.

2.6 OCL

Une contrainte OCL peut se rapporter à une classe, appelée le contexte de la contrainte. Trois stéréotypes OCL sont applicables à une contrainte :

- *inv* pour la spécification d'invariants, i.e. d'une condition qui doit être vraie pour chaque instance de la classe en permanence [SCH02],
- *pre* pour la spécification de pré-conditions des opérations, i.e. d'expressions qui permettent de spécifier des contraintes qui doivent être vérifiées avant l'appel d'une opération,
- *post* pour la spécification de post-conditions des opérations, i.e. d'expressions qui permettent de spécifier des contraintes qui doivent être vérifiées après l'exécution d'une opération.

Dans le cadre de nos travaux sur l'intégrité des données, nous nous intéresserons aux invariants.

Les contraintes suivantes, exprimées sur le diagramme de la Figure 20 présentent les concepts de base d'OCL.

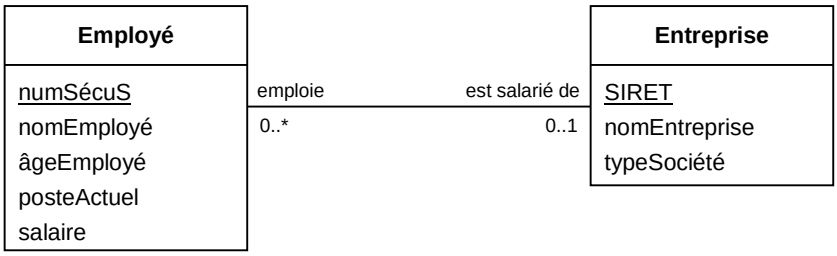


Figure 20. Extrait de Modèle Conceptuel de Données (MCD) d'un système de gestion d'entreprise.

La contrainte OCL suivante exprime le fait que tous les employés dont le poste actuel est chercheur perçoivent un salaire supérieur ou égal à 1500€.

```
context Employé inv:
self.posteActuel = 'chercheur' implies salaire >= 1500
```

`self` référence une instance du contexte, i.e. l'élément du modèle sur lequel porte la contrainte (ici une instance de la classe `Employé`). `implies` indique que la vérification de la condition sur la valeur de l'attribut `posteActuel` implique une contrainte sur la valeur de l'attribut `salaire` qui devra alors être supérieur ou égal à 1500€.

- **Navigation entre les classes et opérations sur les collections**

La navigation permet, à partir d'un objet spécifique, de se déplacer dans le modèle. Par exemple, la contrainte suivante signifie que les petites et moyennes entreprises (PME) n'ont pas plus de 499 salariés.

```
context Entreprise inv:
self.typeSociété = 'PME' implies self.Employé -> size() <= 499
```

La navigation via une association se fait à l'aide de la notation pointée. Ainsi, `self.Employé` retourne la collection des instances de la classe `Employé` accessibles par l'association depuis l'instance `self` de la classe `Entreprise`. L'opération `size()` retourne la taille d'une collection (i.e. le nombre d'objets la composant).

Dans une spécification de contrainte OCL, les noms de rôle des associations peuvent aussi permettre la navigation. Par exemple, il est aussi possible d'écrire la contrainte précédente de la façon suivante ("Employé" a été remplacé par "emploie").

```
context Entreprise inv:
self.typeSociété = 'PME' implies self.emploie -> size() <= 499
```

D'autres opérations existent sur les collections. Par exemple, `size(Objet)` retourne le nombre d'occurrences d'`Objet` dans `self`. L'opération `isEmpty()` retourne vrai si `self` est vide.

- **Opérations sur les éléments d'une collection**

La syntaxe d'une opération portant sur les éléments d'une collection est la suivante :

⟨collection⟩ -> ⟨opération⟩ ([⟨itérateur⟩ |] ⟨expression⟩)

où l'expression booléenne ⟨expression⟩ est évaluée pour chacun des éléments de la ⟨collection⟩. Le résultat dépend de l'⟨opération⟩. ⟨itérateur⟩ est facultatif et sert de référence à l'élément courant dans l'⟨expression⟩.

⟨opération⟩ est l'une des opérations suivantes :

- `exists` représente le quantificateur existentiel qui vérifie si une expression booléenne est vraie pour au moins un élément de ⟨collection⟩.

Par exemple, on peut exprimer le fait qu'il existe (un ou) plusieurs chercheurs dans un établissement de recherche.

```
context Entreprise inv:
self.type='établissement de recherche'
implies self.Employé->exists(poste_actuel='chercheur')
```

- `forAll` représente le quantificateur universel qui vérifie si une expression booléenne est vraie pour toutes les instances de ⟨collection⟩.

Par exemple, on peut spécifier le fait que tous les employés des établissements de recherche ont plus de 18 ans.

```
context Entreprise inv:
self.type='établissement de recherche'
implies self.Employé->forAll(age>=18)
```

- `collect` retourne une nouvelle collection de même taille mais dont les éléments sont généralement d'un type différent que ceux de ⟨collection⟩.

Par exemple, on peut l'utiliser pour collecter dans une contrainte, tous les postes des employés d'une société :

```
self.Employé -> collect (poste_actuel)
```

- `size` retourne le nombre d'éléments d'une collection.

- `select` (resp. `reject`) retourne un sous-ensemble de ⟨collection⟩ qui ne contient que les éléments qui vérifient (resp. ne vérifient pas) l'⟨expression⟩.

Par exemple, on peut l'utiliser pour sélectionner dans une contrainte, les employés majeurs :

```
self.Employé -> select (age>=18)
```

On utilise `allInstances` pour appliquer une opération à toutes les instances d'une classe.
`Employé.allInstances` retourne la collection de toutes les instances d'employés.

Des opérations sur les types d'OCL sont aussi prédéfinies, notamment `oclIsTypeOf(t)`.
Par exemple, `o.oclIsTypeOf(t)` retourne vrai si le type de `t` et le type de l'objet `o` à laquelle l'opération est appliquée sont les mêmes.

Chapitre 3
Extensions spatiales d'OCL

Afin de permettre une définition de contraintes d'intégrité spatiales à la fois expressives et déclaratives, notre proposition consiste à effectuer une extension d'OCL. Ce langage est adapté au formalisme de modélisation le plus répandu, UML. Le méta-modèle d'UML est d'ailleurs en grande partie défini avec le langage OCL. Ainsi, on peut référencer facilement dans les contraintes exprimées en OCL toutes les notations d'UML, comme les associations navigables, les multiplicités, etc. OCL a une syntaxe et une sémantique formelles ; par conséquent, son interprétation est non-ambiguë. Enfin, il existe des implémentations d'OCL pour les bases de données comme OCL2SQL qui est en source libre.

OCL est utilisé pour modéliser un large éventail de contraintes dans les systèmes d'informations et les applications informatiques (contraintes d'intégrité dans les bases de données, pré- et post- conditions des opérations des applications associées,...). Utiliser un seul langage pour décrire tous les types de contraintes permet de renforcer la lisibilité et la cohérence des spécifications du système.

Avant de pouvoir définir des contraintes entre plusieurs objets géométriques, les types spatiaux doivent être intégrés dans OCL. Ainsi, avant d'ajouter les opérateurs topologiques, nous proposons une extension nommée OCL Spatial. Cette première proposition permet d'exprimer des contraintes sur le type des objets spatiaux.

A cette extension, on intègre ensuite des opérations topologiques afin de spécifier des contraintes entre des régions spatiales. Les relations topologiques entre régions issues des modèles de 9IM (9-Intersection Model) et CBM (Calculus-Based Model) sont actuellement supportées par des SGBD (cf. section 2.1). Nous proposons donc en parallèle, deux grandes extensions d'OCL intégrant ces deux groupes de relations : respectivement OCL_{9IM} et OCL_{CBM}.

Enfin, nos 3 dernières propositions (OCL_{TRCR}, OCL_{ADV} et P-OCL) intègrent des relations topologiques entre des régions composites, i.e. des géométries faites de plusieurs régions simples. D'une part, OCL_{TRCR} et P-OCL sont des extensions d'OCL_{CBM} et d'autre part, OCL_{ADV} est une extension d'OCL_{9IM}.
La hiérarchie entre ces diverses propositions est schématisée en Figure 21.

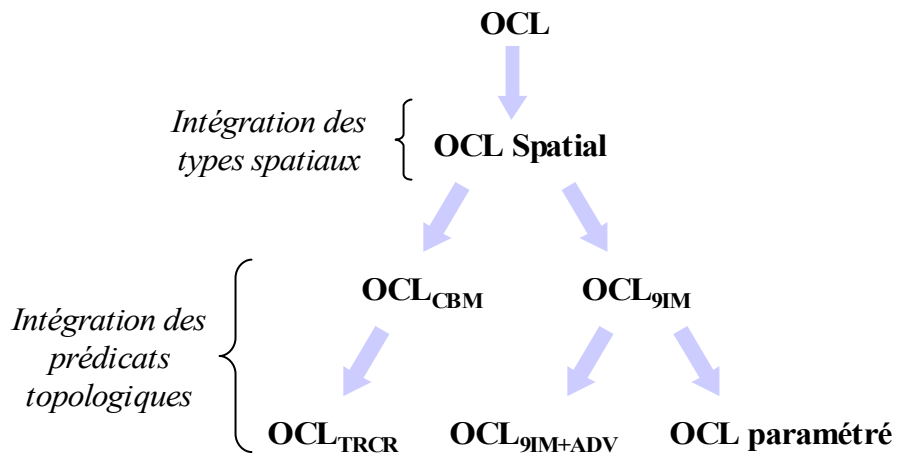


Figure 21. Les extensions spatiales d'OCL proposées

3.1 Ajout de types spatiaux dans OCL : OCL Spatial

Dans un premier temps, nous proposons une extension d'OCL nommée "OCL Spatial" intégrant 4 nouveaux types dans le méta-modèle d'OCL : *Point*, *Polyline*, *Region* et *BasicGeoType* qui généralisent les 3 types précédents (Figure 22). L'utilisation de ces 4 types spatiaux est également possible avec les collections d'OCL : *Set*, *OrderedSet*, *Bag ou Sequence*[1] comme exemplifié dans la Figure 23 ; dans cette figure, l'attribut *geo* est un ensemble d'objets de type *BasicGeoType*.

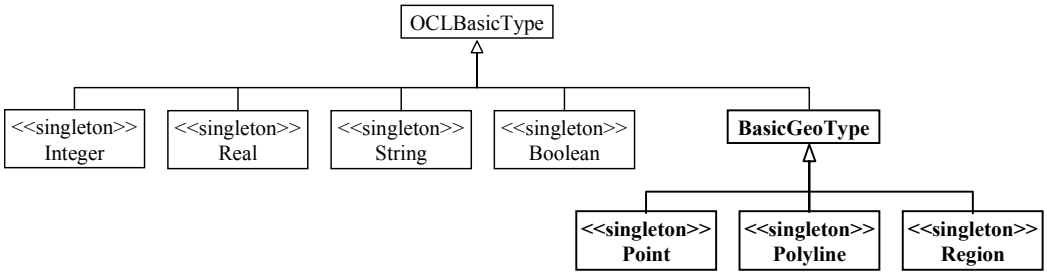


Figure 22. Types basiques d'OCL Spatial

Avec OCL spatial, nous pouvons exprimer des contraintes sur les types spatiaux. Nous pouvons par exemple exprimer des contraintes sur les types et leur cardinalité.

Ainsi la classe Pays (Figure 23) associe à chaque instance une géométrie composée de plusieurs régions et/ou points (pour modéliser les îles par exemple). Cette géométrie est stockée dans un attribut "geo" de

type Set(BasicGeoType). A la classe Pays correspondent les 2 contraintes OCL suivantes, la première portant sur les types et la seconde sur les cardinalités de l'ensemble des éléments de chaque Pays.

```
(1)  context Pays inv:
     self.geo -> forAll(oclIsTypeOf(Region)
     or oclIsTypeOf(Point))

(2)  context Pays inv:
     self.geo -> size() >= 1
```

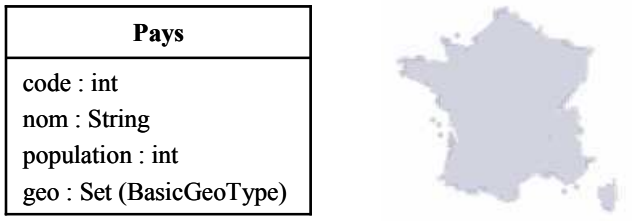


Figure 23. Classe Pays modélisée avec UML et illustration d'une instance de la classe

OCL Spatial autorise également la définition de types complexes, utilisés par exemple dans le cas des représentations multiples. Considérons la classe de la Figure 24, une ville est représentée par un point ou un ensemble de plusieurs polygones en fonction de l'étendue de la ville. L'invariant OCL suivant traduit le même type de données spatiales.

```
context Ville inv:
(self.geo -> forAll(oclIsTypeOf(Region))
and self.geo -> size() >= 1)
or (self.geo -> forAll(oclIsTypeOf(Point))
and self.geo -> size() =1)
```

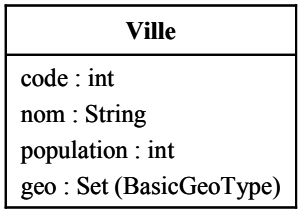


Figure 24. Classe Ville

[1] C'est-à-dire ensemble, ensemble ordonné, ensemble avec répétition possible des éléments, ensemble ordonné avec répétition possible des éléments.

OCL Spatial permet de spécifier des contraintes sur les types géographiques. Cependant, il ne propose pas d'opérateur pour exprimer des contraintes portant sur les relations entre des objets spatiaux. La seconde étape pour étendre OCL à la gestion de contraintes spatiales est de l'enrichir d'opérateurs spécifiques. Potentiellement, ces opérateurs pourraient être de 3 types :

- les opérateurs topologiques, qui sont invariants sous les relations topologiques comme les rotations ou les translations. Les relations topologiques décrivent les relations de voisinage qui s'établissent entre des figures géométriques. Elles ne sont pas altérées par la déformation des figures.
- les opérateurs d'orientation, qui présupposent l'existence d'un espace vectoriel.
- les opérateurs de distance, ou métriques.

Dans cette thèse, nous traiterons uniquement le cas des opérateurs topologiques.

3.2 Gestion des relations topologiques entre régions simples

Cette section traite de l'intégration des relations entre régions simples issues des 2 principaux modèles 9IM et CBM. Nous ne considérons que les régions simples sans trous.

3.2.1 Intégration des relations identifiées par 9IM dans OCL : OCL_{9IM}

OCL_{9IM} est le langage OCL auquel nous avons ajouté les 8 relations d'Egenhofer entre régions simples décrites à la Figure 6. L'objectif de ce langage est de fournir un moyen relativement simple et clair sémantiquement parlant pour permettre aux concepteurs de bases de données de modéliser des contraintes spatiales précises sur un schéma conceptuel modélisé avec UML.

Définition 5. OCL_{9IM}

Chaque prédicat topologique de 9IM est intégré dans OCL Spatial sous la forme de 8 nouveaux opérateurs. La syntaxe générale de ces opérateurs spatiaux est décrite ci-dessous.

```
geoA.opérateur_topo(geoB): Boolean
```

où *opérateur_topo* ∈ { *inside*, *contains*, *covers*, *coveredBy*, *disjoint*, *equal*, *overlap*, *meet* } et *geoA*, *geoB* sont de type *Region*.

geoA, *geoB* sont les paramètres de l'opération. Cette opération retourne vrai ou faux (un booléen) suivant que la relation topologique entre *geoA* et *geoB* est vraie ou fausse. □

Exemple 4.

Par exemple, une station d'épuration (*STEP*) doit être localisée dans la commune qui lui correspond. En OCL_{9IM}, cette contrainte s'écrit comme suit :

```
context STEP inv :
(self.geo).inside(self.code_commune.geo)
or (self.geo).coveredBy(self.code_commune.geo)
```

La contrainte doit être vérifiée pour chaque instance *self* de *STEP*. L'expression *self.code_commune* retourne l'instance de *Commune* associée à *self* par l'association *code_commune*. □

3.2.2 Intégration de CBM dans OCL : OCL_{CBM}

OCL_{CBM} est le langage OCL Spatial auquel nous avons ajouté les 4 relations de base entre deux régions de CBM (voir les relations 1 à 4 de la Définition 1).

Définition 6. OCL_{CBM}

Chaque prédicat topologique de CBM est intégré dans OCL Spatial sous la forme de 5 nouveaux opérateurs. La syntaxe générale de ces opérateurs spatiaux est décrite ci-dessous.

```
geoA.opérateur_topo (geoB):Boolean
```

où *opérateur_topo* ∈ { *touch*, *in*, *overlap*, *disjoint*, *cross* }

et *geoA*, *geoB* sont de type *Region*.

geoA, *geoB* sont les paramètres de l'opération. Cette opération retourne vrai ou faux (un booléen) suivant que la relation topologique entre *geoA* et *geoB* est vraie ou fausse. □

Exemple 5.

Reprenons l'Exemple 4. Une station d'épuration (*STEP*) est localisée dans la commune qui lui correspond. En OCL_{CBM}, cette contrainte s'écrit comme suit :

```
context STEP inv:
(self.geo).in (self.code_commune.geo)
```

La contrainte doit être vérifiée pour chaque instance *self* de *STEP*. L'expression *self.code_commune* retourne l'instance de *Commune* associée à *self* par l'association *code_commune*. □

3.3 Gestion des relations topologiques entre régions composites

Certaines applications nécessitent la manipulation d'objets ayant une géométrie composée de plusieurs régions simples. Il est donc intéressant de pouvoir définir des relations topologiques sur ce type d'objets.

3.3.1 Régions composites

Nous donnons une définition des régions composites que nous utiliserons dans toutes les extensions d'OCL que nous proposerons dans cette section.

Définition 7. Région composite
Une région composite est un ensemble fini non vide CR = $\{R_1, ..., R_i, ..., R_n\}$ avec R_i une région simple sans trou, appelée composant de CR. $\forall\ i \neq j$, $R_i° \cap R_j° = \varnothing$ et $\delta R_i \cap \delta R_j = \varnothing$.

Nous supposons pour la suite que les objets de type *Set(Region)* sont des régions composites respectant cette définition.

3.3.2 Approches basées sur 9IM

OCL_{9IM} permet la spécification de contraintes topologiques entre des objets géométriques simples. Or une région composite peut être vue comme un ensemble de régions simples. Par conséquent, les opérations standards sur les ensembles d'OCL peuvent être utilisées pour accéder à leurs composants qui sont des régions simples. Ainsi, les opérations standards sur les ensembles d'OCL comme *size*, *forAll*, *exists* ou *select* peuvent être appliquées sur des attributs de type *Set(Region)*.

Définition 8. Écriture sous forme "logique" (OCL_{9IM} couplé aux opérations sur les collections d'OCL appliquées aux géométries composées)
La syntaxe générale des opérations sur les collections est :

```
géométrie -> opération_sur_collection(...)
```

où *géométrie* est de type *Set(Region)*. □

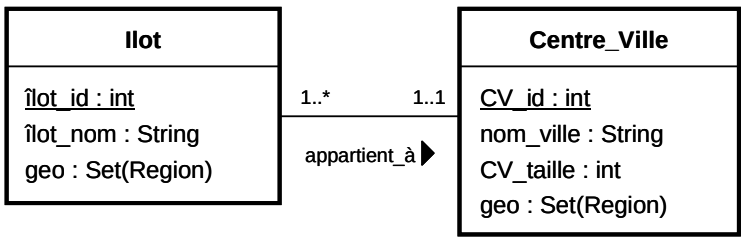


Figure 25. Diagramme de classe du centre ville et de ses îlots de bâtiments

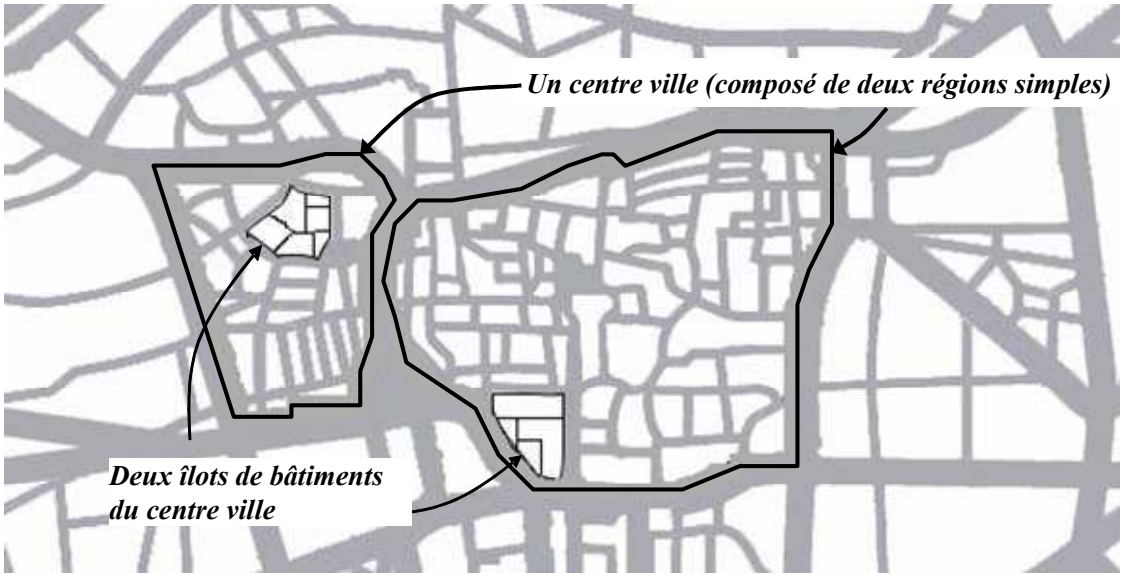


Figure 26. Exemple du centre ville et de ses îlots de bâtiments

Ainsi, il devient possible d'exprimer les relations topologiques entre les régions composites en intégrant les 8 relations de la Figure 6 dans OCL, et en utilisant les opérations sur les ensembles d'OCL. Par exemple, dans le modèle de la Figure 26, les attributs *geo* de la classe *Ilot* et de la classe *Centre_Ville* sont de type *Set(Region)*. Cela signifie, que chacun de ces attributs stockent une région composite. Par conséquent, les opérations d'OCL sur les ensembles peuvent être appliquées à ces attributs, comme illustré dans les exemples suivants (basés sur le diagramme de la Figure 26). Ces classes sont exemplifiées sur la Figure 26.

Exemple 6. Soit *I* un îlot et *C* un centre ville. Si *I* est *C* sont associés par "*appartient_à*" alors pour chaque bâtiment *b* de *I*, il doit exister une partie *c* du centre ville *C* tel que *b* soit dans *c*. Cette contrainte s'exprime en OCL_{9IM} :

```
context Ilot inv:
self.geo -> forAll( b |
self.Centre_Ville.geo -> exists(c | b -> inside(c)))
```

Les attributs *geo* sont de type *Set(Region)* et donc *b* et *c*, sont de type *Region*. □

Nous définissons un autre langage appelé OCL_{ADV} basé sur 9IM et les travaux de [CLA00]. Dans [CLA00], l'auteur définit 7 adverbes à associer aux opérateurs spatiaux de 9IM pour les appliquer sur des géométries composées : *mostly*, $mostly_{rev}$, *completely*, *partially*, *occasionally*, *entirely* et *never*. La sémantique de ces 7 adverbes est donnée sous forme logique dans le Tableau 3 de la Section 2.3.3.

Définition 9. OCL_{ADV}

La syntaxe générale des relations topologiques d'OCL_{ADV} est la suivante :

```
geoA -> opérateur_topo("adverbe", geoB):Boolean
```

où *opérateur_topo* est l'un des 8 opérateurs de 9IM (Figure 6), *adverbe* est l'un des 7 adverbes et *geoA* et *geoB* sont de type *Set(Region)*. Ces opérations retournent vrai ou faux (un booléen) suivant que la relation topologique 9IM appliquées entre les 2 géométries avec l'adverbe est vraie ou fausse. □

Exemple 7.

La contrainte de l'exemple précédent peut s'écrire plus directement en OCL_{ADV} sans utiliser les opérateurs existentiel et universel.

```
context Downtown_Building_Lot inv:
self.geometry->inside("mostlyRev",self.Downtown.geometry)
```
□

3.3.3 Approches basées sur CBM

OCL_{CBM} comme OCL_{9IM} permet la spécification de contraintes topologiques entre des objets géométriques simples. En effet, dans OCL_{CBM}, les opérations standards sur les ensembles d'OCL peuvent être utilisées pour accéder à leurs parties qui sont des régions simples. Ainsi, les opérations standards sur les ensembles d'OCL comme *size*, *forAll*, *exists* ou *select* peuvent être appliquées sur des attributs de type *Set(Region)*.

Définition 10. Écriture sous forme "logique" (OCL_{CBM} couplé aux opérations sur les collections d'OCL appliquées aux géométries composées)

La syntaxe générale est :

```
géométrie -> opération_sur_collection(...)
```

où *géométrie* est de type *Set(Region)*. □

Exemple 8. La contrainte de l'Exemple 6 s'exprime en OCL_{CBM} :

```
context Ilot inv:
self.geo -> forAll( b | self.Centre_Ville.geo -> exists( d | b->in (d)) )
```

Les attributs *geo* sont de type *Set(Region)* et donc, *b* et *c* sont de type *Region*. Il est à noter que dans le cas des régions, la relation "*in"* de CBM inclut aussi la relation de couverture de 9IM. □

Dans [CLE95], les auteurs définissent aussi une sémantique par défaut pour chacun des 4 opérateurs topologiques de CBM appliqués entre 2 régions composites (Tableau 4).

Tableau 4. Rappel de la sémantique sous forme logique des 4 opérateurs de TRCR niveau gros-grain applicables entre 2 régions composites [CLE95]

Opérateurs TRCR	Sémantique sous forme logique de TRCR
⟨A, TRCR_touch, B⟩	∃i∈1..n, ∃j∈1..m | ∀r∈1..n, ∀s∈1..m, ⟨Ai, touch, Bj⟩ ∧ (⟨Ar, disjoint, Bs⟩ ∨ ⟨Ar, touch, Bs⟩)
⟨A, TRCR_in, B⟩	∀i∈1..n, ∃j∈1..m | ⟨Ai, in, Bj⟩
⟨A, TRCR_disjoint, B⟩	∀i∈1..n, ∀j∈1..m | ⟨Ai, disjoint, Bj⟩
⟨A, TRCR_overlap, B⟩	¬⟨A, TRCR_touch, B⟩ ∧ ¬⟨A, TRCR_in, B⟩ ∧ ¬⟨A, TRCR_disjoint, B⟩

OCL_{TRCR} est OCL Spatial auquel nous avons ajouté les 4 opérateurs de TRCR niveau gros-grain applicables entre 2 régions composées.

Définition 11. OCL_{TRCR}

Chaque opérateur topologique de TRCR niveau gros-grain applicable entre 2 régions composées est intégré dans OCL Spatial sous la forme de 4 nouveaux opérateurs. La syntaxe générale de ces opérateurs spatiaux est décrite ci-dessous.

```
geoA -> opérateur_topo (geoB):Boolean
```

où *opérateur_topo* ∈ { *TRCR_touch*, *TRCR_in*, *TRCR_disjoint*, *TRCR_overlap* } et *geoA*, *geoB* sont de type *Set(Region)*.

Cette opération retourne vrai ou faux (un booléen) suivant que la relation topologique TRCR entre *geoA* et *geoB* est vraie ou fausse. Elle retourne faux si l'une des géométries est un ensemble ne comportant aucun élément. □

Exemple 9.

La contrainte de l'Exemple 6 s'écrit en OCL_{TRCR} :

```
context Ilot inv:
self.geo -> TRCR_in (self.Centre_Ville.geo)
```

Nous proposons maintenant une approche quantitative pour exprimer les relations topologiques. Toujours en nous basant sur CBM : P-OCL.

Définition 12. Parametric-OCL (P-OCL)

P-OCL intègre dans OCL_{CBM} de nouvelles opérations. Leur syntaxe générale est la suivante :

```
geoA -> relation_topo (geoB, condA, condB):Boolean
```

où *geoA* et *geoB* sont de type *Set(Region)*. *relation_topo* est l'une des 4 relations topologiques de CBM entre deux régions (*touch*, *in*, *disjoint*, *overlap*).

Cette opération retourne vrai ou faux (un booléen) suivant que les nombres de parties des deux régions composées impliquées dans la relation topologique respectent les conditions spécifiées dans *condA* et *condB* ou non. Autrement dit, l'opération retourne vrai si le nombre de composants de *geoA* (resp. *geoB*) tel qu'il existe un composant de *geoB* (resp. *geoA*) tel que *relation_topo(geoA, geoB)* respecte la condition *condA* (resp. *condB*). Cette condition est la concaténation d'un opérateur booléen, comme "<", "=" ou ">=", et un nombre ou le symbole "*". Le symbole "*" représente le nombre total de parties de la région composée considérée. □

Exemple 10. Reprenons l'Exemple 6 en P-OCL :

```
context Ilot inv:
self.geo -> in (self.Centre_Ville.geo, "=*", ">=1")
```
□

Exemple 11.

Précisons maintenant avec P-OCL que la ville de Montluçon a un centre ville composés de 2 sous-zones (= 2 composants) de 20 bâtiments chacun. Chaque îlot n'appartient qu'à une seule sous-zone de centre ville.

```
context Centre_Ville inv:
self.nom_ville = 'Montluçon' implies
self.geo -> size = 2 and
self.geo -> contains (self.Ilot.geo, "=1", "=20")
```

Plus précisément, il est spécifié dans la contrainte, que chaque sous-zone du centre ville contient 20 bâtiments, et que chaque bâtiment du centre ville n'est à l'intérieur que d'une seule sous-zone (du centre ville). □

Exemple 12.

Nous allons exprimer maintenant avec P-OCL le fait que les centres villes de taille "moyenne" ont des sous-zones de 20 bâtiments à 50 bâtiments. Chaque îlot n'appartient qu'à une seule sous-zone du centre ville.

```
context Centre_Ville inv:
self.CV_taille = 'moyen' implies
self.geo -> contains (self.Ilot.geo, "=1", ">=20") and
self.geo -> contains (self.Ilot.geo, "=1", "<=50")
```
□

3.4 Discussion

OCL Spatial intègre des fonctionnalités pour spécifier des contraintes sur les types des objets spatiaux. Il est possible de fixer des contraintes sur les types spatiaux avec des opérations comme *oclIsTypeOf* déjà présents dans OCL. Comme indiqué dans ce chapitre, il est aussi possible d'exprimer des contraintes sur la cardinalité d'un ensemble de géométries.

OCL Spatial est étendu pour pouvoir exprimer des contraintes entre régions. Les deux catégories de langages proposés se basent sur 9IM et CBM ainsi que sur les modèles associés pour traiter les régions composites (Adverbes, TRCR).

OCL_{9IM} permet de modéliser des contraintes portant sur les 8 relations topologiques d'Egenhofer entre régions simples ou composites. OCL_{ADV} est une extension de OCL_{9IM}, reposant sur les propositions de [CLA00]. OCL_{ADV} facilite l'écriture de contraintes entre régions composites en permettant une expression plus directe de ce type de contraintes.

OCL_{CBM} suit exactement la même approche que OCL_{9IM} mais en se basant sur les relations CBM. Dans le but de faciliter l'expression de contraintes sur les régions composites, le niveau gros-grain de TRCR a été intégré dans OCL_{CBM} afin d'obtenir le langage OCL_{TRCR}. Il devient alors possible d'exprimer directement des contraintes sur les régions composites à l'aide de 4 opérations correspondant chacune à une relation de TRCR (niveau gros-grain). Ces 4 opérations offrent une sémantique très simple pour l'utilisateur, mais en contrepartie elles ne permettent pas de distinguer finemement certaines configurations topologiques. Dans l'esprit du niveau détaillé de TRCR présentant toutes les relations topologiques entre les composants, le langage P-OCL est aussi proposé ; il permet de faciliter la prise en compte d'une approche quantitative dans l'expression des contraintes.

Chapitre 4
Expressivité des différentes propositions d'extensions spatiales d'OCL

Dans ce chapitre, nous étudions d'une part le pouvoir d'expression, ou expressivité (du point de vue spatiale), des différentes propositions d'extensions d'OCL. D'autre part, nous comparons ces langages avec des modèles particulièrement expressifs pour décrire des relations pour les régions composites (9IM pour régions composites et TRCR niveau détaillé).

Dans un premier temps, en Section 4.1, nous montrons que l'ajout des types spatiaux dans OCL (OCL Spatial) permet d'exprimer les types spatiaux exprimables avec les formalismes présentés en Section 2.1 et même plus encore. Les Sections 4.2 à 4.4 sont dédiées aux comparaisons des langages basés sur CBM. Nous comparons OCL_{TRCR} et OCL_{CBM} en Section 4.2 puis P-OCL et OCL_{CBM} en Section 4.3. Nous conclurons que ces 3 langages, OCL_{CBM}, OCL_{TRCR} et P-OCL, sont équivalents en termes d'expressivité. Dans la section 4.4, nous étudions la capacité de P-OCL à exprimer les relations topologiques définies dans le niveau détaillé de TRCR. Les Sections 4.5 et 4.6 traitent des langages basés sur OCL_{9IM}. Nous démontrons l'équivalence des pouvoirs d'expression d'OCL_{9IM} et OCL_{ADV} en section 4.5. Enfin, nous étudions la capacité d'OCL_{9IM} à exprimer les relations topologiques définis dans 9IM pour les régions composites.

4.1 Comparaison : OCL Spatial / pictogrammes des modèles pour les SIG

Comme présenté dans la Section 3.1, OCL Spatial autorise la définition de types géographiques. Une classe spatiale sera une classe qui comporte un attribut ayant un type géométrique. Par exemple, la Figure 27 fait correspondre à la classe spatiale spécifiée avec Perceptory, une classe UML comportant un attribut *geo* du type géographique correspondant au pictogramme Perceptory, ici un ensemble de plusieurs polygones, i.e. *Set(Region)*.

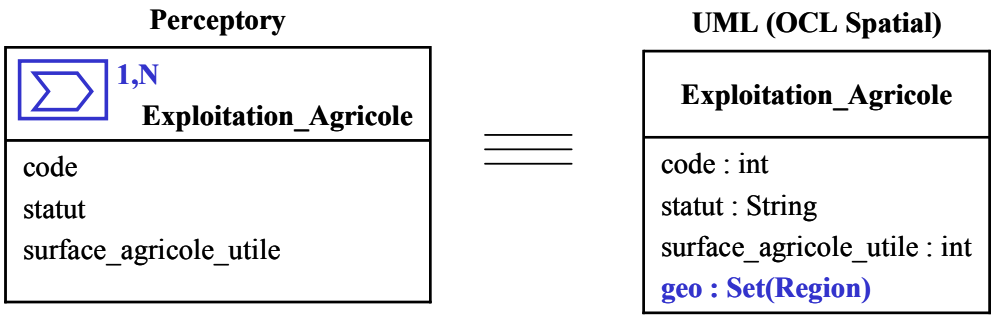


Figure 27. Exemple de classe spatiale avec Perceptory et de la classe UML correspondante utilisable avec OCL Spatial

Ainsi avec OCL spatial, on pourra exprimer les mêmes contraintes sur les types et leurs cardinalités qu'avec les pictogrammes.

Sur la Figure 28, la classe géographique *Pays* présentée dans le formalisme de Perceptory associe à chaque instance une géométrie composite de plusieurs régions et/ou points (pour modéliser les îles par exemple). Comme indiqué à la Section 3.1, à cette classe correspondent les 2 contraintes OCL suivantes, la première portant sur les types et la seconde sur les cardinalités de l'ensemble des éléments de chaque Pays.

```
(1)  context Pays inv:
     self.geo -> forAll(oclIsTypeOf(Region)
     or oclIsTypeOf(Point))

(2)  context Pays inv:
     self.geo -> size() >= 1
```

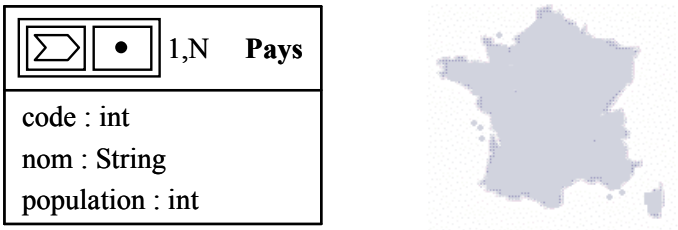


Figure 28. Classe Pays modélisée avec Perceptory et illustration d'une instance de la classe

Dans l'exemple de classe Perceptory de la Figure 29, une ville est représentée par un point ou un ensemble de plusieurs polygones en fonction de l'étendue de la ville. Les invariants OCL suivant traduisent le même type de données spatiales.

```
context Ville inv:
(self.geo -> forAll(oclIsTypeOf(Region))
and self.geo -> size() >= 1)
or (self.geo -> forAll(oclIsTypeOf(Point))
and self.geo -> size() =1)
```

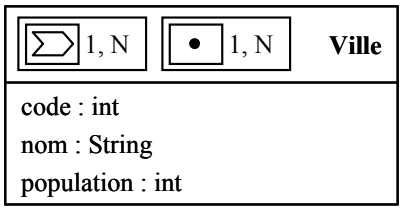


Figure 29. Classe Ville avec une représentation variable suivant l'échelle

Sur les 2 exemples précédents, il est très clair que l'utilisation des pictogrammes est bien plus intuitive que l'emploi d'OCL. Cependant, arrivé à un certain niveau de complexité de contraintes, il peut être nécessaire d'utiliser OCL. OCL Spatial permet d'exprimer des contraintes plus complexes non exprimables par les pictogrammes ; par exemple lorsque la représentation spatiale dépend de la valeur d'un attribut. Supposons que l'on souhaite représenter les éléments de la géométrie d'une ville par des polygones si la population de la ville dépasse 500000, par des points sinon. L'expression OCL Spatial suivante traduit cette contrainte.

```
context Ville inv:
if self.population >= 500000 then
  self.geo -> forAll(oclIsTypeOf(Region))
else
  self.geo -> forAll(oclIsTypeOf(Point))
end if
```

En conclusion, OCL pourrait être utilisé en complément de formalismes comme celui de Perceptory; l'expression de types spatiaux se faisant avec des pictogrammes sauf pour les cas nécessitant une spécification textuelle avec OCL Spatial.

4.2 Comparaison OCL_{TRCR} / OCL_{CBM}

Comme indiqué dans la Section 3.3.3, OCL_{TRCR} est une extension d'OCL_{CBM} (de nouvelles opérations sont intégrées dans le langage OCL_{CBM} pour obtenir OCL_{TRCR}). Dans cette section, nous démontrons que toute expression en OCL_{TRCR} peut être réécrite en OCL_{CBM} sans perte sémantique. En fait, nous démontrons que l'expressivité d'OCL_{CBM} et celle d'OCL_{TRCR} sont équivalentes.

Théorème 1. $OCL_{CBM} \subset OCL_{TRCR}$
Toutes les contraintes écrites en OCL_{CBM} peuvent aussi être spécifiées avec OCL_{TRCR}.

Preuve. Trivial. OCL_{TRCR} est basé sur OCL_{CBM}. OCL_{TRCR} ajoute seulement de nouvelles opérations à OCL_{CBM}. Toutes les expressions qui appartiennent à OCL_{CBM} appartiennent aussi à OCL_{TRCR}. □

Théorème 2. $OCL_{TRCR} \subset OCL_{CBM}$
Toutes les contraintes spécifiées avec OCL_{TRCR} peuvent aussi être exprimées en OCL_{CBM}.

Preuve. Une expression utilisant une opération TRCR d'OCL_{TRCR} peut être réécrite en une expression OCL_{CBM} en utilisant les opérations CBM et les opérations sur les collections d'OCL appliquées aux régions composites (cf. Tableau 5). Cette transformation est basée sur la conversion des expressions logiques du Tableau 1 en expressions OCL_{CBM}. En effet, cette conversion est rendue possible par l'intégration de CBM dans OCL_{CBM} et par l'utilisation des opérations sur les collections d'OCL appliquées aux géométries composites, comme vu en Définition 10. □

Corollaire 1. $\mathbf{OCL_{CBM} \equiv OCL_{TRCR}}$
Les expressivités d'OCL_{CBM} et d'OCL_{TRCR} sont équivalentes. □

Tableau 5. Passage de OCL_{TRCR} à OCL_{CBM}

Expression en OCL_{TRCR}	Expression équivalente en OCL_{CBM}
`A -> TRCR_touch(B)`	`A -> exists(Ai| B -> exists(Bj| Ai.touch(Bj))) and A -> forAll(Ar| B -> forAll(Bs| Ar.disjoint(Bs) or Ar.touch(Bs)`
`A -> TRCR_in(B)`	`A -> forAll(Ai| B -> exists(Bj| Ai.in(Bj)))`
`A -> TRCR_disjoint(B)`	`A -> forAll(Ai| B -> forAll(Bj| Ai.disjoint(Bj)))`
`A -> TRCR_overlap(B)`	`A overlaps B si les 3 expressions` OCL_{CBM} `précédentes retournent "faux"`

4.3 Comparaison P-OCL / OCL_{CBM}

Cette section montre que l'expressivité de P-OCL et celle d'OCL_{CBM} sont équivalentes.

Théorème 3. $OCL_{CBM} \subset$ P-OCL
Toutes les contraintes écrites en OCL_{CBM} peuvent aussi être spécifiées avec P-OCL.

Preuve. Trivial. P-OCL est basé sur OCL_{CBM}. P-OCL ajoute seulement de nouvelles opérations à OCL_{CBM}. Toutes les expressions qui appartiennent à OCL_{CBM} appartiennent aussi à P-OCL. □

Théorème 4. P-OCL $\subset OCL_{CBM}$
Toutes les contraintes spécifiées avec P-OCL peuvent aussi être exprimées en OCL_{CBM}.

Preuve. Une expression utilisant une opération de P-OCL peut être réécrite en une expression OCL_{CBM} en utilisant les opérations CBM, et les opérations sur les collections d'OCL.
Soit deux régions composites A et B. Pour une relation topologique donnée *R*, il faut pouvoir écrire en OCL_{CBM} une contrainte qui sélectionne les composants *a* de A (resp. *b* de B) tel qu'il existe un composant *b* de B (resp. *a* de A) tel que *R*(*a,b*) ; le nombre des composants ainsi sélectionnés doit respecter la condition *condA* (resp. *condB*).
Ainsi, les opérations `A->topo_rel(B, "condA", "condB")` de P-OCL s'écrivent en OCL_{CBM} de la façon suivante :

```
A -> select( part_of_A|
  B -> exists( part_of_B|
    part_of_A.topo_rel(part_of_B))) -> size condA
and
B -> select( part_of_B|
  A -> exists( part_of_A|
    part_of_A.topo_rel(part_of_B))) -> size condB
```

Si *condA*, respectivement *condB*, contient le symbole "*", il est remplacé par `A->size()`, respectivement `B->size()`, dans l'expression OCL_{CBM} précédente.

Corollaire 2. $OCL_{CBM} \equiv$ P-OCL
Les expressivités d'OCL_{CBM} et de P-OCL sont équivalentes. □

4.4 Comparaison P-OCL / TRCR niveau détaillé

Dans cette section, nous étudions si le langage P-OCL est capable d'exprimer les relations topologiques définis dans le niveau détaillé de TRCR.

Comme indiqué dans la section 2.3.2., une approche pour énumérer toutes les relations entre des régions composites est décrite dans [CLE95]. Il s'agit du niveau détaillé de TRCR. Rappelons que le principe du niveau détaillé de cette proposition est de considérer une matrice *n*x*m* où *n* est le nombre de parties de la région composite *A*, et *m* est le nombre de parties de la région composite *B*. La matrice présente l'ensemble des relations qui existent entre les parties de *A* et les parties de *B*. Reprenons l'exemple de la relation topologique présentée dans la Section 2.3.2. et rappelée à la Figure 30 et au Tableau 6.

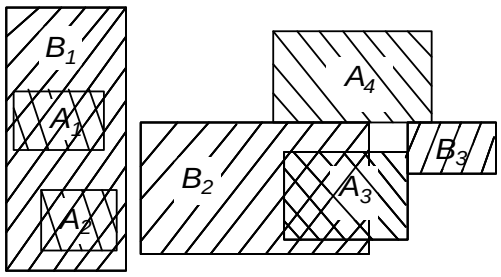


Figure 30. Scène représentant une relation topologique entre deux régions composites *A* et *B*.

Tableau 6. La matrice topologique *n*x*m* pour les régions composites de la Figure 30 (*O=overlap*, *M=meet*, *D=disjoint*, *I=in*)

	B_1	B_2	B_3
A_1	I	D	D
A_2	I	D	D
A_3	D	O	M
A_4	D	M	M

En P-OCL et (donc aussi en OCL_{CBM}), nous pouvons aisément définir le nombre de couples (*i*,*j*) impliqués dans une relation topologique spécifique i.e. le nombre d'apparitions d'une relation CBM dans une matrice sur des lignes et des colonnes distinctes.

Définition 13.

La forme générale des contraintes P-OCL décrivant le nombre d'apparitions de relations CBM sur des lignes et des colonnes distinctes d'une matrice *n*x*m* s'exprime sous la forme (p_1 and p_2) où p_1 et p_2 sont les expressions suivantes :

p_1 =

```
(A -> size = n) and (B -> size = m)
and
A -> relation1(B, sA1, sB1)
and
...
A -> relationp(B, sAp, sBp)
...
and
A -> relationz(B, sAz, sBz)
```

où *relation*$_i$ est l'une des opérations CBM applicables entre deux régions (*disjoint*, *in*, *meet*, *overlap*) et où sA_i (resp. sB_i) est le nombre d'apparitions du prédicat correspondant *relation*$_i$ sur des lignes distinctes (resp. sur des colonnes distinctes).

Si une relation présente dans la cellule *(k,l)* est également présente dans une cellule de la ligne *k* et dans une cellule de la ligne *l*, alors :

p_2 =

```
A->exists(pA| B->select(pB|pA.relation(pB))->size = rA)
and
B->exists(pB| A->select(pA|pA.relation(pB))->size = rB)
```

où *relation* est l'opération CBM qui apparaît dans la cellule (*k*,*l*) considérée et où r_A (resp. r_B) est le nombre de cellules de la ligne *k* (resp. de la colonne *l*) dans lesquelles est présente cette même relation.

Si deux cellules *(k,l)* et *(r,s)* comportent la même relation et que les cellules *(k,s)* et *(r,l)* présentent elles aussi cette même relation, alors :

p_2 =

```
A->exists(pA1| … A->exists(pArB|
  pA1<>…<>pArB
  and (B->select(pB|pA1.relation(pB))->size = rA)
  ...
  and (B->select(pB|pArB.relation(pB))->size = rA))
and
B->exists(pB1| … B->exists(pBrA|
  pB1<>…<>pBrA
  and (A->select(pA|pA.relation(pB1))->size = rB)
  ...
  and (A->select(pA|pA.relation(pBrA))->size = rB))
```

□

Dans la première partie, on spécifie le nombre de composants des deux régions et la configuration des relations. Dans la seconde partie, on traite le cas où plusieurs composants d'une région sont en relation avec plusieurs composants de l'autre région.

Par exemple, le nombre d'apparitions des relations CBM dans le Tableau 6 peut être décrit comme suit avec P-OCL. Les relations entre composants qui ne sont pas spécifiées explicitement sont des relations *disjoint*.

```
(A -> size = 4) and (B -> size = 3)
and
A -> in(B, 2, 1)
and A -> meet(B, 2, 2)
and A -> overlap(B, 1, 1)
and
A -> exists(pA| B -> select(pB| pA.meet(pB))->size = 2)
and B -> exists(pB| A -> select(pA| pA.meet(pB))->size = 2)
```

C'est-à-dire la région *A* est composée de 4 composants et la région *B* est composée de 3 composants. Deux des composants de *A* sont à l'intérieur d'un composant de *B*. Deux composants de *A* sont adjacents à deux composants de *B*. Un et un seul composant de *A* intersecte un et un seul composant de *B*. Et enfin, pour la seconde partie, un composant de A est adjacent à deux composants de B ; et un composant de B est adjacent à deux composants de A.

Dans la Définition 13, les composants des régions ne sont pas nommés. Si chaque composant possédait un attribut "nom de partie" (*name*) en plus de son attribut géométrique (*geo*), une matrice *n*×*m* pourrait se réécrire en OCL_{CBM} grâce à des implications sur l'attribut nom de partie.

Par exemple, la matrice *n*x*m* du Tableau 6 peut être réécrite comme suit :

```
(A -> size = 4) and (B -> size = 3)
and
A -> forAll( part_of_A|
  B -> forAll( part_of_B|
    (part_of_A.name = ″A1″ and part_of_B = ″B1″
     implies
     part_of_A.geo.in(part_of_B.geo))
    and
     ...(idem pour chaque cellule de la matrice n×m)
```

Ainsi, P-OCL permet d'exprimer le nombre de couples (*i*,*j*) impliqués dans une relation topologique spécifique, i.e. le nombre d'apparitions d'une relation CBM dans une matrice sur des lignes et colonnes distinctes. Cependant, il n'y a pas réellement équivalence. En effet, TRCR permet d'exprimer des relations entre des régions composites dont les composants sont nommés, mais on ne peut pas définir des relations entre des composants anonymes. Plus précisément, on peut écrire que A_1 doit être à l'intérieur de B_1 ; mais il n'est pas possible d'écrire que 2 composants anonymes de *A* doivent être à l'intérieur de 2 composants anonymes de *B*. Avec P-OCL, la situation est opposée.

4.5 Comparaison OCL_{ADV} / OCL_{9IM}

Dans cette section, nous démontrons qu'OCL_{9IM} et OCL_{ADV} ont exactement le même pouvoir d'expression.

Tableau 7. De OCL_{ADV} à OCL_{9IM}. *A* et *B* sont de type *Set(Region)*.
Par exemple, *A* and *B* pourraient être "*self.geoA*" et "*self.association.geoB*"

mostly – `A -> relation_topo("mostly", B)` se réécrit : `B -> forAll(Bj| A -> exists(Ai| Ai.relation_topo(Bj)))`
mostly$_{rev}$ – `A -> relation_topo("mostlyRev", B)` se réécrit : `A -> forAll(Ai| B -> exists(Bj| Ai.relation_topo(Bj)))`

completely – `A -> relation_topo("completely", B)` se réécrit :
`B -> forAll(Bj| A -> exists(Ai| Ai.relation_topo(Bj))) and`
`A -> forAll(Ai| B -> exists(Bj| Ai.relation_topo_rev(Bj)))`

partially – `A -> relation_topo("partially", B)` se réécrit :
`A -> exists(Ai| B -> exists(Bj| Ai.relation_topo(Bj))) and`
`A -> forAll(Ar| B -> forAll(Bs| Ar.relation_topo(Bs) or Ar.disjoint(Bs)))`

occasionally – `A -> relation_topo("occasionally", B)` se réécrit : `A -> exists(Ai| B -> exists(Bj| Ai.relation_topo(Bj)))`

entirely – `A -> relation_topo("entirely", B)` se réécrit :
`A -> forAll(Ai| B -> forAll(Bj| Ai.relation_topo(Bj) and`
`Bj.relation_topo_rev(Ai)))`

never – `A -> relation_topo("never", B)` se réécrit : `A -> forAll(Ai| B -> forAll(Bj| not Ai.relation_topo(Bj)))`

Théorème 5. $OCL_{9IM} \subset OCL_{ADV}$

Toutes les contraintes spécifiées avec OCL_{9IM} peuvent aussi être exprimées en OCL_{ADV}.

Preuve. Trivial. OCL_{ADV} est basé sur OCL_{9IM}. OCL_{ADV} ajoute seulement de nouvelles opérations à OCL_{9IM}. Toutes les expressions de contraintes qui appartiennent à OCL_{9IM} appartiennent aussi à OCL_{ADV}. □

Théorème 6. $OCL_{ADV} \subset OCL_{9IM}$
Toutes les contraintes spécifiées avec OCL_{ADV} peuvent aussi être exprimées en OCL_{9IM}.

Preuve. Une expression qui utilise une opération ADV d'OCL_{ADV} peut être réécrite en une expression d'OCL_{9IM} en utilisant les opérations 9IM et les opérations sur les collections d'OCL (cf. Tableau 7). Cette transformation est basée sur la conversion des expressions logiques du Tableau 3 en expressions OCL. En effet, cette conversion est rendue possible par l'intégration de 9IM dans OCL et par l'utilisation des opérations sur les collections d'OCL appliquées aux géométries composites, comme vu en Section 2.3.1. □

Corollaire 3. $\mathbf{OCL_{9IM} \equiv OCL_{ADV}}$
Les expressivités d'OCL_{9IM} et d'OCL_{ADV} sont équivalentes. □

4.6 Comparaison OCL_{9IM} / 9IM pour les régions composites

Dans cette section, nous étudions si le langage OCL_{9IM} est capable de modéliser toutes les relations topologiques exprimables avec 9IM appliqué aux régions composites tels que présentées dans [BEH01].

Les matrices 9IM peuvent être appliquées sur des géométries complexes au sens vu en Section 2.2.1. En effet, 9IM se base sur les intersections des intérieurs, des frontières et des extérieurs de 2 géométries, sans prendre en compte leur nombre de composants. Dans [BEH01], les auteurs énumèrent toutes les matrices possibles pour des régions complexes. Ils identifient 33 matrices 9IM applicables entre 2 géométries complexes non-vides. On a donc 33 relations topologiques possibles entre 2 régions complexes (avec ou sans trou).

Pour délimiter l'expressivité d'OCL_{9IM} (Section 3.2.1), nous comparons les relations exprimables en OCL_{9IM} à celles définissables par une matrice 9IM appliquée aux régions composites. Pour cela, nous proposons un théorème pour passer d'une matrice des 9 intersections à l'expression logique correspondante des contraintes OCL_{9IM} (Définition 8).

Ainsi, nous allons montrer dans cette section qu'OCL_{9IM} permet aux concepteurs d'exprimer les relations topologiques entre des régions composites. Contrairement aux composants des régions complexes, les

composants des régions composites n'ont pas de trous, et leurs frontières sont forcément disjointes (voir Définition 7 en Section 3.3.1).

Dans l'étude que nous présentons, nous avons trouvé que le nombre de matrices 9IM possibles, i.e. le nombre de relations topologiques existant entre 2 régions composites, n'est plus que de 16. Les régions composites étant un cas particulier des régions complexes (i.e. leurs composants ne s'intersectent pas et ils ne comportent pas de trou), il ne peut pas exister de relation entre deux régions composites qui ne soit pas une des 33 identifiées dans [BEH01].

Propriété 1. Matrices 9IM qui caractérisent des relations topologiques entre 2 régions complexes comportant au moins un trou ou des composants adjacents.
Soient *A* et *B* deux régions complexes. Au moins une des deux régions contient un ou plusieurs trous ou comporte des composants adjacents quand la frontière de *A* n'intersecte pas l'extérieur de *B*, et en même temps, l'intérieur de *A* intersecte l'extérieur de *B*, i.e.

i) $(A^\circ \cap \bar{B} = \neg\varnothing) \wedge (\delta A \cap \bar{B} = \varnothing)$,

et réciproquement, c'est-à-dire :

ii) $(\bar{A} \cap B^\circ = \neg\varnothing) \wedge (\bar{A} \cap \delta B = \varnothing)$. □

Preuve.
Sans trou dans *A* ni *B*, l'intérieur de *A* ne peut être à l'extérieur de *B* quand la frontière de *A* n'est pas à l'extérieur de *B*, et réciproquement. En effet, prenons par exemple le cas *(i)*. Comme illustré sur la Figure 31, lorsque $A^\circ \cap \bar{B}$ n'est pas vide, c'est-à-dire lorsque l'intérieur de *A* est au moins en partie à l'extérieur de *B*, alors la frontière de *A* est elle aussi au moins en partie à l'extérieur de *B*. Cela est dû au fait que par définition :

- d'une part, contrairement au cas de la Figure 31.3 une région composite ne comporte pas de trou ; de plus, les frontières de ses composants ne peuvent pas s'intersecter ce qui pourrait créer un "pseudo-trou" (cf [SCH02]).
- et d'autre part, contrairement au cas de la Figure 31.2 où l'union des 4 composants de *B* recouvre entièrement la frontière de *A*, ses composants sont disjoints entre eux.

Les 2 cas de la Propriété 1 correspondent aux patrons de matrice donnés dans le Tableau 8. □

Tableau 8. Patrons de matrice de relations topologiques entre des régions complexes comportant des trous. x∈ {0;1}

Cas (i)			*Cas (ii)*		
x	x	1	x	x	x
x	x	0	x	x	x
x	x	x	1	0	x

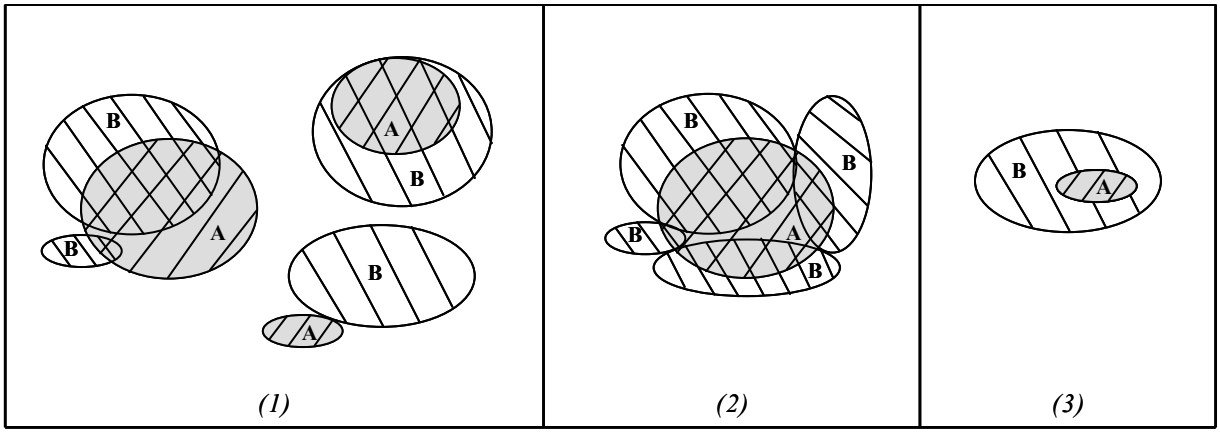


Figure 31. ***(1)*** **Scène topologique entre deux régions composites**
(2) **et** ***(3)*** **Scènes topologiques entre deux régions complexes**

Parmi les 33 matrices 9IM possibles, 17 matrices correspondent au patron du Tableau 8. Ces 17 matrices impliquent donc une région complexe. Les 16 matrices restantes correspondent à des relations topologiques entre régions composites. En effet, à chacune de ses 16 matrices correspond au moins une scène topologique entre deux régions composites, comme dessiné en Annexe. D'où le Corollaire 4.

Corollaire 4. Ensemble *M* des matrices 9IM qui caractérisent des relations topologiques entre 2 régions composites.
Il existe 16 relations topologiques entre 2 régions composites non-vides sans trou. On notera *M* l'ensemble des 16 matrices 9IM correspondantes. □

Nous allons montrer que chacune des 16 relations topologiques possibles entre des régions composites peut se réécrire sous forme logique. Cette forme logique décrit le lien qui existe entre une relation décrite par une matrice 9IM entre deux régions composites, et les relations entre les composants de ces deux régions.

Dans un premier temps, nous allons montrer que chacune des 16 matrices 9IM correspondantes peut être considérée comme une combinaison booléenne de matrices dites de factorisation. Une matrice de factorisation est une matrice 9IM correspondant à une relation topologique entre régions simples, sauf pour la relation *disjoint* qui est séparée en deux cas.

Dans notre modèle, *disjoint* n'est plus commutatif lorsqu'il est appliqué aux parties des géométries composites *A* et *B*. Dans la scène de la Figure 32, 2 parties de *B* sont disjointes de toutes les parties de *A*, alors qu'aucune partie de *A* n'est disjointe de toutes les parties de *B*. On définit donc 2 matrices de disjonction. *A.disjoint1st(B)* signifie qu'au moins une partie de *A* est disjointe de toutes celles de *B*. En effet, dans ce cas-ci, l'extérieur de *A* intersecte l'intérieur, la frontière et l'extérieur de chacune des parties de *B*. Réciproquement, *A.disjoint2nd(B)* signifie qu'au moins une partie de *B* est disjointe de chacune des parties de *A*.

Chacune de ces 2 matrices ne peut pas être la seule relation entre 2 régions composites. Par exemple, en cas d'utilisation de *A.disjoint1st(B)*, au moins une autre relation topologique doit exister entre les composants de *A* et ceux de *B*. Ce peut être l'opération inverse *disjoint2nd* ou l'une de celles définies entre les géométries simples. En effet, dans le cas de deux régions simples, si *A* est disjoint de tous les composants de *B*, alors *B* sera disjoint de tous les composants de *A*.

Nous pouvons maintenant définir précisément les matrices de factorisation.

Tableau 9. Décomposition de *disjoint* en deux matrices de factorisation

A→fDisjoint1st(B)	A→fDisjoint2nd(B)
0 0 1	0 0 0
0 0 1	0 0 0
0 0 1	1 1 1

Définition 14. Matrices de factorisation.
Les matrices 9IM des relations topologiques *fInside*, *fEqual*, *fMeet*, *fCovers*, *fCoveredBy*, *fOverlap* entre régions simples ainsi que les 2 matrices *fDisjoint1st* et *fDisjoint2nd* qui dérivent de *disjoint* sont qualifiées de matrices de factorisation.
L'ensemble *F* des matrices de factorisation est donc :

```
F = { fContains, fInside, fEqual, fMeet, fCovers,
fCoveredBy, fOverlap, fDisjoint1st, fDisjoint2nd }
```

Notons que *disjoint* est la somme booléenne de *disjoint1st* et *disjoint2nd*.

Ces matrices sont données dans le Tableau 10.

Les matrices de factorisation sont utilisables entre les parties des géométries composites, i.e. des formes simples.

Tableau 10. Ensemble des 9 matrices de factorisation *F*

1 1 1 0 0 1 0 0 1 $f_1 = \langle A, fContains, B \rangle$	1 0 0 1 0 0 1 1 1 $f_2 = \langle A, fInside, B \rangle$	1 0 0 0 1 0 0 0 1 $f_3 = \langle A, fEqual, B \rangle$
0 0 1 0 1 1 1 1 1 $f_4 = \langle A, fMeet, B \rangle$	1 1 1 0 1 1 0 0 1 $f_5 = \langle A, fCovers, B \rangle$	1 0 0 1 1 0 1 1 1 $f_6 = \langle A, fCoveredBy, B \rangle$
1 1 1 1 1 1 1 1 1 $f_7 = \langle A, fOverlap, B \rangle$	0 0 1 0 0 1 0 0 1 $f_8 = \langle A, fDisjoint1st, B \rangle$	0 0 0 0 0 0 1 1 1 $f_9 = \langle A, fDisjoint2nd, B \rangle$

Le théorème suivant garantit la décomposition de chacune des 16 matrices de *M* en fonction des matrices de factorisation.

Théorème 7.

Soit *R* une matrice de *M*, c_i un booléen et f_i une matrice de factorisation pour tout *i* de 1 à 9, alors :

$R = \sum c_i f_i$ pour tout i de 1 à 9 □

Selon le Théorème 7, chacune des 16 matrices de *M* est une combinaison de certaines matrices de factorisation. Sachant une matrice *R*, une matrice de factorisation apparaît dans la combinaison si la relation topologique correspondante est possible entre deux composants des deux régions composites (cas des matrices f_1 à f_7), ou bien s'il y a une relation de disjonction entre un composant d'une région composite et tous les composants de l'autre région composite (cas des matrices f_8 et f_9).

Preuve.

Pour établir la preuve du Théorème 7, nous montrons qu'il existe un coefficient non nul dans une cellule d'une des matrices de factorisation résultats de la décomposition si et seulement si il existe un coefficient non nul dans la cellule correspondante de la matrice qui représente la relation entre régions composites. Pour cela, il faut considérer les différents cas d'intersections entre intérieurs, frontières et extérieurs.

Rappelons que l'intérieur d'une région composite A est l'union des intérieurs de ses composants $\{a_1,\ldots,a_n\}$, c'est-à-dire : $A^\circ = a_1^\circ \cup \ldots \cup a_n^\circ$. La frontière d'une région composite A est l'union des frontières de ses composants $\{a_1,\ldots,a_n\}$: $\delta A = \delta a_1 \cup \ldots \cup \delta a_n$. L'extérieur d'une région composite A est l'intersection des extérieurs de ses composants $\{a_1,\ldots,a_n\}$: $\bar{A} = \bar{a_1} \cap \ldots \cap \bar{a_n}$.

Nous nous intéressons d'abord aux cas d'intersection avec l'intérieur d'une région. On peut écrire :

$$\begin{aligned}(A^\circ \cap B^\circ = \neg\varnothing) &\equiv ((a_1^\circ \cup \ldots \cup a_n^\circ) \cap B^\circ = \neg\varnothing)\\ &\equiv ((a_1^\circ \cap B^\circ) \cup \ldots \cup (a_n^\circ \cap B^\circ) = \neg\varnothing)\\ &\equiv ((a_1^\circ \cap (b_1^\circ \cup \ldots \cup b_n^\circ)) \cup \ldots \cup (a_n^\circ \cap (b_1^\circ \cup \ldots \cup b_n^\circ)) = \neg\varnothing)\\ &\equiv ((a_1^\circ \cap b_1^\circ) \cup .. \cup (a_1^\circ \cap b_n^\circ) \cup .. \cup (a_n^\circ \cap b_1^\circ) \cup .. \cup (a_n^\circ \cap b_n^\circ) = \neg\varnothing)\end{aligned}$$

c'est-à-dire que $(A^\circ \cap B^\circ = \neg\varnothing)$ si et seulement si il existe au moins un composant a_i de A et un composant b_j de B tels que $(a_i^\circ \cap b_j^\circ = \neg\varnothing)$; autrement dit, si et seulement si, il existe deux composants a_i de A et b_j de B qui sont en relation *contains*, *inside*, *equal, covers*, *coveredby*, ou *overlap* (matrice de factorisation f_1, f_2, f_3, f_5, f_6 ou f_7). En conséquence, la décomposition d'une matrice représentant une relation entre régions composites et ayant $(A^\circ \cap B^\circ = \neg\varnothing)$ comprend donc forcément une matrice de factorisation f_1, f_2, f_3, f_5, f_6 ou f_7. Soit une décomposition en matrices de factorisation d'une matrice représentant une relation entre régions composites ; il existe dans une matrice de factorisation un coefficient non nul dans la cellule correspondant à l'intersection des intérieurs si et seulement si il existe un coefficient non nul dans la cellule correspondante de la matrice qui représente la relation entre régions composites.

De la même façon, on obtient :

$(A^\circ \cap \delta B = \neg\varnothing) \equiv (a^\circ \cap \delta b = \neg\varnothing)$, c'est-à-dire que $(A^\circ \cap \delta B = \neg\varnothing)$ si et seulement si il existe au moins un composant a_i de A et un composant b_j de B tels que $(a_i^\circ \cap \delta b_j = \neg\varnothing)$ ou encore si et seulement si a_i et b_j sont en relation *contains*, *covers* ou *overlap* (matrices f_1, f_5 ou f_7).

$(\delta A \cap B^\circ = \neg\varnothing) \equiv (\delta a \cap b^\circ = \neg\varnothing)$, c'est-à-dire que $(\delta A \cap B^\circ = \neg\varnothing)$ si et seulement si il existe au moins un composant a_i de A et un composant b_j de B tels que $(\delta a_i \cap b_j^\circ = \neg\varnothing)$ ou encore si et seulement si a_i et b_j sont en relation *inside*, *coveredBy* ou *overlap* (matrices f_2, f_6 ou f_7).

En ce qui concerne les intersections entre frontières, on a :

$(\delta A \cap \delta B = \neg\varnothing) \equiv (\delta a \cap \delta b = \neg\varnothing)$, c'est-à-dire que $(\delta A \cap \delta B = \neg\varnothing)$ si et seulement si il existe au moins un composant a_i de A et un composant b_j de B tels que $(\delta a_i \cap \delta b_j = \neg\varnothing)$ ou encore si et seulement si a_i et b_j sont en relation *equal*, *meet*, *covers*, *coveredBy* ou *overlap* (matrices f_3, f_4, f_5, f_6 ou f_7).

Nous nous intéressons maintenant aux intersections impliquant l'extérieur d'une région.

Comme deux extérieurs s'intersectent toujours, on a obligatoirement :

$(\bar{A} \cap \bar{B} = \neg\emptyset)$ et $\forall i \, \forall j \, (\bar{a_i} \cap \bar{b_j} = \neg\emptyset)$

Ces intersections sont donc toujours équivalentes, puisque toujours non vides.

La définition des régions composites fait que $(\bar{A} \cap B^\circ = \bar{A} \cap \delta B)$ quelles que soient les deux régions composites A et B (ce qui inclut donc le cas particulier des régions simples) (cf Preuve de la Propriété 1). Par conséquent, nous traitons ces deux intersections en même temps.

$((\bar{A} \cap B^\circ = \neg\emptyset) \wedge (\bar{A} \cap \delta B = \neg\emptyset))$

$\equiv ((\bar{A} \cap (b_1^\circ \cup \ldots \cup b_n^\circ) = \neg\emptyset) \wedge (\bar{A} \cap (\delta b_1 \cup \ldots \cup \delta b_n) = \neg\emptyset))$

$\equiv (((\bar{A} \cap b_1^\circ) \cup \ldots \cup (\bar{A} \cap b_n^\circ) = \neg\emptyset) \wedge ((\bar{A} \cap \delta b_1) \cup \ldots \cup (\bar{A} \cap \delta b_n) = \neg\emptyset))$

c'est-à-dire $((\bar{A} \cap B^\circ = \neg\emptyset) \wedge (\bar{A} \cap \delta B = \neg\emptyset))$ si et seulement si il existe au moins un composant b_j de B tel que $(\bar{A} \cap b_j^\circ = \neg\emptyset) \wedge (\bar{A} \cap \delta b_j = \neg\emptyset)$. Là encore, la définition des régions composites implique que le composant soit le même dans les deux égalités puisqu'on aura toujours $(\bar{A} \cap b_j^\circ = \bar{A} \cap \delta b_j)$.

Autrement dit, $((\bar{A} \cap B^\circ = \neg\emptyset) \wedge (\bar{A} \cap \delta B = \neg\emptyset))$ si et seulement si il existe au moins un composant b_j de B qui est en relation *inside*, *meet*, *coveredBy*, *overlap* ou *disjoint* avec tous les composants de A (matrices f_2, f_4, f_6, f_7 ou f_9). Ou encore, b_j n'est jamais en relation *equal*, *coveredBy*, ni *inside* avec A. On distingue donc deux cas :

- si et seulement si il existe au moins un composant b_j de B disjoint de tous les composants de A (matrice f_9),
- ou bien si et seulement si b_j est en relation *inside*, *meet*, *coveredBy* ou *overlap* avec un composant a_i de A (matrices f_2, f_4, f_6 ou f_7). En effet, dans ce cas et dû à la définition des régions composites (deux composants ne s'intersectent jamais), b_j ne sera jamais en relation *equal*, *coveredBy*, ni *inside* avec A.

On a donc $((\bar{A} \cap B^\circ = \neg\emptyset) \wedge (\bar{A} \cap \delta B = \neg\emptyset))$ si et seulement si il existe au moins un composant b_j de B qui est disjoint de tous les composants de A (matrices f_9), ou bien b_j est en relation *contains*, *meet*, *covers* ou *overlap* avec un composant a_i de A (matrices f_2, f_4, f_6 ou f_7).

Il en va de même pour :

$(A^\circ \cap \bar{B} = \neg\emptyset) \wedge (\delta A \cap \bar{B} = \neg\emptyset)$

$\equiv (((a_1^\circ \cap \bar{B}) \cup \ldots \cup (a_n^\circ \cap \bar{B}) = \neg\emptyset) \wedge ((\delta a_1 \cap \bar{B}) \cup \ldots \cup (\delta a_n \cap \bar{B}) = \neg\emptyset))$

Autrement dit, $((A^\circ \cap \bar{B} = \neg\emptyset) \wedge (\delta A \cap \bar{B} = \neg\emptyset))$ si et seulement si il existe au moins un composant a_i de A tel que $(a_i^\circ \cap \bar{B} = \neg\emptyset) \wedge (\delta a_i \cap \bar{B} = \neg\emptyset)$. Là encore, la définition des régions composites implique que le composant soit le même dans les deux égalités puisqu'on aura toujours $(a_i^\circ \cap \bar{B} = \delta a_i \cap \bar{B})$.

Ce qui revient à dire $(A^\circ \cap \bar{B} = \neg\emptyset) \wedge (\delta A \cap \bar{B} = \neg\emptyset)$ si et seulement si il existe au moins un composant a_i de A disjoint de tous les composants de b (matrice f_8), ou bien a_i est en relation *contains*, *meet*, *covers* ou *overlap* avec un composant b_j de B (matrices f_1, f_4, f_5 ou f_7). □

On note que le booléen c_8 associé à la matrice de factorisation *disjoint1st* ne peut être le seul coefficient non nul, idem pour le booléen c_9 associé à la matrice de factorisation *disjoint2nd*.

Exemple 13. Application du théorème.
Nous allons établir la décomposition d'une matrice *R* de *M* sur un exemple.

Considérons la matrice $R = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ qui spécifie une relation topologique entre 2 régions composites.

Nous devons déterminer les booléens de c_1 à c_9 tels que :

$$R = c_1.fContains + c_2.fInside + c_3.fEqual + c_4.fMeet + c_5.fCovers + c_6.fCoveredBy + c_7.fOverlap + c_8.fDisjoint1st + c_9.fDisjoint2nd$$

Plus explicitement, on a :

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} = c_1. \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} + c_2. \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} + c_3. \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + c_4. \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} + c_5. \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$+ c_6. \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} + c_7. \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} + c_8. \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} + c_9. \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Ainsi, on obtient le système de sommes booléennes suivant :

(e_{11}) $c_1 + c_2 + c_3 + c_5 + c_6 + c_7 = 1$
(e_{12}) $c_1 + c_5 + c_7 = 0$
(e_{13}) $c_1 + c_4 + c_5 + c_7 + c_8 = 1$
(e_{21}) $c_2 + c_6 + c_7 = 1$
(e_{22}) $c_3 + c_4 + c_5 + c_6 + c_7 = 0$
(e_{23}) $c_1 + c_4 + c_5 + c_7 + c_8 = 1$
(e_{31}) $c_2 + c_4 + c_6 + c_7 + c_9 = 1$
(e_{32}) $c_2 + c_4 + c_6 + c_7 + c_9 = 1$
(e_{33}) $c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9 = 1$

Les équations (e_{13}) et (e_{23}) sont équivalentes, idem pour (e_{31}) et (e_{32}).

L'équation (e_{12}) implique que $c_1 = c_5 = c_7 = 0$.
L'équation (e_{22}) implique que $c_3 = c_4 = c_5 = c_6 = c_7 = 0$.

Il reste :

(e_{11}) $c_2 = 1$
(e_{13}) $c_8 = 1$
(e_{21}) $c_2 = 1$
(e_{31}) $c_2 + c_9 = 1$
(e_{33}) $c_2 + c_8 + c_9 = 1$
(e_4) $c_1 = c_3 = c_4 = c_5 = c_6 = c_7 = 0$

(e_{11}) et (e_{13}) impliquent que $c_2 = 1$ et $c_8 = 1$.

La solution finale de ce système est : ($c_2 = 1$ et $c_8 = 1$) et ($c_1 = c_3 = c_4 = c_5 = c_6 = c_7 = 0$).
Ainsi puisque $c_2 = 1$ et $c_8 = 1$ alors *R* se décompose par les matrices de factorisation suivantes :
- *fInside* (matrice *f2*) ; ce qui signifie que si *A et B* sont deux régions composites et *R* (*A*,*B*), alors il existe $a \in A$, $b \in B$ tel que *a inside b*,
- *fDisjoint1st* (matrice *f8*) ; ce qui signifie que si *R* (*A*,*B*) alors il existe $a \in A$ tel que *a disjoint B*.
La valeur de c_9 est indéterminée puisque dans (e_{31}) et (e_{33}), c_2 n'est jamais nul. Cela signifie que la présence de la matrice de factorisation *fDisjoint2nd* est indifférente dans la décomposition. □

Dans un deuxième temps, nous allons montrer que chacune des matrices de factorisation peut être exprimée sous une forme logique. La propriété suivante établit ce résultat. Par définition, les matrices de factorisation correspondent aux relations qui existent entre les composants des deux régions composites.

Propriété 2.
D'une part, la situation topologique correspondant à l'une des 7 premières matrices de factorisation qui correspond à l'une des 7 premières relations d'Egenhofer (i.e. *fContains*, *fInside*, *fEqual*, *fMeet*, *fCovers*, *fCoveredBy*, *fOverlap*) se traduit sous forme logique par :

$\langle A, fRelation\_Topologique, B \rangle \Rightarrow$
$\quad \exists\ r \in 1..n, \exists\ s \in 1..m \mid \langle A_r, fRelation\_Topologique, B_s \rangle$

où A_r et B_s sont les parties des régions composites *A* et *B*. En d'autres termes, lorsqu'une matrice de factorisation (autre que *fDisjoint1st* et *fDisjoint2nd*) intervient dans la décomposition d'une matrice, cela implique qu'il existe un couple de composants de *A* et *B* qui vérifie la relation correspondante entre régions simples.

D'autre part, la situation topologique correspondant aux 2 matrices de factorisation *fDisjoint1st* et *fDisjoint2nd* se traduit comme suit :

$\langle A, \texttt{fDisjoint1st}, B\rangle \Rightarrow \exists\ r \in 1..n, \forall\ s \in 1..m \mid \langle A_r, \texttt{disjoint}, B_s\rangle$

$\langle A, \texttt{fDisjoint2nd}, B\rangle \Rightarrow \forall\ r \in 1..n, \exists\ s \in 1..m \mid \langle A_r, \texttt{disjoint}, B_s\rangle$

C'est-à-dire que, lorsqu'une matrice de factorisation *fDisjoint1st* (resp. *fDisjoint2nd*) intervient dans la décomposition d'une matrice, cela implique qu'il existe au moins un composant de *A* (resp. *B*) qui est disjoint de tous les composants de *B* (resp. *A*). □

Preuve.

Par définition, les matrices de factorisation correspondent aux relations topologiques qui existent entre les composants des régions composites. Ces composants sont des régions simples. Ainsi, les 7 premières matrices de factorisation correspondent à l'existence d'au moins un couple de composants des deux régions composites qui vérifie la relation d'Egenhofer correspondante. Pour la matrice *fDisjoint1st* (resp. *fDisjoint2nd*), cela signifie qu'au moins un composant de *A* (resp. de *B*) est disjoint de tous les composants de *B* (resp. de *A*). □

Il nous reste maintenant à dériver l'expression logique de la décomposition complète pour une matrice de *M*. Nous devons distinguer 2 parties dans cette décomposition : la première partie correspond aux coefficients non nuls de la combinaison et la seconde partie traite des coefficients nuls et, implicitement, de ceux dont la valeur est indéterminée. Le passage de ces 2 parties à la forme logique qui leur correspond est donné par le Théorème 8.

Théorème 8. Réécriture logique de la décomposition d'une matrice de *M*

Réécriture logique de la première partie de la décomposition.

Soit N_1 l'ensemble des *i* tels que le coefficient c_i n'est pas nul dans le résultat de l'application du Théorème 7. La première partie de l'expression logique finale est la conjonction des expressions logiques des matrices de factorisation correspondantes (cf. Propriété 2).

Réécriture logique de la seconde partie de la décomposition.

Soit N_2 l'ensemble des *i* tels que le coefficient c_i est nul dans le résultat de l'application du Théorème 7. La seconde partie de l'expression logique finale est

$$\forall\ r \in 1..n, \forall\ s \in 1..m \mid \bigvee_{i(\{1..9\}\backslash N_2} \langle A_r, \texttt{fRelationTopo}_i, B_s\rangle$$

où A_r et B_s sont les parties des régions composites *A* et *B*, et $fRelationTopo_i$ est la relation topologique associée à la matrice de factorisation correspondante.

Dans cette partie, on autorise toutes les relations entre les composants sauf celles qui ne doivent jamais apparaître, c'est-à-dire que seules les relations qui correspondent aux coefficients non-nuls ou non-significatifs dans le résultat du Théorème 7 peuvent exister entre les composants des régions composites.

Réécriture logique de la décomposition.

La réécriture sous forme logique de la décomposition est la conjonction des deux réécritures précédentes. □

Preuve.

Les relations qui correspondent aux matrices de factorisation dont le coefficient vaut 1 apparaissent obligatoirement au moins une fois entre deux composants des deux régions composites.

Par contre, les relations qui correspondent aux matrices de factorisation dont le coefficient vaut 0 n'apparaissent jamais. Autrement dit, seules les relations qui correspondent aux matrices de factorisation dont le coefficient vaut 1 et à celles dont le coefficient est indéterminé peuvent apparaître. □

Ainsi, les relations topologiques qui peuvent exister entre les parties de régions composites sont, d'une part, celles qui correspondent aux coefficients non nuls et, d'autre part, celles qui n'apparaissent pas parmi les coefficients nuls. La forme logique peut être vue comme l'intersection logique de 2 parties : transformation des coefficients non nuls $\wedge$ transformation des coefficients nuls.

La deuxième partie de la forme logique est l'union de chaque matrice de factorisation qui n'apparaît pas parmi les coefficients nuls, appliquée sur toutes les parties des deux régions composites. Cela correspond à l'union de chaque matrice de factorisation qui peut être appliquée entre les parties des deux régions composites impliquées dans la relation.

Exemple 14.

Continuons l'Exemple 13. La première partie du résultat, définie à partir des coefficients non nuls (c_2=1 et c_8=1) conduit à :

$$(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{inside}, B_j \rangle)$$
$$\wedge\ (\exists k \in 1..n, \forall l \in 1..m \mid \langle A_k, \text{disjoint}, B_l \rangle$$

La seconde partie du résultat, définie à partir des coefficients nuls (c_1=c_3=c_4=c_5=c_6=c_7=0) se réécrit comme suit :

$$(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, \text{inside}, B_s \rangle \vee \langle A_r, \text{disjoint}, B_s \rangle)$$

Ainsi, l'expression logique finale est l'intersection de ces deux parties :

$(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{inside}, B_j \rangle)$

$\wedge$ $(\exists k \in 1..n, \forall l \in 1..m \mid \langle A_k, \text{disjoint}, B_l \rangle$

$\wedge$ $(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, \text{inside}, B_s \rangle \vee \langle A_r, \text{disjoint}, B_s \rangle)$

où A_i, B_j, A_k, B_l, A_r et B_s sont les parties des régions composites A et B impliquées dans la relation spécifiée par la matrice R.

Une configuration spatiale possible des deux régions composites A et B vérifiant la relation topologique R est exemplifiée en Figure 32. □

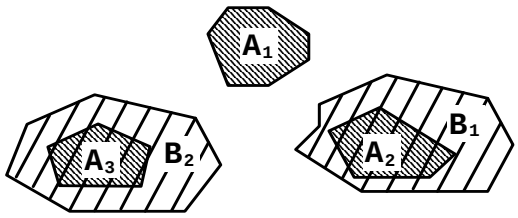


Figure 32. Une configuration spatiale des deux régions composites *A* et *B* correspondant à la matrice *R* de l'Exemple 13

Les solutions pour les 16 matrices qui correspondent à des relations entre deux régions composites sont présentées en Annexe.

4.7 Conclusion de l'étude

Pour conclure cette étude d'expressivité, il ressort des comparaisons faites dans cette partie que les différentes propositions d'extensions spatiales d'OCL basées sur CBM ont un pouvoir d'expression équivalent. Il en va de même pour les propositions d'extensions spatiales d'OCL basées sur 9IM.

Par rapport aux remarques faites en Section 2.2.2, si on considère les relations de base entre deux régions simples, 9IM distingue plus de relations que CBM ; par conséquent, OCL_{9IM} est plus expressif qu'OCL_{CBM}. Cependant, l'intégration dans OCL_{CBM} de la relation "*cross*" et des opérations d'extraction de frontières, évoquées en fin de Section 2.2.2, rendrait OCL_{CBM} plus expressif que OCL_{9IM}.

On remarque ici que l'intégration des relations de base de CBM ou 9IM dans OCL, couplée à l'utilisation des opérations standards sur les collections, permet de décrire des relations topologiques sur des régions composites. Cependant, dans un souci de simplicité de spécification de contraintes, nous avons proposé d'autres langages permettant d'exprimer plus simplement les relations topologiques entre régions compo-

sites. Ainsi, la différence entre des langages comme OCL_{CBM} et OCL_{TRCR} se situe au niveau de la facilité d'expression des contraintes. Quant à P-OCL, il permet de simplifier l'expression de certaines contraintes nécessitant de prendre en compte les nombres d'objets impliqués dans les relations spatiales.

Chapitre 5
Implémentation

Le but de cette partie est de présenter l'outil grâce auquel nous avons testé la proposition des langages de contraintes spatiales OCL_{9IM}, et son application aux géométries composites, ainsi que OCL_{ADV}. Un intérêt majeur est de faciliter le passage entre la description conceptuelle des contraintes et leur implémentation dans les bases de données spatiales. Les contraintes OCL_{9IM} ou OCL_{ADV} sont traduites sous forme de triggers activés sur une base de données, ou bien sous la forme de requêtes permettant de tester l'intégralité de la base.

Parmi les outils qui supportent OCL, il existe un parseur qui inclut la génération de code d'OCL à SQL : *OCL2SQL*. D'autres implémentations d'OCL sont disponibles : *Octopus*, qui génère des expressions Java à partir de pré- et post-conditions OCL, *Bold for Delphi* qui peut générer du code et du SQL, etc [KLA05]. Nous avons choisi d'étendre l'outil *OCL2SQL* qui est open source afin de produire les mécanismes de vérification d'intégrité topologique pour Oracle Spatial.

5.1 Présentation d'OCL2SQL

Le programme open source OCL2SQL est un générateur puissant écrit en Java [DEM01] [FIN00] [DEM04] ; il a été développé à l'aide du compilateur SableCC [GAG98]. Il permet de générer automatiquement à partir d'une expression OCL *c*, une requête SQL (pour Oracle) qui sélectionne toutes les données qui ne satisfont pas *c*. Une fois intégré dans un trigger (sur l'insertion, la suppression et la mise à jour de données), la requête garantit la cohérence d'une base de données. En effet, lors de chaque modification de données, le trigger vérifie si la requête SQL produite retourne des tuples ; si ce n'est pas le cas, la mise à jour est acceptée, sinon la modification des données est rejetée. Par cette technique, il devient impossible d'insérer des données qui violent la contrainte.

Le principe de la génération de code est basé sur des patrons de conversion d'OCL vers SQL pour Oracle comme par exemple :

- conversion du quantificateur universel

  OCL : `collection -> forAll(x| exp_bool_avec_x)`

SQL :
```
not exists (collection
                minus
                select objet from table
                where exp_bool_avec_x)
```

- conversion du quantificateur existentiel

OCL : `collection -> exists(x| exp_bool_avec_x)`

SQL :
```
exists (collection
            intersect
            select objet from table
            where exp_bool_avec_x)
```

Le principe d'OCL2SQL est donc de générer une requête correspondant à la négation de l'expression à respecter dans la contrainte. Cette requête retournera ainsi tous les tuples qui violent la contrainte.

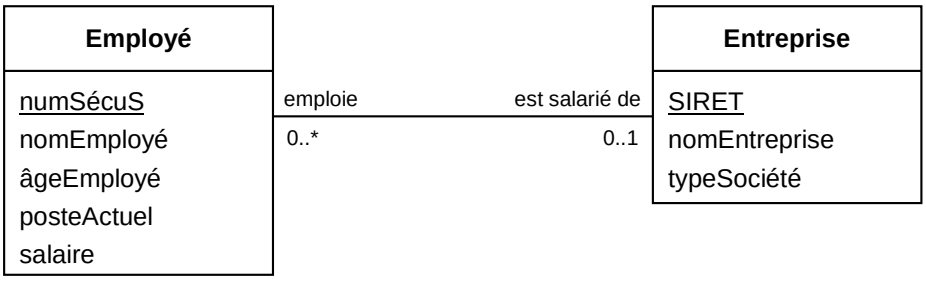


Figure 33. Extrait du Modèle Conceptuel de Données (MCD) d'un système de gestion d'entreprise.

Exemple 15.

Soit la contrainte OCL suivante portant sur le modèle de la Figure 33 (la contrainte a déjà été présentée dans la Section 2.4.3).

```
context Entreprise inv:
self.typeSociété = 'PME' implies self.Employé -> size() <= 499
```

Le code SQL suivant est la requête générée automatiquement à partir de cette contrainte, avec OCL2SQL.

```
select * from ENTREPRISE SELF
 where not (NOT SELF.TYPESOCIETE = 'PME'
            OR (select NVL(COUNT(*),0) from DUAL
                where SELF.NUMSECUS in
```

```
 (select NUMSECUS from EMPLOYE
  where EST_SALARIE_SIRET in
   (select SIRET from ENTREPRISE where SIRET = SELF.SIRET)))
<= 499 );
```

`EST_SALARIE_SIRET` correspond à la clef étrangère présente dans la table `EMPLOYE`.

Il est à noter qu'en pratique, OCL2SQL ne crée que des requêtes qui accèdent aux tables par l'intermédiaire de vues (appelées dans l'outil "*Object Views*"). Une "*Object View*" retourne exactement le contenu d'une table. Dans OCL2SQL, le nom d'une "*Object View*" est le nom de la table correspondante préfixée de "*OV_*". Afin de clarifier la présentation, nous omettons la notion d'"*Object View*" dans la thèse et nous enlevons tous les préfixes "*OV_*" dans les requêtes générées.

5.2 Principe de l'extension spatiale d'OCL2SQL

Nous avons inclus nos extensions spatiales basées sur 9IM à OCL2SQL en ajoutant les nouvelles syntaxes OCL proposées et en fournissant dans un premier temps, la génération automatique de triggers et de requêtes pour le SQL spatial supporté par Oracle. L'objectif final est de supporter à terme d'autres systèmes de gestion de bases de données tels que PostGIS par exemple. Les extensions implémentées sont OCL_{9IM} et OCL_{ADV}.

La Figure 34 présente les différents fichiers nécessaires en entrée de l'outil OCL2SQL étendu au spatial : le diagramme de classe UML sous forme de fichier XMI, un fichier contenant des informations sur les données géographiques, et le fichier contenant la ou les contraintes topologiques exprimées en OCL_{9IM} ou OCL_{ADV}. L'outil génère en sortie des scripts SQL pour Oracle Spatial assurant notamment la création de la base de données correspondant au diagramme UML, et des requêtes et triggers pour la vérification automatique des contraintes.

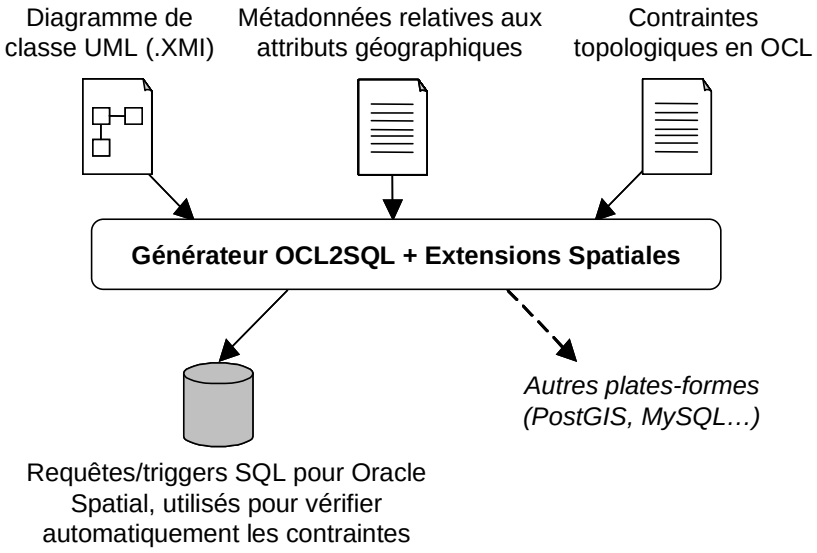


Figure 34. Entrées / sorties du générateur OCL2SQL Spatial

Les contraintes topologiques peuvent être spécifiées en OCL_{9IM} ou OCL_{ADV}

La démarche à suivre pour générer une requête de vérification de contraintes est la suivante :
1) charger le fichier XMI qui contient le diagramme de classe UML sur lequel porte la contrainte, exécuter l'analyse de ce fichier
2) charger les contraintes OCL (fichier OCL) ou les spécifier à l'aide de l'éditeur de contraintes de OCL2SQL,
3) spécifier le répertoire où seront sauvegardés les fichiers résultats,
4) charger le fichier de métadonnées géographiques qui spécifie notamment le référentiel spatial à utiliser pour chaque attribut spatial du modèle XMI,
5) spécifier si les clefs primaires doivent être créées ou si ce sont des attributs du modèle et dans ce cas, charger le fichier qui associe à chaque classe son attribut clef,
6) exécuter la génération de code.

Cinq fichiers sont alors générés, et affichés sur les onglets de sortie ("*Output*") : les scripts de générations de la base ("*Table Schema*") avec la création des tables, pour le cas où la base n'existe pas déjà, le remplissage des tables de métadonnées géographiques, les triggers de restriction sur les types géographiques, etc., les vues d'intégrité qui contiendront les tuples incohérents ("*Integrity Views*"), les triggers sur ces vues correspondant aux contraintes spécifiées ("*Trigger*"), et aussi des requêtes SQL associées ("*Query*") afin de pouvoir tester la contrainte de façon sporadique.

La Figure 35 est extraite d'un diagramme créé avec l'outil PowerAMC, sur la gestion des épandages.
Une zone autorisée à l'épandage (*Allowed_Area*) est identifiée par son code *AA_id* et comporte une date limite de validité (*validity_date*), une représentation géométrique (*geometry*) et également un attribut cal-

culé : le nombre de parcelles effectivement épandues qu'elle comprend (*spread_parcels_number*). Une zone autorisée est suivie par un dossier administratif (*Administrative_File*). C'est dans ce dossier, identifié par *AF_id*, que seront stockés des codes de rapport d'anomalies liées au suivi de l'épandage (*anomaly_report*) le cas échéant. Un dossier peut servir à gérer plusieurs zones autorisées.

Une zone autorisée comprend de zéro à *n* parcelles épandues (*Spread_Parcel*). Une parcelle épandue, identifiée par *SP_id*, comporte une date d'enregistrement (*date_of_record*), c'est-à-dire la date à laquelle l'épandage a eu lieu, et une représentation géométrique (*geometry*). Une parcelle ne devrait être épandue que sur une surface autorisée.

Sur une parcelle, on a épandu des boues (*Numbered_Pack_of_Matters*) identifiées par *NPM_id*, d'un certain type (*organic_matter_type*), dans une certaine quantité (*quantity*) et unité (*unit*). Une boue n'est utilisée que sur zéro ou une parcelle épandue.

Les zones bâties (*Built_Area*) sont identifiées par leur *id* et représentées géographiquement par un ensemble de régions (*Set(Polygon)*).

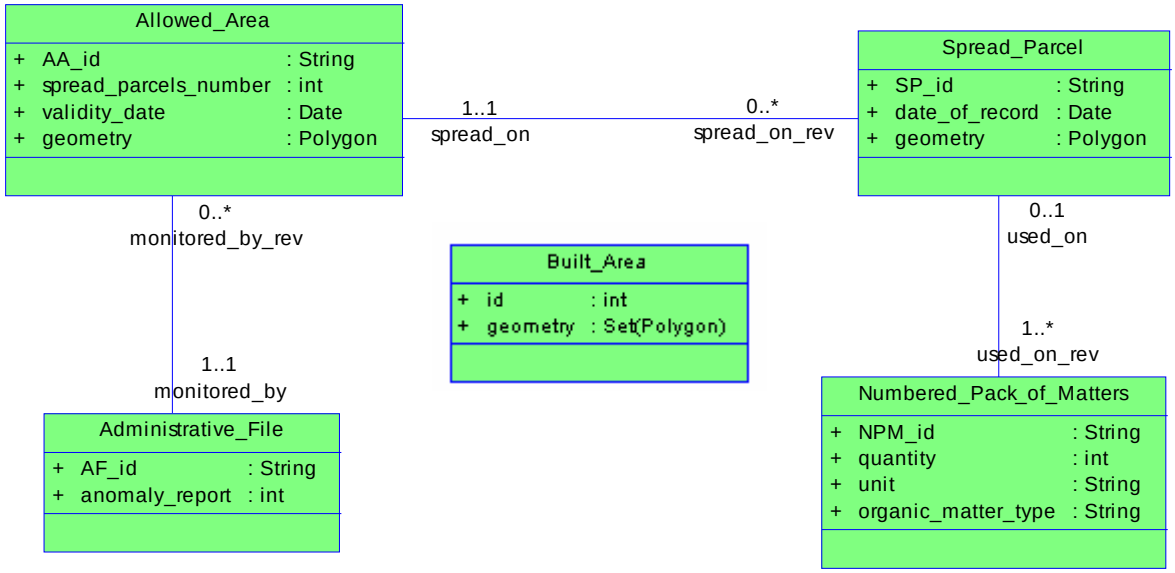


Figure 35. Extrait de diagramme pour la gestion des parcelles épandues issu de PowerAMC

PowerAMC permet d'exporter ce diagramme sous forme d'un fichier XMI donné en Figure 36. Dans le fichier produit, chaque classe est spécifiée avec ses attributs entre des balises. Les codes de type des attributs (*DT1*, *DT4*, etc.) renvoient à un contenu entre deux balises en fin de fichier qui contient le code et le nom complet du type. Dans le fichier XMI, Figure 36, l'attribut *geometry* de la classe *Allowed_Area* est de type *DT4*. Ce code correspond au type géométrique *Polygon*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE XMI (View Source for full doctype...)>
<XMI xmi.version="1.0">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>PowerAMC</XMI.exporter>
        </XMI.documentation>
        <XMI.metamodel xmi.name="UML" xmi.version="1.3" />
    </XMI.header>

    <XMI.content>
        <Model_Management.Model xmi.id="M1">
            <Foundation.Core.ModelElement.name>Exemple_OCL2SQL</Foundation.Core.ModelElement.name>
            <Foundation.Core.Namespace.ownedElement>
                <Foundation.Core.Class xmi.id="C1.1">
                    <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false" />
                    <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false" />
                    <Foundation.Core.ModelElement.name>Allowed_Area</Foundation.Core.ModelElement.name>
                    <Foundation.Core.Classifier.feature>
                        <Foundation.Core.Attribute xmi.id="a1.1.1">
                            <Foundation.Core.StructuralFeature.changeability xmi.value="changeable" />
                            <Foundation.Core.ModelElement.name>AA_id</Foundation.Core.ModelElement.name>
                            <Foundation.Core.Feature.visibility xmi.value="public" />
                            <Foundation.Core.StructuralFeature.type>
                                <Foundation.Core.DataType xmi.idref="DT1" />
                            </Foundation.Core.StructuralFeature.type>
                        </Foundation.Core.Attribute>
                        [...]
                        <Foundation.Core.Attribute xmi.id="a1.1.4">
                            <Foundation.Core.StructuralFeature.changeability xmi.value="changeable" />
                            <Foundation.Core.ModelElement.name>geometry</Foundation.Core.ModelElement.name>
                            <Foundation.Core.Feature.visibility xmi.value="public" />
                            <Foundation.Core.StructuralFeature.type>
                                <Foundation.Core.DataType xmi.idref="DT4" />
                            </Foundation.Core.StructuralFeature.type>
                        </Foundation.Core.Attribute>
                    </Foundation.Core.Classifier.feature>
                </Foundation.Core.Class>
                [...]
                <Foundation.Core.Association xmi.id="A1.3">
                    <Foundation.Core.ModelElement.name>Association_4</Foundation.Core.ModelElement.name>
                    <Foundation.Core.Association.connection>
                        <Foundation.Core.AssociationEnd xmi.id="e1.3.1">
                            <Foundation.Core.AssociationEnd.aggregation xmi.value="none" />
                            <Foundation.Core.AssociationEnd.changeability xmi.value="changeable" />
                            <Foundation.Core.AssociationEnd.isNavigable xmi.value="false" />
                            <Foundation.Core.AssociationEnd.multiplicity>0..*</Foundation.Core.AssociationEnd.multiplicity>
                            <Foundation.Core.ModelElement.name>spread_on_rev</Foundation.Core.ModelElement.name>
                            <Foundation.Core.AssociationEnd.isOrdered xmi.value="false" />
                            <Foundation.Core.AssociationEnd.visibility xmi.value="public" />
                            <Foundation.Core.AssociationEnd.type>
                                <Foundation.Core.Class xmi.idref="C1.3" />
                            </Foundation.Core.AssociationEnd.type>
                        </Foundation.Core.AssociationEnd>
                        <Foundation.Core.AssociationEnd xmi.id="e1.3.2">
                            <Foundation.Core.AssociationEnd.aggregation xmi.value="none" />
                            <Foundation.Core.AssociationEnd.changeability xmi.value="changeable" />
                            <Foundation.Core.AssociationEnd.isNavigable xmi.value="false" />
                            <Foundation.Core.AssociationEnd.multiplicity>1..1</Foundation.Core.AssociationEnd.multiplicity>
                            <Foundation.Core.ModelElement.name>spread_on</Foundation.Core.ModelElement.name>
                            <Foundation.Core.AssociationEnd.isOrdered xmi.value="false" />
                            <Foundation.Core.AssociationEnd.visibility xmi.value="public" />
                            <Foundation.Core.AssociationEnd.type>
                                <Foundation.Core.Class xmi.idref="C1.1" />
                            </Foundation.Core.AssociationEnd.type>
                        </Foundation.Core.AssociationEnd>
                    </Foundation.Core.Association.connection>
                </Foundation.Core.Association>
            </Foundation.Core.Namespace.ownedElement>
        </Model_Management.Model>
            [...]
        <Foundation.Core.DataType xmi.id="DT4">
            <Foundation.Core.ModelElement.name>Polygon</Foundation.Core.ModelElement.name>
        </Foundation.Core.DataType>
    </XMI.content>
</XMI>
```

Figure 36. Fichier XMI correspondant au diagramme de la Figure 35

A l'aide d'OCL2SQL étendu au spatial, on peut spécifier des contraintes topologiques sur les éléments du diagramme de la Figure 35.

Par exemple, on souhaite exprimer le fait que tous les bâtiments d'une zone bâtie sont disjoints des zones où l'épandage est autorisé. Chaque zone construite est représentée géographiquement par une région composite qui correspond au type *Set(Polygon)*.

Avec notre extension spatiale d'OCL2SQL, on charge le fichier XMI généré par Power-AMC avant de l'analyser en exécutant le parseur dans l'onglet "*Input*" présenté en Figure 37. Puis on entre la ou les contraintes OCL, soit en chargeant directement des fichiers OCL (*Load*), soit en les créant à l'aide de l'éditeur de contraintes d'OCL2SQL.

Dans la contrainte de la Figure 37, toute forme de relation topologique entre les surfaces où l'on peut épandre (*Allowed_Area*) et tous les bâtiments des surfaces construites (*Built_Area*) autre que la disjonction est interdite.

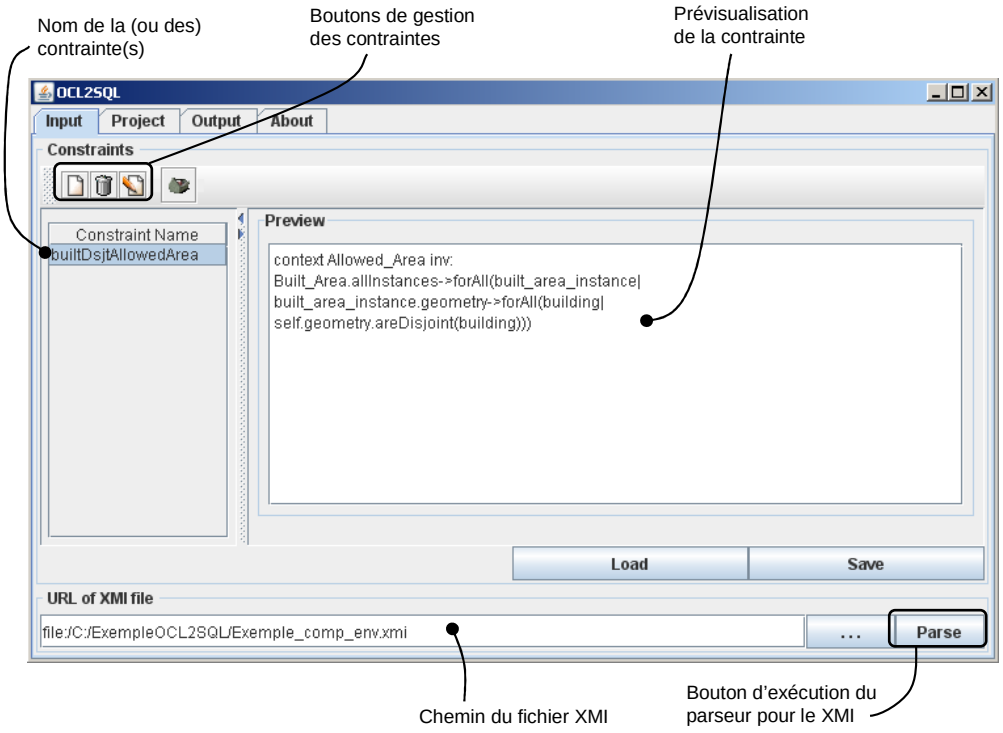


Figure 37. Onglet "Input" de l'outil OCL2SQL

Une fois le fichier XMI analysé et les contraintes spécifiées, l'onglet "*Project*" illustré en Figure 38, permet de fixer certains paramètres tels que les répertoires courants. Le fichier contenant les métadonnées des attributs géographiques du fichier XMI contient notamment pour chaque attribut géométrique, des informations sur leurs dimensions : libellé (latitude/longitude, x/y/z, etc.), valeur minimale et maximale, précision ; des informations sur le système de référence utilisé (code *srid*).

Enfin nous avons également ajouté au cadre "*primary keys*" la possibilité de spécifier si :

- les clefs primaires doivent être ajoutées automatiquement dans le schéma physique généré (*Generated*) ; dans ce cas un nouvel attribut clef primaire est ajouté dans les tables du schéma physique généré,
- ou si ce sont des attributs existants (*Already exist*) déjà spécifiés dans le diagramme UML ; dans ce dernier cas, un fichier contenant les noms des attributs étant identifiants pour chaque classe est entré.

Le bouton "*Execute Project*" lance la génération des différents scripts SQL pour Oracle.

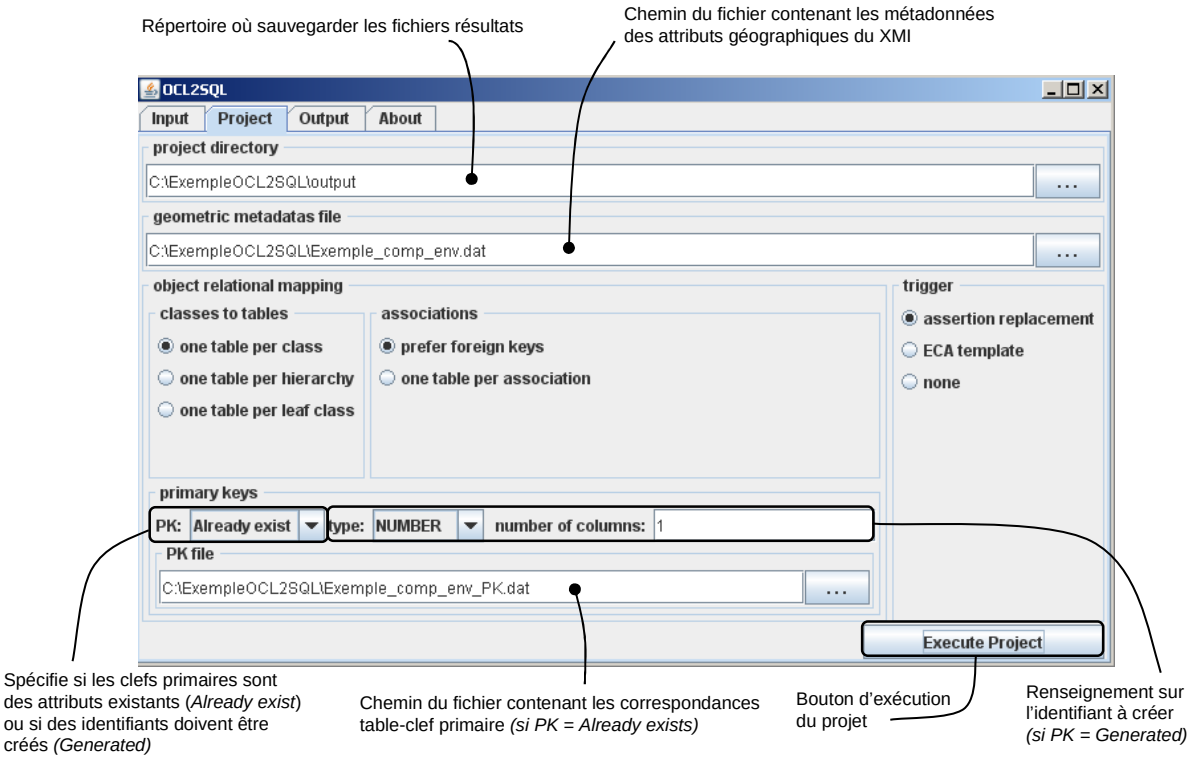


Figure 38. Onglet "Project" de l'outil OCL2SQL

L'exécution du projet génère plusieurs fichiers. Leur contenu est retranscrit dans les cinq onglets de l'onglet "*Output*". Chacun de ces onglets est également sauvegardé dans un script SQL du même nom.

En Figure 39, l'onglet "*Table Schema*" contient les instructions SQL de création de la base correspondant au diagramme de classe du fichier XMI passé en entrée. Il comprend :

- la création des tables avec des structures adaptées pour stocker les différents types géométriques,
- l'insertion dans les tables Oracle des métadonnées géographiques,
- la création des index spatiaux,

- les mécanismes de vérification de type géométrique spécifique ; lorsque le type d'un attribut géométrique est précisé dans le diagramme de classe (*Polygon*, *Set(Polygon)*), un trigger limite l'insertion et la modification à des attributs de ce type. Il s'agit en fait d'une contrainte sur le type similaire à celles présentées en Section 3.1.

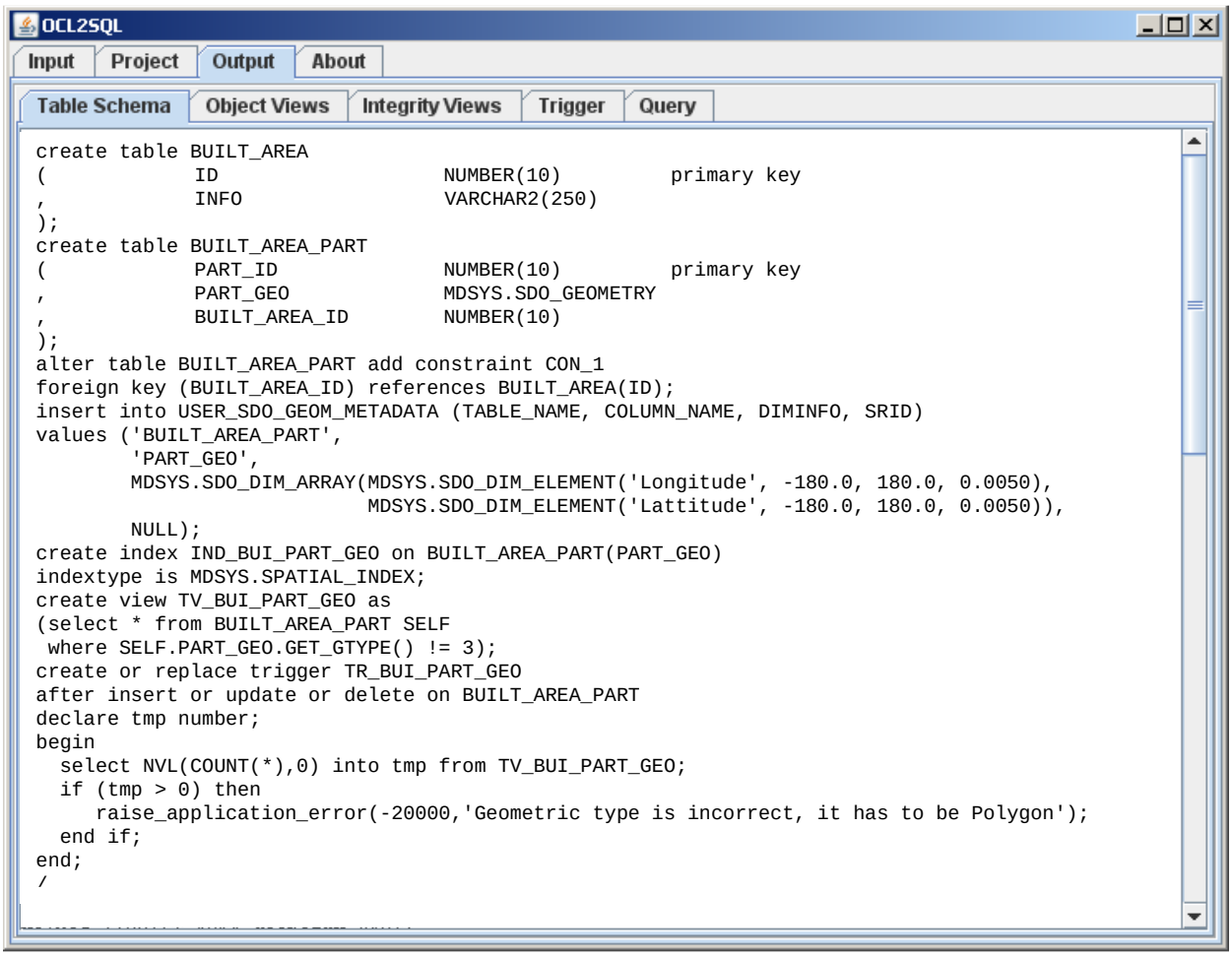


Figure 39. Onglet "Output – Table Schema" de l'outil OCL2SQL

Ainsi, une correspondance entre les modèles conceptuel et physique de schémas de bases de données a été réalisée. Pour les géométries simples, l'attribut spatial est transcrit de façon triviale. Dans le cas de géométries composites, deux possibilités principales s'offraient à nous :

- la conversion d'une classe spatiale en deux tables. Une classe spatiale du modèle conceptuel est convertie en deux tables dans la base de données. La première table est la classe elle-même avec ses attributs alphanumériques. Chaque tuple de la seconde table contient un composant des géométries composites. Ainsi, on peut facilement atteindre chaque partie des géométries composites avec SQL. Par exemple, la table *Built_Area* de la Figure 35 possède un attribut de type

Set(Polygon). Le modèle physique pour la classe spatiale *Built_Area* illustre ce type de schéma physique. Il s'écrit comme suit :

```
BUILT_AREA(id, info)
BUILT_AREA_PART(part_id, part_geo, #built_area_id)
```

- la conversion d'une classe spatiale en une table. Si le SGBD cible supporte les géométries composites, une autre méthode est de convertir chaque classe spatiale du modèle conceptuel en une seule table dans la base de données ; cette table correspond à la classe elle-même et chaque géométrie composite est stockée dans un attribut "géométrie". Dans ce cas, le type de l'attribut est *Set(BasicGeoType)*.

Dans notre extension d'OCL2SQL, afin de pouvoir plus facilement accédé et manipulé chacun des composants des régions composites en SQL, nous avons implémenté la première possibilité. En effet, si l'on considérait la seconde possibilité, écrire par exemple une requête SQL pour calculer la surface de la plus petite partie d'une région composite serait plus difficile pour les utilisateurs de la base de données. Cette requête SQL implique l'utilisation d'un opérateur de décomposition de régions composites [CLE95].

Le code SQL généré par OCL2SQL pour l'exemple présenté est donné en Figure 39.

L'onglet "*Query*", que nous avons ajouté (voir Figure 40), fournit la (ou les) requête(s) permettant de vérifier la (ou les) contrainte(s) OCL de manière sporadique sur la base. Cette requête retourne tous les tuples qui violent la contrainte au moment de son exécution.

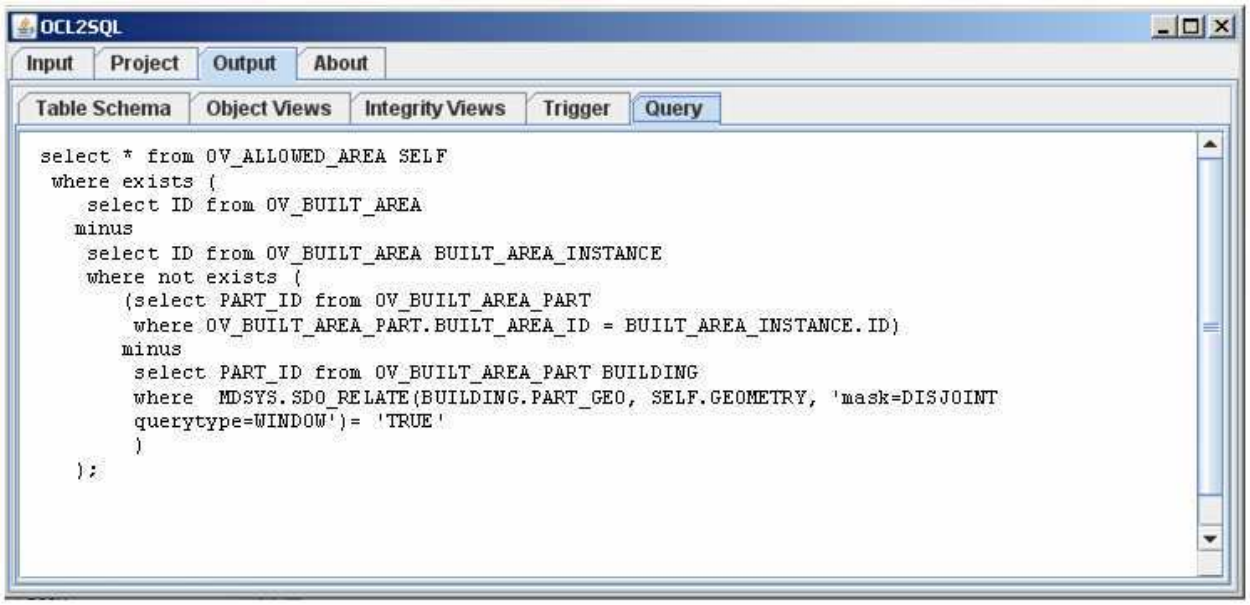


Figure 40. Onglet "Output - Query" de l'outil OCL2SQL

La requête de l'exemple de la Figure 40 retourne tous les tuples de *Allowed_Area* qui sont en interaction avec au moins un bâtiment, c'est-à-dire les tuples qui violent la contrainte.

L'onglet "*Integrity Views*" fournit une vue basée sur cette requête. Ainsi, cette vue contient les tuples qui violent la contrainte. Le trigger défini dans l'onglet "*Trigger*" interdit l'insertion de tuples dans cette vue. Autrement dit, le trigger sur la vue d'intégrité empêche l'insertion de données erronées dans la base. Cependant, au vu des temps d'exécution de certains triggers notamment lorsqu'ils portent sur des données spatiales, il est préférable de tester la base sporadiquement à l'aide de la requête.

5.3 Règles de génération automatique : extensions d'OCL vers SQL Spatial

Si on s'intéresse à tous les langages proposés dans cette thèse, comme les expressions OCL_{ADV}, OCL_{TRCR} et P-OCL peuvent être réécrites en expressions OCL_{9IM} ou OCL_{CBM} (cf. Section 4), nous n'avons qu'à étudier la génération de SQL à partir d'OCL_{9IM} et d'OCL_{CBM}. En effet, la génération de code à partir d'OCL_{ADV} (respectivement d'OCL_{TRCR} et P-OCL) vers SQL reste possible en réécrivant les expressions OCL_{ADV} (respectivement OCL_{TRCR} et P-OCL) en expressions OCL_{9IM} (respectivement OCL_{CBM}), et ensuite en SQL.

SQL Spatial pour Oracle est basé sur les relations topologiques de 9IM, les opérations de OCL_{9IM} sont donc présentes dans le SQL Spatial d'Oracle comme indiqué dans le Tableau 11. Il est aussi possible de convertir OCL_{CBM} en SQL Spatial pour Oracle, en utilisant les équivalences du Tableau 12.

Tableau 11. Règles de passage d'OCL_{9IM} à SQL pour Oracle Spatial

OCL_{9IM}	**SQL pour Oracle Spatial**
`in`	`INSIDE`
`coveredBy`	`COVEREDBY`
`equal`	`EQUAL`
`contains`	`CONTAINS`
`covers`	`COVERS`
`touch`	`TOUCH`
`overlap`	`OVERLAPBDYINTERSECT`
`disjoint`	`DISJOINT`

Tableau 12. Règles de passage d'OCL_{CBM} à SQL pour Oracle Spatial

OCL_{CBM}	SQL pour Oracle Spatial
`in`	`INSIDE+COVEREDBY+EQUAL`
`touch`	`TOUCH`
`overlap`	`OVERLAPBDYINTERSECT`
`disjoint`	`DISJOINT`

Dans tous les cas, pour pouvoir générer du code pour un SGBD spécifique, une conversion directe entre les opérations topologiques de 9IM ou de CBM d'une part, et les opérations SQL d'autre part, doit être possible.

Concernant l'implémentation proposée dans OCL2SQL, nous avons intégré OCL_{9IM} et OCL_{ADV} en utilisant les conversions présentées en Tableau 11 et en Tableau 7.

Le patron de conversion d'OCL2SQL pour des opérations 9IM sur les géométries simples est le suivant :

OCL : `col_geo1.op_spatial(col_geo2)`

SQL :
```
MDSYS.SDO_RELATE(col_geo1, col_geo2,
'mask=eq_op_spatial querytype=WINDOW')= 'TRUE'
```

Nous proposons le patron suivant pour convertir en SQL, les opérations de décomposition de régions composites d'OCL_{9IM} (*forAll*, *exists*, *select*) introduites dans la Définition 8 de la Section 3.3.2. Soit le schéma physique suivant stockant des régions composites : `T(t_id,...)`, `T_Part(part_id, geo_part, #t_id)`. La fonction *e* convertit une expression OCL en SQL.

OCL : `self.composite_geo_attrib->forAll(x|bool_exp_with_x)`

SQL :
```
select * from T where not
    (not exists(select part_id from T_Part
    where T_Part.t_id = T.t_id
    minus
    select part_id from T_Part where e(bool_exp_with_x)))
```

OCL : `self.composite_geo_attrib->exists(x|bool_exp_with_x)`

SQL :
```
select * from T where not
     (exists(select part_id from T_Part
     where T_Part.t_id = T.t_id
     intersect
     select part_id from T_Part where e(bool_exp_with_x)))
```

```
OCL :  self.composite_geo_attrib->select(x|bool_exp_with_x)
SQL :  select * from T where not
           (select part_id from T_Part
           where T_Part.t_id = T.t_id
           minus
           select part_id from T_Part where not e(bool_exp_with_x)))
```

5.4 Bilan

Dans cette partie, nous avons présenté l'implémentation des extensions spatiales OCL_{9IM} et OCL_{ADV} dans l'outil OCL2SQL. Nous avons donc maintenant un générateur de code opérationnel qui permet de produire du code SQL pour Oracle à partir de contraintes d'intégrité spatiales. Ce code permet de tester l'intégrité d'une base de données, ou bien peut être intégré dans des triggers afin d'interdire les modifications de données qui violeraient les contraintes.

La méthodologie consiste à tracer les diagrammes UML avec un AGL puis de les enregistrer au format XMI pour les exploiter avec OCL2SQL, afin de spécifier les contraintes d'intégrité (spatiales ou non spatiales). A partir de ces différentes contraintes, des mécanismes de test d'intégrité peuvent être ainsi générés.

En utilisant OCL2SQL pour l'implémentation de notre extension spatiale, nous avons ainsi pu bénéficier d'un logiciel ouvert offrant un parseur et un générateur de code particulièrement complet ; la génération de code SQL à partir de toutes principales opérations d'OCL étant déjà implémentée.

Cette approche nous est apparue la plus efficace et la plus simple à mettre en œuvre. Une nouvelle phase d'implémentation pourrait permettre de supporter d'autres SGBD.

Chapitre 6
Expérimentation avec une application agricole de gestion des épandages

Nous présentons dans ce chapitre notre première expérimentation pour la validation des langages de contraintes spatiaux proposés. Nous expérimentons nos langages par la modélisation de contraintes spatiales portant sur un projet développé par le Cemagref en lien avec le Ministère de l'Agriculture, le Ministère de l'Environnement, l'ADEME, les Agences de l'Eau. Il s'agit de SIGEMO, le Système Informatisé de Gestion des Epandages de Matières Organiques.

6.1 Présentation de SIGEMO

Nous avons réalisé une expérimentation sur SIGEMO. Le Système Informatisé de Gestion des Epandages de Matières Organiques répond à une forte demande d'information émanant de nombreuses institutions. En effet, un encadrement et un suivi rigoureux des retours au sol des matières organiques par les services de l'Etat est indispensable pour assurer de bonnes pratiques d'un point de vue environnemental, économique et sanitaire.

De nombreuses institutions sont demandeuses d'information sur les épandages : les chambres d'agriculture, les agences de l'eau, l'ADEME, les collectivités locales… La Figure 41 résume les fonctions de ces différents acteurs autour du système SIGEMO. SIGEMO permet de centraliser les informations sur les épandages dans une base de données nationale, accessible depuis le Web par une application spécifique. Les données sur des propositions d'épandage sont entrées dans le système d'information par des bureaux d'étude pour le compte des producteurs de boues. Ensuite, différentes Directions Départementales et Régionales utilisent les informations de SIGEMO pour l'instruction des propositions d'épandage (c'est-à-dire des plans d'épandage). Ces données sont également utiles aux collectivités locales et agences de l'eau, entre autres organismes pour des expertises ou en simple consultation.

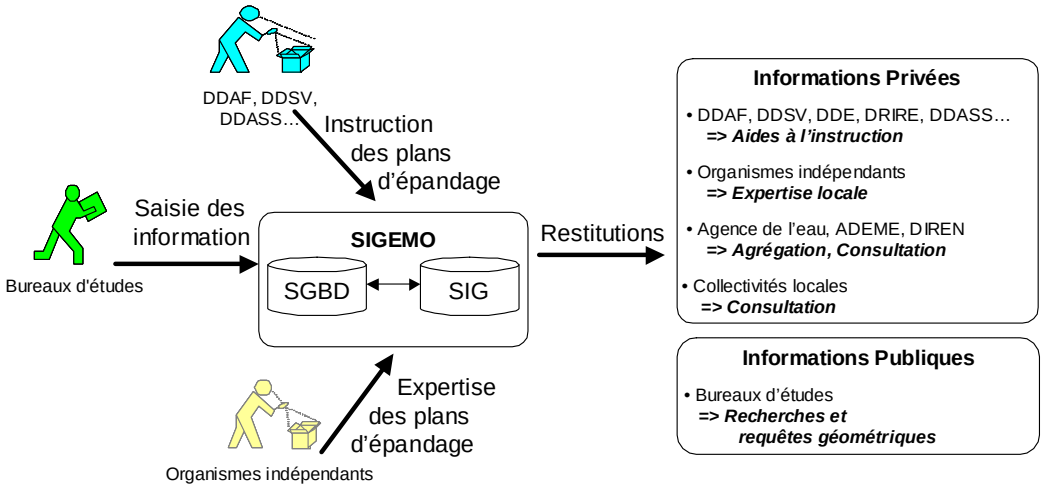


Figure 41. Principaux acteurs et fonctions de SIGEMO [SOU04]

Les accès à la base se font par le Web grâce à une application spécifique

6.1.1 Epandage et enjeux environnementaux

L'épandage sur les terres cultivées a toujours été considéré comme un excellent moyen de recycler les matières organiques telles que par exemple les boues de stations d'épuration ou encore les déchets des industries agroalimentaires [VAN01] [REN01] [SOU03]. Cette technique consistant à répandre les boues sur les champs, permet aux agriculteurs de contribuer à la fertilisation des parcelles, et ce, à très faible coût, tout en valorisant les déchets. Pour s'alimenter et se développer, les plantes doivent en effet prélever dans le sol des composants minéraux et organiques, apportés en grande quantité par l'épandage (nutriments constitués par l'azote, le phosphore, le potassium, différents oligo-éléments). L'impact des matières organiques épandues utilisées sur les cultures repose sur des bases scientifiques et techniques aujourd'hui bien connues [ADE04][BOR01b].

Au-delà des bénéfices directs pour la production agricole, l'épandage de matières organiques réduit aussi l'utilisation d'engrais minéraux, ce qui présente l'avantage important de diminuer l'exploitation des ressources naturelles minérales et énergétiques nécessaires à la fabrication des engrais. Enfin, l'épandage agricole évite de recourir à des méthodes strictement éliminatrices (incinération, mise en décharge) qui occasionnent dans certains cas des répercussions environnementales ou sociales négatives. Globalement, de nos jours, le recyclage par épandage est considéré comme étant un des meilleurs d'un point de vue environnemental, sanitaire et économique.

Du fait de la quantité importante de produits épandus en France (plusieurs millions de tonnes annuelles) et malgré leurs nombreux avantages, les pratiques d'épandage agricole nécessitent un suivi minutieux et fiabilisé [SOU03]. En effet, un épandage excessif et irraisonné pourrait tout bonnement mener à une pollution des sols ; par exemple, lorsque les matières viennent perturber l'équilibre d'un cours d'eau ou d'un sol qui est incapable de les éliminer. C'est typiquement le cas d'une pollution par excès d'azote entraîné par une trop forte concentration de matières en un même point. L'azote est un élément vital à la production végétale. Les plantes l'absorbent sous forme de nitrates. Néanmoins, si trop d'azote est apporté, il est alors lessivé par les pluies et participe à la pollution des eaux de surface ou souterraines. Il en va de même pour les excès de phosphore et potassium.

Il est donc primordial d'avoir une vision précise et un suivi dans le temps des épandages sur tout le territoire français [BOU02], afin de :
- connaître et vérifier les quantités, les zones et les matières épandues ; une règlementation fixe précisément ce qui est autorisé ;
- recouper différentes données et différentes périodes.

6.1.2 Organisation de l'épandage en France

Les organismes producteurs de matières (principalement les stations d'épuration et les industries agro-alimentaires) souhaitant avoir recours à l'épandage doivent établir un projet d'épandage et le faire transmettre aux différents services de l'Etat. Le travail de constitution du projet d'épandage est en principe confié à un bureau d'étude spécialisé. On trouve, entre autres, dans ces dossiers de projets d'épandage, les quantités à épandre, les matières, les zones à traiter, les parcelles agricoles concernées.

Une fois transmis aux services de l'Etat compétents, la proposition d'épandage est instruite et évaluée. Aussi, plusieurs Départements mandatent des organismes indépendants spécialisés (par exemple des Chambres d'Agricultures) pour expertiser de manière plus approfondie les projets d'épandage soumis, ainsi que la répercussion des futures pratiques sur l'environnement proche ou plus lointain. Dans le futur, le recours à de tels organismes devrait tendre à se généraliser [GAL01]. D'autres acteurs peuvent intervenir, solliciter des informations sur les épandages, à différents niveaux (Commune, Agence de l'Eau, Ministère…). La Figure 42 résume le parcours d'un projet d'épandage parmi les principaux acteurs.

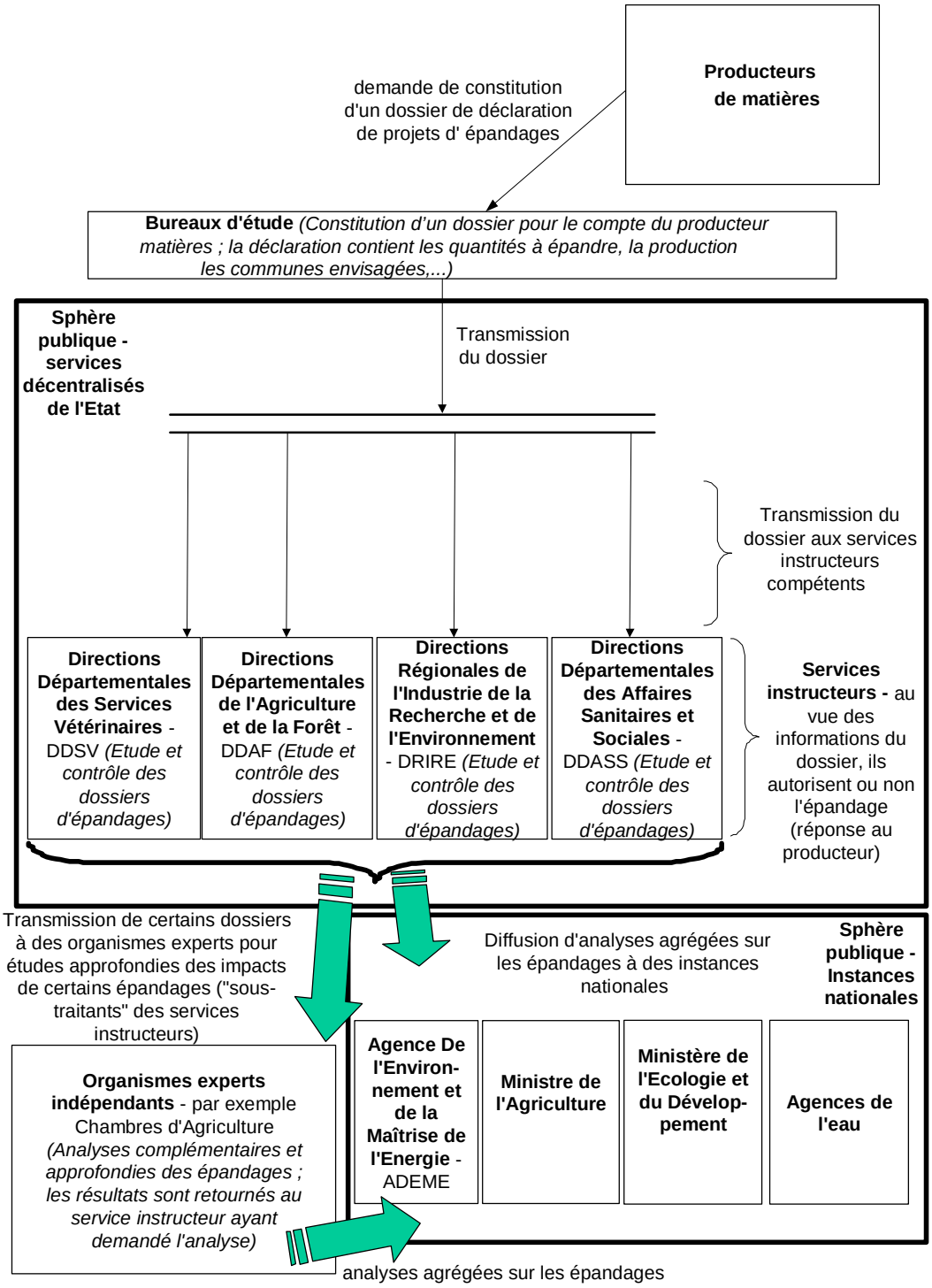


Figure 42. Instruction d'un projet d'épandage par les services de l'Etat

Depuis 2006, le système d'information national SIGEMO a été mis en place pour permettre aux différents acteurs de saisir et transmettre les projets d'épandage par Internet. En fonction de leur rôle, les acteurs peuvent ainsi saisir ou accéder à l'information concernant un plan d'épandage afin d'instruire un dossier

ou de l'expertiser. Toutes les informations sont centralisées dans la base de données nationale SIGEMO. L'outil permet principalement le partage de l'information concernant les plans d'épandage.

6.1.3 Diagramme de classes de SIGEMO

SIGEMO regroupe les informations sur les projets d'épandage au niveau national. Un extrait simplifié du diagramme de classes tracé avec PowerAMC est présenté à la Figure 44.

Le système d'information de SIGEMO s'articule autour de 7 entités principales : le dossier, l'exploitation agricole, la parcelle, la zone d'aptitude, le périmètre d'épandage, le produit d'épandage et la station d'épuration. Nous décrivons ici brièvement ces entités et leurs attributs (voir Dictionnaire de Données SIGEMO).

- Le dossier du projet d'épandage (Classe *DOSSIER*). Il s'agit des dossiers de projet d'épandage contenant notamment les propositions de zones d'épandage, les produits, etc.

- La parcelle d'épandage (Classe *PARCELLE*). Il s'agit d'une parcelle proposée à l'épandage.

- La zone d'aptitude (Classe *ZONE_APTITUDE*). Chaque parcelle d'épandage est subdivisée en zone(s) d'aptitude, pour indiquer que telle ou telle sous-partie de la parcelle est apte à recevoir un ensemble de produits d'épandage en particulier. Dans l'exemple de la Figure 43, on a deux parcelles dont une est subdivisée en deux zones d'aptitude.

- Le produit d'épandage (Classe *PRODUIT*). Il s'agit de matières à épandre issues de stations d'épuration, d'installations d'élevage ou de sites industriels.

- La station d'épuration ou l'unité de traitement des sous-produits (Classe *STEP*). Ce sont les stations d'épuration (ou systèmes de traitement des eaux usées) et les unités de traitement des sous-produits (UTSP). Les UTSP visent à diminuer la charge polluante des sous-produits de l'épuration (boues, graisses, etc.) comme par exemple les décharges, les usines d'incinération, les usines de compostage…

- L'exploitation agricole (Classe *EXPLOITATION_AGRICOLE*). Unité de production géographiquement individualisée dans laquelle une ou plusieurs personnes ont des activités agricoles. Une exploitation agricole peut être localisée sur plusieurs communes, voire plusieurs départements. Dans le cadre de la thématique "Epandage des produits" et dans le système SIGEMO, seules celles concernées par

l'activité d'épandage seront prises en compte. Une exploitation agricole peut aussi intervenir en tant que producteur lorsqu'elle produit des effluents d'élevage ou lorsqu'elle utilise un produit normalisé ou homologué pour l'épandage. Les informations relatives aux exploitations agricoles relèvent du ou des producteurs de données.

- Le périmètre d'épandage (Classe *PERIMETRE*). Le périmètre d'épandage est un ensemble des parcelles d'épandage.

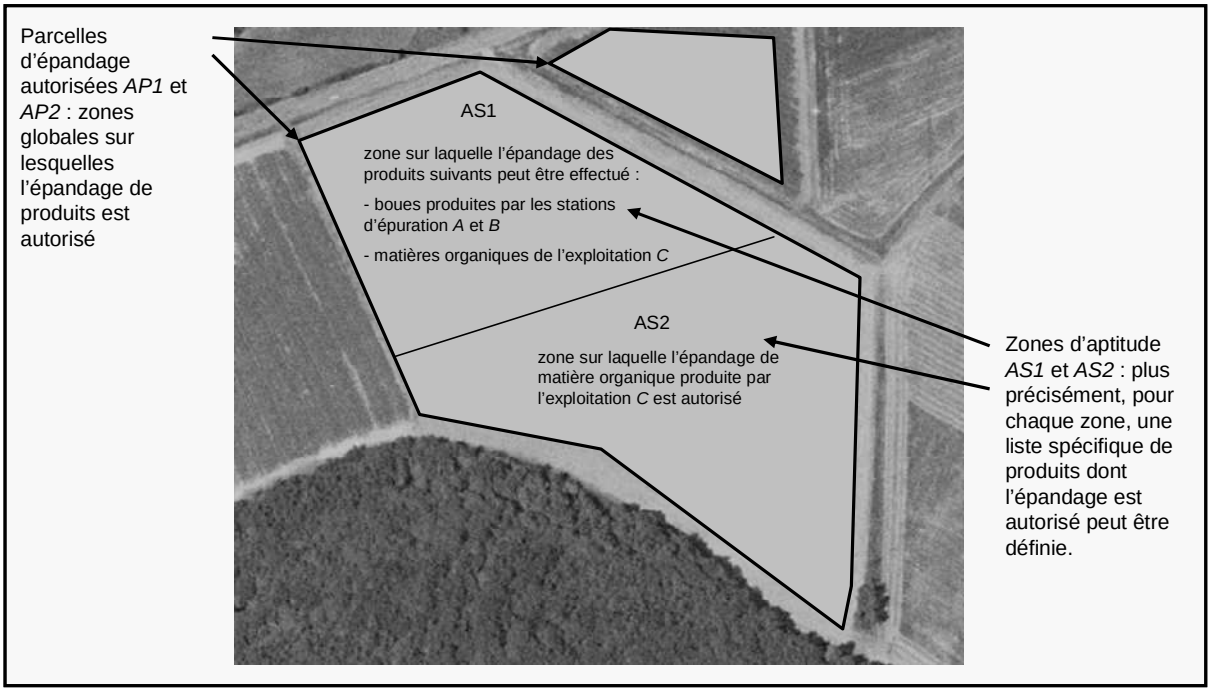


Figure 43. Division de parcelles d'épandage en zones d'aptitude

La représentation géographique des objets est modélisée par des classes en gris claire dans le schéma de la Figure 44. Il est à noter que deux classes ont été définies pour chacun des objets métiers spatialisés (une contenant les données alphanumériques et une autre les données géographiques). Cela est dû au fait que les données alphanumériques et géographiques du système SIGEMO sont gérées respectivement dans un SGBD et dans un SIG.

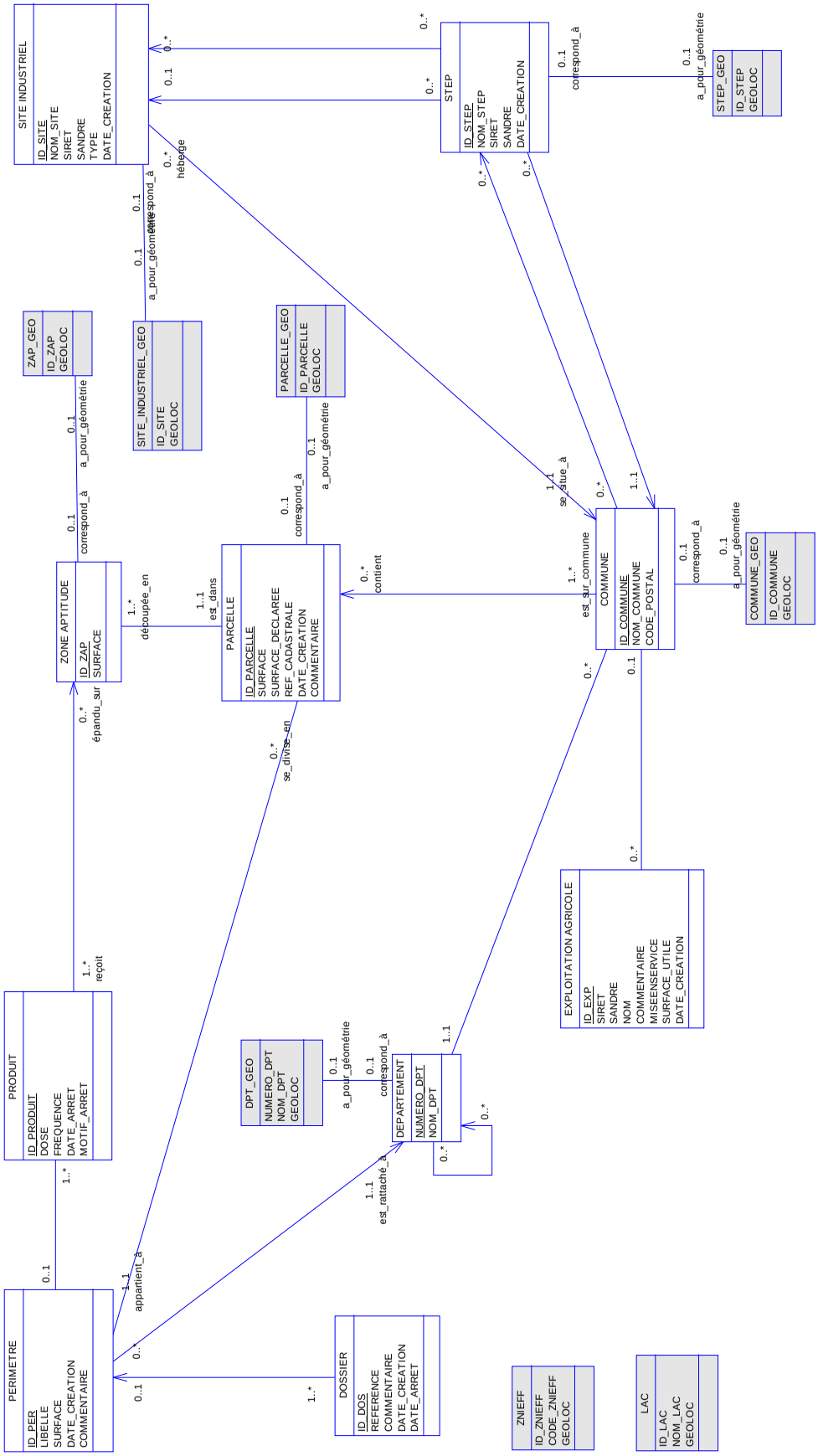


Figure 44. Extrait simplifié du diagramme de classe de SIGEMO

6.2 Exemples de contraintes sur SIGEMO

Après une phase d'analyse au Cemagref, SIGEMO a été développé par la SSII Sword. Avec les experts métier, nous avons défini des contraintes que le système doit respecter afin d'évaluer :

- la qualité effective des données saisies par le biais de l'application SIGEMO,
- la qualité effective des données intégrées dans la base de données de SIGEMO, depuis d'autres sources de données[2].

La base de données de SIGEMO est gérée avec le SGBD Sybase pour les données alphanumériques, et avec le SIG MapInfo pour les données géographiques. Afin de simplifier les tests avec notre extension spatiale d'OCL2SQL, nous avons rapatrié toutes les données (de Sybase et de MapInfo) dans une seule base de données sous Oracle 10g, et ce sans modifier la structure générale du schéma. Ainsi, les requêtes SQL pour Oracle générée avec OCL2SQL ont pu être directement testées sur cette version de la base.

Nous avons défini les contraintes d'intégrité avec OCL_{9IM} puis généré la requête SQL correspondante. Comme nous le verrons dans cette Section, nous avons vérifié les contraintes d'intégrité de façon sporadique sur des jeux de données réels fournis à un des instants particuliers. Ces jeux de données provenaient de divers départements, notamment le Cher (18), l'Yonne (89) et la Haute-Loire (43). Le jeu de données de la Haute-Loire, par exemple, contient 940 parcelles et plus de 1500 zones d'aptitude.

La suite de cette partie est dédiée aux contraintes d'intégrité définies avec les experts métier. Dans un premier temps, les contraintes alphanumériques sont traitées.

6.2.1 Contraintes alphanumériques

Contrainte 1.
La surface épandable déclarée pour une parcelle est toujours inférieure ou égale à la surface effective de la parcelle. Des erreurs de saisie de ces surfaces peuvent conduire à des incohérences dans la base.
Chaque instance de la classe Parcelle a une surface épandable déclarée inférieure ou égale à sa surface totale.

```
context PARCELLE inv:
self.pcl_surface_declaree <= self.surface
```

A partir de cette contrainte, l'outil OCL2SQL génère la requête suivante :

```
select ID_PARCELLE, SURFACE_DECLAREE, SURFACE from PARCELLE SELF
where not (SELF.SURFACE_DECLAREE <= SELF.SURFACE);
```

L'exécution de cette requête sur la base SIGEMO a retourné 1 enregistrement. A l'exécution de cette requête, Oracle affiche le tuple potentiellement incohérent et le temps d'exécution de la requête comme suit :

```
ID_PARCELLE SURFACE_DECLAREE SURFACE
----------- ---------------- -------
       1386            14,01   13,97
Elapsed: 00:00:00.20
```

Ainsi, la parcelle ayant pour identifiant (*ID_PARCELLE*) 1386 viole la contrainte. En effet, la surface épandable déclarée (*SURFACE_DECLAREE*) est ici supérieure à la surface de la parcelle entière (*SURFACE*).

Une fois les enregistrements incohérents identifiés, l'utilisateur les analysera afin de déterminer les causes des problèmes. Sur cet exemple, l'incohérence peut être due à une erreur de saisie d'une des deux surfaces, voire les deux. L'utilisateur devra donc contacter les personnes concernées pour corriger cette erreur.

□

Contrainte 2.

Une parcelle est découpée en différentes zones en fonction des produits que l'on peut y épandre. Ces zones sont appelées zones d'aptitude. Une aptitude est la capacité à recevoir certains types de produits d'épandage. L'ensemble des zones d'aptitude d'une parcelle doit former une partition de celle-ci. Une parcelle est ainsi divisée en une ou plusieurs zones. La surface d'une zone d'aptitude est donc toujours au plus égale à la surface de celle de la parcelle correspondante.

```
context ZONE_APTITUDE inv:
self.surface <= self.est_dans.surface
```

La requête générée par OCL2SQL est la suivante :

[2] Avant l'existence de SIGEMO, plusieurs institutions avaient développé des applications issues d'initiative locale pour saisir des données liées à l'épandage. Lors du lancement de SIGEMO, certaines de ces données ont été intégrées dans la base de données nationale.

```
select * from ZONE_APTITUDE SELF
where not (SELF.SURFACE <= (select SURFACE from PARCELLE
                            where ID_PARCELLE in
                                (select ID_PARCELLE from ZONE_APTITUDE
                                 where ID_ZAP = SELF.ID_ZAP)));
```

Afin de mieux cerner les causes des incohérences, nous pouvons modifier la requête générée par OCL2SQL pour non seulement afficher les zones d'aptitude posant problème, mais aussi les surfaces des parcelles associées. On obtient alors :

```
select ID_ZAP, SELF.ID_PARCELLE, SELF.SURFACE, PARCELLE.SURFACE
from ZONE_APTITUDE SELF, PARCELLE
where PARCELLE.ID_PARCELLE = SELF.ID_PARCELLE
and not (SELF.SURFACE <= (select SURFACE from PARCELLE
                          where ID_PARCELLE in
                                (select ID_PARCELLE from ZONE_APTITUDE
                                 where ID_ZAP = SELF.ID_ZAP)));

    ID_ZAP ID_PARCELLE SELF.SURFACE PARCELLE.SURFACE
---------- ----------- ------------ ----------------
      4163        1780         4,63             4,39
      4168        1785         3,09             2,11
      4172        1789         6,19                6
      4222        1794         7,44             7,14
etc...
765 rows selected.
Elapsed: 00:00:01.71
```

Un grand nombre d'enregistrements violent la contrainte. La raison est que les surfaces des parcelles sont des données saisies par les utilisateurs (la surface déclarée de la parcelle), alors que les zones d'aptitude sont des données calculées à partir de leur saisie graphique à l'écran. □

Contrainte 3.

La date d'arrêt d'un produit d'épandage est la date à laquelle il cesse d'être produit. Les produits d'épandage ne sont pas stockés, donc un produit ne peut être épandu sur une parcelle qu'à une date antérieure à sa date d'arrêt. Pour être épandu sur une parcelle, un produit ne peut pas être arrêté, i.e. sa production doit continuer. Cela se traduit par une contrainte imposant que la date de création d'une parcelle soit toujours antérieure ou égale à la date d'arrêt de production du produit épandu.

```
context PRODUIT inv:
self.épandu_sur -> forAll ( z |
   z.est_dans.date_creation <= self.date_arret )
```

Soit en SQL généré par OCL2SQL :

```
select * from PRODUIT SELF
where not (not exists (select ID_ZAP from ZONE_APTITUDE
                       where ID_PRODUIT in
                             (select ID_PRODUIT from PRODUIT
                              where ID_PRODUIT = SELF.ID_PRODUIT))
                       minus
                       select ID_ZAP from ZONE_APTITUDE Z
                       where (select DATE_CREATION from PARCELLE
                              where ID_PARCELLE in
                                (select ID_PARCELLE from ZONE_APTITUDE
                                 where ID_ZAP = Z.ID_ZAP))
                             <= SELF.DATE_ARRET));
```

Les données erronées peuvent provenir d'une erreur dans la base, par exemple une date d'arrêt non ou mal-renseignée. □

Comme précisé auparavant, les données alphanumériques de la base de données SIGEMO sont sous Sybase, alors que ses données géographiques sont sous MapInfo. Un lien doit donc nécessairement être fait entre ces deux bases Sybase et MapInfo. Une contrainte OCL servira à tester la correspondance entre les parcelles alphanumériques et géographiques. Sous MapInfo, à chaque département *XX*, on fait correspondre une table *ParcelleXX* contenant les parcelles qui dépendent de ce département. Pour simplifier le processus de vérification d'intégrité et l'écriture des contraintes, nous avons intégré toutes ces tables dans une seule table "*Parcelle_Geo*" sous Oracle.

Contrainte 4.

On testera le fait qu'à une parcelle dessinée à l'écran, c'est-à-dire stockée sous MapInfo, correspond toujours une parcelle dans la base de données Sybase. Ceci correspond à la contrainte OCL suivante :

```
context PARCELLE_GEO inv:
PARCELLE.allInstances -> select(pBD |
       self.id_parcelle = pBD.id_parcelle) -> size() = 1
```

□

Contrainte 5.

Et réciproquement, une parcelle présente dans la base Sybase devra l'être également dans MapInfo.

```
context PARCELLE inv:
PARCELLE_GEO.allInstances -> select(pGeo |
       self.id_parcelle = pGeo.id_parcelle) -> size() = 1
```

□

6.2.2 Contraintes spatiales sur des régions simples

Dans la suite de ce chapitre, nous présentons les contraintes géographiques spécifiées sur SIGEMO en collaboration avec les experts métier. Ces contraintes sont exprimées à l'aide de l'extension d'OCL que nous avons proposée dans la Section 3.2.1, OCL_{9IM}. Les requêtes générées par OCL2SQL Spatial ont été utilisées pour tester la base de manière sporadique.

Contrainte 6.

Deux parcelles distinctes n'ont aucune partie commune, excepté éventuellement leurs frontières. En termes topologiques, ceci s'écrit "les parcelles sont disjointes ou adjacentes deux à deux". En OCL_{9IM}, on obtient :

```
context PARCELLE_GEO inv:
PARCELLE_GEO.allInstances -> forAll( self2 |
self <> self2 implies ((self.geoloc).areDisjoint(self2.geoloc)
                    or (self.geoloc).areAdjacent(self2.geoloc)))
```

D'où la requête SQL :

```
select ID_PARCELLE from PARCELLE_GEO SELF
where exists (select ID_PARCELLE from PARCELLE_GEO
              minus
              select ID_PARCELLE from PARCELLE_GEO SELF2
              where SELF.ID_PARCELLE = SELF2.ID_PARCELLE
              OR MDSYS.SDO_RELATE(SELF2.GEOLOC, SELF.GEOLOC,
                          'mask=DISJOINT querytype=WINDOW')= 'TRUE'
              OR MDSYS.SDO_RELATE(SELF2.GEOLOC, SELF.GEOLOC,
                          'mask=TOUCH querytype=WINDOW')= 'TRUE'));
```

En pratique, une grande partie des incohérences trouvées est générée par des imprécisions de saisie. Les parcelles sont dessinées à l'écran par une interface Web. La saisie à une échelle inadaptée peut entraîner une imprécision sur les frontières dessinées des parcelles. Ceci explique qu'un certain nombre de données ne respecte pas la contrainte dans la table *Parcelle*. On pourrait donc imaginer ajouter à cette contrainte un seuil de tolérance dans la précision. Ainsi on réduirait le nombre de tuples retournés et donc à vérifier, en autorisant une marge d'erreur. □

Contrainte 7.

Toute parcelle appartient à une commune et doit donc se situer, au moins en partie, sur le territoire de cette commune. Autrement dit, une parcelle est située dans la commune qui lui est associée ou l'intersecte au moins en partie.

```
context PARCELLE_GEO inv:
self.correspond_à.est_sur_commune -> exists ( com |
    (com.a_pour_géométrie.geoloc).contains(self.geoloc))
or self.correspond_à.est_sur_commune -> exists ( com |
    (com.a_pour_géométrie.geoloc).covers(self.geoloc))
or self.correspond_à.est_sur_commune -> exists ( com |
    (com.a_pour_géométrie.geoloc).overlaps(self.geoloc))
```

Dans cette contrainte, on accède aux tuples de la table *Commune* concernés grâce à la navigation via des associations. □

Contrainte 8.

Dans un périmètre d'épandage, au plus une parcelle contient un ouvrage de dépollution (*STEP*). On obtient en OCL_{9IM} :

```
context PERIMETRE inv:
self.se_divise_en -> select ( parc |
   STEP_GEO.allInstances -> exists ( st |
    (parc.a_pour_géométrie.geoloc).contains(st.geoloc))) -> size <= 1)
```

Là encore, la capacité de navigation d'OCL est utilisée afin de sélectionner les parcelles appartenant à un périmètre précis. □

Contrainte 9.

D'autre part, un seul ouvrage de dépollution est contenu dans une parcelle, et par extension, vue la contrainte précédente, dans un périmètre d'épandage.

```
context PARCELLE_GEO inv:
STEP_GEO.allInstances -> select(st |
       (self.geoloc).contains(st.geoloc) or
       (self.geoloc).covers(st.geoloc)) -> size <= 1
```

Il est à noter que cette contrainte pourrait être spécifiée à l'aide de l'un des formalismes présentés en Section 2.1. En effet, aucune des capacités particulières d'OCL n'entre en jeu dans cette contrainte. □

Contrainte 10.

Les zones ZNIEFF (Zones Naturelles d'Intérêt Ecologique Faunistique et Floristique), définissent des périmètres d'intérêt écologique. On distingue deux types de ZNIEFF identifiés par un code (*code_znieff*). Par exemple, aucun épandage ne devrait être proposé dans les zones ZNIEFF de type 1. Autrement dit, les parcelles du périmètre d'épandage et les zones ZNIEFF de code 1 ne s'intersectent pas. En OCL_{9IM}, cela s'écrit :

```
context PARCELLES_GEO inv:
ZNIEFF.allInstances -> select(code_znieff = 1) -> forAll(z |
                                    (self.geoloc).areDisjoint(z.geoloc)
                                 or (self.geoloc).areAdjacent(z.geoloc))
```

Cette contrainte ne s'applique qu'à une partie des tuples de la table *ZNIEFF*, ceux dont le code est 1. Ces tuples sont retournés par l'opération *select* d'OCL. Cette contrainte dépend d'une condition sur un attribut. Elle ne peut pas être exprimée simplement à l'aide des formalismes présentés en Section 2.1. □

Contrainte 11.

Une zone d'aptitude est toujours à l'intérieur de la parcelle correspondante.

```
context ZAP_GEO inv:
self.geoloc.isInside(self.correspond_à.est_dans.a_pour_géométrie.geoloc)
or
self.geoloc.coveredBy(self.correspond_à.est_dans.a_pour_géométrie.geoloc)
```

□

Contrainte 12.

Deux zones d'aptitude distinctes sont toujours disjointes ou adjacentes.

```
context ZAP_GEO inv:
ZAP_GEO.allInstances->forAll(self2 |
  (not(self.id_zap = self2.id_zap)) implies
  (self.geoloc.areDisjoint(self2.geoloc)
  or self.geoloc.areAdjacent(self2.geoloc)))
```

Ce qui donne en SQL :

```
select * from ZAP_GEO SELF
where exists (select ID_ZAP from ZAP_GEO
              minus
              select ID_ZAP from ZAP_GEO SELF2
              where SELF.ID_ZAP = SELF2.ID_ZAP
              OR MDSYS.SDO_RELATE(SELF2.GEOLOC, SELF.GEOLOC,
                    'mask=DISJOINT querytype=WINDOW')= 'TRUE'
              OR MDSYS.SDO_RELATE(SELF2.GEOLOC, SELF.GEOLOC,
                    'mask=TOUCH querytype=WINDOW')= 'TRUE');
```

□

Contrainte 13.

Un périmètre d'épandage est contenu dans le département qui lui est associé dans la base ou dans un des départements limitrophes de celui-ci.

```
context PERIMETRE inv:
DPT_GEO.allInstances -> select (dept |
  (dept.numero_dpt = self.est_rattaché_à.numero_dpt) or
  (dept.geoloc).areAdjacent(self.est_rattaché_à.a_pour_géométrie.geoloc))
 -> exists (d |
      self.se_divise_en -> forAll (parc |
         (parc.a_pour_géométrie.geoloc).isInside(d.geoloc)
         or (parc.a_pour_géométrie.geoloc).coveredBy(d.geoloc)
         or (parc.a_pour_géométrie.geoloc).overlap(d.geoloc))))
```

Cette contrainte est traduite par OCL2SQL :

```
select * from PERIMETRE SELF
where not (exists (
```

```
(select NUMERO_DPT from DPT_GEO
 minus
 select NUMERO_DPT from DPT_GEO DEPT
 where not
   (DEPT.NUMERO_DPT = (select NUMERO_DPT from DEPARTEMENT
                       where ID_PER in (select ID_PER from PERIMETRE
                                        where ID_PER = SELF.ID_PER)
 OR MDSYS.SDO_RELATE((select GEOLOC from DPT_GEO where NUMERO_DPT in
     (select NUMERO_DPT from DEPARTEMENT where NUMERO_DPT in
       (select NUMERO_DPT from PERIMETRE
        where ID_PER = SELF.ID_PER))),
   (DEPT.GEOLOC),
   'mask=TOUCH querytype=WINDOW')= 'TRUE')
) intersect
select NUMERO_DPT from DPT_GEO D
where not exists
( (select ID_PARCELLE from PARCELLE where ID_PER in
    (select ID_PER from PERIMETRE where ID_PER = SELF.ID_PER))
   minus
   select ID_PARCELLE from PARCELLE PARC
   where MDSYS.SDO_RELATE(D.GEOLOC,
              (select GEOLOC from PARCELLE_GEO where ID_PARCELLE in
                 (select ID_PARCELLE from PARCELLE
                  where ID_PARCELLE = PARC.ID_PARCELLE)),
             'mask=CONTAINS querytype=WINDOW')= 'TRUE'
   OR MDSYS.SDO_RELATE(D.GEOLOC,
              (select GEOLOC from PARCELLE_GEO where ID_PARCELLE in
                 (select ID_PARCELLE from PARCELLE
                  where ID_PARCELLE = PARC.ID_PARCELLE)),
             'mask=COVERS querytype=WINDOW')= 'TRUE'
   OR MDSYS.SDO_RELATE(D.GEOLOC,
              (select GEOLOC from PARCELLE_GEO where ID_PARCELLE in
                 (select ID_PARCELLE from PARCELLE
                  where ID_PARCELLE = PARC.ID_PARCELLE)),
             'mask=OVERLAPBDYINTERSECT querytype=WINDOW')= 'TRUE'
))));
```

On remarque que le fait de dupliquer les tables, en créant une table pour les données alphanumériques issues de Sybase, et une table pour les données géographiques issues de MapInfo, alourdit le code généré automatiquement avec des navigations superflues. □

6.2.3 Contraintes spatiales sur des régions composites

Certains recoupements des données de SIGEMO avec des données géographiques de type région composite ont aussi été considérés. Des contraintes portant sur ces objets ont été définies comme présenté ci-après.

Contrainte 14.
Les lacs sont stockés dans la classe LAC. Leur géométrie est une région composite, de type *Set(Region)* ; les composants correspondent aux différents plans d'eau d'un même lac.
Les parcelles d'épandage ne peuvent pas intersecter un plan d'eau d'un lac.

```
context PARCELLE_GEO inv:
LAC.allInstances -> select( lac |
    lac.geoloc -> forAll ( lac_p |
        (self.geoloc).areDisjoint(lac_p)))
```

Ce qui correspond à la requête SQL suivante :

```
select * from PARCELLE_GEO SELF
 where exists (
    select ID_LAC from LAC
   minus
    select ID_LAC from LAC LAC_INSTANCE
    where not exists (
       (select PART_ID from LAC_PART
        where LAC_PART.ID_LAC = LAC_INSTANCE.LAC_ID)
       minus
        select PART_ID from LAC_PART LAC_P
        where MDSYS.SDO_RELATE(LAC_P.GEO_PART, SELF.GEOMETRY,
                        'mask=DISJOINT querytype=WINDOW')= 'TRUE'));
```

□

Contrainte 15.
La classe *SITE_INDUSTRIEL* stocke les bâtiments des sites industriels. Chaque site industriel est représenté par une région composite. Généralement, au moins un bâtiment du site industriel devrait effectivement être spatialement localisé dans sa commune.

```
context site_industriel_geo inv:
self.geoloc -> exists( batiment |
```

```
     batiment.inside
          (self.correspond_à.se_situe_à.a_pour_géométrie.geoloc))
```

ou bien en OCL_{ADV} :

```
context site_industriel_geo inv:
self.geoloc -> inside ("occasionally",
     (self.correspond_à.se_situe_à.a_pour_géométrie.geoloc))
```

Ici `self.geoloc` retourne la collection de tous les bâtiments de l'instance courante de la classe `SITE_INDUSTRIEL`. La requête générée par OCL2SQL est la suivante :

```
select * from SITE_INDUSTRIEL_GEO SELF
where not (exists (
  (select ID_PART from SITE_INDUS_PART
   where SITE_INDUS_PART.ID_SITE = SELF.ID_SITE)
   intersect
   select ID_PART from SITE_INDUS_PART BATIMENT
   where MDSYS.SDO_RELATE(
     (select GEOLOC from COMMUNE_GEO
      where ID_COMMUNE in
        (select ID_COMMUNE from COMMUNE
         where ID_COMMUNE in
           (select ID_COMMUNE from SITE_INDUSTRIEL
            where ID_SITE in
              (select ID_SITE from SITE_INDUSTRIEL_GEO
               where ID_SITE = SELF.ID_SITE)))),
     BATIMENT.GEO_PART,
     'mask=CONTAINS querytype=WINDOW')= 'TRUE'
));
```

Afin de tester les fonctionnalités de l'extension spatiale d'OCL2SQL, on peut aussi expérimenter une restriction de la contrainte précédente. Faisons l'hypothèse (forte) qu'au moins la moitié des bâtiments d'un site industriel devrait effectivement être spatialement localisée dans sa commune. On peut alors écrire la contrainte suivante.

```
context site_industriel_geo inv:
self.geoloc -> select( batiment |
     batiment.inside
```

```
            (self.correspond_à.se_situe_à.a_pour_géométrie.geoloc))
        -> size >= (self.geoloc -> size) / 2
```

L'opération `size` appliquée à la collection des composants d'un site industriel, c'est-à-dire l'expression `self.geoloc -> size`, retourne le nombre de composants de la géométrie composite du site industriel.

L'exécution de OCL2SQL sur cette contrainte fournit la requête :

```
select * from SITE_INDUSTRIEL_GEO SELF
where not ((select NVL(COUNT(*),0) from SITE_INDUS_PART
            where ID_PART in
              (select ID_PART from SITE_INDUS_PART
               where SITE_INDUS_PART.ID_SITE = SELF.ID_SITE)
               minus
               select ID_PART from SITE_INDUS_PART BATIMENT
               where not (MDSYS.SDO_RELATE(
                 (select GEOLOC from COMMUNE_GEO
                  where ID_COMMUNE in
                    (select ID_COMMUNE from COMMUNE
                     where ID_COMMUNE in
                       (select ID_COMMUNE from SITE_INDUSTRIEL
                        where ID_SITE in
                          (select ID_SITE from SITE_INDUSTRIEL_GEO
                           where ID_SITE = SELF.ID_SITE)))),
                 BATIMENT.GEO_PART,
                 'mask=CONTAINS querytype=WINDOW')= 'TRUE'
))) >= ((select NVL(COUNT(*),0) from SITE_INDUS_PART
         where GEO_PART in
           (select GEO_PART from SITE_INDUS_PART
            where SITE_INDUS_PART.ID_SITE = SELF_GEO.ID_SITE))) / 2);
```

□

D'une manière globale, les temps d'exécution des requêtes vérifiant les contraintes topologiques sur la base SIGEMO peuvent être parfois de l'ordre de plusieurs minutes pour certaines contraintes, sur environ mille parcelles.

6.3 Discussion

Dans cette partie, nous avons présenté la validation de nos propositions de langages sur le projet SIGEMO qui est un système gérant les plans d'épandage au niveau national.

Le code SQL généré par OCL2SQL ne teste pas seulement les données en rapport avec celle qui vient d'être insérée, supprimée ou modifiée dans la base, mais l'ensemble des instances d'une ou plusieurs tables. Ainsi, ce type de tests est plus approprié pour une utilisation sporadique, en exécutant périodiquement une requête, que pour une utilisation sous forme de déclencheurs.

L'expérimentation sur le système SIGEMO a fait ressortir la nécessité d'offrir un moyen plus concis pour spécifier dans les contraintes que plusieurs types de relations sont possibles entre deux objets géographiques (voir Contrainte 13). En effet, lorsque plusieurs relations topologiques sont possibles entre deux objets, chaque relation est spécifiée distinctement dans la contrainte OCL_{9IM} et les objets sur lesquels elles s'appliquent sont répétés pour chacune d'elles. Il serait intéressant de pouvoir spécifier directement toutes les relations qui peuvent exister entre deux objets, comme c'est le cas dans les requêtes SQL pour Oracle à l'aide de l'opérateur "+" entre les différentes relations possibles. Par exemple, soit deux régions simples *A* et *B*, l'expression *A.{inside+coveredBy}(B)* sera équivalente à la contrainte OCL_{9IM} *A.inside(B) or A.coveredBy(B)*. Ainsi, la contrainte 13 pourrait être réécrite d'une façon plus directe :

```
context PERIMETRE inv:
DPT_GEO.allInstances -> select (dept |
  (dept.numero_dpt = self.est_rattaché_à.numero_dpt) or
  (dept.geoloc).areAdjacent(self.est_rattaché_à.a_pour_géométrie.geoloc))
 -> exists (d | self.se_divise_en -> forAll (parc |
    (parc.a_pour_géométrie.geoloc).{isInside+coveredBy+overlap}(d.geoloc)
```

Ce principe pourrait alors être pris en compte dans la génération de code pour Oracle Spatial. Après modification, la requête générée pour la contrainte 13 deviendrait :

```
select * from PERIMETRE SELF
where not (exists (
  (select NUMERO_DPT from DPT_GEO
   minus
   select NUMERO_DPT from DPT_GEO DEPT
   where not
     (DEPT.NUMERO_DPT = (select NUMERO_DPT from DEPARTEMENT
                         where ID_PER in (select ID_PER from PERIMETRE
```

```
                                        where ID_PER = self.ID_PER)
 OR MDSYS.SDO_RELATE((select GEOLOC from DPT_GEO where NUMERO_DPT in
     (select NUMERO_DPT from DEPARTEMENT where NUMERO_DPT in
       (select NUMERO_DPT from PERIMETRE
        where ID_PER = SELF.ID_PER))),
   (DEPT.GEOLOC),
   'mask=TOUCH querytype=WINDOW')= 'TRUE')
) intersect
select NUMERO_DPT from DPT_GEO D
where not exists
( (select ID_PARCELLE from PARCELLE where ID_PER in
    (select ID_PER from PERIMETRE where ID_PER = SELF.ID_PER))
   minus
   select ID_PARCELLE from PARCELLE PARC
   where MDSYS.SDO_RELATE(D.GEOLOC,
              (select GEOLOC from PARCELLE_GEO where ID_PARCELLE in
                 (select ID_PARCELLE from PARCELLE
                  where ID_PARCELLE = PARC.ID_PARCELLE )),
                     'mask=CONTAINS+COVERS+OVERLAPBDYINTERSECT
                     querytype=WINDOW')= 'TRUE'
)));
```

Chapitre 7
Conclusion et perspectives

7.1 Conclusion

Les objectifs de cette thèse étaient, d'une part, de proposer des méthodes expressives pour modéliser les contraintes d'intégrité des bases de données spatiales, et d'autre part, de mettre en place un outil pour vérifier ces contraintes. Le but principal de ces travaux était d'assurer la qualité des informations spatiales dans les bases de données géographiques. Nous avons donc proposé des langages de contraintes adaptés à la manipulation d'informations géographiques et possédant un pouvoir d'expression élevé.

Nous avons étendu le générateur de code OCL2SQL afin de rendre possible la production des scripts SQL correspondants à des contraintes spatiales spécifiées à l'aide des langages proposés. Cet outil a permis de valider nos propositions sur le système SIGEMO : un système de gestion national des projets d'épandage. Des contraintes d'intégrité alphanumériques et spatiales, c'est-à-dire des contraintes que les données doivent respecter pour que le système reste cohérent, ont été spécifiées dans les langages proposés en collaboration avec des experts métier. Après génération automatique du code SQL correspondant, ces contraintes ont été testées sur une copie de la base, rapatriée sous Oracle pour simplifier cette expérimentation. De plus, OCL2SQL génère du code SQL pour Oracle. La génération pour d'autres plates-formes a été initiée lors de l'encadrement de projets universitaires au cours de cette thèse, mais l'essentiel du travail reste à faire.

7.1.1 Langages de contraintes proposés pour les bases de données spatiales

Dans ce mémoire, nous avons proposé différentes extensions spatiales d'OCL pour spécifier précisément les contraintes topologiques dans les bases de données spatiales. La définition de types spatiaux dans l'extension OCL Spatial (Section 3.1) permet d'écrire des contraintes sur les types des objets géographiques. OCL Spatial autorise la spécification des contraintes sur les types exprimables avec les formalismes de modélisation des Systèmes d'Information Géographique et même plus encore (Section 4.1). En effet, avec OCL Spatial, on peut aisément exprimer des types géographiques qui dépendent de la valeur d'un attribut. Par exemple, si le type d'attribut géométrique d'une classe *Ville* dépend de la population de cette ville stockée dans un attribut *nb_habitants*, une instance de *Ville* sera représentée par un point lorsque son *nb_habitant* sera inférieur à 500000, par une région sinon.

Les deux modèles topologiques les plus répandus, 9IM (Section 2.1.1) et CBM (Section 2.1.2), sont intégrés dans les langages proposés, OCL_{9IM} (Section 3.2.1) et OCL_{CBM} (Section 3.2.2). Ces langages offrent un bon niveau d'abstraction pour spécifier les contraintes d'intégrité spatiales. Il ressort qu'il est plus simple et "intuitif" d'écrire les contraintes en OCL_{9IM} ou OCL_{CBM} que directement en SQL. De plus, ces langages offrent des moyens déclaratifs et expressifs pour modéliser les contraintes spatiales.

Nous avons montré que ces deux langages sont adaptés à la spécification de contraintes sur des géométries composites, en les couplant à l'utilisation des opérations sur les collections d'OCL. Cependant, pour simplifier l'écriture de ces contraintes, nous avons proposé plusieurs variantes : d'une part OCL_{ADV} basé sur 9IM (Section 3.3.2), et d'autre part OCL_{TRCR} et P-OCL basés sur CBM (Section 3.3.3). Ces trois langages ont le même pouvoir d'expression que l'extension spatiale dont ils découlent : OCL_{9IM} pour OCL_{ADV} (Section 4.5), et OCL_{CBM} pour OCL_{TRCR} (Section 4.2) et P-OCL (Section 4.3). Nous avons également comparé ces trois langages avec des modèles particulièrement expressifs pour décrire des relations entre deux régions composites (TRCR niveau détaillé et 9IM pour régions composites). OCL_{9IM} permet de spécifier chacune des relations exprimables avec les matrices 9IM appliquées aux régions composites (Section 4.6). OCL_{TRCR} permet d'exprimer les relations décrites par les matrices du niveau détaillé de TRCR et même plus encore (Section 4.4).

7.1.2 Implémentation et expérimentation sur SIGEMO

Dans la Section 5.1, nous présentons notre extension d'un générateur de code qui, à partir de contraintes spécifiées en OCL_{9IM} ou OCL_{ADV}, retourne le code SQL qui permet de vérifier la contrainte sur une base de données.

Nous avons expérimenté les différents langages proposés dans le cadre d'un système permettant la gestion des projets d'épandages : SIGEMO décrit en Section 5.2. Les contraintes définies sur ce système en collaboration avec les experts métiers ont permis de valider nos propositions de langages. La majorité de ces contraintes exprimées en OCL_{9IM} sont recensées dans la Section 5.3 avec quelques exemples de génération de code automatique produit par l'outil OCL2SQL étendu au spatial.

7.2 Perspectives

7.2.1 Langages de contraintes spatiales

Les travaux présentés dans cette thèse portent essentiellement sur les régions composites, ce qui couvre déjà un large spectre d'objets spatiaux. Il serait intéressant d'élargir le champ des recherches à d'autres types d'objets spatiaux : relations entre objets de types différents [MAR94], régions complexes [EGE94] [NGU97] [SCH01], polylignes [EGE94b], ensembles partiellement ordonnés [KAI93], collections d'objets de types différents [ZHO04], etc.

Seules les relations topologiques ont été étudiées dans nos travaux. D'autres relations spatiales, comme par exemple, les relations métriques mériteraient d'être considérées. Dans [NED04] [NED06], les auteurs étendent le modèle des 9-Intersections appliqué à deux polylignes. Pour cela, ils intégrent les valeurs normalisées des longueurs ou surfaces des intersections, et les distances entre deux lignes.

D'autre part, il serait intéressant de travailler sur un langage d'expression de contraintes de plus haut niveau que ceux proposés dans cette thèse. Ce langage ne serait pas nécessairement aussi expressif que nos propositions mais il pourrait être pictogrammique ou visuel. Les propositions de langages pictogrammiques pour l'interrogation des bases de données spatiales pourraient servir de base à un tel langage [LOR96] [POU00]. Un langage visuel de contraintes pourrait, par exemple, s'inspirer de ceux dédiés à l'interrogation et à la manipulation de bases de données spatio-temporelles [BON99] [BON00] [BLA00]. Ce type de langage pourrait être efficacement installé au-dessus de nos propositions. Une intégration de nos langages dans des outils graphiques pour la spécification de contraintes spatiales (comme ceux présentés dans la Section 2.4) offre aussi un autre champ d'investigation.

Pour la spécification de types géographiques complexes, la perspective d'un couplage des langages proposés avec l'utilisation des pictogrammes des formalismes présentés en Section 2.3 pourrait être utilement explorée l'expression des types spatiaux standards se faisant avec des pictogrammes les autres étant spécifiés sous forme textuelle détaillée avec OCL Spatial.

Enfin, la notion de temporalité des objets permettrait notamment de mieux représenter la traçabilité et de gérer des objets mobiles. Des recherches ont déjà été faites pour la spécification de contraintes spatio-temporelles [CLA00b] [CLA01] [ERW02].

7.2.2 Implémentation

Les langages proposés ont été implémentés dans l'outil OCL2SQL. Afin de fournir une solution complète intégrée dans un AGL, notre proposition d'outil pourra par la suite être intégrée à l'extension spatiale de l'AGL Objecteering développée au Cemagref [MIR06]. Cette extension intègre une méthode spécifique pour les SIG basée sur UML et la MDA. La spécification des contraintes topologiques est d'ailleurs déjà possible directement dans cet outil, par le biais de relations entre classes (voir Figure 45). Les contraintes OCL_{9IM} pourront donc offrir un moyen d'expression parfaitement complémentaire. Le lien avec l'AGL sera facilité par le support du format XMI.

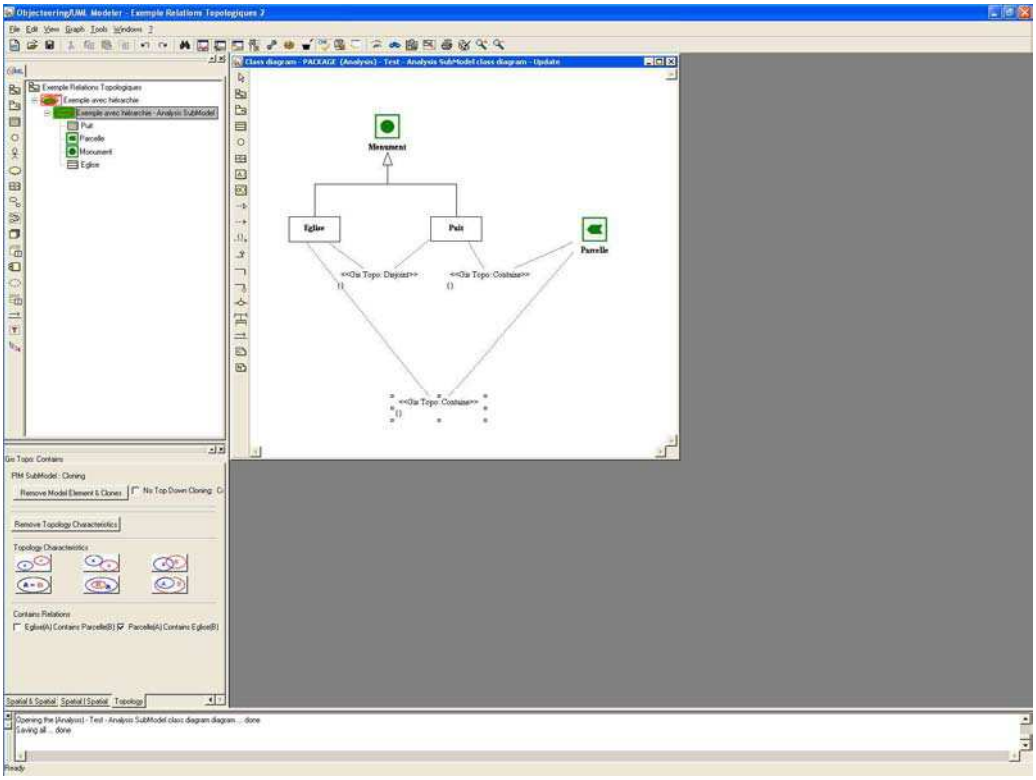


Figure 45. Extension spatiale d'Objecteering développée au Cemagref

En ce qui concerne l'implémentation de notre extension spatiale proprement dite, il nous semble être intéressant d'étendre l'outil OCL2SQL à d'autres plates-formes qu'Oracle Spatial, notamment postGIS.

Bibliographie

[ABI00] S. Abiteboul, R. Hull et V. Vianu, **Fondements des bases de données**, Vuibert, 2000, 715 p.

[ABR96] J.-R. Abrial, **The B-Book, Assigning Programs to Meanings,** Cambridge University Press, 1996, 813 p.

[ADE04] ADEME, **L'Epandage Agricole**, Document d'information ADEME, accessible depuis <http://www.ademe.fr/partenaires/Boues/>, 2004.

[AKE01] D. H. Akehurst et B. Bordbar, **On Querying UML Data Models with OCL**, Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, Springer-Verlag, 2001, pp. 91-103.

[BAU95] C. Bauzer Medeiros et M. Cilia, **Maintenance of Binary Topological Constraints through Active Databases,** ACM GIS'95**,** pp. 127-133.

[BED99] Y. Bédard, **Visual Modelling of Spatial Databases Towards Spatial PVL and UML,** Geomatica, Vol. 53 (2), 1999, pp. 169-186.

[BED04] Y. Bédard, S. Larrivée, M.-J. Proulx et M. Nadeau, **Modeling Geospatial Databases with Plug-Ins for Visual Languages: A Pragmatic Approach and the Impacts of 16 Years of Research and Experimentations on Perceptory,** CoMoGIS'04, Chine, 2004, pp. 5-16.

[BEH01] T. Behr et M. Schneider, **Topological Relationships of Complex Points and Complex Regions,** 20th International Conference on Conceptual Modeling (ER 2001), Yokohama, Japan, 2001, pp. 56-69.

[BEL00] A. Belussi, M. Negri et G. Pelagatti, **An Integrity Constraints Driven System for Updating Spatial Databases,** ACM GIS, Washington D.C., USA, 2000, pp. 121-128.

[BLA00] A. D. Blaser et M. J. Egenhofer, **A Visual Tool for Querying Geographic Databases**, AVI 2000 (Advanced Visual Interfaces), Palermo, Italy, 2000, 8 p.

[BON99] C. Bonhomme, C. Trepied, M.-A. Aufaure et R. Laurini, **A Visual Language for Querying Spatiotemporal Databases**, 7th ACM Symposium on Advances in Geographic Information Systems, Kansas City, USA, 5-6 novembre 1999, 8 p.

[BON00] C. Bonhomme, **Un Langage Visuel Dédié à l'Interrogation et à la Manipulation de Bases de Données Spatio-Temporelles**, Thèse, INSA de Lyon, 22 décembre 2000, 328 p.

[BOR99] K. A. V. Borges, A. H. F. Laender et C. A. Davis Jr., **Spatial Data Integrity Constraints in Object Oriented Geographic Data Modeling**, Proceedings of the International Symposium on Geographic Information System, ACM Press, USA, 2-6 novembre 1999, pp. 1-6.

[BOR01] K. A. V. Borges, C. A. Davis Jr. et A. H. F. Laender, **OMT-G : An Objected-Oriented Data Model for Geographic Applications**, Geoinformatica, Vol. 5 (3), septembre 2001, pp. 221-260.

[BOR01b] O. Borraz, **Communication et Organisation Territoriale. Construire la Confiance Autour des Epandages de Boues Urbaines**, Actes du colloque national Logistique des Epandages des Effluents d'Elevage des Boues de Stations d'Epuration et des Déchets Industriels, Vichy, 8-9 octobre 2001, pp. 165-169.

[BOU02] S. Bourgeois, A. Ker, E. Mathieu et J. Wiart, **La grille d'évaluation GEVAL : Une Méthode pour Evaluer le Niveau de Qualité des Opérations d'Epandage des Boues Municipales**, Ingénieries – Eau, Agriculture, Territoires, n° 31, 2002, pp. 27-35.

[CLA97] C. Claramunt, S. Coulondre et T. Libourel, **Autour des Méthodes Orientées Objet pour la Conception des SIG**, Revue Internationale de Géomatique, Vol. 7 (3-4), Ed. Hermes, Paris, France, 1997, pp. 233-257.

[CLA00] C. Claramunt, **Extending Ladkin's Algebra on Non-Convex Intervals Towards an Algebra on Union-of Regions**, GeoInformatica, Proceedings. of the International ACM Symposium on Advances in Geographic Information Systems, Washington D.C., USA, 2000, pp. 9-14.

[CLA00b] C. Claramunt et B. Jiang, **Hierarchical Reasoning in Space and Time**, 9th International Symposium on Spatial Data Handling (SDH), Pékin, Chine, 10-12 août 2000, pp. 41-51.

[CLA01] C. Claramunt et B. Jiang, **An Integrated Representation of Spatial and Temporal Relationships between Evolving Regions**, Geographical Systems, Vol. 3, 2001, pp. 411-428.

[CLE93] E. Clementini , P. Di Felice et P. van Oosterom, **A Small Set of Formal Topological Relationships Suitable for End-User Interaction**, International Symposium on Advances in Spatial Databases (SSD'93), Singapour, 1993, pp. 277-295.

[CLE94] E. Clementini et P. Di Felice, **A Comparison of Methods for Representing Topological Relationships**, Information Sciences - Applications, Vol. 3 (3), 1995, pp. 149-178.

[CLE94b] E. Clementini et P. Di Felice, **A Model for Representing Topological Relationships between Complex Geometric Features in Spatial Databases**, Information Sciences, Elsevier, 1994, 17 p.

[CLE95] E. Clementini, P. Di Felice et G. Califano, **Composite Regions in Topological Queries**, Information Systems, Vol. 20 (7), 1995, pp. 579-594.

[COA91] P. Coad et E. Yourdon, **Object-Oriented Analysis**, Yourdon Press, 1991, 233 p.

[COC97] S. Cockcroft, **A Taxonomy of Spatial Data Integrity Constraints**, Geoinformatica, Vol. 1(4), 1997, pp. 327-343.

[COC98] S. Cockcroft, **User Defined Spatial Business Rules: Storage, Management and Implementation – A Pipe Network Case Study**, 10th Colloquium of the Spatial Information Research Centre, University of Otago, Dunedin, New-Zealand, 16-19 novembre 1998, pp. 73-81.

[COC01] S. Cockcroft, **Modelling Spatial Data Integrity Rules at the Metadata Level**, 6th International Conference on GeoComputation, Brisbane, Australia, 24-26 Septembre 2001, 6 p.

[COC04] S. Cockcroft, **The Design and Implementation of a Repository for the Management of Spatial Data Integrity Constraints**, Geoinformatica, Vol. 8 (1), 2004, pp. 49-69.

[COH97] A. G. Cohn, B. Bennett, J. Gooday et N. M. Gotts, **Qualitative Spatial Representation and Reasoning with the Region Connection Calculus**, Geoinformatica, Vol. 1 (3), 1997, pp. 273-376.

[COO94] S. Cook et J. Daniels, **Designing Object Systems-Object Oriented Modeling with Syntropy**, Prentice-Hall, New York, 1994.

[DEM99] B. Demuth et H. Hussmann, **Using UML / OCL Constraints for Relational Database Design**, Proceedings of the Conference on the Unified Modelling Language (UML'99), Fort Collins, USA, Lecture Notes in Computer Science, vol. 1723, 28-30 octobre 1999, pp. 598-613.

[DEM01] B. Demuth, H. Hussmann et S. Loecher, **OCL as a Specification Language for Business Rules in Database Applications**, Proceedings of the Conference on the Unified Modelling Language (UML 2001), Toronto, Canada, Lecture Notes in Computer Science, vol. 2185, octobre 2001, pp. 104-117.

[DEM04] B. Demuth, H. Hussmann, S. Loecher et S. Zschaler, **Structure of the Dresden OCL toolkit**. 2nd International Fujaba Days "MDA with UML and Rule-Based Object Manipulation", Darmstadt, Allemagne, 15-17 Septembre, 2004, <http://www.es.tu-darmstadt.de/english/events/fd04/submissions/06_Demuth.pdf>

[DEV04] R. Devillers, **Conception d'un Système Multidimensionnel d'information sur la Qualité des Données Géospatiales**, Thèse, Univ. Laval – Québec / Univ. Marne-la-Vallée, décembre 2004, 167 p.

[EGE92] M. J. Egenhofer et J. R. Herring, **Categorizing Binary Topological Relations between Regions, Lines, and Points in Geographic Databases**, Technical report. Department of Surveying Engineering, University of Maine, Orono, ME, 1992, 28p, <http://www.cs.umn.edu/Research/shashigroup/CS8715/MSD11_egenhofer_herring.pdf>.

[EGE93] M. J. Egenhofer, J. Sharma et D. M. Mark, **A Critical Comparison of the 4-Intersection and 9-Intersection Models for Spatial Relations: Formal Analysis**, Autocarto Vol. 11, Minneapolis, USA, octobre 1993, 12 p.

[EGE94] M. J. Egenhofer, **Definitions of Line-Line Relations for Geographic Databases**, IEEE Data Eng. Bull. 16(3), 1993, pp. 40-45.

[EGE94b] M. J. Egenhofer, E. Clementini et P. Di Felice, **Topological Relations between Regions with Holes**, International Journal of Geographical Information Science, Vol. 8 (2), 1994, pp. 129-144.

[EGE94c] M. J. Egenhofer et R. D. Franzosa, **On the Equivalence of Topological Relations**, International Journal of Geographical Information Science, Vol. 8 (6), 1994, pp. 133-152.

[ERW02] M. Erwig et M. Schneider, **Spatio-Temporal Predicates**, IEEE Transactions on Knowledge and Data Engineering, Vol. 14 (4), 2002, pp.881-901.

[FIN00] F. Finger, **Design and Implementation of a Modular OCL Compiler**, Thèse, Université de Dresden, Allemagne, 29 mars 2000, 115 p.

[FRI01] A. Friis-Christensen, N. Tryfona et C. S. Jensen, **Requirements and Research Issues in Geographic Data Modeling**, Proceedings of the 9th International ACM Symposium on Advances in Geographic Information Systems (ACM GIS 2001), Atlanta, USA, 2001, pp. 2-8.

[GAG98] E. Gagnon, **SableCC, an Object-Oriented Compiler Framework**, Thèse, Université McGill, Montréal, mars 1998, 97 p.

[GAL01] C. Gallian, **L'Appui des Agences de l'Eau dans la Mise en Place des Organismes Indépendants de Suivi des Epandages. Diversité des Dispositifs par Bassin**, Actes du colloque national Logistique des Epandages des Effluents d'Elevage des Boues de Stations d'Epuration et des Déchets Industriels, Vichy, 8-9 octobre 2001, pp. 29-35.

[GUT95] R. H. Güting et M. Schneider, **Realm-Based Spatial Data Types: The Rose Algebra**, 1995, 39 p.

[KAI93] W. Kainz, M. J. Egenhofer et I. Greasley, **Modeling Spatial Relations and Operations with Partially Ordered Sets**, International Journal of Geographical Information Science, Vol. 7 (3), 1993, pp. 215-229.

[KAN04] H.-K. Kang, T.-W. Kim et L. Ki-Joune, **Topological Consistency for Collapse Operation in Multi-Scale Databases**, LCNS, Vol. 3289, CoMoGIS, 2004.

[KLA05] Klasse Objecten, **OCL Tools and Services Web Site**, mars 2005, <http://www.klasse.nl/ocl>

[KOS97] G. Kösters, B.-U. Pagel et H.-W. Six, **GIS-Application Development with GeoOOA**, International Journal of Geographical Information Science, Vol. 11 (4),1997, pp. 307-335.

[LBA97] A. Lbath, **AIGLE : Un Environnement Visuel pour la Conception et la Génération Automatique d'Applications Géomatiques**, Thèse, INSA de Lyon, 4 novembre 1997, 282 p.

[LOR96] F. Di Loreto, F. Ferri, F. Massari et M. Rafanelli, **A Pictorial Query Language for Geographical Databases**, AVI'96, Gubbio, Italy, 1996.

[MAN99] L. Mandel et M. V. Cengarle, **On the Expressive Power of the OCL**, Proceedings of the World Congress on Formal Methods in the Development of Computing Systems, 1999, pp. 854-874.

[MAR01] R. Marcano et N. Lévy, **Transformation d'Annotations OCL en Expressions B**, AFADL'2001 Approches Formelles dans l'Assistance au Développement de Logiciels, Nancy, 11-13 juin 2001, pp. 39-49.

[MAR94] D. M. Mark et M. J. Egenhofer, **Modeling Spatial Relations between Lines and Regions: Combining Formal Mathematical Models and Human Subjects Testing**, Cartography and Geographic Information Systems, Vol. 21 (4), octobre 1994, pp. 195-212.

[MIR06] A. Miralles, **Ingénieriedes modèles pour les applications environnementales**, Thèse, Université de Montpellier II, 11 décembre 2006, 344 p.

[NED04] K. A. Nedas et M. J. Egenhofer, **Splitting Ratios - Metric Details of Topological Line-Line Relations**, 17th International FLAIRS Conference, Miami, USA, 2004, 6 p.

[NED06] K. A. Nedas, M. J. Egenhofer et D. Wilmsen, **Metric Details of Topological Line-Line Relations**, International Journal of Geographical Information Science Vol. 21(1), 2007, pp. 21-48.

[NGU97] V. H. Nguyen, C. Parent et S. Spaccapietra, **Complex Regions in Topological Queries**, 1997, COSIT'97, 18 p.

[OMG06] OMG, **Object Constraint Language Version 2.0**, OMG Specification, mai 2006, 232 p, <http://www.omg.org/docs/formal/06-05-01.pdf>.

[ORA05] Oracle Corp., **Oracle Spatial, User's Guide and Reference, Release 10.2**, Oracle Documentation, juin 2005, 592 p.

[PAN94] D. Pantazis, **Analyse Méthodologique des Phases de Conception et de Développement d'un SIG**, Thèse, Université de Liège, 1994, 556 p.

[PAN96] D. Pantazis, **La Conception des SIG**, Hermès, Paris, 1996, 343 p.

[PAR98] C. Parent, S. Spaccapietra, E. Zimanyi, P. Donini, C. Plazanet, C. Vangenot, N. Rognon et P.-A. Crausaz, **MADS, Modèle Conceptuel Spatio-Temporel**, Revue Internationale de Géomatique, Vol. 7 (3-4), 1998, pp. 317-352.

[PAR99] C. Parent, S. Spaccapietra et E. Zimanyi, **Spatio-Temporal Conceptual Models: Data Structures + Space + Time**, Proceedings of the International ACM Symposium on Advances in Geographic Information Systems (ACM GIS'99), USA, 2-6 novembre 1999, pp. 26-33.

[PAR99b] C. Parent, S. Spaccapietra, E. Zimanyi, P. Donini, C. Plazanet et C. Vangenot, **MADS Ou l'Information Spatio-Temporelle à Portée de ses Utilisateurs**, Quatrièmes rencontres de ThéoQuant, Besançon, France, 11-12 février 1999.

[PAR04] C. Parent, **MADS Formalization**, rapport technique EPFL, http://lbd.epfl.ch/e/research/mads/FormaWeb2.pdf, 2004.

[PAR06] C. Parent, S. Spaccapietra, E. Zimanyi, **Conceptual Modeling for Traditional and Spatio-temporal Applications: the MADS Approach**, Springer Verlag, 2006, 466 p.

[PIN02] F. Pinet, **Vers une Méthode de Conception pour la Télégéomatique : Intégration et Validation au Sein d'un Atelier de Développement**, Thèse, INSA de Lyon, 2002, 243 p.

[POU00] E. Pourabbas et M. Rafanelli, **A Pictorial Query Language for Querying Geographic Databases Using Positional and OLAP Operators**, ACM SIGMOD Record Vol. 31 (2), 2002, pp. 22-27.

[PRA01] I. Pratt-Hartmann, **A Topological Constraint Language with Component Counting**, Journal of Applied Non-Classical Logics, Vol. 11 (3-4), 2001, pp. 1-90.

[PUR00] A. Puricelli, **Réingénierie et Controle Qualité des Données en Vue d'une Migration Technologique**, Thèse, INSA de Lyon, 18 décembre 2000, 263 p.

[RAN92] D. A. Randell, Z. Cui et A. G. Cohn, **A Spatial Logic Based on Regions and Connection**, International Conference on Principles of Knowledge Representation and Reasoning (KR'92), USA, 1992, pp. 165-176.

[REN01] J.C. Renat, **Concevoir et Organiser des Opérations d'Epandage**, Actes du colloque national Logistique des Epandages des Effluents d'Elevage des Boues de Stations d'Epuration et des Déchets Industriels, Vichy, 8-9 octobre 2001, pp. 37-40.

[REN97] J. Renz et B. Nebel, **On the Complexity of Qualitative Spatial Reasoning: a Maximal Tractable Fragment of RCC-8**, QR'97, Cortona, Italy, pp.317-325.

[ROD04] M. A. Rodriguez, **Inconsistency Issues in Spatial Databases**, in Inconsistency Tolerence, LNCS, Vol. 3300, 2004, pp. 237-269.

[SCH01] M. Schneider, **A Design of Topological Predicates for Complex Crisp and Fuzzy Regions**, Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling, 2001, LNCS Vol. 2224, pp. 103-116.

[SCH02] B. Schmid, J. Warmer et T. Clark, **Object Modeling with the OCL: The Rationale Behind the Object Constraint Language**, Springer Verlag, 2002, 281p.

[SCH02b] M. Schneider, **Implementing Topological Predicates for Complex Regions**, Symposium on Geospatial Theory, Processing and Applications, Ottawa, 2002, 16 p.

[SCH06] M. Schneider et T. Behr**, Topological Relationships between Complex Spatial Objects**, ACM Transactions on Database Systems (TODS), Vol.31, 2006, pp. 39-81.

[SER00] S. Servigne, T. Ubeda, A. Puricelli et R. Laurini, **A Methodology for Spatial Consistency Improvement of Geographic Databases**, Geoinformatica, Vol. 4 (1), 2000, pp. 7-34.

[SOU03] V. Soulignac et F. Gibold, **SIGEMO : Détails des fonctionnalités**, Cemagref, Edition du 15/10/2003.

[SOU04] V. Soulignac, F. Gibold, F. Pinet et F. Vigier, **Spreading Matter Management in France within SIGEMO**, Proceedings of the 5th European conference For Information Technologies in Agriculture (EFITA 2005), Vila Real, Portugal, 25-28 Juillet 2005, 8 p.

[UBE97] T. Ubeda, **Contrôle de la Qualité Spatiale des Bases de Données Géographiques : Cohérence Topologique et Corrections d'Erreurs**, Thèse, INSA de Lyon, 1997, 205 p.

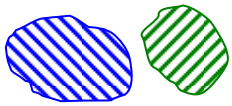
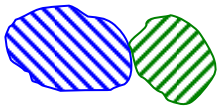
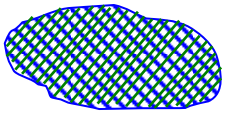
[VAN01] A. Van De Velde, **Démarche Qualité dans une Entreprise d'Epandage**, Actes du colloque national Logistique des Epandages des Effluents d'Elevage des Boues de Stations d'Epuration et des Déchets Industriels, Vichy, 8-9 octobre 2001, pp. 155-158.

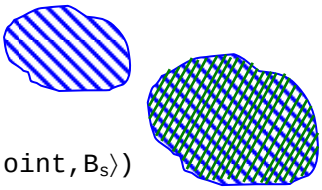
[WAR99] J. Warmer et A. Kleppe, **The Object Constraint Language Precise Modeling with UML**. Addison-Wesley, 1999, 112 p.

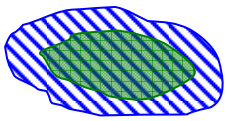
[ZHO04] Z. Zhong, N. Jing, L Chen et Q. Wu, **Representing Topological Relationships Among Heterogeneous Geometry-Collection Features**, Journal of Computer Science and Technology, Vol.19(3), 2004, pp. 280-289.

ANNEXE

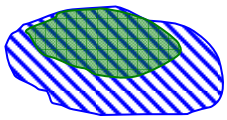
Dans cette annexe, pour chacune des 16 matrices 9IM traduisant les relations topologiques entre deux régions composites, nous donnons les expressions logiques et OCL_{9IM} équivalentes. Ces expressions ont été obtenues en appliquant les Théorèmes de la Section 4.6.

Matrice 9-IM	Expression logique et expression OCL_{9IM} équivalentes
0 0 1 0 0 1 1 1 1	*Matrice 1* ⇔ *disjoint* Les intérieurs et les frontières des 2 régions composites ne s'intersectent jamais. Chaque composant de *A* doit être *disjoint* de tous les composants de *B*. • $\forall i \in 1..n, \forall j \in 1..m \mid \langle A_i, \text{disjoint}, B_j \rangle$ • `A->forAll(Ai| B->forAll(Bj| Ai.disjoint(Bj)))` 
0 0 1 0 1 1 1 1 1	*Matrice 2* ⇔ *meet* Les frontières des 2 régions composites s'intersectent au moins une fois. Au moins un composant de *A* est adjacent à un composant de *B*. Tous les autres couples vérifient une relation d'adjacence ou de disjonction. • $\exists i \in 1..n, \exists j \in 1..m \mid \forall r \in 1..n, \forall s \in 1..m, \langle A_i, \text{meet}, B_j \rangle \wedge (\langle A_r, \text{disjoint}, B_s \rangle \vee \langle A_r, \text{meet}, B_s \rangle)$ • `A->exists(Ai| B->exists(Bj| Ai.meet(Bj))) and A->forAll(Ar| B->forAll(Bs| Ar.meet(Bs) or Ar.disjoint(Bs)))` 
1 0 0 0 1 0 0 0 1	*Matrice 3* ⇔ *equal* Les frontières ne s'intersectent qu'entre elles ; il en va de même pour les intérieurs et les extérieurs des 2 régions composites. Autrement dit, chaque composant d'une région est égal à un composant de la seconde région composite. • $\forall i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{equal}, B_j \rangle \wedge \forall j \in 1..m, \exists i \in 1..n \mid \langle B_j, \text{equal}, A_i \rangle$ • `A->forAll(Ai| B->exists(Bj| Ai.equal(Bj))) and B->forAll(Bj| A->exists(Ai| Bj.equal(Ai)))` 

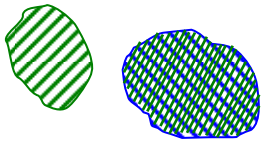
1 0 0
0 1 0
1 1 1
	Matrice 4
	L'extérieur de *B* intersectent seulement l'extérieur de *A*, donc *A* n'est jamais "hors" de la frontière de *B*. L'intérieur (resp. la frontière) de *A* intersecte l'intérieur (resp. la frontière) de *B*, ainsi, chaque composant de *A* est égal à un composant de *B*. L'extérieur de *A* intersecte l'intérieur, l'extérieur et la frontière de *B*, donc au moins un composant de *B* est disjoint de tous les composants de *A*.
	 
	• $(\forall i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{equal}, B_j \rangle) \wedge$
	$(\exists l \in 1..m, \forall k \in 1..n \mid \langle A_k, \text{disjoint}, B_l \rangle) \wedge$
	$(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, \text{equal}, B_s \rangle \vee \langle A_r, \text{disjoint}, B_s \rangle)$
	• `A->forAll(Ai| B->exists(Bj| Ai.equal(Bj))) and`
	`B->exists(Bl| A->forAll(Ak| Ak.disjoint(Bl))) and`
	`A->forAll(Ar| B->forAll(Bs| Ar.equal(Bs) or Ar.disjoint(Bs)))`

1 0 0 1 0 0 1 1 1	*Matrice 5* ⇔ *inside* L'intérieur et la frontière de *A* intersectent seulement l'intérieur de *B*. Autrement dit, chaque composant de *A* est à l'intérieur d'un composant de *B*.  • $\forall i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{inside}, B_j \rangle$ • `A->forAll(Ai| B->exists(Bj| Ai.inside(Bj)))`

<table>
<tr>
<td>1 0 0
1 1 0
1 1 1</td>
<td>
Matrice 6 ⇔ coveredBy
L'intérieur de A intersecte seulement l'intérieur de B et l'extérieur de B intersecte seulement l'extérieur de B comme dans la matrice 5, mais les frontières des deux régions s'intersectent au moins une fois. Cela correspond aux deux cas suivants :

(a) Il existe au moins une relation coveredBy entre un composant de A et un composant de B et chaque composant de A est en relation equal, inside ou coveredBy avec un composant de B.
$\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{coveredBy}, B_j \rangle \wedge$
$\forall r \in 1..n, \exists s \in 1..m, (\langle A_r, \text{equal}, B_s \rangle \vee$
$\langle A_r, \text{inside}, B_s \rangle \vee \langle A_r, \text{coveredBy}, B_s \rangle)$

(b) Il existe au moins un couple de composants des deux régions qui sont égaux, et un composant de A est à l'intérieur d'un composant de B. Les autres couples sont en relation equal ou inside.
$\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{inside}, B_j \rangle \wedge$
$\exists k \in 1..n, k \neq i, \exists l \in 1..m, l \neq j \mid \langle A_k, \text{equal}, B_l \rangle \wedge$
$\forall r \in 1..n, \exists s \in 1..m, (\langle A_r, \text{equal}, B_s \rangle \vee \langle A_r, \text{inside}, B_s \rangle)$

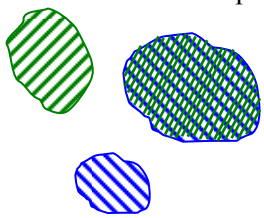
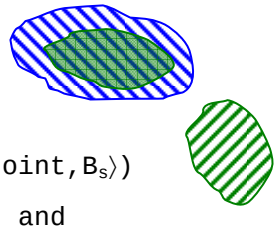
<pre>(A->exists(Ai| B->exists(Bj| Ai.coveredBy(Bj)))
 and A->forAll(Ar| B->exists(Bs| Ar.equal(Bs)
 or Ar.inside(Bs) or Ar.coveredBy(Bs))))
or
(A->exists(Ai| B->exists(Bj| Ai.inside(Bj)))
 and (A->exists(Ak| B->exists(Bl| Ak.equal(Bl)))
 and A->forAll(Ar| B->exists(Bs| Ar.equal(Bs)
 or Ar.inside(Bs))))</pre>

</td>
</tr>
<tr>
<td>1 0 1
0 1 1
0 0 1</td>
<td>
Matrice 7
Inverse de la matrice 4


$(\forall j \in 1..m, \exists i \in 1..n \mid \langle A_i, \text{equal}, B_j \rangle) \wedge$
$(\exists k \in 1..n, \forall l \in 1..m \mid \langle A_k, \text{disjoint}, B_l \rangle) \wedge$
$(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, \text{equal}, B_s \rangle \vee \langle A_r, \text{disjoint}, B_s \rangle)$

<pre>B->forAll(Bj| A->exists(Ai| Ai.equal(Bj))) and
 A->exists(Ak| B->forAll(Bl| Ak.disjoint(Bl))) and
 A->forAll(Ar| B->forAll(Bs| Ar.equal(Bs) or Ar.disjoint(Bs)))</pre>

</td>
</tr>
</table>

1 0 1 0 1 1 1 1 1	*Matrice 8* La frontière de *A* (resp. *B*) n'intersecte jamais l'intérieur de *B* (resp. *A*). Les relations autorisées par cette condition sont *equal*, *disjoint* et *meet*. L'intérieur de *A* intersecte celui de *B*, donc au moins un composant de *A* et un composant de *B* sont égaux. Les intérieurs intersectent les extérieurs de chaque région composites, ainsi, un composant de chaque région doit être adjacent à un composant de l'autre région ou disjoint de tous les composants de l'autre région.  • $\exists i \in 1..n, \exists j \in 1..m, \mid \langle A_i, equal, B_j \rangle \wedge$ $((\exists k \in 1..n, k \neq i, \exists l \in 1..m, l \neq j \mid \langle A_k, meet, B_l \rangle) \vee$ $((\exists k \in 1..n, \forall l \in 1..m \mid \langle A_k, disjoint, B_l \rangle$ $\wedge (\exists l \in 1..m, \forall k \in 1..n \mid \langle A_k, disjoint, B_l \rangle))) \wedge$ $(\forall r \in 1..n, \forall s \in 1..m, \langle A_r, equal, B_s \rangle \vee \langle A_r, meet, B_s \rangle \vee \langle A_r, disjoint, B_s \rangle)$ • `A->exists(Ai| B->exists(Bj| Ai.equal(Bj))) and (A->exists(Ak| B->exists(Bl| Ak.meet(Bl))) or (A->exists (Ak| B->forAll(Bl| Ak.disjoint(Bl))) and B->exists(Bv| A->forAll(At| At.disjoint(Bv))))) and A->forAll(Ar| B->forAll(Bs| Ar.equal(Bs) or Ar.meet(Bs) or Ar.disjoint(Bs)))`
1 0 1 1 0 1 1 1 1	*Matrice 9* La frontière de *B* intersecte seulement l'extérieur de *A*. Les composants des deux régions qui interagissent sont en relation *disjoint* ou *inside*. L'intérieur et l'extérieur de *B* intersectent l'intérieur, l'extérieur et la frontière de *A*. Donc au moins un composant de *A* doit être *disjoint* de tous les composants de *B* et un composant de *A* est à l'intérieur d'un composant de *B*.  • $(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, inside, B_j \rangle) \wedge$ $(\exists k \in 1..n, \forall l \in 1..m \mid \langle A_k, disjoint, B_l \rangle) \wedge$ $(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, inside, B_s \rangle \vee \langle A_r, disjoint, B_s \rangle)$ • `A->exists(Ai| B->exists(Bj| Ai.inside(Bj))) and A->exists(Ak| B->forAll(Bl| Ak.disjoint(Bl))) and A->forAll(Ar| B->forAll(Bs| Ar.inside(Bs) or Ar.disjoint(Bs)))`
1 0 1 1 1 1 1 1 1	*Matrice 10* L'intérieur de *A* n'intersecte jamais la frontière de *B*. Cela correspond aux quatre cas suivants : • *(a)* Un composant de *A* est adjacent à un composant de *B* et un composant de *A* est à l'intérieur d'un composant de *B*.

$(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, meet, B_j \rangle) \wedge$

$(\exists k \in 1..n, \exists l \in 1..m \mid \langle A_k, inside, B_l \rangle) \wedge$

$(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, coveredBy, B_s \rangle \vee \langle A_r, disjoint, B_s \rangle$

$\vee \langle A_r, meet, B_s \rangle \vee \langle A_r, equal, B_s \rangle \vee \langle A_r, inside, B_s \rangle)$

- *(b)* Deux composants de *A* sont respectivement adjacent et recouvert par deux composants de *B*.

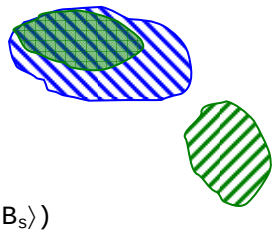
$(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, meet, B_j \rangle) \wedge$

$(\exists k \in 1..n, \exists l \in 1..m \mid \langle A_k, coveredBy, B_l \rangle) \wedge$

$(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, coveredBy, B_s \rangle \vee \langle A_r, disjoint, B_s \rangle$

$\vee \langle A_r, meet, B_s \rangle \vee \langle A_r, equal, B_s \rangle \vee \langle A_r, inside, B_s \rangle)$

- *(c)* Un composant de *A* est recouvert par un composant de *B* et un composant de *A* est disjoint de tous les composants de *B*.



$(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, coveredBy, B_j \rangle) \wedge$

$(\exists k \in 1..n, \forall l \in 1..m \mid \langle A_k, disjoint, B_l \rangle) \wedge$

$(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, coveredBy, B_s \rangle$

$\vee \langle A_r, disjoint, B_s \rangle \vee \langle A_r, equal, B_s \rangle \vee \langle A_r, inside, B_s \rangle)$

- *(d)* Un composant de *A* est à l'intérieur d'un composant de *B*, un composant de *A* est disjoint de tous les composants de *B* et un couple de composants des deux régions sont égaux.

$(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, inside, B_j \rangle) \wedge$

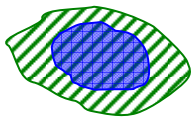
$(\exists k \in 1..n, \forall l \in 1..m \mid \langle A_k, disjoint, B_l \rangle) \wedge$

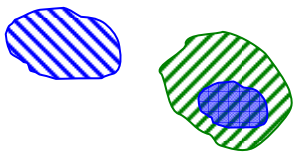
$(\exists t \in 1..n, \exists v \in 1..m \mid \langle A_t, equal, B_v \rangle) \wedge$

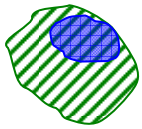
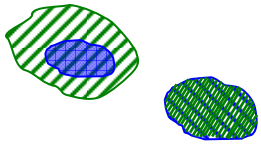
$(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, coveredBy, B_s \rangle$

$\vee \langle A_r, disjoint, B_s \rangle \vee \langle A_r, equal, B_s \rangle \vee \langle A_r, inside, B_s \rangle)$

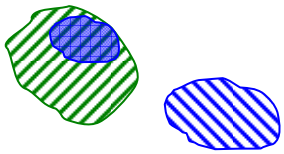
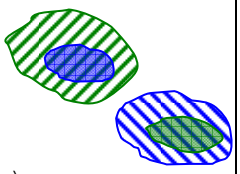
- ```
  ((A->exists(A_i|B->exists(B_j|A_i->meet(B_j))) and
  (A->exists(A_k|B->exists(B_l|A_k->inside(B_l) or
  A_k->coveredBy(B_l))))) or
  (A->exists(A_i|B->forAll(B_j|A_i->coveredBy(B_j))) and
  A->exists(A_k|B->forAll(B_l|A_k->disjoint(B_l)))) or
  (A->exists(A_i|B->exists(B_j|A_i->inside(B_j))) and
  A->exists(A_k|B->forAll(B_l|A_k->disjoint(B_l))) and
  A->exists(A_t|B->exists(B_v|A_t->equal(B_v))))) and
  A->forAll(A_r|B->forAll(B_s|A_r->coveredBy(B_s) or
  A_r->disjoint(B_s) or A_r->meet(B_s)
  or A_r->equal(B_s) or A_r->inside(B_s)))
  ```
  ```

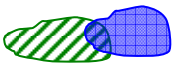
1 1 1 0 0 1 0 0 1	*Matrice 11* ⇔ *contains* L'intérieur et la frontière de *B* intersectent seulement l'intérieur de *A*. Autrement dit, chaque composant de *B* est contenu dans un composant de *A*.  • $\forall j \in 1..m, \exists i \in 1..n \mid \langle A_i, \text{contains}, B_j \rangle$ • `B->forAll(Bj| A->exists(Ai| Ai.contains(Bj)))`

1 1 1
0 0 1
1 1 1
	Matrice 12
	Inverse de la matrice 9
	 
	• $(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{contains}, B_j \rangle) \wedge$
	$(\exists l \in 1..m, \forall k \in 1..n \mid \langle A_k, \text{disjoint}, B_l \rangle) \wedge$
	$(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, \text{contains}, B_s \rangle \vee \langle A_r, \text{disjoint}, B_s \rangle)$
	• `A->exists(Ai| B->exists(Bj| Ai.contains(Bj))) and`
	`B->exists(Bl| A->forAll(Ak| Ak.disjoint(Bl))) and`
	`A->forAll(Ar| B->forAll(Bs| Ar.contains(Bs)`
	`or Ar->disjoint(Bs)))`

1 1 1
0 1 1
0 0 1
	Matrice 13 ⇔ covers
	L'intérieur de *A* intersecte seulement l'intérieur de *B* et l'extérieur de *B* intersecte seulement l'extérieur de *A* comme dans la matrice 5, mais les frontières des deux régions s'intersectent au moins une fois. Cela correspond aux deux cas suivants :
	• *(a)* Il existe au moins une relation *covers* entre un composant de *A* et un composant de *B* et chaque composant de *A* est en relation *equal*, *contains* ou *covers* avec un composant de *B*.
	 
	$\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{covers}, B_j \rangle \wedge$
	$\forall s \in 1..m, \exists r \in 1..n, (\langle A_r, \text{equal}, B_s \rangle \vee$
	$\langle A_r, \text{contains}, B_s \rangle \vee \langle A_r, \text{covers}, B_s \rangle)$
	(b) Il existe au moins un couple de composants des deux régions qui sont égaux, et un composant de *A* contient un composant de *B*. Les autres couples sont en relation *equal* ou *contains*.
	 
	$\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{contains}, B_j \rangle \wedge$
	$\exists k \in 1..n, k \neq i, \exists l \in 1..m, l \neq j \mid \langle A_k, \text{equal}, B_l \rangle \wedge$
	$\forall r \in 1..n, \exists s \in 1..m, (\langle A_r, \text{equal}, B_s \rangle \vee \langle A_r, \text{contains}, B_s \rangle)$
	• `(A->exists(Ai| B->exists(Bj| Ai.covers(Bj)))`
	`and A->forAll(Ar| B->exists(Bs| Ar.equal(Bs)`
	`or Ar.contains(Bs) or Ar.covers(Bs))))`
	`or`
	`A->exists(Ai| B->exists(Bj| Ai->contains(Bj)))`
	`and A->exists(Ak| B->exists(Bl| Ak->equal(Bl)))`
	`and B->forAll(Bs| A->exists(Ar| Ar.equal(Bs)`
	`or Ar.contains(Bs))))`

1 1 1 0 1 1 1 1 1	*Matrix 14* *Inverse de la matrice 10* • *(a)* Un composant de *A* est adjacent à un composant de *B* et un composant de *A* contient un composant de *B*. $(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{meet}, B_j \rangle) \wedge$ $(\exists k \in 1..n, \exists l \in 1..m \mid \langle A_k, \text{contains}, B_l \rangle) \wedge$ $(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, \text{covers}, B_s \rangle \vee \langle A_r, \text{disjoint}, B_s \rangle$ $\vee \langle A_r, \text{meet}, B_s \rangle \vee \langle A_r, \text{equal}, B_s \rangle \vee \langle A_r, \text{contains}, B_s \rangle)$ • *(b)* Un composant de *A* est adjacent à un composant de *B*, et un composant de *A* recouvre un composant de *B* $(\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{meet}, B_j \rangle) \wedge$ $(\exists k \in 1..n, \exists l \in 1..m \mid \langle A_k, \text{covers}, B_l \rangle) \wedge$ $(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, \text{coversB}_s \rangle \vee \langle A_r, \text{disjoint}, B_s \rangle$

	$\vee \langle A_r, meet, B_s\rangle \vee \langle A_r, equal, B_s\rangle \vee \langle A_r, contains, B_s\rangle)$ • *(c)* Un composant de *A* recouvre un composant de *B* et un composant de *B* est disjoint de tous les composants de *A*.  $(\exists i\in 1..n, \exists j\in 1..m \mid \langle A_i, covers, B_j\rangle) \wedge$ $(\exists l\in 1..m, \forall k\in 1..n \mid \langle A_k, disjoint, B_l\rangle) \wedge$ $(\forall r\in 1..n, \forall s\in 1..m \mid \langle A_r, covers, B_s\rangle$ $\vee \langle A_r, disjoint, B_s\rangle \vee \langle A_r, equal, B_s\rangle \vee \langle A_r, contains, B_s\rangle)$ • *(d)* Un composant de *A* contient un composant de *B*, un composant de *A* est égal à un composant de *B* et un composant de *B* est disjoint de tous les composants de *A*. $(\exists i\in 1..n, \exists j\in 1..m \mid \langle A_i, contains, B_j\rangle) \wedge$ $(\exists l\in 1..m, \forall k\in 1..n \mid \langle A_k, disjoint, B_l\rangle) \wedge$ $(\exists v\in 1..m, \exists t\in 1..n \mid \langle A_t, equal, B_v\rangle) \wedge$ $(\forall r\in 1..n, \forall s\in 1..m \mid \langle A_r, covers, B_s\rangle$ $\vee \langle A_r, disjoint, B_s\rangle \vee \langle A_r, equal, B_s\rangle \vee \langle A_r, contains, B_s\rangle)$ • `((A->exists(A_i|B->exists(B_j|A_i->meet(B_j))) and (A->exists(A_k|B->exists(B_l|A_k->inside(B_l) or A_k->coveredBy(B_l))))) or (A->exists(A_i|B->forAll(B_j|A_i->coveredBy(B_j))) and A->exists(A_k|B->forAll(B_l|A_k->disjoint(B_l)))) or (A->exists(A_i|B->exists(B_j|A_i->inside(B_j))) and A->exists(A_k|B->forAll(B_l|A_k->disjoint(B_l))) and A->exists(A_t|B->exists(B_v|A_t->equal(B_v))))) and A->forAll(A_r|B->forAll(B_s|A_r->coveredBy(B_s) or A_r->disjoint(B_s) or A_r->meet(B_s) or A_r->equal(B_s) or A_r->inside(B_s)))`
1 1 1 1 0 1 1 1 1	*Matrice 15* Les frontières des 2 régions composites ne s'intersectent jamais.  • $\exists i\in 1..n, \exists j\in 1..m \mid \langle A_i, contains, B_j\rangle \wedge$ $\exists k\in 1..n, \exists l\in 1..m \mid \langle A_k, inside, B_l\rangle \wedge$ $\forall r\in 1..n, \forall s\in 1..m, (\langle A_r, contains, B_s\rangle \vee \langle A_r, inside, B_s\rangle \vee$ $A_r, disjoint, B_s\rangle)$ • `A->exists(A_i|B->exists(B_j|A_i->contain(B_j))) and A->exists(A_k|B->forAll(B_l|A_k->inside(B_l))) and A->forAll(A_r|B->forAll(B_s|A_r->contain(B_s) or A_r->inside(B_s) or A_r->disjoint(B_s)))`

1 1 1 1 1 1 1 1 1	*Matrice 16 ⇔ overlap* Les 9 intersections sont non nulles. Cela correspond aux trois cas suivants : • *(a)* Il y a une relation *overlap* entre au moins deux composants des deux régions. $\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{overlap}, B_j \rangle$  *(b)* Un composant de *A* est à l'intérieur de ou recouvert par un composant de *B* et un composant de *A* recouvre un composant de *B*, ou réciproquement. $(\exists i \in 1..n, \exists j \in 1..m \mid (\langle A_i, \text{inside}, B_j \rangle \vee \langle A_i, \text{coveredBy}, B_j \rangle) \wedge \exists r \in 1..n, r \neq i, \exists s \in 1..m, s \neq j \mid \langle A_r, \text{cover}, B_s \rangle) \vee (\exists i \in 1..n, \exists j \in 1..m \mid (\langle A_i, \text{contain}, B_j \rangle \vee \langle A_i, \text{cover}, B_j \rangle) \wedge \exists r \in 1..n, r \neq i, \exists s \in 1..m, s \neq j \mid \langle A_r, \text{coveredBy}, B_s \rangle)$ *(c)* There is at least one relation meet and one relation equal between some A's components and some B's components. A's component is in relation meet, equal or disjoint with all B's components and there is a relation inside between an A's component and a B's component and reciprocally. $\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \text{inside}, B_j \rangle \wedge \exists k \in 1..n, k \neq i, \exists l \in 1..m, l \neq j \mid \langle B_l, \text{inside}, A_k \rangle) \wedge \exists r \in 1..n, \exists s \in 1..m \mid (\langle A_r, \text{meet}, B_s \rangle \vee \langle A_r, \text{equal}, B_s \rangle)$ • `(A->exists(A_i| B->exists(B_j| A_i->overlap(B_j)))) or (A->exists(A_i| B->exists(B_j| A_i->inside(B_j) or A_i->coveredBy(B_j))) and A->exists(A_r| B->exists(B_s| A_r->cover(B_s)))) or (A->exists(A_i| B->exists(B_j| A_i->contain(B_j) or A_i->cover(B_j))) and A->exists(A_r| B->exists(B_s| A_r->coveredBy(B_s)))) or (A->exists(A_i| B->exists(B_j| A_i->inside(B_j))) and A->exists(A_k| B->exists(B_l| B_l->inside(A_k))) and A->exists(A_r| B->exists(B_s| A_r->meet(B_s) or A_r->equal(B_s))))`