UNIVERSITE LIBRE DE BRUXELLES
Faculté des Sciences appliquées

Année académique 2001-2002

INTERROGATION DE BASES DE DONNEES SPATIO-TEMPORELLES : CONCEPTION D'UN EDITEUR VISUEL DE REQUETES

Directeur de mémoire : Mr ZIMANYI

Travail de fin d'études
présenté par BUSTELO ALVAREZ Felipe
en vue de l'obtention du grade
d'ingénieur civil informaticien

Résumé

Ce travail de fin d'études s'inscrit dans le cadre du projet de recherche européen MURMUR (Multiple Représentation Multiple Résolution) qui a pour objectif la définition d'un modèle de données qui permet de décrire au niveau conceptuel des données spatio-temporelles sous plusieurs représentations.

Le modèle conceptuel élaboré par MURMUR est MADS (Modélisation d'Applications à Données Spatio-temporelles). MADS est un modèle conceptuel objet/association.

Le but de ce travail de fin d'études est la création d'une interface graphique permettant la création de formules de dérivation pour les attributs dérivés dans l'interface MADS. Une vue d'ensemble du projet sera faite afin de mieux comprendre le contexte dans lequel va s'inscrire ce travail.

Remerciements

Je tiens tout d'abord à remercier M. Zimányi, mon promoteur, qui m'a proposé le sujet du présent travail de fin d'études et qui a toujours été à ma disposition pour répondre à mes questions.

Je remercie mes parents, mon frère et Nathalie qui m'ont aidé moralement pendant toute cette année.

Pour finir, je dis également merci au MFC qui, chaque week-end, m'a permis de me changer les idées.

Table des matières

1. Introduction

Ce travail de fin d'études s'inscrit dans le cadre du projet de recherche MURMUR (Multiple Représentation Multiple Résolution) financé par le programme IST de la Communauté Européenne. Ce projet regroupe deux partenaires suisses, l'université de Lausanne (UNIL) et l'école polytechnique de Lausanne (EPFL), deux partenaires français les Centres de recherche du CEMAGREF et de l'Institut géographique national (IGN) et deux partenaires belges, l'Université Libre de Bruxelles et la société STAR informatic.

Une base de données permet de conserver de façon organisée un ensemble de données. Elle est conçue afin de répondre aux besoins en information d'un ensemble d'utilisateurs. La première étape dans la conception d'une base de données consiste en une modélisation conceptuelle, centrée sur les caractéristiques intrinsèques des données de l'application. Les modèles de type entité-association, ou objet-association, sont les plus connus et pratiqués au niveau conceptuel. Ensuite vient la modélisation logique conduisant à une mise en œuvre sur un logiciel de type relationnel ou orienté objets.

Les bases de données spatio-temporelles sont des bases de données où diverses représentations du monde réel, à différents niveaux de résolution, sont stockées dans une base de données unique et où la gestion de représentations multiples est faite à différents points du temps. Dans ce cas, les modèles conceptuels classiques sont inadaptés car ils ignorent les concepts spatio-temporels. Ce constat a conduit à développer un nouveau modèle conceptuel, bien adapté à l'information spatio-temporelle.

Le projet européen MURMUR a pour objectif la définition d'un modèle de données qui permet de décrire au niveau conceptuel des données spatio-temporelles sous plusieurs représentations (à différentes échelles, à différents

instants et selon différents points de vue) et d'assurer la cohérence de ces représentations multiples.

Le modèle conceptuel élaboré par MURMUR est MADS (Modélisation d'Applications à Données Spatio-temporelles). Une interface visuelle et intuitive sera proposée aux utilisateurs tant pour la définition que pour la manipulation de ces bases de données spatio-temporelles à représentations multiples.

MADS est un modèle conceptuel objet/association. Chaque objet/association a des caractéristiques que l'on appelle attributs. La valeur d'un attribut peut être calculée, c'est-à-dire inférée automatiquement par le système, à partir des valeurs d'autre(s) attribut(s) appartenant au même objet ou à la même association, ou à d'autres objets ou associations qui lui sont liés. Ces attributs sont appelés attributs dérivés.

Mon travail de fin d'études a pour but la création d'une interface graphique permettant la création de formules de dérivation pour les attributs dérivés dans l'interface MADS.

2. Base de données spatio-temporelles

Une base de données est conçue afin de répondre aux besoins en information d'un ensemble d'utilisateurs et conserve cette information de façon organisée. Cette information, stockée dans la base de données, correspond à une description du monde réel appelée représentation. Alors que le monde réel est supposé unique, il en existe une multitude de représentations. Chaque représentation dépend de la finalité qui préside à son élaboration. Des représentations conçues pour des utilisateurs différents pourront être hétérogènes en ce qui concerne l'information décrite, la description de l'information, l'organisation de la description, les contraintes associées à la description,…

Dans le cas des bases de données spatio-temporelles, aussi appelées bases de données géographiques, les données sont des représentations du monde réel faites à différents niveaux de résolution. Celles-ci sont stockées dans une base de données unique dans laquelle la gestion de représentations multiples est faite à différents points du temps.

Par exemple, dans une base de données géographique multi-échelle, le même bâtiment peut avoir une représentation géométrique précise de sa surface à une échelle 1/10 000 et une représentation moins précise à une échelle 1/50 000. En considérant la dimension temporelle, une rivière peut être associée à plusieurs géométries pour représenter l'évolution de son lit au cours des années.

Pour éditer des cartes à des échelles variées, une solution théoriquement idéale consisterait à posséder une seule base de données dans laquelle les informations spatiales sont conservées au niveau de détail le plus précis et d'en dériver les différentes cartes. Mais la généralisation, processus permettant le passage d’une échelle à une autre, ne peut pas aujourd'hui être entièrement réalisée automatiquement et c'est un processus complexe et long, ce qui invalide cette solution. La solution alternative actuellement suivie par les instituts

cartographiques consiste à effectuer une fois le processus interactif de généralisation et à conserver son résultat dans une base de données. Il existe donc une base de données pour chaque carte à une échelle différente. Ces bases de données sont redondantes parce qu'elles décrivent le même espace géographique et sont gérées indépendamment, ce qui entraîne des problèmes d'incohérence des données et ne permet pas la propagation des mises à jour. Les utilisateurs de données spatiales étudient donc la possibilité d'assurer la cohérence de leurs différentes bases de données.

Il est fréquent que différents utilisateurs se partagent la même base de données dans des contextes différents d'utilisation, il est donc nécessaire de pouvoir conserver au sein de la même base de données plusieurs représentations du monde réel. C'est le concept de la « casquette », un utilisateur qui possède une casquette rouge n'a accès qu'aux données ayant la couleur rouge comme attribut. En effet, même s'ils sont intéressés par les mêmes phénomènes du monde réel, chaque utilisateur souhaitera avoir une représentation spécifique à ses besoins. C'est particulièrement vrai dans le domaine spatial où l'espace représenté, l'espace géographique, est partagé.

Pour une administration cadastrale, par exemple, des environnementalistes ou des botanistes, étudiant la même parcelle du monde réel, auront des besoins différents en information pour cette parcelle.

En résumé, dans une base de données spatio-temporelle, les mots clefs sont :

- la multi-représentation, le fait de faire coexister pour une même base de données plusieurs représentations du monde réel ;
- multi-résolution due aux différentes échelles ;
- temporalité, suite à l'évolution de l'information dans le temps.

2.1. Modélisation conceptuelle des données spatio-temporelles

La première étape dans la conception d'une base de données consiste en une modélisation conceptuelle, centrée sur les caractéristiques intrinsèques des données de l'application. Dans le cas de données spatio-temporelles, cette modélisation possède trois dimensions :

- thématique ;
- spatiale ;
- temporelle.

2.1.1. Dimension thématique

La dimension thématique comporte trois parties : la délimitation, l'abstraction et la focalisation.

Un schéma est une description du contenu de la base de données, définissant les types d'objets, des liens, des propriétés et des méthodes qui vont être stockés dans la base de données. Ce schéma est conçu lors de l'étape d'analyse des besoins des utilisateurs et en est une traduction. Lors de cette étape, le sous-ensemble des phénomènes du monde réel à décrire dans la base de données est identifié, c'est la **délimitation** de l'univers. Une représentation du monde réel stockée dans une base de données doit donc être conforme au schéma de la base de données.

Ensuite, les phénomènes retenus sont représentés avec les concepts du modèle. Le processus de représentation associe à un phénomène du monde réel, un élément de la base de données. Celui-ci dépend à la fois du phénomène lui-même (caractéristiques intrinsèques) et de la finalité de la représentation du phénomène. L'élaboration d'une représentation se fait progressivement par une répétition du processus d'**abstraction** (élimination de détails non pertinents) et de **focalisation** (étude plus en détails) qui visent à dégager et caractériser les phénomènes qui nous intéressent.

La représentation d'un phénomène du monde réel est une description simplifiée de celui-ci, mettant en évidence ses caractéristiques essentielles. Une telle

représentation s'exprime à l'aide d'un concept du modèle de données utilisé, possède un type défini par l'ensemble des propriétés qui le composent et possède un ensemble de valeurs pour les propriétés décrites dans le type.

2.1.2. Dimension spatiale

Lorsque le phénomène à représenter est spatial, sa représentation comporte une partie permettant de décrire ses caractéristiques spatiales : son type spatial (par exemple : point, ligne ou surface) et sa valeur (géométrie).

Les SIG (Système d'Information Géographique) offrent deux manières de décrire les relations entre les objets et l'espace: le point de vue discret, où la base de données contient des objets qui peuvent être localisés dans l'espace, et le point de vue continu, où la base de données contient des régions de l'espace sur lesquelles des variables sont définies comme des champs continus de valeurs.

2.1.3. Dimension temporelle

La connaissance de l'évolution temporelle des données, en particulier spatiales, est souvent indispensable pour comprendre la dynamique des phénomènes du monde réel. Dans le domaine géographique, cette évolution temporelle permet, par exemple, de faire des prédictions d'avalanches, prévoir les effets de la pollution sur l'environnement,…

Des données sont dites temporelles si on doit garder une trace de leur évolution dans le temps.

Il y a deux manières d'ajouter la notion de temps à des données : la première garde un historique des valeurs des données et leur prédiction pour le futur. Ces valeurs sont ensuite associées à une période de validité. L'autre méthode tient compte de la dynamique inter-objets : quel objet a créé cet autre objet, cet objet existait avant cet autre objet,… Or, ce type de relations est rarement supporté par les systèmes existants, notamment dans la gestion de phénomènes naturels.

Pratiquement, seule la première méthode est employée : les différentes valeurs prises par les données au cours du temps sont conservées, chacune associée à un

élément temporel qui décrit sa période de validité. Ces éléments temporels sont du type : instant précis, intervalle de temps, ensemble d'instants, ensemble d'intervalles,…

Quand de l'information variant dans le temps est gardée dans un système, une fonctionnalité importante est le « voyage dans le temps », il permet à l'utilisateur de visualiser l'état d'un objet à un moment précis dans le temps. Cependant, cette fonctionnalité est difficile à implémenter pour différentes raisons. Par exemple, il peut exister des périodes pendant lesquelles les données ne sont pas définies dans le temps parce qu'on a eu des problèmes avec les mesures des données pour cette époque. On aurait donc besoin d'un support qui, en plus, puisse remplir l'information incomplète, ce qui ne se fait pas encore dans les bases de données traditionnelles.

2.2. Représentation multiple

L'élaboration d'une représentation du monde réel est guidée par les besoins d'un groupe d'utilisateurs. Chaque représentation est spécifique à un groupe d'utilisateurs que ce soit dans la délimitation de l'univers du discours ou dans la représentation des phénomènes. Le même phénomène peut donc être décrit différemment selon le groupe d'utilisateurs. Le choix du concept, l'élaboration du type ou l'insertion des valeurs, est spécifique à chaque groupe d'utilisateurs. C'est ce qu'on appelle la représentation multiple. Il n'existe donc pas une représentation unique du monde réel.

Les différents groupes d'utilisateurs ne travaillent pas nécessairement sur le même sous-ensemble de phénomènes du monde réel et définiront donc un univers du discours différent.

Par exemple, le constructeur d'une route a besoin de connaître le numéro de la route, les matériaux utilisés pour construire celle-ci et son épaisseur, alors que le cartographe ne s'intéresse qu'à son numéro, son type (route nationale ou départementale) et son nombre de voies. Dans le cas de la représentation de phénomènes naturels, comme les avalanches, les utilisateurs travaillant sur la gestion du risque d'avalanche en montagne et ceux travaillant sur la gestion des bâtiments décrivent les routes, les ponts et les lacs, alors que seul l'utilisateur travaillant sur la gestion des bâtiments décrit les bâtiments et seul l'utilisateur travaillant sur la gestion du risque en montagne décrit les avalanches.

L'univers de représentation de plusieurs utilisateurs peut donc être disjoint ou avoir certains phénomènes en commun.

Nous introduisons la notion de point de vue qui est défini comme la matérialisation des besoins en information d'un groupe d'utilisateurs. Un point de vue détermine un univers du discours et une modélisation conceptuelle de cet univers. Deux points de vue sur le monde réel peuvent être disjoints ou en intersection selon que leur univers du discours possède ou non des phénomènes en commun. Même si un point de vue correspond aux besoins d'un groupe d'utilisateurs, il ne détermine cependant pas une représentation unique du monde réel. En effet, le même groupe d'utilisateurs peut avoir besoin de décrire des représentations alternatives pour le même phénomène. Par exemple, si on considère une base de données dans laquelle sont représentés les Véhicules et les Objets de Collection, une voiture de collection aura deux représentations dans cette base de données: une en tant que véhicule et une en tant qu'objet de collection. Dans les modèles classiques, cette multiplicité de représentation dans des classes différentes est réalisée au moyen de hiérarchies de généralisation/spécialisation.

2.3. Multi-résolution

Un groupe d'utilisateurs peut aussi souhaiter décrire le même phénomène du monde réel à différents niveaux de détail. A la base, toute représentation du monde réel est une description simplifiée de celui-ci.

La description est simplifiée tout d'abord parce qu'elle ne contient qu'un sous-ensemble des phénomènes du monde réel. De plus, pour les phénomènes retenus, elle ne décrit qu'un sous-ensemble de leurs propriétés. Enfin, pour certaines propriétés, la valeur stockée est une approximation de la valeur réelle. En particulier, la géométrie de la plupart des phénomènes géographiques est infiniment complexe et vouloir représenter cette information nécessite d'en faire une approximation. L'approximation de la géométrie des objets répond aussi aux besoins des utilisateurs: ont-ils besoin d'un niveau de détail tel que la géométrie d'une route est représentée comme une surface ou est-ce suffisant de la représenter comme un objet linéaire?

Figure 1
Multi-résolution, représentation d'une route à deux échelles différentes

La figure 1 montre une route représentée à deux échelles différentes, un pilote de rallye utilisera la carte à l'échelle 1/50 000 pour avoir toutes les courbes et ainsi faire un bon chrono lors de sa course mais le conducteur normal employant cette même route dans la vie courante n'a pas besoin de connaître avec précision la dimension et la forme des tournants de la route. De ce fait, une échelle 1/100 000 est suffisante pour ses besoins.

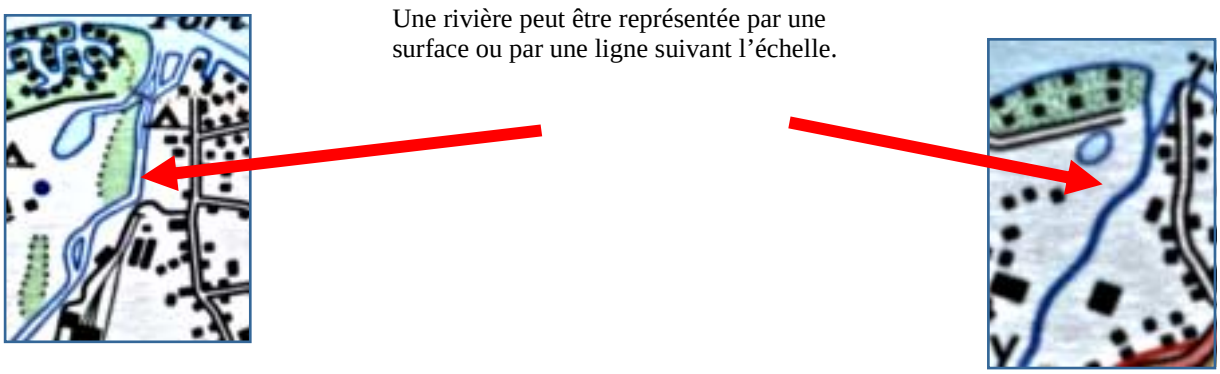


Figure 2
Multi-résolution, effets sur le niveau de détail

La figure 2 présente deux représentations du monde réel à des niveaux de détails différents.

Ces cartes sont une visualisation du contenu de la base de données. Dans la figure 2, on remarque que la carte située à gauche est plus détaillée que la carte située à droite. Si on considère les deux cartes, on constate que dans la carte la moins détaillée certains phénomènes du monde réel ne sont plus représentés : l'îlot dans le delta, de même que la zone de prairie autour de la rivière ont disparu. De plus, la géométrie de certains phénomènes est plus approximative. La zone d'eau qui débouche dans le delta a perdu le cours d'eau le reliant à ce delta, on constate que certains petits détails de la géométrie de la zone située dans la partie supérieure gauche de l'image ont été simplifiés : les cours d'eau ont disparu.

Cette description simplifiée du monde réel, aussi appelée abstraction, peut être plus ou moins réductrice en terme d'information : elle possède un certain niveau de détail. Etablir le niveau de détail d'une représentation du monde réel revient à évaluer la richesse de la description spatiale et thématique de cette représentation. Cette richesse descriptive s'exprime par le nombre de phénomènes représentés et par la richesse de chaque représentation de ce phénomène. Plus la représentation du monde réel comprend de représentations de phénomènes et moins celles-ci sont approximatives, plus la description est riche.

Lorsqu'on est en présence de plusieurs représentations du monde réel, connaître le niveau de détail d'une représentation permet de choisir les données les plus appropriées à chaque utilisation. De plus, connaître le niveau de détail d'une représentation permet d'informer sur la qualité des données manipulées et permet ainsi d'interpréter au mieux les résultats des analyses portant sur ces données.

Enfin, le niveau de détail d'une représentation est aussi un indicateur du volume des données. Il convient donc de décrire de façon précise le niveau de détail d'une représentation.

3. Projet MURMUR

3.1. Besoins

La coexistence de modélisations réalisées conformément aux spécifications de plusieurs groupes d'utilisateurs induit des besoins spécifiques : différents groupes d'utilisateurs ne travaillent pas nécessairement sur le même sous-ensemble de phénomènes du monde réel et définissent donc des univers du discours différents et le même phénomène peut être décrit différemment selon le groupe d'utilisateurs.

Cette coexistence au sein d'une même base de données de plusieurs représentations différentes d'une même information aboutit à une situation de représentation multiple. La problématique d'utilisation de la représentation multiple implique de pouvoir rapprocher les différentes représentations d'une même information, soit pour vérifier et maintenir la cohérence des données soit pour naviguer entre les représentations.

La multi-résolution résulte de la coexistence dans la base de données, de données décrivant un même univers de discours (par exemple : en information spatiale, une même zone géographique) à des niveaux de détail différents, avec des résolutions spatiales et/ou sémantiques différentes. Dans les applications géographiques, les données multi-représentées correspondent à un ensemble de données faisant l'objet d'une acquisition groupée, en général, à partir d'une même source sur laquelle on a appliqué des opérations de généralisation cartographique dont on a mémorisé le résultat.

Les modèles de données actuels (relationnel, orienté objet et relationnel objet) permettent la modélisation directe de certaines situations de multi-représentation mais ne couvrent pas la totalité des besoins dans ce domaine, il en va de même

pour la multi-résolution. Ce constat a conduit à développer un nouveau modèle conceptuel, bien adapté à l'information spatio-temporelle.

3.2. Modèle développé par MURMUR

Le projet européen MURMUR a pour objectif la définition de ce modèle de données qui permet de décrire au niveau conceptuel des données spatio-temporelles sous plusieurs représentations (à différentes échelles, à différents instants et selon différents points de vue) et d'assurer la cohérence de ces représentations multiples.

Ce nouveau modèle est basé sur un formalisme simple afin d'éviter les ambiguïtés dans son interprétation. Il est capable de décrire la complexité des phénomènes du monde réel tant pour la description de l'espace que pour la description dans le temps des données. De plus, ce modèle supporte la multi-representation et la multi-résolution.

MURMUR fournit des outils pour la gestion de représentations multiples qui sont adéquats pour des données géographiques. Une des motivations est l'importance sociale des données géographiques. Une autre motivation est que la diversité des profils utilisateurs est beaucoup plus large dans des applications géographiques par rapport à des applications de bases de données conventionnelles, où une base de données sert généralement une communauté d'utilisateurs appartenant à la même organisation. Beaucoup d'applications sont concernées par le problème des représentations multiples, comme la gestion de l'environnement, la prévention des risques naturels, les systèmes multi-échelles de production de cartes, la gestion cadastrale, et les applications de navigation embarquées.

Dans une perspective scientifique, le principal objectif du projet est de gérer les problèmes dus à la coexistence de plusieurs représentations.

Le modèle d'une base de données spatio-temporelles possède trois dimensions : thématique, spatiale et temporelle. L'élaboration du schéma avec un modèle classique (relationnel ou orienté-objet) fixe une représentation dans la dimension thématique. L'espace de représentation peut être vu comme un espace multidimensionnel, où chaque dimension est relative à la prise en compte d'un paramètre de représentation.

La dimension thématique constitue en quelque sorte la dimension de base de cet espace, car il ne peut y avoir représentation sans définition explicite d'une structure de données. Un point dans cet espace de représentation à une dimension est donc une représentation thématique d'un phénomène.

L'ajout d'une dimension d'étendue spatiale permet de localiser l'information par rapport à un ensemble d'espaces géographiques possibles. Lorsqu'on fait varier la coordonnée dans l'axe d'étendue spatiale ; on obtient, à schéma constant, un ensemble d'instances différentes. La dimension temporelle permet d'associer à une information sa localisation dans le temps. Un point dans cet espace de représentation ainsi augmenté est la représentation spatiale et thématique d'un phénomène à un instant donné. Lorsqu'on fait varier la coordonnée dans l'axe du temps, on obtient sur une ligne parallèle à cet axe toutes les représentations du même phénomène à des instants divers.

Le point essentiel du modèle proposé par MURMUR est **l'orthogonalité** des dimensions structurelle, spatiale et temporelle ce qui les rend indépendantes les une des autres.

Ce modèle conceptuel élaboré par MURMUR est MADS (Modélisation d'Applications à Données Spatio-temporelles). Il propose une interface visuelle et intuitive aux utilisateurs tant pour la définition que pour la manipulation de ces bases de données spatio-temporelles à représentations multiples.

4. Modèle conceptuel MADS

L’élaboration du modèle conceptuel MADS (Modélisation d’Applications à Données Spatio-temporelles) a été guidée par les objectifs suivants:

- l’orthogonalité des dimensions structurelle, spatiale et temporelle ;
- une mise en évidence des aspects semblables des concepts spatiaux et temporels ;
- une définition formelle des concepts ;
- la possibilité de décrire de manière explicite des relations spatiales, temporelles et dynamiques entre les entités ;
- la possibilité de décrire des champs spatiaux continus ;
- la provision de types spatiaux génériques au-delà des types de base ;
- des notations visuelles intuitives.

MADS est un modèle objet/association, dont l'un des objectifs majeurs est d'assurer l'orthogonalité entre la modélisation des structures de données et celle de la spatialité. En effet, un modèle conceptuel spatio-temporel doit offrir aux utilisateurs une grande richesse d’expression, pour répondre à la diversité des besoins, et en même temps leur permettre de décrire un schéma de données lisible et facile à appréhender. Un élément clé pour la réalisation de ce double objectif est l’orthogonalité des trois dimensions structurelle, spatiale et temporelle du modèle. Orthogonalité signifie ici qu’un choix dans l’une des dimensions ne doit pas limiter les possibilités de choix dans une autre dimension.

En particulier, le choix d’une représentation structurelle d’une information (comme objet, attribut, agrégation ou association) doit pouvoir être fait librement, que cette information ait des caractéristiques spatio-temporelles ou non. L’orthogonalité des concepts permet d’obtenir un modèle à la fois simple,

puisque ces concepts sont indépendants, et puissant puisque les concepts peuvent être combinés librement.

La dimension structurelle de MADS permet la modélisation des données classiques en offrant un ensemble de concepts bien connus : type d'objet, attribut, méthode, type d'association, lien de généralisation, lien d'agrégation, ainsi qu'un ensemble de contraintes d'intégrité associées. Les objets et les associations peuvent avoir une structure complexe grâce aux attributs complexes (composés d'autres attributs) et multivalués (prenant plusieurs valeurs). Les types d'objet peuvent être organisés dans des hiérarchies de généralisation (classes/sous-classes) ou d'agrégation. Le type d'association permet la modélisation explicite des liens n-aires entre objets. Les types d'association et d'agrégation peuvent avoir des attributs et des méthodes, tout comme les types d'objet.

Suivant le principe d'orthogonalité, la spatialité et la temporalité peuvent être associées aux types d'objet, aux liens d'association et d'agrégation, ainsi qu'aux attributs. Le modèle offre également la possibilité de représenter des champs continus dans l'espace grâce au concept d'attribut variable. Au point de vue ergonomique, la spatialité et la temporalité sont visualisées dans les diagrammes par des pictogrammes, ce qui en permet une appréhension visuelle immédiate et non ambiguë. Ces aspects ergonomiques sont essentiels pour le développement d'outils d'édition visuelle d'un schéma spatio-temporel ou de formulation visuelle interactive de requêtes d'interrogation et de mise à jour.

Les SIG (Système d'Information Géographique) offrent deux manières de décrire les relations entre les objets et l'espace : le point de vue discret, où la base de données contient des objets qui peuvent être localisés dans l'espace, et le point de vue continu, où la base de données contient des régions de l'espace sur lesquelles des variables sont définies comme des champs continus de valeurs. La vue continue de l'espace est aussi importante car elle permet de décrire des phénomènes dont la valeur varie dans l'espace: c'est, par exemple, le cas des applications souhaitant conserver l'altitude ou le type d'occupation du sol d'une région.

Il existe deux types de champs continus :

- les champs continus décrivant une propriété de l'espace comme la température ou l'altitude. En chaque point, il y a au maximum une valeur pour ces champs ;
- les champs continus décrivant une propriété d'un objet de spatialité à une ou deux dimensions, dont la valeur dépend de l'objet et de la position dans l'emprise de l'objet. Par exemple, le nombre de voies d'une route, une mesure de qualité de l'eau d'une rivière, le degré de toxicité d'un nuage de

pollution. Ces valeurs n'existent qu'à l'intérieur de la géométrie de l'objet. Si des objets se chevauchent (exemple : nuages toxiques), les points situés à l'intersection auront éventuellement une valeur différente suivant le nuage. Ces champs continus peuvent être attachés à la géométrie d'un objet mais aussi à celle de n'importe quel attribut spatial.

Le modèle MADS adopte le point de vue discret, car il est plus fréquent chez ses utilisateurs (gestionnaires du territoire, des eaux, des routes...). Néanmoins, MADS fournit également des concepts pour pouvoir exprimer une vue continue.

Tous ces desiderata ont mené pour le modèle MADS à :

- des schémas clairs et des diagrammes lisibles parce que simples et purement conceptuels (non déformés par des contraintes d'implémentation) ;
- des schémas complets avec l'explicitation des composantes spatiale et temporelle.

Ce sont les qualités principales du modèle MADS.

4.1. Multi-représentation

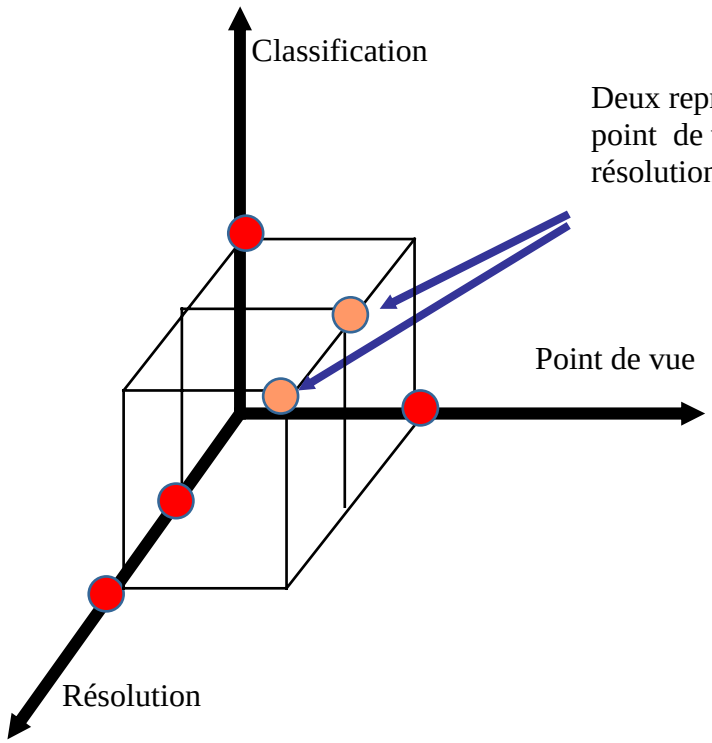
L'approche MADS retient trois dimensions de multi-représentation:

- le **point de vue** : les coordonnées sur cet axe représentent les différents points de vue existants. Ces coordonnées constituent un ensemble discret et en principe non ordonné de points ;

- la **résolution** : le terme générique de résolution recouvre la résolution spatiale et la résolution sémantique. Les coordonnées sur cet axe représentent les résolutions avec lesquelles les représentations ont été élaborées. Les coordonnées sur cet axe peuvent constituer un ensemble ordonné, par exemple du moins précis au plus précis, ou plus généralement un ensemble partiellement ordonné (les résolutions sémantiques n'étant pas nécessairement comparables) ;

- la **classification** : les coordonnées sur cet axe constituent un ensemble discret de points qui représentent les différentes instances dans la population de la base de données. Cette dimension permet de rendre compte de la population pertinente pour chaque combinaison <point de vue,résolution>. Cette population est située sur l'axe vertical qui s'élève à

partir du point correspondant à l'intersection qui matérialise la combinaison sélectionnée. La dimension rend compte également des choix de classification (et donc d'instanciation) faits lors de la conception du schéma de la base. Cette possibilité de classification multiple participe aussi à la multi-représentation. A un même objet (ou association) du monde réel peuvent ainsi correspondre plusieurs instances dans la base de données. Chaque instance correspond à une coordonnée sur l'axe classification. Une instance d'un type décrivant plusieurs représentations donnera plusieurs points dans notre espace tridimensionnel, tous situés sur un même plan orthogonal à l'axe classification. Si un objet est représenté par un ensemble d'instances décrivant une unique représentation, la ligne reliant les points correspondants sera une polyligne zigzaguant dans l'espace en trois dimensions (le changement de classification peut s'accompagner d'un changement de résolution et/ou de point de vue) ;



Deux représentations du même objet au même point de vue et classification mais à deux résolutions différentes.

Figure 3
Espace tridimensionnel de représentation

Un point dans notre espace représente une instance en tant que membre de la population d'un type donné, dans une résolution et un point de vue donnés. Les deux points, dans la figure 3, sont deux représentations du même phénomène décrites au sein d'une seule instance d'un type, selon le même point de vue et à

différentes résolutions. Chaque instance mémorisée dans la base de données doit être caractérisée relativement aux axes de notre espace multidimensionnel, ses coordonnées dans cet espace sont donc définies.

Les figures 4 et 5 montrent deux cas particuliers intéressants dans l'espace tridimensionnel de multi-représentation.

La figure 4 montre le cas d'une base de données mono-résolution.

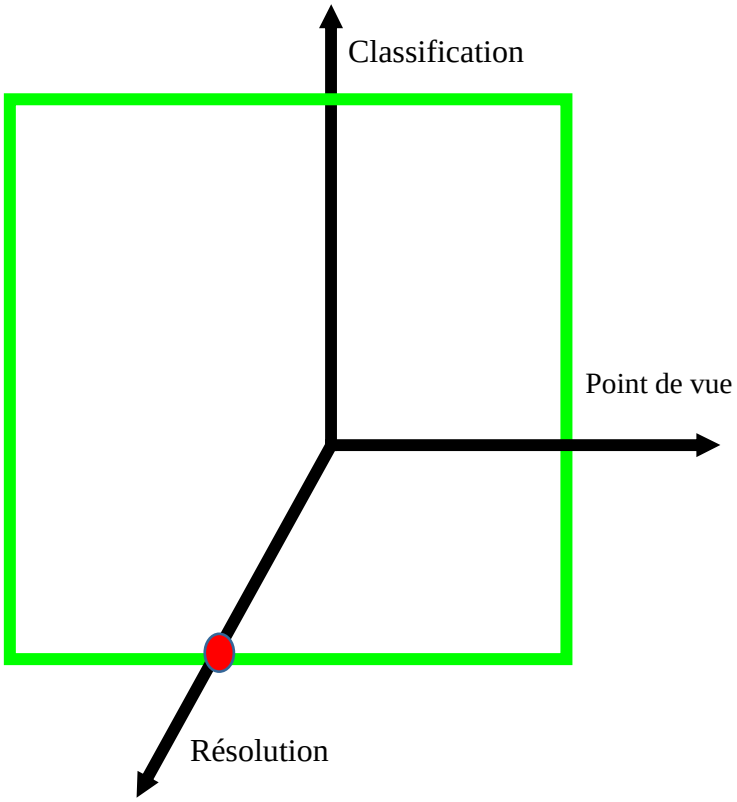


Figure 4
Représentation d'une base de données mono-résolution dans l'espace multi-représentation

Dans la figure 5, la base de données est réduite à une résolution et un point de vue donnés. Ce cas est celui qui représente une carte géographique classique.

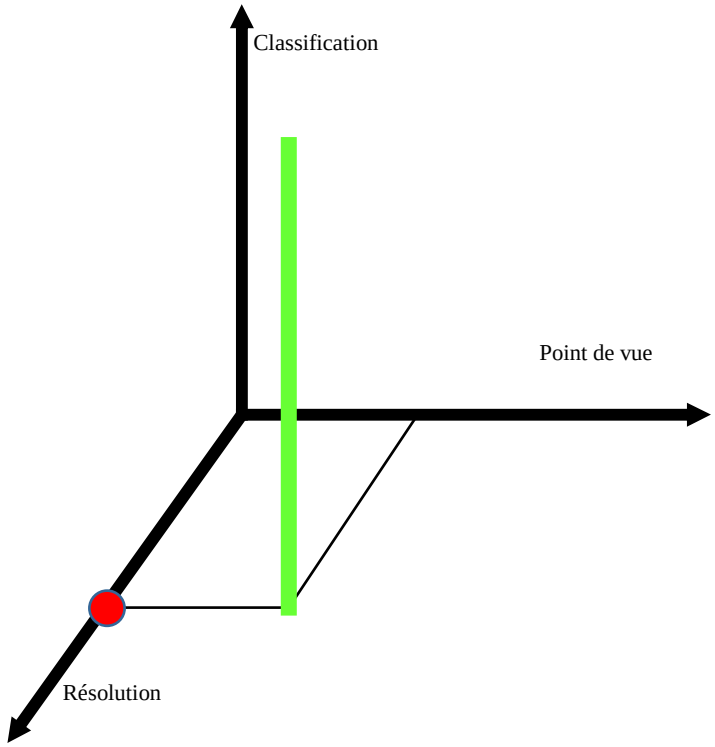


Figure 5
La base de données réduite à une résolution et un point de vue donnés

4.2. Types de données abstraits (Abstract Data Type, ADT)

4.2.1. Types de données abstraits spatiaux

Pour décrire la spatialité des phénomènes du monde réel représentés dans la base de données, MADS fournit un ensemble de types abstraits spatiaux, organisés en une hiérarchie de généralisation (figure 6). A chaque type spatial est associé un ensemble de méthodes permettant de définir et manipuler les instances de ce type, par exemple, la méthode longueur est associée au type abstrait ligne.

Les types abstraits de MADS sont: point, ligne (line), ligne orientée (oriented line), surface simple (simple area), géo simple (simple geo), ensemble de points

(point set), graphe (line set), digraphe (oriented line set), surface complexe (complex area), géo complexe (complex geo) et géo.

Le type abstrait **point** est utilisé pour les phénomènes qui sont représentés par un unique point. Une **ligne** est décrite par un ensemble de points et une ou plusieurs équations linéaires, peuvent ainsi être décrites les lignes droites et courbes, fermées ou non. Une **ligne orientée** est une ligne dont les extrémités sont différenciées : il existe un point de début et un point de fin (la ligne est orientée du point de début vers le point de fin). La ligne orientée est une spécialisation de ligne. Une **surface simple** est une surface qui peut posséder des trous mais pas d'îles (pas d'île extérieure à la surface ni île à l'intérieur d'un trou). **Géo simple** est un type abstrait générique qui généralise les types point, ligne et surface simple (qui forment une partition de géo simple) et est utilisé pour représenter tout type spatial simple. MADS propose ainsi des types abstraits pour décrire les ensembles.

Un **géo complexe** représente un ensemble d'éléments de type hétérogène qui peut donc inclure des points, des lignes, et des surfaces. Les géo complexes peuvent être utilisés, par exemple, pour représenter un ensemble hydrographique composé de rivières (lignes orientées), lacs (surfaces), et de réservoirs (points).

Le type géo Complexe est un type abstrait générique qui généralise les sous-types : **ensemble de points**, le **graphe** (ensemble de lignes), le **digraphe** (ensemble de lignes orientées), et la **surface complexe** (ensemble de surfaces simples).

Le type **ensemble de points** peut, par exemple, être utilisé pour représenter les maisons d'une commune. Un **graphe** (ensemble de lignes) permet par exemple de représenter un réseau de routes. Un **digraphe** (spécialisation de **ensemble de lignes**) est un ensemble de lignes orientées, qui permet par exemple de représenter une rivière et ses affluents. Une **surface complexe** est un ensemble de surfaces simples ne se chevauchant pas (incluant les surfaces avec îles), et est très utile, par exemple, pour représenter les régions administratives.

Pour finir, **géo** est le type abstrait le plus générique, généralisant les types abstraits **géo simple** et **géo complexe**. La sémantique associée à un élément décrit par un **géo** est : « cet élément possède une emprise spatiale ». Il n'informe pas sur le type de l'emprise. La définition du type précis de chaque occurrence se fera lors de sa création.

La spatialité d'un élément peut donc être soit définie précisément (par exemple : **point**, **ligne orientée**), soit être laissée indéterminée (par exemple : **géo**). Le

type spatial des instances peut ainsi être décrit dans le schéma par un type générique, regroupant plusieurs choix possibles (**géo**), le choix précis étant fait lors de l'insertion des valeurs de chaque instance ou groupe d’instances. C'est particulièrement utile pour décrire des phénomènes spatiaux dont les instances peuvent avoir des types différents.

Par exemple, la spatialité des villes, petites et grandes, peut être décrite par des points et des surfaces, le type abstrait utilisé sera alors **géo simple**. Les types abstraits décrits ci-dessus permettent d'offrir une vue discrète de l'espace.

La spatialité d'un type d'objet ou d'association est décrite par un attribut prédéfini, géométrie, dont le domaine de valeur est un type abstrait spatial. Les types d'objet et d'association spatiaux portent l'icone du type abstrait spatial de leur géométrie à droite de leur nom dans le diagramme. Un attribut spatial peut être :

- un attribut simple (avec des valeurs atomiques) ou complexe (c’est à dire composé d’autres attributs simples ou complexes) ;
- monovalué (avec une seule valeur) ou multivalué (avec un ensemble de valeurs) ;
- obligatoire (avec une valeur dans chaque objet) ou facultatif (avec une valeur dans certains objets et pas de valeur dans d’autres).

La hiérarchie complète pour les types abstraits spatiaux est décrite dans la figure 6. Celle-ci montre aussi quelle est l'icone associée à chaque type de donnée abstrait.

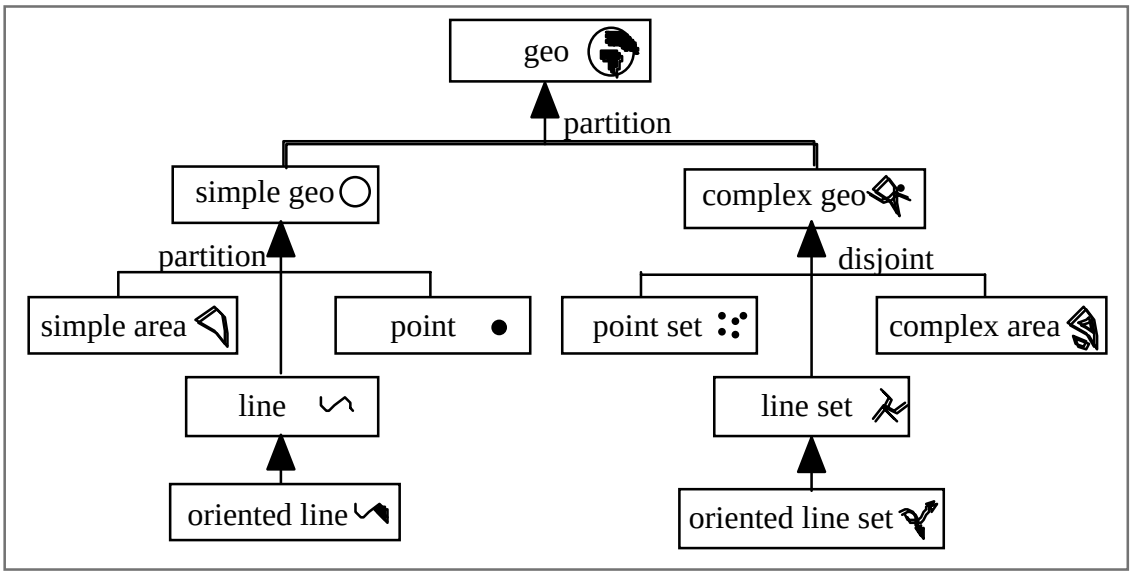


Figure 6
Hiérarchie MADS pour les types abstraits spatiaux

4.2.2. Types de données abstraits temporels

La connaissance de l'évolution temporelle des données, en particulier spatiales, est souvent indispensable pour comprendre la dynamique des phénomènes du monde réel. C'est pourquoi le modèle MADS offre un jeu de concepts pour décrire l'historique dans le passé, et l'évolution prévue dans le futur des objets, attributs et associations. On ne s'intéresse ici qu'au temps dit de validité, décrivant la période de validité de l'information du point de vue de l'application.

MADS définit les types de données abstraits temporels suivants : l'**instant**, l'**intervalle** (interval), **ensemble d'instants** (instant set), **ensemble d'intervalles** (interval set), les types abstraits génériques **temps simple** (simple time), **temps complexe** (complex time) et **élément temporel** (temporal element). Un **instant** fait référence à un moment précis dans le temps. Un **intervalle** est un ensemble d'instants contigus. Un **ensemble d'intervalles** ou

un **ensemble d'instants** définit respectivement un ensemble disjoint d'intervalles ou d'instants. Un élément temporel est un ensemble contenant des **instants disjoints**, **intervalles disjoints**, **instants** ou **intervalles**. C'est le type abstrait le plus générique, généralisant les types abstraits **temps complexe** et **temps simple**. Les types d'objet et d'association temporels portent l'icone du type abstrait temporel de leur géométrie dans le diagramme.

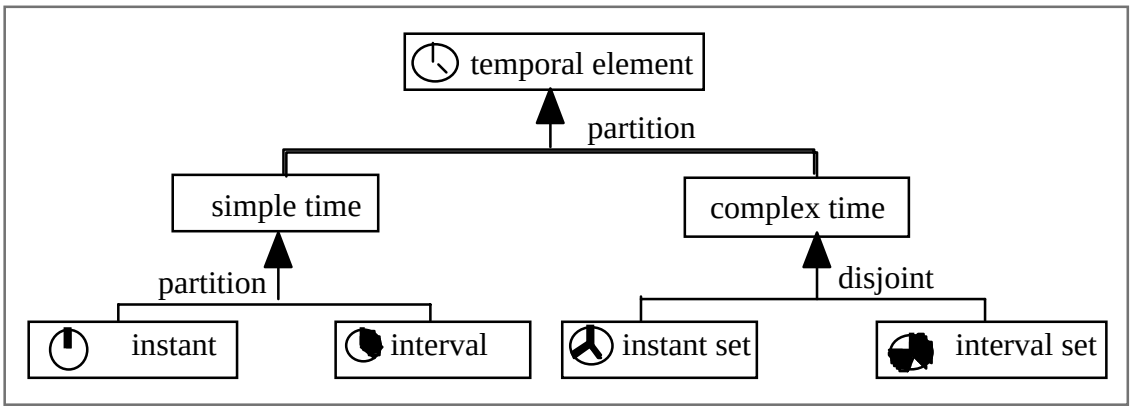


Figure 7
Hiérarchie MADS pour les types abstraits temporels

Le cycle de vie d'un objet permet de modifier ses caractéristiques : pendant qu'un objet est actif, l'objet est pleinement utilisable (lectures, écritures, mises à jour, lier/délier à des associations,...). Pendant qu'il est suspendu, il n'est plus modifiable; seules les consultations (et le changement d'état vers actif ou inactif) sont possibles. Enfin, pendant qu'il est inactif ou pas encore créé (deux états symétriques), il n'est plus du tout accessible. Pour y accéder, l'utilisateur doit émettre une requête spécifiant un temps de validité pour la requête appartenant à une période d'activité de l'objet (voire à une période de suspension dans le cas d'une requête de consultation pure).

Un type d'objet temporel peut avoir des attributs non-temporels et temporels. Il n'y a pas en MADS de contrainte par défaut entre la durée de validité d'un attribut temporel et le cycle de vie de l'objet auquel il appartient, ni entre celle d'un attribut complexe et celle de ses composants.

Cependant, si l'application en a besoin, des contraintes d'intégrité temporelles ad hoc peuvent être spécifiées par le concepteur, tel que la validité d'un composant doit toujours être incluse dans la validité de son composé.

De la même façon, déclarer un type d'association temporelle permet de garder trace du cycle de vie de ses instances. Celles-ci peuvent être créées, suspendues, réactivées et détruites. Toujours dans l'optique de ne pas restreindre arbitrairement le pouvoir de modélisation des concepteurs, en MADS, un type d'association temporelle peut relier des types d'objets temporels ou non-temporels, et vice versa. Comme pour les attributs, les cardinalités reliant les types d'objet à un type d'association temporelle sont interprétées comme instantanées, définissant le nombre minimal et maximal d'occurrences de l'association pouvant lier un objet à un instant donné.

4.3. Caractéristiques principales de l'algèbre MADS

MADS est un modèle dans lequel le *type objet* et le *type association* sont au même titre des concepts fondamentaux. Pour respecter cette dualité, l'algèbre MADS est une algèbre multi-sorte, où les opérateurs manipulent aussi bien des types d'objet que des types d'association.

Pour assurer la puissance expressive du langage, l'algèbre MADS est une algèbre fermée : le résultat de tout opérateur peut être réutilisé comme opérande d'un autre opérateur. Concrètement, les résultats des opérateurs de l'algèbre peuvent être des types d'objet, des types d'association ou les deux. Ceci est indispensable pour permettre d'écrire des expressions comprenant n'importe quelle séquence d'opérateurs.

Toutes les propriétés du résultat (attributs, méthodes, rôles, liens) sont déduites à partir des propriétés correspondantes des opérandes suivant les opérations spécifiées dans l'expression algébrique. Cette déduction des propriétés du résultat à partir des propriétés d'un autre type d'objet ou d'association est possible grâce à l'héritage automatique et implicite par l'intermédiaire des liens *is-a*, qui s'applique à tous les types des propriétés et à la dérivation des valeurs d'attributs par la définition explicite de formules de dérivation. Ces formules peuvent exprimer une navigation dans la base de données, à travers tout type de lien, afin d'atteindre la valeur source de la dérivation.

Les structures de données MADS sont assimilées à une structure de graphe, ce qui facilite la navigation dans la base de données. La navigation a pour fonction de permettre de vérifier l'existence d'objets ou d'associations reliés par un quelconque chemin dans le schéma à l'instance opérande sélectionnée, d'exprimer des prédicats de sélection sur une opérande qui peuvent impliquer

des tests sur des valeurs d'attributs et de permettre la création d'une formule de dérivation.

Pour désigner ces attributs, objets et associations recherchés lors de la navigation, MADS utilise la notation pointée. Dans cette notation, faire suivre le nom d'un type d'objet par un nom de rôle permet de traverser le rôle pour atteindre l'association correspondante ou de passer à travers une association pour atteindre un objet lié. Les liens d'attribut permettent de descendre (ou remonter) dans la structure des objets, associations et attributs complexes. Ainsi il est possible d'atteindre l'élément visé en suivant des liens de n'importe quel type.

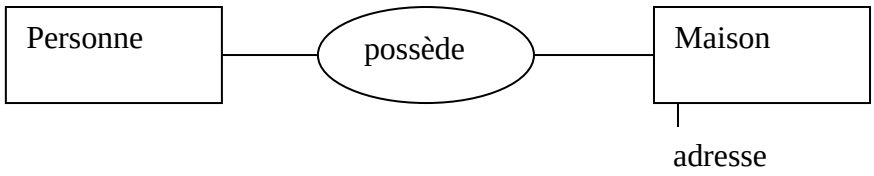


Figure 8
Exemple de notation pointée

Par exemple, si Jean est une personne et que le type *Personne* est lié par un type d'association *possède* au type *Maison* dont on cherche à atteindre l'adresse, la notation Jean.possède.Maison.adresse désigne un chemin qui identifie l'adresse de la maison que possède Jean.

4.4. Présentation du logiciel MADS

Cette section présente une vue d'ensemble du logiciel MADS. Voici les principaux éléments de ce logiciel :

- Fenêtre Messages : le logiciel MADS affiche dans cette partie les messages d'erreur ou de confirmation. Ainsi, lors de la vérification d'un schéma, on aura la confirmation de la validité du schéma ou les éventuelles erreurs commises.

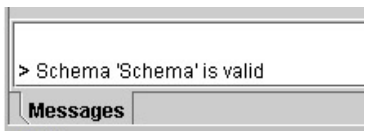


Figure 9
Fenêtre Messages

- Dictionnaire de données (data dictionary) : cette fenêtre montre sous forme d'arbre tous les éléments du schéma créé ainsi que les valeurs et propriétés de ces éléments. L'utilisateur peut directement créer ou modifier les valeurs des éléments et ajouter des commentaires dans cette fenêtre.

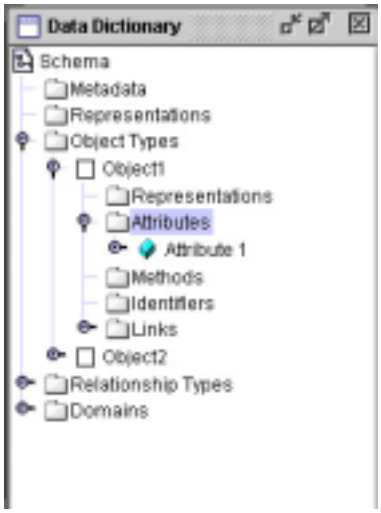


Figure 10
Dictionnaire de données

- Overview : grâce à cette fenêtre, on a une vue d'ensemble du schéma créé, car la fenêtre permettant la construction du schéma n'offre qu'une vue locale permettant la construction.

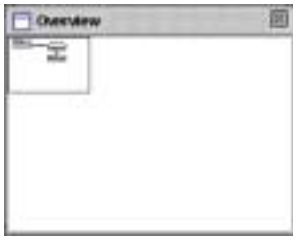


Figure 10
Vue d'ensemble du schéma

- Schema : il s’agit de l’éditeur de schémas où aura lieu la construction intuitive du schéma grâce aux outils visuels.

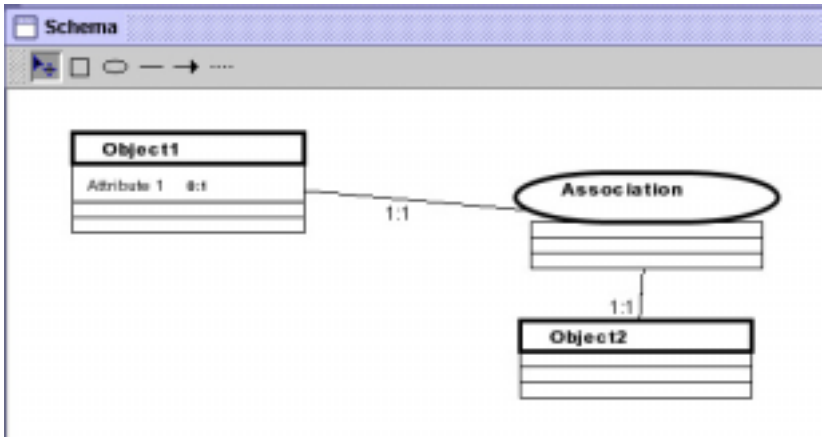


Figure 11
Fenêtre de composition de schéma

- La barre d’outils de la fenêtre schéma possède de gauche à droite les boutons permettant la création d’objets, relations, role, liens is-a et may-be. Rappelons tout d'abord que le lien *is-a* est traditionnellement utilisé dans les bases de données orientées-objets pour prendre en compte les besoins de représentation multiple. Il permet de relier deux types d'objet, un sous-type et un sur-type, pour exprimer que chaque objet dans la population du sous-type représente le même phénomène du monde réel qu'un autre objet dans la population du sur-type. Cette multiplicité de classification peut s'appliquer indépendamment des aspects point de vue et résolution. MADS offre également le lien *may-be*, qui permet d'exprimer que les populations de deux types d'objet peuvent être en intersection. Une instance de lien *may-be* lie deux représentations différentes du même objet du monde réel à deux instances liées qui ont le même identificateur d'instance (oid).

Figure 12
Barre d’outils de la fenêtre schéma

- La barre d’outils supérieure permet grâce aux différents boutons :

  - de créer un nouveau schéma ;
  - d’ouvrir un schéma ;
  - de sauver un schéma ;
  - d’imprimer un schéma ;
  - d’effacer un élément sélectionné ;
  - de valider un schéma ;
  - de donner la version de MADS utilisée.

Figure 13
Barre d’outils principale

La figure 14 offre une vue d’ensemble du logiciel MADS.

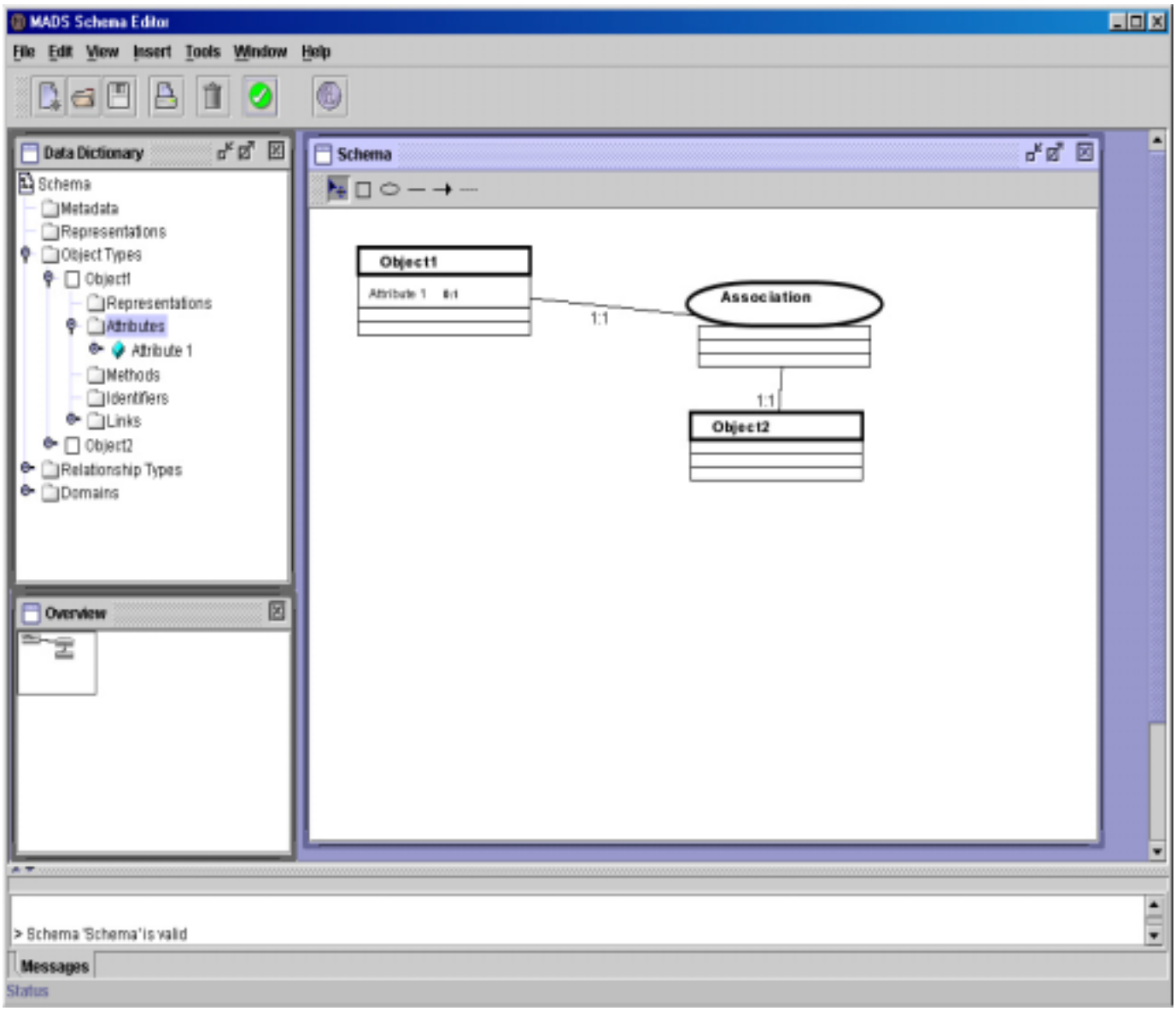


Figure 14
Vue d’ensemble du logiciel MADS

4.5. Structure des données

Une structure de données a été créée pour répondre aux besoins de l'éditeur de schéma MADS. Cette structure de données est indépendante de l'interface graphique.

L'éditeur de schéma pour le projet MURMUR est un éditeur graphique et intuitif. Cette interface est créée afin de permettre l'utilisation de toutes les caractéristiques de la syntaxe MADS. Une des notions fondamentales est que seul le schéma lui-même est important, et non sa représentation. Cette décision a mené au développement séparé de la structure de données et de l'interface. De cette manière, une structure proche de la syntaxe a été développée sans interférence de la représentation graphique.

La structure utilisée est une structure en arbre. Une validation de schéma est nécessaire à l'intérieur de la structure de données pour assurer la cohérence de ce qui est créé avec la syntaxe. L'approche suivie pour la validation est une approche « top to bottom », un schéma n'est valide que si ses composants sont valides.

De tous les éléments à l'intérieur du schéma, une seule région est complètement en dehors de la multi-représentation, c'est la partie domaine. Même si les domaines ont des attributs et méthodes, comme les types relations ou objets, leurs définitions sont bien plus simples. La structure de données a été divisée en deux sous-projets : d'un côté le domaine, de l'autre le reste de la structure (le schéma et autres éléments).

De plus, la structure est fortement liée. Elle est en fait doublement liée ce qui permet de la parcourir du haut vers le bas et d'un point à un autre, ce qui est très utile pour trouver les caractéristiques d'un attribut dérivé.

4.5.1. Analyse détaillée de la structure

Le schéma est le container de tous les éléments, même du domaine. C'est le **sommet de la structure**. Le schéma maintient les meta-data telles que les dates de modification ou création, la personnalisation faite par l'utilisateur.

Au **deuxième niveau**, on trouve les composants du schéma contenant les représentations, les types objets et types relations.

La notion de représentation est définie par le point de vue et la résolution. Un seul de ces paramètres peut être nul. La validation est faite en vérifiant qu'il n'y ait pas plus d'un paramètre de représentation vide et qu'il n'y ait pas deux représentations qui regroupent le même point de vue et résolution.

Les types objets et relations ont des caractéristiques similaires. Les deux ont des attributs et méthodes et peuvent être liés ensemble. Ils utilisent aussi la notion d'identificateur, qui est semblable à la notion de clé dans le modèle relationnel de base de données. Comme différence, on remarquera que les objets doivent avoir un nom unique à l'intérieur du schéma tandis que les noms des relations peuvent être répétés.

Pour la validation des objets, on vérifiera que lors d'un changement de nom ou lors de la création d'un objet, il n'y ait pas deux noms identiques. Pour les relations, la validation vérifiera qu'il y ait au moins deux objets associés à une relation.

Le **troisième niveau** est composé de deux parties, la première composée des attributs et méthodes et la deuxième concerne les liens entre types.

Les attributs et méthodes ont en commun les notions de propriété, ils appartiennent à un objet ou à une relation.

Les attributs sont séparés en trois catégories :

- simple : l'attribut est limité à un domaine ;
- complexe : l'attribut est composé d'autres attributs sans restriction sur leur genre ;
- dérivé : l'attribut est défini en faisant référence à d'autres attributs.

Tous les types d'attributs ont une cardinalité.

Les méthodes sont définies par un code et une signature, elle-même composée d'un nom et de paramètres. Ces paramètres ne supportent pas la multi-représentation. Les méthodes possèdent des paramètres sortants et entrants, les paramètres sortants déterminent le type sortant (le return) de la méthode.

Pour la validation d'un attribut dérivé, on vérifiera si la référence existe et peut être atteinte. L'attribut référé doit être dans un chemin pointé qui le connecte à l'attribut dérivé. Pour les méthodes, la validation est faite en vérifiant qu'elles n'ont qu'un et un seul paramètre sortant.

Il existe trois types de liens :

- *is-a*, orienté fils à parent, entre deux mêmes types (objet/objet ou relation/relation) ;
- *may-be*, symétrique (non orienté), entre deux mêmes types (objet/objet ou relation/relation) ;
- *role*, orienté relation à objet, entre deux types différents (objet/relation), notion de cardinalité.

Tous les types de liens lient deux types (objet/relation, objet/objet ou relation/relation). La validation vérifie les types liés par un lien.

4.5.2. Implémentation de la structure

Seules la description de l'implémentation du premier niveau, celui-ci contenant tous les éléments permettant une vue d'ensemble d'un schéma et la description du niveau trois traitant les attributs, qui sont l'objet de ce travail, seront développées ici.

Le langage de programmation utilisé pour implémenter MADS est Java.

Pour le premier niveau, le schéma est codé dans la classe TSchema. Cette classe utilise des *linked list* Java pour stocker tous les éléments d'un schéma :

représentations, domaines, objets et relations. Elle stocke aussi le schéma prédéfini. Quand un nouveau schéma est créé, ce type prédéfini est utilisé pour l'initialiser. Les méta-data sont fournies par la classe TMetadata, qui permet donc la définition des propriétés personnalisées et le stockage de l'historique du schéma.

La figure 15 montre le diagramme des classes de TSchéma.

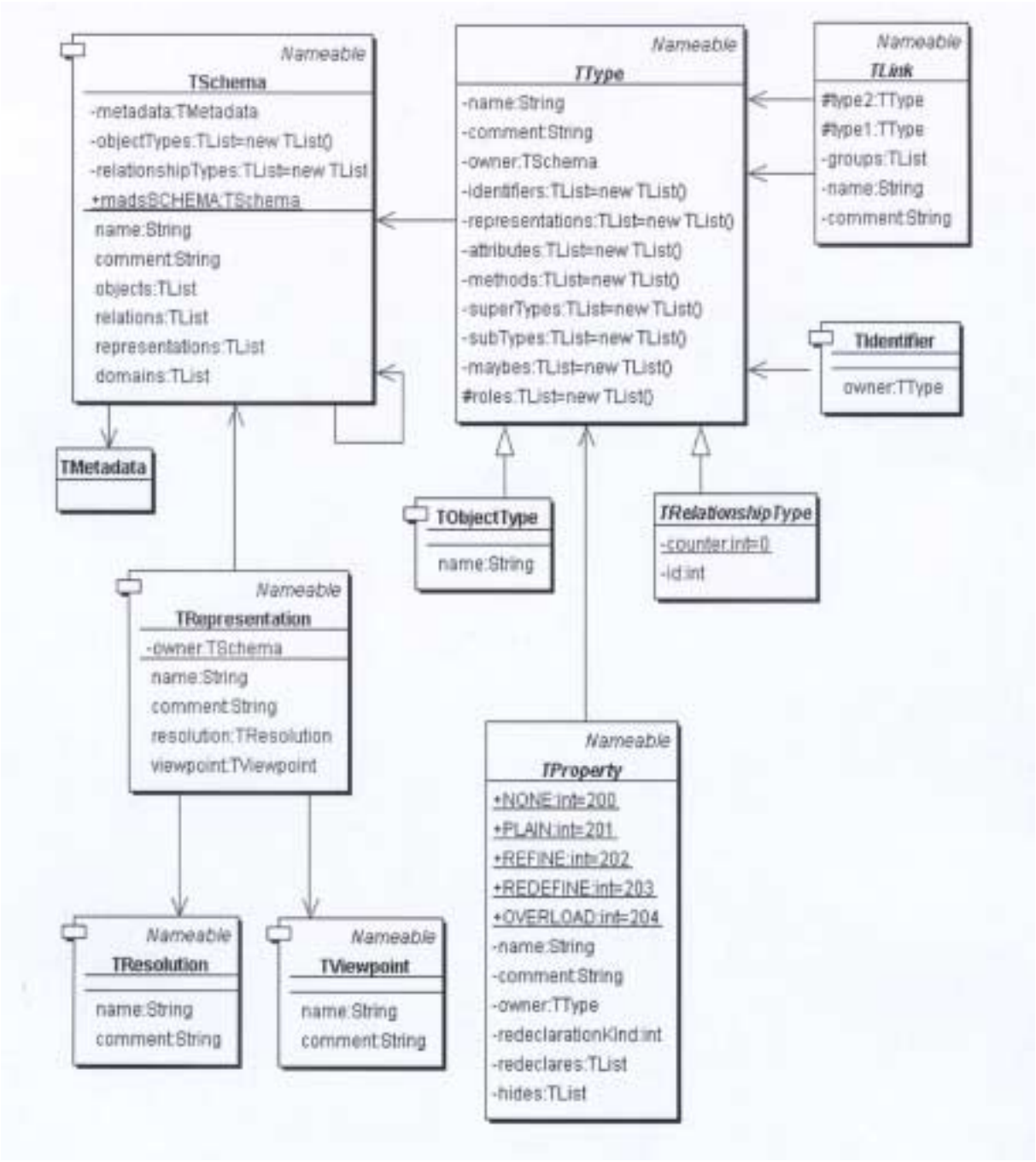


Figure 15
Diagramme des classes de TSchema

Pour le niveau trois, les propriétés communes aux méthodes et attributs sont développées à l'intérieur de la classe TProperty. Pour les attributs, une classe TAttribute est définie, elle hérite de TProperty et contient une liste de TAttributeDefinition, liée aux représentations. Cette définition contient toutes les propriétés communes aux trois types d'attributs. Viennent ensuite les classes TAttributeSimple, TAttributeComplex et TAttributeDerived qui permettent d'ajouter les propriétés propres à chaque type. Pour la notion de cardinalité, une classe TCardinality est définie. Pour fournir la variance, une classe TVariance est créée, elle est subdivisée en TSpatialVariance et TTimeVariance. La classe TMethod contient toutes les caractéristiques propres aux méthodes. TMethodParameter est la classe qui permet la gestion des paramètres.

La figure 16 montre le diagramme des classes de TProperty.

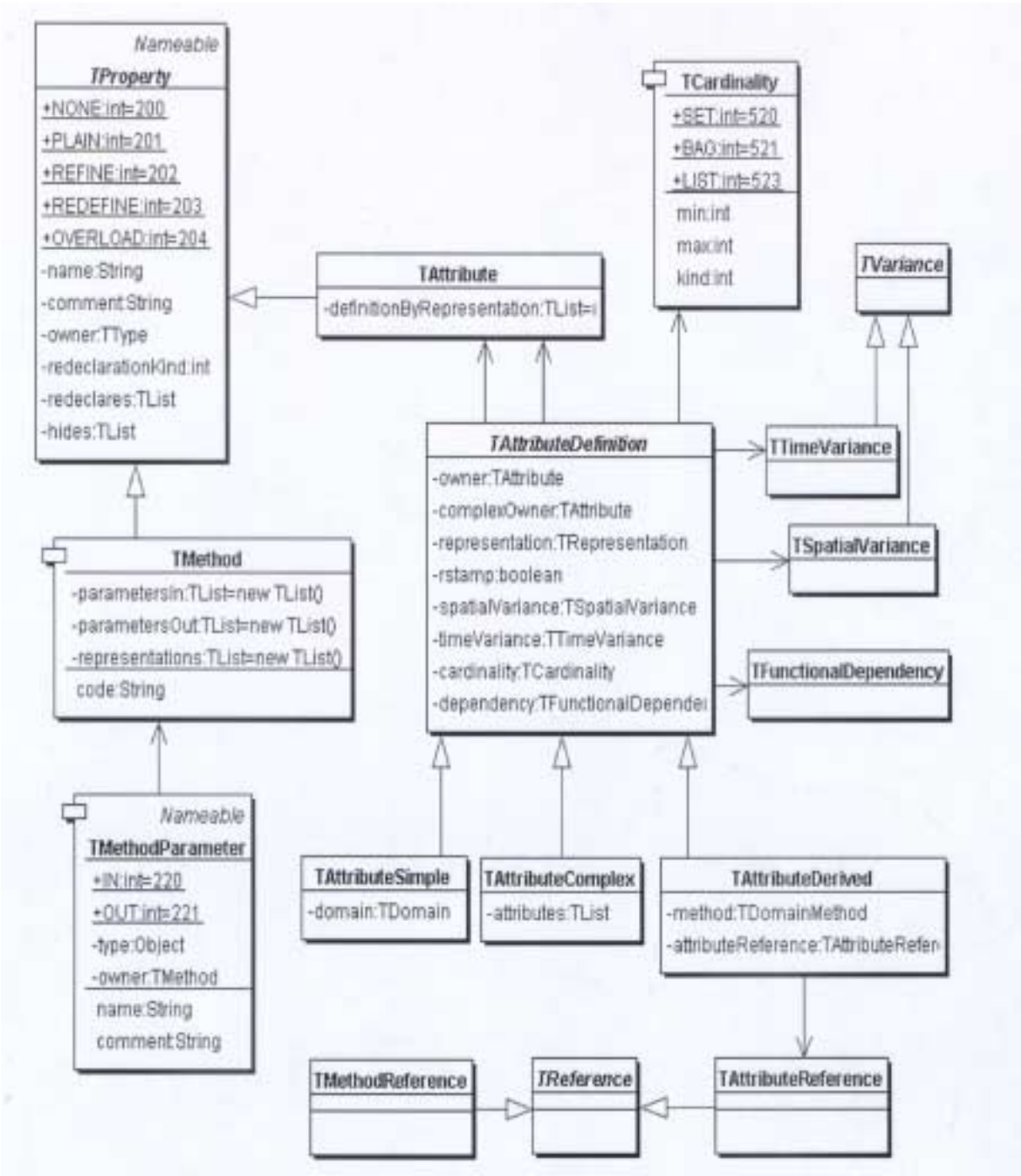


Figure 16
Diagramme des classes de TProperty

5. Attributs dérivés

MADS est un modèle conceptuel objet/association. Un objet représente une entité du monde réel. Un type objet décrit un ensemble d'objets avec des structures et comportements similaires. Une relation est un lien entre deux ou plusieurs objets, où chaque objet joue un rôle donné. Un type relation décrit un ensemble de liens avec des caractéristiques similaires (liant des objets du même type, avec les mêmes rôles et des propriétés similaires). Les relations sont une partie essentielle de la modélisation conceptuelle. Les objets et relations peuvent être localisés dans l'espace et le temps.

Chaque objet/association a des caractéristiques que l'on appelle attributs. Un attribut représente une propriété du monde réel qui caractérise le propriétaire de cet attribut. La valeur d'un attribut peut être calculée, c'est-à-dire inférée automatiquement par le système, à partir des valeurs d'autre(s) attribut(s) appartenant au même objet ou à la même association, ou à d'autres objets ou associations qui lui sont liés. Ces attributs sont appelés attributs dérivés.

Les formules permettant de créer ces attributs dérivés, peuvent impliquer une navigation dans la base de données, à travers tout type de lien, afin d'atteindre la valeur source de la dérivation. Pour désigner ces attributs, objets et associations, MADS utilise la notation pointée. Dans cette notation, faire suivre le nom d'un type d'objet par un nom de *role* permet de traverser le *role* pour atteindre l'association correspondante ou de passer à travers une association pour atteindre un objet lié. Les liens d'attribut permettent de descendre (ou remonter) dans la structure des objets, associations et attributs complexes. Ainsi, il est possible d'atteindre l'élément visé en suivant des liens de n'importe quel type.

5.1. Les attributs dérivés dans le logiciel MADS

5.1.1. Accès aux attributs dérivés

Pour avoir accès au menu attribut dérivé, il faut accéder au menu permettant de changer le type d'un attribut. Pour cela, on doit :

1. faire un click droit sur un attribut soit dans le dictionnaire de données soit dans l'éditeur de schéma,
2. sélectionner « ouvrir propriétés » (figure 17),
3. sélectionner le bouton d'options derived (figure 18),
4. le bouton Derivation Formula et le champ qui contiendra la formule de dérivation deviennent alors accessibles (figure 18). En poussant sur le bouton, on rentre dans le menu permettant la création de la formule de dérivation.

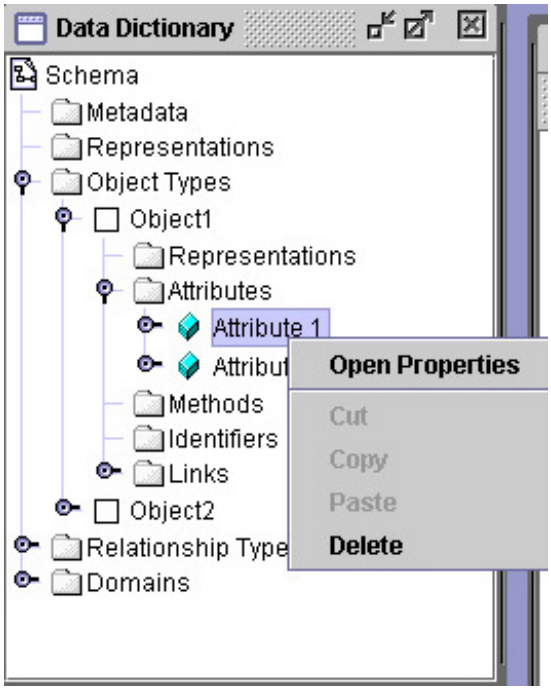


Figure 17
Entrée dans le menu « propriétés des attributs »

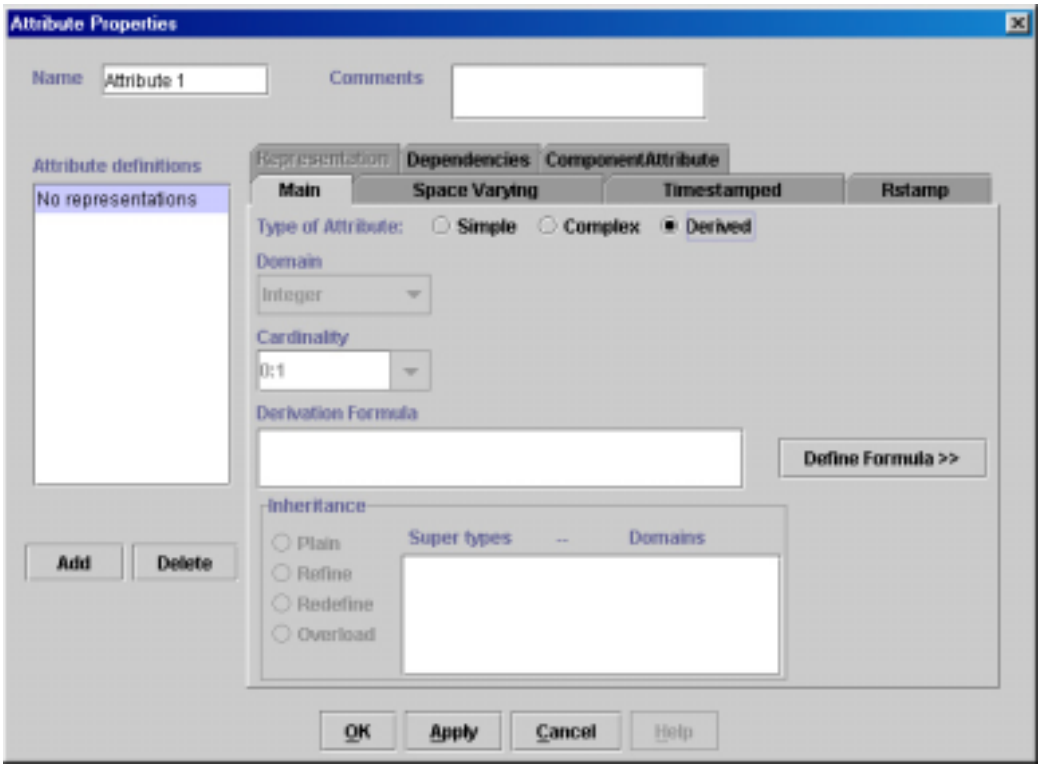


Figure 18
Choix du type attribut dérivé

5.1.2. Présentation de l'interface graphique Derivation Formula

La fenêtre de l'interface graphique Derivation Formula est composée de trois parties :

- la partie supérieure contenant les boutons d'options indiquant le type de l'élément contenant l'attribut référé ;
- la partie centrale permettant la construction de la formule de dérivation ;
- la partie inférieure contenant les boutons Ok, Apply, Cancel et Help. Les boutons Ok et Apply servent à valider la formule.

5.2. Les différentes possibilités

5.2.1. Attribut dérivé appartenant à un objet

Si l'attribut dérivé appartient à un objet alors l'attribut référé peut se trouver :

- sur le même objet que l'attribut dérivé ;
- sur un autre objet ;
- sur une relation.

Les boutons d'options permettant ces différents choix ne sont activés que si les conditions permettant d'avoir ce cas-là sont réunies.

La première condition est élémentaire, il faut que l'élément possédant l'élément dérivé soit un objet.

Pour que le bouton d'options (radio button) permettant d'atteindre un attribut référé sur le même objet soit activable, il faut au moins deux attributs (un dérivé et un dérivant) sur cet objet.

Pour pouvoir atteindre une référence sur un autre objet, il faut qu'il y ait au moins un attribut sur un autre objet, dans ce cas le bouton d'options permettant ce choix est activé.

Le bouton d'options permettant d'avoir un attribut référé sur une relation n'est activé que s'il y a au moins un attribut sur une relation.

5.2.2. Attribut dérivé appartenant à une relation

Si l'attribut dérivé appartient à une relation, alors l'attribut référé peut se trouver :

- sur la même relation que l'attribut dérivé ;
- sur un objet.

Les boutons d'options permettant ces différents choix ne sont activés que si les conditions permettant d'avoir ce cas sont réunies.

La première condition est élémentaire, il faut que l'élément possédant l'élément dérivé soit une relation.

Pour que le bouton d'options permettant d'atteindre un attribut référé sur la même relation soit activable, il faut au moins deux attributs (un dérivé et un dérivant) sur cette relation.

Pour pouvoir atteindre une référence sur un objet, il faut qu'il y ait au moins un attribut sur un objet, dans ce cas le bouton d'options permettant ce choix est activé.

5.3. Création de la formule

L'interface permettant la construction de la formule est différent pour chacun des cas présentés ci-dessus. Les fonctions possibles pour faire la formule sont Min, Max, Sum, Avg, Count, Union.

Les sections suivantes montrent l'allure de l'interface permettant la création de la formule de dérivation dans les différents cas possibles.

5.3.1. Attribut dérivé appartenant à un objet

5.3.1.1. L'attribut référé sur le même objet que l'attribut dérivé

Dans le cas où l'attribut dérivé et l'attribut référé se trouvent sur le même objet, les seules choses à faire sont de choisir l'attribut référé dans le menu déroulant (combo box) permettant ce choix et de selectionner la fonction agissant dans la formule de dérivation (figure 19).

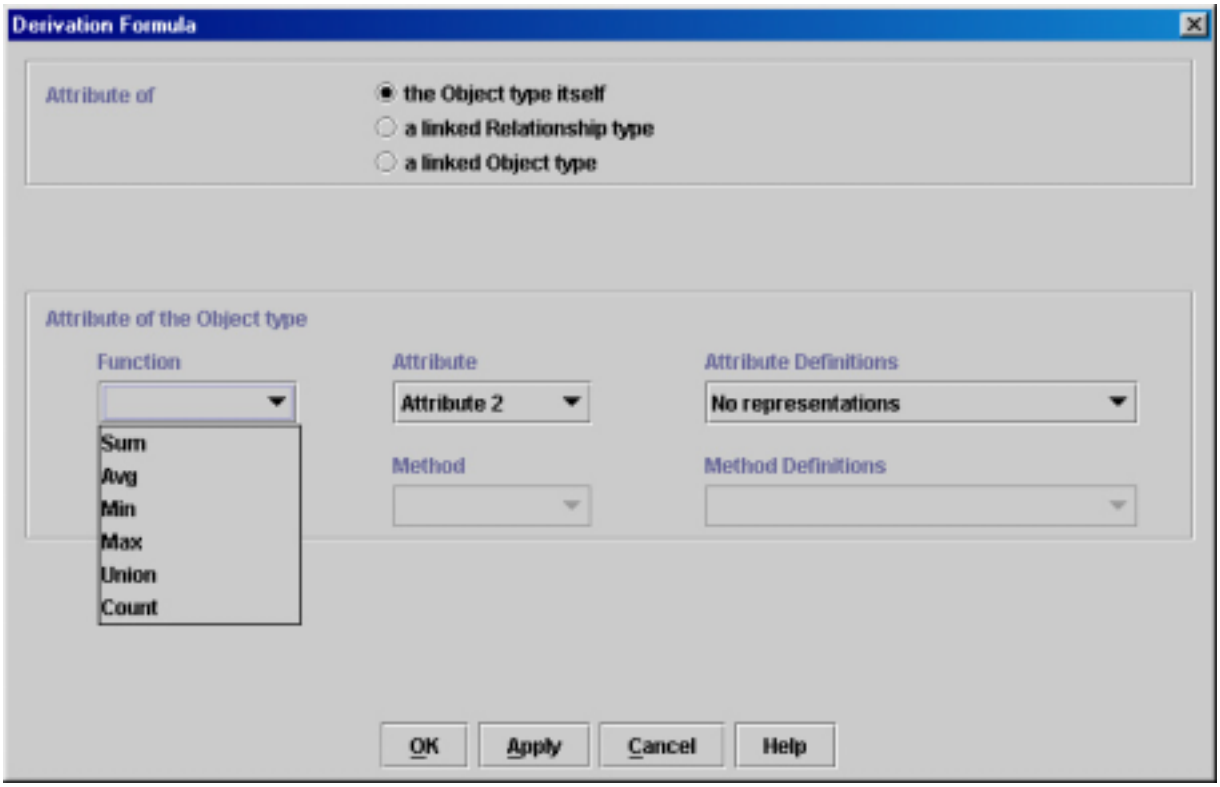


Figure 19
Attribut dérivé dont la référence se trouve dans le même objet

5.3.1.2. Attribut référé appartenant à un autre objet

Dans le cas où l'attribut référé appartient à un autre objet, il faut choisir la fonction. Grâce au bouton *expand*, on peut parcourir la structure pour obtenir l'attribut visé. Le bouton *expand* n'est visible que s'il existe plus de deux objets liés par des relations. Pour rappel, MADS utilise la notion de chemin pointé pour atteindre un attribut dans sa structure. Il faut donc créer ce chemin pointé pour pouvoir créer la formule de dérivation.

Le parcours du chemin se fait, dans notre cas, par un ensemble <objet1-relation-objet2>. Le menu déroulant *relation* contient toutes les relations qui sont directement liées (qui possèdent un lien entre objet1 et relation) à objet1, le menu déroulant *object type* contient tous les objet2 qui sont directement liés par un lien à la relation du triplet <objet1-relation-objet2>, le menu déroulant *Object type attribute* contient tous les attributs de objet2. Les liens permettant de revenir en arrière dans le chemin sont éliminés des différents menus déroulants.

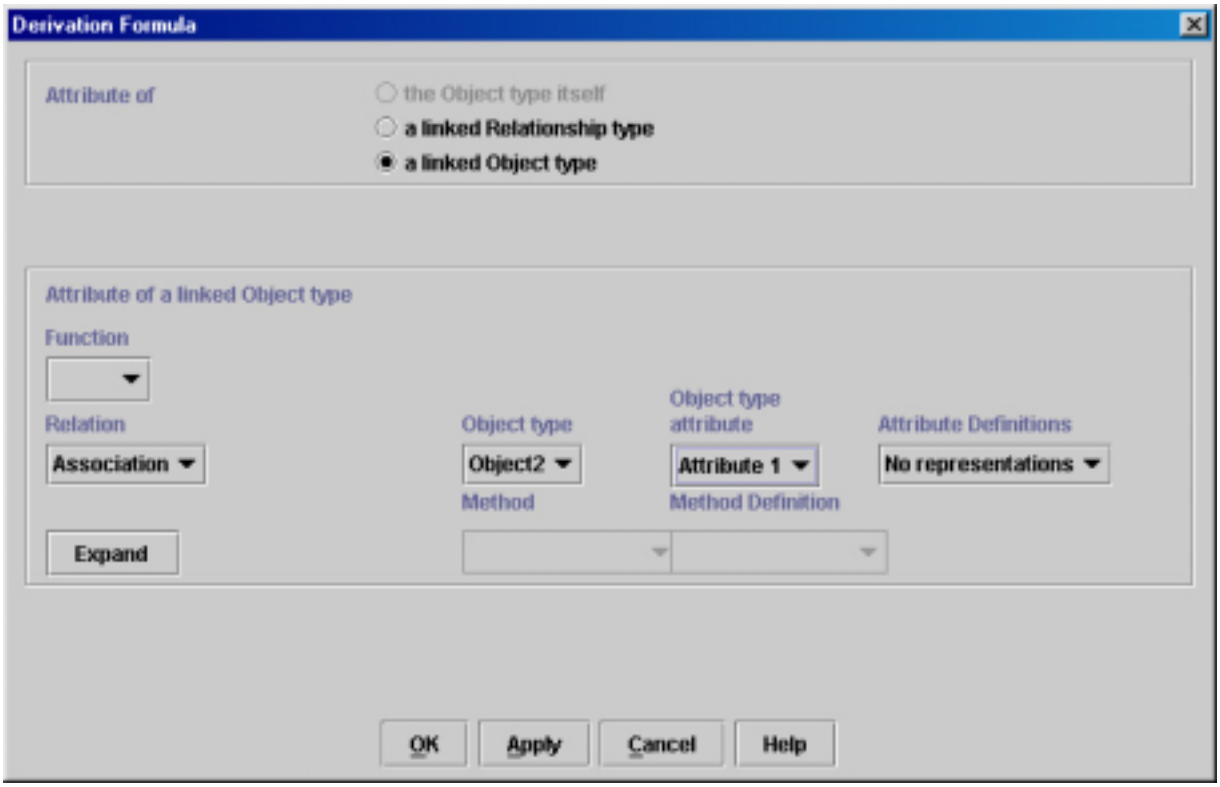


Figure 20
Attribut dérivé dont la référence se trouve dans un autre objet

Pour expliquer la manière dont la structure est parcourue pour atteindre l'attribut nécessaire à la formule, c'est-à-dire la manière dont le chemin pointé va être créé, on va se baser sur l'exemple de la figure 21.

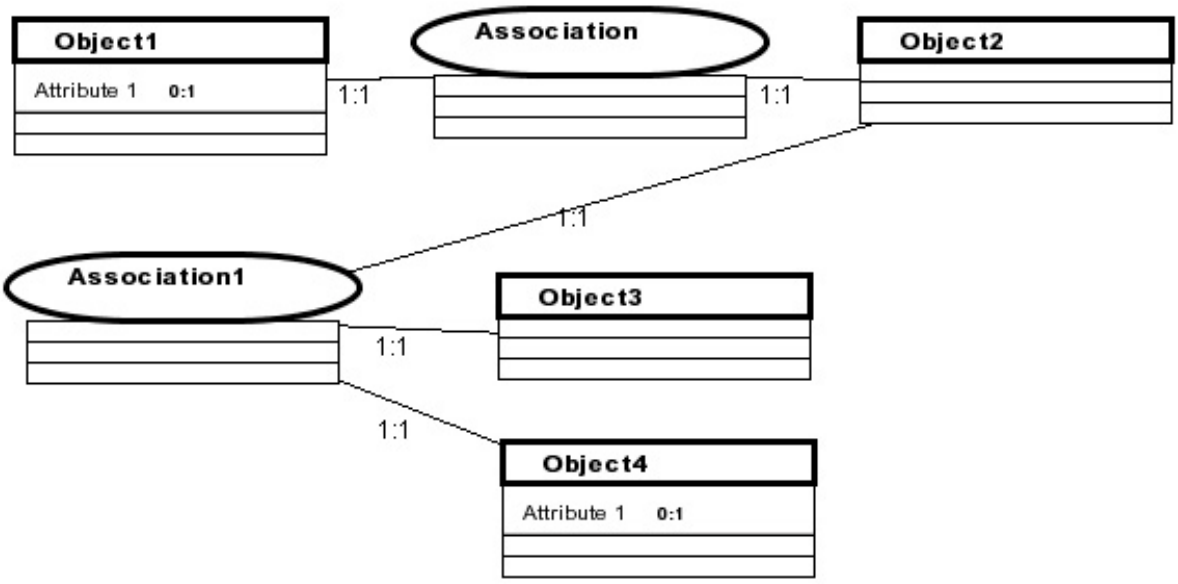


Figure 21
Exemple de schéma

Dans cet l'exemple, nous sommes dans le cas où l'attribut dérivé appartient à un objet, cet attribut est Attribute1 dans Object1 et on veut atteindre Attribute1 dans Objet4.

Au départ, le premier triplet est <Object1-Association-Object2> et le menu déroulant *relation* contient Association, le menu déroulant *object type* contient Object2 et le menu déroulant *object type attribute* est vide. A ce moment, le bouton *expand* est encore visible puisqu'il y a encore un objet possédant un attribut dans le chemin que l'on peut parcourir à partir de Object2 et qu'il existe encore des triplets possibles à partir de Object2.

A ce stade, si l'utilisateur appuie par erreur sur le bouton Ok qui valide la fonction, étant donné qu'il n'y a pas d'attribut dans Object2, on ne peut pas construire de formule. Un message d'erreur est donc envoyé à l'utilisateur. Pour continuer de parcourir son chemin, l'utilisateur doit appuyer sur le bouton *Expand*, le nouveau triplet est <Object2-Association1-Object*> avec * représentant 3 ou 4.

Etant donné qu'il n'est plus possible de créer un triplet <objet-relation-objet>, ce qui signifie que l'on est en fin de chemin, le bouton *Expand* disparaît. Cette fois, le menu déroulant *relation* contient Association1, le menu déroulant *object type* contient Object3 et Object4 et le menu déroulant *object type attribute* est vide ou contient attribute1 en fonction que l'on choisisse dans le menu déroulant objet Object3 ou Object4 respectivement.

L'utilisateur choisira l'Object4 dans le menu déroulant o*bject type* et l'Attribut1 sera d'office choisi vu que c'est le seul attribut. S'il y a plusieurs attributs l'utilisateur choisit dans le menu déroulant l'attribut qu'il désire. Si l'utilisateur avait choisi l'Object3, étant donné qu'il n'y a pas d'attribut dans Object3, on ne pourrait pas construire de formule, donc un message d'erreur serait envoyé à l'utilisateur.

Pour valider sa formule, l'utilisateur appuie sur le bouton Ok, celle-ci est envoyée vers la zone graphique Derivation Formula du menu « attribut properties » (figure 18).

Si l'utilisateur veut la plus grande valeur, la formule crée dans cet exemple sera : Max(Association.Object2.Association1.Object4.Attribute1).

5.3.1.3. Attribut référé appartenant à une relation

Dans le cas où l'attribut référé appartient à une relation, il faut choisir la fonction. Grâce au bouton *expand*, on peut parcourir la structure pour obtenir l'attribut référé. Le bouton *expand* n'est visible que s'il existe au moins deux associations dont une avec un attribut. Il faut aussi construire un chemin pointé pour atteindre l'attribut référé.

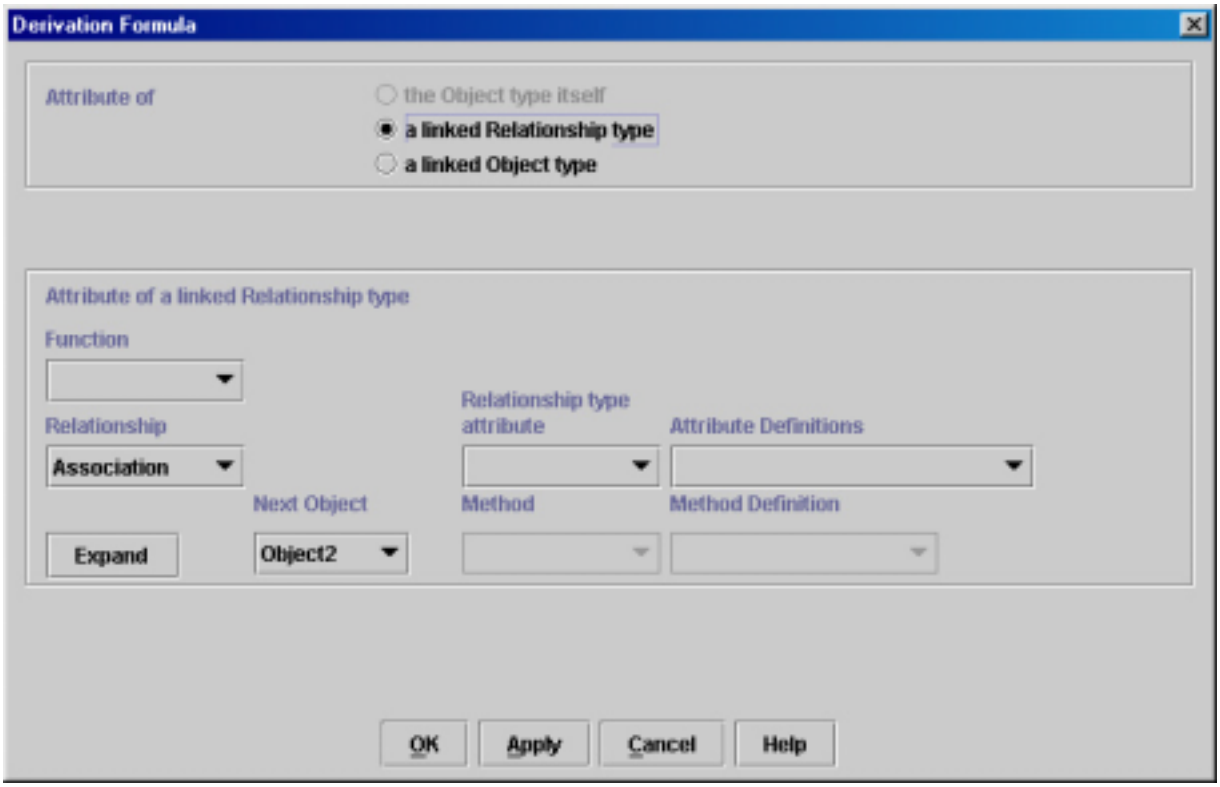


Figure 22
Attribut dérivé appartenant à un objet dont la référence se trouve dans une relation

Le principe de création de chemin est le même mais ici on travaille par couple <objet-relation>. Le menu déroulant *Relationship* contient les relations et le menu déroulant *Relationship type attribute* contient les attributs de ces relations. Le bouton e*xpand* devient invisible si on n'a plus de paires <objet-relation>, on est alors en fin de chemin et il est impossible d'aller plus loin. De même, lors de la validation, on verifie qu'il y ait au moins un attribut dans le menu déroulant *Relationship type attribute.*

Le menu déroulant *Next Object* donne la liste de tous les objets liés à la relation du couple <objet-relation> à l'exception de objet afin de ne pas revenir sur ses pas dans le chemin. Lorsqu'on appuie sur e*xpand*, le couple <objet-relation>

sera en fait l'objet sélectionné dans le menu déroulant Next Object et les relations qui lui sont liées.

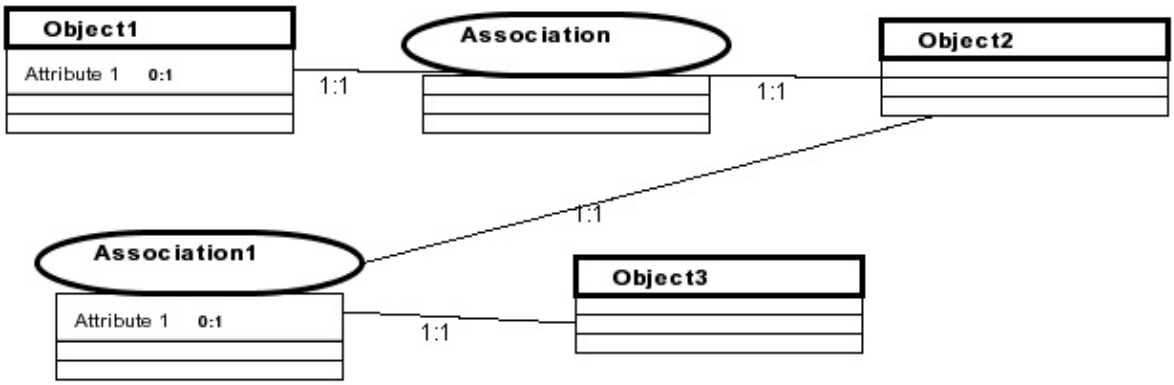


Figure 23
Exemple de schéma

En prenant la figure 23, l'attribut dérivé est Attribute1 qui se trouve dans Object1. Le menu déroulant *Relationship* ne contient que Association car c'est la seule relation liée à Object1. Le menu déroulant *Next Object* ne contient que Object2 car c'est le seul objet lié à Association (on ne met pas Object1 pour éviter de retourner en arrière dans le chemin). L'utilisateur doit clicker sur e*xpand* pour atteindre Association1. Cela étant fait, vu qu'il n'y a plus de paire <objet-relation> par la suite, le bouton *expand* devient invisible et le menu déroulant *Next Object* aussi. Cette fois le menu déroulant *Relationship type attribute* contient un attribut, Attribute1. On peut donc valider la formule.

Dans le cas où on veut compter le nombre d'éléments, la formule sera : Count(Association.Object2.Association1.Attribute1).

5.3.2. Attribut dérivé appartenant à une relation

5.3.2.1. Attribut dérivé et attribut référé dans la même relation

Dans le cas où l'attribut dérivé et l'attribut référé se trouvent dans la même relation, les seules choses à faire sont de choisir l'attribut référé dans le menu déroulant permettant le choix et de choisir la fonction agissant dans la formule de dérivation.

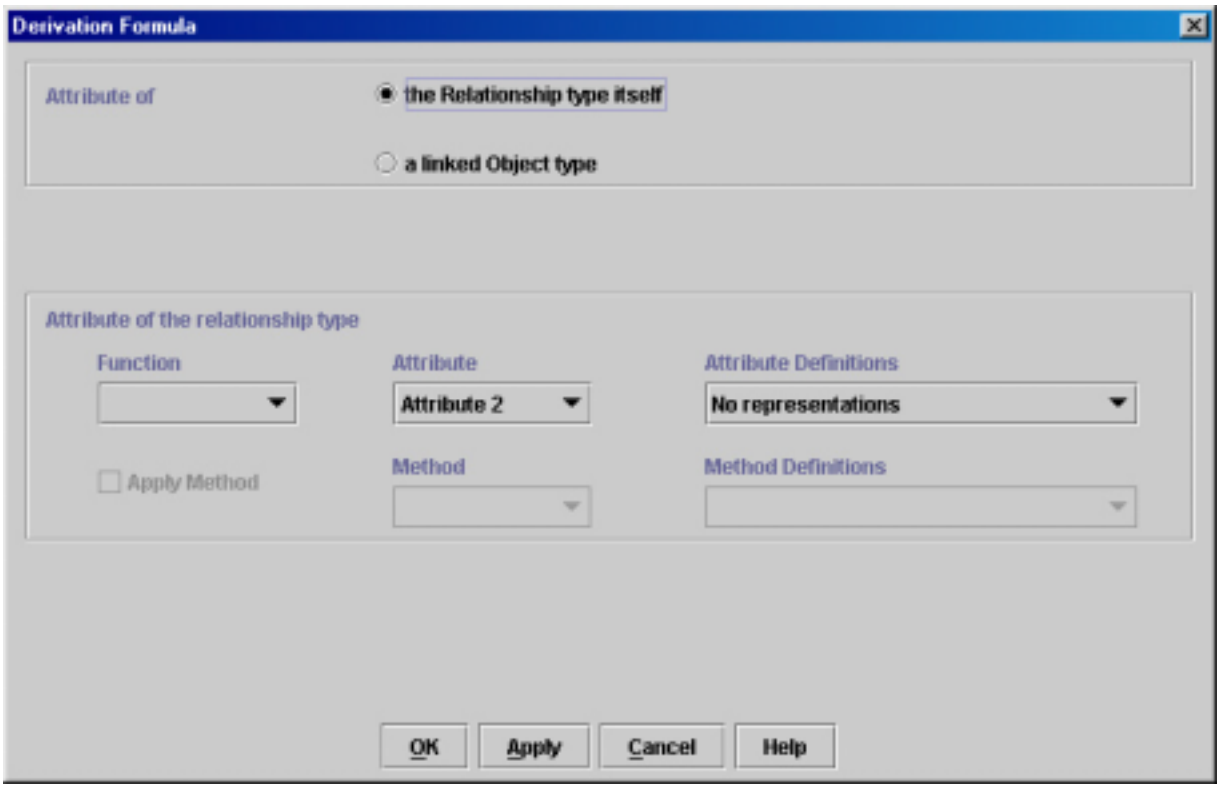


Figure 24
L'attribut dérivé et l'attribut référé se trouvent dans la même relation

5.3.2.2. Attribut référé appartenant à un objet

Si l'attribut référé appartient à un objet, le cas est similaire à celui dont l'attribut dérivé est dans un objet et l'attribut référé appartient à une relation mais cette fois ci, on considère des paires <relation-objet>.

Il faut choisir la fonction. Grâce au bouton *expand*, on peut parcourir la structure pour obtenir l'attribut référé. Le bouton *expand* n'est visible que s'il existe au moins deux objets dont un avec un attribut. Il faut aussi construire un chemin pointé.

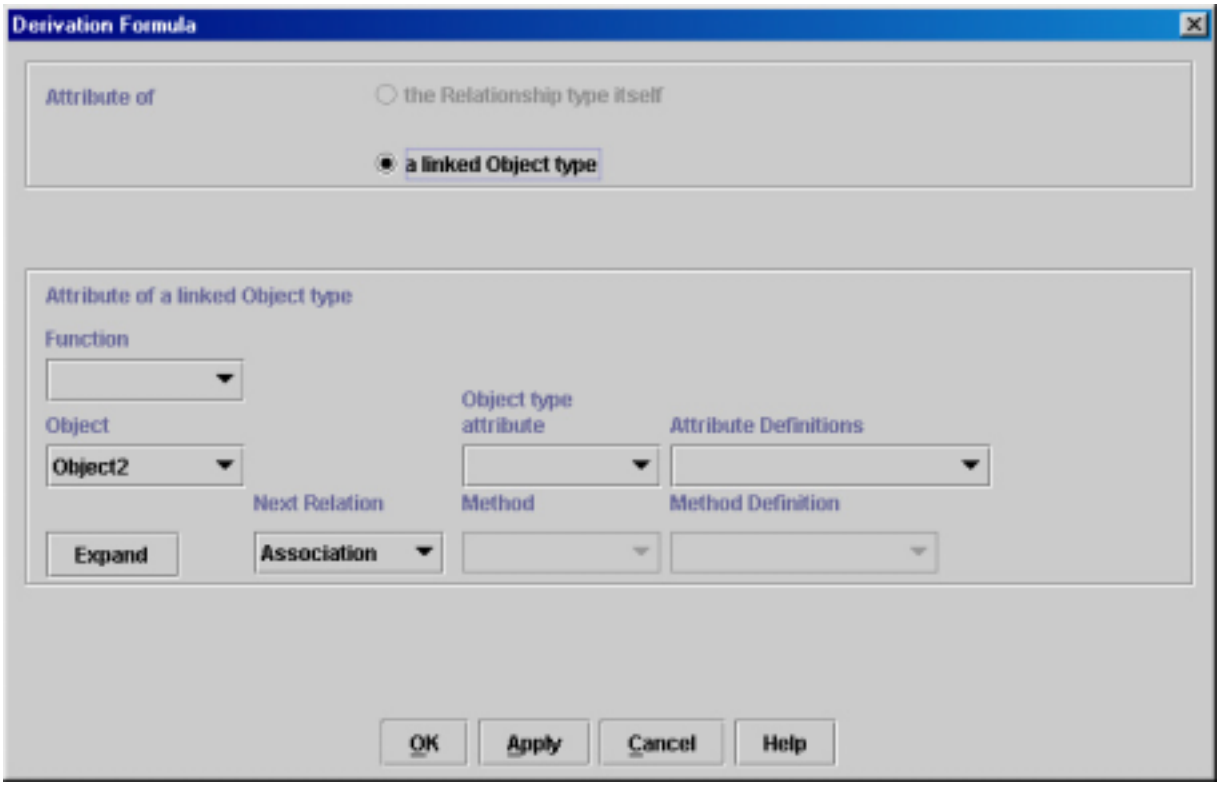


Figure 25
Attribut dérivé appartenant à une relation dont la référence se trouve dans un objet

Le menu déroulant *Object* contient les objets et le menu déroulant *Object type attribute* contient les attributs de ces objets. Le bouton e*xpand* devient invisible si on n'a plus de paires <relation-objet>. On est alors en fin de chemin et il est impossible d'aller plus loin. De même, lors de la validation, on vérifie qu'il y ait au moins un attribut dans le menu déroulant *Object type attribute*. Le menu déroulant *Next Relation* donne la liste de toutes les relations liées à l'objet du

couple <relation-objet> à l'exception de relation afin de ne pas revenir sur ses pas dans le chemin. Lorsqu'on appuie sur e*xpand*, le couple <relation-objet> sera en fait la relation sélectionnée dans le menu déroulant *Next Relation* et les objets qui lui sont liés.

En se basant sur la figure 23, si on veut sommer la formule sera :
Sum(Object2.Association. Object1.Attribute1).

5.4. Code

Le programme est écrit en Java. Son code source complet et commenté est en annexe. Son but est d'une part d'offrir une interface graphique permettant à l'utilisateur de construire une formule de dérivation et de visualiser cette formule, d'autre part, de construire une liste liée contenant le chemin afin de permettre la validation de celui-ci.

Voici la démarche suivie :

5.4.1. Constructeur

La classe DerivationFormula est constituée du constructeur, de méthodes permettant la mise à jour en cascade de tous les menus déroulants, et de méthodes permettant la construction de l'interface graphique.

Le constructeur est composé des méthodes init() et fillFields(). Il « dessine » les différentes interfaces grâce à la méthode init(). La méthode fillFields() vérifie quels sont les boutons d'options qui seront affichés et remplit les menus déroulants pour la première fois. Dès que les menus déroulants sont remplis pour la première fois, le rôle du constructeur est fini.

5.4.2. Affichage

La fenêtre de l'interface graphique Derivation Formula est « dessinée » par la méthode init(). Cette fenêtre est composé de trois parties :

- la partie supérieure contenant les boutons d'options servant à indiquer le type de l'élément contenant l'attribut référé ;
- la partie centrale permet la construction de la formule de dérivation ;
- la partie inférieure contenant les boutons Ok, Apply, Cancel et Help, les boutons Ok et Apply servent à valider la formule.

La méthode fillFields() vérifie quels sont les boutons d'options qui seront affichés et remplit les menus déroulants pour la première fois. Fillfields() déterminera le type du propriétaire (owner) de l'attribut dérivé et en fonction du type du propriétaire de l'attribut dérivé, s'il appartient à un objet, alors l'attribut référé peut se trouver sur le même objet que l'attribut dérivé, sur un autre objet ou sur une relation. Par contre si l'attribut dérivé appartient à une relation alors l'attribut référé peut se trouver sur la même relation que l'attribut dérivé ou sur un objet.

L'affichage de l'interface graphique dépend des choix de l'utilisateur. L'utilisateur choisit un attribut qu'il désigne comme un attribut dérivé, la méthode checking(), contenue dans fillFields, détermine si l'élément possédant cet attribut est un objet ou une relation (grâce au test owner instanceof TObjectType ou owner instanceof TRelationshipType).

Ceci permet l'affichage de la partie supérieure contenant les boutons d'options adéquats. Dans le cas où le owner est un objet, les boutons d'options permettant d'avoir un attribut référé sur le même objet que l'attribut dérivé, sur un autre objet ou sur une relation. Sinon les boutons d'options permettant d'avoir un attribut référé sur la même relation que l'attribut dérivé ou sur un objet, seront affichés car le owner sera une relation.

Ensuite, moyennant les tests suivants, la méthode checking() rend ces boutons d'options accessibles ou non :

- pour que le bouton d'options permettant d'atteindre un attribut référé sur le même objet soit activable, il faut au moins deux attributs (un dérivé et un dérivant) sur cet objet ;

- pour pouvoir atteindre une référence sur un autre objet, il faut qu'il y ait au moins un attribut sur un autre objet. Dans ce cas le bouton d'options permettant ce choix est activé ;
- le bouton d'options permettant d'avoir un attribut référé sur une relation n'est activé que s'il y a au moins un attribut sur une relation.

L'utilisateur peut ensuite choisir parmi les boutons d'options actifs. En fonction de ce choix, la partie centrale correspondante sera affichée.

5.4.3. Menus déroulants

Tous les menus déroulants possèdent des ActionListener. Ils sont donc à l'écoute et dès que l'utilisateur fait un choix, une mise à jour en cascade des menus déroulants dépendant de ce choix est réalisée.

Ce sont les ActionListener des différents menus déroulants qui remplissent et mettent à jour le contenu des menus déroulants. Les méthodes qui sont à l'écoute de ces ActionListener, sont appelées update* où * est le nom du menu déroulant.

Les paragraphes suivants décrivent cette mise à jour.

Le programme travaille par triplets <objet-relation-objet> dans le cas où l'attribut dérivé appartient à un objet et l'attribut dérivant à un autre objet, par paire <objet-relation> dans le cas où l'attribut dérivé appartient à un objet et l'attribut dérivant à une relation et par paire <relation-objet> dans le cas où l'attribut dérivé appartient à une relation et l'attribut dérivant à un objet.

Un chemin est une suite alternative d'objets et de relations. Pour la création de ce chemin, chaque objet ou relation selon le cas dans lequel on se trouve s'apparente à un croisement qui proposera plusieurs directions possibles.

En effet, pour le cas du triplet <objet-relation-objet>, lorsqu'on a choisi le premier objet, on a ensuite toute une série de relations possibles liées à cet objet. On a donc une liste de relations. Le choix de la relation dans cette liste permettra de fixer quelle sera la liste des choix possibles pour le deuxième objet.

Ces choix sont représentés par les différents menus déroulants. Etant donné que les choix sont fixés les uns après les autres et que le choix fait sur l'élément précédent fixe les choix à venir, il faut une mise à jour en cascade des menus déroulants. Cette mise à jour se fait en partant d'un choix, ce choix se répercute

sur l'élément suivant qui à son tour lorsqu'il est fixé répercute son choix sur le suivant, et ainsi de suite.

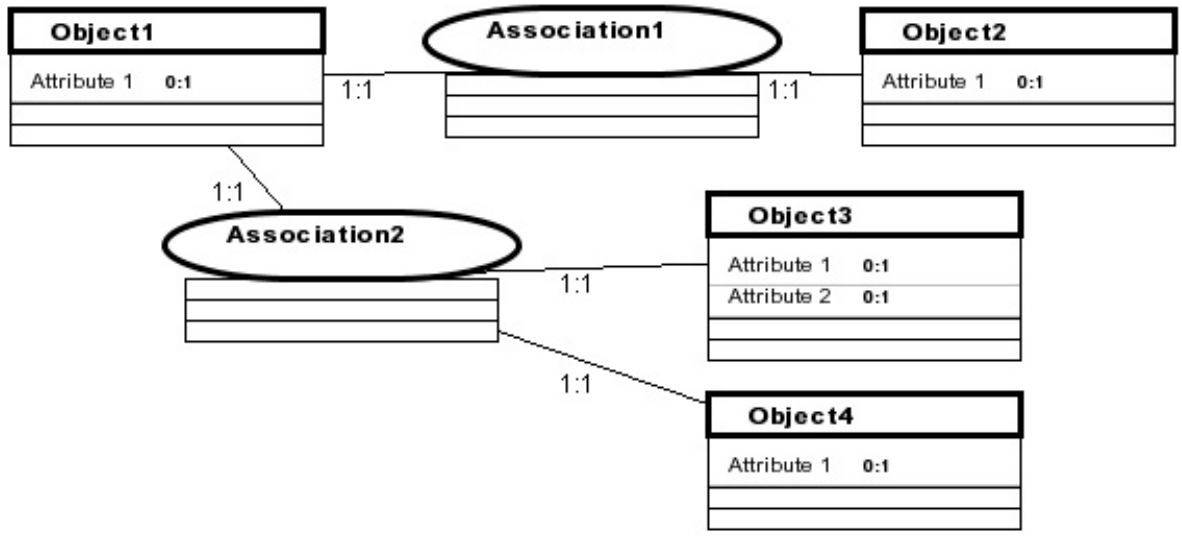


Figure 26
Exemple de mise à jour en cascade des menus déroulants

Pour illustrer cette mise en cascade, nous allons nous baser sur la figure 26. Nous sommes dans le cas où l'attribut dérivé est Attribute1 dans Object1 et où la l'attribut référé est Attribute2 dans Object3. Dans ce cas l'interface présente trois menus déroulants : les menus déroulants association, objet et attribut (figure 20).

Object1 représente le premier carrefour. Le schéma offre deux possibilités : Association1 et Association2. Ce choix est donné par le menu déroulant *association* qui contient les associations Association1 et Association2. Celles-ci représentent les carrefours suivants. Association1 n'offre qu'une seule possibilité et Association2 deux. Si on prend le chemin contenant Association1, donc si on sélectionne Association1 dans le menu déroulant association, les menus déroulant objet et attribut contiendront Object2 et Attribute1 respectivement. Si par contre, on choisit Association2, le menu déroulant objet contiendra Object3 et Object4. Le menu déroulant attribut contiendra Attribute1 et Attribute2 si Objet3 a été sélectionné ou Attribute1 si Object4 a été sélectionné.

Le tableau suivant représente les contenus des différents menus déroulants pour l'exemple de la figure 26 :

Menu déroulant Association	Menu déroulant Object	Menu déroulant Attribute
Association1	Object2	Attribute1
Association2	Object3	Attribute1
		Attribute2
	Object4	Attribute1

Il y a donc une hiérarchie pour la mise à jour des menus déroulants. Le menu déroulant *Association* implique la mise à jour du menu déroulant *Object* qui lui-même implique la mise à jour du menu déroulant *Attribute*.

Dans le cas où l'attribut dérivé appartient à un objet et l'attribut dérivant à une relation, on travaille par paire <objet-relation>. Le principe est le même : lorsqu'on a choisi le premier objet, on a ensuite toute une série de relations possibles liées à cet objet. Le choix de la relation permettra de fixer quelle sera la liste des choix possibles pour l'objet suivant ou pour l'attribut désiré.

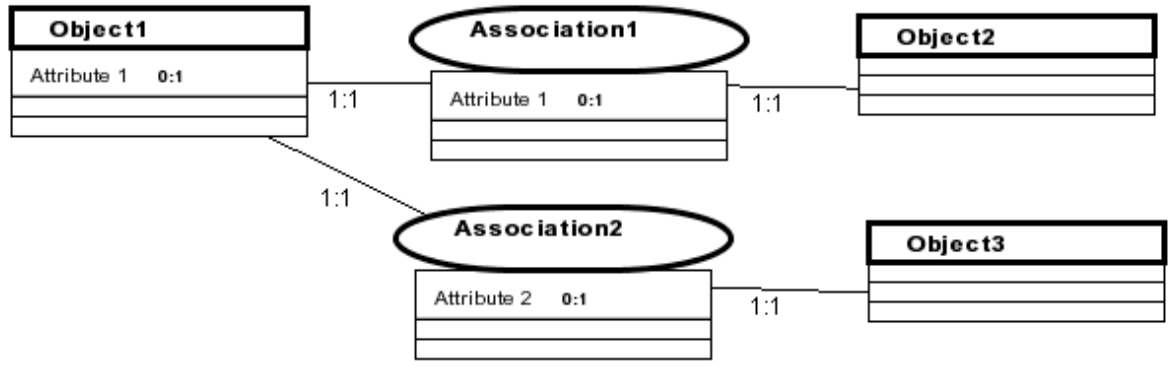


Figure 27
Exemple de mise à jour en cascade des menus déroulants

Pour la figure 27, nous sommes dans le cas où l'attribut dérivé est Attribute1 dans Object1 et où l'attribut référé est Attribute2 dans Association2. Dans ce cas l'interface présente trois menus déroulants : les menus déroulants association, objet suivant et attribut (figure 22).

Object1 représente le premier carrefour, le schéma offre deux possibilités : Association1 et Association2. Ce choix est donné par le menu déroulant

association qui contient les associations Association1 et Association2. Celles-ci représentent les carrefours suivants, Association1 et Association2 n'offrent qu'une seule possibilité. Le menu déroulant *objet* suivant contiendra Object2 ou Object3 suivant que l'on suive Association1 ou Association2 respectivement.

Le tableau suivant représente les contenus des différents menus déroulants pour l'exemple de la figure 27:

Menu déroulant Relationship	Menu déroulant Attribute	Menu déroulant Next Object
Association1	Attribute1	Object2
Association2	Attribute2	Object3

La hiérarchie de mise à jour des menus déroulants est ici différente, le menu déroulant *Association* implique la mise à jour du menu déroulant *Next Object* et la mise à jour du Menu déroulant *Attribute* simultanément. En effet, ils sont tous les deux liés directement sur l'association.

Dans le cas où l'attribut dérivé appartient à une relation et l'attribut dérivant à un objet, on travaille par paire <relation-objet>. Le principe est exactement le même que celui où l'on traite les paires <objet-relation>, il suffit de remplacer le mot objet par relation et le mot relation par objet dans les explications.

5.4.4. Chemin

La liste liée contenant le chemin est construite au fur et à mesure qu'on parcourt ce chemin. Le parcours de la structure se fait en appuyant sur le bouton *expand.*
Les méthodes qui sont à l'écoute de l'ActionListener du bouton *expand* sont les méthodes expand*. Les méthodes expand* où les * représentent le cas dans le quel on se trouve, c'est-à-dire attribut dérivé appartenant à un objet et attribut référé appartenant à un autre objet, attribut dérivé appartenant à un objet et attribut référé appartenant à une relation et attribut dérivé appartenant à une relation et attribut référé appartenant à un objet.

Le chemin parcouru pour atteindre l'attribut de référence est stocké dans une liste liée. Il sera aussi affiché dans la zone texte de la fenêtre « attribute properties » (figure 18).

Lors de la validation, cette liste sera parcourue pour vérifier si l'attribut est toujours présent et accessible. En effet, une fois la formule créée, le chemin est sauvé mais par la suite le schéma peut être modifié, ce qui peut provoquer un changement qui peut altérer le chemin ou l'utilisateur peut malencontreusement effacer l'attribut de référence. Si c'est le cas, après la validation, un message d'erreur indiquant que l'attribut visé n'est plus accessible sera affiché dans la fenêtre Messages (figure 9).

Pour créer une formule de dérivation, il faut indiquer le chemin à suivre pour atteindre les attributs référés. Pour cela, il faut pouvoir parcourir la structure des données.

La structure utilisée par MADS est une structure en arbre. L'approche suivie pour la validation est une approche « top to bottom », un schéma n'est valide que si ses composants sont valides.

Les différents éléments d'un schéma sont doublement liés, ce qui permet de parcourir la structure du haut vers le bas et d'un point à un autre. Ceci est très utile pour trouver les caractéristiques d'un attribut dérivé.

Les différents éléments d'un schéma sont placés dans des listes liées (linked list de Java), l'essentiel du travail consiste à parcourir ces listes et à en sélectionner les éléments adéquats.

Dans ce but, on utilise des outils déjà développés antérieurement pour parcourir la structure, sélectionner des éléments,… Ces outils permettent de sélectionner des éléments d'un schéma et les renvoient sous forme d'une liste liée. Ces outils (les méthodes getRoles, getObject, getRelation, getAttributes) permettent de sélectionner les roles, objets, relations, attributs respectivement.

5.4.5. Validation

Lorsque l'utilisateur appuie sur le bouton OK, la méthode setFields(), qui est à l'écoute de l'ActionListener du bouton OK, vérifie s'il on peut atteindre l' attribut de référence de la formule de dérivation. Si c'est le cas cette formule sera affichée dans la zone de texte Derivation Formula de la fenêtre « Attribute properties » (figure 18). Sinon un message d'erreur sera envoyé.

6. Conclusions

Le travail que j'ai effectué était avant tout un travail de programmation. En effet, il s'agissait de créer une interface graphique permettant la création d'une formule de dérivation pour les attributs dérivés.

Il fallait néanmoins avoir une vue d'ensemble du projet afin de comprendre le contexte dans lequel mon travail allait s'inscrire. La première partie était donc destinée à comprendre ce qu'est une base de données spatio-temporelle et les causes qui ont mené à concevoir le projet MURMUR.

Une deuxième partie, consistait à connaître les fonctionnalités du logiciel MADS ainsi qu'à découvrir la structure des données dans MADS. Ce fut la partie la plus pénible sans aucun doute. Pour maitriser la structure des données MADS et l'utilisation des outils permettant de parcourir cette structure, il m'a fallut revoir beaucoup de concepts fondamentaux de Java afin de pouvoir dominer les listes liées.

Néanmoins, une fois la structure maîtrisée, la programmation était très agréable.

Enfin pour finir, il fallait penser comment présenter l'interface graphique permettant de créer une formule de dérivation. Il fallait traiter tous les cas et pour chacun d'eux fournir une formule et si c'était possible, parcourir la structure à la recherche d'un attribut référé.

Pour ce qui est du traitement des différents cas, la première séparation logique m'a semblée être celle faite par le type du propriétaire de l'attribut dérivé. On affiche ainsi tous les cas où les owner sont du type objet ensemble et tous ceux du type relation ensemble. Pour permettre ce choix, les boutons d'options sont un bon choix car ces cas sont fixes et ces boutons peuvent être activés sous certaines conditions.

Pour la création du chemin, la notion de choix et le fait que ce choix est variable, m'ont conduit au menu déroulant. C'est la manière la plus simple pour permettre un choix variable.

Pour la mise à jour en cascade des menus déroulants, j'ai du revoir la notion d'ActionListener.

Dans un premier temps, je n'avais pas créé le bouton expand et on ne pouvait se déplacer que vers ses voisins immédiats. J'ai ensuite créé ce bouton permettant de parcourrir la structure.

D'un point de vue théorique, le projet MURMUR vient combler les lacunes limitant les géographes dans leurs besoins informatiques. La solution future serait de permettre la généralisation automatique d'une carte quelconque à partir de la carte la plus détaillée. On est encore loin d'y arriver.

7. Bibliographie

1. ESTEBAN ZIMÁNYI, Multiple representations and multiple resolutions in Spatio-temporal databases. Conférence donnée le 17/04/2002 dans le cadre du cours « Questions actuelles d'informatique » (INFO 368).

2. CHRISTINE PARENT, MURMUR Project Deliverable 8 : Query Language Specification. Version: 6. Date: 11/01/2001.

3. STEFANO SPACCAPIETRA, MURMUR Project Deliverable 5: Data Model specification. Version: 4. Date: 18/09/2000.

4. STEFANO SPACCAPIETRA, MURMUR Project Deliverable 11: Schema Editor. Version: 3. Date: 21/03/2001.

5. CHRISTINE PARENT, MURMUR Project Deliverable 4: State of the Art review. Version: 4. Date: 18/09/2000.

6. ROLAND BURNET, MURMUR Project Deliverable 7: Service analysis. Version: 6. Date: 08/01/2001.

7. Site internet du Projet MURMUR : http://lbdwww.epfl.ch/e/MurMur/

8. Site internet : http://java.sun.com, section Java tutorials.

Annexe

Annexe : Listing du code du programme Derivation Formula

```
package mads.editor.ui;

import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JScrollPane;
import javax.swing.JComboBox;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.BorderFactory;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.DefaultComboBoxModel;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Container;
import java.awt.Frame;
import java.awt.Dimension;
import java.awt.Insets;
import java.awt.Dialog;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.event.KeyEvent;
import java.util.Iterator;
import mads.tstructure.core.TType;
import mads.tstructure.core.TAttribute;
import mads.tstructure.core.TAttributeDefinition;
import mads.tstructure.core.TAttributeDerived;
import mads.tstructure.core.TRole;
import mads.tstructure.core.TAttributeReference;
import mads.tstructure.core.TObjectType;
import mads.tstructure.core.TRelationshipType;
import mads.tstructure.core.TRepresentation;
import mads.tstructure.utils.TList;
import mads.tstructure.utils.exceptions.InvalidElementException;

public class DerivationFormula extends JDialog {
```

```
  private TType object;
  private TType objectInitial;
  private TAttribute attribute;
  private TAttributeDefinition attributeDef;
  private TAttributeDerived derivedAttribute;
  private JTextArea toUpdate;
  private final static int SELF = 100;
  private final static int RELATION = 101;
  private final static int OBJECT = 102;
  private JRadioButton rSelf;
  private JRadioButton rRelation = new JRadioButton("a linked Relationship type");
  private JRadioButton rObject = new JRadioButton("a linked Object type");
  private JPanel pSelf;
  private JPanel pRelation;
  private JPanel pObject;
  private boolean error = false;
  //---------------------variable for self derive attribute----------------------
  private JComboBox cbSelfFunction;
  private JComboBox cbSelfAttribute;
  private JComboBox cbSelfAttributeRstamps;
  private JComboBox cbSelfMethod;
  private JCheckBox ckSelfMethod;
  private JComboBox cbSelfMethodRstamps;
  //---------------------variable for linked relationship attribute----------------------
  private JComboBox cbRelationFunction;
  private JComboBox cbRelationRole;
  private JComboBox cbRelNextObject = new JComboBox();
  private JComboBox cbRelationAttribute;
  private JComboBox cbRelationAttRstamp;
  private JComboBox cbRelationMethod;
  private JComboBox cbRelationMethodRstamp;
  //-----------------------variables for linked object type with relationship----------------------------
------
  private JComboBox cbObjectFunction = new JComboBox();
  private JComboBox cbObjectRole = new JComboBox();
  private JComboBox cbObjNextObject = new JComboBox();
  private JComboBox cbObjectAttribute = new JComboBox();
  private JComboBox cbObjectAttRstamp = new JComboBox();
  private JComboBox cbObjectMethod = new JComboBox();
  private JComboBox cbObjectMethodRstamp = new JComboBox();
  //-----------------------variables for linked object type thru relationship-----------------------------
-----
  private JComboBox cbLinkObjRole = new JComboBox();
  //private JComboBox cbLinkNextObject = new JComboBox();
  private JComboBox cbLinkObjAttRstamp = new JComboBox();
  private JComboBox cbLinkObjMethod = new JComboBox();
  private JComboBox cbLinkObjMethodRstamp = new JComboBox();

  private JComboBox cbLinkObjectAttribute = new JComboBox();
  private JComboBox cbLinkObjName = new JComboBox();
```

```
  //-----------------------------------------------------------------------------------------
  private GridBagLayout gBag = new GridBagLayout();
  private GridBagConstraints gBagConst = new GridBagConstraints();
  private int radioType;
  private TList listRole;
  private TList listRoler;

  private TAttributeReference attRef;
  private TRole roleRel;
  private TRole roleObj;
  private TRole roleObjRel;
  private TRelationshipType saveRel;
  private TObjectType saveobj;
  private JButton btParOR = new JButton("Expand");
  private JButton btParOO = new JButton("Expand");
  private JButton btParRO = new JButton("Expand");
  private String formula;
  private boolean click = true;
  private boolean clickor = true;
  private boolean clickro = true;
  private String fonction;

  public DerivationFormula(JFrame frame, TAttributeDerived derivedAttribute, JTextArea
textField) {
    super(frame, "Derivation Formula", true);
    this.derivedAttribute = derivedAttribute;
    this.attributeDef = derivedAttribute.getOwner();
    this.attribute = attributeDef.getOwner();
    this.toUpdate = textField;
    object = attribute.getOwner();
    objectInitial = object;
    init();
    fillFields();
  }

  public boolean getError() {
    return error;
  }

  private void init() {
    JPanel panel = (JPanel)this.getContentPane();
    JPanel innerPanel;
    JLabel label;
    if (object instanceof TObjectType)
      rSelf = new JRadioButton("the Object type itself");
    else if (object instanceof TRelationshipType)
      rSelf = new JRadioButton("the Relationship type itself");
    //------------------------ panel for rself,rrelation,robject buttons itself-----------------------------
    gBagConst.gridx = 0;
    gBagConst.gridy = 0;
```

```
gBagConst.weightx = 1.0;
gBagConst.weighty = 0.3;
gBagConst.gridwidth = GridBagConstraints.REMAINDER;
gBagConst.anchor = GridBagConstraints.NORTHWEST;
{

  innerPanel = new JPanel(gBag);
  innerPanel.setBorder(BorderFactory.createEtchedBorder());
  gBagConst.insets = new Insets(10, 10, 0, 0);
  label = new JLabel("Attribute of");
  gBag.setConstraints(label, gBagConst);
  innerPanel.add(label);
  gBagConst.insets = new Insets(5, 200, 0, 0);
  gBag.setConstraints(rSelf, gBagConst);
  innerPanel.add(rSelf);
  if (object instanceof TObjectType) {
    gBagConst.insets = new Insets(25, 200, 0, 0);
    gBag.setConstraints(rRelation, gBagConst);
    innerPanel.add(rRelation);
  }
  gBagConst.insets = new Insets(45, 200, 0, 0);
  gBag.setConstraints(rObject, gBagConst);
  innerPanel.add(rObject);
  ButtonGroup rg = new ButtonGroup();
  rg.add(rObject);
  rg.add(rRelation);
  rg.add(rSelf);
  //--------------------------Listener on radio buttons-----------------------------------
  rSelf.addItemListener(
      new ItemListener() {
        public void itemStateChanged(ItemEvent e) {
          if (e.getStateChange() == ItemEvent.SELECTED) {
            pSelf.setVisible(true);
          } else if (e.getStateChange() == ItemEvent.DESELECTED) {
            pSelf.setVisible(false);
          }
        }
      });
  rObject.addItemListener(
      new ItemListener() {
        public void itemStateChanged(ItemEvent e) {
          if (e.getStateChange() == ItemEvent.SELECTED) {
            pObject.setVisible(true);
          } else if (e.getStateChange() == ItemEvent.DESELECTED) {
            pObject.setVisible(false);
          }
        }
      });
}
gBagConst.anchor = GridBagConstraints.NORTH;
```

```
    gBagConst.fill = GridBagConstraints.HORIZONTAL;
    gBagConst.insets = new Insets(10, 10, 0, 10);
    gBag.setConstraints(innerPanel, gBagConst);
    panel.add(innerPanel);
    //---------------------------------------for self ----------------------------------------------------
-----------------
    {
      gBagConst.fill = GridBagConstraints.NONE;
      gBagConst.anchor = GridBagConstraints.NORTHWEST;

      innerPanel = new JPanel(gBag);
      pSelf = innerPanel;
      innerPanel.setBorder(BorderFactory.createEtchedBorder());
      gBagConst.insets = new Insets(5, 10, 0, 0);
      if (object instanceof TObjectType)
        label = new JLabel("Attribute of the Object type");
      else
        label = new JLabel("Attribute of the relationship type");
      gBag.setConstraints(label, gBagConst);
      innerPanel.add(label);
      gBagConst.insets = new Insets(30, 40, 0, 0);
      label = new JLabel("Function");
      gBag.setConstraints(label, gBagConst);
      innerPanel.add(label);
      gBagConst.insets = new Insets(50, 40, 5, 0);
      cbSelfFunction = new JComboBox();
      gBag.setConstraints(cbSelfFunction, gBagConst);
      addFunctions(cbSelfFunction);
      cbSelfFunction.setPreferredSize(new Dimension(115, 25));
      innerPanel.add(cbSelfFunction);
      gBagConst.insets = new Insets(30, 210, 0, 0);
      label = new JLabel("Attribute");
      gBag.setConstraints(label, gBagConst);
      innerPanel.add(label);
      gBagConst.insets = new Insets(50, 210, 5, 0);
      cbSelfAttribute = new JComboBox();
      cbSelfAttribute.setPreferredSize(new Dimension(115, 25));

      cbSelfAttribute.addItemListener(new CbSelfAttListener());
      gBag.setConstraints(cbSelfAttribute, gBagConst);
      innerPanel.add(cbSelfAttribute);
      gBagConst.insets = new Insets(30, 390, 0, 0);
      label = new JLabel("Attribute Definitions");
      gBag.setConstraints(label, gBagConst);
      innerPanel.add(label);
      gBagConst.insets = new Insets(50, 390, 5, 0);
      cbSelfAttributeRstamps = new JComboBox();
      gBag.setConstraints(cbSelfAttributeRstamps, gBagConst);
      cbSelfAttributeRstamps.setRenderer(new ComboRenderer());
      cbSelfAttributeRstamps.setPreferredSize(new Dimension(250, 25));
```

```
      innerPanel.add(cbSelfAttributeRstamps);
      gBagConst.insets = new Insets(95, 40, 5, 0);
      ckSelfMethod = new JCheckBox("Apply Method");
      gBag.setConstraints(ckSelfMethod, gBagConst);
      ckSelfMethod.setEnabled(false);
      innerPanel.add(ckSelfMethod);
      gBagConst.insets = new Insets(90, 210, 0, 0);
      label = new JLabel("Method");
      gBag.setConstraints(label, gBagConst);
      innerPanel.add(label);
      gBagConst.insets = new Insets(110, 210, 5, 0);
      cbSelfMethod = new JComboBox();
      cbSelfMethod.setPreferredSize(new Dimension(115, 25));
      cbSelfMethod.setEnabled(false);
      gBag.setConstraints(cbSelfMethod, gBagConst);
      innerPanel.add(cbSelfMethod);
      gBagConst.insets = new Insets(90, 390, 5, 0);
      label = new JLabel("Method Definitions");
      gBag.setConstraints(label, gBagConst);
      innerPanel.add(label);
      gBagConst.insets = new Insets(110, 390, 5, 0);
      cbSelfMethodRstamps = new JComboBox();
      cbSelfMethodRstamps.setPreferredSize(new Dimension(250, 25));
      cbSelfMethodRstamps.setEnabled(false);
      gBag.setConstraints(cbSelfMethodRstamps, gBagConst);
      innerPanel.add(cbSelfMethodRstamps);


    }
    gBagConst.gridy = 1;
    gBagConst.anchor = GridBagConstraints.NORTH;
    gBagConst.fill = GridBagConstraints.HORIZONTAL;
    gBagConst.insets = new Insets(10, 10, 0, 10);
    gBag.setConstraints(innerPanel, gBagConst);
    panel.add(innerPanel);
    innerPanel.setVisible(false);
    //--------------------------For linked relationship rRelatioship-----------------------------------
--------------------
    {
      if (object instanceof TObjectType) {
        gBagConst.fill = GridBagConstraints.NONE;
        gBagConst.anchor = GridBagConstraints.NORTHWEST;

        innerPanel = new JPanel(gBag);
        pRelation = innerPanel;
        innerPanel.setBorder(BorderFactory.createEtchedBorder());
        gBagConst.insets = new Insets(5, 10, 0, 0);
        label = new JLabel("Attribute of a linked Relationship type");
        gBag.setConstraints(label, gBagConst);
        innerPanel.add(label);
```

```
gBagConst.insets = new Insets(30, 10, 0, 0);
label = new JLabel("Function");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(50, 10, 5, 0);
cbRelationFunction = new JComboBox();
addFunctions(cbRelationFunction);
cbRelationFunction.setPreferredSize(new Dimension(115, 25));
gBag.setConstraints(cbRelationFunction, gBagConst);
innerPanel.add(cbRelationFunction);
gBagConst.insets = new Insets(80, 10, 0, 0);
label = new JLabel("Relationship");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(100, 10, 5, 0);
cbRelationRole = new JComboBox();
cbRelationRole.setPreferredSize(new Dimension(115, 25));
gBag.setConstraints(cbRelationRole, gBagConst);
cbRelationRole.setActionCommand("RelationRole");
cbRelationRole.addActionListener(new ButtonListener());
cbRelationRole.setRenderer(new ComboBoxRenderer());
innerPanel.add(cbRelationRole);


btParOR.setActionCommand("ExpandOR");
btParOR.setMnemonic(KeyEvent.VK_O);
btParOR.setToolTipText("Expand formula");
btParOR.addActionListener(new ButtonListener());
gBagConst.insets = new Insets(150, 10, 0, 0);
gBag.setConstraints(btParOR, gBagConst);
innerPanel.add(btParOR);

gBagConst.insets = new Insets(125, 130, 0, 0);
label = new JLabel("Next Object");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(150, 130, 5, 0);
gBag.setConstraints(cbRelNextObject, gBagConst);
cbRelNextObject.setPreferredSize(new Dimension(90, 25));
cbRelNextObject.setRenderer(new ComboBoxRenderer());
innerPanel.add(cbRelNextObject);




gBagConst.insets = new Insets(65, 250, 0, 0);
label = new JLabel("Relationship type");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
```

```
gBagConst.insets = new Insets(80, 250, 0, 0);
label = new JLabel("attribute");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(100, 250, 5, 0);
cbRelationAttribute = new JComboBox();
cbRelationAttribute.setPreferredSize(new Dimension(115, 25));
cbRelationAttribute.setActionCommand("RelAttDefinitions");
cbRelationAttribute.addActionListener(new ButtonListener());
gBag.setConstraints(cbRelationAttribute, gBagConst);
//cbAvAttribute.setRenderer(renderer);
innerPanel.add(cbRelationAttribute);
gBagConst.insets = new Insets(80, 370, 0, 0);
label = new JLabel("Attribute Definitions");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(100, 370, 5, 0);
cbRelationAttRstamp = new JComboBox();
cbRelationAttRstamp.setRenderer(new ComboRenderer());
cbRelationAttRstamp.setPreferredSize(new Dimension(210, 25));
gBag.setConstraints(cbRelationAttRstamp, gBagConst);
innerPanel.add(cbRelationAttRstamp);
gBagConst.insets = new Insets(125, 250, 0, 0);
label = new JLabel("Method");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(150, 250, 5, 0);
cbRelationMethod = new JComboBox();
gBag.setConstraints(cbRelationMethod, gBagConst);
cbRelationMethod.setPreferredSize(new Dimension(115, 25));
cbRelationMethod.setEnabled(false);
innerPanel.add(cbRelationMethod);
gBagConst.insets = new Insets(125, 370, 0, 0);
label = new JLabel("Method Definition");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(150, 370, 5, 0);
cbRelationMethodRstamp = new JComboBox();
gBag.setConstraints(cbRelationMethodRstamp, gBagConst);
cbRelationMethodRstamp.setPreferredSize(new Dimension(155, 25));
cbRelationMethodRstamp.setEnabled(false);
innerPanel.add(cbRelationMethodRstamp);
rRelation.addItemListener(
   new ItemListener() {
     public void itemStateChanged(ItemEvent e) {
       if (e.getStateChange() == ItemEvent.SELECTED) {
         pRelation.setVisible(true);
       } else if (e.getStateChange() == ItemEvent.DESELECTED) {
         pRelation.setVisible(false);
       }
```

```
            }
          });
      }
    }
    gBagConst.gridy = 1;
    gBagConst.anchor = GridBagConstraints.NORTH;
    gBagConst.fill = GridBagConstraints.HORIZONTAL;
    gBagConst.insets = new Insets(10, 10, 0, 10);
    gBag.setConstraints(innerPanel, gBagConst);
    panel.add(innerPanel);
    innerPanel.setVisible(false);
    //-------------------------------------------------------------------------------------------------------
-------
    {
      if (object instanceof TObjectType) {
        gBagConst.fill = GridBagConstraints.NONE;
        gBagConst.anchor = GridBagConstraints.NORTHWEST;

        innerPanel = new JPanel(gBag);
        pObject = innerPanel;
        innerPanel.setBorder(BorderFactory.createEtchedBorder());
        gBagConst.insets = new Insets(5, 10, 0, 0);
        label = new JLabel("Attribute of a linked Object type");
        gBag.setConstraints(label, gBagConst);
        innerPanel.add(label);
        gBagConst.insets = new Insets(30, 10, 0, 0);
        label = new JLabel("Function");
        gBag.setConstraints(label, gBagConst);
        innerPanel.add(label);
        gBagConst.insets = new Insets(50, 10, 5, 0);
        gBag.setConstraints(cbObjectFunction, gBagConst);
        addFunctions(cbObjectFunction);
        cbObjectFunction.setPreferredSize(new Dimension(115, 25));
        innerPanel.add(cbObjectFunction);


        btParOO.setActionCommand("ExpandOO");
        btParOO.setMnemonic(KeyEvent.VK_O);
        btParOO.setToolTipText("Expand formula");
        btParOO.addActionListener(new ButtonListener());
        gBagConst.insets = new Insets(150, 10, 0, 0);
        gBag.setConstraints(btParOO, gBagConst);
        innerPanel.add(btParOO);

        gBagConst.insets = new Insets(80, 10, 0, 0);
        label = new JLabel("Relation");
        gBag.setConstraints(label, gBagConst);
        innerPanel.add(label);
        gBagConst.insets = new Insets(100, 10, 5, 0);
        gBag.setConstraints(cbLinkObjRole, gBagConst);
```

```
  cbLinkObjRole.setRenderer(new ComboBoxRenderer());
  cbLinkObjRole.setPreferredSize(new Dimension(115, 25));
  cbLinkObjRole.setActionCommand("ObjectLinkRole");
  cbLinkObjRole.addActionListener(new ButtonListener());
  innerPanel.add(cbLinkObjRole);

  gBagConst.insets = new Insets(80, 250, 0, 0);
  label = new JLabel("Object type");
  gBag.setConstraints(label, gBagConst);
  innerPanel.add(label);
  gBagConst.insets = new Insets(100, 250, 5, 0);

  gBag.setConstraints(cbLinkObjName, gBagConst);
  cbLinkObjName.setRenderer(new ComboBoxRenderer());
  cbLinkObjName.setPreferredSize(new Dimension(115, 25));

  innerPanel.add(cbLinkObjName);
  gBagConst.insets = new Insets(65, 370, 0, 0);
  label = new JLabel("Object type");
  gBag.setConstraints(label, gBagConst);
  innerPanel.add(label);
  gBagConst.insets = new Insets(80, 370, 0, 0);
  label = new JLabel("attribute");
  gBag.setConstraints(label, gBagConst);
  innerPanel.add(label);
  gBagConst.insets = new Insets(100, 370, 5, 0);
  gBag.setConstraints(cbLinkObjectAttribute, gBagConst);
  cbLinkObjectAttribute.setPreferredSize(new Dimension(115, 25));
  cbLinkObjectAttribute.setActionCommand("ObjLinkAttDefinitions");
  cbLinkObjectAttribute.addActionListener(new ButtonListener());
  innerPanel.add(cbLinkObjectAttribute);
  gBagConst.insets = new Insets(80, 490, 0, 0);
  label = new JLabel("Attribute Definitions");
  gBag.setConstraints(label, gBagConst);
  innerPanel.add(label);
  gBagConst.insets = new Insets(100, 490, 5, 0);
  gBag.setConstraints(cbLinkObjAttRstamp, gBagConst);
  cbLinkObjAttRstamp.setRenderer(new ComboRenderer());
  cbLinkObjAttRstamp.setPreferredSize(new Dimension(185, 25));
  innerPanel.add(cbLinkObjAttRstamp);
} else {

  gBagConst.fill = GridBagConstraints.NONE;
  gBagConst.anchor = GridBagConstraints.NORTHWEST;

  innerPanel = new JPanel(gBag);
  pObject = innerPanel;
  innerPanel.setBorder(BorderFactory.createEtchedBorder());
  gBagConst.insets = new Insets(5, 10, 0, 0);
  label = new JLabel("Attribute of a linked Object type");
```

```
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(30, 10, 0, 0);
label = new JLabel("Function");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(50, 10, 5, 0);
gBag.setConstraints(cbObjectFunction, gBagConst);
addFunctions(cbObjectFunction);
cbObjectFunction.setPreferredSize(new Dimension(115, 25));
innerPanel.add(cbObjectFunction);
gBagConst.insets = new Insets(80, 10, 0, 0);
label = new JLabel("Object");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(100, 10, 5, 0);
gBag.setConstraints(cbObjectRole, gBagConst);
cbObjectRole.setRenderer(new ComboBoxRenderer());
cbObjectRole.setPreferredSize(new Dimension(115, 25));
cbObjectRole.setActionCommand("ObjectRole");
cbObjectRole.addActionListener(new ButtonListener());
innerPanel.add(cbObjectRole);


btParRO.setActionCommand("ExpandRO");
btParRO.setMnemonic(KeyEvent.VK_O);
btParRO.setToolTipText("Expand formula");
btParRO.addActionListener(new ButtonListener());
gBagConst.insets = new Insets(150, 10, 0, 0);
gBag.setConstraints(btParRO, gBagConst);
innerPanel.add(btParRO);

gBagConst.insets = new Insets(125, 130, 0, 0);
label = new JLabel("Next Relation");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(150, 130, 5, 0);
gBag.setConstraints(cbObjNextObject, gBagConst);
cbObjNextObject.setPreferredSize(new Dimension(110, 25));
cbObjNextObject.setRenderer(new ComboBoxRenderer());
innerPanel.add(cbObjNextObject);



gBagConst.insets = new Insets(65, 250, 0, 0);
label = new JLabel("Object type");
gBag.setConstraints(label, gBagConst);
innerPanel.add(label);
gBagConst.insets = new Insets(80, 250, 0, 0);
label = new JLabel("attribute");
```

```
        gBag.setConstraints(label, gBagConst);
        innerPanel.add(label);
        gBagConst.insets = new Insets(100, 250, 5, 0);
        gBag.setConstraints(cbObjectAttribute, gBagConst);
        cbObjectAttribute.setPreferredSize(new Dimension(115, 25));
        cbObjectAttribute.setActionCommand("ObjAttDefinitions");
        cbObjectAttribute.addActionListener(new ButtonListener());
        innerPanel.add(cbObjectAttribute);
        gBagConst.insets = new Insets(80, 370, 0, 0);
        label = new JLabel("Attribute Definitions");
        gBag.setConstraints(label, gBagConst);
        innerPanel.add(label);
        gBagConst.insets = new Insets(100, 370, 5, 0);
        gBag.setConstraints(cbObjectAttRstamp, gBagConst);
        cbObjectAttRstamp.setRenderer(new ComboRenderer());
        cbObjectAttRstamp.setPreferredSize(new Dimension(185, 25));
        innerPanel.add(cbObjectAttRstamp);
      }
      gBagConst.insets = new Insets(125, 250, 0, 0);
      label = new JLabel("Method");
      gBag.setConstraints(label, gBagConst);
      innerPanel.add(label);
      gBagConst.insets = new Insets(150, 250, 5, 0);
      gBag.setConstraints(cbObjectMethod, gBagConst);
      cbObjectMethod.setPreferredSize(new Dimension(115, 25));
      cbObjectMethod.setEnabled(false);
      innerPanel.add(cbObjectMethod);
      gBagConst.insets = new Insets(125, 370, 0, 0);
      label = new JLabel("Method Definition");
      gBag.setConstraints(label, gBagConst);
      innerPanel.add(label);
      gBagConst.insets = new Insets(150, 370, 5, 0);
      gBag.setConstraints(cbObjectMethodRstamp, gBagConst);
      cbObjectMethodRstamp.setPreferredSize(new Dimension(155, 25));
      cbObjectMethodRstamp.setEnabled(false);
      innerPanel.add(cbObjectMethodRstamp);
    }
    gBagConst.gridy = 1;
    gBagConst.anchor = GridBagConstraints.NORTH;
    gBagConst.fill = GridBagConstraints.HORIZONTAL;
    gBagConst.insets = new Insets(10, 10, 0, 10);
    gBag.setConstraints(innerPanel, gBagConst);
    panel.add(innerPanel);
    innerPanel.setVisible(false);
    //-------------------------------------------------------------------------------------------------------
-----
    gBagConst.fill = GridBagConstraints.NONE;
    gBagConst.anchor = GridBagConstraints.NORTHWEST;
    JPanel panButton = new JPanel();
    JButton btCancel = new JButton("Cancel");
```

```
    JButton btApply = new JButton("Apply");
    JButton btHelp = new JButton("Help");
    JButton btOk = new JButton("OK");
    btOk.setActionCommand("Ok");
    btOk.setMnemonic(KeyEvent.VK_O);
    btOk.setToolTipText("Set fields and close the window");
    btOk.addActionListener(new ButtonListener());
    btApply.setActionCommand("Apply");
    btApply.setMnemonic(KeyEvent.VK_A);
    btApply.setToolTipText("Set fields ");
    btApply.addActionListener(new ButtonListener());
    btCancel.setActionCommand("Cancel");
    btCancel.setMnemonic(KeyEvent.VK_C);
    btCancel.setToolTipText("Close the window");
    btCancel.addActionListener(new ButtonListener());
    panButton.add(btOk);
    panButton.add(btApply);
    panButton.add(btCancel);
    panButton.add(btHelp);
    gBagConst.gridy = 2;
    gBagConst.insets = new Insets(0, 0, 0, 0);
    gBagConst.anchor = GridBagConstraints.SOUTH;
    gBag.setConstraints(panButton, gBagConst);
    panel.add(panButton);
    panel.setLayout(gBag);

  }



  /** Fill cbSelfAttributeRstamps combo box */
  private void fillSelfAttributeRstamps() {
    if (cbSelfAttribute.getSelectedItem() == null) return;
    TAttribute attribute = (TAttribute)cbSelfAttribute.getSelectedItem();
    cbSelfAttributeRstamps.removeAllItems();
    Iterator iter = attribute.getDefinitions().listIterator();
    while (iter.hasNext()) {
      TAttributeDefinition def = (TAttributeDefinition)iter.next();
      System.out.println("fillSelfAttributeRstamps attribute.getDefinitions().size() : " +
attribute.getDefinitions().size());
      if (!(def.equals(attributeDef))) {
        cbSelfAttributeRstamps.addItem(def);
      }
    }

  }

  /**  */
  private void fillRelationName() {
    if (cbRelationRole.getSelectedItem() == null) return;
```

```
    TRelationshipType relSel = (TRelationshipType)cbRelationRole.getSelectedItem();//feli

    cbRelNextObject.removeAllItems(); //feli
    if (relSel == null) return;

    updateRelationAttributes(relSel);
  }

  private void updateRelationAttributes(TRelationshipType rel) {
    if (rel == null) return;
    cbRelationAttribute.removeAllItems();
    Iterator iter = rel.getAllAttributes().listIterator();
    while (iter.hasNext()) {
      TAttribute att = (TAttribute)iter.next();
      cbRelationAttribute.addItem(att);
    }

      Iterator it = rel.getRoles().listIterator();
    while (it.hasNext()) {
      TRole roleObjectName = (TRole)it.next();

      TObjectType choice = (TObjectType)roleObjectName.getObject();
      cbRelNextObject.addItem(choice);
    }
    cbRelNextObject.removeItem(object);


    TObjectType obsave = (TObjectType)cbRelNextObject.getSelectedItem();
    if (obsave.getRoles().size() == 1) {
    cbRelNextObject.setVisible(false);
    btParOR.setVisible(false);
    }else {
    cbRelNextObject.setVisible(true);
    btParOR.setVisible(true);
    }
    if ((cbRelationAttribute.getSelectedItem()== null) && (obsave.getRoles().size() == 1)) {
     JOptionPane.showMessageDialog(null, "OR URA Cancel,you can't do this formula!",
"Alert Message", JOptionPane.WARNING_MESSAGE);
    DerivationFormula.this.dispose();
    }


    if (attRef != null) {
      TAttributeDefinition att = attRef.getReferredAttributeDefinition();
      cbRelationAttribute.setSelectedItem(att.getOwner());
      cbRelationAttRstamp.setSelectedItem(att);
    }
  }
```

```
 private void updateRelationAttDef() {
  ((DefaultComboBoxModel)cbRelationAttRstamp.getModel()).removeAllElements();
  TAttribute att = (TAttribute)cbRelationAttribute.getSelectedItem();
  Iterator itDef = att.getDefinitions().listIterator();
  while (itDef.hasNext()) {
    TAttributeDefinition attDef = (TAttributeDefinition)itDef.next();
    cbRelationAttRstamp.addItem(attDef);
  }
 }


 private void fillLinkObjectName() { //fill object name in liked object thru relation
    if (cbLinkObjRole.getSelectedItem() == null){ return;};
  TRelationshipType relSel = (TRelationshipType)cbLinkObjRole.getSelectedItem();

  updateLinkObjName(relSel);

 }

 private void updateLinkObjName(TRelationshipType rel) {
     cbLinkObjName.removeAllItems(); //feli
  if (rel == null) return;

  Iterator it = rel.getRoles().listIterator();
  while (it.hasNext()) {
    TRole roleObjName = (TRole)it.next();

    cbLinkObjName.addItem(roleObjName.getObject());
  }
  cbLinkObjName.removeItem(object);

updateAttributes();

 }

private void fillObjectName() { //fill object name in liked object with relation
  if (cbObjectRole.getSelectedItem() == null) {

    return;
  }
       cbObjNextObject.removeAllItems() ;

  Iterator iter = object.getRoles().listIterator();
  while (iter.hasNext()) {
    roleObj = (TRole)iter.next();
    if (roleObj.getObject().equals(cbObjectRole.getSelectedItem())) {


      updateObjectAttributes();
    }
```

```
    }

 TObjectType objectnext = (TObjectType)cbObjectRole.getSelectedItem();
    Iterator iternext = objectnext.getRoles().listIterator();
    while (iternext.hasNext()){
    TRole rolbject = (TRole)iternext.next();
    TRelationshipType relnext = (TRelationshipType)rolbject.getRelation();
    cbObjNextObject.addItem(relnext);
    }
     cbObjNextObject.removeItem(object);


 }

  private void updateObjectAttributes() {
    TObjectType obj = roleObj.getObject();

    if (obj == null) return;
    cbObjectAttribute.removeAllItems();

    Iterator iter = obj.getAllAttributes().listIterator();
    while (iter.hasNext()) {
      cbObjectAttribute.addItem(iter.next());
    }



    if (attRef != null) {
      TAttributeDefinition att = attRef.getReferredAttributeDefinition();
      cbObjectAttribute.setSelectedItem(att.getOwner());
      cbObjectAttRstamp.setSelectedItem(att);
    }


    TObjectType objesave = (TObjectType)cbObjectRole.getSelectedItem();
    if (objesave.getRoles().size() == 1) {
    cbObjNextObject.setVisible(false);
    btParRO.setVisible(false);
    }else {
    cbObjNextObject.setVisible(true);
    btParRO.setVisible(true);
    }
    if ((cbObjectAttribute.getSelectedItem()== null) && (objesave.getRoles().size() == 1)) {
     JOptionPane.showMessageDialog(null, "RO UOA Cancel,you can't do this formula!",
"Alert Message", JOptionPane.WARNING_MESSAGE);
    DerivationFormula.this.dispose();
    }
```

```
}


 private void fillObjectRelation() {


  if (object instanceof TObjectType) {
    Iterator iter = ((TObjectType)object).getRolesInherit().listIterator();
    while (iter.hasNext()) {
      roleObjRel = (TRole)iter.next();
      if (roleObjRel.getObject().equals(cbObjectRole.getSelectedItem())) {//feli

        updateRelationObjects();

      }
    }
    return;
  }
  if (object instanceof TRelationshipType) {
    Iterator iter = ((TRelationshipType)object).getRoles().listIterator();
    while (iter.hasNext()) {
      roleObjRel = (TRole)iter.next();
      if (roleObjRel.getObject().equals(cbObjectRole.getSelectedItem())) {

        updateRelationObjects();

      }
    }
    return;
  }
}

private void updateRelationObjects() {
  TRelationshipType relObj = roleObjRel.getRelation();

  if (relObj == null) return;
  cbLinkObjName.removeAllItems();
  Iterator iter = relObj.getRoles().listIterator();
  while (iter.hasNext()) {
    TRole roleObjLink = (TRole)iter.next();
    cbLinkObjName.addItem(roleObjLink.getObject());
  }
  cbLinkObjName.removeItem(object);
  if (cbLinkObjName.getItemCount() > 0) cbLinkObjName.setSelectedIndex(0);


  if (attRef != null) {
    TAttributeDefinition att = attRef.getReferredAttributeDefinition();
    cbLinkObjName.setSelectedItem(att.getOwner().getOwner());
```

```
  }
 }

 private void updateAttributes() {
  cbLinkObjectAttribute.removeAllItems();



  if (cbLinkObjName.getSelectedItem() == null) return;
  TType obj = (TType)cbLinkObjName.getSelectedItem();
  Iterator iterAtt = obj.getAllAttributes().listIterator();

  while (iterAtt.hasNext()) {

   TAttribute attrib = (TAttribute)iterAtt.next();
   cbLinkObjectAttribute.addItem(attrib);
  }
  updateLinkAttDef();

  if (attRef != null) {

   TAttributeDefinition att = attRef.getReferredAttributeDefinition();
   cbLinkObjectAttribute.setSelectedItem(att.getOwner());
   cbObjectAttRstamp.setSelectedItem(att);
  }


  TObjectType objSel = (TObjectType)cbLinkObjName.getSelectedItem();
   if (objSel.getRoles().size() == 1) {
    btParOO.setVisible(false);
    if (cbLinkObjectAttribute.getSelectedItem() == null) {
    JOptionPane.showMessageDialog(null, "OO UA Cancel,you can't do this formula!",
"Alert Message", JOptionPane.WARNING_MESSAGE);
      DerivationFormula.this.dispose();
    }
   }else {
      btParOO.setVisible(true);  }

 }

 private void updateLinkAttDef() {
  cbLinkObjAttRstamp.removeAllItems();
  if (cbLinkObjectAttribute.getSelectedItem() == null) return;
  TAttribute att = (TAttribute)cbLinkObjectAttribute.getSelectedItem();
  Iterator iterAttDef = att.getDefinitions().listIterator();
  while (iterAttDef.hasNext()) {

   TAttributeDefinition attribDef = (TAttributeDefinition)iterAttDef.next();
   cbLinkObjAttRstamp.addItem(attribDef);
```

```
  }
 }



 /** to add functions in the combo box */
 private void addFunctions(JComboBox cb) {
  cb.addItem("Sum");
  cb.addItem("Avg");
  cb.addItem("Min");
  cb.addItem("Max");
  cb.addItem("Union");
  cb.addItem("Count");
 }

 private void checking() {

  if (object.getAttributeDefinitions().size() < 2) {
   rSelf.setEnabled(false);
  } else {
   rSelf.setEnabled(true);
  }
  //-------------------------linked object thru relation--------------------
     if (object instanceof TRelationshipType) {

    if (((TRelationshipType)object).getRoles().size() == 0) {
     rObject.setEnabled(false);
    } else {

      if (getLinkedAsso() != null) {

       Iterator itLinkObject = getLinkedAsso().listIterator();
       while (itLinkObject.hasNext()) {
         TRole linkRole = (TRole)itLinkObject.next();
         if ((linkRole.getObject().getAllAttributes().size() == 0)&&
(linkRole.getObject().getRoles().size() == 0)) {
           rObject.setEnabled(false);
         } else {
          rObject.setEnabled(true);
         }
       } //end while

    } else {
     rObject.setEnabled(false);
    }
    }


  } //end if
  //-----------------------------------------------------------------------------
```

```
   if (object instanceof TObjectType) {

     if (((TObjectType)object).getRoles().size() == 0) {
       rRelation.setEnabled(false);
     } else {

        if (getLinkedObject() != null) {
       if (getLinkedObject().size() != 0) {

         Iterator itLinkObject = getLinkedObject().listIterator();
         while (itLinkObject.hasNext()) {
           TRole linkRole = (TRole)itLinkObject.next();
           if ((linkRole.getRelation().getAllAttributes().size() != 0)||
(linkRole.getRelation().getRoles().size() > 1)) {
             rRelation.setEnabled(true);
           } else {
             rRelation.setEnabled(false);
           }
         } //end while
       } else {
         rRelation.setEnabled(false);
       }
     } else {
       rRelation.setEnabled(false);
     }
     }




     //-----------------------------------------------------------------------------------------
     if (getLinkedObject() != null) {
       if (getLinkedObject().size() != 0) {

         Iterator itLinkObject = getLinkedObject().listIterator();
         while (itLinkObject.hasNext()) {
           TRole linkRole = (TRole)itLinkObject.next();
           if ((linkRole.getObject().getAllAttributes().size() != 0)||
(linkRole.getRelation().getRoles().size() > 1)) {
             rObject.setEnabled(true);
           } else {
             rObject.setEnabled(false);
           }
         } //end while
       } else {
         rObject.setEnabled(false);
       }
     } else {
```

```
      rObject.setEnabled(false);
    }
  } //end if of Tobject
  //-----------------for selection of radio buttons-------------------------
  if (!rSelf.isEnabled()) {
    if (!rRelation.isEnabled()) {
      if (rObject.isEnabled()) {
        rObject.setSelected(true);
      }
    } else {
      rRelation.setSelected(true);
    }
  }
  if (!rRelation.isEnabled()) {
    if (!rObject.isEnabled()) {
      if (rSelf.isEnabled()) {
        rSelf.setSelected(true);
      }
    } else {
      rObject.setSelected(true);
    }
  }
  if (!rObject.isEnabled()) {
    if (!rSelf.isEnabled()) {
      if (rRelation.isEnabled()) {
        rRelation.setSelected(true);
      }
    } else {
      rSelf.setSelected(true);
    }
  }



  //---------------------------------------------------------------------
} //end checking

private TList getLinkedObject() {
  if (object instanceof TObjectType) {
    Iterator itObject = ((TObjectType)object).getRoles().listIterator();
    while (itObject.hasNext()) {
      TRole rObject = (TRole)itObject.next();
      TRelationshipType rel = rObject.getRelation();

      listRole = rel.getRoles();
      listRole.remove(rObject);

    }
  }
  return listRole;
```

```
  }

   private TList getLinkedAsso() {

       if (object instanceof TRelationshipType) {
     Iterator itObject = ((TRelationshipType)object).getRoles().listIterator();
     while (itObject.hasNext()) {
       TRole rObjectr = (TRole)itObject.next();

       TObjectType obje = rObjectr.getObject();

       listRoler = obje.getRoles();
       listRoler.remove(rObjectr);


     }
    }
    return listRoler;

  }


  private void fillFields() {
    if (object == null) return;


    attRef = derivedAttribute.getAttributeReference();


/*---------------------- Filling the self panel---------------------------------- */

    cbSelfFunction.setSelectedItem(derivedAttribute.getFunction());
    checking();
    Iterator iter = object.getAllAttributes().listIterator();
    while (iter.hasNext()) {
      TAttribute attSelf = (TAttribute)iter.next();

      if (!attSelf.equals(attribute)) {
        cbSelfAttribute.addItem(attSelf);

      } else {
        if (attSelf.getDefinitions().size() > 1) {

          cbSelfAttribute.addItem(attSelf);
        }
      }
    }
    if (attRef != null) {
      TAttributeDefinition attDef = attRef.getReferredAttributeDefinition();
      cbSelfAttribute.setSelectedItem(attDef.getOwner());
```

```
      cbSelfAttributeRstamps.setSelectedItem(attDef);
    }

/*------------------------- Filling the linked relationship panel------------------------- */

   if (object instanceof TObjectType) {
      cbRelationFunction.setSelectedItem(derivedAttribute.getFunction());



      iter = ((TObjectType)object).getRolesInherit().listIterator();

      while (iter.hasNext()) {
        TRole role = (TRole)iter.next();
        TRelationshipType rel = role.getRelation(); //feli
        cbRelationRole.addItem(rel);//feli
      }

      if (attRef != null) {
        TRole roleRef = attRef.getRoleRef();
        //cbRelationRole.setSelectedItem(roleRef);//feli
        cbRelationRole.setSelectedItem(roleRef.getRelation());//feli
      }
    }

/*----------------------------------- Filling the linked object panel ----------------------------*/

   cbObjectFunction.setSelectedItem(derivedAttribute.getFunction());
    iter = object.getRoles().listIterator();

    while (iter.hasNext()) {
      TRole role = (TRole)iter.next();
      TRelationshipType rel = role.getRelation(); //feli
      TObjectType obj = role.getObject();//feli
      cbObjectRole.addItem(obj);//feli

      cbLinkObjRole.addItem(rel);// feli
    } //31.08
    if (attRef != null) {
      TRole roleRef = attRef.getRoleRef();
      cbObjectRole.setSelectedItem(roleRef.getObject());//feli
    } //31.08
    if (derivedAttribute != null) {
      radioType = derivedAttribute.getType();
      if (radioType == SELF) { rSelf.setSelected(true); }
      if (radioType == RELATION) { rRelation.setSelected(true); }
      if (radioType == OBJECT) { rObject.setSelected(true); }
    }
  }
```

```
  private void setType() {

    if (rSelf.isSelected()) { derivedAttribute.setType(SELF); return; }
    if (rRelation.isSelected()) { derivedAttribute.setType(RELATION); return; }
    if (rObject.isSelected()) { derivedAttribute.setType(OBJECT); return; }
  }

  private String converDefinitionToString(TAttributeDefinition attributeDefinition) {
    String s = " ";
    TAttribute owner = attributeDefinition.getOwner();
    TList defs = attributeDefinition.getRepresentations();
    if (defs.size() == 0) {
      if (owner.getOwner().getOwner().getRepresentations().size() == 0)
        s = "No representations";
      else {
        s = "Not yet defined reps";
      }
    } else {
      for (int i = 0; i < defs.size(); i++) {
        s = s + "(" + ((TRepresentation)(defs.get(i))).getName() + ") ";
      }
    }
    return s;
  }

  //-----------------------------------------------------------------------------------------------------------
------
  private boolean setFields() {
    setType();
    if (rSelf.isSelected()) {
      if (cbSelfFunction.getSelectedItem() != null) {
        if (cbSelfAttribute.getSelectedItem() != null) {
          String funct = (String)cbSelfFunction.getSelectedItem();
          TAttributeReference attributeRef = new TAttributeReference(attributeDef);
          TAttributeDefinition attDefinition =
(TAttributeDefinition)cbSelfAttributeRstamps.getSelectedItem();
          derivedAttribute.setFunction(funct);
          if (attDefinition != null) {
            attributeRef.setReferredAttributeDefinition(attDefinition);
            derivedAttribute.setAttributeReference(attributeRef);
            toUpdate.setText(funct + "(" + attDefinition.getOwner().getName() + "." +
                converDefinitionToString(attDefinition) + ")");

          }
        }
      } else {
        JOptionPane.showMessageDialog(null, "Function cannot be null", "Alert Message",
JOptionPane.WARNING_MESSAGE);
        return true;
      }
```

```
    } //end if rself

     if (rRelation.isSelected()) { //for linked relationship

         if (clickor ) {

              TAttribute attributtest = (TAttribute)cbRelationAttribute.getSelectedItem();

             if ( attributtest!= null ) {


      if (cbRelationFunction.getSelectedItem() != null) {
        String funct = (String)cbRelationFunction.getSelectedItem();
        TAttributeDefinition attDefinition =
(TAttributeDefinition)cbRelationAttRstamp.getSelectedItem();
        TAttributeReference attributeRef = new TAttributeReference(attributeDef);
        TRole roleObj = (TRole)cbRelationRole.getSelectedItem();
        TRelationshipType relObj = (TRelationshipType)cbRelationRole.getSelectedItem();
//feli
        derivedAttribute.setFunction(funct);
        attributeRef.setReferredAttributeDefinition(attDefinition);
        try {
          attributeRef.addRoletoPath(roleObj);
        } catch (InvalidElementException exc) {
        }
        derivedAttribute.setAttributeReference(attributeRef);

         toUpdate.setText(funct + "(" + relObj.getName() + "."    +
           attDefinition.getOwner().getName() + "." + converDefinitionToString(attDefinition) +
")");
      } //cbRel

      else {//cbRel
        JOptionPane.showMessageDialog(null, "OR1 Function cannot be null", "Alert
Message", JOptionPane.WARNING_MESSAGE);
        return true;
      }
            }// if atttest
      else {//atttest
       JOptionPane.showMessageDialog(null, "OR You can't do this formula because there is
no attribute!", "Alert Message", JOptionPane.WARNING_MESSAGE);
       return true;}




         }else{ // du if clickor
            TAttribute attributtest = (TAttribute)cbRelationAttribute.getSelectedItem();
```

```
            if ( attributtest!= null ) {


                    TAttributeDefinition attDefinition =
(TAttributeDefinition)cbRelationAttRstamp.getSelectedItem();
        TAttributeReference attributeRef = new TAttributeReference(attributeDef);
        TRole roleObj = (TRole)cbRelationRole.getSelectedItem();
        TRelationshipType relObj = (TRelationshipType)cbRelationRole.getSelectedItem();
//feli
        derivedAttribute.setFunction(fonction);
        attributeRef.setReferredAttributeDefinition(attDefinition);
        try {
          attributeRef.addRoletoPath(roleObj);
        } catch (InvalidElementException exc) {
        }
        derivedAttribute.setAttributeReference(attributeRef);



        TRelationshipType relparc = (TRelationshipType)cbRelationRole.getSelectedItem();
          TObjectType objpar = (TObjectType)cbRelNextObject.getSelectedItem();
          formula = formula + object.getName() +"."+ relparc.getName() +".";


        TAttribute attribu = (TAttribute)cbRelationAttribute.getSelectedItem();
        formula = formula + attribu  +"."+converDefinitionToString(attDefinition) + ")";

        toUpdate.setText(formula);

             object = objectInitial;
            }else{
             JOptionPane.showMessageDialog(null, "OR You can't do this formula because there
is no attribute!", "Alert Message", JOptionPane.WARNING_MESSAGE);
            return true;
            }
         }

    } //end if rrelationship

    if (rObject.isSelected()) { //for linked object type


           if (object instanceof TRelationshipType){

          if (clickro ) {

             TAttribute attributtest = (TAttribute)cbObjectAttribute.getSelectedItem();

            if ( attributtest!= null ) {
```

```
    if (cbObjectFunction.getSelectedItem() != null) {
      String funct = (String)cbObjectFunction.getSelectedItem();
      TAttributeDefinition attDefinition =
(TAttributeDefinition)cbObjectAttRstamp.getSelectedItem();
      TAttributeReference attributeRef = new TAttributeReference(attributeDef);
      TObjectType obj = (TObjectType)cbObjectRole.getSelectedItem(); //feli
      TRole roleObj = (TRole)cbLinkObjRole.getSelectedItem(); //feli
      TRelationshipType relObj = (TRelationshipType)cbLinkObjRole.getSelectedItem();
//feli

      derivedAttribute.setFunction(funct);
      attributeRef.setReferredAttributeDefinition(attDefinition);
      try {
        attributeRef.addRoletoPath(roleObj);
      } catch (InvalidElementException exc) {
        System.out.println("DerivationFormula.checking: Exception adding role ref");
      }
      derivedAttribute.setAttributeReference(attributeRef);

      toUpdate.setText(funct + "(" + obj.getName() + "." +
      attDefinition.getOwner().getName() + "." + converDefinitionToString(attDefinition) +
")");



    } //cbobj

    else {///cbobj
      JOptionPane.showMessageDialog(null, "R01 Function cannot be null", "Alert Message",
JOptionPane.WARNING_MESSAGE);
      return true;
    }
         }// if atttest
    else {//atttest
     JOptionPane.showMessageDialog(null, "You can't do this formula because there is no
attribute!", "Alert Message", JOptionPane.WARNING_MESSAGE);
     return true;}

       }
        else{ // du if clickor
          TAttribute attributtest = (TAttribute)cbObjectAttribute.getSelectedItem();

          if ( attributtest!= null ) {



      TAttributeDefinition attDefinition =
(TAttributeDefinition)cbObjectAttRstamp.getSelectedItem();
```

```
        TAttributeReference attributeRef = new TAttributeReference(attributeDef);
        TObjectType obj = (TObjectType)cbObjectRole.getSelectedItem(); //feli
        TRole roleObj = (TRole)cbLinkObjRole.getSelectedItem(); //feli
        TRelationshipType relObj = (TRelationshipType)cbLinkObjRole.getSelectedItem();
//feli
        derivedAttribute.setFunction(fonction);
        attributeRef.setReferredAttributeDefinition(attDefinition);
        try {
          attributeRef.addRoletoPath(roleObj);
        } catch (InvalidElementException exc) {
          System.out.println("DerivationFormula.checking: Exception adding role ref");
        }
        derivedAttribute.setAttributeReference(attributeRef);


        TRelationshipType relat = (TRelationshipType)cbObjNextObject.getSelectedItem();
          TObjectType objparc = (TObjectType)cbObjectRole.getSelectedItem();
          formula = formula + object.getName() +"."+ objparc.getName() +".";


        TAttribute attribu = (TAttribute)cbObjectAttribute.getSelectedItem();
        formula = formula + attribu  +"."+converDefinitionToString(attDefinition) + ")";

        toUpdate.setText(formula);
            object = objectInitial;

            }else{ //if att test
             JOptionPane.showMessageDialog(null, "You can't do this formula because there is
no attribute!", "Alert Message", JOptionPane.WARNING_MESSAGE);
            return true;
            }
         }// else du if clickor


        } // instance trelationtype




        else // instance trelationtype
      {


              if (click ) {  //&& (object.getRoles().size()  1 )

             TAttribute attributtest = (TAttribute)cbLinkObjectAttribute.getSelectedItem();

            if ( attributtest!= null ) {
```

```
    if (cbObjectFunction.getSelectedItem() != null) {

        String funct = (String)cbObjectFunction.getSelectedItem();
      TAttributeDefinition attDefinition =
(TAttributeDefinition)cbLinkObjAttRstamp.getSelectedItem();
      TAttributeReference attributeRef = new TAttributeReference(attributeDef);
     // TRole roleObj = (TRole)cbObjectRole.getSelectedItem(); //feli
      TRelationshipType relObj = (TRelationshipType)cbLinkObjRole.getSelectedItem();
//feli
      System.out.println("setfields cas obj relObj.getName() " + relObj.getName());
      derivedAttribute.setFunction(funct);
      attributeRef.setReferredAttributeDefinition(attDefinition);
   //   try {
   //     attributeRef.addRoletoPath(roleObj);
   //   } catch (InvalidElementException exc) {
   //     System.out.println("DerivationFormula.checking: Exception adding role ref");
    //  }
      derivedAttribute.setAttributeReference(attributeRef);

       toUpdate.setText(funct + "(" + relObj.getName() + "." +
           attDefinition.getOwner().getOwner().getName() + "." +
attDefinition.getOwner().getName() + "." +
           converDefinitionToString(attDefinition) + ")");


    } //cbobj

    else {///cbobj
      JOptionPane.showMessageDialog(null, "O01 Function cannot be null", "Alert Message",
JOptionPane.WARNING_MESSAGE);
      return true;
    }
         }// if atttest
    else {//atttest
     JOptionPane.showMessageDialog(null, "You can't do this formula because there is no
attribute!", "Alert Message", JOptionPane.WARNING_MESSAGE);
      return true;}

       }
        else{ // du if clickor
          TAttribute attributtest = (TAttribute)cbLinkObjectAttribute.getSelectedItem();

          if ( attributtest!= null ) {



      TAttributeDefinition attDefinition =
(TAttributeDefinition)cbLinkObjAttRstamp.getSelectedItem();
      TAttributeReference attributeRef = new TAttributeReference(attributeDef);
      TRole roleObj = (TRole)cbObjectRole.getSelectedItem(); //feli
```

```
      TRelationshipType relObj = (TRelationshipType)cbLinkObjRole.getSelectedItem();
//feli

      derivedAttribute.setFunction(fonction);
      attributeRef.setReferredAttributeDefinition(attDefinition);
      try {
        attributeRef.addRoletoPath(roleObj);
      } catch (InvalidElementException exc) {
        System.out.println("DerivationFormula.checking: Exception adding role ref");
      }
      derivedAttribute.setAttributeReference(attributeRef);


       TRelationshipType asso = (TRelationshipType)cbLinkObjRole.getSelectedItem();
     TObjectType obje = (TObjectType)cbLinkObjName.getSelectedItem();
     formula = formula + asso.getName()+"."+obje.getName()+".";


      TAttribute attribu = (TAttribute)cbLinkObjectAttribute.getSelectedItem();
      formula = formula + attribu  +"."+converDefinitionToString(attDefinition) + ")";

      toUpdate.setText(formula);

          object = objectInitial;
          }else{ //if att test
           JOptionPane.showMessageDialog(null, "You can't do this formula because there is
no attribute!", "Alert Message", JOptionPane.WARNING_MESSAGE);
          return true;
          }
       }// else du if clickoo


   }  //else du else de instance
          } //if robject
   return false;
  } //end setFields



  private void expandOO() {
   //   if (cbObjectFunction.getSelectedItem() != null) {



     object = (TObjectType)cbLinkObjName.getSelectedItem();
     System.out.println("expand");
     saveRel = (TRelationshipType)cbLinkObjRole.getSelectedItem();
     System.out.println("expand saveRel   " + saveRel.getName());
```

```
    cbLinkObjRole.removeAllItems() ;
    cbLinkObjAttRstamp.removeAllItems() ;
    cbLinkObjectAttribute.removeAllItems() ;
    cbLinkObjName.removeAllItems() ;



    refillFieldsOO();

  }


  private void refillFieldsOO() {
    if (object == null) return;


    attRef = derivedAttribute.getAttributeReference();


    Iterator iter = object.getAllAttributes().listIterator();



//---------------------------------- Filling the linked object panel ----------------------------

   cbObjectFunction.setSelectedItem(derivedAttribute.getFunction());
    iter = object.getRoles().listIterator();

    while (iter.hasNext()) {
      TRole role = (TRole)iter.next();
      TRelationshipType rel = role.getRelation(); //feli
      TObjectType obj = role.getObject();//feli
      cbObjectRole.addItem(obj);//feli
      if ( ! (rel.equals(saveRel))) {

      cbLinkObjRole.addItem(rel);// feli
      }



    }
    if (attRef != null) {
      TRole roleRef = attRef.getRoleRef();
      cbObjectRole.setSelectedItem(roleRef.getObject());//feli
    } //31.08
    if (derivedAttribute != null) {
      radioType = derivedAttribute.getType();
      if (radioType == SELF) { rSelf.setSelected(true); }
      if (radioType == RELATION) { rRelation.setSelected(true); }
      if (radioType == OBJECT) { rObject.setSelected(true); }
```

```
    }
  }
 private void expandOR() {

     object = (TObjectType)cbRelNextObject.getSelectedItem();

     System.out.println("expand");
     saveRel = (TRelationshipType)cbRelationRole.getSelectedItem();
     System.out.println("expand saveRel   " + saveRel.getName());

     cbRelationRole.removeAllItems() ;
     cbRelationAttRstamp.removeAllItems() ;
     cbRelationAttribute.removeAllItems() ;


     refillFieldsOR();

  }

private void refillFieldsOR() {
   if (object == null) return;


   attRef = derivedAttribute.getAttributeReference();



  if (object instanceof TObjectType) {
     cbRelationFunction.setSelectedItem(derivedAttribute.getFunction());


    Iterator iter = ((TObjectType)object).getRolesInherit().listIterator();

     while (iter.hasNext()) {
       TRole role = (TRole)iter.next();
       TRelationshipType rel = role.getRelation(); //feli
         if ( ! (rel.equals(saveRel))) {

     cbRelationRole.addItem(rel);//feli
     }

     }

     if (attRef != null) {
       TRole roleRef = attRef.getRoleRef();
       //cbRelationRole.setSelectedItem(roleRef);//feli
       cbRelationRole.setSelectedItem(roleRef.getRelation());//feli
     }
   }
 if (derivedAttribute != null) {
```

```
    radioType = derivedAttribute.getType();
    if (radioType == SELF) { rSelf.setSelected(true); }
    if (radioType == RELATION) { rRelation.setSelected(true); }
    if (radioType == OBJECT) { rObject.setSelected(true); }
   }
  }

  private void expandRO() {

     object = (TRelationshipType)cbObjNextObject.getSelectedItem();

     saveobj = (TObjectType)cbObjectRole.getSelectedItem();

     cbObjectRole.removeAllItems() ;
     cbObjectAttRstamp.removeAllItems() ;
     cbObjectAttribute.removeAllItems() ;

     refillFieldsRO();

  }

 private void refillFieldsRO() {
    if (object == null) return;


    attRef = derivedAttribute.getAttributeReference();



/*---------------------------------- Filling the linked object panel ----------------------------*/

   cbObjectFunction.setSelectedItem(derivedAttribute.getFunction());
    Iterator iter = object.getRoles().listIterator();

    while (iter.hasNext()) {
      TRole role = (TRole)iter.next();

      TObjectType obj = role.getObject();
       if ( ! (obj.equals(saveobj))) {
      cbObjectRole.addItem(obj);
      }



    }
    if (attRef != null) {
      TRole roleRef = attRef.getRoleRef();
      cbObjectRole.setSelectedItem(roleRef.getObject());
    }
    if (derivedAttribute != null) {
```

```
    radioType = derivedAttribute.getType();
    if (radioType == SELF) { rSelf.setSelected(true); }
    if (radioType == RELATION) { rRelation.setSelected(true); }
    if (radioType == OBJECT) { rObject.setSelected(true); }
   }
  }

  //-------------------------------------------------------------------------------------------------------------
---------------------




  private class CbSelfAttListener implements ItemListener {
   public void itemStateChanged(ItemEvent e) {
    if (e.getStateChange() == ItemEvent.SELECTED) {
     fillSelfAttributeRstamps();
    }
   }
  }


  private class ButtonListener implements ActionListener {
   public void actionPerformed(ActionEvent ev) {
    if (ev.getActionCommand().equals("Ok")) {
     error = setFields();
     if (!error)
      DerivationFormula.this.dispose();
    }
    if (ev.getActionCommand().equals("Apply")) {
     setFields();
    }


    if (ev.getActionCommand().equals("ExpandOO")) {


    if (click) {

        if (cbObjectFunction.getSelectedItem() != null) {
            fonction = (String)cbObjectFunction.getSelectedItem();
            formula = fonction + "(";

           cbObjectFunction.setVisible(false);
           TRelationshipType asso = (TRelationshipType)cbLinkObjRole.getSelectedItem();
    TObjectType obje = (TObjectType)cbLinkObjName.getSelectedItem();
    formula = formula + asso.getName()+"."+obje.getName()+".";

           expandOO();
```

```
            click=false;
          }
          else {
              if (rObject.isSelected()) {

           JOptionPane.showMessageDialog(null, "OO Function can not be null, you will not
change this function later!", "Alert Message", JOptionPane.WARNING_MESSAGE);
              } }

      }else {
         cbObjectFunction.setVisible(false);

            System.out.println("click=false");


       TRelationshipType asso = (TRelationshipType)cbLinkObjRole.getSelectedItem();
      TObjectType obje = (TObjectType)cbLinkObjName.getSelectedItem();
      formula = formula + asso.getName()+"."+obje.getName()+".";



       TObjectType objtest = (TObjectType)cbLinkObjName.getSelectedItem();
       if (objtest.getRoles().size() == 1) {
       JOptionPane.showMessageDialog(null, "OO Ex Cancel,you can't do this formula!",
"Alert Message", JOptionPane.WARNING_MESSAGE);
         DerivationFormula.this.dispose();
       }


         expandOO();
      }

      }

       if (ev.getActionCommand().equals("ExpandOR")) {


      if (clickor) {

          if (cbRelationFunction.getSelectedItem() != null) {
              fonction = (String)cbRelationFunction.getSelectedItem();

              TRelationshipType relpar = (TRelationshipType)cbRelationRole.getSelectedItem();
      formula = fonction + "("+ relpar.getName()+ ".";


             cbRelationFunction.setVisible(false);
             expandOR();
         clickor=false;
          }
```

```
        else {


         JOptionPane.showMessageDialog(null, "Function cannot be null, you will not change
this function later!", "Alert Message", JOptionPane.WARNING_MESSAGE);
             }

    }else {System.out.println("click=false");
       cbRelationFunction.setVisible(false);
       TRelationshipType relparc = (TRelationshipType)cbRelationRole.getSelectedItem();
       TObjectType objpar = (TObjectType)cbRelNextObject.getSelectedItem();
       formula = formula + object.getName() +"."+ relparc.getName() +".";
      System.out.println("formule = " + formula);

      if (objpar.getRoles().size() == 1) {
   JOptionPane.showMessageDialog(null, "OR Ex Cancel,you can't do this formula!", "Alert
Message", JOptionPane.WARNING_MESSAGE);
  DerivationFormula.this.dispose();
  }


      expandOR();
    }

    }




        if (ev.getActionCommand().equals("ExpandRO")) {

    System.out.println("coucou boutonro");
    if (clickro) {
       System.out.println("click=true");
        if (cbObjectFunction.getSelectedItem() != null) {
            fonction = (String)cbObjectFunction.getSelectedItem();
            System.out.println("fonction = " + fonction);
            TObjectType objpar = (TObjectType)cbObjectRole.getSelectedItem();
    formula = fonction + "("+ objpar.getName()+ ".";
    System.out.println("formule = " + formula);

           cbObjectFunction.setVisible(false);
           expandRO();
       clickro=false;
        }
        else {


         JOptionPane.showMessageDialog(null, "Function cannot be null, you will not change
this function later!", "Alert Message", JOptionPane.WARNING_MESSAGE);
```

```
            }

     }else {System.out.println("click=false");
        cbObjectFunction.setVisible(false);
        TRelationshipType relat = (TRelationshipType)cbObjNextObject.getSelectedItem();
        TObjectType objparc = (TObjectType)cbObjectRole.getSelectedItem();
        formula = formula + object.getName() +"."+ objparc.getName() +".";
       System.out.println("formule = " + formula);

       if (relat.getRoles().size() == 1) {
    JOptionPane.showMessageDialog(null, "RO Ex ,you can't do this formula!", "Alert
Message", JOptionPane.WARNING_MESSAGE);
   DerivationFormula.this.dispose();
   }


       expandRO();
     }

     }



     if (ev.getActionCommand().equals("RelationRole")) {
      fillRelationName();
     }
     if (ev.getActionCommand().equals("ObjectRole")) {
      //fillObjectRelation();
      fillObjectName();

     }
     if (ev.getActionCommand().equals("ObjectLinkRole")) {
      fillLinkObjectName();
     }

     if (ev.getActionCommand().equals("RelAttDefinitions")) {
      ((DefaultComboBoxModel)cbRelationAttRstamp.getModel()).removeAllElements();
      TAttribute att = (TAttribute)cbRelationAttribute.getSelectedItem();
      if (att == null) return;
      Iterator iterDef = att.getDefinitions().listIterator();
      while (iterDef.hasNext()) {
       TAttributeDefinition def = (TAttributeDefinition)iterDef.next();
       cbRelationAttRstamp.addItem(def);
      }
     }
     if (ev.getActionCommand().equals("ObjAttDefinitions")) {
      ((DefaultComboBoxModel)cbObjectAttRstamp.getModel()).removeAllElements();
      TAttribute att = (TAttribute)cbObjectAttribute.getSelectedItem();
      if (att == null) return;
      Iterator iterDef = att.getDefinitions().listIterator();
```

```
      while (iterDef.hasNext()) {
        TAttributeDefinition def = (TAttributeDefinition)iterDef.next();
        cbObjectAttRstamp.addItem(def);
      }
    }
    if (ev.getActionCommand().equals("ObjLinkAttDefinitions")) {
      ((DefaultComboBoxModel)cbLinkObjAttRstamp.getModel()).removeAllElements();
      TAttribute att = (TAttribute)cbLinkObjectAttribute.getSelectedItem();
      if (att == null) return;
      Iterator iterDef = att.getDefinitions().listIterator();
      while (iterDef.hasNext()) {
        TAttributeDefinition def = (TAttributeDefinition)iterDef.next();
        cbLinkObjAttRstamp.addItem(def);
      }
    }
    if (ev.getActionCommand().equals("Cancel")) {
      DerivationFormula.this.dispose();
    }
   } //end action performed
  } //end button listener
}
```