Des hyperclasses aux composants pour l'ingénierie des systèmes d'information

Slim Turki

▶ **To cite this version:**

Slim Turki. Des hyperclasses aux composants pour l'ingénierie des systèmes d'information. Interface homme-machine [cs.HC]. Université Joseph-Fourier - Grenoble I; University of Geneva, 2005. Français. NNT : . tel-00067785

HAL Id: tel-00067785

https://theses.hal.science/tel-00067785

Submitted on 9 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GENÈVE
Université de Genève
Faculté des Sciences Économiques et Sociales

Université Joseph Fourrier de Grenoble
École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique

DES HYPERCLASSES AUX COMPOSANTS POUR L'INGÉNIERIE DES SYSTÈMES D'INFORMATION

THÈSE

en co-tutelle présentée par

Slim TURKI

pour l'obtention des grades de

Docteur ès sciences économiques et sociales

Option Systèmes d'Information de l'Université de Genève et de

Docteur en informatique

Option Systèmes d'Information de l'Université Joseph Fourier de Grenoble

Membres du jury

Marie-France BRUANDET, Codirectrice, Professeur, Université Joseph Fourrier de Grenoble

Jean-Pierre GIRAUDIN, Professeur, Université Pierre Mendès France de Grenoble

Dimitri KONSTANTAS, Président du jury, Professeur, Université de Genève

Michel LÉONARD, Directeur, Professeur, Université de Genève

Josiane MOTHE, Rapporteur, Professeur, Institut de Recherche en Informatique de Toulouse

Yves PIGNEUR, Rapporteur, Professeur, Université de Lausanne

Thèse no 592 SES/UNIGE
Genève, 2005

La Faculté des sciences économiques et sociales de l'Université de Genève, sur préavis du jury, a autorisé l'impression de la présente thèse, sans entendre, par là, émettre aucune opinion sur les propositions qui s'y trouvent énoncées et qui n'engagent que la responsabilité de leur auteur.

Genève le 5 juillet 2005

Le doyen de la Faculté des sciences économiques et sociales de l'Université de Genève

Pierre ALLAN

Impressions d'après le manuscrit de l'auteur

© Slim Turki 2005, Tous droits réservés.

Remerciements

« *C'est le temps que tu as perdu pour ta rose qui fait ta rose si importante* »[1]

Ce manuscrit vient conclure plusieurs années passées à l'étude du domaine des systèmes d'information. Ce travail a certes requit beaucoup d'efforts et de sacrifices, mais m'a apporté beaucoup de moments de bonheur et de joie. Avec le temps, ce qui n'était qu'un domaine de recherche s'est transformé en passion personnelle. Au-delà des résultats et des publications, cette expérience m'a beaucoup enrichi : elle fut l'occasion d'apprendre, de découvrir, de réfléchir et d'innover, mais surtout de connaitre et de travailler avec des personnes exceptionnelles. C'est à ces personnes que j'aimerais rendre hommage et dire toute ma gratitude ; sans elles je ne serais pas allé bien loin…

Plus que mon directeur de thèse, le professeur Michel Léonard a était pendant toutes ses années mon maître et mon guide. Il a su me transmettre sa passion pour le domaine des systèmes d'information, et je lui suis profondément reconnaissant. Sans son esprit visionnaire, sa patience, sa disponibilité et son appui infaillible cette thèse n'aurais jamais vu le jour.

Je tiens à adresser mes plus vifs remerciements au professeur Marie-France Bruandet pour m'avoir encadré pendant mon mémoire de DEA, et d'avoir accepté de codiriger ma thèse dans le cadre de la convention de cotutelle. Je remercie également les professeurs Dimitri Konstantas d'avoir présidé le jury, Josiane Mothe et Yves Pigneur d'avoir accepté d'être rapporteurs, et Jean-Pierre Giraudin pour ses enseignements et sa participation au jury. Je salue chaleureusement le docteur Jean-Henry Morin, qui a relu mon mémoire préliminaire et dont les commentaires et les suggestions m'ont été fort utiles.

La collaboration entre l'Université de Genève et le Centre des Technologies de l'Information (CTI) de l'État de Genève a été une expérience très instructive et a permis de faire avancer mes travaux, notamment en les confrontant à la réalité des systèmes d'information en exploitation. J'adresse mes remerciements les plus distingués à M. Jean-Marie Leclerc, directeur du CTI, et Mme Christine Aïdonidis, directrice de la division des architectures de systèmes d'information au CTI, pour les opportunités qu'ils m'ont offertes et pour leur accueil.

Le soutien et la présence de Mme Evelyne Kohl, secrétaire au centre universitaire d'informatique (CUI) de l'Université de Genève, n'ont pu qu'avoir des effets bénéfiques dans l'avancement de mes travaux. Je remercie également Mmes Dorothée Hauser et Amélia Bossard, bibliothécaires au CUI, et Ms. Daniel Agulleiro et Nicolas Mayencourt, Ingénieurs-système du CUI pour leur disponibilité pendant toutes ces années.

[1] Antoine de Saint Exupery, Le petit prince, CHAPITRE XXI.

Je tiens aussi à saluer le travail de Mmes Sylvianne Flammier, Élisabeth Perrin de l'Université Joseph Fourrier et les membres du service des relations internationales de l'Université de Genève pour la mise en place de la convention de cotutelle.

Les membres de la *MATIS Geneva Team*, anciens comme actuels, ont toujours été à mes côtés, pour me soutenir ou pour m'aider. Je garde un excellent souvenir du travail avec Didier Buloz et Christophe Santos sur la plate-forme M7, et du travail avec Thibault Estier et Lina Al-Jadir sur F2 et sur l'évolution des systèmes d'information. J'aimerais remercier Jolita Ralyté pour ses nombreuses corrections d'articles et ses critiques toujours constructives ; Nicolas Arni-Bloch dont j'ai eu l'honneur d'encadrer les travaux de mémoires de licence et de diplôme, et que j'ai le plaisir maintenant d'avoir comme collègue, pour toutes ses interminables et enrichissantes discussions ; Abdelaziz Khadraoui qui sait trouver les mots justes pour aller de l'avant ; Thoa Pham, ma voisine de bureau, pour sa gentillesse et sa disponibilité ; Thang Le Dinh pour ses conseils ; Mehdi Snene et Jorges Pardellas pour leur humour et leur soutien. Je salue aussi les autres étudiants que j'ai encadré pendant ma thèse et dont le travail a permis de faire progresser une partie de ma recherche.

Sans le soutien de ma famille et de mes amis, mon travail n'aurait jamais vu la fin. À ceux qui ont partagé mes angoisses et mes doutes, à ceux qui ont su me réconforter et me soutenir, je dédie ce manuscrit. Je pense évidemment à mes parents, à mes grands-parents, à mes sœurs, à mon frère, à mes oncles et tantes, à Moncef, à Josette, à Ines, à Sinene, à Bouyahia, à Moufida, à Messaoud, à Mohamed-Mohsen Gammoudi et bien sûr à ma Josephine.

À ceux que je n'ai pas cités et qui ont su transformer des instants anodins en précieux moments de pur bonheur, un grand merci.

Résumé

Nous proposons un cadre conceptuel pour l'ingénierie des systèmes d'information (SIs) par composants. Ce cadre est basé sur les concepts d'hyperclasse et de composant de SI.

Le concept d'hyperclasse est une généralisation du concept de classe. Construite à partir d'un ensemble de classes connexe et complet, une hyperclasse permet d'exprimer des concepts que le niveau de classe n'aurait permis d'exprimer, et se comporte comme une classe : elle dispose d'hyperobjets, d'hyperattributs et d'hyperméthodes, équivalents des concepts d'objet, d'attribut et de méthode pour une classe. Un hyperobjet de l'hyperclasse est formé à partir des objets des classes de l'hyperclasse, atteints par navigation, à partir d'une classe particulière de l'hyperclasse qui est sa classe racine, et en suivant un graphe de navigation. Un hyperattribut de l'hyperclasse est un attribut de l'une de ses classes. Une hyperméthode est une méthode associée à l'hyperclasse qui peut avoir comme opérandes des hyperattributs, des hyperobjets, d'autres hyperméthodes de l'hyperclasse, les classes de l'hyperclasse, leurs objets et leurs méthodes de classes. Le concept d'hyperclasse offre une forme d'indépendance entre la structure du SI et ses traitements.

Un composant de SI est une entité autonome et cohérente, dans un modèle de SI, qui regroupe les représentations des espaces informationnel et opérationnel associés à une zone de responsabilité. Il est défini à partir d'une hyperclasse, d'un ensemble de transactions et d'un ensemble de règles d'intégrité du SI. La notion de transaction est associée à une activité productrice ou consommatrice d'informations dans un processus de prise de décision. Les règles d'intégrité sont définies sur un SI pour garantir sa cohérence durant son exploitation.

Les concepts d'hyperclasse et de composant de SI sont munis d'ensembles complets d'opération d'évolution. Ils sont définis indépendamment des méthodes, des langages et des technologies dans lesquelles le SI est implanté.

Le cadre conceptuel que nous proposons permet notamment de prendre en charge les situations de recouvrement de composants de SI, leur évolution, l'évolution du SI et ses répercussions sur ses composants.

Mots-clés

Composant de SI, hyperclasse, évolution, intégration, recouvrement.

Abstract

We propose a conceptual framework for the component based information systems (IS) engineering. This framework is based on the concepts of hyperclass and IS component.

The concept of hyperclass is a generalization of the concept of class. Defined on a connex and complete set of classes, a hyperclass allows expressing concepts that the level of class would not allow to express, and behaves like a class: it has hyperobjects, hyperattributes and hypermethods, equivalents of the concepts of object, attribute and method for a class. A hyperobjet of a hyperclass is built from the objects of the classes of the hyperclass, reached by navigation, starting from a particular class of the hyperclass, which is its class root. A hyperattribute is an attribute of one of the hyperclass classes. A hypermethod is a method of the hyperclass which can have hyperattributes, hyperobjects, others hypermethods, classes of hyperclasses, their objects and methods as operands. The concept of hyperclass offers a type of independence between the structure and the treatments of an IS.

An IS component is an autonomous and coherent entity in an IS model, that gathers the informational and operational aspects associated to a zone of responsibility. It is built from a hyperclass, a set of transactions and a set of integrity rules. A transaction is associated to an activity producing or consuming information in a process of decision-making. The integrity rules are defined to guarantee the coherence of the IS.

Complete sets of evolution operations are defined on the concepts of hyperclass and IS component. They are defined independently of the methods, languages and technologies in which the IS is implemented. The conceptual framework that we propose in particular makes it possible to handle overlap of components, their evolution, the evolution of IS and its effects on its components.

Keywords

IS component, hyperclass, evolution, integration, overlap.

Table des Matières

Table des Figures

INTRODUCTION

INGÉNIERIE DES SYSTÈMES D'INFORMATION

1. INTRODUCTION AUX SYSTÈMES D'INFORMATION

A l'ère de l'information et des technologies de communication, consciemment ou inconsciemment, chacun de nous, est en contact quasi-permanent avec un ou plusieurs systèmes d'information (SIs dans la suite). Les appréciations et les points de vue peuvent varier, mais l'impact des SIs sur la société, l'économie et la vie quotidienne de chacun de nous est indéniablement perceptible.

La « définition usuelle d'un SI ressemblait naguère à ceci : « Le système d'information est l'ensemble des informations formalisables circulant dans l'entreprise et caractérisées par des liens de dépendance, ainsi que des procédures et des moyens nécessaires pour les définir, les rechercher, les formaliser, les conserver, les distribuer ». Mais cette définition n'indique ni à quoi sert le système d'information, ni comment il est construit : elle ignore sa dynamique. Pour décrire celle-ci, il faut distinguer deux faces du système d'information : l'une orientée vers les moyens (« *système informatique* »), l'autre vers les besoins et usages, auxquels la réflexion sur le système d'information donne désormais une place croissante » (Volle, 2001)[2].

Historiquement, les SIs ont débuté avec les outils de gestion. Il était alors question de « robotiser, à l'aide de l'informatique, des tâches fastidieuses et répétitives liées au traitement des données, afin de gagner en rapidité et fiabilité » (Cauvet & Rosenthal-Sabroux, 2001)[3]. Cette « informatisation a offert de nouvelles possibilités, et a induit une nécessaire réorganisation des tâches humaines ainsi qu'une organisation du processus informationnel » (Léonard, 1999)[4]. L'informatisation d'une activité humaine devenait donc plus qu'une simple robotisation d'une tâche ; elle faisait appel à une prise en compte globale de l'information, des traitements, de l'organisation et des aspects humains de l'activité. Afin de prendre en compte cette globalité la notion de système d'information est apparue. Elle peut cependant fortement varier suivant les disciplines (informatique, organisation, management, etc.) qui la travaillent.

Pour nous, un « SI est une construction formée d'informations, de traitements, de règles d'organisation et de ressources humaines et techniques. Les ensembles d'informations sont des représentations partielles de faits qui intéressent l'institution, l'organisation ou l'entreprise. Les traitements constituent des procédés d'acquisitions, de mémorisation, de transformation, de recherche, de présentation et de communication d'informations. Les règles d'organisation régissent l'exécution de traitements informationnels. Les ressources humaines et techniques sont ce qui est requis pour le fonctionnement du SI » (Bodart & Pigneur, 1989)[5]. Les SIs sont donc

---

[2] (Volle, 2001) Volle M., Économie des nouvelles technologies, Chapitre 11, Quelques indications sur les systèmes d'information. Economica, 2001.

[3] (Cauvet & Rosenthal-Sabroux, 2001) Cauvet C., Rosenthal-Sabroux C., Ingénierie des systèmes d'information. Hermes Sciences Publications, 2001. ISBN : 2746202190.

[4] (Léonard, 1999) Léonard M., Introduction aux systèmes d'information, notes de cours, Université de Genève, 1999.

[5] (Bodart & Pigneur, 1989) Bodart F., Pigneur Y., Conception assistée des systèmes d'information. 2ème édition, Masson, Paris, 1989.

formés à partir de représentations partielles de la réalité (informations, traitements, règles) qui sont mises en œuvre dans un espace informatique réalisé grâce à des ressources techniques (ordinateur, réseaux, etc.). Leur fonctionnement n'est cependant possible que grâce à des acteurs humains qui sont en interaction avec le SI.

Nous nous intéressons au domaine des SIs comme supports aux activités humaines dites « métier ». Cette catégorie regroupe les SIs des institutions, des entreprises et des organisations. Dans ce domaine, les SIs ont une influence directe sur les performances, l'efficacité, la compétitivité : la production, la gestion, la diffusion, la sélection et l'utilisation de l'information y prennent de plus en plus d'importance face à la complexité des tâches et des techniques et face à la concurrence continuellement croissante. L'enjeu est aujourd'hui de mettre en place la meilleure solution, aussi bien technologique qu'organisationnelle, pour « concevoir un système d'information numérique qui permette à l'acteur à son poste de travail, dans sa situation, d'obtenir les informations circulantes, de partager ses connaissances tacites et d'accéder aux informations sources de connaissances qui lui sont nécessaires pour comprendre et résoudre les problèmes qu'il rencontre, prendre des décisions, exercer son activité et capitaliser les connaissances produites dans l'exercice de cette activité » (Cauvet & Rosenthal-Sabroux, 2001)[6].

2. INGÉNIERIE DES SYSTÈMES D'INFORMATION

Un « SI est une combinaison de pratiques de travail, d'information, de personnes et de technologies de l'information en vue d'atteindre certains buts » (Alter, 1992)[7]. L'ingénierie des SIs est un domaine complexe où les aspects techniques sont tout aussi importants que les exigences économiques, sociales ou culturelles. La capacité à choisir, modéliser, gérer, partager, retrouver et utiliser des informations et à les faire évoluer au sein d'une organisation humaine et technique, reste les défis de cette discipline.

Quand un SI est déployé dans une partie ou dans l'organisation tout entière, « cette organisation agit dans un environnement qui est influencé par des aspects technologiques, culturels et socio-politiques. En construisant un SI, ces valeurs ont un poids fort car elles déterminent la structure de l'entreprise, sa culture, sa stratégie et ses choix technologiques » (Warboys et al., 1999)[8].

Dans une certaine littérature, certaines conférences scientifiques et souvent dans le discours marketing concernant des produits logiciels, une confusion plane entre *système d'information* et *système informatique*, entre *ingénierie de SI* et *ingénierie logicielle*. « Les SIs

---

[6] (Cauvet & Rosenthal-Sabroux, 2001) Cauvet C., Rosenthal-Sabroux C., Ingénierie des systèmes d'information. Hermes Sciences Publications, 2001. ISBN : 2746202190.

[7] (Alter, 1992) Alter S., Information Systems. Prentice Hall, September 2001, ISBN: 0-13-043242-3

[8] (Warboys & al., 1999) Warboys B., Kawalek P., Robertson I., Greenwood R., Business Information Systems: a Process Approach. McGraw-Hill. 1999.

sont des systèmes socio-techniques » (Mumford & al., 1984)[9] dont la technologie de l'information est une composante essentielle ; ils peuvent être considérés comme « l'intégration d'une infrastructure et de divers systèmes, qui se servent de cette infrastructure » (Galliers, 1994)[10]. Les « SIs sont significatifs seulement quand ils sont considérés dans un contexte » (Ozkan, 2004)[11] et « la distinction principale entre un système logiciel et un SI est que le logiciel est limité au processus de développement d'un système logiciel, tandis qu'un SI est considéré dans le contexte organisationnel dans lequel le logiciel est employé » (Von Hellens, 1997)[12].

L'ingénierie de SI et l'ingénierie logicielle sont très différentes ; ni les traditions, ni les repères ou les références ni même les objectifs n'y sont les mêmes. Même si le paradigme orienté objet est de plus en plus adopté comme moyen privilégié d'expression et de communication, chacune des deux ingénieries garde ses spécificités. Les Français, les Suisses romands, les Wallons et les Québécois ont beau tous parler le français, il n'en demeure pas que se sont des peuples (et des pays) très différents[13].

Pour structurer notre étude sur l'ingénierie des SIs, nous adoptons un cadre de référence construit sur trois pôles : (i) les *activités métier,* (ii) l'*informatique* et (iii) *l'information*.

---

[9] (Mumford & al., 1984) Mumford E., Hirschheim R., Fitzgerald G., Wood-Harper T., Research Methods in Information Systems. Proc. of the IFIP WG 8.2 Colloquim, Manchester Business School, September 1984.

[10] (Galliers, 1994) Galliers R.D., Relevance and Rigor in Information Systems Research: Community, in Business Process Reengineering - Information Systems Opportunities and Challenges. Proc. of the IFIP TC8 Open Conference on BPR, 1994.Elsevier, Holland.

[11] (Ozkan, 2004) Ozkan S., Information Systems Quality versus Software Quality: A Process Based Assessment Model within the context of Organisation. Proc. of AIM'2004, May 2004, INT, Evry, Paris, France.

[12] (Von Hellens, 1997) Von Hellens L.A., Information systems quality versus software quality: a discussion from a managerial, an organisational and an engineering viewpoint. Information and Software Technology. 39, P. 801-808.

[13] *Redde Caesari quae sunt Caesaris* : idée originale de M. Léonard.

2.1. CADRE DE RÉFÉRENCE POUR L'INGÉNIERIE DES SIS

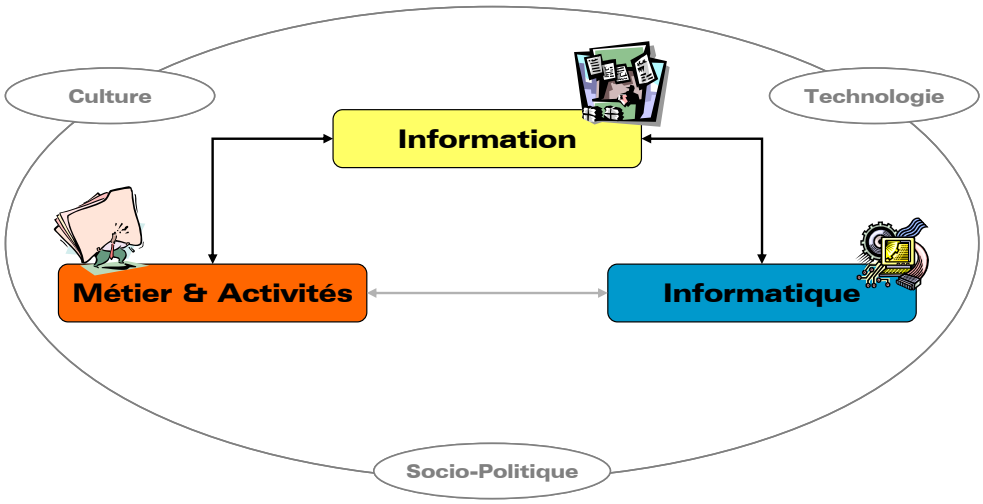

Figure 1 : Cadre de référence pour l'ingénierie des systèmes d'information

La Figure 1 représente le cadre de référence que nous adoptons pour notre étude sur l'ingénierie des SIs. Ce cadre est construit autour de trois pôles : (i) les *activités métier,* (ii) l'*informatique* et (iii) *l'information*.

2.1.1. Plate-forme informatique

« Un système d'information utilise le plus souvent l'informatique pour remplir adéquatement son rôle. Un tel système permet de transformer des données à l'état brut en informations utiles pour les gestionnaires. Cependant, il ne faut pas confondre système d'information et technologie informatique. L'informatique n'est qu'un outil très utile permettant d'exploiter un système d'information (..). Les moyens de l'informatique sont la puissance de calcul (mesurée en MIPS), la mémoire (mesurée en octets), l'intelligence *programmée* (logiciels et applications), les réseaux (largeur de bande, protocoles), les outils (langages, ateliers de génie logiciel, passerelles, middleware), etc. » (Volle, 2001)[14]. C'est ainsi qu'au niveau informatique d'un SI qu'est définie l'architecture logicielle et que sont fixés les choix technologiques.

La plate-forme informatique est définie comme « un ensemble organisé d'objets techniques – matériels, logiciels, applicatifs – dont la mise en œuvre réalise l'infrastructure d'un système d'information ». (Morley & al., 2000)[15].

Traditionnellement, les responsables de plate-forme informatique de SI ont à leur charge le choix et la gestion de l'architecture et du parc informatique, matériel et logiciel, la gestion de la maintenance, l'administration du système du SI, et bien sûr la réalisation informatique.

---

[14] (Volle, 2001) Volle M., Économie des nouvelles technologies, Chapitre 11, Quelques indications sur les systèmes d'information. Economica, 2001.

[15] (Morley & al., 2000) Morley C., Hugues J., Leblanc B., UML pour l'analyse d'un système d'information. Dunod Informatiques, 2000.

La qualité d'un produit informatique (logiciel) est évaluée en fonction de sa *fiabilité*, ses *performances* et son niveau de *sécurité*.

2.1.2. Les activités métier

Un système d'information ne peut être réduit à sa seule composante informatique. « Gérer le SI de l'entreprise c'est avant tout avoir à faire avec des Hommes » (Lesca & Lesca, 1995)[16] : ce sont des personnes qui sont à l'origine des informations, qui les font circuler, les traitent et les utilisent et qui font partie intégrante du SI et peu de membres de l'entreprise peuvent en être exclus.

Le SI doit supporter efficacement les activités humaines. Ces activités, pour être efficientes, nécessitent la plupart du temps une interaction avec le SI et « les personnes qui les assument ne sont pas que de simples utilisateurs informatiques : leurs responsabilités, leur façon d'appréhender leurs activités, leur propre efficacité dépendent du SI » (Léonard 2003)[17]. Le développement de SI comme support aux activités métiers, et pas seulement comme automatisation de ces activités, nécessite de prendre en considération le contexte de travail de ces personnes.

Un *métier* est un « secteur d'activité dans lequel une entreprise a acquis un savoir-faire, une habileté technique résultant de l'expérience ou d'une longue pratique ». Une *activité* est « l'action d'une personne ou d'une entreprise dans un domaine défini ; domaine dans lequel s'exerce cette action ». Un *domaine* est un « champ d'activité ou l'étendue de sa compétence » [18]. Un domaine peut être une activité, une problématique, une discipline, etc.

Le pôle des *activités métier* de notre cadre de référence décrit les activités supportées par le SI. Ce niveau est généralement considéré sous une (i) *dimension procédurale*, et sous une (ii) *dimension organisationnelle.* La description procédurale se base sur les concepts de *processus*, d'*activité* et de *tâche.* La description organisationnelle s'appuie sur les concepts de *rôle* et de *responsabilité*.

2.1.2.1. Dimension procédurale

Les *processus* de l'entreprise ou de toute autre organisation donnent sens à ses structures : ils organisent le déroulement des activités et formalisent les responsabilités. « En s'appuyant sur ces processus, le travail peut être rationalisé, optimisé et la suite des activités peut être fixée et standardisée » (Probst & al., 1997)[19]. Un processus est un « ensemble actif de phénomènes,

---

[16] (Lesca & Lesca, 1995) Lesca H., Lesca E., Gestion de l'Information : qualité de l'information et performances de l'entreprise. Litec, 1995,

[17] (Léonard, 2003) Léonard M., IS engineering getting out of classical system engineering. Proc. ICEIS'2003. April 2003, Angers, France.

[18] Le Petit Larousse illustré 1999. © Larousse, 1998.

[19] (Probst & al., 1997) Probst G., Mercier J-Y., Bruggimann O., Rakotobarison A., Organisation et Management. Éditions d'Organisation, Paris, 1997.

organisé dans le temps » et une « façon de procéder, une suite ordonnée d'opérations » (Le Petit Robert). C'est « un enchaînement ordonné d'opérations, organisé dans le temps ».

« Chaque processus est reparti en *activités*, et chaque activité est repartie en *tâches* » (Scholz-Reiter & Stickel, 1996)[20]. Ces processus sont des processus artificiels qui existent pour atteindre le but de l'organisation ou pour changer l'état du monde afin de répondre à des besoins. « Pour arriver à une satisfaction de ces besoins, il faut une certaine structuration des activités des membres qui conduisent à l'obtention des objectifs d'une organisation » (Warboys & al., 1999)[21]. « Chaque processus a un point de début et aussi un point de fin. De la même manière, chaque tâche d'un processus est initialisée et terminée par une action prise qui déclenche au niveau informationnel un changement d'état » (Scholz-Reiter & Stickel, 1996).

Une *activité* est une « unité de travail qui représente un état du processus qui essaie d'atteindre partiellement l'objectif de l'entreprise » (Scholz-Reiter & Stickel, 1996). Une « activité est subdivisée en entités plus petites, effectuées par un acteur, qui puissent être complétées immédiatement » (Scholz-Reiter & Stickel, 1996). (Warboys & al., 1999) définissent l'activité comme un comportement d'intérêt qui provoque un changement d'état. Les activités représentent un sous-ensemble d'un processus spécifique, sans pour autant représenter des éléments atomiques.

Une *tâche* est une subdivision d'une activité. C'est un élément atomique qui ne peut pas être divisé. Elle tire son origine de l'activité d'un processus. Selon (Nurcan, 1998)[22] (cité dans (Bui, 2002)[23]), une tâche est une suite d'actions primitives exécutées par un rôle individuel qui attribue des droits et devoirs à la personne. Il y a donc toujours des responsabilités qui correspondent à une tâche. Par ailleurs, l'exécution des tâches implique une interaction avec les ressources informationnelles pour baser des décisions et pouvoir prendre des initiatives.

2.1.2.2. Dimension organisationnelle

La dimension organisationnelle d'un SI concerne les conditions de son fonctionnement dans l'organisation à travers les *rôles* concernées par la collecte, la communication, le traitement, l'utilisation des informations ou encore les rapports entre la structure de l'information et la structure de l'organisation. Les organisations diffèrent dans leurs structures pour diverses raisons, notamment quant à leur besoin d'échange d'information ; la structure de l'organisation dépend, de manière contingente des besoins de traitement de l'information et de communication.

---

[20] (Scholz-Reiter & Stickel, 1996) Scholz-Reiter B., Stickel E., Business Process Modelling. Springer Verlag Berlin, Heidelberg, 1996.

[21] (Warboys & al., 1999) Warboys B., Kawalek P., Robertson I., Greenwood R., Business Information Systems: a Process Approach. McGraw-Hill. 1999.

[22] (Nurcan, 1998) Nurcan, S., Main Concepts for Cooperative Work Place Analysis, Proceedings of Telecooperation Conference of the 15th, IFIP World Computer Congress 1998.

[23] (Bui, 2002) Bui Minh Tu D., Élaboration et Intégration des Spécifications de Workflow Coopératif dans la Conception des Systèmes d'Information, Mémoire de DEA en Systèmes d'information, Université de Genève, 2002.

Le concept de « rôle fait l'association entre un agent (personne) et un objet ou une tâche » (Warboys et al., 1999)[24]. Un « rôle attribue certaines caractéristiques comportementales à une personne en même temps qu'à des normes, des descriptions et des standards » (Biddle & Thomas dans Warboys & al., 1999). Un rôle a donc deux aspects : celui de la responsabilité, c'est à dire l'attribution des droits, devoirs et pouvoirs (*rôle organisationnel*), et celui de l'action (*rôle opérationnel*) (Warboys et al., 1999).

2.1.2.3. Zone de responsabilité

Les responsabilités organisationnelles liées à l'exercice d'un métier ou d'une fonction dans une entreprise traitent de l'information (champ informationnel) en effectuant leurs activités (champ opérationnel). Nous appelons *zone de responsabilité* (Figure 2) « l'ensemble des activités qu'une responsabilité organisationnelle peut effectuer en traitant de l'information » (Léonard & Parchet, 1998)[25].

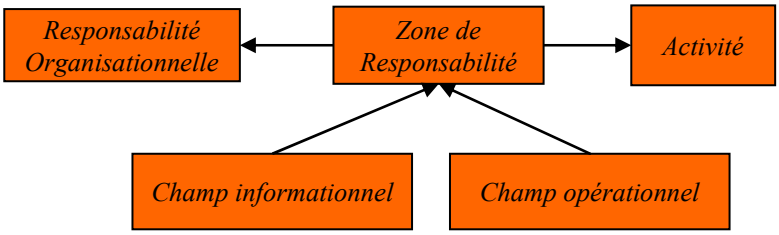

Figure 2 : Zone de responsabilité

2.1.3. Information

L'information est la ressource fondamentale et la matière première dans un SI. Une information dans un SI est tout élément de connaissance susceptible d'être codé pour être conservé, traité ou communiqué.

Le *niveau informationnel* de notre cadre de référence offre une plate-forme intermédiaire entre le niveau des activités et le niveau informatique. Un des intérêts du niveau informationnel est de préserver les spécificités de chacun des deux espaces en mettant en valeur une ressource qui leur est commune : l'information. L'ingénierie de SI exige dès lors le recours à des modèles qui favorisent la rigueur et la communicabilité entre les différents acteurs : informaticiens et gestionnaires et responsables de SI.

La description du niveau informationnel est une étape intermédiaire et incontournable pour passer du niveau des activités vers le niveau informatique et vis-versa. Elle permet de décliner un concept métier en information avant de le traduire en objet(s) informatique(s). Il ne s'agit pas de modéliser le métier ou l'informatique, mais le noyau du SI, c'est-à-dire la surface

---

[24] (Warboys & al., 1999) Warboys B., Kawalek P., Robertson I., Greenwood R., Business Information Systems: a Process Approach. McGraw-Hill. 1999.

[25] (Léonard & Parchet, 1998) Léonard M., Parchet O., Étude des situations de recouvrements de l'information pour la conception des SI. Actes d'INFORSID'98, Mai 1998, Montpellier, France.

informationnelle sur laquelle se construit le SI : « le souci de l'information vient d'abord et que la technologie vient ensuite » (Lesca & Lesca, 1995)[26].

Le niveau informationnel du SI prend la forme d'une ou de plusieurs modèles que nous appelons dans la suite *modèle de SI*.

2.2. MODÈLE DE SYSTÈME D'INFORMATION

Un modèle de SI est une représentation formelle ou semi-formelle représentant la structuration de l'information dans le SI et son évolution au cours du temps. Ce niveau est décrit à travers des diagrammes qui représentent la structure du SI (aspects statiques) et son comportement (aspects dynamiques et règles d'intégrité).

Dans ce travail, nous nous intéressons au niveau *informationnel* et à ses interactions avec les niveaux des activités et de l'informatique. Nous nous intéressons également à la représentation et à la modélisation de l'information dans les SIs. Un modèle de SI demeure une composante essentielle du « développement durable » du SI de l'entreprise ou de l'institution : bien construit, il constitue une démarche précieuse pour limiter les risques dans le développement d'un SI.

En écologie, un développement durable est une « développement qui répond aux besoins du présent sans compromettre la capacité des générations futures de répondre aux leurs »[27]. En ingénierie des SIs, une politique de développement durable est une politique de développement de SIs qui répondent non seulement aux besoins actuels identifiés de l'entreprise ou de l'institution, mais aussi des SIs capables d'évoluer et de s'ouvrir à d'autres SIs pour maintenir leur alignement avec les activités qu'ils supportent. Les modèles de SI garantissent d'une part la traçabilité des évolutions que subit le SI, et d'autre part d'appréhender de nouvelles situations qu'aurait à supporter le SI.

2.2.1. Modèle

Un modèle est une « structure abstraite et formalisée utilisée pour rendre compte d'un ensemble de phénomènes qui possèdent entre eux certaines relations »[28]. Son caractère abstrait facilite la compréhension du système étudié : il réduit sa complexité, permet de le simuler, le représente et reproduit ses comportements. Concrètement, un modèle réduit (décompose) la réalité, dans le but de disposer d'éléments de travail exploitables par des moyens mathématiques ou informatiques.

---

[26] (Lesca & Lesca, 1995) Lesca H., Lesca E., Gestion de l'Information : qualité de l'information et performances de l'entreprise. Litec, 1995,

[27] Définition adoptée par l'Union internationale pour la conservation de la nature (UICN) et reprise dans le rapport Notre avenir à tous de la Commission Brundtland, cité dans *GrandDictionnaire.com*

[28] Le Petit Larousse illustré 1999. © Larousse, 1998.

2.2.2. Modèle de SI

Un modèle de SI est une représentation des concepts du domaine supporté par le SI et de leur organisation qui décrit de manière formelle les informations qui sont manipulées.

Un concept est une « représentation générale et abstraite d'un objet ou d'un ensemble d'objets» [29]. C'est une « représentation d'un aspect de la réalité, isolé par l'esprit, et une unité de pensée constituée d'un ensemble de caractères attribués à un objet ou à une classe d'objets »[30].

Dans un modèle de SI, il ne s'agit pas de modéliser les objets du « monde réel » mais l'information utilisée et échangée pour pouvoir accomplir les activités du métier. Les objets informationnels doivent en même temps pouvoir être dérivés en objets informatiques. Un modèle informationnel représente la manière dont les informations sont structurées et la manière avec laquelle elles évoluent. Un modèle de SI est une représentation non ambiguë (i) des activités humaines et institutionnelles dans leurs dépendances avec le SI, (ii) des différents composants d'un SI et de leurs interactions, (iii) et des interactions entre ce monde des activités et celui du SI.

Dans ce travail, nous adoptons le paradigme orienté objet pour la modélisation des SIs. « Les méthodes objets introduisent la notion d'objet regroupant les structures de données, les contrôles et les traitements et proposent des spécifications globales d'un SI par l'intermédiaire de spécifications complémentaires statiques et dynamiques » (Giraudin, 2001)[31]. Un modèle de SI prend la forme de plusieurs diagrammes représentant la structure du SI (aspects statiques) et son comportement (aspects dynamiques et règles d'intégrité).

2.2.2.1. Aspects statiques

Les diagrammes de classes constituent une expression semi-formelle des propriétés statiques du SI. Ils décrivent les classes du SI, leurs attributs, leurs identifiants, les méthodes des classes, les liens qui relient ces classes.

2.2.2.2. Aspects dynamiques

L'état d'un objet d'une classe évolue suite à l'exécution d'opérations (création suppression, mise-à-jour) ou de méthodes de classes. Les modèles dynamiques ont pour objectif de décrire les règles d'évolution des objets au cours du temps. Dans les méthodes objets, la dynamique des objets est traditionnellement décrite à l'aide de diagrammes d'états/transitions (dérivés des machines d'états finis) ou des réseaux de Petri.

---

[29] Le Petit Larousse illustré 1999. © Larousse, 1998.

[30] Office de la Langue Française, 2000, *GrandDictionnaire.com*

[31] (Giraudin, 2001) Giraudin, J-P., Chabre-Peccoud, M., Rieu, D., Saint-Marcel, Ch., Ingénierie des systèmes d'information, Modèles de spécification pour l'ingénierie des systèmes d'information. P 115-148, ISBN: 2-7462-0219-0, Hermes Sc. Europe Ltd, 2001.

Nous adoptons les graphes de *Gavroche* pour la modélisation des aspects dynamiques d'un SI. Ce formalisme est issu des travaux de (Léonard, 1999)[32] et (Pham, 2003)[33].

2.2.2.3. Aspects réglementaires

Les règles d'intégrité (RIs) sont des constituants essentiels d'un SI. Leur rôle est de préserver sa cohérence et sa consistance durant son exploitation.

Une RI est une condition qui doit être validée par l'information stockée dans le SI. Elle est définie sur une ou plusieurs classes du diagramme de classes, et sa validation s'effectue de manière algorithmique. Lorsque l'ensemble des RIs est valide, le SI est dit *cohérent*.

2.2.3. Méta-modèle pour la modélisation des SIs

Un méta-modèle est un modèle qui définit le langage pour l'expression d'un modèle. Il définit : (i) les éléments du modèle (les concepts manipulés), et (ii) la sémantique de ces éléments (leur définition et le sens de leur utilisation).

Un modèle est un outil de communication permettant à plusieurs acteurs de confronter leur vision du système et de sa retranscription. Il est essentiel que les modèles utilisés respectent au mieux des normes reconnues ou, à tout le moins, des standards partagés par les divers acteurs. La définition d'un méta-modèle fixe un langage commun pour la production et l'exploitation de modèles de SI.

2.2.4. Évolution de SI

Un modèle de SI n'est que le reflet d'une perception particulière du monde des activités avec des besoins particuliers de stockage, de recherche et de manipulation des informations. Une meilleure connaissance du domaine peut faire changer et évoluer cette perception. Ainsi, le modèle peut évoluer au cours du temps pour « prendre en compte de nouveaux besoins, s'adapter à des changements organisationnels, corriger des erreurs de conception ou améliorer les spécifications » (Al-Jadir 1997)[34]. Pratiquement, « un système et une application orientés objet dits évolutifs doivent garantir, même dans le cas de perturbations, la cohérence, la consistance et la pertinence des informations prises en compte et ce tout au long de leurs cycle de vie. Le processus doit être harmonieux et doit éviter au maximum l'introduction d'incohérences et une refonte inutile desdits systèmes et/ou applications » (Oussalah, 1999)[35].

---

[32] (Léonard, 1999) Léonard M., M7 : approche évolutive des systèmes d'information. Proc. Inforsid'99. La Garde, France, mai 1999.

[33] (Pham, 2003) Pham T.-T.-T., Intégration des aspects statiques et dynamiques des Systèmes d'Information. Mémoire préliminaire, Université de Genève, 2003.

[34] (Al-Jadir 1997) Al-Jadir L., Evolution-oriented database systems, PhD thesis, Uni. de Genève, 1997.

[35] (Oussalah, 1999) Génie Objet : analyse et conception de l'évolution, Sous la direction de Chabane Oussalah, HERMES Sciences Publications, Paris 1999. ISBN 2-7462-0029-5.

Sauf cas particulier, un modèle de SI ne peut être complet. Un modèle de SI peut être complété à la suite de l'élargissement du domaine du SI. Il peut être modifié du fait de modifications apparues dans l'environnement de l'institution ou de l'entreprise. « La nécessité de l'approche évolutive provient d'un regard particulier porté sur le domaine scientifique des SI qui privilégie l'adéquation des mondes vivants et artificiels. Les difficultés du passage de l'un à l'autre montrent combien sont réductrices les approches construites autour d'une simple traduction du vivant en artificiel ainsi que celles de l'informatique triomphante proposant la seule démarche du fait accompli. En fait de telles approches nient les difficultés de l'évolution des SI, voire même de la nécessité de leur étude » (Léonard, 1999). « Prendre en compte l'évolution de schémas dans un système à objets, c'est admettre que celui-ci peut évoluer dans le temps. C'est donc reconnaître que la modélisation d'un système ou d'une application peut changer, soit parce qu'elle ne correspond plus à celle utilisée jusqu'à alors, soit parce que des erreurs y ont été découvertes. Cette remise en cause implique souvent des mises à jour difficiles et coûteuses. Elles sont difficiles car elles ne doivent pas introduire d'erreurs dans le nouveau schéma, et parce qu'elles doivent préserver le fonctionnement de l'application dans les parties qui ne sont pas affectées par les modifications » (Nguyen, 1997)[36].

2.2.5. Modularité, interopérabilité et transversalité dans les SIs

La *modularité* est la « caractéristique d'un système qui peut être subdivisé en sous-systèmes qui interagissent entre-deux » (Computer Language Company Inc.)[37]. L'*interopérabilité* est « l'aptitude de deux ou plusieurs systèmes ou composants d'échanger de l'information et d'exploiter l'information échangée » (IEEE, 1990)[38] (WordNet, 2.0)[39].

« L'historique de la profession informatique montre assez bien que l'on est passé progressivement de l'univers des applications autonomes à celui des applications imbriquées, du traitement, du traitement automatisé des procédures administratives ou industrielles à l'aide à la décision, du traitement par lot au traitement transactionnel, des techniques hiérarchiques aux techniques relationnelles. Tout cela augmente la complexité des systèmes et, tant du point de vue technique que fonctionnel, ces systèmes ne sont plus à la dimension d'un seul individu, même talentueux… » (Bouchy, 1994)[40]. « Les SI actuels couvrent des domaines d'activités de plus en plus étendus et complexes, de sorte qu'il est devenu difficile de les appréhender dans leur globalité que ce soit du point de vue de leur conception, de leur utilisation ou même de leur administration. Dans ce contexte, l'idée du partitionnement des composants d'un SI en plusieurs sous-ensembles plus facilement appréhendables s'est largement développée tant dans les

---

[36] (Nguyen, 1997) Nguyen G. T., Objets et évolution, Chapitre 6, Ingénierie Objet : Objets, techniques, concepts (Sous la direction de Oussalah C.), P205-232. InterEditions, Collection Informatiques, 1997. ISBN : 2729606424.

[37] (Computer Language Company Inc.) Computer Language Company Inc. *Computerlanguage.com*

[38] (IEEE, 1990) IEEE Standard Computer Dictionary.

[39] (WordNet, 2.0) WordNet 2.0 by Princeton University.

[40] (Bouchy, 1994) Bouchy S., L'ingénierie des systèmes d'information évolutifs. Eyrolles en 1994 ISBN 2-212-08790-X

méthodologies de conception de SI que dans les technologies de développement des SI » (Léonard & Parchet, 1998)[41]. Un SI peut être considéré comme un « jeu de composants en corrélation qui collectent ou récupèrent, traitent, stockent et distribuent l'information pour soutenir des processus décisionnel, la coordination, le contrôle, l'analyse et la visualisation dans une organisation » (Laudon & Laudon, 2001)[42].

Par ailleurs, l'organisation des activités d'une entreprise ne peut pas être considérée comme la simple juxtaposition de grandes fonctions (exemple : vente, approvisionnement, etc.). À la vision verticale classique se superpose une vision transversale, où les activités sont aussi bien appréhendées dans leurs aspects intra-fonctionnels que trans-fonctionnels.

2.2.6. Qualité d'une modélisation informationnelle

Une modélisation de SI demande pour être pertinente un modèle qui permette de :

1. décrire des situations de manière non ambiguë,
2. d'établir la fidélité entre un modèle de SI et sa réalisation informatique et
3. d'établir la fidélité entre un modèle de SI et l'organisation de l'entreprise ou de l'institution.

La fidélité est la propriété de « celui qui ne s'écarte pas du modèle »[43]. La fidélité de la réalisation informatique par rapport à un modèle informationnel nécessite que ce modèle soit de qualité, c'est-à-dire, *rigoureux*, *non-ambigu*, et *cohérent.* La *fidélité* aux modèles informationnels dans le développement et la maintenance est un critère de qualité de la réalisation informatique du SI.

3. DÉMARCHE ET APPORTS

Notre démarche vise à construire un cadre conceptuel pour l'ingénierie des SIs par composants. Ce cadre, globalement décrit dans la Figure 3, vise la construction de modélisations informationnelles pertinentes et évolutives, et est basé sur les concepts d'*hyperclasse* et de *composant de SI*.

Un modèle de SI prend la forme de plusieurs diagrammes représentant la structure du SI (aspects statiques) et son comportement (aspects dynamiques et règles d'intégrité).

---

[41] (Léonard & Parchet, 1998) Léonard M., Parchet O., Étude des situations de recouvrements de l'information pour la conception des SI. Actes d'INFORSID'98, Mai 1998, Montpellier, France.

[42] (Laudon & Laudon, 2001) Laudon K. C., Laudon J. P., Management Information Systems. 7th Ed. Prentice Hall, 2001.

[43] Le Petit Larousse illustré 1999. © Larousse, 1998.

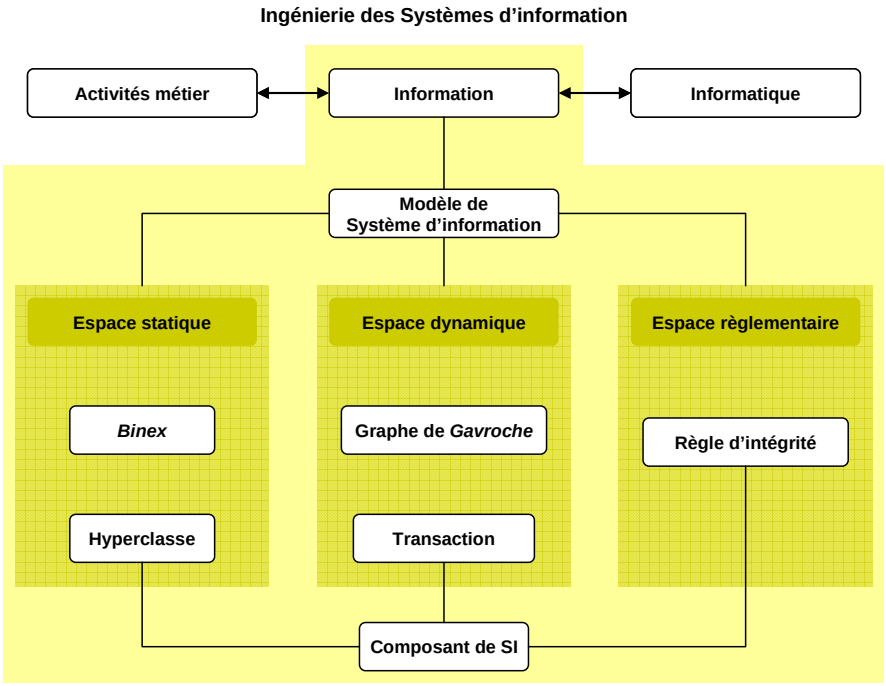

Figure 3 : Organisation générale des concepts traités

Pour la modélisation des aspects statiques, nous utilisons le méta-modèle *binex*. *Binex* est un modèle orienté objet *binaire* et *existentiel.* Il a la particularité de n'autoriser que des liens binaires existentiels et de spécialisation-généralisation entre ses classes.

Pour la modélisation des aspects dynamiques, nous adoptons le formalisme des *graphes de Gavroche*. Un graphe de Gavroche est un graphe biparti où les nœuds correspondent un à un à des *classes* de l'espace statique et les étoiles à des *transactions*. Une transaction dans les graphes de Gavroche est inspirée des concepts de « *transaction commerciale* » et de « *transaction bancaire* » et est associée à une activité productrice ou consommatrice d'informations dans un processus de prise de décision. Les aspects réglementaires sont décrits à travers un ensemble de règles d'intégrité, dont le rôle est de préserver la cohérence et la consistance du SI durant son exploitation.

Dans l'espace statique d'un modèle de SI, l'utilisation des classes s'avère souvent insuffisante pour prendre en compte tous les concepts du domaine. Le concept d'*hyperclasse* est une généralisation du concept de classe. Construite à partir d'un sous-ensemble de classes, une hyperclasse décrit le champ informationnel d'une zone de responsabilité du SI. Au concept d'hyperclasse, sont associés les concepts d'hyperobjet, d'hyperattribut et d'hyperméthode, respectivement équivalent aux concepts d'objet, d'attribut et de méthode pour une classe. Un hyperobjet est une instance de l'hyperclasse, construite par navigation entre les objets des classes de l'hyperclasse. Un hyperattribut est un attribut d'une classe de l'hyperclasse. Une hyperméthode est une méthode associée à l'hyperclasse.

Le concept d'hyperclasse introduit une forme de modularité dans les modèles statiques de SI, permettant de gérer le partage des responsabilités autour de la gestion des informations contenues dans un SI. Il offre des mécanismes d'accès aux données basés sur la navigation,

permettant de réduire la connaissance nécessaire des détails de la modélisation autant pour des utilisateurs que pour des concepteurs, développeurs, etc. du SI.

Un composant de SI est une structure construite à partir d'une hyperclasse, d'un ensemble de transactions et d'un ensemble de règles d'intégrités d'un modèle de SI. Il intègre ainsi des propriétés statiques, des propriétés dynamiques et des propriétés réglementaires du SI ou d'une partie du SI en un même concept. Un composant de SI représente les espaces informationnel et opérationnel dont a besoin une zone de responsabilité pour exercer convenablement ses activités.

En permettant aux zones de responsabilité d'assurer localement leurs activités, et leur collaboration avec d'autres zones de responsabilité du SI, le concept de composant de SI rend possibles l'étude et la prise en charge, de manière formelle, les problématiques de recouvrement et de partage de l'information dans un SI.

Tous les modèles et méta-modèles que nous utilisons dans ce travail sont munis d'ensembles complets d'opérations d'évolution, permettant de créer, de supprimer ou de mettre à jour chacun de leurs concepts.

Le premier chapitre est consacré à la présentation de *binex*. Les deuxièmes et troisièmes chapitres sont respectivement consacrés aux hyperclasses et aux opérations d'évolution des hyperclasses. Le quatrième chapitre introduit les formalismes de graphes de Gavroche, les règles d'intégrité et le concept de composant de SI. Dans le cinquième chapitre, nous comparons nos travaux à des travaux scientifiques proches. Dans le sixième chapitre, nous présentons les réalisations techniques qui supportent nos propos. Finalement, nous faisons le bilan de nos travaux et proposons les perspectives que nous envisageons pour les poursuivre.

Dans l'immédiat, nous introduisons l'exemple M@TIS que nous utilisons pour illustrer nos propos.

4. EXEMPLE

Nous avons choisi d'illustrer nos propos dans ce travail à travers un exemple concret qui nous est particulièrement cher : *M@TIS*.

M@TIS[44] est le système d'information de la formation européenne de troisième cycle en management et technologies des systèmes d'information *MATIS*. Il a été mis en place pour coordonner les activités de la formation. Le système a été progressivement développé et mis en production depuis l'an 2000, en partie dans le cadre d'une collaboration entre les équipes *LSR-IMAG* de *Grenoble* et *MatisGe* de *Genève*, et à travers la préparation de plusieurs travaux de mémoires de licence en systèmes d'information à l'Université de Genève ((Schmidt & al.,

[44] *http://DiplomeMatis.unige.ch*

2001)[45], (Moix, 2002)[46], (Leung & Olivari, 2002)[47]). Le projet *M@TIS* a par ailleurs donné lieu à deux publications dans l' « *International Conference on Engineering Education* » (*ICEE*) : (Giraudin & Freireire, 2001)[48] et (Giraudin & al, 2002)[49].

La formation *MATIS* (*« Management and Technology of Information Systems* ») est un diplôme d'études approfondies (DEA) en systèmes d'information, organisée dans le cadre de l'Association Transfrontalière Universitaire et regroupant les établissements universitaires et grandes écoles françaises et suisses de la Conférence Universitaire Rhône-Alpes et de la Conférence Universitaire de Suisse Occidentale : *Université de Genève*, *Université Joseph Fourier de Grenoble*, *Université Pierre-Mendés France de Grenoble*, *Université de Savoie*, *École Polytechnique Fédérale de Lausanne*, *Université de Lausanne* et l'*Institut National Polytechnique de Grenoble*. Elle intègre les enseignements de deux domaines des systèmes d'information qui sont complémentaires et séparés en deux spécialisations : gestion des systèmes d'information (GSI) et technologie des systèmes d'information (TSI).

Lors de leur inscription, les étudiants choisissent leur spécialité, pour laquelle ils suivront des cours spécifiques en plus des cours proposés en tronc commun avec le reste des étudiants. L'intérêt des cours de tronc commun est la confrontation et l'échange des points de vue entre les étudiants des deux spécialités.

Les cours sont dispensés sous forme de modules ou d'ateliers, pour lesquels la participation active des étudiants est demandée. Pour chaque matière, l'étudiant présente un travail de recherche, effectué seul ou à plusieurs, et passe un examen. Les cours sont organisés en trois trimestres de huit semaines dont le dernier est consacré au travail de mémoire.

Au terme du programme, l'étudiant doit présenter un mémoire de diplôme, effectué en général dans un des laboratoires participant à la formation, et doit le soutenir devant un jury composé du collège des enseignants.

Dans le cadre de cet accord, les étudiants ont accès aux différentes institutions partenaires, à leurs moyens informatiques et à leur centre documentaire. Ils peuvent entreprendre des projets et des thèses avec des professeurs qui ne sont pas de leur site académique d'origine.

M@TIS associe des informations tant administratives que pédagogiques et de recherche afin de disposer d'une organisation efficiente qui renforce les enseignements en présentiel et qui

---

[45] (Schmidt & al., 2001) Schmidt Ch., Nastasi S., Iseli B., Mise en place d'un système d'information: l'intranet M@TIS. Mémoire de licence en systèmes d'information à l'Université de Genève. Octobre 2001.

[46] (Moix, 2002) Moix E.,Conception et mise en place d'un système d'information pour la formation MATIS. Mémoire de licence en systèmes d'information à l'Université de Genève. Avril 2002.

[47] (Leung & Olivari, 2002) Leung K-F, Olivari A., Conception et développement d'une solution web pour la formation MATIS. Novembre 2002.

[48] (Giraudin & Freireire, 2001) Giraudin J. P., Freireire J.-J.-C., An educational intranet for a formation in management and technology of information systems. Proceedings of ICEE 2001, Oslo, Norway, August 2001.

[49] (Giraudin & al., 2002) Giraudin J. P., Freireire J.-J.-C., Turki S., Assessment of a difficult voyage in Pentagonie - Evaluation of a pedagocical intranet project. Proceedings of ICEE 2002, Manchester, UK, August 2002.

facilite l'initiation à la recherche des étudiants qui suivent cette formation. Par rapport aux aspects organisationnels, *M@TIS* coordonne un ensemble d'informations tant administratives que de gestion des enseignements. Par rapport aux contenus pédagogiques, *M@TIS* comporte des informations sur les modules pour préparer et accompagner l'enseignement présentiel et le poursuivre. Par rapport aux aspects d'initiation à la recherche, *M@TIS* doit faciliter, pour chaque étudiant, d'une part son appropriation d'un domaine d'études et d'autre part sa bonne insertion dans sa problématique de recherche.

Le diagramme de classes de M@TIS est présenté dans L'Annexe A.

CHAPITRE I

BINEX

INTRODUCTION

Un modèle de SI est pertinent s'il permet (i) de décrire des situations de manière non ambiguë, (ii) d'établir la fidélité entre un modèle de SI et sa réalisation informatique et (iii) d'établir la fidélité entre un modèle de SI et l'organisation de l'entreprise ou de l'institution. La construction de modèles pertinents nécessite un cadre conceptuel qui garantit à la fois leur rigueur, leur non-ambiguïté, leur cohérence informationnelle et leur évolution indépendamment des méthodes, des langages et des technologies dans lesquelles le SI pourrait être implanté.

Nous adoptons le modèle *binex* comme méta-modèle pour la modélisation informationnelle des aspects statiques des systèmes d'information. *Binex* est destiné à produire des modèles informationnels non-ambigus, évolutifs et pouvant directement être implantés.

Le modèle *binex* est construit autour de diagrammes de classes représentent l'organisation et la structuration de l'information dans le SI. Ces propriétés, qualifiées de *statiques*, concernent les informations de nature persistante. Un diagramme de classes constitue une expression semi-formelle (ou formelle) des propriétés statiques du SI. Il décrit les classes du SI, leurs attributs, leurs identifiants, les méthodes des classes, les liens qui relient ces classes.

Binex est un modèle orienté objet dit *binaire* et *existentiel*. En plus de respecter les principes du paradigme orienté objet, il introduit des concepts ou des restrictions accentuant son pouvoir d'expression sémantique. Par rapport aux autres langages de modélisation de SI, *binex* ne considère que les liens existentiels et les liens de spécialisation-généralisation sans pour autant réduire le potentiel d'expression informationnelle mais en augmentant la rigueur d'expression.

Dans ce chapitre, nous présentons la base du méta-modèle *binex* et nous définissons également un ensemble complet d'opérations pour sa manipulation et son évolution.

1. LE MODÈLE *BINEX*

Le modèle *binex* est construit autour de la notion de diagramme de classes. Un diagramme de classes regroupe et structure un ensemble de classes, d'objets, d'attributs, d'identifiants, de méthodes et de liens entre classes et objets. *Binex* a la particularité de n'utiliser que les liens existentiels et les spécialisations-généralisations entre classes.

1.1. CLASSE

Une classe est un « modèle abstrait définissant des variables et des méthodes pour un type donné d'objet, et à partir duquel sont créés des objets concrets possédant des valeurs particulières »[50].

Les objets d'une classe ont en commun une structure informationnelle comprenant :

- des *attributs* associés à des domaines de valeurs, pour lesquels ces objets prennent une valeur de leur domaine ;
- des *identifiants* et ;
- des *méthodes* qui, appliquées aux objets de la classe, transforment les valeurs qu'ils prennent pour les attributs.

Représentation graphique

Une classe est représentée sous forme d'un rectangle à trois compartiments (Figure 4) : (i) le compartiment supérieur indique le nom de la classe ; (ii) le second compartiment est réservé aux attributs de la classe et à ses identifiants et (iii) le troisième compartiment contient les méthodes définies sur la classe.

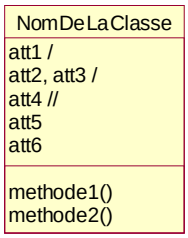

Figure 4 : Représentation graphique d'une classe

Le symbole « / » sépare les différents identifiants de la classe. Chaque identifiant est composé d'un ou de plusieurs attributs. Le symbole « // » clôt l'ensemble des identifiants de la classe.

Pour simplifier les schémas, les compartiments des attributs et des méthodes sont parfois masqués.

1.1.1. Attribut

Un attribut est une « information structurelle relative à une donnée, décrivant son contexte et son sens »[51]. C'est une propriété de la classe portant un nom et ayant un domaine.

---

[50] Office de la Langue Française, 2000, *GrandDictionnaire.com*

[51] Office de la Langue Française, 2000, *GrandDictionnaire.com*

1.1.1.1. Domaine et type de base

Un attribut prend ses valeurs dans un ensemble de valeurs possibles appelé *domaine*. Un domaine est un type conceptuel qui définit l'ensemble des valeurs que peut prendre l'attribut. Il est défini à partir d'un *type de base*. Un type de base est du texte, un mot, une unité numérique, un booléen, ou une date, une durée, etc.

Un *domaine* est un ensemble fini ou infini de valeurs, comme par exemple le domaine des *booléens* : {*Vrai*, *Faux*}, le domaine des *diplômes universitaires* : {*Certificat*, *Baccalauréat universitaire*, *Licence*, *Maîtrise universitaire*, *Doctorat*}, etc.), ou l'intervalle des entiers [3.. 12] ou l'ensemble des chaînes de caractères de longueur 10.

Les *bornes inférieures* et *supérieures* du domaine définissent l'ensemble des valeurs possibles en fonction du type de base sur lequel est défini le domaine.

Plusieurs attributs peuvent avoir le même domaine. Un objet prend au plus une valeur pour cet attribut. Elle peut être *obligatoire* (l'objet prend obligatoirement une valeur pour cet attribut) ou *facultative* (l'objet peut prendre une *valeur obscure* (définie dans la suite) pour cet attribut).

1.1.1.2. Valeur par défaut

La *valeur par défaut* pour un attribut d'une classe indique la valeur que prend tout objet de la classe pour lui si aucune valeur n'est indiquée lors de la création de l'objet.

1.1.1.3. Valeur obscure - claire

Une *valeur obscure* est une valeur inconnue ou une valeur impossible. Une valeur qui n'est pas obscure est une *valeur claire*.

Pour chaque attribut *att* d'une classe *cl*, il est nécessaire d'indiquer si un objet de *cl* peut prendre une *valeur obscure* pour *att*.

1.1.1.4. Attribut permanent

Un attribut *att* d'une classe est *permanent*, si la valeur claire prise pour *att* par tout objet de la classe ne peut être ensuite modifiée.

Dans *binex*, et de la même manière que dans une relation universelle (Maier & Ullman, 1984)[52], le nom d'un attribut a une sémantique unique (un *goût unique*, « *unique taste »*) dans un même diagramme de classes d'un SI. Un attribut peut apparaître dans plusieurs classes (même nom d'attribut).

[52] (Maier & al., 1984) Maier D., Ullman J.D., Vardi, M. Y., On the foundations of the Universal Relation Model. ACM Transactions on Database Systems, Vol. 9, No. 2, June 1984, Pages 283-308.

1.1.1.5. Attribut identifiant, référentiel, simple

Un attribut peut être un *attribut identifiant* quand il appartient à l'un des identifiants d'une classe (voir le paragraphe [1.1.3. Identifiant d'une classe, Page 24]), ou un *attribut référentiel* lorsqu'il est utilisé pour exprimer un lien entre deux classes. Si un attribut apparaît dans plus d'une classe dans un modèle, il est nécessairement un attribut identifiant dans au moins une classe de la spécification et un attribut référentiel dans une ou plusieurs autres classes.

Un attribut qui n'est ni un attribut identifiant ni un attribut référentiel est dit *attribut simple*.

1.1.1.6. Notation

Nous notons *cl.att* l'occurrence de l'attribut *att* de la classe *cl*, et cl^+ l'ensemble des attributs de *cl*.

1.1.2. Objet de classe

Un objet d'une classe est une *occurrence* ou *instance* de cette classe. Tous les objets d'une même classe ont une même structure informationnelle et une même structure de comportement.

La structure d'un objet est le *n-uplet* qui prend une valeur pour chaque attribut de la classe.

Notation

Nous notons *o[att]* la valeur prise par l'objet *o* de la classe *cl* pour l'occurrence d'attribut *cl.att*.

1.1.3. Identifiant d'une classe

Un identifiant permet de distinguer entre eux les objets d'une même classe *cl* suivant les valeurs qu'ils prennent pour les attributs formant cet identifiant. C'est un ensemble minimal d'attributs de *cl* tels que si deux objets de *cl* prennent les mêmes valeurs pour ces attributs, ces deux objets sont identiques.

Une classe peut admettre plusieurs identifiants, mais doit en avoir au moins un, éventuellement formé de tous les attributs de la classe.

Un identifiant *id* de *cl* est dit *obligatoire* si pour tout objet *o* de *cl* et pour tout attribut *att* d'*id*, *cl.att* est toujours clair. *id* est dit *permanent* si la valeur prise pour tout attribut *att* d'*id* ne peut être modifiée pour tout objet *o* de *cl* après son instanciation.

Dans *binex*, tous les identifiants sont (i) *obligatoires* et (ii) *permanents.*

Représentation graphique

Un identifiant est représenté par la suite d'attributs qui le forment séparés par des virgules. Les identifiants sont séparés entre eux par le symbole « / ». Ce symbole est remplacé par « // » à la fin du dernier identifiant de la classe.

Dans l'exemple de la Figure 5, la classe *Personne* admet deux identifiants :

1. le premier est construit sur l'attribut *numAVS* (numéro d'Assurance vieillesse et survivants en Suisse, équivalent au numéro de sécurité sociale de la personne en France). Le numéro AVS est attribué à vie à une personne en Suisse à son premier emploi rémunéré. L'attribut *numAVS* est obligatoire et permanent et identifie de façon unique toute personne dans le système ;
2. le deuxième identifiant est construit sur les attributs *nom* et *prénom* de la personne. Bien que les nom et prénom d'une personne permettent de l'identifier dans le système, les valeurs prisent par ces attributs peuvent changer dans le temps. Si une personne change de nom, il faut supprimer l'objet correspondant et recréer un nouvel objet avec le même numéro AVS, le même prénom, la même date de naissance et le nouveau nom.

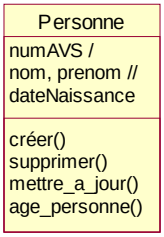

Figure 5 : Exemple de classe

La Figure 6 montre le méta-modèle *binex* relatif aux classes, aux attributs, aux identifiants et aux objets, représenté dans le formalisme *binex*. Les flèches entre les classes représentent des liens existentiels entre elles.

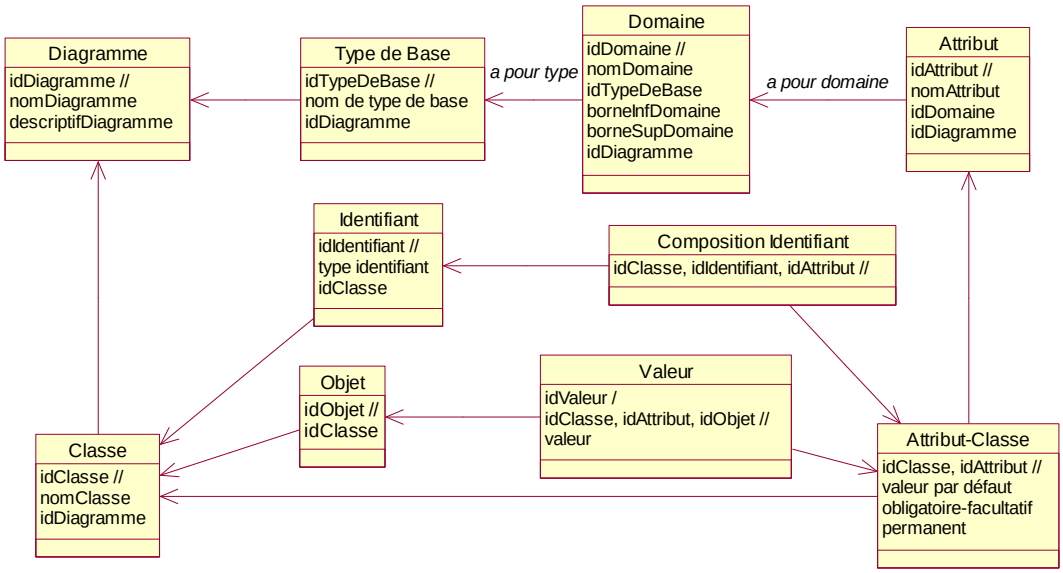

Figure 6 : Méta-modèle *binex* relatif aux classes, aux attributs, aux identifiants et aux objets

1.1.4. Méthode de classe

Une *méthode de classe* est une opération associée à la classe, qui permet d'accéder et de modifier les données des objets de la classe.

Invoquer une méthode sur un objet (i) modifie les valeurs prises par certains des ses attributs ou (ii) retourne des valeurs calculées à partir de valeurs prises par les attributs pour l'objet.

La spécification d'une méthode, indépendamment du code qui l'implante, est fournie par sa *signature* qui définit le type des arguments d'entrée et de sortie. La signature d'une méthode est constituée du nom de l'opération et, éventuellement, des paramètres d'appel.

La définition ou *déclaration* d'une méthode se fait selon la syntaxe suivante :

```
type_de_donnée Nom_De_La_Methode(type1 argument1, type2 argument2, …) {
 liste d'instructions
}
```

Une instruction est un ordre exprimé en langage de programmation, dont l'interprétation entraîne l'exécution d'une opération élémentaire (ou opérateur) de type déterminé. Une suite d'instructions constitue un programme. Les instructions portent sur les valeurs prises par les objets de la classe pour ses attributs.

Les opérateurs sont des opérations élémentaires qui permettent d'évaluer et de manipuler des opérandes[53]. On distingue plusieurs types d'opérateurs: les opérateurs de calcul, les

[53] Opérande : n. m. [mathématique] Élément sur lequel on effectue une opération. Cela peut être une constante (symbolique ou pas), ou une variable.

opérateurs d'assignation, les opérateurs d'incrémentation, les opérateurs de comparaison et les opérateurs logiques.

En plus des variables locales à la méthode, les opérandes dans les instructions peuvent être des objets de la classe, des valeurs prises par des objets pour certains attributs ou des méthodes de la même classe.

1.1.4.1. Appel d'une méthode de classe

Pour être exécutée, une méthode est appelée par son nom suivi d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée:

```
Nom_De_La_méthode();
```

ou

```
Nom_De_La_méthode(argument1, argument2, …);
```

Dans *binex*, une méthode ne peut être invoquée sur un objet de sa classe que si elle est active. Elle peut être *active* ou *inactive*. L'état de la méthode est indiqué par l'attribut *étatMéthode* dans la classe *Méthode* du méta-modèle (Figure 7).

1.1.4.2. Représentation graphique

Une méthode est représentée par son nom dans le compartiment inférieur de la classe.

Dans l'exemple de la Figure 5, la classe *Personne* a quatre méthodes : les trois méthodes de base (*créer()*, *supprimer()* et *consulter()*) et la méthode *âge_personne()*.

La Figure 7 représente le méta-modèle *binex* relatif aux méthodes de classe. La classe *Attribut-Méthode-Classe* indique les occurrences d'attributs utilisées dans les instructions de la méthode de classe.

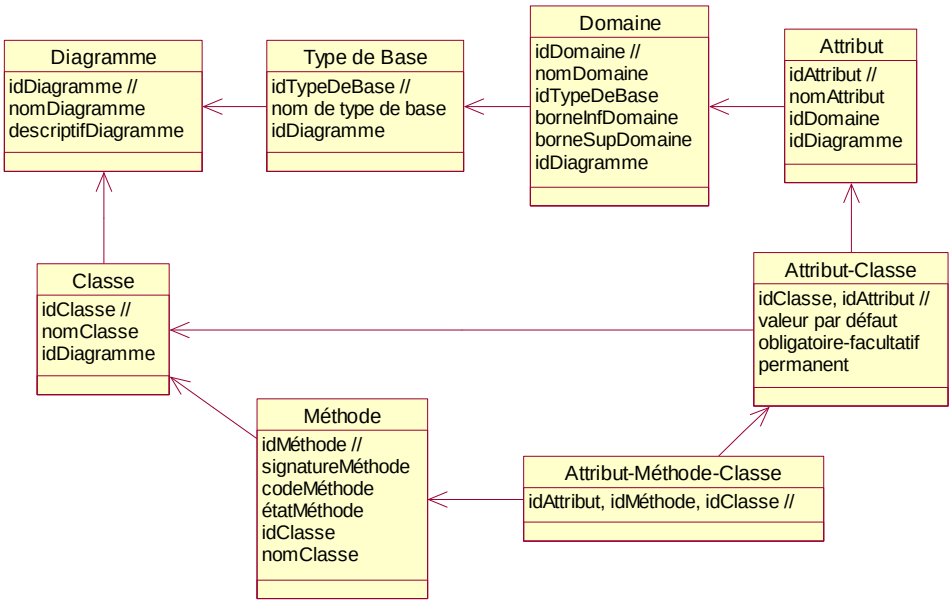

Figure 7 : Méta-modèle *binex* relatif aux méthodes de classe

1.2. LIENS ENTRE CLASSES

Nous parlons de *lien* ou *dépendance* entre classes. Un lien est un « rapport logique ou de dépendance »[54]. Une dépendance est un « rapport entre entités ou entre attributs qui indique que l'existence d'une entité ou d'un attribut n'a d'intérêt que si une autre entité ou un autre attribut existe » [55]. Les liens entre classes décrivent conceptuellement, au niveau des classes, les dépendances qui relient les objets de ces classes.

Dans *binex*, nous utilisons exclusivement deux types de liens pour relier les classes entre elles : les *liens existentiels* et les *liens de spécialisation-généralisation*. Les liens de spécialisation-généralisation sont un cas particulier des dépendances existentielles.

1.2.1. Lien existentiel

Un lien existentiel exprime une connexion sémantique bidirectionnelle particulière entre deux classes : si une classe cl_A est *liée existentiellement* à une classe cl_B, l'existence de tout objet o_A de cl_A est conditionnée par l'existence de l'objet o_B de cl_B auquel o_A est relié : la suppression d'o_B entraîne la suppression d'o_A. Le concept représenté par la classe cl_A a besoin de la classe cl_B pour pouvoir être défini. Les classes cl_A et cl_B sont respectivement les *extrémités initiale* et *terminale* du lien existentiel ; cl_A est la *classe dépendante* et cl_B est la *classe de référence*.

Les classes cl_A et cl_B ont en commun au moins les attributs d'un identifiant de cl_B. Chacun de ces attributs est considéré comme attribut référentiel dans cl_A. Les objets o_A et o_B reliés entre eux prennent les mêmes valeurs pour leurs attributs communs à cl_A et cl_B. o_A est dit *dépendant (existentiellement)* à o_B. o_A et o_B sont dits (*directement*) *liés* (ou *reliés*).

Si les classes cl_A et cl_B ont en commun d'autres attributs, les objets de cl_A et cl_B liés entre eux prennent les mêmes valeurs pour ces attributs communs : les attributs dans *binex* ont une sémantique unique dans un même diagramme de classes et il est incohérent d'avoir des objets liés prenant des valeurs différentes pour le même attribut. Les valeurs communes prises pour les attributs communs aux classes cl_A et cl_B (identifiants et référentiels) par leurs objets reliés matérialisent au niveau des objets les liens qui existent entrent leurs classes.

En raison notamment des restrictions sur la mise-à-jour des valeurs des identifiants dans *binex*, si une classe cl_A est liée existentiellement à une classe cl_B, tout objet o_A de cl_A dépendant du même et unique objet o_B de cl_B auquel il reste relié.

Dans *binex* (Figure 9), les liens entre classes sont *existentiels* et *binaires* (un lien existentiel relie exactement deux classes distinctes) et peuvent être parcourus dans les deux sens au niveau des objets.

---

[54] Le Petit Larousse illustré 1999.

[55] Office de la Langue Française, 2000, *GrandDictionnaire.com*

1.2.1.1. Attribut référentiel

Soit la classe cl_A liée existentiellement à la classe cl_B.

Tout attribut commun à à cl_A et à cl_B et appartenant à un identifiant de cl_B est un attribut référentiel.

1.2.1.2. Représentation graphique

Dans la représentation graphique, il existe un arc d'extrémité initiale le nœud de cl_A et d'extrémité terminale le nœud de cl_B alors cl_A est liée existentiellement à cl_B.

Figure 8 : Représentation d'un lien existentiel entre deux classes

1.2.1.3. Cardinalité des liens existentiels

Une caractéristique d'un lien entre classes est sa cardinalité, qui permet de contraindre le nombre d'objets minimum et maximum connectés à chacune de ses extrémités.

Si la classe cl_A est liée existentiellement à la classe cl_B, tout o_A de cl_A ne peut exister que s'il est relié à un objet o_B de cl_B durant tout son cycle de vie. o_B peut être relié à aucun ou à plusieurs objets de cl_A. La cardinalité du côté de cl_B est 0..n. Du côté de cl_A, les cardinalités minimale et maximale sont obligatoirement égales à 1.

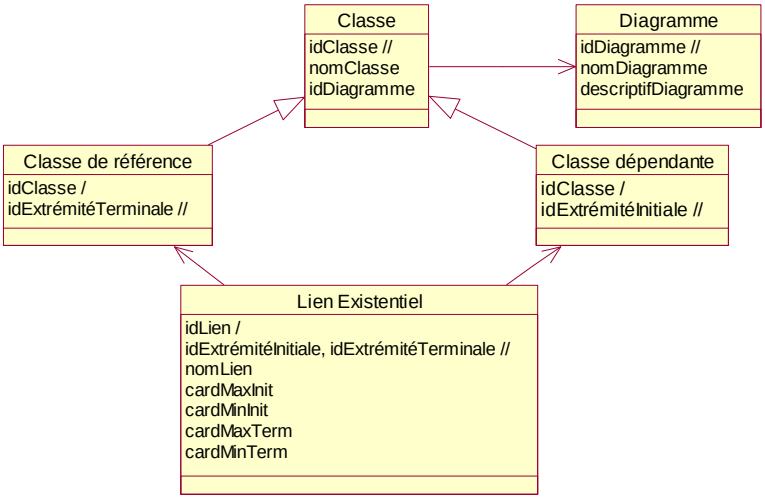

Figure 9 : Lien existentiel entre classes dans *binex*

Exemple de lien existentiel

Au terme de la formation *MATIS*, chaque étudiant présente un projet de recherche, effectué en général dans l'un des laboratoires participant à la formation, et le soutient devant un jury composé du collège des enseignants.

Dans *M@TIS* (Figure 10), la classe *Projet* est liée existentiellement à la classe *Étudiant*. *Étudiant* a un identifiant construit sur l'attribut *idEtudiant*. *idEtudiant* est aussi attribut référentiel dans *Projet*.

Chaque objet de *Projet* est toujours relié à un et un seul objet d'*Étudiant*, et lui reste relié. La suppression d'un objet *Étudiant* supprime l'objet de *Projet* qui lui est relié.

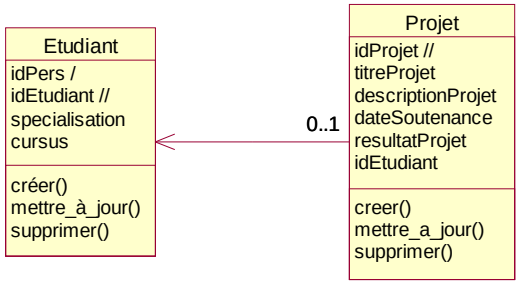

Figure 10 : Exemple de lien existentiel

La contrainte de cardinalité *0..1* définie sur *Projet* dans le lien (*Projet*, *Étudiant*) fait qu'un objet d'*Étudiant* peut avoir au plus un objet de *Projet* qui lui soit relié.

1.2.2. Spécialisation-généralisation

La *spécialisation-généralisation* est un concept abstrait qui exprime la relation *est-un* dans une hiérarchie de classes. C'est un principe qui est inhérent à la nature et qui a permis, bien avant l'apparition de l'informatique, d'établir un ordre entre les concepts de nombreux domaines scientifiques. Ce concept, issu des modèles de données sémantiques, a été introduit dans le monde des bases de données par (Smith & Smith, 1977)[56]. Dans le monde objet, la spécialisation-généralisation est utilisée pour classifier les classes et leurs objets en fonction de leurs points communs et de leur spécificité dans un même graphe. Ces classes sont ordonnées dans une hiérarchie de classes : « la spécialisation-généralisation de classes introduit une relation transitive et antisymétrique (relation d'ordre) qui crée une organisation hiérarchique de classes ; elle est de type treillis dans le cas de la spécialisation-généralisation multiple » (Charbou-Pecou & al., 2002)[57].

---

[56] (Smith & Smith, 1977) Smith J.M., Smith D.C.P., Database Abstractions: Aggregation and Generalization. ACM TODS, Vol. 2, No. 2, 1977.

[57] (Charbou-Pecou & al., 2002) Charbou-Pecou, M., Freire J., J. C., Front, A., Giraudin, J-P., Guetari, R., Ingénierie Objet. Chapitre 1 : Objets, langages et méthodes. Editions Dunod, 2002. ISBN : 2729606424.

La spécialisation-généralisation permet de partager les points communs entre les classes tout en préservant leurs différences. Elle met en relation une classe avec une ou plusieurs versions affinées de la classe. La *généralisation* est une fonction qui, à une classe d'origine, dite *sous-classe*, fait correspondre une classe plus générale, dite *super-classe*. La fonction inverse de la généralisation, qui fait correspondre à une classe toutes les sous-classes dont elle est super-classe, est appelée *spécialisation*. Elle permet d'ajouter des propriétés spécifiques à une classe (extension) pour obtenir une sous-classe ou restreindre des propriétés ancêtres (restriction). Le lien qui s'établit entre la super-classe et ses sous-classes est un lien connu comme un lien « *is-a »* ou « *est-un »* ou « *est- une-sorte-de »*.

Un objet d'une sous-classe est la spécialisation d'un objet d'une super-classe ; il est appelé *sous-objet*, du deuxième, son *super-objet*. L'objet d'une classe est une instance de tous les ancêtres de la classe. Par conséquent, toutes les caractéristiques et opérations des classes ancêtres doivent s'appliquer aux objets de la sous-classe.

« Le concept de spécialisation-généralisation est à considérer du double point de vue ensembliste et descriptif. Du point de vue ensembliste, il doit y avoir inclusion ensembliste des objets d'une sous-classe dans l'ensemble des objets de ses super-classes. En ce qui concerne l'aspect descriptif, l'ensemble des descriptions d'une super-classe doit être applicable aux objets des sous-classes de celle-ci ; il y a donc inclusion du type d'une super-classe dans les types de ses sous-classes. » (Charbou-Pecou & al., 2002)[58].

Dans le monde objet, les termes « *spécialisation-généralisation »* et « *héritage »* sont fréquemment utilisés comme synonymes. L'*héritage* est un mécanisme informatique qui est une des manières les plus courantes d'implanter le concept de spécialisation-généralisation. La spécialisation-généralisation facilite la modélisation en organisant les classes, l'héritage des opérations aide, lors de l'implantation, pour la réutilisation de codes.

La construction d'un SGBD (Système de Gestion de Bases de Données) basé sur le modèle entité-association étendu (*ECRINS*, (Junet & al., 1986)[59]) et d'un autre SGBD, orienté-objet, (*F2*, (Al-Jadir & al., 1995)[60]) ainsi que d'autres approches proposées par différents auteurs ont démontré que le concept de spécialisation-généralisation peut être implanté par un mécanisme autre que l'héritage (simple ou multiple) : la *spécialisation dynamique* est une implantation particulière du concept de spécialisation-généralisation au niveau d'un SGBD.

---

[58] (Charbou-Pecou & al., 2002) Charbou-Pecou, M., Freire J., J. C., Front, A., Giraudin, J-P., Guetari, R., Ingénierie Objet. Chapitre 1 : Objets, langages et méthodes. Editions Dunod, 2002. ISBN : 2729606424.

[59] (Junet & al., 1986) Junet M., Falquet G., Léonard M., ECRINS/86: An Extended Entity-Relationship Data Base Management System and its Semantic Query Language, Proc. Int. Conf. on Very Large Data Bases, VLDB, Kyoto 1986.

[60] (Al-Jadir & al., 1995) Al-Jadir L., Estier T., Falquet G., Léonard M., Evolution Features of the F2 OODBMS, Proc. Int. Conf. on Database Systems for Advanced Applications, DASFAA, Singapore 1995.

1.2.2.1. Héritage

L'idée de l'héritage de classes est de fournir un mécanisme qui permet la définition de nouvelles classes qui héritent de propriétés d'autres classes existantes. Pour chaque nouvel objet, tous les attributs sont instanciés localement tandis que les méthodes sont partagées à travers le modèle. Les nouvelles classes héritent des modèles de leurs super-classes. Dans l'héritage, toutes les instances d'une classe sont conformes avec la description donnée par le modèle de la classe. Une fois créés, les objets gardent leurs propriétés et restent dans la classe pendant tout leur cycle de vie.

1.2.2.2. Spécialisation dynamique

La spécialisation dynamique (Bachman & Daya, 1977)[61], (Junet & al., 1986)[62], (Al-Jadir & al., 1995)[63], (Gottlob & al., 1996)[64], (Papazoglou & Krämer, 1997)[65], (Al-Jadir & Léonard, 1999)[66], (Dahchour & al., 2002)[67] permet d'exprimer la nature dynamique de l'objet : un objet peut appartenir à plusieurs classes à la fois dans la même hiérarchie de spécialisation : nous disons alors qu'il est *multi-instancié*. Il peut devenir instance d'une autre classe (obtenir une autre classification) ou ne plus être instance d'une classe (abandonner une classification).

Contrairement à une spécialisation statique où un objet prend une position dans un arbre de spécialisation et la conserve pendant tout son cycle de vie, avec la spécialisation dynamique l'objet peut changer suivant son cycle de vie, tout en restant toujours un objet de la classe racine. Ainsi, dans l'exemple de la Figure 11, une *Personne* peut devenir *Étudiant* puis *Enseignant* et ensuite ne plus être *Étudiant*. C'est dans ce sens que nous qualifions cette spécialisation est qualifiée de dynamique.

---

[61] (Bachman & Daya, 1977) Bachman Ch. W., Daya M., The role concept in data models, VLDB 3rd, Tokyo, 1977.

[62] (Junet & al., 1986) Junet M., Falquet G., Léonard M., ECRINS/86: An Extended Entity-Relationship Data Base Management System and its Semantic Query Language, Proc. Int. Conf. on Very Large Data Bases, VLDB, Kyoto 1986.

[63] (Al-Jadir & al., 1995) Al-Jadir L., Estier T., Falquet G., Léonard M., Evolution Features of the F2 OODBMS, Proc. Int. Conf. on Database Systems for Advanced Applications, DASFAA, Singapore 1995.

[64] (Gottlob & al., 1996) Gottlob G., Schrefl M., Röck B., Extending object-oriented systems with roles, ACM Transactions on Information Systems, Vol.14, No.3, July 1996.

[65] (Papazoglou & Krämer, 1997) Papazoglou M.P., Krämer B.J., A database model for object dynamics, VLDB Journal volume 6, 1997.

[66] (Al-Jadir & Léonard, 1999) Al-Jadir L., Léonard M., If we refuse the Inheritance…, Proc. Int. Conf. on DEXA, Firenze, 1999.

[67] (Dahchour & al., 2002) Dahchour M., Pirotte A., Zimanyi E., A generic role model for dynamic object, Proc. Int. Conf. CAISE 2002, Toronto 2002.

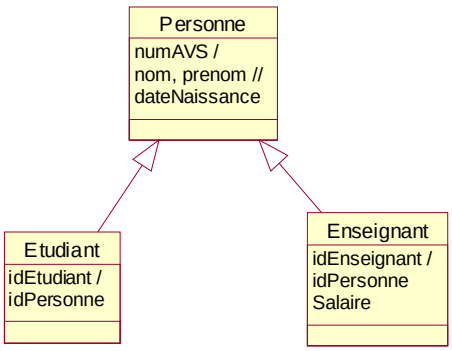

Figure 11 : Exemple de spécialisation dynamique

Par ailleurs, la spécialisation dynamique permet à un objet d'une super-classe d'être spécialisé par plusieurs sous-objets dans une même sous-classe : un fait du monde réel peut être représenté par plusieurs objets ; l'objet avec ses spécialisations est un *multi-objet*. Quand un objet devient instance d'une sous-classe, un sous-objet est créé dans la sous-classe et celui-ci dépend existentiellement de son super-objet, c'est-à-dire que si on supprime le super-objet, le sous-objet est aussi supprimé. L'objet décomposé sur les différentes classes de son arbre de spécialisation est reconstruit par navigation entre ses différents super-objet (objets parents) jusqu'à son objet racine.

L'instanciation multiple constitue un moyen pour éviter l'explosion combinatoire de classes peu peuplées, nécessaires dans le cadre de l'héritage multiple. Dans l'exemple de la Figure 11, grâce à l'instanciation multiple, un objet peut appartenir simultanément aux classes *Personne*, *Étudiant* et *Enseignant* pour prendre en compte les personnes qui sont à la fois étudiantes et employées. Si cette catégorie de personnes (les assistants à l'Université de Genève, les ATER en France par exemple) n'a pas de propriétés particulières, l'instanciation multiple évite de créer une classe *Étudiant-Enseignant* comme sous-classe d'*Étudiant* et d'*Enseignant*.

1.2.2.3. La spécialisation-généralisation dans binex

Dans *binex*, quand nous traitons le concept de spécialisation-généralisation, nous admettons la spécialisation dynamique comme mode d'implantation sous-jacent.

Dans *binex* (Figure 12), un lien de spécialisation-généralisation est un cas particulier des liens existentiels dans *binex* : un sous-objet ne peut exister que s'il est lié de manière existentielle à un objet d'une super-classe de sa classe. Il reste relié au même super-objet durant tout son cycle de vie. Si un super-objet est supprimé, ses sous-objets le sont aussi.

En cas de généralisation multiple (cas où une classe admet plusieurs super-classes), chaque sous-objet est lié existentiellement à un objet de chacune de ses super-classes.

Comme la sous-classe est liée existentiellement à chacune de ses super-classes, elle admet comme attributs (référentiels) les attributs d'au moins un identifiant de chacune de ses super-classes.

L'objet spécialisé est décomposé sur l'arbre de spécialisation et l'information est reconstituée par navigation dans le multi-objet. Nous parlons de propriétés (attributs et méthodes) *atteignables* de la super-classe au niveau de ses sous-classes.

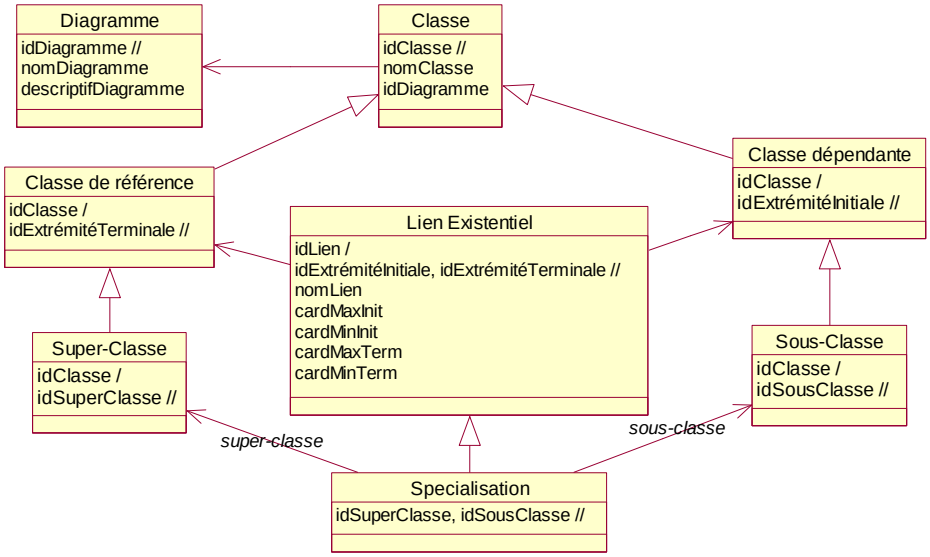

Figure 12 : Méta-modèle *binex* relatif à la spécialisation-généralisation de classes

1.2.2.4. Représentation graphique

Dans la représentation graphique, il existe un arc à tête creuse d'extrémité initiale le nœud de cl_A et d'extrémité terminale le nœud de cl_D alors cl_A est une spécialisation de cl_D et cl_D est une spécialisation de cl_A.

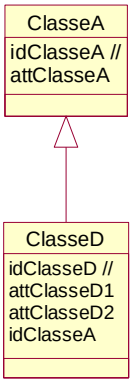

Figure 13 : Représentation graphique d'un lien de spécialisation-généralisation dans *binex*

1.2.3. Zone de dépendance d'une classe

Nous définissons la *zone de dépendance* d'une classe *cl* comme étant l'ensemble formé de :

1. la classe *cl* ;
2. l'ensemble des classes liées existentiellement à *cl* et en particulier celles qui en sont une spécialisation, et
3. l'ensemble des classes liées existentiellement et donc en particulier celles qui en sont une spécialisation d'une classe de la zone de dépendance de *cl*.

1.2.4. Remarque

Les liens existentiels et les liens de spécialisation-généralisation sont un cas particulier des associations (*relationship*) du modèle *entité-relation* (Chen, 1976)[68]. Dans certaines extensions de ce modèle ((Morejon, 1994)[69], (Bagui & Earp, 2003)[70]), l'appellation « entités fortes / faibles » (*strong / weak entities*) ou « *entités dominantes* / *dominées* » est parfois utilisée pour désigner respectivement les classes de référence et les classes dépendantes.

1.3. DIAGRAMME DE CLASSES

Un diagramme de classes est une construction qui regroupe les classes, attributs, identifiants, méthodes et liens entre classes et objets représentant la structuration de l'information dans tout ou partie d'un SI.

1.3.1. Diagramme de classes

Un diagramme de classes *D* est un graphe orienté défini par un triplet (C, Λ, Γ) où :

- *C* désigne l'ensemble des classes de *D* ;
- $\Lambda \subseteq CxC$ tel que $(cl_1,cl_2) \in \Lambda$ si et seulement si cl_1 dépend existentiellement de cl_2 ;
- $\Gamma \subseteq CxC$ tel que $(cl_1,cl_2) \in \Gamma$ si et seulement si cl_1 est une spécialisation de cl_2 (ou si cl_2 est une généralisation de cl_1).

$$D = < C, \Lambda, \Gamma >$$

Les liens de spécialisation-généralisation sont un cas particulier des liens existentiels donc si $(cl_1,cl_2) \in \Gamma$ alors $(cl_1,cl_2) \in \Lambda$.

---

[68] (Chen, 1976) Chen. P., The entity-relationship model: Toward a unified view of data. ACM Transactions on Database Systems, 1(1):9-36, 1976.

[69] (Morejon, 1994) Morejon J., Merise, vers une modélisation orientée objet. Les Editions d'Organisation 1994.

[70] (Bagui & Earp, 2003) Bagui S., Earp R., Database Design Using Entity-Relationship Diagrams. Auerbach Pub, June 2003. ISBN: 0849315484.

$$(cl_1,cl_2)\in\Gamma \Rightarrow (cl_1,cl_2)\in\Lambda$$

Règle

Si $(cl_1,cl_2)\in\Lambda$ alors $(cl_2,cl_1)\notin\Lambda$.

Définitions

1. Classes adjacentes

Si $(cl_1,cl_2)\in\Lambda$ ou si $(cl_2,cl_1)\in\Lambda$, les classes cl_1 et cl_2 sont dites *adjacentes* dans D.

2. Connexité d'un diagramme de classes

Un diagramme de classes D est *connexe* si et seulement si pour toute paire (cl_1,cl_2) de classes de D, il existe une séquence $\{cl_1, cl_{11}\ldots cl_{1i}\ldots cl_n, cl_2\}$ de classes adjacentes.

3. Sous-diagramme de classes

Dans un diagramme de classes $D(C, \Lambda, \Gamma)$, un *sous-diagramme de classes* $D_{sd}(C_{sd},\Lambda_{sd},\Gamma_{sd})$ est un diagramme de classes tel que :

- $C_{sd} \subseteq C$; C_{sd} désigne l'ensemble des classes de D_{sd} ;
- $\Lambda_{sd}\subseteq \Lambda$ et $\forall(cl_1,cl_2)\in C_{sd}xC_{sd}$, $(cl_1,cl_2)\in\Lambda_{sd}$ si et seulement si $(cl_1,cl_2)\in\Lambda$;
- $\Gamma_{sd}\subseteq \Gamma$ et $\forall(cl_1,cl_2)\in C_{sd}xC_{sd}$, $(cl_1,cl_2)\in\Gamma_{sd}$ si et seulement si $(cl_1,cl_2)\in\Gamma$.

4. Complétude d'un sous-diagramme de classes

Un sous-diagramme de classes $D_{sd}(C_{sd}, \Lambda_{sd}, \Gamma_{sd})$ est *complet* dans $D(C, \Lambda, \Gamma)$ si une classe cl_i de C_{sd} dépend existentiellement d'une classe cl_j, alors cl_j appartient à C_{sd}.

$$\forall cl_i\in C_{sd}, \text{ si } \exists cl_j \in C \text{ tel que } (cl_i,cl_j)\in \Lambda_{sd} \text{ alors } cl_j\in C_{sd}$$

Exemple

Considérons le diagramme de classes $D_{M@TIS}$ de *M@TIS*, représenté par la Figure 14.

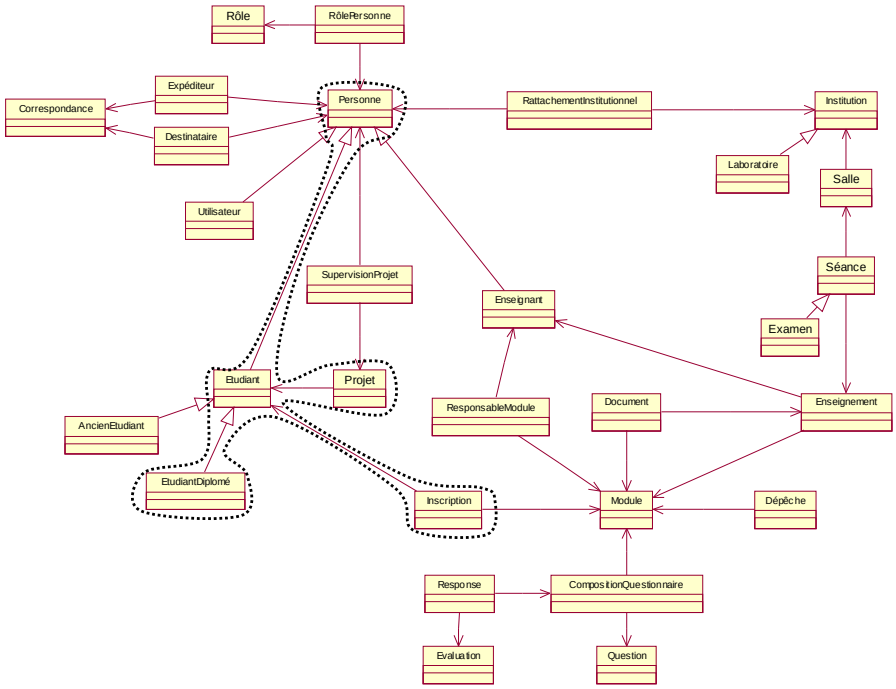

Figure 14 : $D_{M@TIS}$, le diagramme de classes de *M@TIS*

Soit $D_{résultats\_étudiant}$ le sous-diagramme de classes de $D_{M@TIS}$ construit sur l'ensemble de classes {*Personne*, *Étudiant*, *ÉtudiantDiplômé*, *Projet*, *Inscription*}. $D_{résultats\_étudiant}$ n'est pas complet dans $D_{M@TIS}$ car il existe un lien existentiel d'*Inscription* vers la classe *Module*. Pour que $D_{résultats\_étudiant}$ devienne complet dans $D_{M@TIS}$, il faudrait que la classe *Module* et que le lien (*Inscription*, *Module*) en fasse partie.

1.3.2. Au niveau des objets

1.3.1.3.1. Fonction type

O désigne l'ensemble des objets.

Nous définissons la fonction *type* $T: O \rightarrow C$ partout définie ; elle associe un objet à sa classe.

1.3.1.3.2. Objets directement reliés

Dans un état donné du SI, un objet o_1 de la classe cl_1 est *directement relié* à un objet o_2 de la classe cl_2 si :

- si les classes cl_1 et cl_2 sont adjacentes, et ;
- si connaissant les valeurs prises par o_1, il est possible d'identifier l'objet o_2.

1.3.1.3.3. Fonction Lien

La fonction *lien L* est définie sur *OxO* telle que $(o_1, o_2) \in L$ si o_1 est directement relié à o_2.

1.3.1.3.4. Objets reliés

Un objet o_1 est relié à un objet o_2, s'il existe une séquence d'objets $\{o_1, o_{11} \ldots o_{1i} \ldots o_n, o_2\}$ telle que :

- o_1 et o_{11} sont directement reliés ;
- o_n et o_2 sont directement reliés ;
- pour chaque paire d'objets o_{1i} et o_{1i+1} consécutifs de la séquence, o_{1i} et o_{1i+1} sont directement reliés.

Par convention, un objet est relié à lui même.

1.3.1.3.5. Fermeture d'un objet

La fermeture o^* d'un objet o désigne l'ensemble des objets reliés à o.

1.3.3. Espace statique

L'espace statique $ES(D(C, \Lambda, \Gamma), O, T, L)$ d'un système d'information est formellement défini par :

- un diagramme de classes $D(C, \Lambda, \Gamma)$,
- un ensemble d'objets O,
- une fonction type T,
- et une fonction lien L.

La Figure 15 représente le méta-modèle *binex*, exprimé en *binex*.

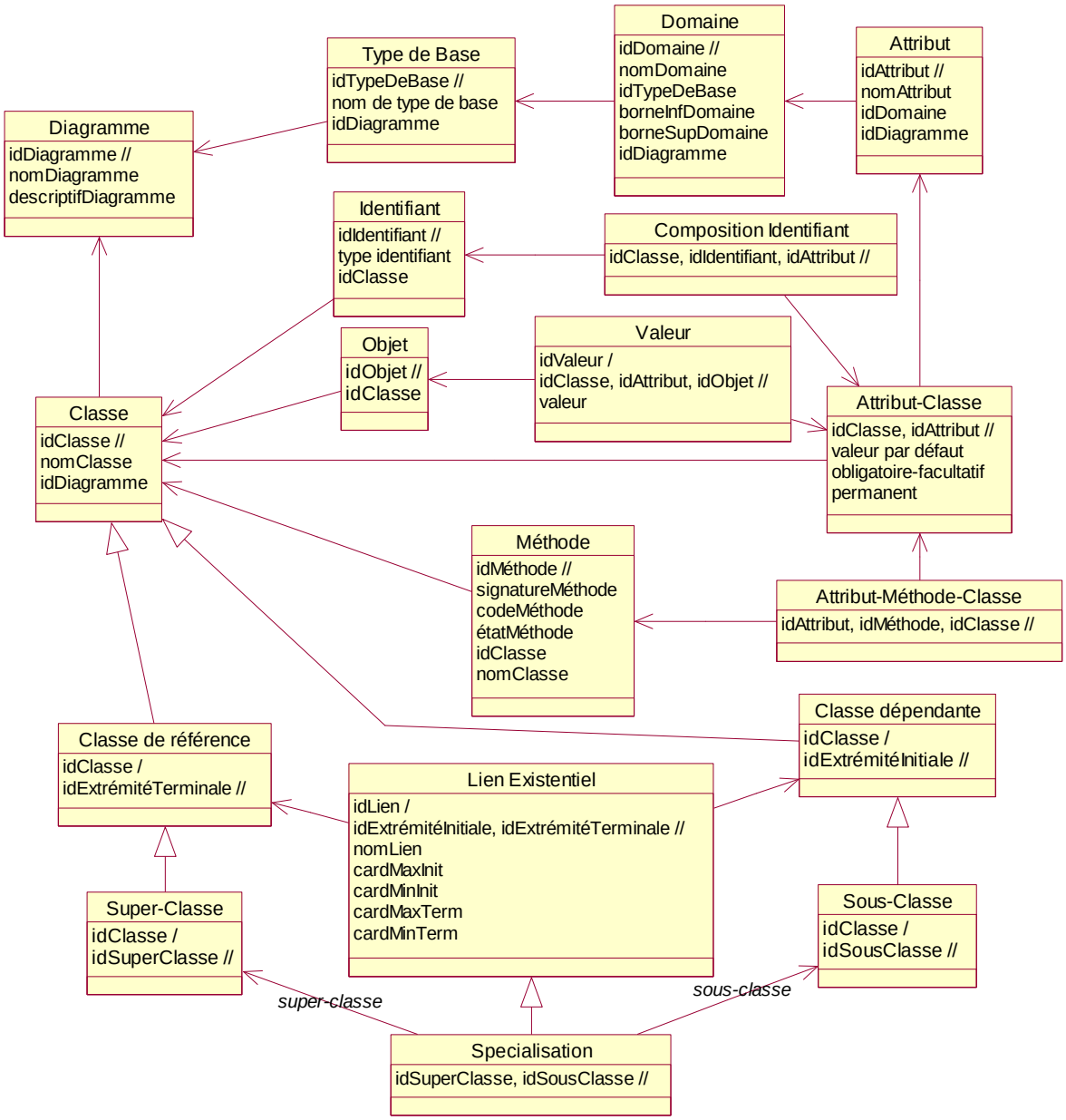

Figure 15 : Le méta-modèle *binex*

1.4. OPÉRATIONS SUR LES OBJETS DES CLASSES

Les opérations de base sur les objets des classes sont les trois méthodes standards associées implicitement à chacune des classes. Ces méthodes sont : (i) la *création*, (ii) la *suppression* et (iii) la *mise-à-jour* d'objets. Elles sont présentées sous forme de tableaux dont la première ligne indique les paramètres de l'opération, la deuxième les pré-conditions à son application et la troisième les post-actions sur les objets.

1.4.1. Création d'un objet

La création d'un objet o_A d'une classe cl_A consiste à attribuer des valeurs à ses attributs. Il est possible d'assigner une valeur par défaut à un ou plusieurs attributs de la classe. Chaque nouvel objet de la classe prend la valeur par défaut pour l'attribut concerné à sa création.

La création d'objet est soumise à un ensemble de conditions.

Créer un objet

Paramètres :	▪ La classe cl_A ; ▪ L'ensemble (v_1, v_2, … v_n) des valeurs prises par le nouvel objet pour les attributs (att_1, att_2, … att_n) de cl_A.
Pré-cond. :	▪ Les valeurs prises par les attributs identifiants sont toutes claires et vérifient la propriété d'unicité des identifiants de la classe ; ▪ Si cl_A dépend existentiellement d'une classe cl_B, il est nécessaire d'avoir un objet o_B de cl_B auquel o_A est relié, tel que o_A et o_B prennent les mêmes valeurs pour les attributs communs aux deux classes.
Post-act. *Objets :*	▪ Créer o_A dans cl_A ; ▪ Attribuer les valeurs v_1, v_2, … et v_n respectivement aux attributs att_1, att_2, … att_n de o_A.

Exemple de création d'objet

Étudiant est une sous-classe de *Personne* ; pour qu'un nouvel objet *Etu003* puisse être créé dans la classe *Étudiant*, il faut qu'il existe dans *Personne* un (super-)objet auquel *Etu003* est relié.

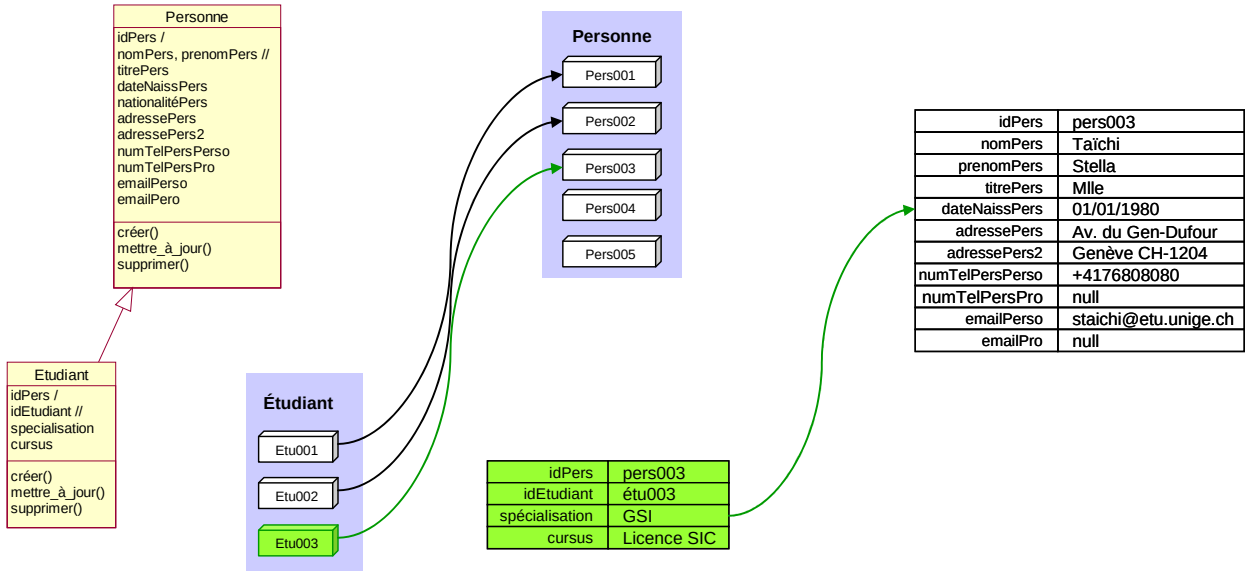

Figure 16 : Exemple de création d'objet dans la classe *Étudiant*

Dans la Figure 16, *Etu003* prend les valeurs *(pers003, étu003, GSI, Licence SIC)* pour les attributs *(idPers, idEtudiant, spécialisation, cursus)* de la classe *Étudiant*. Il est relié à l'objet

Pers003 de *Personne* : il prend la même valeur pour l'attribut *idPers* qui est commun aux deux classes.

{idPers} et *{idEtudiant}* sont les deux identifiants d'*Étudiant* ; les valeurs prises par *Etu003* pour chacun d'eux doivent être claires et uniques.

1.4.2. Suppression d'un objet

De part la nature existentielle des liens entre classes dans *binex*, la suppression d'un objet entraîne la suppression des objets qui en dépendent et de ses sous-objets.

Supprimer l'objet

Paramètres :	▪ La classe cl_A ; ▪ La valeur d'un identifiant de l'objet o_A à supprimer de cl_A.
Pré-cond. :	▪ Aucune
Post-act. *Objets :*	▪ S'il existe une classe cl_B qui dépend existentiellement de cl_A, supprimer tout objet o_B relié à o_A ; ▪ Appliquer récursivement l'opération de suppression à tout objet dépendant ou sous-objet supprimé.

Autrement dit, la suppression de l'objet o_A entraîne la suppression de tous les objets de la fermeture de o_A dans la zone de dépendance de cl_A.

Exemple de suppression d'objet

Dans l'exemple de la Figure 17, la zone de dépendance de la classe *Étudiant* est formée par l'ensemble des classes {*ÉtudiantDiplômé*, *Inscription*, *Projet, SuppervisionProjet*}.

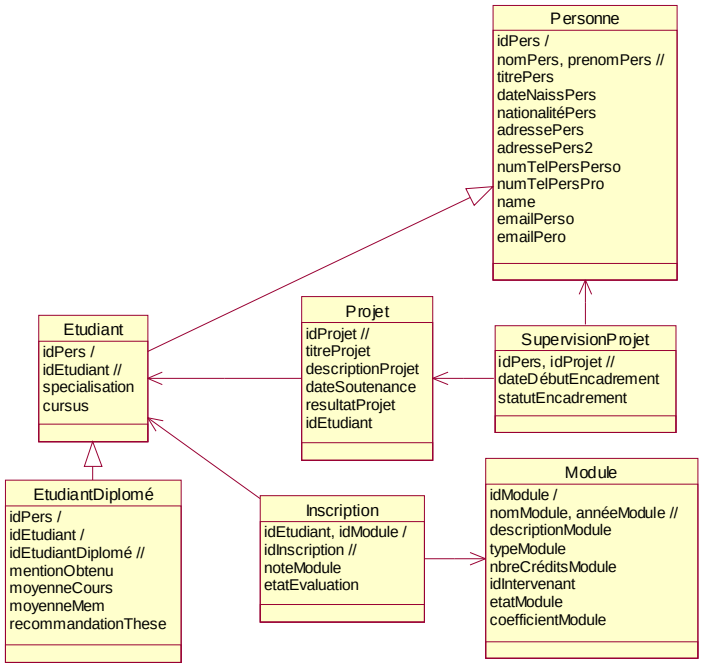

Figure 17 : Suivi de formation dans *M@TIS*

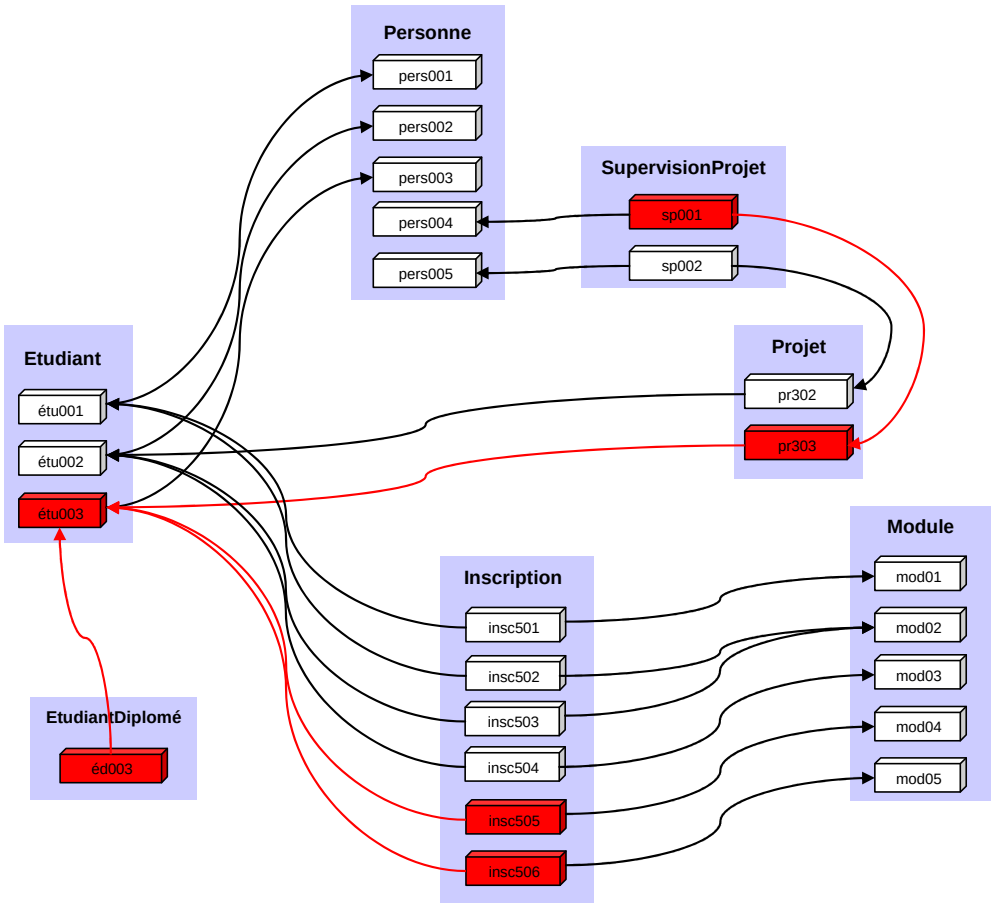

Figure 18 : Exemple de suppression d'un objet

La Figure 18 montre les objets des classes et les liens qui existent entre eux. La suppression de l'objet *étu03* entraîne la suppression de tous les objets de sa fermeture dans la zone de dépendance de la classe *Étudiant* : l'objet *éd003* de la classe *ÉtudiantDiplômé*, les objets *insc505* et *insc506* de la classe *Inscription*, l'objet *pr303* de la classe *Projet* et l'objet *sp001* de la classe *SupervisionProjet*.

1.4.3. Mise-à-jour d'un objet

Dans *binex*, pour toute classe et tout objet de la classe seule la mise à jour d'une valeur d'un attribut simple[71] est autorisée. Par ailleurs, il est nécessaire que deux objets liés ayant des attributs communs prennent les mêmes valeurs pour ces attributs communs.

Chaque classe admet au moins un identifiant (permanent et obligatoire), et les mécanismes d'accès aux objets et de navigation entre les objets se basent uniquement sur ces identifiants, qui avec les attributs référentiels matérialisent au niveau des objets les liens qui existent entrent leurs classes. La modification des valeurs prises par ces identifiants peut provoquer des incohérences et affecter la consistance des liens entre objets.

[71] Attribut qui n'est ni un attribut identifiant ni attribut référentiel.

Quand la modification de la valeur d'un attribut identifiant ou référentiel est inévitable, il est nécessaire de supprimer l'objet et de créer un nouvel objet avec les nouvelles valeurs.

Exiger que tous les identifiants des classes dans *binex* soient à la fois obligatoires et permanents est une solution qui garantit la cohérence des objets et des liens entre ces objets. En effet, ces liens (voir la section [1.2.1. Lien existentiel, Page 28]) sont basés sur les valeurs prises par les attributs identifiants et référentiels.

Si la mise à jour de la valeur prise par un attribut faisant partie d'un identifiant permanent est nécessaire, il est nécessaire de créer un nouvel objet et de supprimer l'objet initialement à mettre à jour.

Mettre à jour l'objet (attributs simples)

Paramètres :	▪ La classe cl_A ; ▪ La valeur d'un identifiant de l'objet o_A de cl_A à mettre à jour ; ▪ L'ensemble (v_1, v_2, … v_n) des nouvelles valeurs prises par o_A pour les attributs (att_1, att_2, … att_n) de cl_A.
Pré-cond. :	▪ Aucune valeur d'un attribut identifiant ou référentiel de l'objet o_A n'est modifiée.
Post-act. *Objets :*	▪ Modifier les valeurs prises par o_A pour ses attributs simples.

Exemples de mise-à-jour d'objet

Les attributs *noteModule* et *étatÉvaluation* peuvent être mis à jour dans tous les objets de la classe *Inscription*. Les attributs *idÉtudiant* et *idModule* ne peuvent être mis à jour : ils font partie de l'identifiant de la classe.

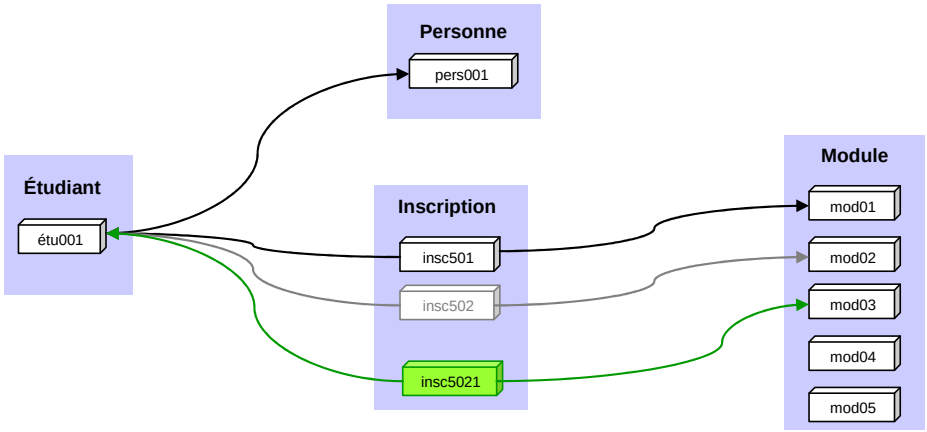

Figure 19 : Exemple de mise-à-jour d'objet

Toutefois, si par exemple (Figure 19) l'inscription de l'étudiant *étu001* au module *mod02* devait être mise à jour pour corriger une erreur et remplacée par une inscription au module *mod03*, il serait nécessaire de supprimer l'objet *inscr502* et de créer un nouvel objet *inscr5021* reliant *étu001* à *mod03* ; *inscr5021* prendrait les mêmes valeurs pour *noteModule* et *étatÉvaluation* que *inscr502*.

2. ÉVOLUTION DANS LE MÉTA-MODÈLE *BINEX*

Dans un système d'information, le modèle informationnel tient une place centrale entre d'une part les activités métier qu'il supporte et d'autre part sa réalisation informatique. Maintenir l'alignement de ce modèle par rapport au métier et par rapport à l'informatique nécessite qu'il puisse évoluer au cours du temps. L'évolution du modèle peut s'avérer nécessaire pour s'adapter à des changements du côté du métier supporté, corriger des erreurs de conception ou améliorer les spécifications.

« Prendre en compte l'évolution de schémas dans un système à objets, c'est admettre que celui-ci peut évoluer dans le temps. C'est donc reconnaître que la modélisation d'un système ou d'une application peut changer, soit parce qu'elle ne correspond plus à celle utilisée jusqu'à alors, soit parce que des erreurs y ont été découvertes » (Nguyen, 1997)[72]. Dans ce sens, *binex* est doté d'un ensemble complet d'opérations pour la manipulation et l'évolution des modélisations informationnelles. Cet ensemble est obtenu par l'application des trois opérateurs génériques (i) *créer*, (ii) *supprimer* et (iii) *mettre à jour* à chacun des éléments du modèle *binex*.

Nous avons défini ces opérations de manière à garantir même en cas d'évolution le maintien de la cohérence, de la rigueur, de la non-ambiguïté de la modélisation de SI et de sa fidélité au domaine qu'elle représente.

2.1. UNIFORMITÉ DES OBJETS DANS *BINEX*

Dans *M@TIS*, la suppression de l'objet *étu03* de la classe *Étudiant* (Figure 18) entraîne la suppression de tous les objets de sa fermeture dans sa zone de dépendance: l'objet *éd003* de la classe *ÉtudiantDiplômé*, les objets *insc505* et *insc506* de la classe *Inscription*, l'objet *pr303* de la classe *Projet* et l'objet *sp001* de la classe *SupervisionProjet*.

La suppression de la classe *Étudiant* entraîne la suppression :

1. des objets d'*Étudiant ;*
2. des identifiants { *Étudiant.idÉtudiant* } { *Étudiant.idPers*} d'*Étudiant* ;
3. des occurrences d'attributs *Étudiant.idÉtudiant*, *Étudiant.idPers, Étudiant.spécialisation* et *Étudiant.cursus* dans *Étudiant* ;
4. des liens de et vers Étudiant : (*Étudiant* → *Personne*), (*Projet* → *Étudiant*), (*Inscription* → *Étudiant*) et (*Projet* → *ÉtudiantDiplômé*) ;
5. des méthodes d'*Étudiant* ;
6. sa sous-classe *ÉtudiantDiplômé* et
7. de la classe *Étudiant* elle même.

---

[72] (Nguyen, 1997) Nguyen G. T., Objets et évolution, Chapitre 6, Ingénierie Objet : Objets, techniques, concepts (Sous la direction de Oussalah C.), P205-232. InterEditions, Collection Informatiques, 1997. ISBN : 2729606424.

La classe *Étudiant* est à la fois une classe et un objet de la classe *Classe* du méta-modèle *binex*. La surpression de l'objet *Étudiant* de *Classe* entraîne la suppression de tous les objets qui lui sont existentiellement liés : ses identifiants, ses occurrences, ses méthodes, ses sous-classes et les liens où *Étudiant* est une extrémité initiale ou terminale.

Il existe donc un comportement commun entre la suppression d'un étudiant (objet du SI) et la suppression d'*Étudiant* (classe, et objet du méta-modèle) : la suppression récursive des objets dépendants. Les objets du méta-modèle ne sont pas différents des objets du SI. Les opérations de création, de suppression et de mise à jour leur sont applicables de la même manière. Il existe toutefois un comportement spécifique aux objets du méta-modèle : la manipulation des objets du méta-modèle doit prendre en compte les objets du SI.

« Nous disons qu'il y a uniformité d'accès et de manipulation quand il n'y a pas de distinction entre la manipulation d'information relatives au domaine du SI et la manipulation d'information relatives au modèle ou au méta-modèle de SI » (Al-Jadir 1997)[73]. Dans ce sens, le méta-modèle *binex* est conforme au modèle *binex* : les objets du SI et les objets du modèle de SI sont représentés sous une forme similaire. Les mêmes méthodes de base sont utilisées pour manipuler à la fois les objets du SI et les objets du méta-modèle *binex*. Ceci rend le modèle *binex* uniforme, flexible et facilement extensible.

2.2. COMPLÉTUDE DES OPÉRATIONS D'ÉVOLUTION

« Un ensemble d'opérations d'évolution de modèle est dit *complet* s'il contient les primitives *créer()*, *supprimer()* et *mettre-à-jour()* définies sur les objets de chaque classe du méta-modèle (..) Cette définition est plus forte que celle de (Banerjee & al., 1987)[74] qui considère un ensemble d'opérations complet s'il est possible de transformer un modèle M1 en n'importe quel modèle M2 à travers une séquence finie de ses opérations ; cette définition ne prend pas en compte les objets du SI » (Al-Jadir 1997).

L'ensemble des opérations définies sur le méta-modèle *binex* est obtenu par application des méthodes de base *créer()*, *supprimer()* et *mettre-à-jour()* associées à toute classe dans binex, tout en tenant compte des objets du SI : cet ensemble est complet.

2.3. OPÉRATIONS SUR LE MODÈLE BINEX

Les opérations sur le modèle sont présentées sous forme de tableaux à quatre lignes. Les premières lignes indiquent les paramètres de chaque opération et les deuxièmes les pré-conditions à son application. Les troisièmes et quatrièmes lignes indiquent respectivement les

---

[73] (Al-Jadir, 1997) Al-Jadir L., Evolution-oriented database systems, PhD thesis, Uni. de Genève, 1997.

[74] (Banerjee & al., 1987) BanerjeeJ. Kim W., Kim H-J., Korth H.F., Semantics and Implementation of Schema Evolution in object-oriented databases. Proc. Int. Conf. On Management of Data, ACM SIGMOD, San Francisco 1987.

post-actions à entreprendre sur la modélisation et sur les objets pour maintenir la cohérence de la modélisation.

2.3.1. Diagramme de classes

Créer un diagramme de classes

Paramètres :	▪ Nom du diagramme de classes *D* ; ▪ Description de *D*.
Pré-cond. :	▪ Le nom de *D* n'est pas déjà utilisé pour un autre diagramme.
Post-act. Modèle :	▪ Créer *D*.
Post-act. Objets :	▪ Aucune.

Supprimer un diagramme de classes

Paramètres :	▪ Nom du diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer toutes les classes de *D* ; ▪ Supprimer les attributs de *D* ; ▪ Supprimer les domaines de *D* ; ▪ Supprimer tous les types de base de *D ;* ▪ Supprimer *D*.
Post-act. Objets :	▪ Supprimer tous les objets de *D*.

Renommer le diagramme de classes

Paramètres :	▪ Nouveau nom du diagramme de classes *D*.
Pré-cond. :	▪ Le nouveau nom de *D* n'est pas déjà utilisé pour un autre diagramme de classes.
Post-act. Modèle :	▪ Renommer *D*.
Post-act. Objets :	▪ Aucune.

2.3.2. Classe

Créer une classe

Paramètres :	▪ Nom de la classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Le nom de *cl* est unique dans *D*.
Post-act. Modèle :	▪ Créer *cl* dans D.
Post-act. Objets :	▪ Aucune.

Supprimer une classe

La suppression d'une classe *cl* entraîne la suppression de son sous-arbre de spécialisation.

Paramètres :	▪ La classe *cl*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ si *cl* est l'extrémité initiale d'un lien existentiel *lex*, alors supprimer *lex* ; ▪ si *cl* est l'extrémité terminale d'un lien existentiel *lex*, alors supprimer *lex* ; ▪ si *cl* a un identifiants *id*, alors supprimer l'identifiant *id* ; ▪ supprimer l'occurrence *cl.att* de tout attribut att dans *cl* ; ▪ supprimer toute méthode de *cl*.
Post-act. Objets :	▪ Supprimer les objets appartenant à la fermeture de chacun des objets de *cl* dans la zone de dépendance de *cl* ; ▪ Supprimer tous les objets de *cl*.

Il est possible de limiter l'impact de la suppression d'une classe sur une modélisation en détachant les sous-arbres de spécialisation de ses sous classes (op. « Supprimer le lien de spécialisation-généralisation ») et en les rattachant directement aux super-classes de la classes à supprimer (op. « Créer un lien de spécialisation-généralisation ») ;

Renommer la classe

Paramètres :	▪ La classe *cl* ; ▪ Le nouveau nom de *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Le nouveau nom de *cl* est unique dans *D*.
Post-act. Modèle :	▪ Renommer *cl*.
Post-act. Objets :	▪ Aucune.

2.3.3. Type de base

Créer un nouveau type de base

Paramètres :	▪ Le nom du nouveau type de base *tb* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Le nom de *tb* n'est pas utilisé dans *D*.
Post-act. Modèle :	▪ Créer *tb* dans *D*.
Post-act. Objets :	▪ Aucune.

Supprimer un type de base existant

Paramètres :	▪ Le type de base *tb* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucun domaine de *D* n'utilise *tb*.
Post-act. Modèle :	▪ Supprimer *tb*.
Post-act. Objets :	▪ Aucune.

Renommer le type de base

Paramètres :	▪ Le type de base *tb* ; ▪ Le nouveau nom de *tb* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucun domaine de *D* n'utilise *tb* ; ▪ Le nouveau nom de *tb* est unique dans *D*.
Post-act. Modèle :	▪ Renommer *tb*.
Post-act. Objets :	▪ Aucune.

2.3.4. Domaine

Créer un domaine

Paramètres :	▪ Le nom du domaine *dom* ; ▪ *tb*, le type de base de *dom* ; ▪ Les bornes inférieure et supérieure de *dom* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *tb* appartient à *D* ; ▪ Le nom de *dom* n'est pas déjà utilisé pour un autre domaine de *D*.
Post-act. Modèle :	▪ Créer *dom* dans *D*.
Post-act. Objets :	▪ Aucune.

Supprimer un domaine

Paramètres :	▪ Le domaine *dom* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *dom* n'est le domaine d'aucun attribut *de D*.
Post-act. Modèle :	▪ Supprimer *dom* de *D.*
Post-act. Objets :	▪ Aucune.

Renommer le domaine

Paramètres :	▪ Le domaine *dom* ; ▪ Le nouveau nom de *dom* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Le nouveau nom du domaine *dom* n'est pas déjà utilisé pour un autre domaine de *D*.
Post-act. Modèle :	▪ Renommer *dom*.
Post-act. Objets :	▪ Aucune.

Modifier le type de base d'un domaine

Paramètres :	▪ Le domaine *dom* ; ▪ Le nouveau type de base *tb* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *tb* inclut le type de base initial de *dom* (par exemple : Entier →Réel ; Caractère → chaîne de caractères ; etc.)
Post-act. Modèle :	▪ Modifier le type de base de *dom*.
Post-act. Objets :	▪ Aucune.

Modifier la borne inférieure du domaine

Paramètres :	▪ Le domaine *dom* ; ▪ La borne inférieure de *dom* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ La nouvelle borne inférieure est inférieure à la borne inférieure initiale de *dom* ; ▪ La nouvelle borne inférieure est inférieure à la borne supérieure de *dom.*
Post-act. Modèle :	▪ Modifier la borne inférieure de *dom*.
Post-act. Objets :	▪ Aucune.

Modifier la borne supérieure du domaine

Paramètres :	▪ Le domaine *dom* ; ▪ La borne supérieure de *dom* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ La nouvelle borne supérieure est supérieure à la borne supérieure initiale de *dom* ; ▪ La nouvelle borne supérieure est supérieure à la borne inférieure de *dom.*
Post-act. Modèle :	▪ Modifier la borne supérieure de *dom*.
Post-act. Objets :	▪ Aucune.

Modifier la longueur maximale du domaine (pour un domaine de type « chaîne de caractères »)

Paramètres :	▪ Le domaine *dom* ; ▪ La nouvelle longueur maximale ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ La nouvelle longueur maximale est supérieure à la borne supérieure initiale de *dom*.
Post-act. Modèle :	▪ Modifier la longueur maximale de *dom*.
Post-act. Objets :	▪ Pour chaque objet *o* d'un classe *cl* ayant un attribut *att* qui a pour domaine *dom*, si la taille de la chaîne de caractère *att* dans *o* a une taille supérieure à la nouvelle longueur maximale de *dom*, tronquer la chaîne de caractères dans *o*.

2.3.5. Attribut

Créer un nouvel attribut

Paramètres :	▪ Nom de l'attribut *att* ; ▪ *dom*, domaine d'*att* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Le nom d'*att* est unique dans *D*.
Post-act. Modèle :	▪ Créer *att* dans *D*.
Post-act. Objets :	▪ Aucune.

Supprimer un attribut

Paramètres :	▪ Nom de l'attribut *att* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ L'attribut *att* n'appartient à aucun identifiant dans *D*.
Post-act. Modèle :	▪ Supprimer toute occurrence d'*att* dans les classes *cl* de *D*.
Post-act. Objets :	▪ Supprimer la valeur prise pour *att* dans tout objet de *cl*.

Renommer l'attribut

Paramètres :	▪ L'attribut *att* ; ▪ le nouveau nom de l'attribut ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Le nouveau nom d'*att* est unique dans *D*.
Post-act. Modèle :	▪ Renommer *att*.
Post-act. Objets :	▪ Aucune.

Modifier le domaine de l'attribut

Paramètres :	▪ L'attribut *att* ; ▪ Le nouveau domaine d'*att* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Les ancien et nouveau domaines sont construits sur le même type de base ; ▪ L'ancien domaine de l'attribut est inclus dans le nouveau domaine.
Post-act. Modèle :	▪ Modifier le domaine d'*att*.
Post-act. Objets :	▪ Aucune.

2.3.6. Attribut-Classe : occurrence d'attribut dans une classe

Créer une occurrence d'un attribut dans une classe

Paramètres :	▪ La classe *cl* ; ▪ L'attribut *att* ; ▪ Les propriétés d'*att* dans *cl* : valeur par défaut, obligatoire ou facultatif, permanent ou pas. ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ L'attribut *att* n'appartient pas déjà à *cl* ;
Post-act. Modèle :	▪ Définir *cl.att*.
Post-act. Objets :	▪ A chaque objet *o* de *cl* attribuer, si elle est définie, la valeur par défaut pour *cl.att*.

Supprimer l'occurrence d'un attribut dans la classe

Paramètres :	▪ *cl.att*, l'occurrence de l'attribut *att* dans la classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ L'attribut *att* n'appartient à aucun identifiant de *cl* ;
Post-act. Modèle :	▪ Dissocier *att* de *cl*.
Post-act. Objets :	▪ Supprimer la valeur prise par *att* de tout objet de *cl*.

Mettre-à-jour l'occurrence d'un attribut dans une classe

Paramètres :	▪ *cl.att*, l'occurrence de l'attribut *att* dans la classe *cl* ; ▪ Les nouvelles propriétés de *cl.att* : valeur par défaut, obligatoire ou facultatif, permanent ou pas. ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Si *att* fait partie d'un identifiant de *cl*, il ne peut ni devenir facultatif ou non-permanent.
Post-act. Modèle :	▪ Mettre-à-jour *cl.att*.
Post-act. Objets :	▪ Si *cl.att* devient obligatoire, à chaque objet *o* de *cl* attribuer, si elle est définie, la valeur par défaut pour *cl.att* ; ▪ Si une nouvelle valeur par défaut est définie sur *cl.att*, à chaque objet *o* de *cl* prenant une valeur obscure pour *att*, attribuer la valeur par défaut pour *cl.att*.

2.3.7. Identifiant d'une classe

Créer un nouvel identifiant d'une classe

Paramètres :	▪ la classe *cl ;* ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Créer le nouvel identifiant.
Post-act. Objets :	▪ Aucune.

Supprimer l'identifiant d'une classe

Paramètres :	▪ La classe *cl* ; ▪ L'identifiant *id* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* admet au moins un autre identifiant qu'*id*.
Post-act. Modèle :	▪ Supprimer la composition d'*id* : les objets de *Composition Identifiant* dépendants d'*id* ; ▪ Supprimer *id*.
Post-act. Objets :	▪ Aucune.

2.3.8. Composition Identifiant

Ajouter un attribut à l'identifiant de la classe

Paramètres :	▪ L'identifiant *id* de la classe *cl* ; ▪ *cl.att*, l'occurrence de l'attribut *att* dans *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Les objets de *cl* satisfont le nouvel identifiant.
Post-act. Modèle :	▪ Ajouter *cl.att* à *id*.
Post-act. Objets :	▪ Aucune.

Supprimer un attribut de l'identifiant de la classe

Paramètres :	▪ L'identifiant *id* de la classe *cl* ; ▪ *cl.att*, l'occurrence de l'attribut *att* dans *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Les objets de *cl* satisfont le nouvel identifiant.
Post-act. Modèle :	▪ Supprimer *cl.att* d'*id*.
Post-act. Objets :	▪ Aucune.

2.3.9. Classe de référence

Créer une classe de référence

Paramètres :	▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* n'est pas une classe de référence.
Post-act. Modèle :	▪ Spécialiser *cl* en classe de référence.
Post-act. Objets :	▪ Aucune.

Supprimer une classe de référence

Paramètres :	▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* est spécialisée en classe de référence ; ▪ Il n'existe pas de lien existentiel ayant *cl* comme extrémité terminale.
Post-act. Modèle :	▪ Supprimer le sous-objet de *cl* dans *Classe de référence*.
Post-act. Objets :	▪ Aucune.

2.3.. Classe dépendante

Créer une classe dépendante

Paramètres :	▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* n'est pas une classe dépendante.
Post-act. Modèle :	▪ Spécialiser *cl* en classe dépendante.
Post-act. Objets :	▪ Aucune.

Supprimer une classe dépendante

Paramètres :	▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* est spécialisée en classe de référence ; ▪ Il n'existe pas de lien existentiel ayant *cl* comme extrémité initiale.
Post-act. Modèle :	▪ Supprimer le sous-objet de *cl* dans *Classe dépendante*.
Post-act. Objets :	▪ Aucune.

2.3.10. Lien Existentiel

Créer un lien existentiel

Paramètres :	▪ cl_{dep}, une classe dépendante ; ▪ cl_{ref}, une classe de référence ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ cl_{dep} n'est pas liée existentiellement à cl_{ref} ; ▪ cl_{ref} n'est pas liée existentiellement à cl_{dep} ; ▪ cl_{dep} admet parmi ses attributs les attributs d'au moins un des identifiants de cl_{ref}.
Post-act. Modèle :	▪ Créer un lien existentiel de cl_{dep} vers cl_{ref}.
Post-act. Objets :	▪ Pour chaque objet $o_{i,dep}$ de cl_{dep}, s'il n'existe pas un objet $o_{j,ref}$ de cl_{ref} prenant les mêmes valeurs pour les attributs communs à cl_{dep} et cl_{ref}, supprimer $o_{i,dep}$.

Supprimer un lien existentiel

Paramètres :	▪ Le lien existentiel *lienex* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer *lienex* de *D*.
Post-act. Objets :	▪ Aucune.

Modifier les cardinalités du lien existentiel

Paramètres :	▪ Le lien existentiel *lienex* ; ▪ Les nouvelles cardinalités de *lienex* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ La nouvelle cardinalité minimale (respectivement maximale) de *lienex* est inférieure (respectivement supérieure) à la sa minimale (respectivement maximale) cardinalité actuelle ; ▪ La nouvelle cardinalité minimale de *lienex* est inférieure à sa cardinalité maximale.
Post-act. Modèle :	▪ Modifier les cardinalités de *linex*.
Post-act. Objets :	▪ Aucune.

2.3.11. Super-classe

Créer une super-classe

Paramètres :	▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* est une classe de référence ; ▪ *cl* n'est pas une super-classe.
Post-act. Modèle :	▪ Spécialiser *cl* en sous-classe.
Post-act. Objets :	▪ Aucune.

Supprimer une super-classe

Paramètres :	▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* est une classe dépendante ; ▪ *cl* n'est pas une sous-classe.
Post-act. Modèle :	▪ Supprimer le sous-objet de *cl* dans *Classe de référence*.
Post-act. Objets :	▪ Aucune.

2.3.12. Sous-classe

Créer une sous-classe

Paramètres :	▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* est un en classe dépendante ; ▪ *cl* n'est pas une sous-classe.
Post-act. Modèle :	▪ Spécialiser *cl* en classe dépendante.
Post-act. Objets :	▪ Aucune.

Supprimer une sous-classe

Paramètres :	▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl* est spécialisée en classe de référence ; ▪ Il n'existe pas de lien existentiel ayant *cl* comme extrémité initiale.
Post-act. Modèle :	▪ Supprimer le sous-objet de *cl* dans *Sous-Classe*.
Post-act. Objets :	▪ Aucune.

2.3.13. Spécialisation

Créer une spécialisation-généralisation

Paramètres :	▪ cl_{ss}, la sous-classe ; ▪ cl_{sp}, la super-classe ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ cl_{ss} est liée existentiellement à cl_{sp} par un lien existentiel *lienex* ; ▪ cl_{sp} n'est pas un descendant de cl_{ss} (pour éviter les boucles dans l'arbre de spécialisation).
Post-act. Modèle :	▪ Spécialiser *linex* en un lien de spécialisation-généralisation de cl_{ss} à cl_{sp}.
Post-act. Objets :	▪ Aucune.

Supprimer une spécialisation-généralisation

Paramètres :	▪ La classe cl_{ss}, ▪ La classe cl_{sp}, super-classe de cl_{ss}; ▪ Le lien de spécialisation-génralisation *liensg* reliant cl_{ss} à cl_{sp} ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucune
Post-act. Modèle :	▪ Supprimer *liensg* ; le lien existentiel de cl_{ss} à cl_{sp} est maintenu dans *D*.
Post-act. Objets :	▪ Aucune.

Modifier la super-classe d'une classe

Paramètres :	▪ La classe cl_{ss}, ▪ La nouvelle super-classe cl_{sp} ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ La nouvelle super-classe cl_{sp} est dans l'arbre de spécialisation de cl_{ss} et n'est pas un descendant de cl_{sp} (pour éviter les cycles).
Post-act. Modèle :	▪ Supprimer le lien de spécialisation-généralisation qui existe entre cl_{ss} et son ancienne super-classe. ▪ Créer un lien de spécialisation-généralisation entre cl_{ss} et cl_{sp}.
Post-act. Objets :	▪ Pour chaque objet *o* de cl_{ss}, si *o* n'est pas relié à un objet de la nouvelle super-classe cl_{sp}, supprimer *o* de cl_{ss} et les objets des descendants de cl_{ss}, qui lui sont reliés.

Du moment que les classes cl_{ss} et cl_{sp} sont dans le même arbre de spécialisation, chaque objet de cl_{ss} et cl_{sp} est relié à un objet dans la classe racine de leur arbre de spécialisation.

2.3.14. Méthode

Créer une nouvelle méthode

Paramètres :	▪ Nom de la méthode *mth* ; ▪ La classe *cl* ; ▪ Le traitement associé à *mth* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Le nom de *mth* n'est pas déjà utilisé par une autre méthode dans *cl*.
Post-act. Modèle :	▪ Créer la nouvelle méthode *mth* ; ▪ Pour chaque attribut *cl.att* invoqué dans *mth*, créer un objet dans *Attribut-Méthode-Classe*.
Post-act. Objets :	▪ Aucune.

Supprimer une méthode

Paramètres :	▪ La méthode *mth* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer tous les objets de *Attribut-Méthode-Classe* reliés à *mth* ; ▪ Supprimer *mth*.
Post-act. Objets :	▪ Aucune.

Renommer une méthode

Paramètres :	▪ La méthode *mth* ; ▪ Le nouveau nom de *mth* ; ▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Le nouveau nom de *mth* n'est pas déjà utilisé par une autre méthode dans *cl*.
Post-act. Modèle :	▪ Renommer *mth*.
Post-act. Objets :	▪ Aucune.

Activer / désactiver une méthode

Paramètres :	▪ La méthode *mth* ; ▪ Le nouvel état de *mth* ; ▪ La classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Activer ou désactiver *mth*.
Post-act. Objets :	▪ Aucune.

2.3.15. Attribut-Méthode-Classe

Créer un objet dans Attribut-Méthode-Classe

Paramètres :	▪ La méthode *mth* ; ▪ *cl.att*, l'occurrence de l'attribut *att* dans *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ *cl.att* est invoquée dans *mth*.
Post-act. Modèle :	▪ Créer un objet dans *Attribut-Méthode-Classe* reliant *cl.att* à *mth*.
Post-act. Objets :	▪ Aucune.

Supprimer un objet d'Attribut-Méthode-Classe

Paramètres :	▪ La méthode *mth* ; ▪ *cl.att*, l'occurrence de l'attribut *att* dans *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer L'objet d'*Attribut-Méthode-Classe* reliant *cl.att* à *mth*.
Post-act. Objets :	▪ Aucune.

La Figure 20 représente le méta-modèle *binex* muni de ses opérations d'évolution.

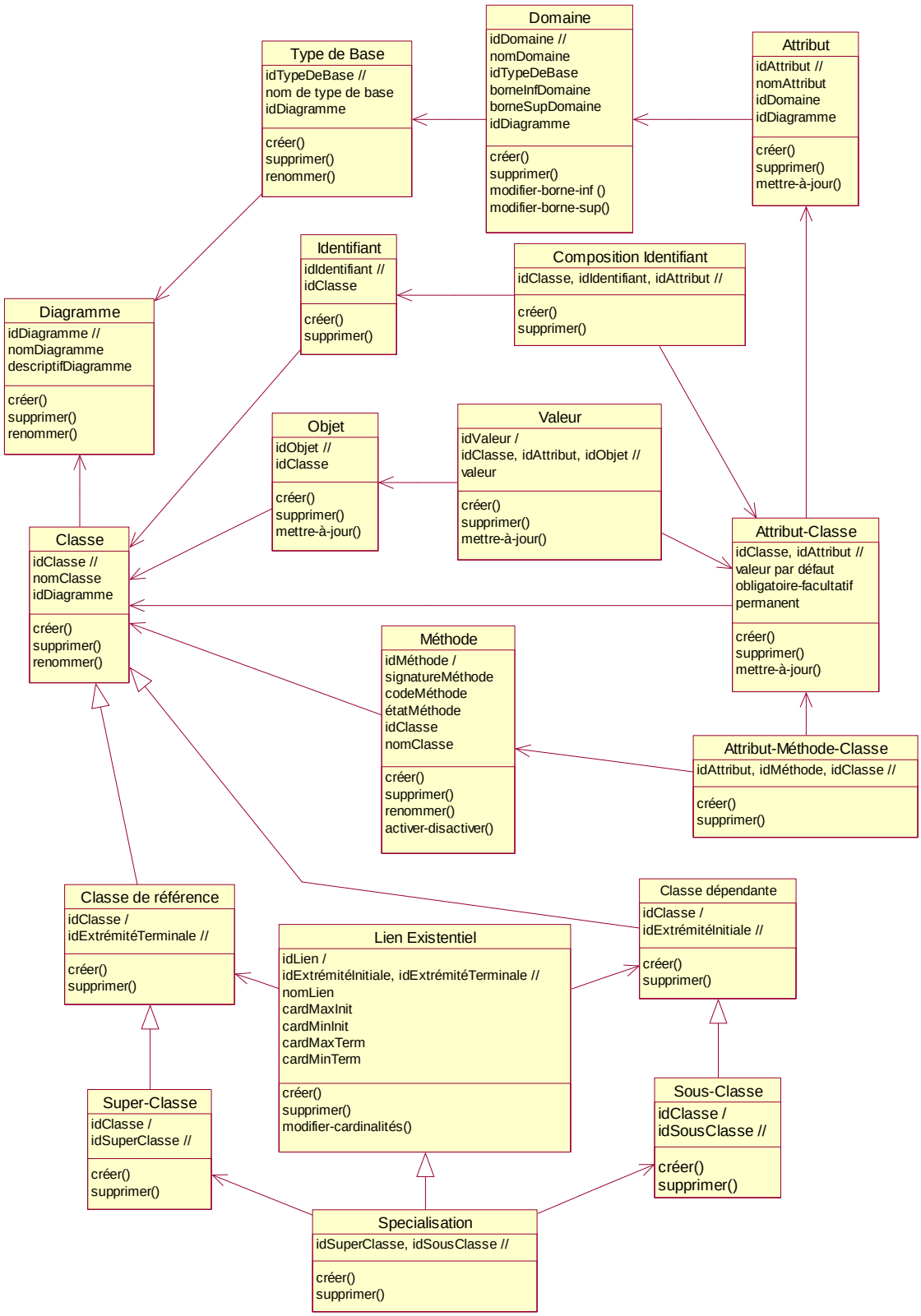

Figure 20 : Le méta-modèle *binex* muni de ses opérations d'évolution

CONCLUSION

L'ingénierie d'un SI implique en général des acteurs et des communautés d'acteurs souvent hétérogènes. Disposer d'un formalisme de modélisation rigoureux et non-ambiguë est une nécessité pour dépasser cette hétérogénéité.

Le modèle *binex*, que nous avons introduit dans ce chapitre, est construit à partir d'un minimum de concepts simples : des classes, des objets, les liens existentiels ou de spécialisation généralisation. Ce nombre réduit de concepts est un avantage lors de (i) l'analyse pour l'étude de la surface informationnelle sur laquelle est construit le SI ; (ii) lors de l'implantation d'un modèle : les concepts utilisés sont directement implantables ; et lors de (iii) l'évolution du SI : *binex* est muni d'un ensemble complet d'opérations d'évolution.

Dans le chapitre suivant, nous introduisons le concept d'hyperclasse que nous proposons pour maîtriser de façon modulaire l'étendue sans cesse croissante des SIs.

CHAPITRE II

HYPERCLASSE

INTRODUCTION

Le modèle *binex* est construit autour des concepts de classe, de lien entre ces classes, d'objet et de lien entre objets. Cependant, le niveau de classe et d'objet s'avère souvent trop restreint dans un modèle informationnel pour prendre en considération des concepts complexes du domaine du SI. Dans *M@TIS*, par exemple, l'évaluation et la communication des résultats académiques d'un étudiant de la formation nécessitent des informations sur l'étudiant (nom, prénom, adresse et spécialisation), les notes qu'il a obtenues pour les différents modules où il est inscrit et pour son projet de recherche. Ces informations sont réparties dans les classes *Étudiant*, *Personne*, *Inscription*, *Module* et *ÉtudiantDiplômé*. De la même manière, la gestion des enseignements et la planification des séances de cours ou d'examen nécessitent en même temps des informations sur les modules, les enseignants et les salles. Ces informations sont représentées dans les classes *Module*, *Enseignement*, *Séance*, *Examen*, *Salle*, *Institution*, *Enseignant* et *Personne.*

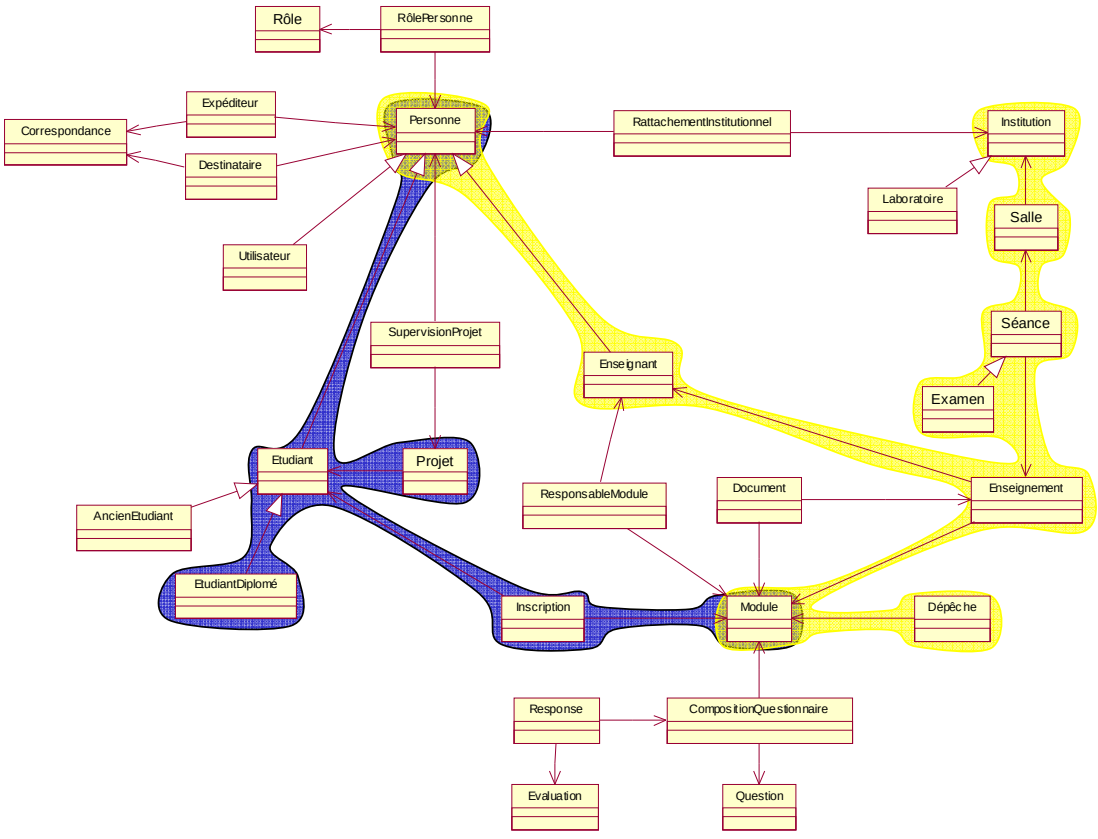

Figure 21 : Exemples d'hyperclasses dans le diagramme de classes de *M@TIS*

Dans un diagramme de classes, nous proposons de modéliser les concepts qui impliquent plus qu'une classe par des *hyperclasses*. Une hyperclasse *hcl* est définie par :

1. un diagramme d'hyperclasse D_{hcl} : *hcl* est définie sur un sous-diagramme du diagramme de classes ; D_{hcl} regroupe les classes et les liens entre les classes de *hcl* ainsi que les occurrences d'attributs et les méthodes de classes visibles au niveau de *hcl*. Un graphe de navigation associé à *hcl* décrit la navigation entre les objets des classes de *hcl* ;
2. un ensemble d'hyperobjets : un hyperobjet est une instance de *hcl* ; il est construit par navigation à partir des objets des classes de *hcl* ;
3. un ensemble d'hyperattributs : un hyperattribut est un attribut de *hcl* ; il est défini à partir d'une occurrence d'un attribut dans une classe de *hcl*, et
4. un ensemble d'hyperméthodes : une hyperméthode est une méthode associée à *hcl* et qui implique une ou plusieurs de ses classes.

Le concept d'hyperclasse est une généralisation du concept traditionnel de *classe* de l'approche orientée objet.

Sur le diagramme de classes de *M@TIS* (Figure 21), nous définissons deux hyperclasses : (i) *hcl_résultats_étudiant*, hyperclasse dédiée à la gestion des résultats et à la préparation des relevés de notes des étudiants de la formation ; elle est construite sur les classes *Étudiant*, *Personne*, *Inscription*, *Module* et *ÉtudiantDiplômé,* et (ii) *hcl_planning*, hyperclasse construite sur les classes *Module*, *Enseignement*, *Séance*, *Examen*, *Salle*, *Institution*, *Enseignant*, et *Personne* ; elle permet au coordinateur de planifier les séances de cours.

La Figure 22 représente $D_{hcl\_résultats\_étudiant}$, le diagramme de *hcl_résultats_étudiant*. $D_{hcl\_résultats\_étudiant}$ montre les classes de *hcl_résultats_étudiant* et les liens entre elles, les attributs et les méthodes de ces classes visibles dans *hcl_résultats_étudiant*.

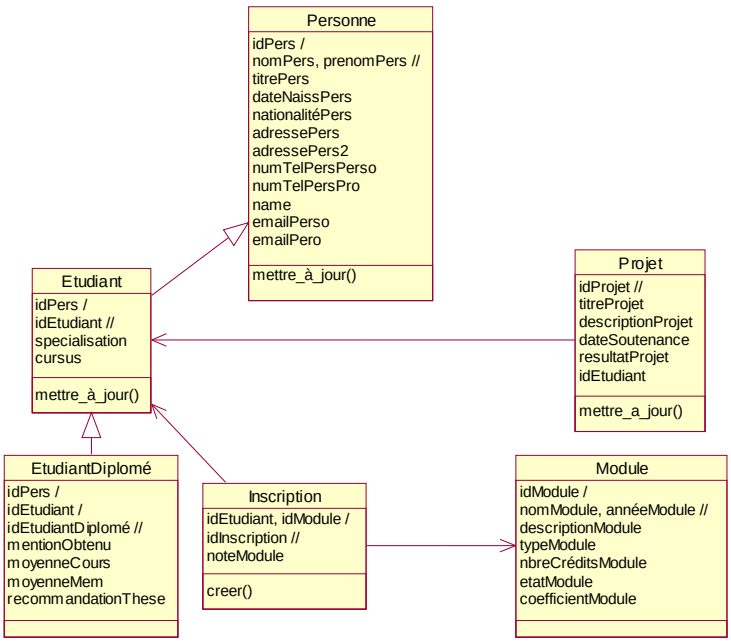

Figure 22 : $D_{hcl\_résultats\_étudiant}$, le diagramme de *hcl_résultats_étudiant*

La Figure 23 représente $G_{hcl\_résultats\_étudiant}$, le graphe de navigation de *hcl_résultats_étudiant*. Les objets des classes de *hcl_résultats_étudiant* sont atteints en partant d'objets d'*Étudiant* : *Étudiant* est la *classe racine* de l'hyperclasse.

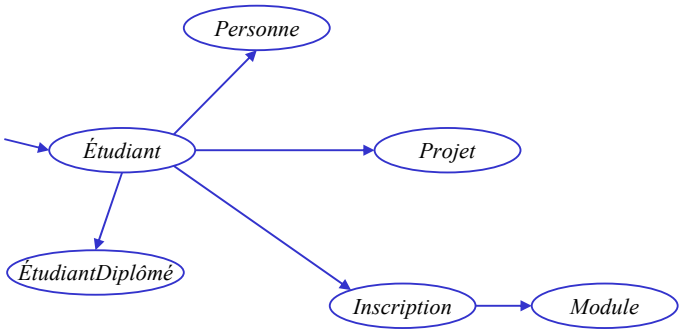

Figure 23 : $G_{hcl\_résultats\_étudiant}$, le graphe de navigation de *hcl_résultats_étudiant*

Un hyperobjet de *hcl_résultats_étudiant* est composé d'un objet de la classe racine et de l'ensemble des objets des classes de l'hyperclasse atteints par navigation sur la base de $G_{hcl\_résultats\_étudiant}$. La Figure 24 montre un exemple d'hyperobjet de *hcl_résultats_étudiant*.

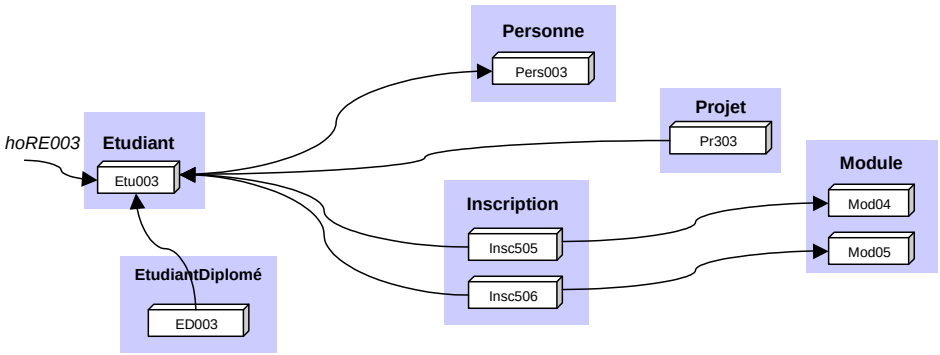

Figure 24 : Exemple d'hyperobjet de *hcl_résultats_étudiant*

Une hyperméthode, associée à *hcl_résultats_étudiant*, permet de calculer pour chacun de ses hyperobjets la moyenne de l'étudiant à partir de ses notes aux modules auxquels il est inscrit et de la note de son projet de recherche. Cette hyperméthode utilise les objets d'*Inscription* et de *Projet* appartenant à l'hyperobjet concerné.

Dans ce chapitre, nous introduisons les concepts d'hyperclasse, de diagramme d'hyperclasse, de graphe de navigation, d'hyperobjet, d'hyperattribut et d'hyperméthode. Nous montrons ensuite l'intérêt de ces concepts dans l'ingénierie des SI.

1. DIAGRAMME D'HYPERCLASSE

Une hyperclasse est une entité définie sur un sous-diagramme du diagramme de classes du SI. Cet ensemble est appelé *diagramme de l'hyperclasse* ; il décrit les classes et les liens entre les classes de l'hyperclasse. Il est connexe et complet.

Le graphe de navigation est une structure complémentaire au digramme de l'hyperclasse qui définit comment s'effectue la navigation entre les objets de ses classes.

1.1. DIAGRAMME D'HYPERCLASSE

Soit *hcl* une hyperclasse définie sur le diagramme de classes $D(C, \Lambda, \Gamma)$.

Le *diagramme de l'hyperclasse hcl* est un sous-diagramme de *D*, défini par $D_{hcl}(C_{hcl}, \Lambda_{hcl}, \Gamma_{hcl})$, où :

- C_{hcl} désigne l'ensemble des classes de l'hyperclasse, $C_{hcl} = \{cl_i ; cl_i \in hcl\}$;
- C_{hcl} est un sous-ensemble des classes de D : $C_{hcl} \subseteq C$;
- $\Lambda_{hcl} \subseteq \Lambda$ et $\forall (cl_1, cl_2) \in C_{hcl} \times C_{hcl}$, $(cl_1, cl_2) \in \Lambda_{hcl}$ si et seulement si $(cl_1, cl_2) \in \Lambda$;
- $\Gamma_{hcl} \subseteq \Gamma$ et $\forall (cl_1, cl_2) \in C_{hcl} \times C_{hcl}$, $(cl_1, cl_2) \in \Gamma_{hcl}$ si et seulement si $(cl_1, cl_2) \in \Gamma$.

Si $(cl_1, cl_2) \in \Lambda_{hcl}$ ou si $(cl_2, cl_1) \in \Lambda_{hcl}$, les classes cl_1 et cl_2 sont dites *adjacentes* dans D_{hcl}.

Propriétés du diagramme d'hyperclasse

Le diagramme $D_{hcl}(C_{hcl}, \Theta_{hcl})$ de *hcl* est un sous-diagramme de *D* (i) *connexe*, (ii) *complet*.

1. Connexité

Le diagramme D_{hcl} de *hcl* est *connexe* si pour toute paire (cl_1, cl_2) de classes de C_{hcl}, il existe une séquence $\{cl_1, cl_{11} \ldots cl_{1i} \ldots cl_n, cl_2\}$ de classes adjacentes.

La propriété de connexité garantit que toute classe de *hcl* est atteignable à partir d'au moins une autre classe de *hcl* et qu'aucune classe n'est donc isolée dans D_{hcl}.

2. Complétude

Le diagramme $D_{hcl}(C_{hcl}, \Lambda_{hcl}, \Gamma_{hcl})$ de *hcl* est *complet* dans $D(C, \Lambda, \Gamma)$ si pour toute classe cl_i de C_{hcl} qui dépend existentiellement d'une classe cl_j, cl_j appartient à C_{hcl}.

$$\forall cl_i \in C_{hcl},\ si\ \exists cl_j \in C\ tel\ que\ (cl_i, cl_j) \in \Lambda_{hcl}\ alors\ cl_j \in C_{hcl}$$

La propriété de complétude garantit qu'une hyperclasse contient tous les éléments qui participent à sa définition.

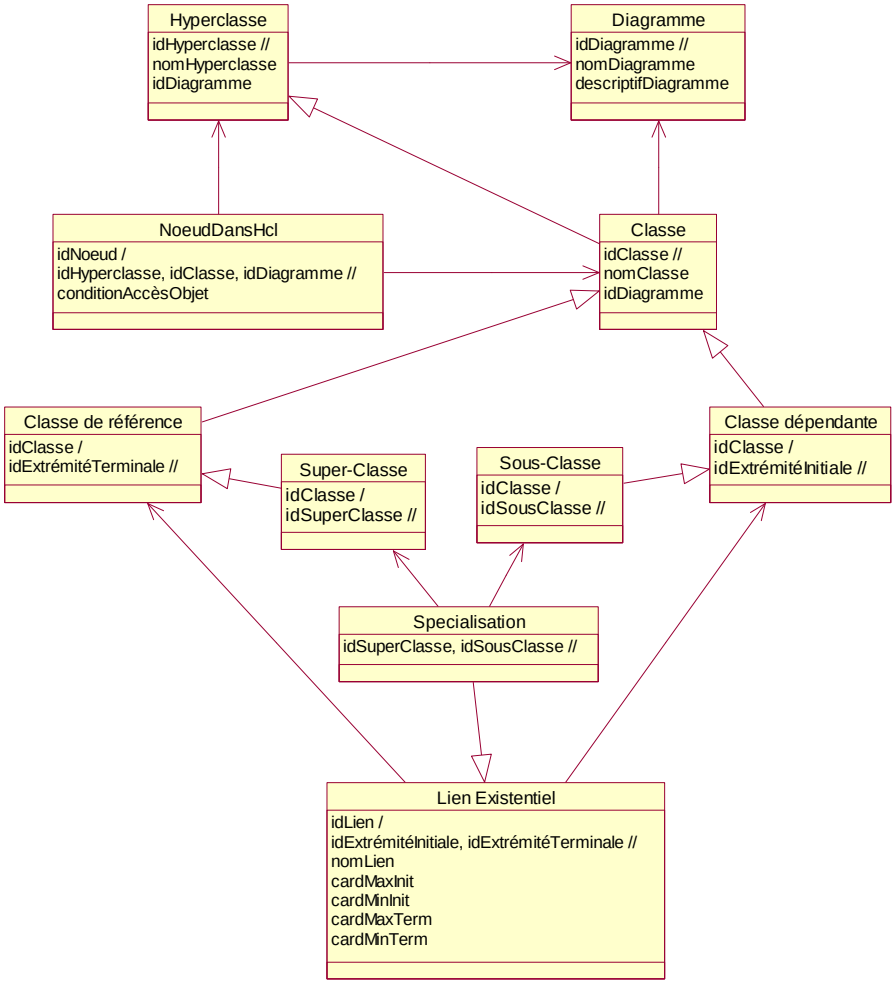

Figure 25 : Modèle d'hyperclasse relatif au *diagramme d'hyperclasse*

Les objets de classe *NœudDansHcl* du méta-modèle d'hyperclasse signale l'appartenance d'une classe du diagramme de classes à l'hyperclasse.

1.2. NAVIGATION DANS L'HYPERCLASSE

Le diagramme d'une hyperclasse *hcl* indique les classes et les liens entre classes qui font partie de *hcl*. Le graphe de navigation de *hcl* est une structure qui décrit comment s'effectue la navigation entre les objets de ses différentes classes. Il est construit à partir de son diagramme.

Chaque classe de *hcl* dispose d'un ou de plusieurs chemins d'accès. Un chemin d'accès est un sous-graphe du graphe de navigation. Le ou les chemins d'accès d'une classe dans *hcl* indiquent comment sont atteints ses objets à partir des objets de la classe racine.

Le graphe de navigation et les chemins d'accès sont complémentaires dans la définition de la navigation entre les objets des classes de *hcl*. Le graphe de navigation est l'union des

différents chemins d'accès de l'hyperclasse. Il garantit la cohérence des différents chemins d'accès.

1.2.1. Classe racine

La *classe racine* est une classe choisie parmi les classes de l'hyperclasse. Ses objets, dits *objets racines de l'hyperclasse*, sont le point de départ de la construction des hyperobjets. Ce sont aussi les objets racines qui identifient les hyperobjets de l'hyperclasse.

Le choix de la classe racine a une incidence directe sur la structure et la sémantique des hyperobjets de l'hyperclasse.

1.2.2. Graphe de navigation

Soient $D_{hcl}(C_{hcl}, \Lambda_{hcl}, \Gamma_{hcl})$ le diagramme de l'hyperclasse *hcl* et cl_{cr} sa classe racine.

G_{hcl}, le *graphe de navigation* de *hcl*, est un *graphe orienté* construit sur les ensembles N_{hcl} de nœuds et Φ_{hcl} d'arcs de G_{hcl}.

$$G_{hcl}=(N_{hcl}, \Phi_{hcl})$$

L'ensemble N_{hcl} des nœuds de G_{hcl} est défini à partir de la bijection $b : C_{hcl} \times N_{hcl} : cl_i \rightarrow n_i$, qui à chaque classe cl_i de C_{hcl} fait correspondre un nœud n_i de N_{hcl}.

Φ_{hcl} constitue l'ensemble des arcs de G_{hcl} et est un sous-ensemble de $N_{hcl} \times N_{hcl}$. $(n_i,n_j) \in \Phi_{hcl}$ s'il existe un arc du nœud n_i vers le nœud n_j dans G_{hcl}.

1.2.2.1. Propriétés

1. Si $(n_i, n_j) \in \Phi_{hcl}$ alors $(cl_1,cl_2) \in \Lambda_{hcl}$ ou $(cl_2,cl_1) \in \Lambda_{hcl}$ où $cl_i = b^{-1}(n_i)$ et $cl_j = b^{-1}(n_j)$.
2. Dans G_{hcl}, Φ_{hcl} est un écoulement du nœud racine n_{rc} $(n_{rc} = b(cl_{rc}))$.

   Rappel (Annexe B) :

   Dans un graphe $G=(N, U)$, un *écoulement* du nœud n_1 est un ensemble d'éléments $E=\{u_1, u_2, \ldots, u_n\}$ de U tels que :
   - ces éléments sont des arcs ;
   - et l'un d'entre eux admet n_1 comme extrémité initiale ;
   - et pour chaque extrémité n_i de chacun de ces arcs, il existe un chemin de n_1 à n_i, formé d'éléments de E.
3. G_{hcl} ne contient pas de boucles :

$$\forall n_i \in N_{hcl} ; (n_i, n_i) \notin \Phi_{hcl}$$

4. Aucun arc de G_{hcl} ne pointe vers le nœud racine n_r.

$$\forall n_i \in N_{hcl} ; \Phi(n_i, n_r) \notin \Phi_{hcl}$$

5. Il y a au plus un arc entre deux nœuds de G_{hcl} :

$$\forall n_i, n_j \in N_{hcl} ; (n_i, n_j) \in \Phi_{hcl} \Rightarrow (n_j, n_i) \notin \Phi_{hcl}$$

Autrement dit, un lien entre deux classes ne peut être parcouru qu'au plus une fois, dans un sens ou un autre.

1.2.2.2. Nœuds adjacents

Dans G_{hcl}, deux nœuds n_i et n_j sont dits adjacents si $(n_i, n_j) \in \Phi_{hcl}$ ou $(n_j, n_i) \in \Phi_{hcl}$.

1.2.3. Chemin d'accès

Un chemin d'accès de la classe cl_i de l'hyperclasse *hcl* indique un chemin possible pour atteindre les objets de cl_i à partir d'un objet (racine) de cl_{rc}. cl_i peut avoir un ou plusieurs chemins d'accès dans *hcl*. Le $x^{ème}$ chemin d'accès $G^x_{i, hcl}$ de la classe cl_i dans *hcl* est un graphe construit sur un ensemble $N^x_{i, hcl}$ de nœuds et un ensemble d'arcs définis par la relation $\Phi^x_{i, hcl}$.

$$G^x_{i, hcl} = (N^x_{i, hcl}, \Phi^x_{i, hcl})$$

Où $N^x_{i, hcl}$ est un sous-ensemble des nœuds de G_{hcl} : $N^x_{i, hcl} \subseteq N_i$. $G^x_{i, hcl}$ indique les nœuds et les arcs à parcourir à partir du nœud racine n_r (tel que $n_{rc} = b(cl_{rc})$) pour atteindre le nœud n_i (tel que $n_i = b(cl_i)$).

1.2.3.1. Propriétés

1. $G^x_{i, hcl}$ est un sous graphe de G_{hcl} :
   1.a. $N^x_{i, hcl} \subseteq N_i$;
   1.b. $\Phi^x_{i, hcl} \subseteq \Phi_i$;
   1.c. $\forall n_i, n_j \in N^x_{i, hcl} ; (n_i, n_j) \in \Phi^x_{i, hcl} \Rightarrow (n_j, n_i) \in \Phi_{hcl}$;
2. Dans $G^x_{i, hcl}$, $\Phi^x_{i, hcl}$ est un écoulement du nœud racine n_{rc} ;
3. Dans $G^x_{i, hcl}$, n_i est le seul nœud puits[75] ;
4. Le nœud racine n_r est l'extrémité initiale (nœud source[76]) de tout chemin d'accès : $\forall i, x, n_r \in N^x_{i, hcl}$;
5. Dans $G^x_{i, hcl}$, tous les nœuds de $N^x_{i, hcl}$, excepté le nœud racine, sont atteints au moins une fois : $\forall n_j \in N^x_{i, hcl} - \{n_r\}$; $\exists n_i \in N^x_{i, hcl}$ tel que $(n_i, n_j) \in \Phi^x_{i, hcl}$.

1.2.3.2. Condition de connexité de l'hyperclasse

Il existe au moins un chemin d'accès $G^x_{i, hcl}$ valide pour toute classe cl_i de *hcl*.

---

[75] Dans un graphe orienté, un nœud est dit *puits* s'il n'est l'extrémité initiale d'aucun arc.

[76] Dans un graphe orienté, un nœud est dit *source* s'il n'est l'extrémité terminale d'aucun arc.

1.2.3.3. Cohérence des différents chemins d'accès

Les chemins d'accès des différentes classes d'une hyperclasse doivent rester cohérents entre eux. Le graphe de navigation sert à les coordonner et à donner une vue d'ensemble de l'écoulement dans l'hyperclasse.

1.2.3.4. Propriété

Le graphe de navigation G_{hcl} de l'hyperclasse *hcl* est l'union des différents chemins d'accès de toutes les classes de *hcl* :

$$N_{hcl} = \cup_{i,x} N^{x}_{i,\,hcl} \text{ et } \Phi_{hcl} = \cup_{i,x} \Phi^{x}_{i,\,hcl}$$

La Figure 26 représente le modèle de diagramme d'hyperclasse, de graphe de navigation et des chemins d'accès.

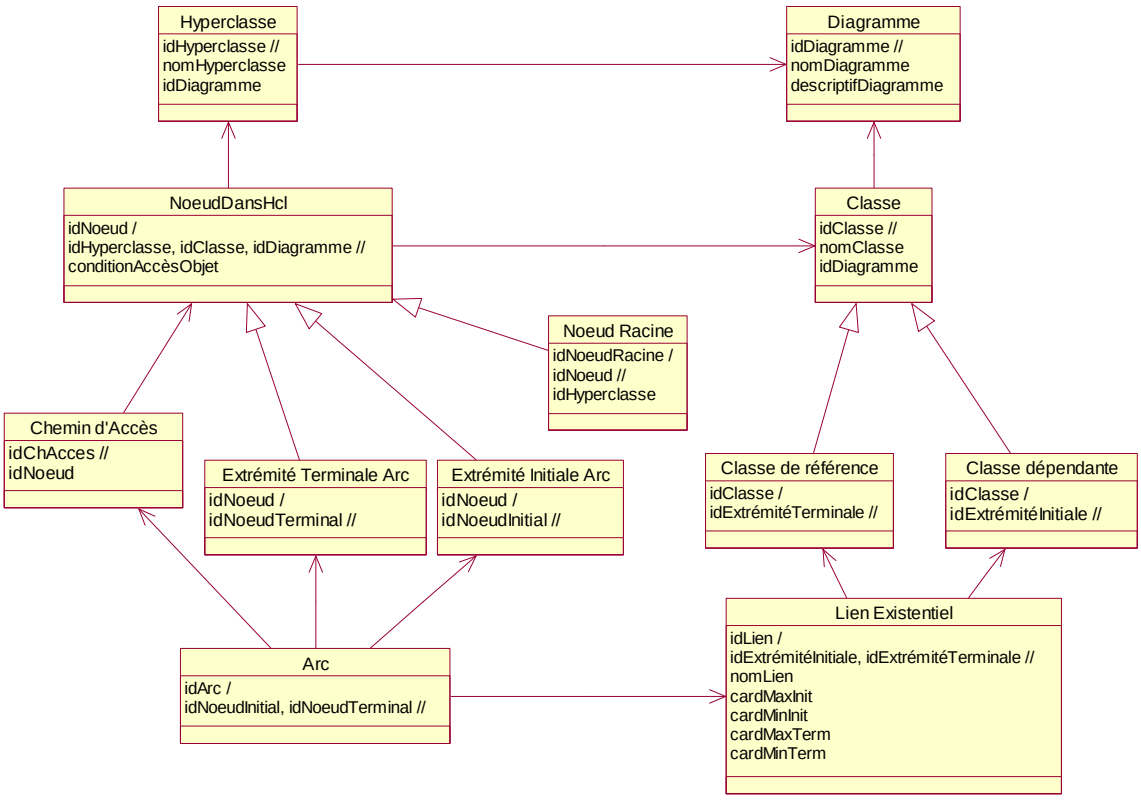

Figure 26 : Modèle d'hyperclasse relatif au diagramme d'hyperclasse, au graphe de navigation, à la classe racine et aux chemins d'accès

1.2.4. Nœud Prédécesseur et Successeur dans une hyperclasse

La notion d'écoulement au sein d'une hyperclasse *hcl* permet de définir deux relations d'ordre partiel prédécesseur et successeur (voir Annexe B) pour décrire la position relative de deux classes par rapport au sens de l'écoulement dans *hcl*. Ces relations sont utilisées dans la

consolidation de diagrammes de classes (1.2.2. Consolidation de diagramme de classes dans le cas d'une suppression de classe, P. 130).

1.2.4.1. Nœud directement prédécesseur et successeur dans une hyperclasse

Le nœud n_i est dit *directement prédécesseur* du nœud n_j dans G_{hcl} si et seulement s'il existe un arc de n_i vers n_j, en d'autres termes, si $(n_i, n_j) \in \Phi_{hcl}$.

Le nœud n_j est dit *directement successeur* du nœud n_i dans G_{hcl} si et seulement s'il existe un arc de n_i vers n_j, en d'autres termes, si $(n_i, n_j) \in \Phi_{hcl}$.

1.2.4.2. Nœuds prédécesseur et successeur dans une hyperclasse

Le nœud n_j est dit *successeur* du nœud n_i dans G_{hcl} si et seulement s'il existe un chemin partant de n_i vers n_j. Le nœud n_i est alors dit prédécesseur du nœud n_i dans G.

$successeur_{hcl}(n_i)$ et $prédécesseur_{hcl}(n_i)$ désignent respectivement les ensembles des nœuds successeurs et prédécesseurs de n_i dans *hcl*.

1.2.4.3. Remarques

La source d'un graphe de navigation est son nœud racine. Une *source* est un nœud qui n'est atteint par aucun arc et qui n'a aucun prédécesseur dans le graphe.

Un *puits* dans un graphe de navigation est un nœud dont aucun arc n'est issu (sans successeur) : $n_i \in N_{i, hcl}$ est un nœud puits si et seulement si $\forall n_j \in N_{i, hcl}, (n_i, n_j) \notin \Phi_{hcl}$.

1.2.5. Exemples de graphe de navigation

Exemple #1

Considérons l'hyperclasse *hcl_Résultats_Étudiant* (Figure 27) définie sur les classes *Étudiant*, *Personne*, *Projet*, *Inscription*, *Module* et *ÉtudiantDiplomé*. Cette hyperclasse est connexe et complète dans le diagramme de classes de *M@TIS*. Elle représente l'espace informationnel nécessaire au coordinateur de la formation pour consulter et gérer les résultats des étudiants.

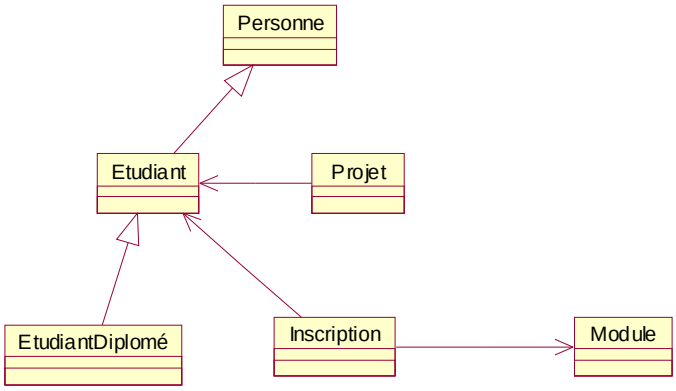

Figure 27 : Diagramme de l'hyperclasse *hcl_Résultats_Étudiant*

Étant donne que l'évaluation des résultats des étudiants est individuelle, la classe *Étudiant* est définie comme classe racine de *hcl_Résultats_Étudiant*. Ses objets servent à identifier les hyperobjets de *hcl_Résultats_Étudiant*.

La Figure 28 représente $G_{hcl}=(N_{hcl}, \Phi_{hcl})$ le graphe de navigation de *hcl_Résultats_Étudiant.*

N_{hcl} ={*Étudiant*, *Personne*, *Projet*, *Inscription*, *Module*, *ÉtudiantDiplômé*} est l'ensemble des nœuds de G_{hcl}.

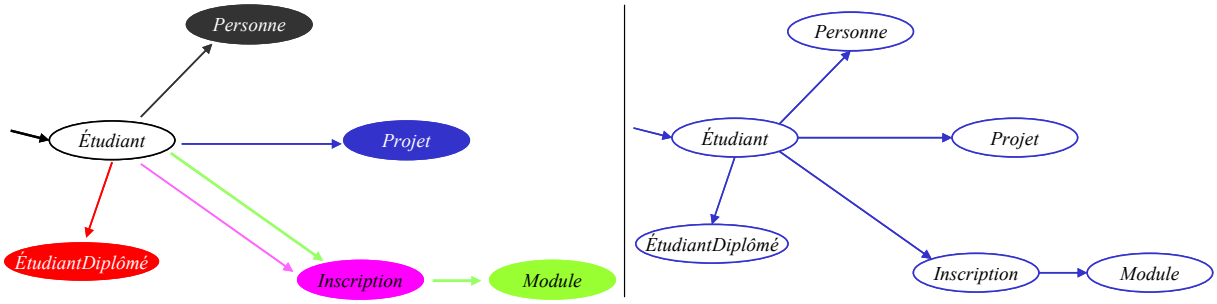

Figure 28 : Chemin d'accès dans *hcl_Résultats_Étudiant* et $G_{hcl\_Résultats\_Étudiant}$

Chaque nœud de N_{hcl} correspond à une et une seule classe de *hcl*, et réciproquement.

Chaque classe de *hcl_Résultats_Étudiant* a au moins un chemin d'accès (excepté la classe racine). Les arcs d'un chemin d'accès sont représentés par des flèches de la même couleur que le nœud concerné.

Le chemin d'accès à *Personne* est constitué d'un seul arc : (*Étudiant*, *Personne*) ; celui de *Module* de deux arcs : {*(Étudiant*, *Inscription*), (*Inscription, Module*)}, etc.

G_{hcl} est l'union des différents chemins d'accès aux classes de *hcl_Résultats_Étudiant*.

Les arcs *(Étudiant*, *Inscription*), (*Étudiant*, *ÉtudiantDiplômé*) et (*Étudiant*, *Projet*) ont un sens opposé au sens des liens sur lesquels ils sont construits.

Exemple #2

Considérons l'hyperclasse *hcl_intervenant_module* (Figure 29) construite sur les classes *Module*, *Enseignement*, *Document*, *ResponsableModule*, *Enseignant*, *Personne*.

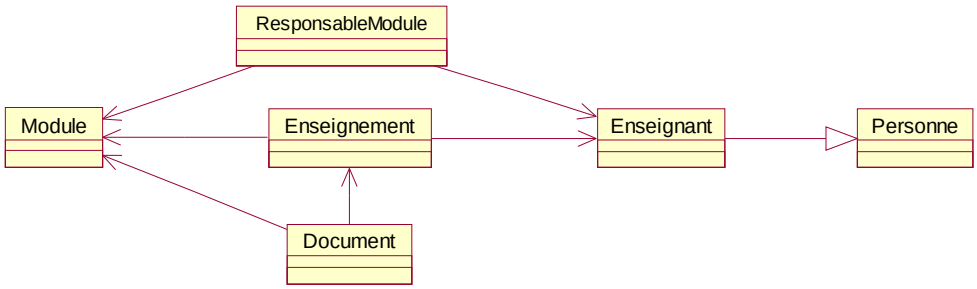

Figure 29 : diagramme de l'hyperclasse *hcl_intervenant_module*

La Figure 30 représente le graphe de navigation de *hcl_intervenant_module.*

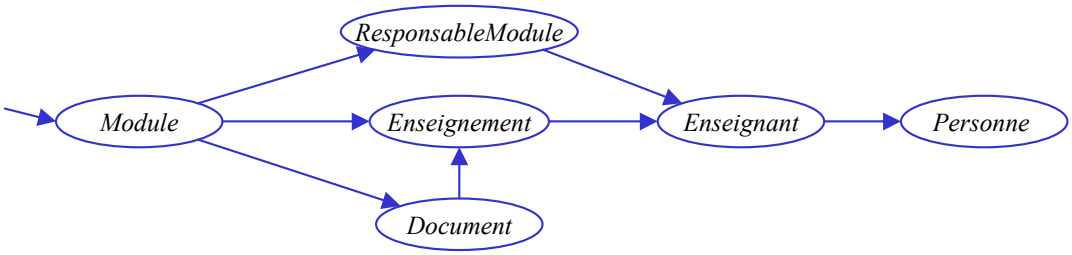

Figure 30 : Graphe de navigation de *hcl_intervenant_module*

Trois chemins d'accès différents sont définis sur le nœud *Enseignant* (Figure 31). Chacun d'eux a une sémantique particulière : (i) le premier chemin indique les enseignants participant aux enseignements du module, (ii) le deuxième indique l'enseignant responsable du module et (iii) le troisième les enseignants qui ont mis en ligne des documents dans le cadre du module.

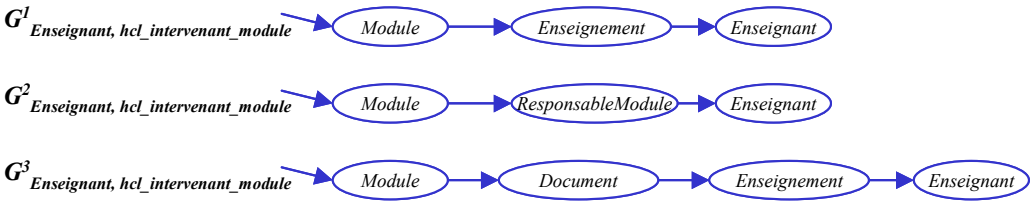

Figure 31 : Les 3 chemins d'accès à *Enseignant* dans *hcl_intervenant_module*

2. HYPEROBJET

De la même manière qu'une classe possède des objets, une hyperclasse possède des *hyperobjets*. Un hyperobjet est une *instance* ou un *objet* d'une hyperclasse.

Un hyperobjet est une construction ayant la structure d'un graphe où les nœuds sont des objets des classes de *hcl* et les arcs les liens qui existent entre ces objets. La construction d'un

hyperobjet est une opération récursive qui se base sur la notion de fermeture de son objet racine suivant le graphe de navigation de l'hyperclasse.

Un hyperobjet $ho_{i,\ hcl}$ de *hcl* est un ensemble d'objets liés, composé d'un objet $o_{i,\ r}$ de son nœud racine n_r (correspondant à sa classe racine cl_r) et des objets des classes de hcl atteints par navigation en se basant sur G_{hcl}. $o_{i,\ r}$ est dit *objet racine* de $ho_{i,\ hcl}$.

$$ho_{i,\ hcl} = \{ (o_{i,\ r})^*_{hcl}, lho_{i,\ hcl} \}$$

$(o_{i,\ r})^*_{hcl}$ désigne la fermeture de $o_{i,\ r}$, *objet racine* de $ho_{i,\ hcl}$, dans l'hyperclasse *hcl*.

$lho_{i,\ hcl}$ désigne l'ensemble des liens entre les objets de $ho_{i,\ hcl}$ qui sont pris en compte dans $ho_{i,\ hcl}$. C'est la restriction de la fonction *lien* L définie sur *OxO* au sous-ensemble $(o_{i,r})^*_{hcl} \times (o_{i,r})^*_{hcl}$.

2.1. RAPPEL : FERMETURE D'UN OBJET[77]

2.1.1. Objets directement reliés

Dans un état donné du SI, un objet o_1 de la classe cl_1 est *directement relié* à un objet o_2 de la classe cl_2 :

- si les classes cl_1 et cl_2 sont adjacentes, et ;
- si connaissant les valeurs prises par o_1, il est possible de retrouver l'objet o_2.

Les objets o_1 et o_2 prennent les mêmes valeurs pour les attributs communs à cl_1 et à cl_2 .

2.1.2. Fonction Lien

La fonction *lien L* est définie sur *OxO* telle que $(o_1, o_2) \in L$ si o_1 est directement relié à o_2.

2.1.3. Objets reliés

Un objet o_1 est relié à un objet o_2, s'il existe une séquence d'objets $\{o_1, o_{11} \ldots o_{1i} \ldots o_n, o_2\}$ telle que :

- o_1 et o_{11} sont directement reliés ;
- o_n et o_2 sont directement reliés ;
- pour chaque paire d'objets o_{1i} et o_{1i+1} consécutifs de la séquence, o_{1i} et o_{1i+1} sont directement reliés.

2.1.4. Fermeture d'un objet

La fermeture $o*$ d'un objet o désigne l'ensemble des objets reliés à o.

o^* est formée de (i) l'objet o, (ii) l'ensemble des objets directement reliés à o et (iii) la fermeture de chacun des objets appartenant à o^*.

---

[77] Chapitre I, 1.3.2. Au niveau des objets, P.37

2.2. CONSTRUCTION D'UN HYPEROBJET

Un hyperobjet $ho_{i,\,hcl}$ de *hcl* est construit à partir de la fermeture de son objet racine $o_{i,\,r}$ dans *hcl*. Trois cas sont à considérer selon que l'on calcule la fermeture d'un objet racine, d'un objet d'un nœud-puits ou la fermeture d'un objet d'un nœud non-puits.

2.2.1. Fermeture d'un objet racine

Deux cas sont à considérer dans le calcul de la fermeture d'un objet racine selon que l'hyperclasse est définie sur (i) une seule classe ou sur (ii) plusieurs classes.

2.2.1.1. Hyperclasse composée d'une seule classe

Si l'hyperclasse *hcl* ne contient qu'une classe (uniquement une classe racine), la fermeture d'un objet se limite à l'objet racine lui-même :

$$ho_{i,\,hcl} = \{\{o_{i,\,r}\}, \varnothing\}$$

2.2.1.2. Hyperclasse définie sur plus d'une classe

Si l'hyperclasse *hcl* est définie sur plus qu'une classe, les objets directement reliés à $o_{i,r}$ dans *hcl* sont des objets des nœuds *directement successeurs* de n_r (nœud racine de *hcl*) dans G_{hcl}.

La fermeture de $o_{i,\,r}$ est formée de :

1. l'objet $o_{i,\,r}$ et
2. des fermetures dans *hcl* de chacun des objets directement reliés à $o_{i,r}$ dans *hcl* .

$$(o_{i,\,r})^*_{hcl} = \{o_{i,\,r}\} \cup \{\cup_{j,\,y}\{o_{j,\,y})^*_{hcl}\} \text{ tel que } T(o_{j,\,y}) \in \text{successeur}_{hcl}(T(o_{i,\,r})) \text{ et } [(o_{i,\,r}, o_{j,\,y}) \in L \text{ ou } (o_{j,\,y}, o_{i,\,r}) \in L]$$

$$\text{Si } (o_{i,\,r}, o_{j,\,y}) \in L \text{ alors } (o_{i,\,r}, o_{j,\,y}) \in lho_{i,hcl} \text{ sinon } [(o_{j,\,y}, o_{i,\,r}) \in L \text{ et } (o_{j,\,y}, o_{i,\,r}) \in lho_{i,\,hcl}]$$

L et *T* désignent respectivement les fonctions *Lien* et *Type*.

2.2.1.3. Rappel : Fonction Type

O désigne l'ensemble des objets.

La fonction *type T:* $O \rightarrow C$ est partout définie ; elle associe un objet à sa classe.

2.2.2. Fermeture d'un objet d'un nœud-puits

Un nœud-puits dans une hyperclasse *hcl* est un nœud qui n'a aucun successeur dans le graphe de navigation de *hcl*.

La fermeture d'un objet d'un nœud-puits est égale à l'objet lui-même.

Soit $o_{i,\,p}$ un objet du nœud n_p correspondant à la classe $type(o_{i,\,p})$ dans G_{hcl} :

$$\textbf{Si successeur}_{hcl}(n_p)=\varnothing \textbf{ alors } (o_{i,p})^*_{hcl}=\{o_{i,p}\}$$

2.2.3. Fermeture d'un objet d'un nœud non-puits

Dans le graphe de navigation de *hcl*, un nœud non-puits est un nœud qui admet au moins un successeur dans G_{hcl}. Autrement dit, un nœud non-puits est le prédécesseur dans G_{hcl} d'au moins un autre nœud.

Dans *hcl*, la fermeture d'un objet $o_{i,x}$ d'un nœud non-puits n_i , correspondant à la classe $type(o_{i,x})$ dans G_{hcl}, est l'ensemble formé par :

1. l'objet $o_{i,x}$ et
2. les fermetures dans *hcl* de chacun des objets directement reliés à $o_{i,x}$ dans *hcl* . Les objets directement reliés à $o_{i,x}$ dans *hcl* sont des objets des nœuds *directement successeurs* de n_i dans G_{hcl}.

$$\textbf{Si successeur}_{hcl}(n_i)\neq\varnothing \textbf{ alors } (o_{i,x})^*_{hcl}=\{o_{i,x}\}\cup\{\cup_{j,y}\{o_{j,y}\}^*_{hcl}\} \textbf{ tel que}$$
$$T(o_{j,y})\in \textbf{successeur}_{hcl}(T(o_{i,x})) \textbf{ et } [(o_{i,x}, o_{j,y})\in L \textbf{ ou } (o_{j,y}, o_{i,x})\in L]$$

$$\textbf{Si } (o_{i,x}, o_{j,y})\in L \textbf{ alors } (o_{i,x}, o_{j,y})\in lho_{i,hcl} \textbf{ sinon } ((o_{j,y}, o_{i,x})\in L) \textbf{ alors } (o_{j,y}, o_{i,x})\in lho_{i,hcl}.$$

N.B : si *hcl* est définie sur plus qu'une classe, le calcul de la fermeture dans *hcl* d'un des ses objets-racines respecte ce canevas.

2.2.4. Remarques

- Les liens entre les objets d'un hyperobjet conservent leurs orientations et demeurent navigables dans les deux sens.
- Dans un même hyperobjet, un objet d'une classe peut être atteint plus qu'une fois.
- Plusieurs objets d'une même classe peuvent faire partie d'un même hyperobjet ; il n'est pas nécessaire que le même objet soit atteint par tous les chemins d'accès de sa classe pour faire partie de l'hyperobjet.

2.3. IDENTIFICATION DES HYPEROBJETS

A partir d'un objet de la classe racine, il n'est possible de construire au plus un hyperobjet dans une même hyperclasse. Ainsi, les hyperobjets sont identifiés par leurs objets racines.

2.4. EXEMPLES D'HYPEROBJETS

Considérons les objets des classes de l'hyperclasse *hcl_Résultats_Étudiant* (Figure 27). Le graphe de navigation de *hcl_Résultats_Étudiant* est représenté par la Figure 28. *Étudiant* est sa classe racine ; elle admet 3 objets : *étu001*, *étu002* et *étu003*. Il est donc possible de construire 3 hyperobjets de *hcl_Résultats_Étudiant*.

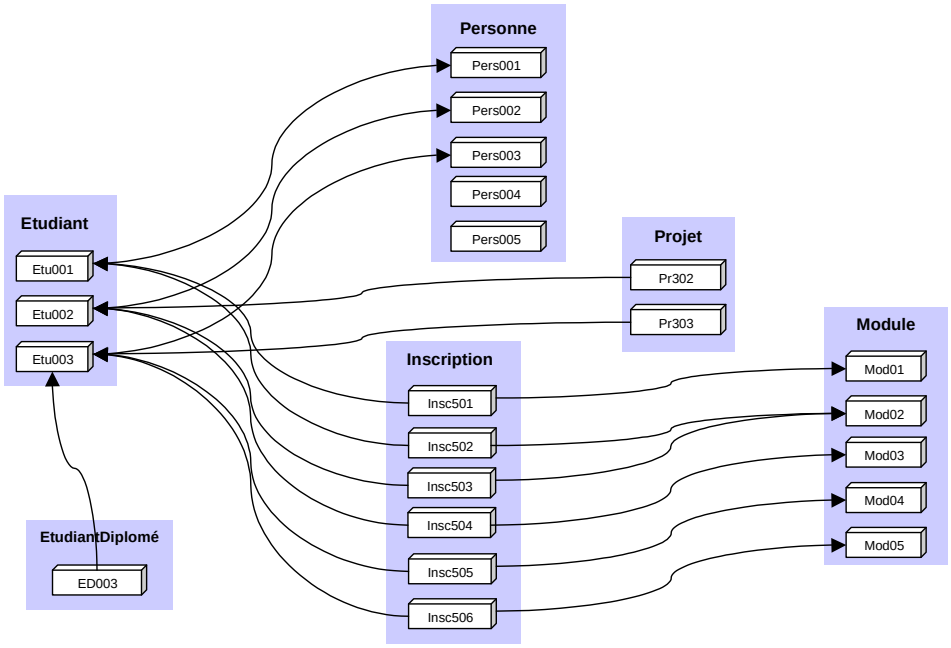

Figure 32 : Objets des classes de *hcl_Résultats_Étudiant*

La Figure 33 représente l'hyperobjet *hoRE001* de *hcl_Résultats_Étudiant* qui a pour objet racine *étu001*.

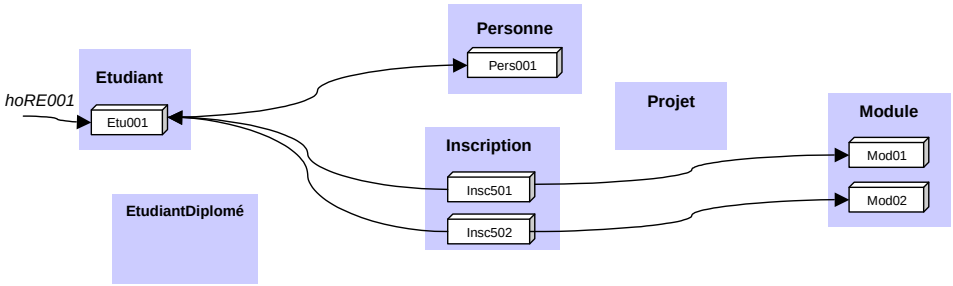

Figure 33 : L'hyperobjet *hoRE001*

Le nœud correspondant à *Étudiant* a trois successeurs dans $G_{hcl\_Résultats\_Étudiant}$, le graphe de navigation de *hcl_Résultats_Étudiant* : (i) *ÉtudiantDiplômé*, (ii) *Inscription* et (iii) *Projet*.

Les objets *pers001* de *Personne,* et, *insc501* et *insc5002* de *Inscription* sont directement reliés à *étu001* alors :

$$(étu001)^*_{hcl\_Résultats\_Étudiant} = \{étu001\} \cup (pers001)^*_{hcl\_Résultats\_Étudiant} \cup (insc501)^*_{hcl\_Résultats\_Étudiant} \cup (insc502)^*_{hcl\_Résultats\_Étudiant}$$

Soit *lhoRE001* la fonction de lien entre les objets de *hoRE001* :

{(étu001, pers001), (insc501, étu001), (insc502,étu001)}∈lhoRE001.

Personne est un nœud-puits dans $G_{hcl\_Résultats\_Étudiant}$ et la fermeture d'un objet d'un nœud-puits est égale à l'objet lui-même : $(pers001)^*_{hcl\_Résultats\_Étudiant} = \{pers001\}$.

Le nœud *Module* est le seul successeur du nœud *Inscription* dans $G_{hcl\_Résultats\_Étudiant}$.

$(insc501)^*_{hcl\_Résultats\_Étudiant} = \{ insc501\} \cup (mod01)^*_{hcl\_Résultats\_Étudiant}$

$(insc502)^*_{hcl\_Résultats\_Étudiant} = \{ insc502\} \cup (mod02)^*_{hcl\_Résultats\_Étudiant}$

{ (insc501, mod01), (insc502,mod01)}∈lhoRE001.

Module est un nœud-puits dans $G_{hcl\_Résultats\_Étudiant}$:

$(mod01)^*_{hcl\_Résultats\_Étudiant} = \{ mod01\}$;

$(mod02)^*_{hcl\_Résultats\_Étudiant} = \{ mod02\}$;

Donc *hoRE001 =({ étu001, pers001, insc501, insc502, mod01, mod02}, {(étu001, pers001), (insc501, étu001), (insc502,étu001), (insc501, mod01), (insc502,mod01)})*

De la même manière, nous construisons les hyperobjets *hoRE002* et *hoRE003*, représentés respectivement par la Figure 34 et la Figure 35.

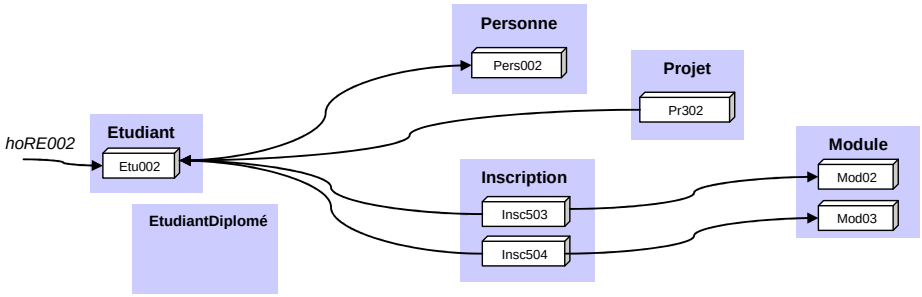

Figure 34 : L'hyperobjet *hoRE002*

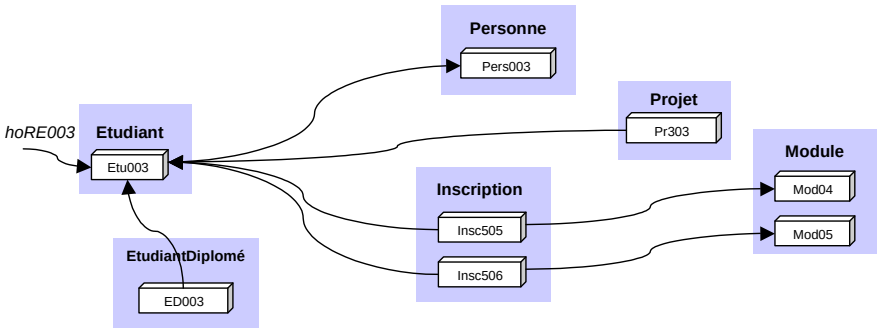

Figure 35 : L'hyperobjet *hoRE003*

3. HYPERATTRIBUT

Le concept d'*hyperattribut* est une généralisation du concept d'attribut pour les classes : un hyperattribut est un attribut d'une hyperclasse.

Un hyperattribut *hatt* de l'hyperclasse *hcl* est une occurrence d'attribut $cl_n.att$ d'une classe cl_n de *hcl* atteint via un des chemins d'accès $G^x_{n,\,hcl}$ de cl_n dans *hcl*.

$$\boldsymbol{hatt = cl_n.att_{/x}}$$

La valeur prise par l'hyperobjet ho_i de *hcl* pour *hatt* ($cl_n.att_{/x}$) est l'ensemble de valeurs prises pour *att* par les objets de *cl* appartenant à ho_i atteints en parcourant le chemin d'accès G^x_{nhcl}.

$$\boldsymbol{ho_i\,[hatt] = \{o_{j,n}[att]\ ;\ cl_n \in hcl\ \&\ cl_n \ni att\ \&\ o_{j,n} \in (ho_i)^*_{/n,\,x}\ \&\ o_{j,n} \in cl_n\}}$$

$(ho_i)^*_{/n,x}$ désigne la fermeture de l'hyperobjet ho_i par rapport aux nœuds du $x^{ème}$ chemin d'accès $G^x_{n,\,hcl}$ de la classe cl_n dans *hcl*.

3.1. Exemple d'hyperattribut #1

Dans l'hyperclasse *hcl_Résultats_Étudiant* (Figure 27), la classe *Module* a un seul chemin d'accès : $G^1_{Module,\ hcl\_Résultats\_Étudiant}$ *=({Étudiant, Inscription, Module},{(Étudiant, Inscription), (Inscription, Module)}).*

L'hyperattribut $Module.nomModule_{/1}$ prend pour valeurs pour l'hyperobjet *hoRE001* (Figure 33) les valeurs prises pour l'attribut *nomModule* par les objets *Mod01* et *Mod02* de *Module*.

3.2. Exemple d'hyperattribut #2

Dans l'hyperclasse *hcl_intervenant_module* (Figure 29), trois chemins d'accès différents sont définis sur le nœud *Enseignant* (Figure 31). Chacun d'eux a une sémantique particulière : (i) le premier chemin indique les enseignants participant aux enseignements du module, (ii) le deuxième indique l'enseignant responsable du module et (iii) le troisième les enseignants qui ont mis en ligne des documents dans le cadre du module.

Dans un hyperobjet de *hcl_intervenant_module*, les valeurs prises par l'hyperattribut $Ensignant.idEnseignant_{/1}$ sont les valeurs prises par les objets de la classe *Enseignant* pour l'attribut *idEnseignant* et atteints à partir d'un objet racine en suivant le chemin d'accès $G^1_{Enseignant,\ hcl\_intervenant\_module}$. Ils indiquent les identifiants des enseignants participant aux enseignements d'un module.

Les valeurs prises par l'hyperattribut $Ensignant.idEnseignant_{/2}$ sont les valeurs prises par les objets de la classe *Enseignant* pour l'attribut *idEnseignant* et atteints à partir d'un objet racine en suivant le chemin d'accès $G^2_{Enseignant,\ hcl\_intervenant\_module}$. Ils indiquent les identifiants des enseignants responsables d'un module.

3.3. Visibilité des attributs au niveau d'une hyperclasse

Au niveau d'une hyperclasse, les attributs de ses classes ne sont perceptibles qu'à travers ses hyperattributs : l'hyperattribut indique le chemin à parcourir à partir d'un objet racine pour atteindre les valeurs prises dans l'hyperobjet pour un attribut. Une propriété de l'hyperattribut indique si l'attribut correspondant est modifiable ou non au niveau de l'hyperclasse. C'est ainsi que c'est au niveau des hyperclasses que se décide la visibilité et le type d'accès aux attributs d'une classe et non au niveau de la classe.

Dans *binex*, chaque classe dispose au moins d'un identifiant, et les mécanismes d'accès aux objets et de navigation entre les objets se basent sur les attributs identifiants et référentiels qui matérialisent au niveau des objets les liens qui existent entrent leurs classes. Pour assurer la navigation entre les classes et les objets des classes d'une hyperclasse, tous les attributs identifiants ou référentiels utilisés dans la navigation sont visibles et accessibles (et non modifiables, car permanents) par défaut, au niveau d'une hyperclasse *hcl* : si un lien est parcouru dans le graphe de navigation de *hcl*, dans un sens ou un autre, les attributs identifiants et référentiels qui le matérialisent sont automatiquement visibles au niveau de *hcl*. Les autres attributs des classes de *hcl* peuvent être visibles ou modifiables ou non au niveau de *hcl.*

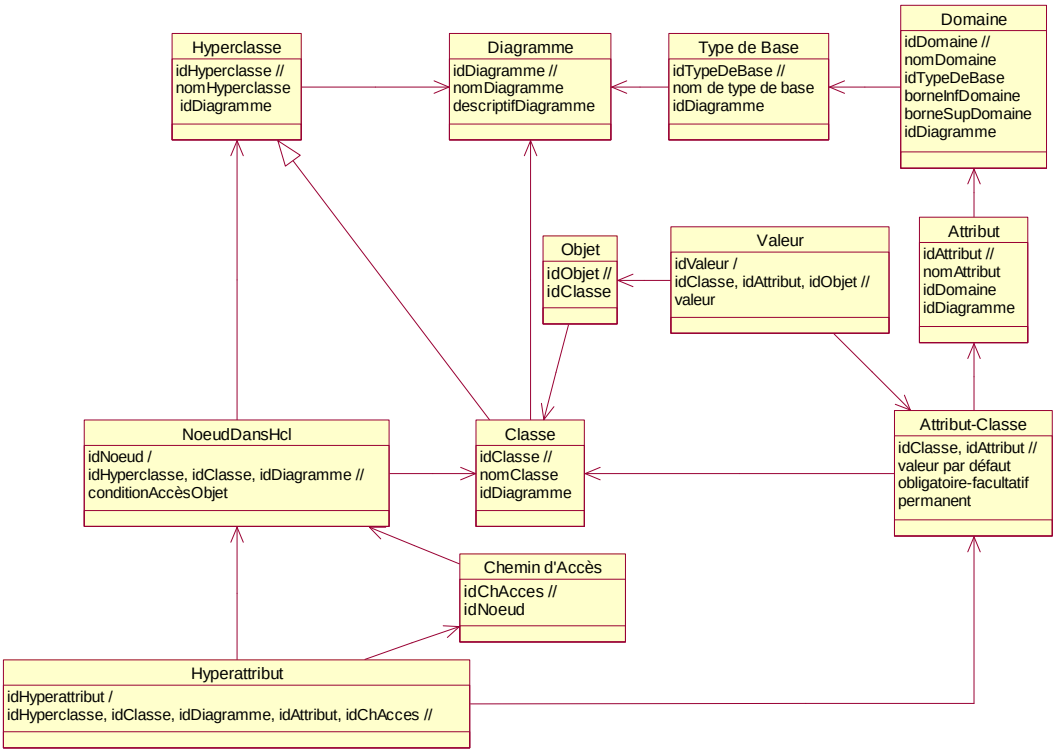

Figure 36 : Modèle d'hyperclasse relatif aux hyperattributs

3.4. Hyperattributs implicites

Un hyperattribut implicite est un hyperattribut pour lequel il n'est pas nécessaire de préciser le chemin d'accès à parcourir pour atteindre son attribut correspondant dans sa classe.

Il existe deux types d'hyperattributs implicites :

3.4.1. hatt = cl_n.att

Le nom de l'hyperattribut *hatt* dans *hcl* est le nom de l'attribut correspondant *att* préfixé du nom de la classe cl_n à laquelle il appartient :

$$hatt = cl_n.att$$

hatt est dans ce cas l'union des hyperattributs définis sur *att* pour chaque chemin d'accès de cl_n dans *hcl*.

$$hatt = \cup_x cl_n.att_{/x}$$

Dans ce cas, l'ensemble des valeurs prises par l'hyperobjet ho_i de *hcl* pour l'hyperattribut *hatt* ($cl_n.att$) est l'ensemble de valeurs prises pour *att* par les objets de *cl* appartenant à ho_i.

$$ho_i\,[hatt] = \{o_{j,n}[att] \;;\; cl_n \in hcl \;\&\; cl_n \ni att \;\&\; o_{j,n} \in ho_i \;\&\; o_{j,n} \in cl_n\}$$

Exemple

Les valeurs prises par l'hyperattribut *Inscription.idModule* de *hcl_Résultats_Étudiant* dans l'hyperobjet *hoRE001* (Figure 33) sont :

```
hoRE001.Inscription.idModule={mod01.idModule, mod02.idModule}
```

3.4.2. hatt = att

Le nom de *hatt* dans *hcl* est le nom de l'attribut *att* correspondant dans sa classe d'origine *cl*.

$$hatt = att$$

Cette option ne précise pas la classe de *hcl* à laquelle *att* appartient. Les valeurs prises par un hyperobjet ho_i pour *hatt* sont l'union des ensembles de valeurs prises pour *att* par les objets des classes cl_n de *hcl* ayant *att* comme attribut et appartenant à ho_i.

$$hatt = \cup_n cl_n.att$$

$$ho_i\,[hatt] = \{o_{j,n}[att] \text{ tel que } cl_n \in hcl \;\&\; cl_n \ni att \;\&\; o_{j,n} \in ho_i\}$$

Exemple

Considérons l'hyperclass *hcl_Résultats_Étudiant* (Figure 27 et Figure 28).

L'attribut *idModule* apparaît dans *hcl_Résultats_Étudiant* dans les classes *Inscription* et *Module*. Les valeurs prises par l'hyperattribut *idModule* de *hcl_Résultats_Étudiant* dans l'hyperobjet *hoRE001* (Figure 33) sont :

```
hoRE001.idModule={inscr501.idModule, inscr502.idModule,
mod01.idModule, mod02.idModule}
```

4. HYPERMÉTHODE

Une *méthode de classe* est une opération associée à une classe, qui permet d'accéder et de modifier ses objets. Invoquer une méthode sur un objet (i) modifie les valeurs prises par certains de ses attributs ou (ii) retourne des valeurs calculées à partir de valeurs prises par les attributs pour l'objet. D'une manière similaire, il est possible de définir des méthodes d'hyperclasse ou *hyperméthodes* sur les hyperclasses : une hyperméthode est une méthode associée à une hyperclasse.

Une hyperméthode *hm*, tout comme une méthode, se présente sous la forme d'un ensemble d'instructions. En plus des variables locales traditionnelles des méthodes, une instruction dans *hm* peut impliquer les hyperattributs de *hcl*, ses hyperobjets, les classes de *hcl*, leurs méthodes, leurs objets, et les autres hyperméthodes de *hcl*.

Les opérations de (i) *création*, (ii) *suppression*, (iii) *mise-à-jour* et de (iv) *consultation* d'hyperobjets sont associées à chaque hyperclasse : ce sont des *hyperméthodes de base*.

4.1. HYPERMÉTHODES DE BASE

Les hyperméthodes de base sont les opérations de (i) *création*, de (ii) *suppression*, de (iii) *mise-à-jour* et de (iv) *consultation* des hyperobjets d'une hyperclasse *hcl*.

Dans ces opérations, il est nécessaire de tenir compte du fait que les classes de *hcl* peuvent aussi appartenir à d'autres hyperclasses ou peuvent être soumises à un ensemble de règles d'intégrité.

Un hyperobjet est calculé à partir des objets des classes de *hcl.* Ces objets doivent respecter les liens existentiels et de spécialisation-généralisation définis dans le diagramme de classes. Grâce à la condition de complétude définie sur les classes des hyperclasses, aucune des classes d'une hyperclasse ne dépend existentiellement d'une classe externe à l'hyperclasse. Cependant, des classes n'appartenant pas à l'hyperclasse peuvent dépendre de classes de l'hyperclasse.

Les hyperméthodes de base sont présentées sous forme de tableaux dont la première ligne indique les paramètres de l'opération, la deuxième les pré-conditions à son application, la troisième les post-actions sur les objets du SI.

4.1.1. Création d'un hyperobjet

La création d'un nouvel hyperobjet est initiée par la création d'un nouvel objet de la classe racine de l'hyperclasse.

Créer un hyperobjet

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ La classe cl_r racine de *hcl* ; ▪ L'ensemble (v_1, v_2, … v_n) des valeurs prises par le nouvel objet racine pour les attributs (att_1, att_2, … att_n) de cl_r.
Pré-cond. :	▪ Même pré-conditions que pour la création d'un nouvel objet dans cl_r.
Post-act. Objets :	▪ Créer un objet dans cl_r.

Les valeurs prises par le nouvel objet racine (du nouvel hyperobjet) déterminent les liens qui existent entre lui et les objets de ses classes adjacentes et par conséquent la structure initiale de l'hyperobjet. L'hyperobjet peut être ensuite étendu par mises à jour successives.

4.1.2. Suppression d'un hyperobjet

La suppression d'un hyperobjet ho_i consiste en la suppression de son objet racine $o_{i,r}$ de la classe racine cl_r.

Si la zone d'influence de la classe racine cl_r est incluse dans l'ensemble des classes de l'hyperclasse, l'objet racine $o_{i,r}$ et tous les objets qui lui sont reliés sont supprimés. En effet, dans *binex*, la suppression d'un objet o_A entraîne la suppression de tous les objets de la fermeture de o_A dans la zone de dépendance de cl_A.

Si l'objet $o_{i,r}$ est référencé par d'autres objets, trois cas peuvent se présenter :

- si $o_{i,r}$ n'est référencé par aucun objet d'aucune autre classe, appartenant ou non à l'hyperclasse, $o_{i,r}$ peut être supprimé ;
- si $o_{i,r}$ ou un des objets qui en dépendent, est référencé par au moins un objet d'une classe qui n'appartient pas à l'hyperclasse, $o_{i,r}$ ne peut être supprimé ;
- si ni $o_{i,r}$ ni aucun des objets qui en dépendent directement ou indirectement, n'est référencé par un objet d'une classe qui n'appartient pas à l'hyperclasse, dans ce cas, $o_{i,r}$ et tous les objets qui en dépendent directement ou indirectement dans l'hyperobjet sont supprimés.

Supprimer un hyperobjet

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ ho_i l'hyperobjet à supprimer ; ▪ La valeur d'un identifiant de $o_{i,r}$ l'objet racine de ho_i.
Pré-cond. :	▪ Ni $o_{i,r}$, ni aucun des objets qui en dépendent directement ou indirectement, n'est référencé par un objet d'une classe qui n'appartient pas à l'hyperclasse.
Post-act. Objets :	▪ Supprimer $o_{i,r}$ et tous les objets qui en dépendent directement ou indirectement dans ho_i.

Exemple de suppression d'hyperobjet

Reprenons l'exemple de l'hyperclasse *hcl_Résultats_Étudiant* (Figure 27). La zone de dépendance de sa classe racine *Étudiant* comprend les classes *Personne*, *Inscription*, *Projet* et *SupervisionProjet* en plus de la classe *Étudiant*.

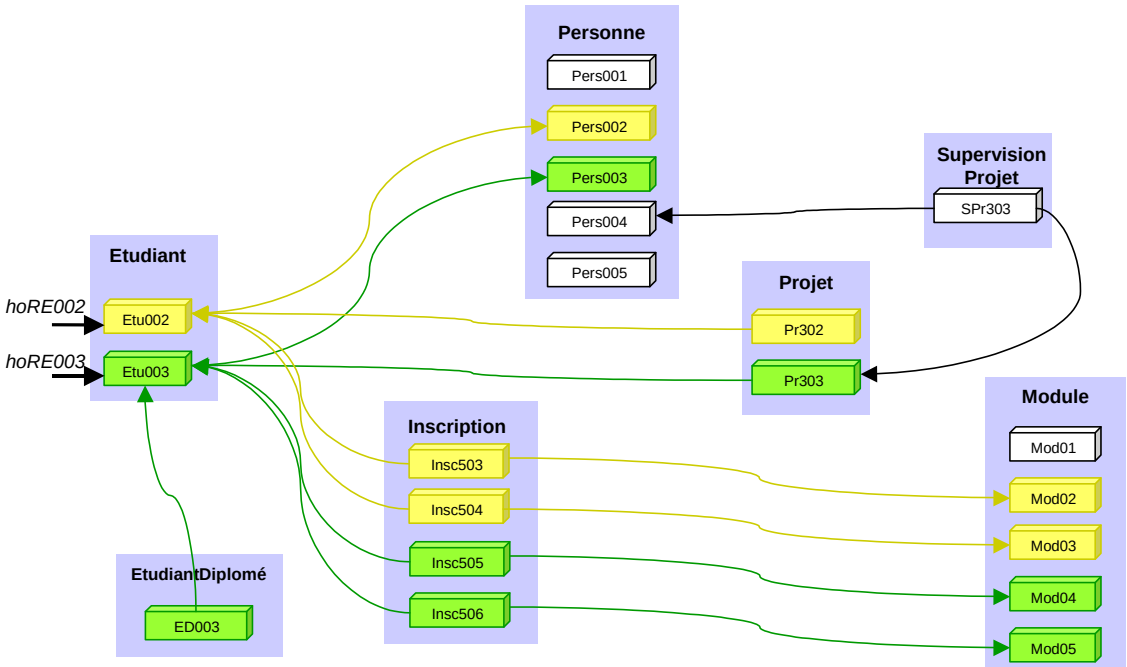

Figure 37 : exemple de suppression d'hyperobjet

La classe *SupervisionProjet* ne fait pas partie de *hcl_Résultats_Étudiant* mais appartient à la zone de dépendance d'*Étudiant.* Dans la Figure 37, la suppression de l'objet *Etu003* d'*Étudiant* entraîne la suppression des objets *Pers003* de *Personne*, *Insc505* et *Insc506* d'*Inscription*, *Pr303* de *Projet* et *SPr303* de *SupervisionProjet*. La suppression l'hyperobjet *hoRE003* est alors impossible.

La suppression de l'hyperobjet *hoRE002* est possible. Il n'existe aucun objet de *SupervisionProjet* dépendant de *Pr302*. Elle entraine la suppression des objets *Pers002* de *Personne*, *Insc503* et *Insc504* d'*Inscription*, *Pr302* de *Projet* et Etu02 d'*Étudiant*.

4.1.3. Mise à jour d'hyperobjet

La mise-à-jour d'un hyperobjet, au même titre que la mise-à-jour d'un objet, concerne la mise-à-jour de la valeur prise par un ou par plusieurs de ses hyperattributs.

Dans le modèle *binex*, seule la mise à jour des valeurs des attributs simples est autorisée. Si l'attribut correspondant à l'hyperattribut à mettre à jour est un attribut identifiant ou référentiel dans une des classes de l'hyperclasse, la mise-à-jour de sa valeur n'est pas autorisée. Si l'attribut correspondant à l'hyperattribut à mettre à jour dans l'hyperobjet ho_i n'est ni attribut identifiant ni attribut référentiel dans aucune des classes de l'hyperclasse, trois cas sont à considérer, suivant le nommage adopté pour l'hyperattribut.

La valeur prise par un objet d'une classe pour un attribut donné est vide ou est un singleton. La valeur prise par un hyperobjet d'une hyperclasse pour un hyperattribut donné est un ensemble qui peut être vide ou composé de un ou plusieurs éléments. La mise à jour de la valeur $ho_i[hatt]$ prise par l'hyperattribut *hatt* en une valeur *v'* ($ho_i[hatt] := \{v'\}$) dans l'hyperobjet ho_i consiste à :

1. si $hatt = cl_n.att_x$, $ho_i\ [hatt] = \{o_{j,n}[att]\ ;\ cl_n \in hcl, cl_n \ni att, o_{j,n} \in (ho_i)^*_{/n,\ x}$ et $o_{j,n} \in cl_n\}$ affecter la valeur *v'* à l'attribut *att* dans tout objet $o_{j,n}$ de la classe cl_n appartenant à la fermeture de ho_i atteint en parcourant le chemin d'accès $G^x_{n,\ hcl}$;
2. si $hatt = cl_n.att$, $ho_i\ [hatt] = \{o_{j,n}[att]\ ;\ cl_n \in hcl, cl_n \ni att, o_{j,n} \in ho_i$ et $o_{j,n} \in cl_n\}$ affecter la valeur *v'* à l'attribut *att* dans tout objet $o_{j,n}$ de la classe cl_n appartenant à la fermeture de ho_i ;
3. si $hatt = att$, $ho_i[hatt] = \{o_{j,n}[att]$; $cl_n \in hcl, cl_n \ni att$ et $o_{j,n} \in ho_i\}$ affecter la valeur *v'* à l'attribut *att* dans tout objet $o_{j,n}$ de la classe cl_n, où $o_{j,n}$ appartient à la fermeture de ho_i et où cl_n a *att* comme attribut.

Il est évident que les valeurs prises par les hyperattributs ne peuvent être mises à jour que dans la limite de ce qu'autorisent les règles d'intégrité définies sur le diagramme de classes.

Mettre à jour un hyperobjet

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ ho_i l'hyperobjet à mettre à jour ; ▪ La valeur d'un identifiant de $o_{i,r}$ l'objet racine de ho_i ; ▪ L'hyperattribut *hatt* à mettre à jour ; ▪ La nouvelle valeur *v'* de *hatt*.
Pré-cond. :	▪ L'attribut *att* correspondant à *hatt* ne fait partie d'aucun identifiant des classes de l'hyperclasse.
Post-act. *Objets :*	▪ si $hatt = cl_n.att_x$, $ho_i\ [hatt] = \{o_{j,n}[att])$ où $cl_n \in hcl$, $cl_n \ni att$, $o_{j,n} \in (ho_i)^*_{/n,\ x}$ et $o_{j,n} \in cl_n\}$ affecter la valeur *v'* à l'attribut att dans tout objet $o_{j,n}$ de la classe cl_n appartenant à la fermeture de ho_i atteint en parcourant le chemin d'accès $G^x_{n,\ hcl}$. ▪ si $hatt = att$, $ho_i\ [hatt] = \{o_{j,n}[att]; cl_n \ni att\ \&\ o_{j,n} \in ho_i^*\}$, affecter la valeur *v'* à l'attribut *att* dans tout objet $o_{j,n}$ de la classe cl_n, où $o_{j,n}$ appartient à la fermeture de ho_i et où cl_n a *att* comme attribut. ▪ si $hatt = cl.att$, $ho_i\ [hatt] = \{\ o_{j,n}[att]; o_{j,n} \in ho_i^*\}$, affecter la valeur *v'* à l'attribut *att* dans tout objet $o_{j,n}$ de la classe cl_n appartenant à la fermeture de ho_i.

4.1.4. Consultation d'hyperobjet

Le but de cette opération est de fournir sous une forme facilement exploitable par les programmes, en particulier les hyperméthodes, tout ou partie d'un hyperobjet.

1. Consulter un hyperobjet

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ ho_i l'hyperobjet à consulter ; ▪ La valeur d'un identifiant de $o_{i,r}$ l'objet racine de ho_i.
Pré-cond. :	▪ Aucune.

Résultat retourné :	▪ Cette opération retourne le résultat de calcul de ho_i sous la forme d'un graphe orienté d'objets.

2. Consulter une partie d'un hyperobjet

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ ho_k l'hyperobjet à consulter ; ▪ Les classes cl_i et cl_j de *hcl* à consulter.
Pré-cond. :	▪ cl_i est un prédécesseur de cl_j dans *hcl*.
Résultat retourné :	▪ L'opération de consultation d'une partie d'un hyperobjet concerne deux classes cl_i et cl_j de *hcl* et un hyperobjet ho_k de *hcl*. ▪ En supposant que cl_i est un prédécesseur de cl_j dans *hcl,* l'opération de consultation recherche pour chaque objet $o_{i,n}$ de cl_i appartenant à ho_k l'ensemble des objets $o_{j,m}$ de cl_j appartenant à ho_k qu'il est possible d'atteindre à partir de $o_{i,n}$ en constituant dans à ho_k une séquence d'objets reliant $o_{i,n}$ à $o_{j,m}$ et en respectant l'écoulement du graphe de navigation de hcl pour passer de cl_i à cl_j. ▪ Le résultat retourné est un ensemble de paires d'objets ($o_{i,n}$, $o_{j,m}$) de cl_i et cl_j, telles que les objets $o_{i,n}$ et $o_{j,m}$ sont reliés dans ho_i. Les objets $o_{i,n}$ et $o_{j,m}$ sont représentés par la valeur de l'un de leurs identifiants.

Exemple de consultation partielle d'hyperobjet

Considérons l'hyperclasses *hcl_Résultats_Etudiant* (Figure 27) et les classes *Inscription* et *Module*. L'opération de consultation de l'hyperobjet *hoRE003* portant sur les objets de ces classes retourne l'ensemble {(*insc505*, *mod04*), (*insc506*, *mod05*)}. *insc505* de *Inscription* est relié à *mod04* de *Module* et *insc506* de *Inscription* est relié à *mod05* de *Module*.

3. Ensemble des objets d'une classe dans un hyperobjet atteints par un chemin d'accès

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ ho_k l'hyperobjet à consulter ; ▪ $G^x_{i,\,hcl}$, le $x^{ème}$ chemin d'accès à la classe cl_i de *hcl*.
Pré-cond. :	▪ cl_i est une classe de *hcl* ; ▪ $G^x_{i,\,hcl}$ est un chemin d'accès valide.
Résultat retourné :	▪ L'ensemble des objets d'une classe dans un hyperobjet atteints via $G^x_{i,\,hcl}$; ▪ Cet ensemble correspond au calcul de la fermeture de l'objet racine de ho_k dans l'ensemble des nœuds de $G^x_{i,\,hcl}$.

4.2. DÉCLARATION ET APPEL D'UNE HYPERMÉTHODE

4.2.1. Déclaration d'une hyperméthode

La déclaration d'une hyperméthode est identique à celle d'une méthode de classe :

```
type_de_donnée Nom_De_L_Hypermethode(type1 argument1, type2
argument2, …) {
 liste d'instructions
}
```

4.2.2. Appel d'une hyperméthode

Pour être exécutée, une hyperméthode est appelée par son nom suivi d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée:

```
Nom_De_L_Hypermethode();
```

ou

```
Nom_De_L_Hypermethode (argument1, argument2, …);
```

4.3. OPÉRANDES ET OPÉRATEURS D'UNE HYPERMÉTHODE

4.3.1. Opérandes

En plus des variables locales traditionnelles des méthodes (variables, objets, méthodes, valeurs prises par les objets pour certains attributs, etc.), une hyperméthode *hm* peut invoquer dans ses instructions :

1. les hyperattributs de *hcl*, implicites ou explicites, en utilisant l'une des trois options de nommage possibles ($cl_n.att_{/x}$, $cl_n.att$, att). Le choix de l'option de nommage a une conséquence directe sur les résultats obtenus ;
2. les hyperobjets de *hcl* et les résultats de leur consultation ;
3. les méthodes de base de manipulation des hyperobjets (consultation, création, suppression, mise-à-jour)
4. les classes de *hcl* et leurs objets ;
5. les méthodes des classes[78] de *hcl* ;
6. les autres hyperméthodes de *hcl.*

4.3.2. Opérateurs

Les opérateurs sont des opérations élémentaires qui permettent d'évaluer et de manipuler des opérandes. Nous distinguons plusieurs types d'opérateurs: les opérateurs de calcul, les opérateurs d'assignation, les opérateurs d'incrémentation, les opérateurs de comparaison et les opérateurs logiques.

Compte tenu de la nature ensembliste des résultats de calcul ou de consultation des hyperobjets, des opérateurs adaptés sont utilisés dans les instructions des hyperméthodes.

[78] De la même manière que pour les attributs et les hyperattributs, une méthode de classe n'est visible au niveau d'une classe que si elle est signalée comme telle.

4.4. CONTEXTE D'UNE HYPERMÉTHODE

Nous appelons *contexte d'une hyperméthode hm* de *hcl* l'ensemble des éléments du diagramme de *hcl* qui sont directement invoqués dans les instructions de *hm* : classes, attributs, méthodes, hyperattributs, autres hyperméthodes et chemins d'accès.

La Figure 38 représente le modèle des méthodes et hyperméthodes.

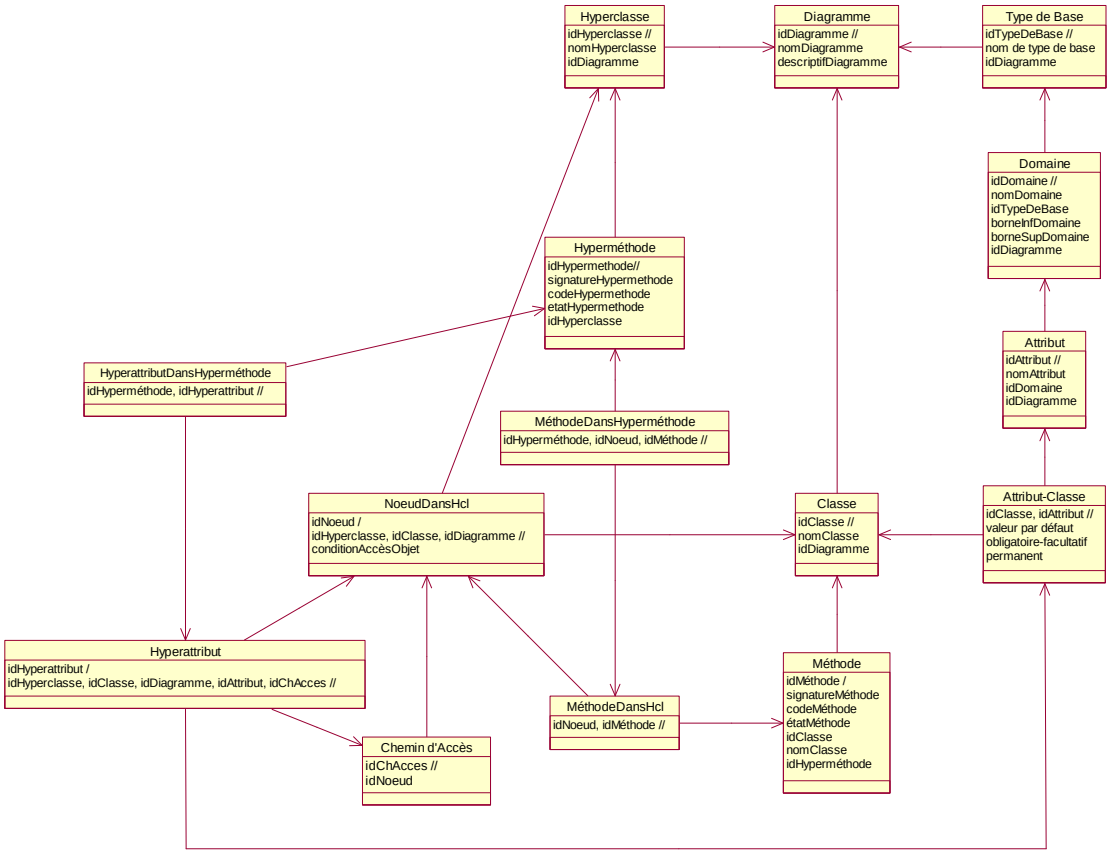

Figure 38 : Modèle d'hyperclasse relatifs aux hyperméthodes

Exemple #1 : Calcul de moyenne d'un étudiant

Considérons l'hyperclasse *hcl_Résultats_Étudiant*. Pour calculer la moyenne de chaque étudiant à la fin de la formation, nous définissons l'hyperméthode *moyenne_etudiant()*. Cette hyperméthode récupère les notes obtenues par un étudiant pour tous les modules où il s'est inscrit et effectue d'abord la moyenne de ses notes pour les épreuves écrites. La moyenne générale de l'année est la moyenne du résultat de l'écrit et du projet de recherche.

```
float moyenne_etudiant(idho)
{
set notes_pondérées= multiplication(idho.consultation(Inscription,
Module), Inscription.noteModule, Module.coefficientModule) ;
float moyenne_ecrit :=moyenne(notes_ pondérées) ;
return ((moyenne_ecrit + idho.notemodule)/2) ;
}
```

L'opération *idho.consultation(Inscription, Module)* retourne toutes les paires d'objets de *Inscription* et de *Module* reliés dans l'hyperobjet *ho* (et identifié par *idho*).

L'opération *multiplication*(*ensemble_de_paires*(Cl_1, Cl_2), $Cl_1.att_1$, $Cl_2.att_2$) est une opération ensembliste, qui retourne sous forme d'ensemble les résultats de la multiplication des valeurs prises pour les attributs att_1 et att_2 par les paires d'objets de cl_1 et cl_2, présents dans l'ensemble de paires. *multiplication*(*idho.consultation(Inscription, Module), Inscription.noteModule, Module.coefficientModule*) retourne les notes pondérées.

L'opération *moyenne(ensemble_valeurs)* retourne la moyenne des valeurs passées en paramètre.

Exemple #2 : Évaluation de l'implication d'un enseignant

Dans cet exemple, nous considérons une hyperclasse fictive que nous appelons *hcl_Implication_Enseignant* et dont le diagramme est représenté par la Figure 39. Cette hyperclasse permettrait d'évaluer l'implication des enseignants de la formation dans les différents modules proposés. Nous avons pour cela imaginé un système de primes qui dépendent de la durée des séances de cours et de leur localisation.

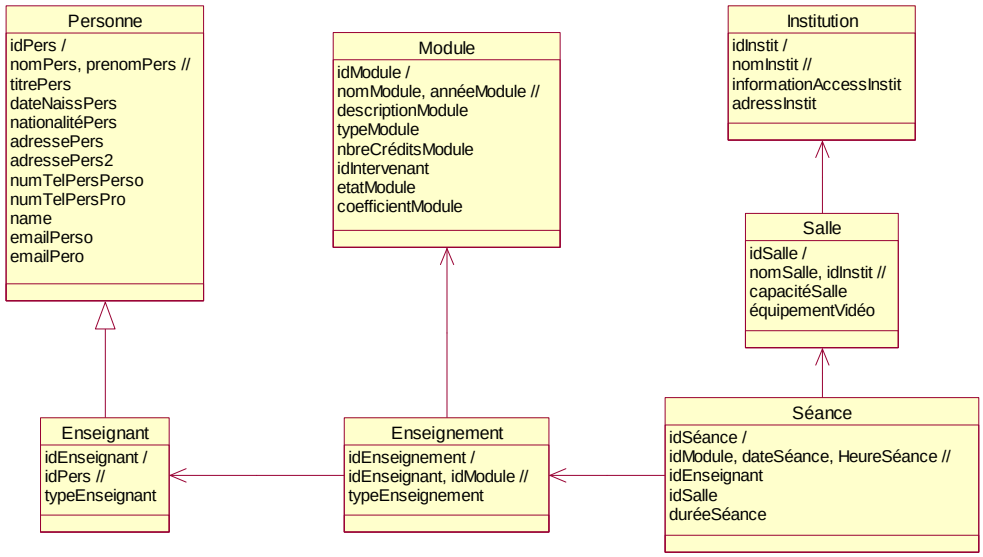

Figure 39 : Diagramme de l'hyperclasse *hcl_Implication_Enseignant*

L'évaluation de l'implication est individuelle pour chaque enseignant de la formation. La Figure 40 représente le graphe de navigation de *hcl_Implication_Enseignant* ; *Enseignant* est sa *classe racine*.

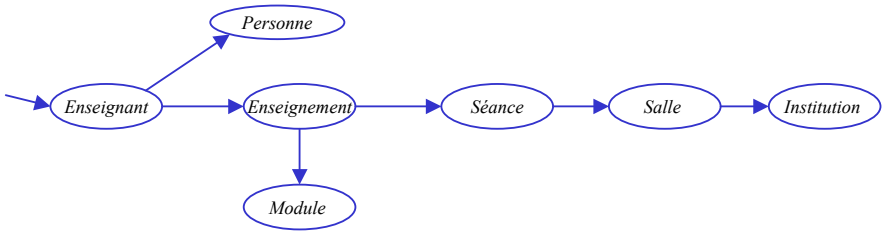

Figure 40 : Graphe de navigation de *hcl_Implication_Enseignant*

Le calcul des primes s'effectue grâce à une hyperméthode *calcul_prime_enseignant()* que nous définissons dans la suite.

```
float calcul_prime_enseignant(idho)
{
set prime_seance= multiplication(idho.consultation(Seance,
Institution), Seance.dureeSeance, Institution.primeHoraire) ;
float prime_enseignant :=somme(prime_seance) ;
return (prime_enseignant);
}
```

Chaque séance a une durée (*Seance.dureeSeance*) et chaque institution a une prime horaire qui lui est propre (*Institution.primehoraire*). Pour chaque séance qu'assure un enseignant, nous calculons une prime (*prime_seance*). La prime de l'enseignant est la somme de ses primes de séances.

L'opération *somme(ensemble_valeurs)* est une opération qui retourne la somme des valeurs passées en paramètre.

4.5. CONSERVATION DES HYPERMÉTHODES ET DE LEURS RÉSULTATS

Dans notre démarche, nous veillons à garantir une relative indépendance entre le diagramme d'une hyperclasse et ses hyperméthodes, en particulier lors de situations d'évolution de l'hyperclasse.

Il s'agit de préserver les hyperméthodes des dysfonctionnements dus à l'évolution de la définition d'une hyperclasse : (i) les hyperméthodes doivent rester opérationnelles ; nous parlons de *conservation d'hyperméthode* et (ii) les hyperméthodes doivent continuer à retourner le même résultat ; nous parlons de *conservation du résultat*.

4.5.1. Conservation d'hyperméthode

Nous parlons de *conservation d'hyperméthode* lorsque les hyperméthodes définies sur une hyperclasse *hcl* restent applicables sur ces hyperobjets, et continuent à fonctionner après la modification de *hcl*, sans aucune modification ou recompilation.

Pour garantir la conservation de méthodes, certaines conditions doivent être vérifiées. La conservation d'hyperméthode est possible lorsque les éléments du contexte de l'hyperméthode n'ont pas été supprimés par l'opération d'évolution de l'hyperclasse : il est nécessaire que les classes, attributs, méthodes, hyperattributs, autres hyperméthodes et chemins d'accès directement impliqués dans l'hyperméthode fassent toujours partie de l'hyperclasse et y demeurent accessibles.

4.5.2. Conservation de résultat

Nous parlons de *conservation de résultat* lorsque des hyperméthodes continuent à restituer les mêmes valeurs après l'évolution de la structure de l'hyperclasse. Ce résultat est possible si l'hyperméthode continue à atteindre les mêmes informations qu'elle aurait atteintes avant l'évolution de l'hyperclasse.

Dans une hyperclasse *hcl*, la conservation de résultat ne peut être garantie dans les situations suivantes :

1. en cas de création ou de suppression d'un hyperattribut $cl.att/_{chacc}$, l'ensemble d'objets de *cl* atteints dans *hcl* est modifié ; si des hyperattributs implicites basés sur *cl.att* ou *att* sont utilisés dans une hyperméthode *hm* de *hcl*, la conservation de résultat ne peut être garantie ;

2. en cas de suppression ou de mise à jour d'un chemin d'accès *chacc* d'une classe *cl,* l'ensemble d'objets de *cl* atteints dans *hcl*, via les hyperattributs explicites ou implicites, est différent. La conservation de résultat ne peut pas être garantie pour les hyperméthodes impliquant *cl* ou des occurrences d'attributs de *cl*.

5. HOMOGÉNÉITÉ DES CONCEPTS DE CLASSE ET D'HYPERCLASSE

Une hyperclasse est définie sur un ensemble de (une ou plusieurs) classes d'un diagramme de classes. Une hyperclasse dispose d'hyperattributs, d'hyperméthodes et d'instances sous forme d'hyperobjets. Le concept d'hyperclasse est une généralisation du concept de classe :

1. une classe est une hyperclasse particulière construite sur une seule classe ;
2. une occurrence d'attribut dans une classe est un hyperattribut de l'hyperclasse ;
3. une méthode de classe est une hyperméthode qui n'agit que sur les objets de la classe ;

Ainsi la mise en place du concept d'hyperclasse n'exige pas beaucoup d'opérations supplémentaires mais sophistique seulement davantage les opérations existantes qui ont été développées pour supporter le concept de classe.

Une hyperclasse admet ainsi deux formes:

1. d'un côté, une hyperclasse se compose d'un ensemble connexe et complet de classes. C'est sa forme *déployée* ;
2. d'un autre côté, une hyperclasse est une classe avec les mêmes propriétés bien connues des classes : attributs, méthodes, liens avec d'autres classes, versions, objets, identifiant d'objet, parties publiques et privées... C'est sa forme *repliée*.

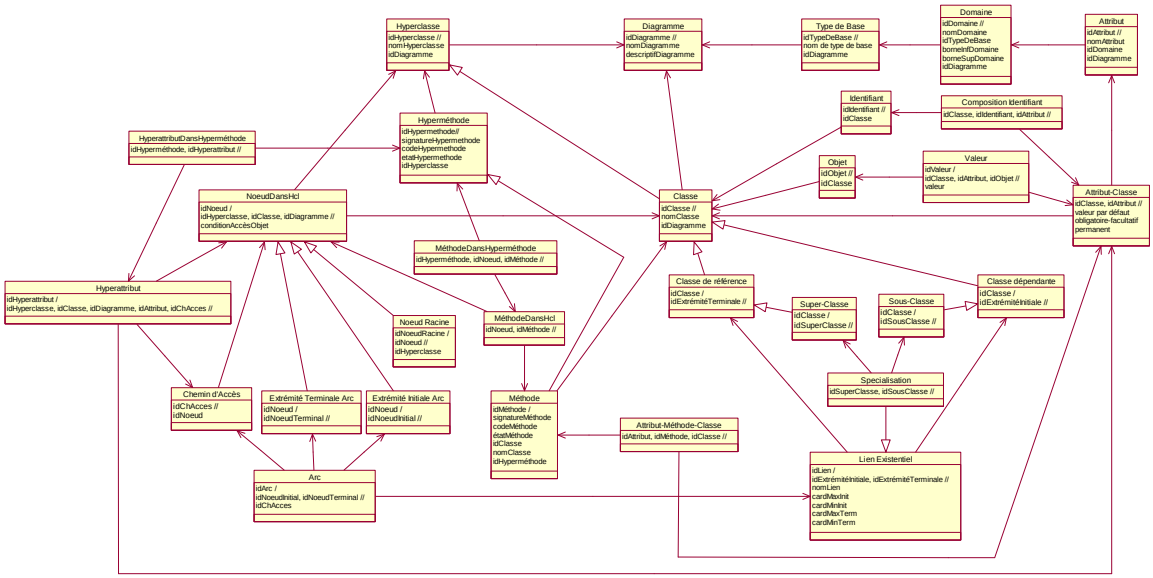

Figure 41 : Modèle d'hyperclasse exprimé en *binex*

6. LES HYPERCLASSES DANS L'INGÉNIERIE DES SIS

Nous nous intéressons aux interactions entre un modèle de SI et l'espace des activités.

6.1. HYPERCLASSES ET ZONES DE RESPONSABILITÉ

Les responsabilités organisationnelles liées à l'exercice d'un métier ou d'une fonction dans une entreprise ou dans une institution traitent de l'information en effectuant leurs activités. Nous appelons *zone de responsabilité* l'ensemble des activités qu'une responsabilité organisationnelle effectue en traitant de l'information. L'*espace informationnel* d'une zone de responsabilité représente l'ensemble des informations nécessaires pour l'exercice de ces activités.

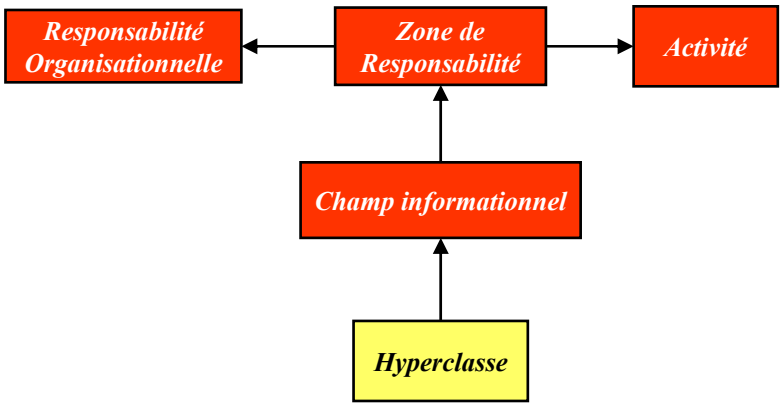

Figure 42 : Zones de responsabilité et hyperclasses

Un diagramme de classes représente la structuration de l'information dans le SI : les informations et données manipulées au niveau des zones de responsabilité sont contenues dans les objets des classes du SI et des liens entre ces objets. Une hyperclasse est la représentation au niveau d'un modèle de SI d'un espace informationnel : c'est une projection sur le diagramme de classes de l'espace informationnel d'une zone de responsabilité.

L'Annexe F décrit les hyperclasses définies dans *M@TIS* pour supporter les activités de chaque acteur de la formation.

Unité sémantique d'un espace informationnel

Une hyperclasse *hcl* est la représentation d'un espace informationnel au niveau d'un modèle de SI ; son diagramme C_{hcl} présente alors une *unité sémantique*.

L'unité sémantique d'une hyperclasse est le sens que traduit l'association de l'ensemble des concepts que représentent les classes qui en font partie et leurs liens.

En linguistique, la sémantique porte sur le sens de la combinaison des mots. Dans le cas des hyperclasses et par extension, la sémantique porte sur le sens que traduit le regroupement d'un ensemble de classes, des concepts qu'elles représentent et de leur association.

6.2. INTEROPÉRABILITÉ DES HYPERCLASSES

L'interaction de plusieurs zones de responsabilité autour du SI d'une entreprise ou d'une institution fait que la présence de zones de recouvrement entre leurs différents espaces informationnels est inéducable. Ces chevauchements s'ils ne sont pas détectés dès les premières phases d'analyse et de conception du SI occasionnent des conflits et perturbe le fonctionnement du SI et des activités métiers qui leur sont sous-jacentes. « Les zones de responsabilités ne peuvent pas disposer des informations d'une manière privée. Elles sont amenées à partager les informations de leurs espaces informationnels respectifs, c'est à dire non seulement à accepter que d'autres espaces informationnels puissent accéder leurs informations, mais aussi qu'ils

puissent les modifier » (Léonard & Parchet, 1998)[79]. Lorsque de telles situations de recouvrement d'informations par différents espaces informationnels sont mises en évidence, il convient d'étudier la coordination des activités autour des informations partagées et de préciser les protocoles de partage de ces informations entre les différents espaces et ceci afin de permettre aux zones de responsabilités impliquées par une situation de recouvrement d'assurer localement leurs activités.

6.2.1. Recouvrement d'hyperclasses

Une hyperclasse est définie sur un ensemble d'éléments d'un diagramme de classes. Lorsque nous considérons plusieurs hyperclasses, deux cas de figure peuvent se présenter :

1. les hyperclasses sont *disjointes* et mettent en évidence une *situation d'indépendance* ;
2. les hyperclasses sont *non-disjointes* et caractérisent une *situation de recouvrement*.

Nous appelons *zone de recouvrement* de plusieurs hyperclasses le sous-ensemble d'éléments du diagramme de classes que des hyperclasses ont en commun.

Le fait qu'il existe au moins une classe commune à plusieurs hyperclasses indique que les hyperclasses et leurs objets ne peuvent pas être considérés de manière indépendante. Le *recouvrement d'hyperclasses* met en évidence des situations de *recouvrement d'espace informationnels,* indiquant que (i) certaines problématiques et responsabilités organisationnelles concernées par les zones de responsabilité mises en jeu ne doivent pas être envisagées de manière indépendante et (ii) qu'il faut préciser la nature des règles de coordination qui réglementent (réglementeront) les interactions des zones de responsabilité d'une part, et les interactions entre les hyperclasses d'autre part. Nous parlons alors d'*interopérabilité des hyperclasses*. L'interopérabilité est « l'aptitude de deux ou plusieurs systèmes ou composants d'échanger de l'information et d'exploiter l'information échangée » (IEEE, 1990)[80] (WordNet, 2.0)[81].

Le recouvrement de zones de responsabilité se traduit au niveau du SI par un recouvrement des hyperclasses correspondantes. Les zones de recouvrement ne partagent pas alors uniquement les classes qui leurs sont communes, mais aussi les objets de ces classes : les hyperobjets des hyperclasses sont constitués à partir des objets des classes des hyperclasses. Les objets d'une classe commune à plusieurs hyperclasses sont partagés par ces hyperclasses. Une modification de l'état d'un objet d'une classe commune à plusieurs hyperclasses (classe appartenant à une zone de recouvrement d'hyperclasses) est perceptible au niveau de ces différentes hyperclasses.

Lorsque de telles situations de recouvrement d'informations par différents espaces informationnels sont mises en évidence, il convient d'étudier la coordination des activités autour

---

[79] (Léonard & Parchet, 1998) Léonard M., Parchet O., Étude des situations de recouvrements de l'information pour la conception des SI. Actes d'INFORSID'98, Mai 1998, Montpellier, France.

[80] (IEEE, 1990) IEEE Standard Computer Dictionary.

[81] (WordNet, 2.0) WordNet 2.0 by Princeton University.

des informations partagées et de préciser les protocoles de partage de ces informations entre les différents univers et hyperclasses : il est nécessaire de clarifier les règles de coordination et de spécifier des protocoles de coordination des activités, sous-jacents à une situation de recouvrement, appelés *protocoles de recouvrement*. De la même manière que des protocoles de recouvrement sont nécessaires en cas de recouvrement de zones de responsabilité, des protocoles de recouvrement sont nécessaires dans ce type de situation pour réglementer l'accès et la manipulation des classes et objets communs.

6.2.2. Protocoles de recouvrement

Le rôle des protocoles de recouvrement est de cordonner les activités autour des informations exercées par les différentes zones de responsabilités utilisant les hyperclasses d'un SI. Au niveau du SI, ils réglementent les interactions entre deux ou plusieurs hyperclasses impliquées dans une zone de recouvrement. Ces protocoles doivent nécessairement être précisés à partir du moment où une configuration du modèle des hyperclasses sans recouvrement ne peut être proposée.

En définissant précisément quelle zone de responsabilité est responsable de quelle partie d'évolution des objets contenue dans une zone de recouvrement, les protocoles de recouvrement fournissent une base pertinente pour comprendre et travailler la coordination des activités autour des informations du SI. L'étude des situations de recouvrement et notamment des contraintes de coordination qu'elles induisent, conduit à définir et à affiner des protocoles de partage de l'information entre zones de responsabilité.

Une partie des protocoles de recouvrement peut être prise en charge au niveau des spécifications statiques du SI, notamment à travers le concept d'hyperclasse. Ces protocoles peuvent être finalisés au niveau des spécifications dynamiques.

Par défaut, au niveau d'une hyperclasse, toutes les classes appartenant à l'hyperclasse sont accessibles en consultation, tout comme les attributs identifiants et référentiels de ces classes. Toutefois, il est possible de définir de manière explicite la nature des accès autorisés aux autres éléments de l'hyperclasse pouvant faire partie de zones de recouvrement :

1. Visibilité des objets et des hyperobjets

Dans une hyperclasse, il est possible de spécifier, dans le méta-modèle d'hyperclasse à travers l'attribut *conditionAccèsObjet* de la classe *NœudDansHcl* (Figure 38), si :

1. tous les objets des classes de l'hyperclasse sont visibles ;
2. uniquement les objets appartenant à des hyperobjets de l'hyperclasse sont visibles, ou si ;
3. uniquement un sous-ensemble des hyperobjets de l'hyperclasse est visible. Ce sous ensemble peut être décrit par une condition sur l'hyperobjet ou en explicitant l'ensemble des objets racines autorisés.

2. Visibilité des méthodes de classe

Dans une hyperclasse, il est possible de spécifier à travers la classe *MéthodeDansHyperméthode* (Figure 38), pour chacune de ses classes, les méthodes qui sont accessibles, notamment les méthodes de base de création, de mise-à-jour ou de suppression d'objets.

3. Hyperattributs

La définition d'un hyperattribut *hatt* dans une hyperclasse *hcl* indique que l'attribut *att* de la classe, sur lequel est défini *hatt*, est visible au niveau de *hcl* ; *att* est masqué au niveau de *hcl* sinon.

6.2.3. Exemples de recouvrement d'hyperclasses

Dans *M@TIS*, chacune des cinq catégories d'acteurs de la formation (*étudiant*, *enseignant*, *responsable de module*, *coordinateur* et *technicien*) dispose d'un espace informationnel qui décrit l'espace informationnel qui leur est attribué dans le cadre de leurs interactions avec le système. Ces espaces informationnels sont représentés par un ensemble d'hyperclasses (voir Annexe F) qui ont en commun plusieurs classes du diagramme de classes et mettent ainsi en évidence plusieurs situations de recouvrement entre les compétences des différents acteurs de la formation.

Voici deux exemples de situations de recouvrement dans *M@TIS*.

Les modules

La classe *Module* est partagée par plusieurs hyperclasses associées au coordinateur, aux étudiants, aux enseignants et aux responsables de module.

Le *coordinateur* est le seul acteur habilité à introduire de nouveaux modules dans *M@TIS*, mais aussi de les supprimer. Il a aussi la possibilité de mettre à jour les informations relatives aux modules. Dans *hcl_planning*[82], le *coordinateur* a les privilèges suivants :

1. Accès à tous les objets de la classe *Module* ;
2. Création, suppression et mise-à-jour d'objets dans *Module* ;
3. Accès à tous les attributs de *Module*.

Un *responsable de module* est entre autre responsable de la description de son module. Dans *hcl_gestion_module*[83], il a les privilèges suivants :

1. Accès en consultation à tous les objets de la classe *Module*, même ceux dont il n'est pas responsable ;

---

[82] Hyperclasse construite sur les classes *Module*, *Enseignement*, *Séance*, *Examen*, *Salle*, *Institution*, *Enseignant*, *ResponsableModule*, *Document* et *Personne*.

[83] Hyperclasse définie sur les classes *Module*, *Enseignement*, *Séance*, *Examen*, *Salle*, *Institution*, *Enseignant*, *ResponsableModule*, *Document* et *Personne*.

2. Suppression et mise-à-jour des objets dans *Module* dont il est responsable : ce sont les objets de Module, atteints dans *hcl_gestion_module* en partant de l'objet de *ResponsableModule* correspondant au *responsable de module* concerné ;
3. Accès à tous les attributs de *Module*.

Un *enseignant* ou un *étudiant* dans ses hyperclasses a les privilèges suivant sur *Module* :

1. Accès en consultation à tous les objets de la classe *Module*, même ceux auxquels il ne participe pas ;
2. Accès à tous les attributs de *Module*.

Les résultats des étudiants

Les résultats obtenus par un étudiant sont représentés par l'occurrence d'attribut *Inscription.noteModule* ; la classe *Inscription* est commune à plusieurs hyperclasses associées au coordinateur, aux étudiants, aux enseignants et aux responsables de module.

Chaque *responsable de module* est responsable de l'attribution des notes des étudiants de son module ; il a les privilèges suivants dans ses hyperclasses sur *Inscription* :

1. Accès en consultation à tous les objets de la classe *Inscription*, même ceux reliés à un module dont il n'est pas responsable ;
2. Mise-à-jour d'*Inscription.noteModule* dans les objets d'*Inscription* reliés à son module.

Chaque *étudiant* est responsable de son inscription à un module. Dans ses hyperclasses, il a donc les privilèges suivant sur *Inscription* :

1. Création d'objets d'Inscription liés à l'objet d'*Étudiant* qui lui correspond ;
2. Accès en consultation aux attributs *Inscription.idModule* et *Inscription.idÉtudiant* dans tous les objets de la classe *Inscription* ;
3. Accès en consultation à l'attribut *Inscription.noteModule* uniquement dans les objets liés à l'objet d'*Étudiant* qui lui correspond.

Un *coordinateur* ou un *enseignant* a les privilèges suivants sur Inscription :

4. Accès en consultation à tous les objets de la classe *Inscription*.

CONCLUSION

Dans ce chapitre, nous avons introduit les concepts d'hyperclasse, d'hyperobjet, d'hyperattribut et d'hyperméthode. Ces concepts ouvrent de nouvelles possibilités pour l'ingénierie des SI, tant pendant les phases de conception que de développement ou d'exploitation ou de manutention :

1) le concept d'hyperclasse introduit une forme de modularité dans la gestion de modélisation de SI ;
2) le concept d'hyperattribut réduit la connaissance nécessaire des détails de la modélisation autant pour des utilisateurs que pour des concepteurs, développeurs, etc. du SI ;

3) le concept d'hyperméthode constitue un avantage pour les concepteurs de SI ; ils n'ont pas à choisir à quelle classe particulière de l'hyperclasse associer une méthode ;

4) les espaces informationnels dans un SI ne peuvent plus disposer des informations de manière privée. L'étude du partage des responsabilités autour de la gestion des informations contenues dans un SI repose sur trois notions essentielles :

   a) le concept d'*hyperclasse*, représentation de l'espace informationnel d'une zone de responsabilité autour du SI ;

   b) la notion de *recouvrement* pour présenter des situations de partage de l'information entre les différentes hyperclasses d'un SI ;

   c) la notion de *protocole de recouvrement*, pour exprimer la coordination des activités des personnes autour du SI. Les concepts proposés dans ce chapitre offrent un ensemble de moyens pour définir ces protocoles de recouvrement et pour gérer de manière fine l'interopérabilité d'hyperclasses.

Le concept d'hyperclasse est une généralisation du concept de classe. Sa mise en place n'exige pas beaucoup d'opérations supplémentaires mais sophistique seulement davantage les opérations existantes qui ont été développées pour supporter le concept de classe. Dans le chapitre suivant, nous définissons les opérations d'évolution et de manipulation de diagrammes de classes avec des hyperclasses.

CHAPITRE III

OPÉRATIONS SUR LES HYPERCLASSES

INTRODUCTION

Ce chapitre est consacré aux opérations sur les hyperclasses. Ces opérations sont organisées en trois ensembles : (i) *évolution*, (ii) *extraction* et (iii) *intégration* d'hyperclasses. Elles constituent une base embryonnaire pour une ingénierie des SI basée sur les hyperclasses et plus tard sur les composants de SI.

1. Évolution d'hyperclasse

L'espace informationnel que représente une hyperclasse dans un diagramme de classes est évolutif : il peut être amené à être restructuré, réduit, ou étendu en intégrant de nouveaux concepts ou d'autres espaces informationnels. Pour maintenir et préserver l'adéquation entre une hyperclasse et un espace informationnel en cas d'évolution, nous proposons un ensemble complet d'opérations de manipulation et de restructuration des hyperclasses.

2. Extraction d'hyperclasse

Il s'agit d'extraire à partir de diagrammes de classes des hyperclasses représentant des situations remarquables ou récurrentes pour constituer des bibliothèques de solutions génériques. Grâce aux opérations d'évolution, des variantes de ces hyperclasses peuvent être construites et intégrer dans d'autres modèles de SI.

3. Intégration d'hyperclasse

Traditionnellement, l'intégration de modèles consiste à prendre en compte deux modèles de SI et de proposer un modèle global et unifié. Dans le cas des hyperclasses, nous nous intéressons à l'intégration d'une hyperclasse dans un modèle existant de SI, et à l'intégration d'une hyperclasse dans une autre hyperclasse du même modèle.

1. ÉVOLUTION D'HYPERCLASSE

La première partie de cette section est consacrée aux opérations sur les hyperclasses. Ces opérations ont la particularité de n'altérer ni de modifier aucun élément du diagramme de classes sur lequel est définie l'hyperclasse.

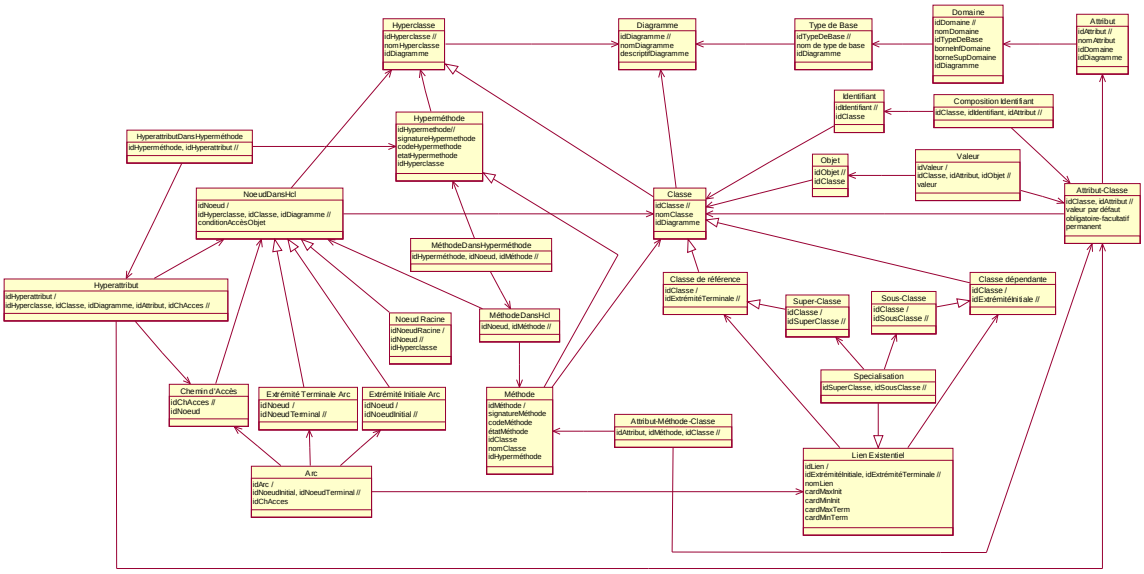

Figure 43 : Modèle d'hyperclasse exprimé en *binex*

Dans la seconde partie, nous revisitons certaines opérations d'évolution de *binex*. En effet, le méta-modèle d'hyperclasse est défini par extension de *binex*. De part la relation entre les deux méta-modèles et la nature existentielle des liens qui y sont employés, plusieurs opérations d'évolution de *binex* ont des répercussions sur les hyperclasses définies dans un digramme de classes.

1.1. OPÉRATIONS SUR LES HYPERCLASSES

L'ensemble des opérations sur les hyperclasses (Table 1) est obtenu par l'application des trois opérations génériques (i) *créer*, (ii) *supprimer* et (iii) *Mettre à jour* sur chacun des éléments du méta-modèle d'hyperclasse, et est donc complet.

Ces opérations de restructuration des hyperclasses ont une particularité : elles ne modifient pas le diagramme de classes sur lequel est définie l'hyperclasse manipulée : aucune classe, aucun attribut, aucun lien ni méthode du diagramme de classes ni aucun objet n'est créé, supprimé ou mis à jour. Ces opérations ne modifient pas non plus les autres hyperclasses définies sur le même diagramme de classes.

Chacune de ces opérations doit préserver la validité de l'hyperclasse manipulée ; l'hyperclasse qui subit une évolution doit être maintenue connexe et complète dans le diagramme de classes. La *connexité* et la *complétude* de l'hyperclasse sont considérées comme des règles d'intégrité définies sur le méta-modèle d'hyperclasse. Dans la Table 1, les indications *R1* et *R2* signalent que l'opération concernée constitue un risque d'enfreindre respectivement la condition de connexité et de complétude de l'hyperclasse.

	Créer()		*Supprimer()*		*Mettre-à-jour()*	
1. Hyperclasse	1.1. Créer une hyperclasse		1.2. Supprimer une hyperclasse		1.3. Renommer une hyperclasse	
	Atteignable		*Atteignable*		*Atteignable*	
	R1[84]	*R2*[85]	-	-	-	-
2. Nœud dans hyperclasse	2.1. Inclure une classe terminale dans une hyperclasse		2.2. Exclure une classe d'une hyperclasse		2.3. Modifier la condition de visibilité des objets d'une classe dans une hyperclasse	
	Atteignable		*Atteignable*		*Atteignable*	
	R1	*R2*	*R1*	*R2*	-	-
3. Nœud/classe racine	3.1. Définir un nœud racine		3.2. Supprimer un nœud racine		-	
	Non atteignable		*Non atteignable*		*Non applicable*	
	-	-	-	-	-	-
4. Chemin d'accès	4.1. Créer un chemin d'accès		4.2. Supprimer un chemin d'accès		4.3. Inclure un nœud dans un chemin d'accès 4.4. Exclure un nœud d'un chemin d'accès	
	Atteignable		*Atteignable*		*Non atteignable*	
	R1	*R2*	*R1*	-	*R1*	*R2*
5. Arc	5.1. Ajouter un arc à un chemin d'accès		5.2. Supprimer un arc d'un chemin d'accès		-	
	Atteignable		*Atteignable*		*Non applicable*	
	-	-	*R1*	-	-	-
6. Hyperattribut	6.1. Créer un hyperattribut		6.2. Supprimer un hyperattribut		-	
	Atteignable		*Atteignable*		*Non applicable*	
	-	-	-	-	-	-
7. Méthode de classe dans l'hyperclasse	7.1. Rendre visible une méthode de classe		7.2. Rendre invisible une méthode de classe		-	
	Atteignable		*Atteignable*		*Non applicable*	
	-	-	-	-	-	-

[84] Risque sur la connexité de l'hyperclasse.

[85] Risque sur la complétude de l'hyperclasse.

8. Hyperméthode	8.1. Créer une hyperméthode		8.2. Supprimer une hyperméthode		8.3. Modifier une hyperméthode 8.4. Activer / désactiver une hyperméthode 8.4. Renommer une hyperméthode	
	Atteignable		*Atteignable*		*Atteignables*	
	-	-	-	-	-	-

Table 1 : Opérations sur le modèle d'hyperclasse

L'exécution de certaines opérations sur les hyperclasses met automatiquement le modèle dans un état invalide, et il est nécessaire de les combiner avec d'autres opérations pour aboutir à un résultat valide de l'évolution. Ces opérations ne peuvent qu'être invoquées à travers d'autres opérations et sont dites *non atteignables*

D'autres opérations sur les hyperclasses ne sont pas applicables :

1. Modifier le noeud racine

Une hyperclasse a une classe racine, et il n'est pas autorisé de la modifier.

2. Mettre à jour un arc de chemin d'accès

Un arc est défini sur une paire de nœuds d'un chemin d'accès. Il ne peut qu'être supprimé.

3. Mettre à jour un hyperattribut

Un hyperattribut est défini par rapport à l'occurrence d'un attribut dans une classe d'une hyperclasse, et par rapport à un chemin d'accès à cette classe dans l'hyperclasse. La mise-à-jour d'un hyperattribut s'effectue à travers la mise à jour de l'un de ces deux éléments.

4. Mettre à jour une méthode de classe dans l'hyperclasse

Le concept de méthode de classe dans une hyperclasse représente la visibilité de la méthode au niveau de l'hyperclasse. Une méthode est soit visible soit invisible dans une hyperclasse.

La conservation, dans la mesure du possible, du maximum d'informations accessibles au niveau de l'hyperclasse, la préservation des hyperméthodes des hyperclasses et la conservation de leurs résultats restent les priorités des opérations sur les hyperclasses. Nous parlons de *conservation d'hyperméthode* lorsque les hyperméthodes définies sur une hyperclasse *hcl* restent applicables sur ces hyperobjets, et continuent à fonctionner après la modification de *hcl*, sans aucune modification ou recompilation. Nous parlons de *conservation de résultat* lorsque des hyperméthodes continuent à retourner les mêmes valeurs même après l'évolution de la

structure de l'hyperclasse. Ce résultat est possible si l'hyperméthode continue à atteindre les mêmes informations qu'elle aurait atteintes avant l'évolution de l'hyperclasse.

Les opérations sur le modèle d'hyperclasse sont présentées sous forme de tableaux dont la première ligne indique les paramètres de l'opération, la deuxième les pré-conditions à leur application, la troisième les post-actions sur le modèle et la quatrième les post-actions et leur impact sur les hyperméthodes. Si nécessaire, une cinquième ligne signale des situations particulières ou indique des remarques importantes.

1. Hyperclasse

1.1. Créer une hyperclasse

Paramètres :	▪ *hcl*, nom de la nouvelle hyperclasse ; ▪ *ecl,* ensemble des classes de *hcl* ; ▪ cl_{cr}, la classe racine de *hcl*.
Pré-cond. :	▪ Le nom de l'hyperclasse n'est pas déjà utilisé dans le diagramme de classes pour une autre classe ou hyperclasse ; ▪ *ecl* n'est pas un ensemble vide ; ▪ $cl_{cr} \in ecl$; ▪ *ecl* est connexe et complet dans le diagramme de classes.
Post-act. Modèle :	▪ Créer l'hyperclasse *hcl* ; ▪ Initialiser G_{hcl}, le graphe de navigation de *hcl* : pour chaque classe cl_i d'*ecl*, créer un nœud n_i dans G_{hcl} ; le nœud correspondant à cl_{cr} est désigné comme nœud racine du graphe de navigation ; ▪ Créer les chemins d'accès des classes de *hcl* ; ▪ Définir les hyperattributs construits sur les attributs identifiants et les attributs référentiels des classes de *hcl ;* ▪ Définir les conditions d'accès aux hyperobjets et aux objets des classes de *hcl* ; ▪ Définir la visibilité des méthodes des classes d'*ecl* dans *hcl*.
Impact h-méthodes :	▪ Définir les hyperméthodes de *hcl*.
Remarques :	▪ Dans une hyperclasse, il existe au moins un hyperattribut pour chaque occurrence d'un attribut identifiant ou référentiel dans une classe de l'hyperclasse. ▪ Si une ou plusieurs classes de *hcl* font partie d'une ou de plusieurs autres hyperclasses du diagramme de classes, elles caractérisent une ou plusieurs situations de recouvrement ⇒ mettre en place les protocoles de recouvrement adaptés (voir la section 6.2. Interopérabilité des hyperclasses P.95).

Exemple de création d'hyperclasse

L'opération `créerHyperclasse(hcl_étudiant, {Étudiant, Personne, Inscription, Module, Projet})` créer une nouvelle hyperclasse nommée

hcl_étudiant (Figure 44) dans le diagramme de classes de l'exemple *M@TIS*, définie sur les classes *Étudiant*, *Personne*, *Inscription*, *Module* et *Projet*.

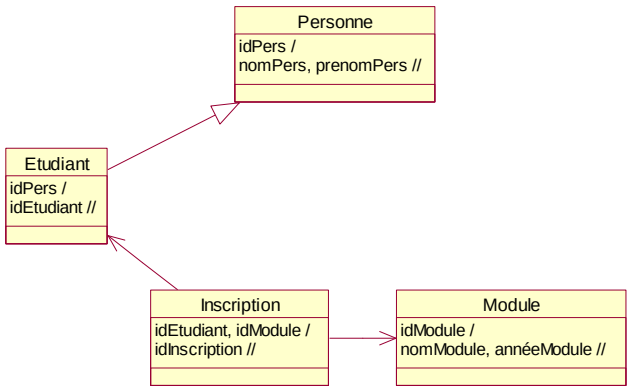

Figure 44 : Diagramme de l'hyperclasse *hcl_étudiant*

La définition du graphe de navigation $G_{hcl\_Résultats\_Étudiant}$ (Figure 45) de *hcl_Résultats_Étudiant* débute par la création d'un nœud correspondant à chacune des classes *Étudiant*, *Personne*, *ÉtudiantDiplômé*, *Inscription*, *Module* de l'hyperclasse. Le nœud correspondant à la classe racine (*Étudiant*) est désigné comme le nœud racine de $G_{hcl\_Résultats\_Étudiant}$. $G_{hcl\_Résultats\_Étudiant}$ est ensuite construit par union des chemins d'accès à chacun de ses nœuds à partir du nœud racine.

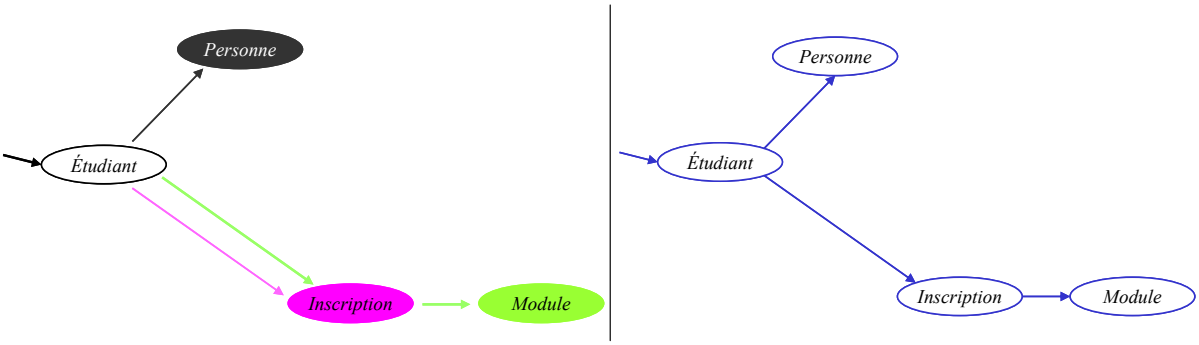

Figure 45 : Chemins d'accès et graphe de navigation de *hcl_étudiant*

Les attributs identifiants et les attributs référentiels de chacune des classes de *hcl_étudiant* sont par défaut visibles au niveau de l'hyperclasse. Au moins un hyperattribut pour chacun de ces attributs est défini dans *hcl_étudiant*.

Toutes les classes de *hcl_étudiant* font partie de plusieurs autres hyperclasses (*hcl_résultats_étudiants*, *hcl_Implication_Enseignant, etc.*). Il est nécessaire de définir pour chaque classe faisant partie d'une ou plusieurs zones de recouvrement la nature des accès autorisés à ces objets, et l'ensemble des méthodes de classes visibles au niveau de chaque hyperclasse.

1.2. Supprimer une hyperclasse

Paramètres :	▪ L'hyperclasse *hcl*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer les hyperattributs de *hcl* ; ▪ Supprimer les différents chemins d'accès aux classes de *hcl ;* ▪ Supprimer les nœuds qui correspondent aux classes de *hcl* ; ▪ Supprimer *hcl*.
Impact h-méthodes :	▪ Supprimer les hyperméthodes de *hcl*.
Remarque :	▪ La suppression de l'hyperclasse n'entraîne ni la suppression des classes sur laquelle elle est définie, ni celle de leurs attributs ou de leurs objets.

Exemple de suppression d'hyperclasse

L'opération `supprimerHyperclasse(hcl_étudiant)`supprime l'hyperclasse *hcl_étudiant* du diagramme de classes de *M@TIS*. Aucune des classes *Étudiant*, *Personne*, *Inscription* et *Module* ni aucun de leurs objets n'est supprimé ou modifié par cette opération.

1.3. Renommer une hyperclasse

Paramètres :	▪ Le nouveau nom de l'hyperclasse.
Pré-cond. :	▪ Le nouveau nom de l'hyperclasse n'est pas déjà utilisé dans le diagramme de classes pour une autre classe ou hyperclasse.
Post-act. Modèle :	▪ Renommer l'hyperclasse.
Impact h-méthodes :	▪ Aucun.

2. Nœud dans hyperclasse

2.1. Inclure une classe terminale dans une hyperclasse

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ *ecl,* ensemble des classes de *hcl* ; ▪ G_{hcl}, le graphe de navigation de *hcl* ; ▪ *cl*, la classe à insérer dans *hcl*.
Pré-cond. :	▪ L'ensemble ($ecl \cup \{cl\}$) est connexe et complet dans le diagramme de classes.
Post-act. Modèle :	▪ Créer un nouveau nœud correspondant à *cl* dans le graphe de navigation de *hcl* ; ▪ Créer un ou plusieurs chemins d'accès à *cl* dans *hcl* (op. *Créer un chemin d'accès*) ; ▪ Créer le ou les chemins d'accès à *cl* ; ▪ Pour chaque attribut identifiant et chaque attribut référentiel de *cl*, créer un hyperattribut dans *hcl* ; ▪ Définir la condition de visibilité des objets de *cl* dans *hcl* ; ▪ Définir la visibilité des méthodes de *cl* dans *hcl*.
Impact h-méthodes :	▪ Ni les chemins d'accès des classes initiales de *hcl*, ni ses hyperattributs ne sont modifiés par cette opération, donc la conservation des hyperméthodes et des résultats est garantie pour les hyperméthodes de *hcl* déjà définies.
Remarque :	▪ Si *cl* fait partie d'une ou de plusieurs autres hyperclasses du diagramme de classes, son insertion dans *hcl* peut créer une ou plusieurs situations de recouvrement (voir la section 6.2. Interopérabilité des hyperclasses P.95).

Exemple d'insertion d'une classe dans une hyperclasse

Lors de l'inclusion de la classe *ÉtudiantDiplômé* dans *hcl_étudiant* (Figure 46), un nouveau nœud correspondant à la classe *ÉtudiantDiplômé* est crée dans $G_{hcl\_étudiant}$, le graphe de navigation de *hcl_étudiant* (Figure 47).

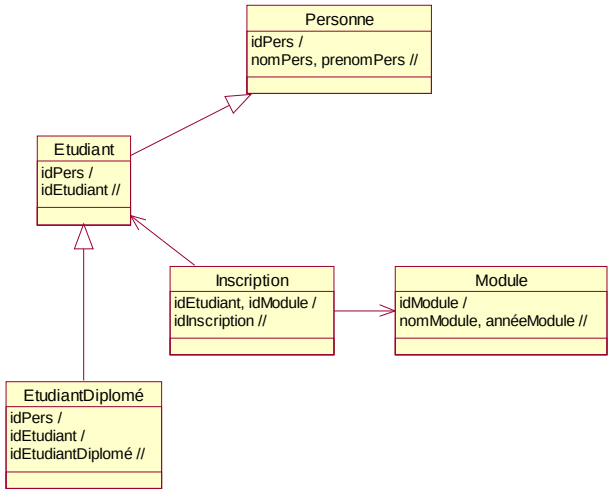

Figure 46 : Nouveau diagramme de l'hyperclasse *hcl_étudiant*

La classe *ÉtudiantDiplômé* ne peut avoir qu'un seul chemin d'accès à partir d'*Étudiant* : $G^{1}_{\text{ÉtudiantDiplômé, hcl\_étudiant}}$ ={{*Étudiant*, *ÉtudiantDiplômé*} (*Étudiant*, *ÉtudiantDiplômé*)}. Le graphe de navigation de *hcl_étudiant* est mis à jour pour intégrer le chemin d'accès à la nouvelle classe.

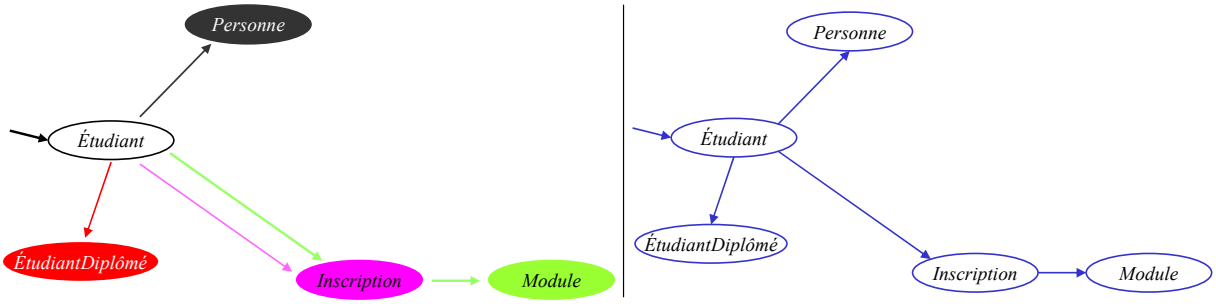

Figure 47 : Nouveau chemin d'accès à *ÉtudiantDiplômé* dans *hcl_étudiant* et $G_{hcl\_étudiant}$ mis à jour

idPers, *idÉtudiant* et *idÉtudiantDiplômé* sont des attributs identifiants de la classe *ÉtudiantDiplômé*. Ils sont automatiquement visibles au niveau de *hcl_étudiant* et les hyperattributs *ÉtudiantDiplômé.idPers*$_{/1}$, *ÉtudiantDiplômé.idÉtudiant*$_{/1}$ et *ÉtudiantDiplômé.idÉtudiantDiplômé*$_{/1}$ sont créés.

La visibilité des autres attributs des classes, de leurs méthodes et de leurs objets est décidée en fonction de la zone de responsabilité à laquelle sera associée *hcl_étudiant*.

Exemple d'inclusion d'un ensemble de classes dans une hyperclasse

Il est possible d'insérer en même temps plusieurs classes dans une hyperclasse. La Figure 48 montre le diagramme de *hcl_étudiant* après l'inclusion des classes *Projet* et *SupervisionProjet*.

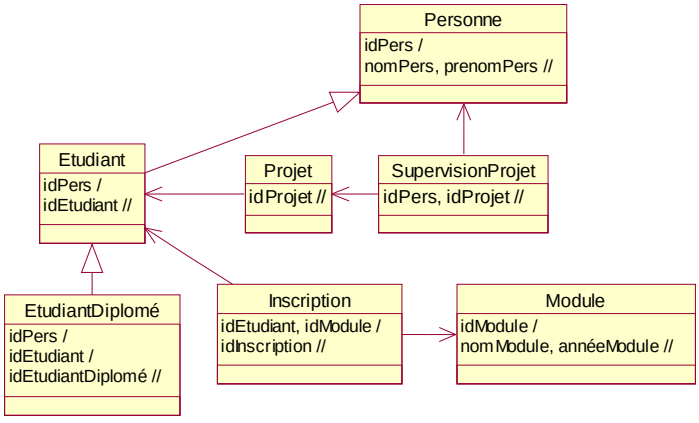

Figure 48 : Nouveau diagramme de l'hyperclasse *hcl_étudiant*

$G_{hcl\_étudiant}$ est mis à jour de manière à intégrer les nœuds correspondants aux classes *Projet* et *SupervisionProjet* et leurs chemins d'accès (Figure 49).

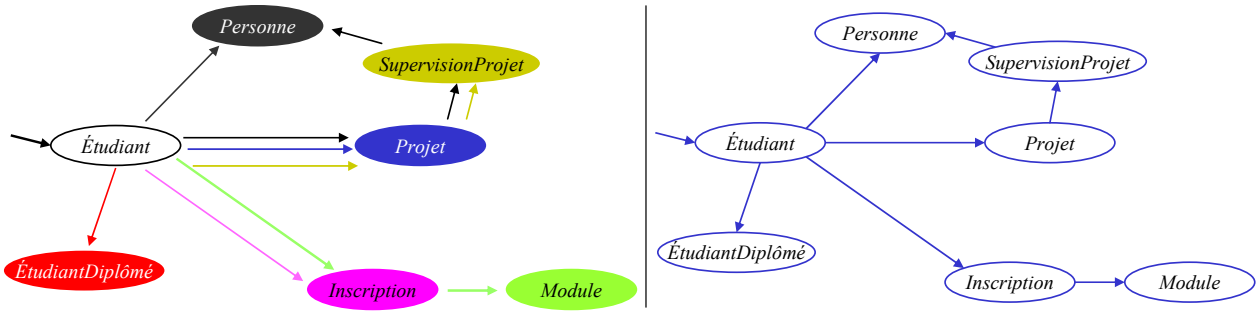

Figure 49 : $G_{hcl\_étudiant}$ après l'insertion de *Projet* et de *SupervisionProjet*

2.2. Exclure une classe d'une hyperclasse

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ *ecl*, l'ensemble des classes de *hcl* ; ▪ La classe *cl* à exclure d'*ecl* ; *nd* est le nœud correspondant à *cl* dans le graphe de navigation de *hcl*.
Pré-cond. :	▪ $cl \in ecl$; ▪ *cl* n'est pas la classe racine de *hcl*.
Post-act. Modèle :	▪ Supprimer le ou les chemins d'accès à *cl* dans *hcl* ; ▪ Pour tout chemin d'accès à une classe de *hcl,* parcourant *nd*, supprimer *nd* du chemin (op. *Supprimer un nœud du chemin d'accès*) ; ▪ Supprimer *nd* du graphe de navigation de *hcl* ; ▪ Exclure de *hcl* toute classe qui ne dispose plus d'un chemin d'accès valide (op. *Exclure une classe de l'hyperclasse sans la consolider*) ; ▪ Supprimer les hyperattributs construits sur des attributs de classes exclues de *hcl ;* ▪ Supprimer les hyperattributs basés sur des chemins d'accès rendus invalides.
Impact h-méthodes :	▪ L'exclusion entraîne la suppression de chemins d'accès et l'exclusion de classes ; ▪ Il est nécessaire de revisiter les hyperméthodes de *hcl* dont des éléments de leurs contextes ont été supprimés ; ▪ Les hyperméthodes dont le contexte n'est pas modifié bénéficient de la conservation d'hyperméthode et de résultat.

Exemple d'exclusion de classe #1

Considérons l'hyperclasse *hcl_Résultats_Étudiant* (Figure 50), construite sur l'ensemble de classes {*Étudiant*, *Personne*, *Projet*, *ÉtudiantDiplômé*, *Inscription*, *Module*}.

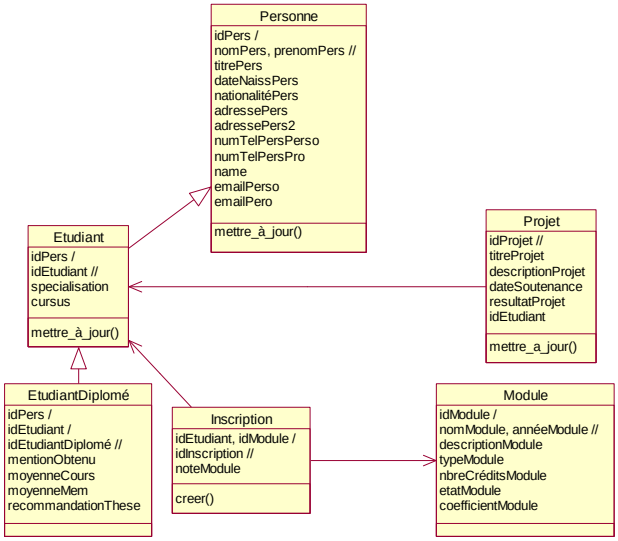

Figure 50 : Diagramme initial de *hcl_Résultats_Étudiant*

La Figure 51 représente le graphe de navigation et les chemins d'accès aux classes de *hcl_Résultats_Étudiant.*

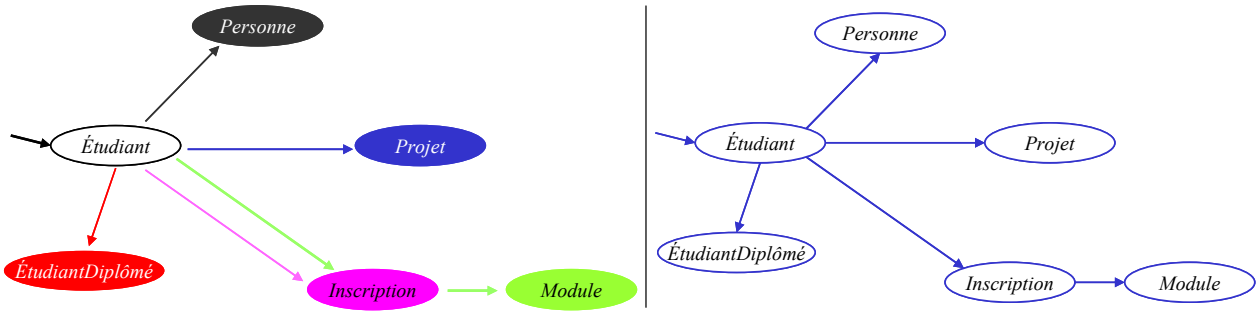

Figure 51 : Chemin d'accès dans *hcl_Résultats_Étudiant* et $G_{hcl\_Résultats\_Étudiant}$

L'exclusion de la classe *Inscription* entraîne :

1. la suppression du chemin d'accès à Inscription. Ce chemin est constitué de l'arc (*Étudiant*, *Inscription*) ;
2. la suppression du chemin d'accès à *Module* ; celui-ci utilise le nœud *Inscription* ;
3. la suppression du nœud *Inscription* du graphe de navigation de *hcl_Résultats_Étudiant* ;
4. l'exclusion de *Module* : son seul chemin d'accès est supprimé ;
5. la suppression de tous les hyperattributs construits sur des attributs d'*Inscription* et de *Module* ;
6. l'inactivation de toutes les hyperméthodes de *hcl_Résultats_Étudiant* dont des éléments de leurs contextes ont été supprimés.

La Figure 52 montre le diagramme et le graphe de navigation de *hcl_Résultats_Étudiant* après l'exclusion d'*Inscription* et de *Module*.

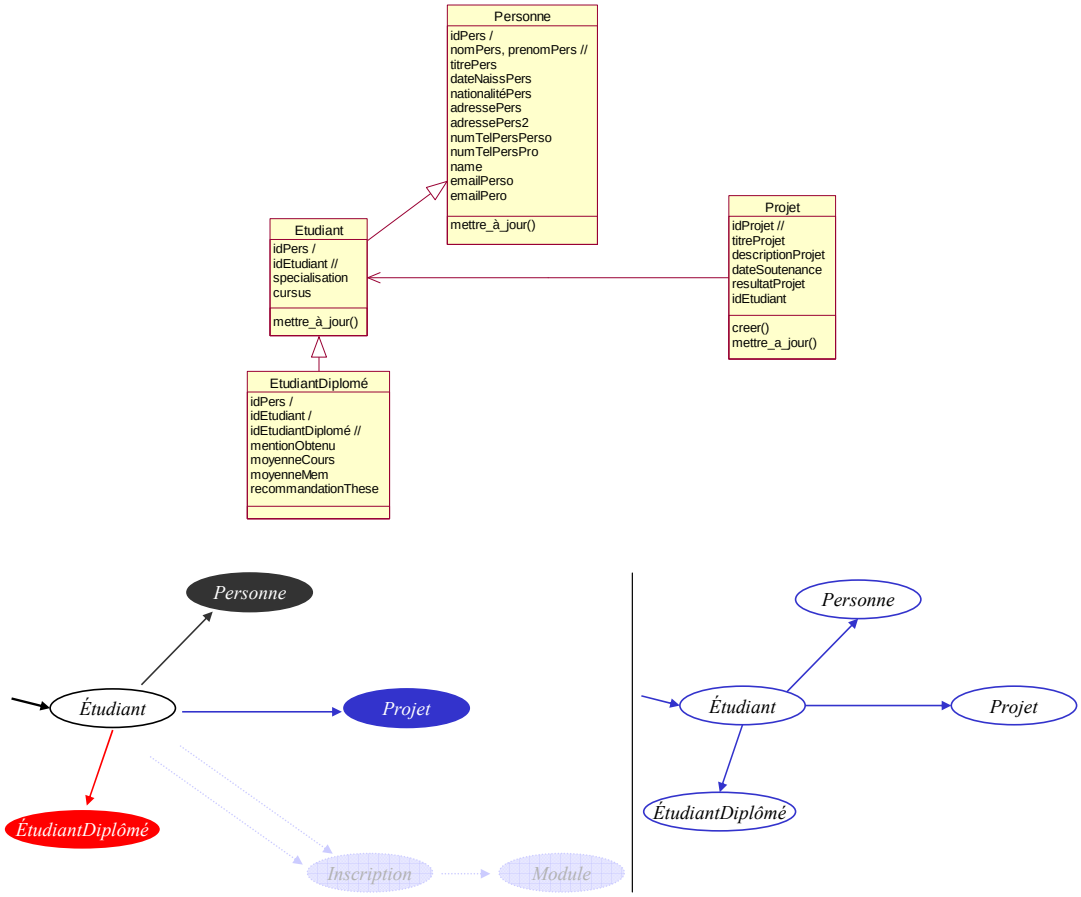

Figure 52 : Exclusion d'*Inscription* de *hcl_Résultats_Étudiant*

Exemple d'exclusion de classe #2

Considérons l'hyperclasse *hcl_Participation_Module* dont le diagramme et le graphe de navigation $G_{hcl\_Participation\_Module}$ sont respectivement représentés par la Figure 53 et la Figure 54.

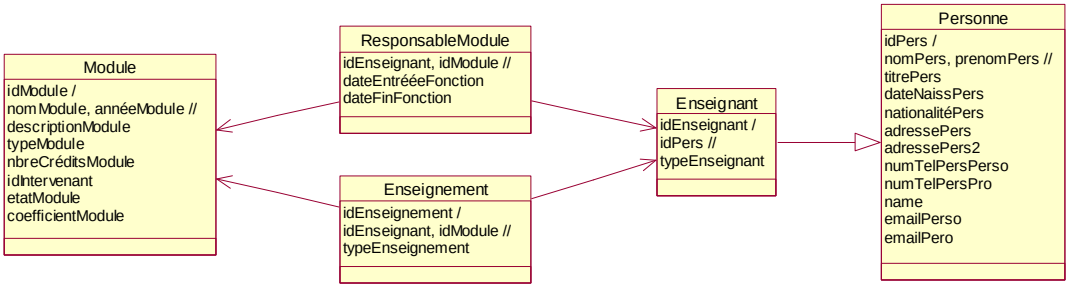

Figure 53 : Diagramme initial de *hcl_Participation_Module*

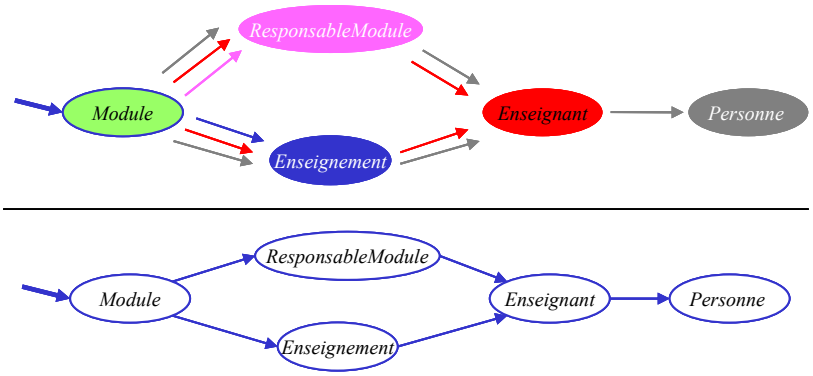

Figure 54 : Graphe de navigation initial de *hcl_Participation_Module*

Le chemin d'accès à *ResponsableModule* comprend uniquement l'arc (*Module*, *ResponsableModule*), tout comme celui de *Enseignement* qui comprend l'arc (*Module*, *Enseignement*).

Enseignant dispose de deux chemins d'accès : l'un passant par *Enseignement* ($G^1_{Enseignant,\ hcl\_Participation\_Module}$) et l'autre par *ResponsableModule* ($G^2_{Enseignant,\ hcl\_Participation\_Module}$) (Figure 55).

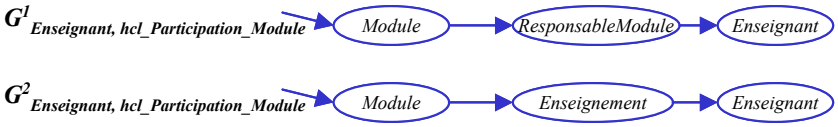

Figure 55 : Les deux chemins d'accès à *Enseignant* dans *hcl_Participation_Module*

Personne a un seul chemin d'accès représenté par la Figure 56.

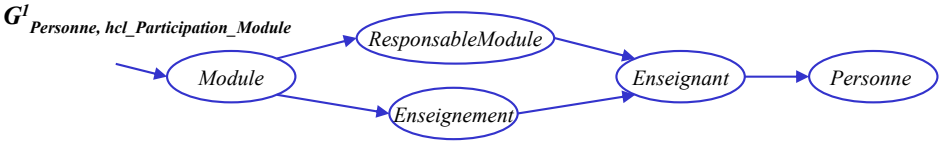

Figure 56 : Le chemin d'accès à *Personne* dans *hcl_Participation_Module*

L'exclusion de la classe *ResponsableModule* entraîne :

1. la suppression du chemin d'accès à *ResponsableModule* ;
2. le chemin d'accès $G^1_{Enseignant,\ hcl\_Participation\_Module}$ à *Enseignant* utilise le nœud *ResponsableModule* ; la suppression de *ResponsableModule* rend ce chemin obsolète (voir l'exemple de suppression de nœud d'un chemin d'accès plus bas) ;
3. le chemin d'accès $G^1_{Personne,\ hcl\_Participation\_Module}$ à *Personne* utilise le nœud *ResponsableModule* ; la suppression de *ResponsableModule* transforme ce chemin mais il reste valide (voir l'exemple de suppression de nœud d'un chemin d'accès plus bas) ;

La Figure 57 représente le diagramme de *hcl_Participation_Module* et son graphe de navigation $G_{hcl\_Participation\_Module}$ après l'exclusion de *ResponsableModule*.

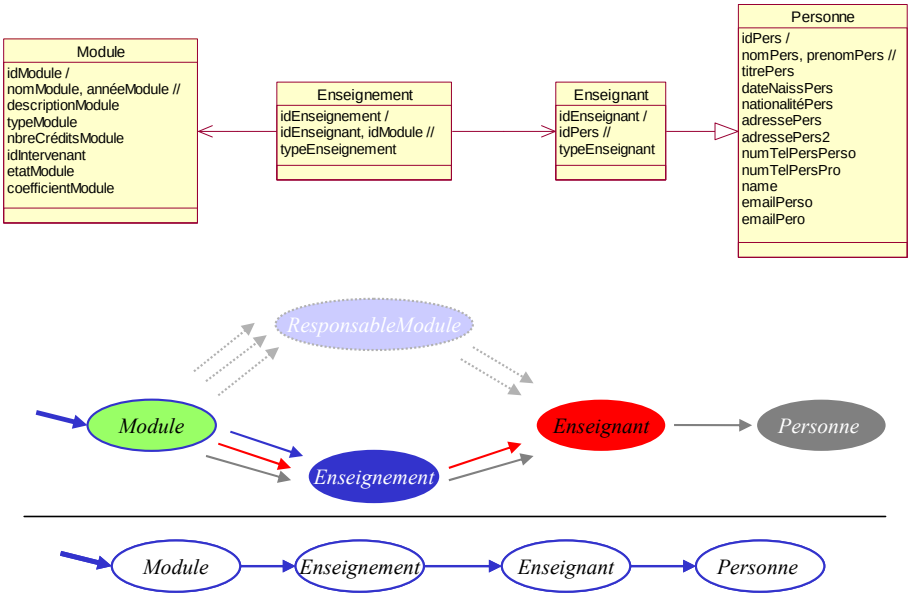

Figure 57 Exclusion de *ResponsableModule* de *hcl_Participation_Module*

L'hyperclasse *hcl_Participation_Module* reste connexe et complète dans le diagramme de classes. Elle représente dorénavant uniquement les enseignants participant aux enseignements de la formation.

Les hyperattributs $Enseignant.idEnseignant_{/2}$, $Enseignant.idPers_{/2}$ et $Enseignant.typeEnseignant_{/2}$ sont basés sur le chemin d'accès $G^2_{Enseignant,\ hcl\_Participation\_Module}$, supprimé lors de l'exclusion de ResponsableModule de *hcl_Participation_Module* ; ces hyperattributs sont supprimés de *hcl_Participation_Module*.

2.3. Modifier la condition de visibilité des objets d'une classe dans une hyperclasse

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ *cl*, classe de *hcl*. ▪ *condvis*, la nouvelle condition de visibilité des objets de *cl* dans *hcl*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Mettre à jour la condition de visibilité des objets de *cl* dans *hcl*.
Impact h-méthodes :	▪ La structure de l'hyperclasse n'est pas modifiée ; la conservation de méthode est garantie. ▪ La modification de condition de visibilité des objets de *cl* dans *hcl* peut entraîner une modification des hyperobjets de hcl ; la conservation de résultat n'est pas garantie.
Remarques :	▪ Par défaut, tous les objets de *cl* sont visibles dans *hcl.*

3. Nœud/classe racine

3.1. Définir un nœud racine

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ *ecl*, l'ensemble des classes de *hcl* ; ▪ cl_{cr}, la classe racine de *hcl*.
Pré-cond. :	▪ *hcl* ne possède pas de classe racine ; ▪ $cl_{cr} \in ecl$.
Post-act. Modèle :	▪ Spécialiser le nœud n_{rc} correspondant à cl_{cr} dans G_{hcl} en nœud racine.
Impact h-méthodes :	▪ Aucun.

3.2. Supprimer un nœud racine

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ cl_{cr}, la classe racine de *hcl*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer la spécialisation en nœud racine du nœud n_{rc} correspondant à cl_{cr} dans G_{hcl} ; ▪ Supprimer G_{hcl} et l'ensemble des chemins d'accès des classes de *hcl* dans G_{hcl} ; ▪ Supprimer tous les hyperattributs de *hcl*.
Impact h-méthodes :	▪ Désactiver l'ensemble des hyperméthodes de *hcl*.

4. Chemin d'accès

4.1. Créer un chemin d'accès

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ La classe *cl* ; ▪ G_{hcl}, le graphe de navigation de *hcl* ; ▪ L'ensemble des chemins d'accès des classes de *hcl* dans G_{hcl} ; ▪ L'ensemble des nœuds et des arcs du nouveau chemin d'accès *chacc* à *cl.*
Pré-cond. :	▪ *chacc* est valide ; ▪ $G_{hcl} \cup chacc$ est valide.
Post-act. Modèle :	▪ Créer les arcs de *chacc* ; ▪ Mettre à jour G_{hcl}.
Impact h-méthodes :	▪ Cette opération peut avoir un impact sur le résultat des hyperméthodes ayant *cl* ou ses attributs dans leurs contextes. Selon l'option de nommage pour les hyperattributs basés sur des attributs de *cl*, un nouveau chemin d'accès peut permettre d'atteindre de nouveaux objets de *cl.*

Exemples de création de chemin d'accès

Considérons l'hyperclasse *hcl_Résultats_Étudiant*. Nous créons les chemins d'accès à chacune des classes de l'hyperclasse à partir de la classe racine *Étudiant* :

1. Le chemin d'accès à *Personne* est {{*Étudiant*, *Personne*}, {(*Étudiant*, *Personne*)}} : il est constitué de l'arc (*Étudiant*, *Personne*) ;
2. Le chemin d'accès à *Projet* est {{*Étudiant*, *Projet*}, {(*Étudiant*, *Projet*)}} ;
3. Le chemin d'accès à *ÉtudiantDiplômé* est {{*Étudiant*, *ÉtudiantDiplômé*}, {(*Étudiant*, *ÉtudiantDiplômé*)}} ;
4. Le chemin d'accès à *Inscription* est {{*Étudiant*, *Inscription*}, {(*Étudiant*, *Inscription*)}} ;
5. Le chemin d'accès à Module est {{*Étudiant*, *Inscription*, *Module*}, {(*Étudiant*, *Inscription*), (*Inscription*, *Module*)}} ;

Les cinq chemins d'accès, représentés dans la Figure 58, sont valides et cohérents entre eux. Leur union forme le graphe de navigation de *hcl_Résultats_Étudiant*.

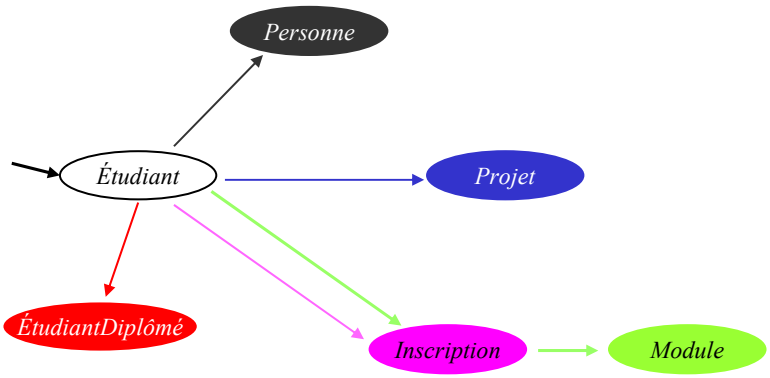

Figure 58 : Les chemins d'accès aux classes de *hcl_Résultats_Étudiant*

4.2. Supprimer un chemin d'accès

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ G_{hcl}, le graphe de navigation de *hcl* ; ▪ L'ensemble des chemins d'accès des classes de *hcl* dans G_{hcl} ; ▪ Le chemin d'accès *chacc* à la classe *cl* à supprimer.
Pré-cond. :	▪ *cl* admet au moins un autre chemin d'accès dans *hcl* ; ▪ G_{hcl}–*chacc* est valide.
Post-act. *Modèle :*	▪ Supprimer les arcs de *chacc* ; ▪ Mettre à jour G_{hcl} ; ▪ Supprimer les hyperattributs de *hcl* basés sur *chacc*.
Impact *h-méthodes :*	▪ Inactiver toute hyperméthode de *hcl* utilisant un hyperattribut supprimé comme opérande ; ▪ La suppression d'un chemin d'accès peut modifier l'ensemble d'objets de *cl* atteints dans *hcl,* en particulier par les hyperattributs implicites. La conservation de résultat ne peut pas être garantie pour les hyperméthodes impliquant *cl* ou des attributs de *cl*.

Exemple de suppression de chemin d'accès

Dans l'hyperclasse *hcl_Participation_Module* (Figure 59),la classe *Enseignant* dispose de deux chemins d'accès : l'un passant par *Enseignement* ($G^1_{Enseignant,\ hcl\_Participation\_Module}$) et l'autre par *ResponsableModule* ($G^2_{Enseignant,\ hcl\_Participation\_Module}$) (Figure 60).

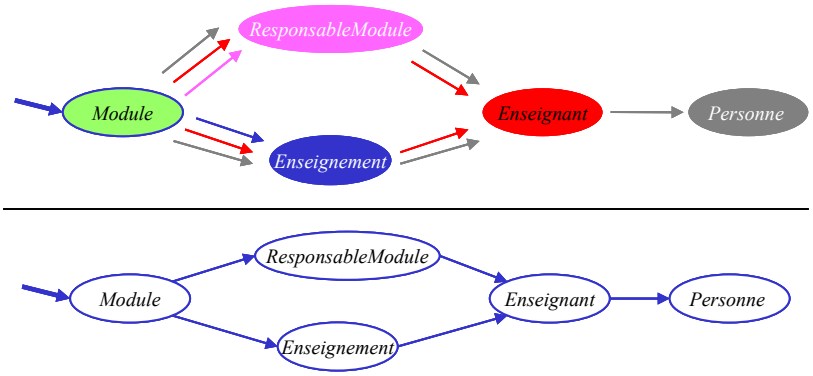

Figure 59 : Graphe de navigation initial de *hcl_Participation_Module*

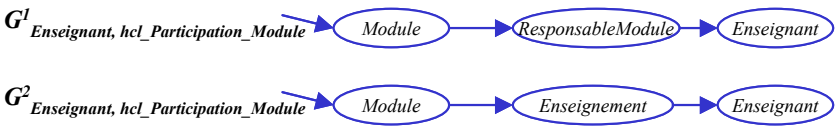

Figure 60 : Les deux chemins d'accès à *Enseignant* dans *hcl_Participation_Module*

La suppression de l'un des ces deux chemins d'accès est possible. La suppression de $G^2_{Enseignant,\ hcl\_Participation\_Module}$ ne rend pas $G_{hcl\_Participation\_Module}$ invalide (Figure 61).

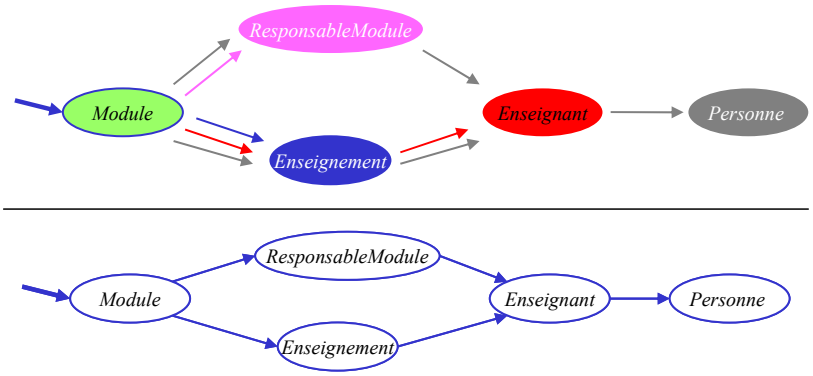

Figure 61 : Nouveau graphe de navigation de *hcl_Participation_Module*

4.3. Inclure un nœud dans un chemin d'accès

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ G_{hcl}, le graphe de navigation de *hcl* ; ▪ Le chemin d'accès *chacc* à la classe *cl* ; ▪ Le nœud n_j à ajouter à *chacc* ; ▪ Au moins un arc (n_i, n_j) à ajouter à *chacc* (pour permettre d'atteindre dans *chacc*).
Pré-cond. :	▪ *chacc* augmenté de l'arc (n_i, n_j) est valide ; ▪ G_{hcl} augmenté de l'arc (n_i, n_j) est valide.
Post-act. Modèle :	▪ Ajouter le nœud n_j à à *chacc* ; ▪ Ajouter l'arc (n_j, n_i) à *chacc* ; ▪ Mettre à jour G_{hcl}.
Impact h-méthodes :	▪ La classe *cl* continue à être atteinte dans l'hyperclasse, la conservation de méthodes peut être garantie. ▪ L'évolution d'un chemin d'accès peut modifier l'ensemble d'objets de *cl* atteints dans *hcl*. La conservation de résultat ne peut pas être garantie pour les hyperméthodes impliquant *cl*.

4.4. Exclure un nœud d'un chemin d'accès

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ G_{hcl}, le graphe de navigation de *hcl* ; ▪ Le chemin d'accès *chacc* à la classe *cl* ; ▪ Le nœud n_i à supprimer de *chacc.*
Pré-cond. :	▪ n_i n'est pas le nœud correspondant à la classe *cl*.
Post-act. Modèle :	▪ Supprimer de *chacc* tout arc ayant n_i comme extrémité initiale ou terminale ; ▪ Pour tout nœud n_j de *chacc* initialement successeur de n_i qui n'est plus atteint dans *chacc*, supprimer n_j de *chacc* : ▪ Exécuter récursivement l'opération ; ▪ Si le nœud correspondant à la classe *cl* devait être supprimé, supprimer *chacc*. Si *chacc* est le seul chemin d'accès à *cl*, annuler l'opération dans sa totalité.
Impact h-méthodes :	▪ La classe *cl* continue à être atteinte dans l'hyperclasse, la conservation de méthodes peut être garantie. ▪ L'évolution d'un chemin d'accès peut modifier l'ensemble d'objets de *cl* atteints dans *hcl*. La conservation de résultat ne peut pas être garantie pour les hyperméthodes impliquant *cl*.

5. Arc de chemin d'accès

5.1. Ajouter un arc à un chemin d'accès

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ G_{hcl}, le graphe de navigation de *hcl* ; ▪ Le chemin d'accès *chacc* à la classe *cl* ; ▪ L'arc (n_i, n_j) à ajouter à *chacc*.
Pré-cond. :	▪ Les nœuds n_i et n_j font partie de G_{hcl} ; ▪ L'arc (n_j, n_i) ne fait pas partie de G_{hcl} ; ▪ *chacc* augmenté de l'arc (n_i, n_j) est valide ; ▪ G_{hcl} augmenté de l'arc (n_i, n_j) est valide ;
Post-act. Modèle :	▪ Ajouter l'arc (n_j, n_i) à *chacc* ; ▪ Mettre à jour G_{hcl}.
Impact h-méthodes :	▪ La classe *cl* continue à être atteinte dans l'hyperclasse, la conservation de méthodes peut être garantie. ▪ L'évolution d'un chemin d'accès peut modifier l'ensemble d'objets de *cl* atteints dans *hcl*. La conservation de résultat ne peut pas être garantie pour les hyperméthodes impliquant *cl*.

5.3. Supprimer un arc d'un chemin d'accès

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ G_{hcl}, le graphe de navigation de *hcl* ; ▪ Le chemin d'accès *chacc* à la classe *cl* ; ▪ L'arc (n_i, n_j) à supprimer de *chacc*.
Pré-cond. :	▪ *chacc* privé de (n_i, n_j) reste valide ; ▪ si *chacc* est le seul chemin d'accès contenant l'arc (n_i, n_j), G_{hcl} privé de l'arc (n_j, n_i) reste valide.
Post-act. Modèle :	▪ Supprimer (n_i, n_j) de *chacc*.
Impact h-méthodes :	▪ La classe *cl* continue à être atteinte dans l'hyperclasse, la conservation de méthodes peut être garantie. ▪ L'évolution d'un chemin d'accès peut modifier l'ensemble d'objets de *cl* atteints dans *hcl*. La conservation de résultat ne peut pas être garantie pour les hyperméthodes impliquant *cl*.

Exemples d'évolution de chemin d'accès

Dans l'hyperclasse *hcl_Participation_Module*, la classe *Enseignant* ne dispose plus que d'une seul chemin d'accès passant par *Enseignement* ($G^{l}_{Enseignant,\ hcl\_Participation\_Module}$) (Figure 62).

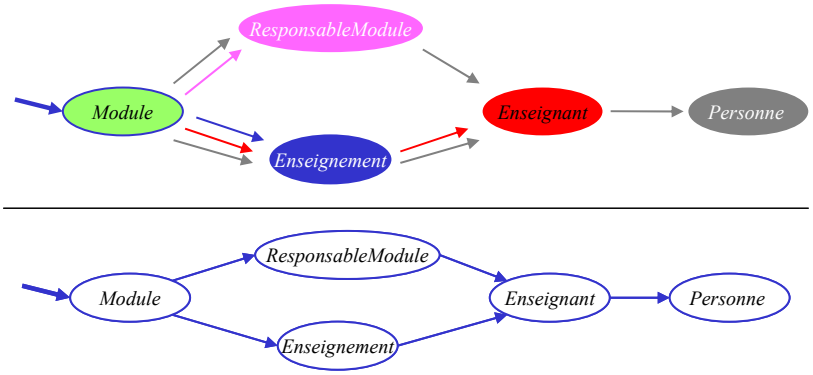

Figure 62 : Nouveau graphe de navigation de *hcl_Participation_Module*

Il est possible d'ajouter le nœud *ResponsableModule* et l'ensemble d'arcs {*(Module, ResponsableModule)*, *(ResponsableModule, Enseignant)*} à $G^{l}_{Enseignant,\ hcl\_Participation\_Module}$ (Figure 63) tout en maintenant sa validité et celle de $G_{hcl\_Participation\_Module}$ (Figure 64).

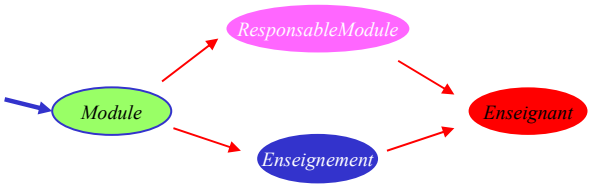

Figure 63 : $G^{l}_{Enseignant,\ hcl\_Participation\_Module}$ après l'ajout des arcs {*(Module, ResponsableModule)*, *(ResponsableModule, Enseignant)*

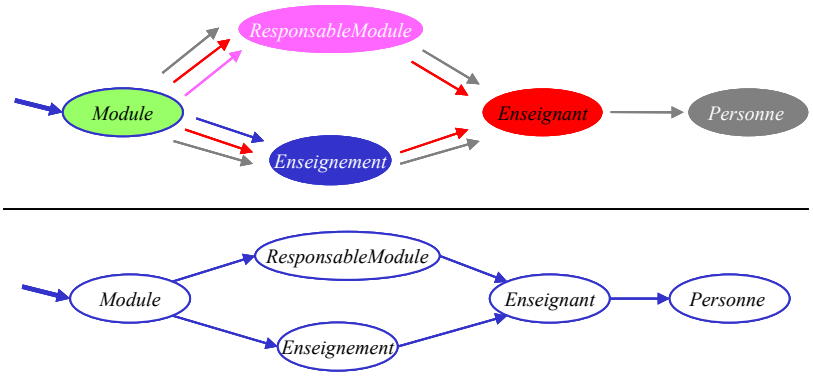

Figure 64 : Graphe de navigation de *hcl_Participation_Module*

La suppression du nœud *ReponsableModule* de $G^1_{Enseignant,\ hcl\_Participation\_Module}$ entraîne la suppression des arcs (*Module*, *ReponsableModule*) et (*ReponsableModule*, *Enseignant*). *Enseignant* se trouve alors isolé et le chemin d'accès est supprimé dans sa totalité. $G^2_{Enseignant,\ hcl\_Participation\_Module}$ devient le seul chemin d'accès à *Enseignant*.

Considérons le cas de *Personne*. $G^1_{Personne,\ hcl\_Participation\_Module}$ (Figure 65) est le seul chemin d'accès dans *hcl_Participation_Module* à *Personne*.

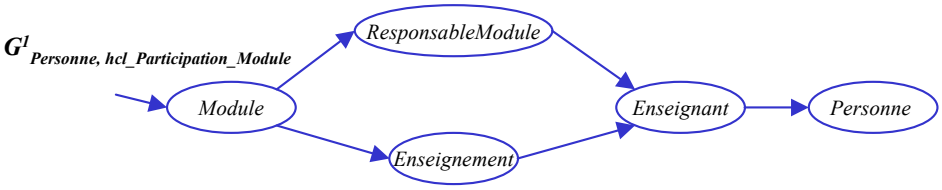

Figure 65 : Le chemin d'accès à *Personne* dans *hcl_Participation_Module*

La suppression du nœud *ReponsableModule* de $G^1_{Personne,\ hcl\_Participation\_Module}$ entraîne la suppression des arcs *(Module, ReponsableModule)* et *(ReponsableModule, Enseignant)*. Cependant *Personne* reste accessible via le chemin [*Module* →*Enseignement* →*Enseignant* →*Personne*] (Figure 66).

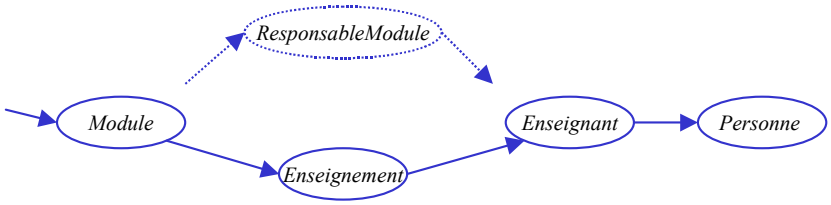

Figure 66 : Chemin d'accès à *Personne* après la suppression de *ResponsableModule*

6. Hyperattribut

6.1. Créer un hyperattribut

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ *cl.att*, l'attribut de la classe *cl* de *hcl* ; ▪ *chacc*, un chemin d'accès à *cl* dans *hcl*.
Pré-cond. :	▪ *chacc* est valide.
Post-act. *Modèle :*	▪ Créer l'hyperattribut *cl.att/chacc* dans *hcl*.
Impact *h-méthodes :*	▪ Si des hyperattributs implicites basés sur *cl.att* ou *att* sont utilisés dans une hyperméthode *hm* de *hcl*, la conservation de résultat n'est pas garantie : la création de l'hyperattribut *cl.att/chacc* peut permettre d'atteindre de nouveaux objets de *cl* ou de classes ayant *att* comme attribut.

Exemple de création d'hyperattribut

Un étudiant accède à ses résultats via l'hyperclasse *hcl_Résultats_Étudiant* (Figure 67). Il a donc, entre autres, accès aux attributs *noteModule* d'*Inscription*, *mentionObtenue*, *moyenneCours*, *moyenneMem* et *recommandationThèse* d'*ÉtudiantDiplômé*. Les hyperattributs *Inscription.noteModule*$_{/1}$, *ÉtudiantDiplômé.mentionObtenue*$_{/1}$, *ÉtudiantDiplômé.moyenneCours*$_{/1}$, *ÉtudiantDiplômé.moyenneMem*$_{/1}$ et *ÉtudiantDiplômé.recommandationThèse*$_{/1}$ sont alors créés dans *hcl_Résultats_Étudiant*.

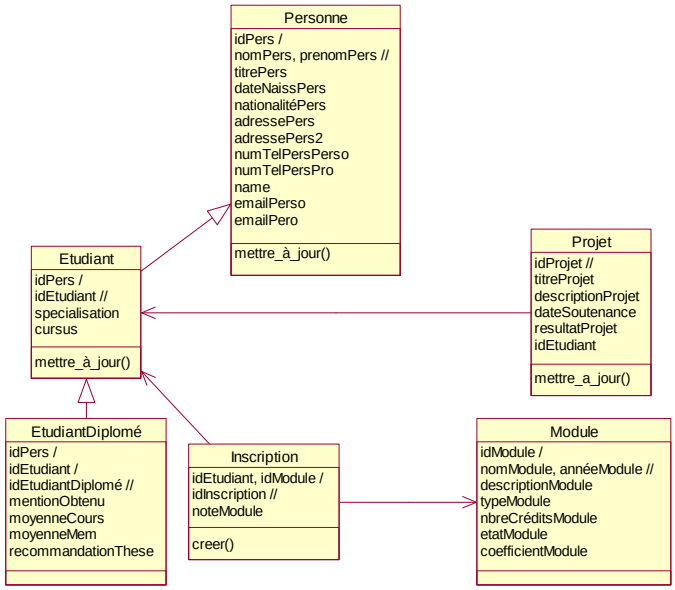

Figure 67 : Diagramme de *hcl_Résultats_Étudiant*

6.2. Supprimer un hyperattribut

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ $cl.att_{/i}$, l'hyperattribut de *hcl* à supprimer.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer l'hyperattribut $cl.att_{/i}$ de *hcl*.
Impact h-méthodes :	▪ Si *chacc* est le seul chemin d'accès à *cl* dans *hcl*, l'hyperattribut implicite *cl.att* devient invalide dans les hyperméthodes de *hcl*. Il est nécessaire d'inactiver toute hyperméthode de *hcl* invoquant *cl.att* ; ▪ Si *chacc* est le seul chemin d'accès à *cl* dans *hcl* et si *cl* est la seule classe de *hcl* contenant l'attribut att, l'hyperattribut implicite *att* devient invalide dans les hyperméthodes de *hcl*. Il est nécessaire d'inactiver toute hyperméthode de *hcl* invoquant *att* ; ▪ Si des hyperattributs implicites basés sur *cl.att* ou *att* sont utilisés dans une hyperméthode *hm* de *hcl*, la conservation de résultat n'est pas garantie : la suppression de l'hyperattribut *cl.att/chacc* peut réduire l'ensemble d'objets de *cl* ou de classes ayant *att* comme attribut atteints.

7. Méthode de classe dans l'hyperclasse

7.1. Rendre visible / invisible une méthode de classe

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ *md*, une méthode de la classe *cl*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Suivant l'état initial de *md*, la rendre visible ou invisible dans *hcl*.
Impact h-méthodes :	▪ Si *md* fait partie d'une hyperméthode *hm* et si *md* est rendue invisible, inactiver *hm*.
Remarques :	▪ Les méthodes de *cl* sont invisibles dans *hcl* jusqu'à ce qu'elles soient explicitement indiquées comme visibles.

Exemples d'activation / inactivation de méthode de classe dans une hyperclasses

Supposons que l'hyperclasse *hcl_Résultats_Étudiant* (Figure 68) soit l'espace informationnel permettant aux étudiants de s'inscrire aux modules et de consulter leurs résultats.

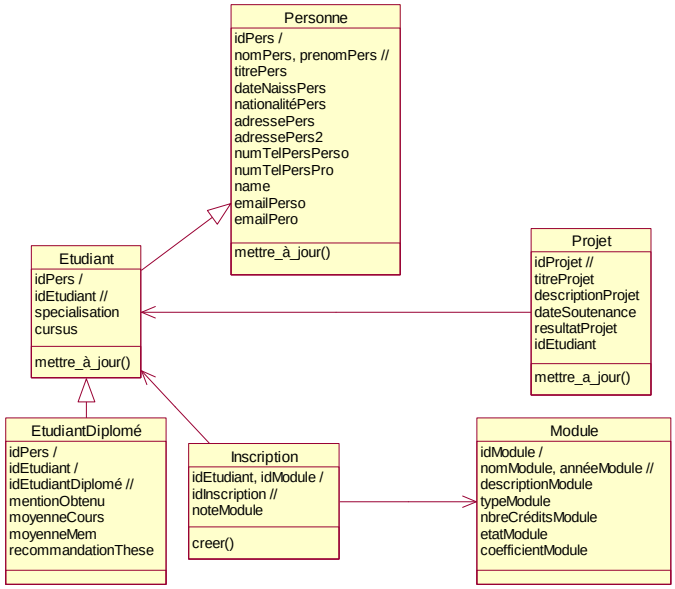

Figure 68 : Diagramme initial de *hcl_Résultats_Étudiant*

L'étudiant est responsable de ses informations personnelles et de la description de son projet de recherche. Il a besoin de pouvoir mettre à jour les données qui le concernent dans les classes *Personne, Étudiant* et *Projet* : les méthodes *Personne.mettre_à_jour()*, *Étudiant.mettre_à_jour()* et *Projet.mettre_à_jour()* sont alors visibles dans *hcl_Résultats_Étudiant*.

L'inscription d'un étudiant à un module nécessite la création d'un nouvel objet de la classe *Inscription* ; la méthode *Inscription.créer()* est alors visible dans *hcl_Résultats_Étudiant*.

8. Hyperméthode

8.1. Créer une hyperméthode

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ Nom de l'hyperméthode *hm* ; ▪ Le traitement associé à *hm*.
Pré-cond. :	▪ Le nom de *hm* n'est pas déjà utilisé par une autre hyperméthode de *hcl*.
Post-act. *Modèle :*	▪ Aucune.
Impact *h-méthodes :*	▪ Créer l'hyperméthode *hm*.

8.2. Supprimer une hyperméthode

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ L'hyperméthode *hm* ;
Pré-cond. :	▪ Aucune.
Post-act. *Modèle :*	▪ Aucune.
Impact *h-méthodes :*	▪ Supprimer *hm*.

8.3. Modifier une hyperméthode

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ L'hyperméthode *hm* ; ▪ Le nouveau traitement associé à *hm*.
Pré-cond. :	▪ Aucune.
Post-act. *Modèle :*	▪ Mettre-à-jour le traitement associé à *hm*.
Impact *h-méthodes :*	▪ Modifier *hm* e.

8.4. Activer / désactiver une hyperméthode

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ L'hyperméthode *hm* ; ▪ Le nouvel état de *hm* ;
Pré-cond. :	▪ Aucune.
Post-act. *Modèle :*	▪ Aucune.
Impact *h-méthodes :*	▪ Selon état initial de *hm*, l'activer ou la désactiver.

8.5. Renommer une hyperméthode

Paramètres :	▪ L'hyperclasse *hcl ;* ▪ L'hyperméthode *hm* ; ▪ Le nouveau nom de l'hyperméthode.
Pré-cond. :	▪ Le nouveau nom de l'hyperméthode n'est pas déjà utilisé par une autre hyperméthode de *hcl*.
Post-act. *Modèle :*	▪ Aucune.
Impact *h-méthodes :*	▪ Renommer *hm*.

1.2. ÉVOLUTION DU DIAGRAMME DE CLASSES

Le méta-modèle d'hyperclasse est une extension du méta-modèle *binex*. Étant donné que les liens qui y sont employés sont existentiels, il est nécessaire de revoir les opérations d'évolution de *binex* pour prendre en compte son extension.

Les zones de dépendance de plusieurs classes du méta-modèle *binex* ont évolué lors de cette extension, en particulier les classes (i) *Méthode*, (ii) *Lien Existentiel,* (iii) *Attribut-Classe* (iv) *Classe* et (v) *Diagramme* auxquels des éléments du méta-modèle d'hyperclasse sont directement liés. Nous listons dans la suite les actions supplémentaires à entreprendre en cas de suppression d'un objet de l'une des classes (i) *Diagramme,* (ii) *Méthode*, (iii) *Attribut-Classe,* (iv) *Lien Existentiel* et (v) *Classe* et du méta-modèle *binex.*

La création de nouveaux objets dans l'une de ses classes ou la mise-à-jour de leurs objets ne perturbent pas les éléments définis dans le méta-modèle d'hyperclasse. Toutefois, les opérations de suppression peuvent avoir une répercussion sur les hyperclasses déjà définies ; chacune de ces opérations peut être à l'origine de la modification ou de suppression de chemins d'accès, de l'exclusion de classes, ou de la désactivation d'hyperméthodes. Il est possible de limiter l'impact d'une évolution de digramme de classes sur ses hyperclasses, notamment en le consolidant. Pour cela, nous définissons dans la suite plusieurs situations remarquables pour la consolidation d'un diagramme dans le cas d'une suppression de classe.

1.2.1. Opérations sur un diagramme de classes

1.2.1.1. Supprimer un diagramme de classes

Paramètres :	▪ Nom du diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer toutes les hyperclasses de *D* ; ▪ Supprimer toutes les classes de *D* ; ▪ Supprimer les attributs de *D* ; ▪ Supprimer les domaines de *D* ; ▪ Supprimer tous les types de base de *D ;* ▪ Supprimer *D*.
Post-act. Objets :	▪ Supprimer tous les objets de *D*.

1.2.1.2. Supprimer une méthode

Paramètres :	▪ La méthode *mth* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer tous les objets d'*Attribut-Méthode-Classe* reliés à *mth* ; ▪ Dissocier *mth* de toute hyperclasse *hcl* dans laquelle *mth* est visible : supprimer tous les objets de *MéthodeDeClasseDansHyperclasse* reliés à *mth* ; ▪ Supprimer *mth*.
Post-act. Objets :	▪ Aucune.

1.2.1.3. Supprimer l'occurrence d'un attribut dans une classe

Paramètres :	▪ *cl.att*, l'occurrence de l'attribut *att* dans la classe *cl* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ L'attribut *att* n'appartient à aucun identifiant de *cl* ;
Post-act. Modèle :	▪ Dissocier *att* de *cl* : supprimer l'objet correspondant d'*Attribut-Classe* ; ▪ Supprimer tout hyperattribut construit sur *cl.att*.
Post-act. Objets :	▪ Supprimer la valeur prise par *att* de tout objet de *cl*.

1.2.1.4. Supprimer un lien existentiel

Paramètres :	▪ Le lien existentiel *lienex* ; ▪ Le diagramme de classes *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ Supprimer tout arc d'un d'accès construit sur *linex* ; ▪ Supprimer *lienex* de *D*.
Post-act. Objets :	▪ Aucune.

1.2.1.5. Supprimer une classe

Paramètres :	▪ La classe *cl*.
Pré-cond. :	▪ Aucune.
Post-act. Modèle :	▪ si *cl* est l'extrémité initiale d'un lien existentiel *lex*, alors supprimer *lex* ; ▪ si *cl* est l'extrémité terminale d'un lien existentiel *lex*, alors supprimer *lex* ; ▪ si *cl* a un identifiants *id*, alors supprimer l'identifiant *id* ; ▪ supprimer l'occurrence *cl.att* de tout attribut att dans *cl* ; ▪ supprimer toute méthode de *cl* ; ▪ supprimer tout nœud d'hyperclasse défini sur *cl*.
Post-act. Objets :	▪ Supprimer les objets appartenant à la fermeture de chacun des objets de *cl* dans la zone de dépendance de *cl* ; ▪ Supprimer tous les objets de *cl*.

La suppression de la classe *cl* entraîne : (i) l'exclusion de *cl* de toutes les hyperclasses, (ii) la suppression de tout hyperattribut définis sur une occurrence d'attribut dans *cl* et (iii) la suppression de tout arc d'un graphe de navigation construit sur un lien existentiel ayant cl comme extrémité initiale ou terminale. Toute méthode de *cl* est supprimée et devient naturellement invisible au niveau des hyperclasses.

1.2.2. Consolidation de diagramme de classes dans le cas d'une suppression de classe

La suppression d'une classe *cl* du diagramme de classes entraîne, entre autre, l'exclusion de *cl* de toute hyperclasse *hcl* dont elle fait partie. L'exclusion de *cl* de *hcl* peut entraîner la suppression de plusieurs chemins d'accès dans *hcl*, l'exclusion de classes autres que *cl* et par conséquent la réduction de l'espace informationnel de l'hyperclasse et l'inactivation de plusieurs de ses hyperméthodes. Dans *hcl*, toute classe successeur d'une classe exclue qui n'est plus atteinte au niveau de l'hyperclasse est à son tour exclue : elle subit une exclusion collatérale.

Pour réduire la perte d'information au niveau des hyperclasses et l'impact de la suppression d'une classe sur le fonctionnement et le résultat de leurs hyperméthodes, nous proposons de consolider le diagramme de classes et les hyperclasses de manière à maintenir l'accès au maximum de leurs classes. La consolidation consiste à reconstituer des liens dans le diagramme de classes et les chemins d'accès dans les hyperclasses pour continuer à atteindre le maximum de classes.

Pour une classe et une hyerclasse données, l'opération de consolidation se base sur la sauvegarde de la connexité de l'hyperclasse. Elle n'est pas intéressante dans les cas où la classe exclue de l'hyperclasse est une classe puits (n'ayant pas de successeur dans le graphe de navigation de l'hyperclasse).

Consolidation de diagramme de classes dans le cas d'une suppression de classe

Paramètres :	▪ Le diagramme de classes *D* ; ▪ La classe *cl* à supprimer de D.
Pré-cond. :	▪ Aucune.
Post-act. Modél. :	▪ Dans chaque hyperclasse *hcl* contenant *cl*, considérer localement *cl* : pour chaque paire de nœud directement prédécesseur et de nœuds directement successeur de *cl*, exécuter la sous-opération correspondante.
Post-act. Objets :	▪ Aucune.
Impact h-méthodes :	▪ Il est nécessaire de revisiter les hyperméthodes de *hcl* dont les éléments de leurs contextes ont été supprimés ; ▪ Les hyperméthodes dont le contexte n'est pas modifié bénéficient de la conservation d'hyperméthode et de résultat. En effet, la consolidation est définie de manière à ce que les mêmes objets continuent à être atteints.

Il s'agit de considérer localement la classe à supprimer de *D* dans le contexte de chaque hyperclasse *hcl* à laquelle elle appartient. Si *cl* appartient à plusieurs hyperclasses de *D*, il convient de coordonner les actions à entreprendre sur *D* et chacune de ces hyperclasses.

Dans l'exemple de la Figure 69, la classe *cl* a plusieurs prédécesseurs et plusieurs successeurs dans *hcl*. La consolidation consiste à préserver les liens qui existe entre le nœud directement prédécesseurs et le nœud directement successeurs de *cl* dans *hcl*, et qui seraient mis en péril par l'exclusion de *cl*. Quatre cas de figure peuvent se présenter.

Chaque paire de nœuds directement prédécesseur et directement successeur est examinée. Dans le cas de *cl*, il faut étudier les liens entre les paires $(Pred_1, Succ_1)$, ... $(Pred_1, Succ_m)$, $(Pred_2, Succ_1)$, ... $(Pred_2, Succ_m)$ ... $(Pred_n, Succ_1)$... et $(Pred_n, Succ_m)$.

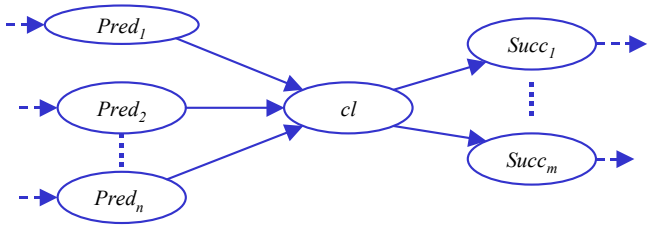

Figure 69 : voisinage immédiat dans G_{hcl} de la classe *cl* à supprimer

1er cas (Figure 70)

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ La classe *cl* ; ▪ La paire *(Pred$_i$, Succ$_j$)* de prédécesseur et de successeur de *cl*.
Pré-cond. :	▪ *cl*∈ *ecl* ; ▪ *cl* n'est pas la classe racine de *hcl* ; ▪ *cl* n'est pas une classe puits de *hcl* ; ▪ *Pred$_i$* dépend existentiellement de *cl* ; ▪ *cl* dépend existentiellement de *Succ$_j$* ; ▪ Les arcs *(Pred$_i$, cl)* et *(cl, Succ$_j$)* appartiennent au graphe de navigation de *hcl*.
Post-act. Modél. :	▪ Créer un lien existentiel de *Pred$_i$* à *Succ$_j$;* ▪ Dans chaque chemin d'accès d'une classe de *hcl* passant par *cl* remplacer les arcs *(Pred$_i$, cl)* et *(cl, Succ$_j$)* par l'arc *(Pred$_i$, Succ$_j$)* ; ▪ Supprimer le nœud *cl* du graphe de navigation de *hcl* et des ses chemins d'accès.
Post-act. Objets :	▪ Chaque objet de *Pred$_i$* initialement relié à un objet de *Succ$_j$* via un objet de *cl* est directement relié au même objet de *Succ$_j$* via le lien existentiel *(Pred$_i$, Succ$_j$)*.
Remarques :	▪ Sans la consolidation, si *Succ$_j$* n'était atteinte dans *hcl* que via *cl*, elle aurait été à son tour exclue de *hcl*.

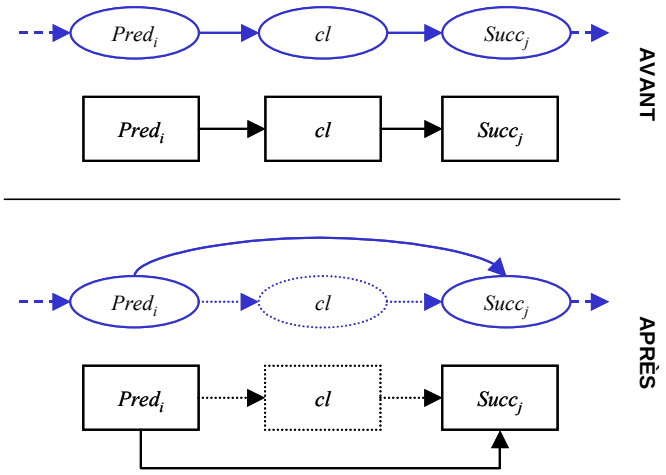

Figure 70 : 1er cas de figure

La Figure 70 représente une partie du voisinage de *cl* avant et après la modification. La situation est décrite par une partie du graphe de navigation de *hcl* (sous-graphe en bleu) et une partie du diagramme de classes (sous-graphe en noir) centrés au tour de *cl*.

2ème cas (Figure 71)

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ La classe *cl* ; ▪ La paire *(Pred$_i$, Succ$_j$)* de prédécesseur et de successeur de *cl*.
Pré-cond. :	▪ *cl*∈ *ecl* ; ▪ *cl* n'est pas la classe racine de *hcl* ; ▪ *cl* n'est pas une classe puits de *hcl* ; ▪ *Succ$_j$* dépend existentiellement de *cl* ; ▪ *cl* dépend existentiellement de *Pred$_i$* ; ▪ Les arcs *(Pred$_i$, cl)* et *(cl, Succ$_j$)* appartiennent au graphe de navigation de *hcl*.
Post-act. Modél. :	▪ Créer un lien existentiel de *Succ$_j$* à *Pred$_i$;* ▪ Dans chaque chemin d'accès d'une classe de *hcl* passant par *cl* remplacer les arcs *(Pred$_i$, cl)* et *(cl, Succ$_j$)* par l'arc *(Pred$_i$, Succ$_j$)* ; ▪ Supprimer le nœud *cl* du graphe de navigation de *hcl* et des ses chemins d'accès.
Post-act. Objets :	▪ Chaque objet de *Succ$_j$* initialement relié à un objet de *Pred$_i$* via un objet de *cl* est directement relié au même objet de *Pred$_i$* via le lien existentiel *(Succ$_j$, Pred$_i$)*.
Remarques :	▪ Sans la consolidation, si *Succ$_j$* n'était atteinte dans *hcl* que via *cl*, elle aurait été à son tour exclue de *hcl*.

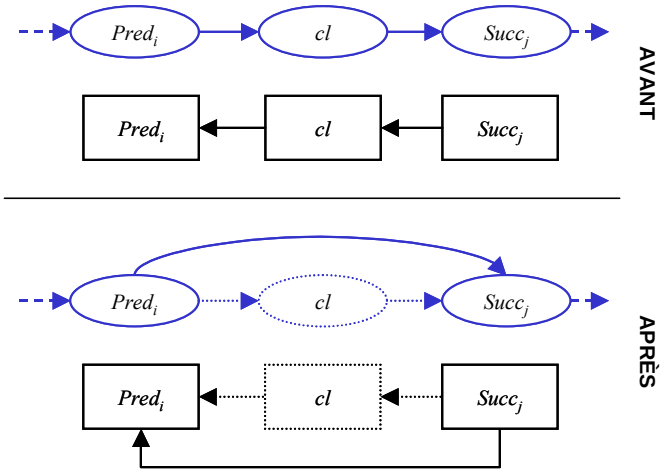

Figure 71 : 2ème cas de figure

3ème cas (Figure 72)

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ La classe *cl* ; ▪ La paire *(Pred$_i$, Succ$_j$)* de prédécesseur et de successeur de *cl*.
Pré-cond. :	▪ *cl*∈ *ecl* ; ▪ *cl* n'est pas la classe racine de *hcl* ; ▪ *cl* n'est pas une classe puits de *hcl* ; ▪ *Pred$_i$* dépend existentiellement de *cl* ; ▪ *Succ$_j$* dépend existentiellement de *cl* ; ▪ Les arcs *(Pred$_i$, cl)* et *(cl, Succ$_j$)* appartiennent au graphe de navigation de *hcl*.
Post-act. Modél. :	▪ Créer une classe *cl'* qui dépend existentiellement de *Pred$_i$* et de *Succ$_j$;* ▪ Inclure *cl'* dans *hcl* ; ▪ Dans chaque chemin d'accès d'une classe de *hcl* passant par *cl* remplacer l'arc *(Pred$_i$, cl)* par l'arc *(Pred$_i$, cl')* et l'arc *(cl, Succ$_j$)* par l'arc *(cl', Succ$_j$)* ; ▪ Supprimer le nœud *cl* du graphe de navigation de *hcl* et des ses chemins d'accès.
Post-act. Objets :	▪ Chaque objet de *Pred$_i$* initialement relié à un objet de *Succ$_j$* via un objet de *cl* est relié au même objet de *Succ$_j$* via un objet de *cl'*.
Remarques :	▪ Sans la consolidation, si *Succ$_j$* n'était atteinte dans *hcl* que via *cl*, elle aurait été à son tour exclue de *hcl*.

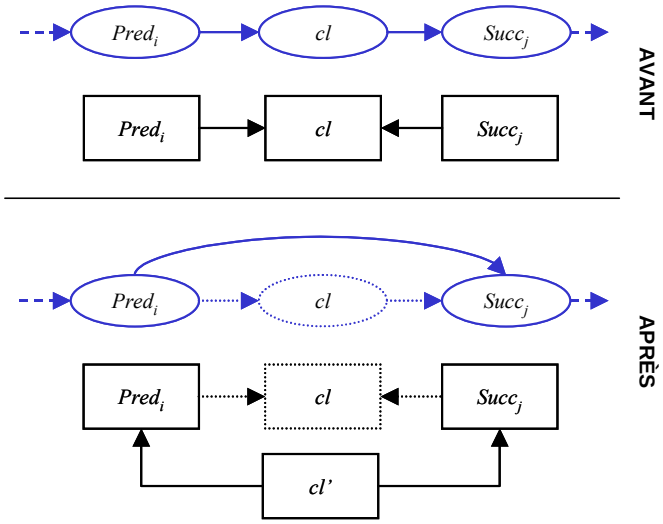

Figure 72 : 3ème cas de figure

4ème cas (Figure 73)

Dans situation décrite par la Figure 73, la consolidation consisterait, au niveau du diagramme de classes, à créer une classe *cl'* qui dépend existentiellement de $Pred_i$ et de $Succ_j$, et au niveau des objets à relier chaque objet de $Pred_i$ initialement relié à un objet de $Succ_j$ via un objet de *cl* au même objet de $Succ_j$ via un objet de *cl'*. Ce traitement revient donc à créer une classe *cl'* quasi-identique à *cl*, c'est à dire, à recréer la classe *cl* à supprimer.

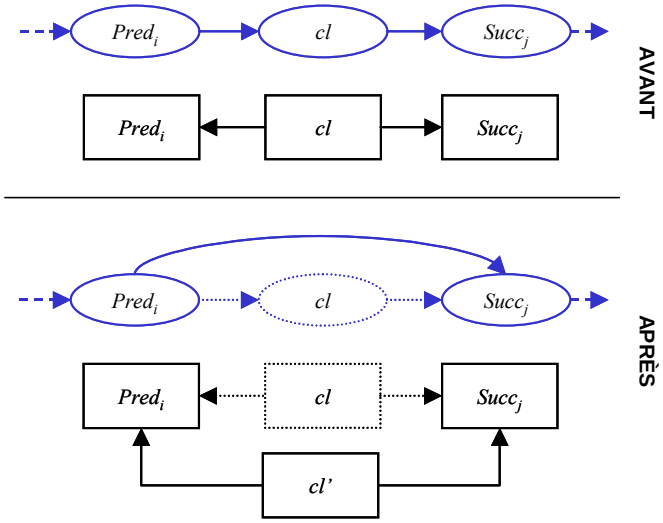

Figure 73 : 4ème cas de figure

Dans une telle situation, nous proposons de ne pas consolider le diagramme de classes. Si $Succ_j$ n'était atteinte dans *hcl* que via *cl*, elle est à son tour exclue de *hcl*.

Exemple de consolidation d'un diagramme de classes

Nous reprenons l'exemple de l'hyperclasse *hcl_Implication_Enseignant* (Figure 39). Nous considérons le scénario dans lequel la disponibilité des salles et l'affectation des salles aux séances de cours ne venait plus à être assurée parle coordinateur de la formation, et où on se contenterait d'indiquer aux étudiants le site (l'institution où se déroulera le cours).

Dans ce cas, la classe *Salle* est exclue des hyperclasses qui l'utilisent.

La suppression de *Salle* entraîne dans *hcl_Implication_Enseignant* l'exclusion d'*Institution*. Si *Institution* n'est plus accessible au niveau de *hcl_Implication_Enseignant*, il n'est plus possible d'exécuter l'hyperméthode *calcul_prime_enseignant()* qui utilise l'attribut *primeHoraire*. La consolidation du diagramme de classes et de l'hyperclasse permet d'éviter le dysfonctionnement de cette hyperméthode.

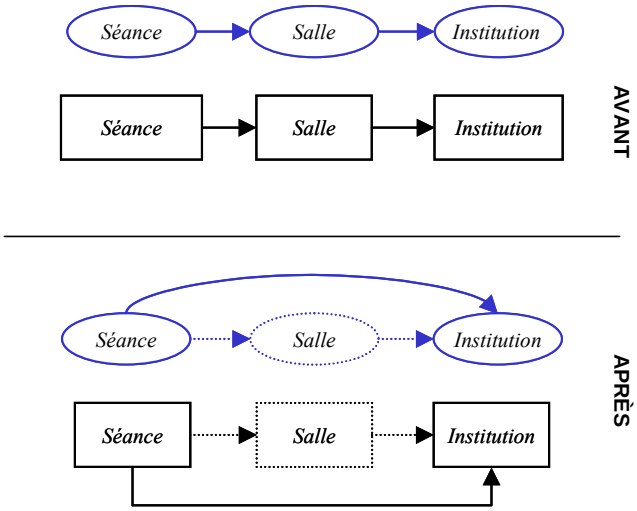

Figure 74 : Consolidation de *hcl_Implication_Enseignant*

calcul_prime_enseignant() continue à atteindre les mêmes objets de Séance et d'Institution, la conservation de résultat peut alors être garantie.

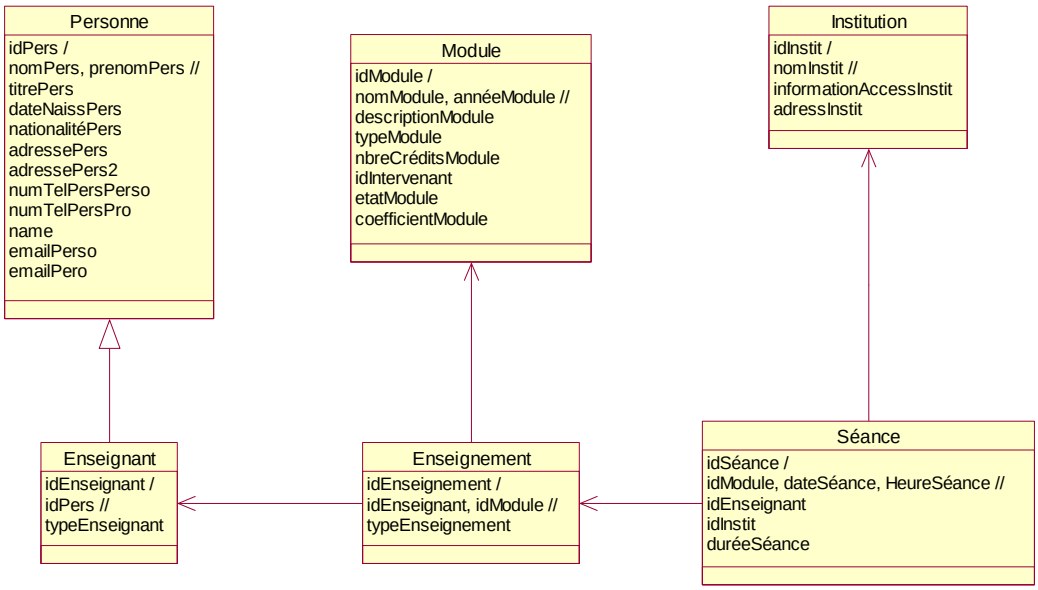

Figure 75 : Nouveau diagramme de *hcl_Implication_Enseignant*

2. EXTRACTION D'HYPERCLASSE

Il s'agit d'extraire des hyperclasses pour constituer des bibliothèques de variantes d'hyperclasses.

2.1. EXTRAIRE UNE HYPERCLASSE

Cette opération consiste à créer un nouveau diagramme de classes à partir d'un diagramme d'hyperclasse définie sur un digramme de classe existant. La complétude de l'hyperclasse garantit qu'elles contiennent tous les éléments qui la définissent lors de son extraction.

L'extraction d'hyperclasse sert à constituer des bibliothèques d'hyperclasses (i) représentant des situations génériques ou remarquables rencontrées et (ii) pouvant être intégrer dans d'autres diagrammes de classes.

Extraire une hyperclasse

Paramètres :	▪ L'hyperclasse *hcl*.
Pré-cond. :	▪ *hcl* est valide, c'est-à-dire, connexe et complète.
Post-act. Modél. :	▪ Créer un nouveau diagramme de classes *D'* ; ▪ Pour tout attribut, type de base, identifiant, classe, lien existentiel ou de spécialisation-généralisation et méthode de classe dans le diagramme de *hcl*, créer un élément identique correspondant dans *D'* ; ▪ Définir une hyperclasse *hcl'* identique à *hcl* sur *D'*.
Post-act. Objets :	▪ Pour chaque objet d'une classe de *hcl visible* dans *hcl*, créer un objet identique dans la classe correspondante dans *D'*.

Exemple d'extraction d'hyperclasse

Dans *M@TIS*, chaque module peut être évalué à son terme par les étudiants qui l'ont suivi. Dans le système actuel, l'évaluation d'un module est anonyme mais réservée aux seuls étudiants qui y sont inscrits. Grâce à l'attribut booléen *étatÉvaluation* dans la classe *Inscription*, un étudiant ne peut évaluer qu'une seule fois le même module.

L'évaluation de module est une situation que nous avons rencontrée lors de plusieurs projets de SI éducatifs, tels que le SI de la formation continue *Interactive-MATIS* ou le SI du département des systèmes d'information *SYINF* de l'*Université de Genève*. La solution retenue pour *M@TIS* peut alors être considérée comme suffisamment générique et être exploitée pour répondre à des situations similaires, d'où l'idée d'extraction de l'hyperclasse *hcl_évaluation_module* du diagramme de classes de *M@TIS* (Figure 76).

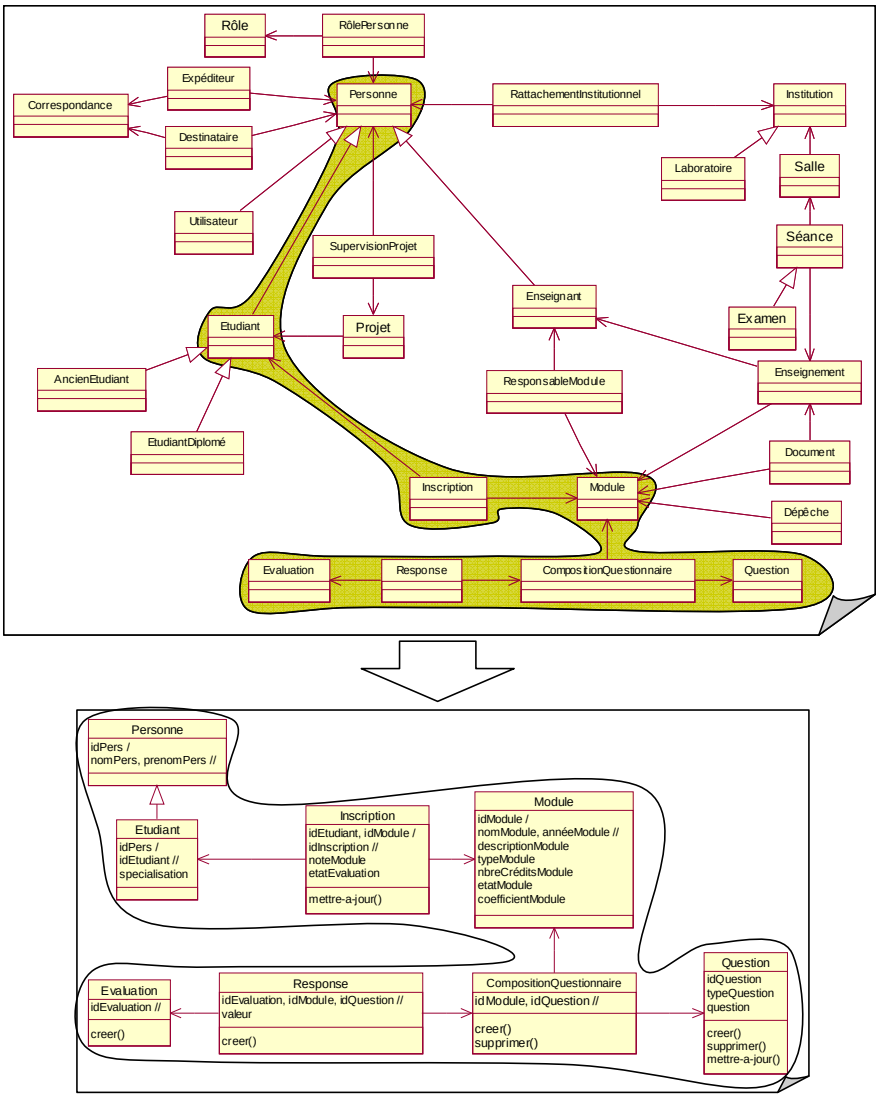

Figure 76 : Extraction de l'hyperclasse *hcl_évaluation_module* du diagramme de classes de *M@TIS*

L'hyperclasse *hcl_évaluation_module* est définie sur les classes *Personne*, *Étudiant*, *Inscription*, *Module*, *CompositionQuestionnaire*, *Question*, *Réponse* et *Évaluation*.

À partir du diagramme de classes de *M@TIS* et de l'hyperclasse *hcl_évaluation_module*, nous construisons un nouveau diagramme de classes qui reprend (i) les classes et (ii) les liens existentiels et de spécialisation-généralisation du diagramme de *hcl_évaluation_module*, ainsi que les attributs et méthodes des classes de *hcl_évaluation_module* qui y sont visibles. Sur ce nouveau digramme, nous construisons une hyperclasse identique à *hcl_évaluation_module*.

2.2. VARIANTE D'HYPERCLASSE

Le concept de *variante* ou de *version d'hyperclasse* est défini formellement dans le cas des hyperclasses fonctionnelles que nous présentons dans le chapitre IV (Hyperclasses fonctionnelles). Nous l'employons ici dans un sens très large.

À partir d'une hyperclasse extraite d'un diagramme de classes, il est possible de dériver une ou plusieurs *variantes* qui sont des évolutions de l'hyperclasse d'origine, obtenues par application des opérations d'évolution du diagramme de classes ou du diagramme de l'hyperclasse.

Voici quelques variantes de l'hyperclasse *hcl_évaluation_module* extraite du diagramme de classes de *M@TIS.*

2.2.1. Variante anonyme

Telle qu'elle a été extraite du diagramme de classes de *M@TIS, hcl_évaluation_module* (Figure 76) permet de gérer les évaluations anonymes des modules.

2.2.2. Variante personnalisée

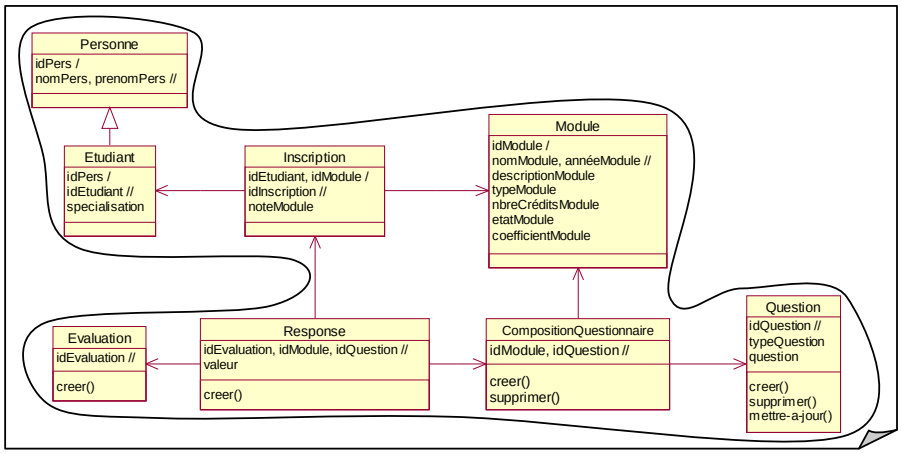

Figure 77 : Évaluation personnalisée

La Figure 77 représente une variante de *hcl_évaluation_module* où il est possible de retrouver l'étudiant qui a rempli une évaluation d'un module et où un étudiant peut évaluer plusieurs fois le même module. Un lien existentiel de *Réponse* à *Inscription* a été rajouté. L'attribut booléen *étatÉvaluation* a été supprimé d'*Inscription*.

2.2.3. Variante élémentaire

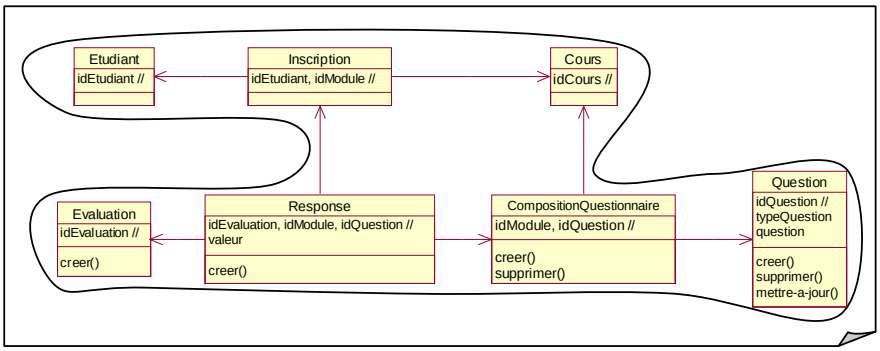

Figure 78 : Variante élémentaire de *hcl_évaluation_module*

La Figure 78 représente une variante élémentaire de *hcl_évaluation_module* : la classe *Personne* et les attributs qui ne sont pas indispensables pour la gestion des évaluations ont été supprimés. Comme pour la variante précédente, un lien existentiel de *Réponse* à *Inscription* a été rajouté, rendant l'évaluation personnalisée.

3. INTÉGRATION D'HYPERCLASSE

Traditionnellement, l'intégration de modèles de SI consiste à prendre en compte deux modèles complets de SI et de les intégrer en un même modèle. L'intégration d'hyperclasse prend deux formes : (i) une hyperclasse peut être intégrée soit dans une autre hyperclasse du même diagramme de classes, pour étendre son espace informationnel ou (ii) dans un diagramme de classes pour étendre le domaine qu'il représente.

L'intégration d'hyperclasse est intéressante du moment qu'en plus du diagramme de l'hyperclasse, il est possible de préserver les hyperméthodes de l'hyperclasse et ses autres propriétés.

3.1. INTÉGRATION D'UNE HYPERCLASSE SECONDAIRE DANS UNE HYPERCLASSE PRIMAIRE

L'intégration est une manière d'étendre et d'enrichir l'espace informationnel d'une hyperclasse en y incluant une ou plusieurs autres hyperclasses.

L'opération, que nous décrivons dans la suite, consiste à fusionner deux hyperclasses, d'un même diagramme de classes, en incluant dans une hyperclasse, dite *primaire*, une hyperclasse *secondaire*. Le résultat obtenu est une hyperclasse avec un espace informationnel étendu. L'hyperclasse obtenue par intégration des deux hyperclasses doit être valide, c'est-à-dire connexe et complète.

Par ailleurs, il est nécessaire de tenir compte que chacune des hyperclasses à intégrer puisse être peuplée (que ses classes contiennent des objets) et avoir des hyperméthodes. L'opération d'intégration doit assurer autant que possible la conservation des hyperobjets, des hyperméthodes et de leurs résultats.

L'hyperclasse résultant de l'intégration d'hyperclasses est construite sur l'ensemble formé par l'union de leurs ensembles respectifs de classes. Chacune des hyperclasses à inclure étant complète, leur union est complète.

La connexité de l'hyperclasse résultant de l'intégration nécessite que les deux hyperclasses aient au moins une classe en commun. Nous nous intéressons au cas où la classe racine de l'hyperclasse secondaire fait partie de l'hyperclasse principale. Plusieurs cas peuvent alors se présenter :

1) la classe racine de l'hyperclasse secondaire est la seule classe commune aux deux hyperclasses ;

2) Les deux hyperclasses ont la même classe racine qui est leur unique classe commune ;

3) La classe racine de l'hyperclasse secondaire n'est pas la seule classe commune aux deux hyperclasses.

3.1.1. La classe racine de l'hyperclasse secondaire est la seule classe commune aux deux hyperclasses

Paramètres :	▪ hcl_1, l'hyperclasse principale ; ▪ hcl_2, l'hyperclasse secondaire.
Pré-cond. :	▪ $(ecl_1 \cap ecl_2) = \{cl_{r2}\}$; ecl_1 et ecl_2 sont respectivement les ensembles des classes de hcl_1 et de hcl_2 ; cl_{r2} est la classe racine de hcl_2.
Post-act. Modél. :	▪ Inclure les classes d'ecl_2 dans ecl_1 : $ecl_1 := (ecl_1 \cup ecl_2)$; ▪ Créer un nouveau nœud correspondant à chacune des classes d'ecl_2 dans le graphe de navigation de hcl_1 ; ▪ Pour chaque chemin d'accès à une classe cl_2 d'ecl_2 dans hcl_2, et pour chaque chemin d'accès à cl_{r2} dans hcl_1, créer un chemin d'accès dans hcl_1 construit par concaténation des deux chemins d'accès : le chemin d'accès de cl_{r1} à cl_{r2} dans hcl_1 et le chemin d'accès de cl_{r2} à cl_2 dans hcl_2 ; ▪ Pour chaque hyperattribut $hatt_2$ de hcl_2, défini sur un attribut *att* d'une classe cl_2 d'ecl_2 et atteint par un chemin d'accès acc_2, définir un hyperattribut dans hcl_1 pour atteindre *att* à travers le ou les nouveaux chemins d'accès construits par concaténation du chemin d'accès à cl_{r2} dans hcl_1 et d'acc_2.
Impact h-méthodes :	▪ cl_{r2}, la classe racine de hcl_2 est la seule classe commune aux deux hyperclasses. L'introduction de nouveaux nœuds et de leurs chemins d'accès ne modifie pas les chemins d'accès initiaux aux classes de hcl_1. La conservation des hyperméthodes de hcl_1 est alors garantie. ▪ Les hyperobjets de hcl_1 se trouvent étendus par l'intégration de hcl_2, mais continuent à atteindre les mêmes objets des classes initiales de hcl_1. Toutefois, de nouveaux hyperattributs issus de hcl_2 sont définis sur hcl_1. La conservation de résultat est garantie seulement pour les hyperméthodes de hcl_1 qui n'utilisent pas d'hyperattributs implicites dont le résultat est modifié par la définition des nouveaux hyperattributs. ▪ L'intégration de hcl_2 dans hcl_1 rend tous les éléments de hcl_2 accessibles dans hcl_1 ; les hyperméthodes de hcl_2 peuvent alors être invoquées au niveau de hcl_1. La conservation de résultat ne peut être garantie pour ces hyperméthodes.
Remarque :	▪ Tout comme l'intégration de classe dans une hyperclasse, l'intégration d'une hyperclasse dans une autre crée une ou plusieurs situations de recouvrement (voir la section 6.2. Interopérabilité des hyperclasses P.95).

Exemple

L'exemple porte sur l'intégration de l'hyperclasse *hcl_participation_module* dans l'hyperclasse *hcl_évaluation_module* dans le diagramme de classes de *M@TIS* (Figure 79).

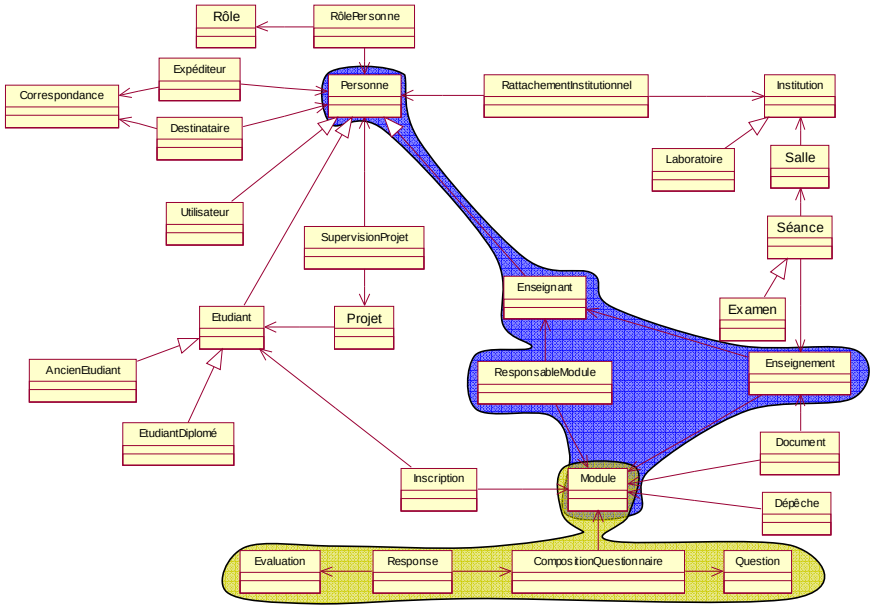

Figure 79 : *hcl_participation_module* et *hcl_évaluation_module* dans le diagramme de classes de *M@TIS*

L'hyperclasse *hcl_évaluation_module* (Figure 80) est définie sur les classes *Évaluation*, *Réponse*, *CompositionQuestionnaire*, *Question* et *Module*. Elle décrit l'espace informationnel lié à la gestion des évaluations des modules par les étudiants. *Évaluation* est sa classe racine.

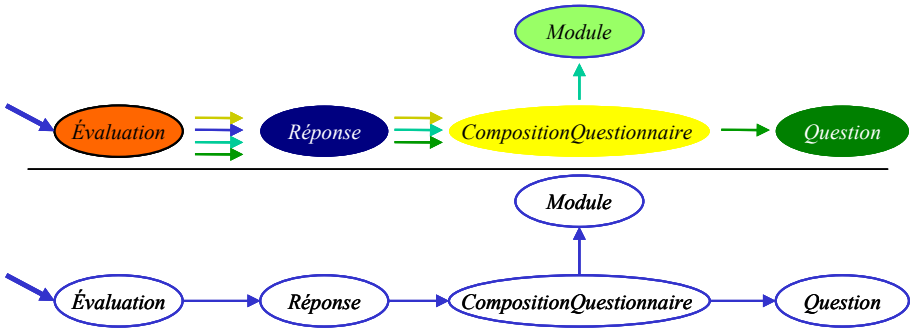

Figure 80 : Chemins d'accès et graphe de navigation de *hcl_évaluation_module*

L'hyperclasse *hcl_participation_module* (Figure 82) est définie sur les classes *Module*, *Enseignement*, *ResponsableModule*, *Enseignant* et *Personne*. Elle décrit la participation des enseignants aux modules de *MATIS*. *Module* est sa classe racine.

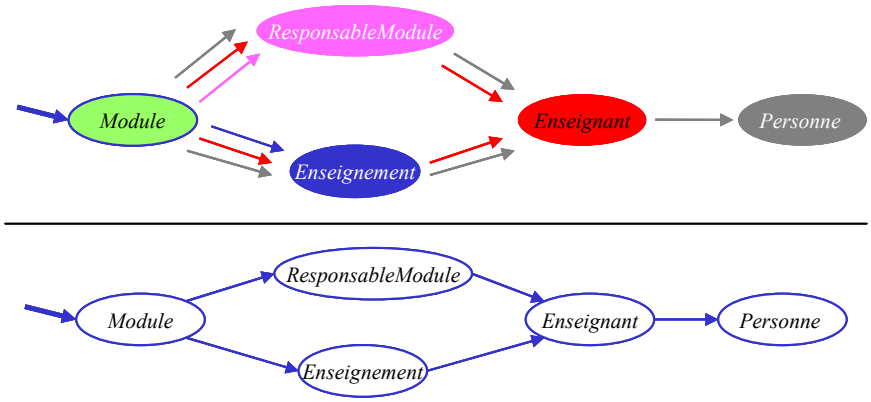

Figure 81 : Chemins d'accès et graphe de navigation de *hcl_participation_module*

L'intégration de *hcl_participation_module* (hyperclasse secondaire) dans *hcl_évaluation_module* (hyperclasse primaire) étend l'espace informationnel de la gestion des évaluations des modules pour y inclure les intervenants : les enseignants et les responsables de modules. La classe module est la seule classe commune aux deux hyperclasses et c'est la classe racine de l'hyperclasse secondaire. L'opération consiste à :

1. Inclure les classes Enseignement, ResponsableModule, Enseignant et Personne dans hcl_évaluation_module ; la classe Module en fait déjà partie ;

2. Créer un nouveau nœud correspondant à chacune de ces classes dans le graphe de navigation de *hcl_évaluation_module* ;

3. Pour chaque chemin d'accès à chacune des classes *Enseignement*, *ResponsableModule*, *Enseignant* et *Personne* dans *hcl_participation_module*, créer un nouveau chemin d'accès issu de la concaténation de ce chemin et du chemin d'accès à Module dans *hcl_évaluation_module* ;

4. Pour chaque hyperattribut $hatt_2$ de *hcl_participation_module*, défini sur un attribut *att* d'une classe cl_2 de *hcl_participation_module* et atteint par un chemin d'accès acc_2, définir un hyperattribut dans *hcl_évaluation_module* pour atteindre *att* à travers le nouveau chemin d'accès construit par concaténation du chemin d'accès à cl_{r2} dans *hcl_évaluation_module* et d'acc_2.

Le résultat de l'intégration de *hcl_participation_module* dans *hcl_évaluation_module* est représenté par la Figure 82.

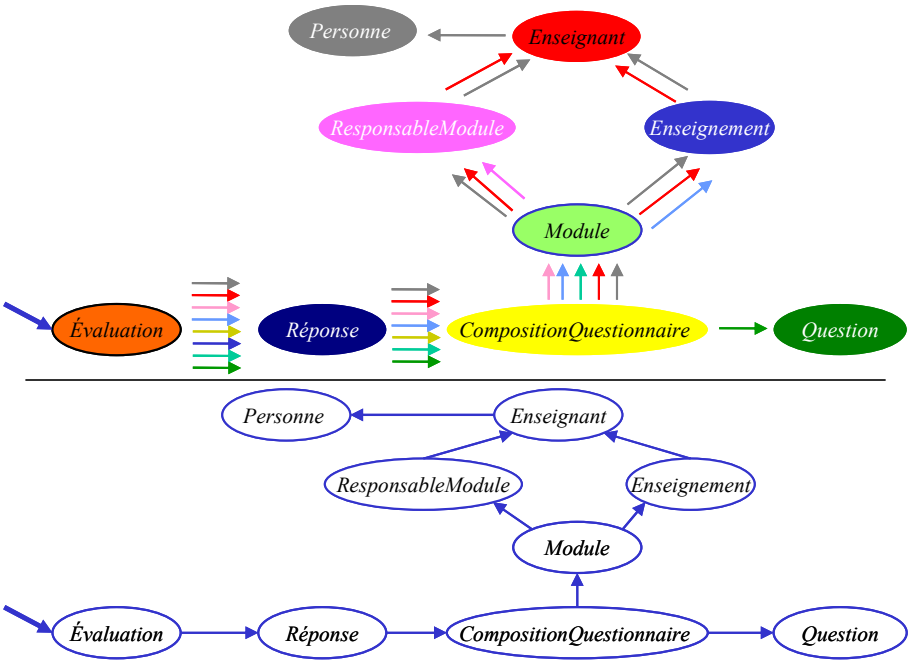

Figure 82 : Chemins d'accès et graphe de navigation de *hcl_évaluation_module* après l'intégration de *hcl_participation_module*

Seule l'hyperclasse *hcl_évaluation_module* est modifiée lors de cette opération ; *hcl_participation_module* ne subit aucune modification (Figure 83).

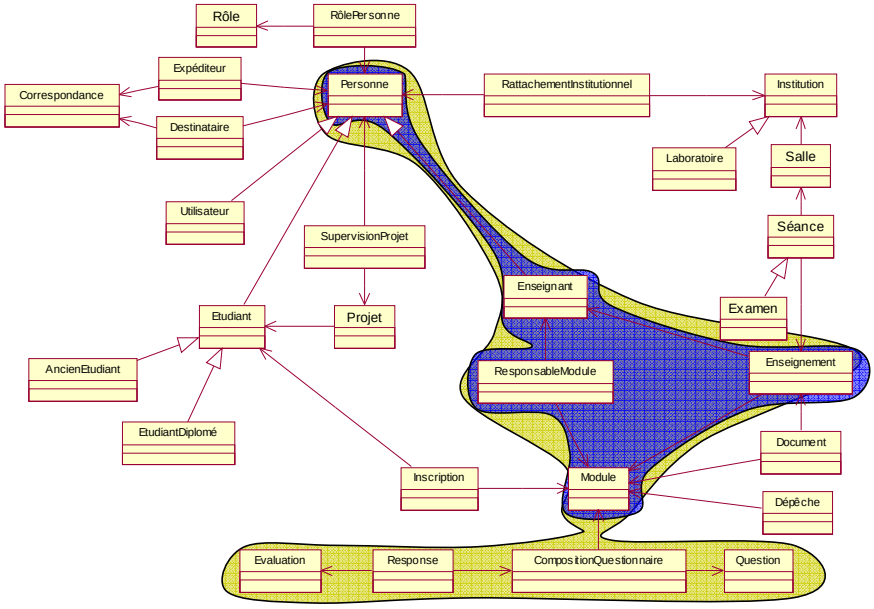

Figure 83 : Résultat de l'intégration de : *hcl_participation_module* dans *hcl_évaluation_module*

L'hyperméthode initialement définie sur *hcl_évaluation_module* pour comptabiliser le résultat des évaluations de module reste exécutable dans *hcl_participation_module*. Toutefois, en raison du changement de classe racine, la conservation des ses résultats ne peut être garantie.

3.1.2. Les deux hyperclasses ont la même classe qui est leur unique classe commune

Dans le cas où les hyperclasses primaire hcl_1 et secondaire hcl_2 ont la même classe racine cl_{r1} et où cette classe est leur seule classe commune, les chemins d'accès aux classes d'ecl_2 dans hcl_1 et les chemins d'accès aux classes d'ecl_2 dans hcl_2 sont identiques : l'opération de concaténation de chemins d'accès définie dans l'opération précédente est transparente car le chemin d'accès à la classe racine de hcl_2 dans hcl_1 ne contient aucun arc.

Paramètres :	▪ hcl_1, l'hyperclasse principale ; ▪ hcl_2, l'hyperclasse secondaire.
Pré-cond. :	▪ $(ecl_1 \cap ecl_2) = \{cl_{r1}\}$; ecl_1 et ecl_2 sont respectivement les ensembles des classes de hcl_1 et de hcl_2 ; cl_{r1} est la classe racine de hcl_1 et de hcl_2.
Post-act. Modél. :	▪ Même actions et remarques que l'opération précédente.
Post-act. Objets :	▪ Aucune.
Impact h-méthodes :	▪ La structure initiale de hcl_1 n'est pas modifiée par l'intégration ; la préservation des hyperméthodes de hcl_1 est garantie ; ▪ Il est possible de récupérer les hyperméthodes de hcl_2 au niveau de hcl_1. ▪ Bien que les chemins d'accès aux classes de hcl_2 ne sont pas modifiés, la conservation des résultats des hyperméthodes ne peut être garantie si des hyperattributs implicites y sont utilisés et si l'opération d'intégration modifie l'ensemble des attributs atteints dans l'hyperclasse primaire.

Exemple

L'exemple porte sur l'intégration de l'hyperclasse *hcl_rattachement* dans l'hyperclasse *hcl_correspondance* dans le diagramme de classes de *M@TIS* (Figure 84).

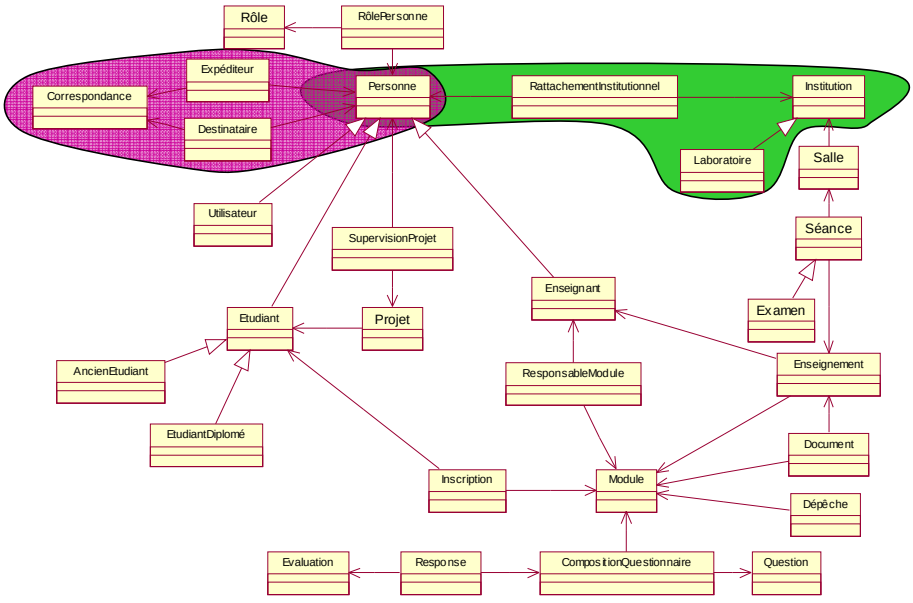

Figure 84 : *hcl_correspondance* et *hcl_rattachement* dans le diagramme de classes de *M@TIS*

L'hyperclasse *hcl_correspondance* (Figure 85) est définie sur les classes *Personne*, *Expéditeur*, *Destinataire* et *Correspondance*. Elle décrit la communication officielle d'une personne avec la direction de la formation *MATIS*. *Personne* est sa classe racine.

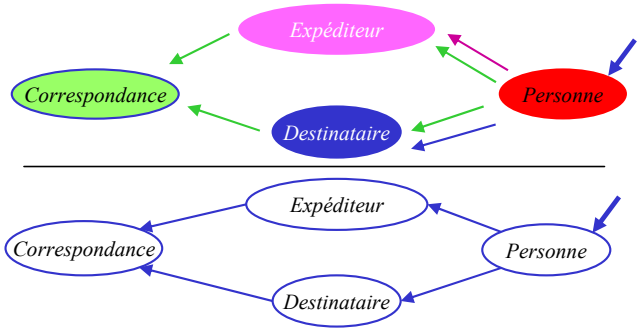

Figure 85 : Chemins d'accès et graphe de navigation de *hcl_correspondance*

L'hyperclasse *hcl_rattachement* (Figure 86) est définie sur les classes *Personne*, *RattachementInstitutionnel*, *Institution* et *Laboratoire*. Elle décrit l'appartenance d'un membre de la communauté *MATIS* aux institutions et laboratoires partenaires de la formation. *Personne* est sa classe racine.

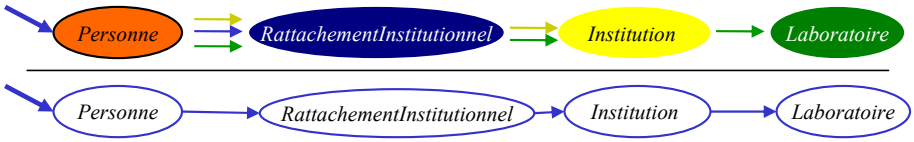

Figure 86 : Chemins d'accès et graphe de navigation de *hcl_rattachement*

La classe Personne est la classe racine des deux hyperclasses *hcl_correspondance* et *hcl_rattachement* ; elle est aussi la seule classe commune à ces deux hyperclasses.

L'intégration de *hcl_rattachement* (hyperclasse secondaire) dans *hcl_correspondance* (hyperclasse primaire) (Figure 87) étend l'espace informationnel de la gestion des correspondances pour y inclure le rattachement institutionnel des personnes.

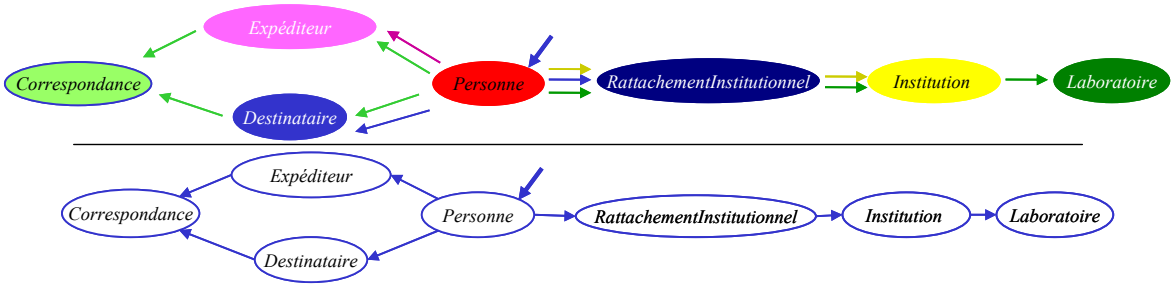

Figure 87 : Chemins d'accès et graphe de navigation de *hcl_correspondance* après l'intégration de *hcl_rattachement*

L'intégration ne modifie pas les chemins d'accès des classes initiales de *hcl_correspondance* ; le fonctionnement de ses hyperméthodes n'est pas altéré. Par ailleurs, les

hyperméthodes initialement définies sur *hcl_rattachement* peuvent être associées à *hcl_correspondance* : les chemins d'accès aux classes issues de *hcl_correspondance* sont identiques dans *hcl_rattachement*. Toutefois, il est nécessaire que tous les éléments des contextes de ses hyperméthodes continuent à être accessibles au niveau de *hcl_correspondance* pour assurer leur fonctionnement.

3.1.3. La classe racine de l'hyperclasse secondaire n'est pas la seule classe commune aux deux hyperclasses

S'il y a plus qu'une classe commune aux deux hyperclasses, il est difficile de résoudre de manière automatique tous les conflits d'écoulement qui peuvent apparaître dans l'hyperclasse résultant de l'intégration. Dans cette situation, il est plus intéressant de définir une nouvelle hyperclasse sur l'union des ensembles de classes des deux hyperclasses ou de se limiter à l'inclusion de l'ensemble de classes de hcl_2 dans hcl_1.

3.1.4. Différence entre l'inclusion de classes et l'intégration d'hyperclasses

Dans la section précédente, nous avons défini l'opération « *Inclure une classe dans une hyperclasse* » qui peut aussi appliquée plusieurs fois pour inclure un ensemble de classes dans une hyperclasse. L'ensemble de classes insérée dans une hyperclasse ne présente pas nécessairement d'unité sémantique ou même la structure d'une hyperclasse. C'est une simple extension de l'espace informationnel de l'hyperclasse par ajout de classes.

Lors de l'intégration d'une hyperclasse secondaire hcl_2 dans une hyperclasse primaire hcl_1, hcl_2 en tant qu'hyperclasse présente une unité sémantique, dispose d'un graphe de navigation, de chemins d'accès, d'hyperattributs et d'hyperméthodes. Il est alors intéressant d'exploiter ses éléments lors de l'intégration tout en préservant les hyperméthodes et les hyperobjets de l'hyperclasse principale. En effet, l'hyperclasse secondaire ne subit pas de modification, c'est seulement la définition de l'hyperclasse principale qui évolue.

L'opération d'intégration d'hyperclasses peut être considérée comme un cas particulier de l'opération d'inclusion d'un ensemble de classes dans une hyperclasse. L'intégration d'hyperclasse respecte de ce fait les pré-conditions de l'inclusion de classes, à savoir, la connexité et la complétude de l'ensemble formé par les deux hyperclasses.

3.2. INTÉGRATION D'UNE HYPERCLASSE DANS UN DIAGRAMME DE CLASSES

Il est possible de faire évoluer un diagramme de classes *D* en y intégrant une (ou plusieurs) hyperclasse *hcl*. Il s'agit, dans un premier temps, de considérer le diagramme de *hcl* comme un diagramme de classes à intégrer dans *D,* de définir une hyperclasse *hcl'* semblable à *hcl* dans I, et d'inclure *hcl'* dans les hyperclasses de *D*.

Le paragraphe suivant décrit les étapes classiques d'intégration de diagrammes de classes.

3.2.1. Étapes d'intégration de modèles

Un processus d'intégration de modèles est composé de quatre étapes distinctes (Batini, 1986)[86] :

1. Pré-intégration

L'étape de pré-intégration consiste à préparer les modèles à l'intégration. Une analyse des modèles est effectuée avant l'intégration pour décider de la politique d'intégration, du choix des modèles à intégrer, de l'ordre de l'intégration, et une attribution possible de préférences aux modèles entiers ou à des parties des modèles. Les stratégies globales pour l'intégration et le nombre des schémas à intégrer en même temps, sont également décidées dans cette phase.

2. Comparaison

L'étape de comparaison consiste à analyser et à comparer les modèles pour déterminer les correspondances parmi leurs éléments et pour détecter des conflits possibles.

L'activité fondamentale dans cette étape consiste à vérifier tous les conflits dans la représentation des mêmes objets dans les différents modèles. Selon les approches de comparaison, les types de conflit présents sont différents mais il est possible de distinguer essentiellement deux types de conflits : les *conflits sémantiques* et les *conflits structurels* :

1. Les *conflits sémantiques* se produisent lorsque les modèles incorporent des formes (des noms) pour les divers objets représentés. Ceci a comme conséquence une prolifération des noms ainsi qu'une contradiction possible parmi des noms dans les schémas composants. Les rapports problématiques parmi des noms sont de deux types :

   1.1. *Homonymes* : quand la même forme (des noms identiques) est employée pour deux concepts différents, provoquant la contradiction (à moins que détectée);

   1.2. *Synonymes* : quand le même concept du monde réel est décrit sous des formes différentes (des noms différents) dans les deux schémas examinés

2. Les *conflits structurels* surgissent en raison de choix différents de la modélisation ou des contraintes d'intégrité. Les divers conflits identifiés par différentes méthodologies sont : les conflits de type, les conflits de dépendance, les conflits d'identifiants et les conflits de contraintes.

3. Résolution des conflits

L'étape de résolution des conflits a pour but de résoudre les conflits détectés durant l'étape précédente. Pour chaque conflit détecté, un effort est fait pour le résoudre de sorte que la fusion de divers schémas soit possible. La résolution automatique des conflits n'est

[86] (Batini, 1986) Batini C., Lenrerini M., Navathe S.B., A comparative analysis of methodologies for database schema integration. ACM Computing Surveys, Vol. 18, No. 4, December 1986.

généralement pas faisable ; l'interaction étroite avec des concepteurs et des utilisateurs est exigée avant que des compromis puissent être réalisés dans n'importe quelle activité réelle d'intégration.

4. Fusion

L'étape de fusion permet la superposition des modèles préalablement mis en conformité, afin d'obtenir un modèle représentatif de tous les modèles fournis au départ. Les résultats intermédiaires sont analysés et, au besoin, restructurés afin de réaliser plusieurs qualités nécessaires à l'élaboration finale.

3.2.2. Intégration d'une hyperclasse dans un diagramme de classes

La première phase de l'intégration d'une hyperclasse *hcl* dans un diagramme de classes *D* consiste à intégrer le diagramme de *hcl* dans *D*. Cette phase implique les quatre étapes d'intégration de modèles, décrites dans le paragraphe précédent.

L'intégration de *hcl* en tant qu'hyperclasse dans *D* s'effectue en deux étapes.

Dans la première étape (Figure 88), une hyperclasse *hcl'*, semblable à *hcl*, est définie dans *D* ; *hcl'* est construite sur les classes de *D* similaires à celles du diagramme de *hcl*, et sur les classes introduites dans *D* lors de l'intégration du diagramme de *hcl* dans *D*.

hcl' n'est pas nécessairement identique à *hcl* ; c'est d'abord une adaptation de *hcl* à la nouvelle structure de *D.* Ensuite, pour que *hcl'* soit valide, il faut qu'elle soit connexe, complète dans D et présenter une unité sémantique. Ces conditions peuvent nécessiter l'insertion dans *hcl'* de classes de *D* qui n'ont pas d'équivalent dans le diagramme de *hcl*, ou la suppression de certaines de ses classes.

Dans la seconde étape *hcl'* est incluse dans une hyperclasse *hcl''*, déjà définie sur *D*. *hcl'* est en quelque sorte *absorbée* par *hcl''*.

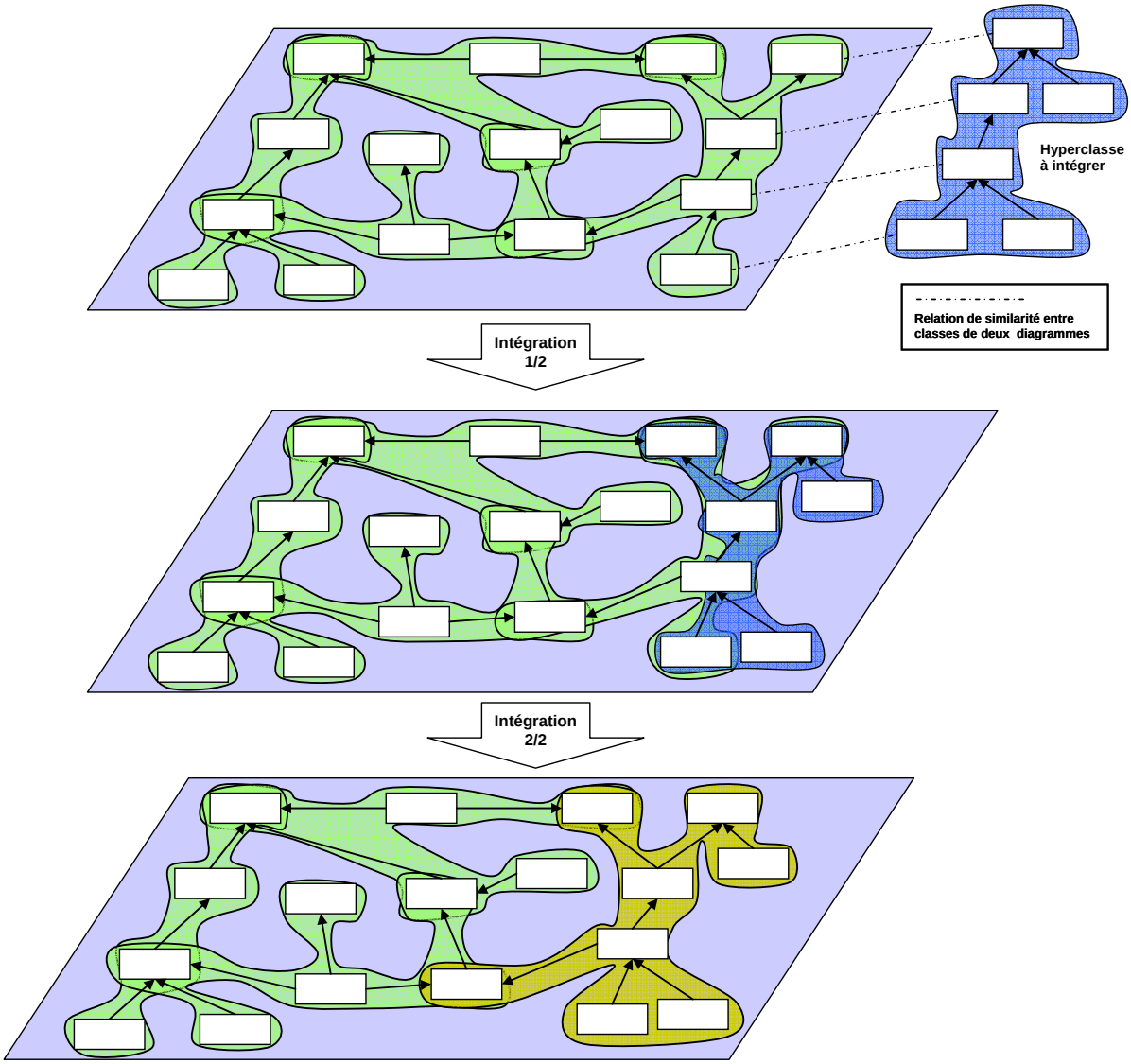

Figure 88 : Intégration de *hcl* dans *D*

CONCLUSION

Le concept d'hyperclasse introduit une forme intéressante de modularité au niveau des modèles statiques de SI. Il permet de prendre en charge la complexité de ces modèles et de maîtriser les situations de recouvrements de compétences qui y apparaissent. Il ne simplifie pas uniquement les phases d'analyse et de conception du SI ; le concept d'hyperclasse permet aussi d'envisager de manière modulaire la gestion et la manutention du SI.

Le concept d'hyperclasse introduit aussi une forme d'indépendance entre la structure du SI et ses traitements. Dans certaines conditions, il est possible de préserver les hyperméthodes d'une hyperclasse et leurs résultats même en cas d'évolution de l'hyperclasse.

Nous avons défini les opérations d'évolution, d'extraction et d'intégration d'hyperclasses, dans une optique d'ingénierie modulaire et évolutive des SIs. Ces opérations permettent à la fois de maîtriser à travers les hyperclasses la complexité de la gestion et de l'évolution des SIs, et de réaliser des opérations plus ou moins élaborées de « copier-coller » intelligents d'hyperclasses et plus tard de composants de SI.

Dans le chapitre suivant, nous introduisons le concept de composant de SI.

CHAPITRE IV

COMPOSANT DE SYSTÈME D'INFORMATION

INTRODUCTION

Pour être consistent, un modèle de SI doit prendre en charge non seulement la structuration de l'information (aspects statiques), mais aussi son comportement, c'est-à-dire, ses aspects dynamiques et règlementaires. Ces aspects peuvent difficilement être isolés sans perte de cohérence du modèle.

Dans ce chapitre, nous présentons un méta-modèle qui intègre à la fois les aspects statiques, dynamiques et réglementaires : il utilise (i) le modèle *binex* pour les aspects statiques, (ii) les *graphes de Gavroche* pour les aspects dynamiques et (iii) un modèle particulier pour la gestion des règles d'intégrité du SI. Ces méta-modèles sont interdépendants pour garantir une plus grande consistance et cohérence des modèles de SI. Ils sont le résultat de travaux de recherche en cours ((Pham, 2003)[87] et (Luu, 2003)[88]).

La propriété de modularité qu'introduisent les hyperclasses au niveau des diagrammes de classes demeure importante et pertinente au niveau des modèles de SI. Nous introduisons alors le concept de composant de SI. Un composant de SI est défini sur une hyperclasse, un ensemble de transactions et un ensemble de règles d'intégrité. Il regroupe ainsi des spécifications structurelles qui décrivent le champ informationnel d'une zone de responsabilité, et des spécifications dynamiques et réglementaires qui décrivent son champ opérationnel.

1. MODÈLE DE SYSTÈME D'INFORMATION

Un modèle de SI prend la forme de plusieurs diagrammes représentant la structure du SI ((i) les *aspects statiques*) et son comportement ((ii) les *aspects dynamiques* et (iii) et les *aspects règlementaires*).

1.1. ASPECTS STATIQUES DANS UN SI

Dans ce travail, nous utilisons le modèle *binex* étendu avec le concept d'hyperclasse pour la modélisation des aspects statiques d'un SI.

---

[87] (Pham, 2003) Pham T., Intégration des aspects statiques et dynamiques des Systèmes d'Information, Mémoire préliminaire, Université de Genève, 2003.

[88] (Luu, 2003) Luu H-T-H., Une approche pour la gestion et l'évolution des règles d'intégrité d'un système d'information. Mémoire préliminaire, Université de Genève, Décembre 2003.

1.2. ASPECTS DYNAMIQUES DANS UN SI

Les modèles dynamiques ont pour objectif de décrire les règles d'évolution des objets au cours du temps. Ils représentent les types d'évènements qui surviennent dans le SI et les changements d'états résultant du traitement de ces événements. L'état d'un objet d'une classe évolue suite à l'exécution d'opérations (création, suppression, mise-à-jour) ou de méthodes de classes.

« Dans les méthodes objets, la dynamique des objets est décrite soit à l'aide de diagrammes d'états/transitions (dérivés des machines d'états finis), soit à l'aide de spécifications formelles utilisant des langages de règles ou d'autres langages abstraits » (Bouzeghoub & al. 1997)[89]. Les formalismes les plus utilisés pour les modèles dynamiques restent les diagrammes d'état-transition d'*UML* (Annexe H) ou les réseaux de Petri (Annexe I). Dans notre travail, nous avons adopté le formalisme des *graphes de Gavroche*. Ce formalisme a été introduit dans (Léonard, 1999)[90] pour décrire les propriétés dynamiques d'un SI.

1.2.1. Formalisme des graphes de Gavroche

Un graphe de *Gavroche* est un graphe biparti ; c'est un réseau de *nœuds* et d'*étoiles* où les nœuds correspondent un à un à des *classes* et les étoiles à des *transactions*.

La notion de *transaction* utilisée dans les graphes de Gavroche est inspirée des concepts de « *transaction commerciale* » et de « *transaction bancaire* » et est associée à une activité productrice ou consommatrice d'informations dans un processus de prise de décision.

Une *transaction tr* dans un graphe de Gavroche a une ou plusieurs c*lasses (nœuds) de base* (°*tr*) et une ou plusieurs *classes de rajout* (*tr*°). La transaction *tr* se base sur les informations contenues dans les objets des classes de °*tr* et produit des objets des classes *tr*°.

Au niveau d'une transaction, les classes de base et les classes de rajout représentent les ressources informationnelles respectivement nécessaires et produites. Ces éléments sont primordiaux, car sans les bonnes ressources aucune décision justifiée ne peut être prise. L'exploitation d'une information de base (°*tr*) pour produire une nouvelle information (*tr*°) constitue la valeur ajoutée au niveau d'une transaction.

°tr = ensemble des classes de base de tr

tr° = ensemble des classes de rajout de tr

Les ressources informationnelles nécessaires sont des « données et informations qui fournissent une base pour créer plusieurs alternatives dans un contexte de prise de décision ».

---

[89] (Bouzeghoub & al. 1997) Bouzeghoub M., Gardarin G., Valduriez P., Les objets. Paris : Eyrolles, 1997. ISBN 2212089570.

[90] (Léonard, 1999) Léonard M., M7 : approche évolutive des systèmes d'information. Proc. Inforsid'99. La Garde, France, mai 1999.

Ces ressources doivent être « consistantes et cohérentes afin de permettre une prise de décision … et aussi pour minimiser le temps nécessaire pour choisir l'alternative optimale dans un contexte donné » (Daft, 2000). Par la prise de décision ou par le choix d'une alternative au niveau d'une transaction, les informations de base sont transformées et réarrangées : ce traitement produit de nouvelles informations au niveau des classes de rajout.

Une transaction (étoile dans un réseau biparti) a au moins une classe de base et au moins une classe de rajout. Les ensembles de classes de base et de classes de rajout sont disjoints.

$$°tr \neq \emptyset \text{ et } tr° \neq \emptyset$$

$$°tr \cap tr° = \emptyset$$

1.2.1.1. Domaine d'une transaction

Le *domaine d'une transaction tr* est l'union de ses classes de base et ses classes de rajout.

$$Domaine(tr) = °tr \cup tr°$$

Il représente l'espace informationnel nécessaire pour la conclusion de la transaction et couvre « les relations entre les informations qui sont éditées, travaillées et manipulées dans un processus de prise de décision » (Ott & Natansky, 1997)[91].

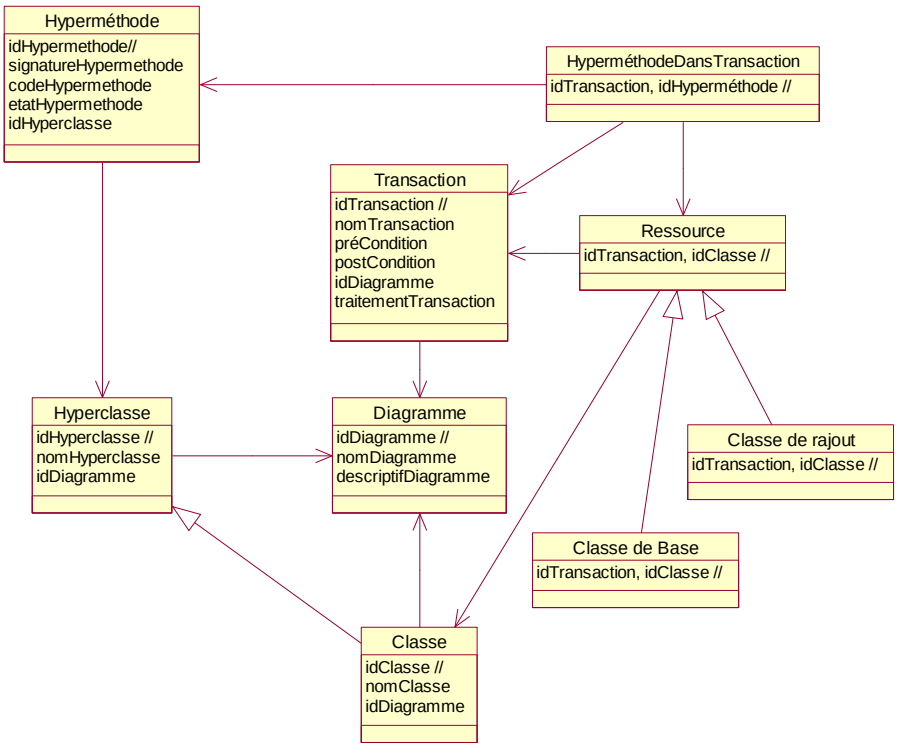

Figure 89 : Modèle des *graphes de Gavroche*

La Figure 89 présente le modèle des transactions. En plus de ses classes de base et de rajout, une transaction est définie par (i) un *traitement*, (ii) une *pré-condition* et (iii) une *post-condition*.

1.2.1.2. Traitement

Dans les graphes de Gavroche, le traitement associé à une transaction est une séquence d'opérations, traitées comme une unité, qui exploite les objets de classes de base pour produire des objets dans les classes de rajout de la transaction. Ces opérations utilisent les hyperméthodes définies sur le diagramme de classes, en particulier les méthodes des classes de base et de rajout de la transaction, notamment les primitives prédéfinies de mise à jour ou de création ou de suppression d'objets des classes.

Une transaction produit nécessairement des objets dans ses classes de rajout ; seules les transactions génératrices (créatrices, productrices) d'objets sont considérées dans les graphes de *Gavroche* (Pham, 2003)[92].

1.2.1.3. Pré-condition et post-condition à une transaction

Une *pré-condition* à une transaction est une condition au déclenchement de la transaction. Elle concerne les objets des classes de base.

Une *post-condition* à une transaction est une condition à la création d'objets dans les classes de rajout et donc à la terminaison de la transaction.

Les pré et post-conditions à une transaction prennent la forme d'expressions booléennes pouvant être complexes et portant sur les objets respectivement des classes de base et de rajout de la transaction.

1.2.1.4. Exemples de transaction

Pour l'inscription des étudiants aux modules de la formation *MATIS*, nous définissons une transaction que nous appelons *InscriptionAuModule* (Figure 90).

---

[91] (Ott & Natansky, 1997) Ott M., Nastansky L., Groupware Based Organization – Design for dynamic Workflow Management and Office Systems. Workgroup Computing Competence Center Paderborn, Universität-GH Paderborn. 1997.

[92] (Pham, 2003) Pham T., Intégration des aspects statiques et dynamiques des Systèmes d'Information, Mémoire préliminaire, Université de Genève, 2003.

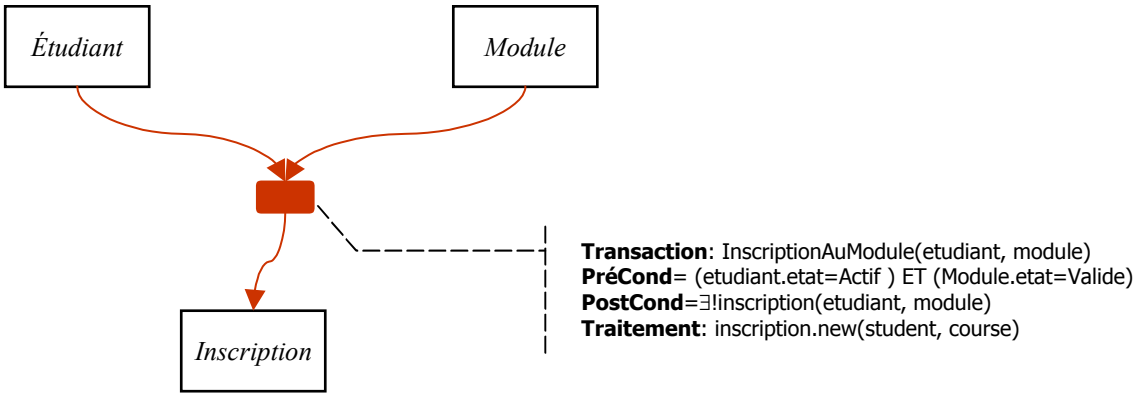

Figure 90 : La transaction *InscriptionAuModule*

La transaction *InscriptionAuModule* admet *Étudiant* et *Module* comme classes de bases et *Inscription* comme classe de rajout.

Pour un objet *étudiant* et un objet de *module* respectant la pré-condition [*(étudiant.état=Actif) ET (Module.état=Valide)*] le traitement associé à la transaction (créer un nouvel objet d'*Inscription* associant *étudiant* à *module*) est exécuté, à condition de ne pas enfreindre la post-condition (il n'existe pas déjà d'objet d'*Inscription* reliant les objets *étudiant* et *module*).

La Figure 91 représente deux autres transactions de *M@TIS* intégrées avec son diagramme de classes.

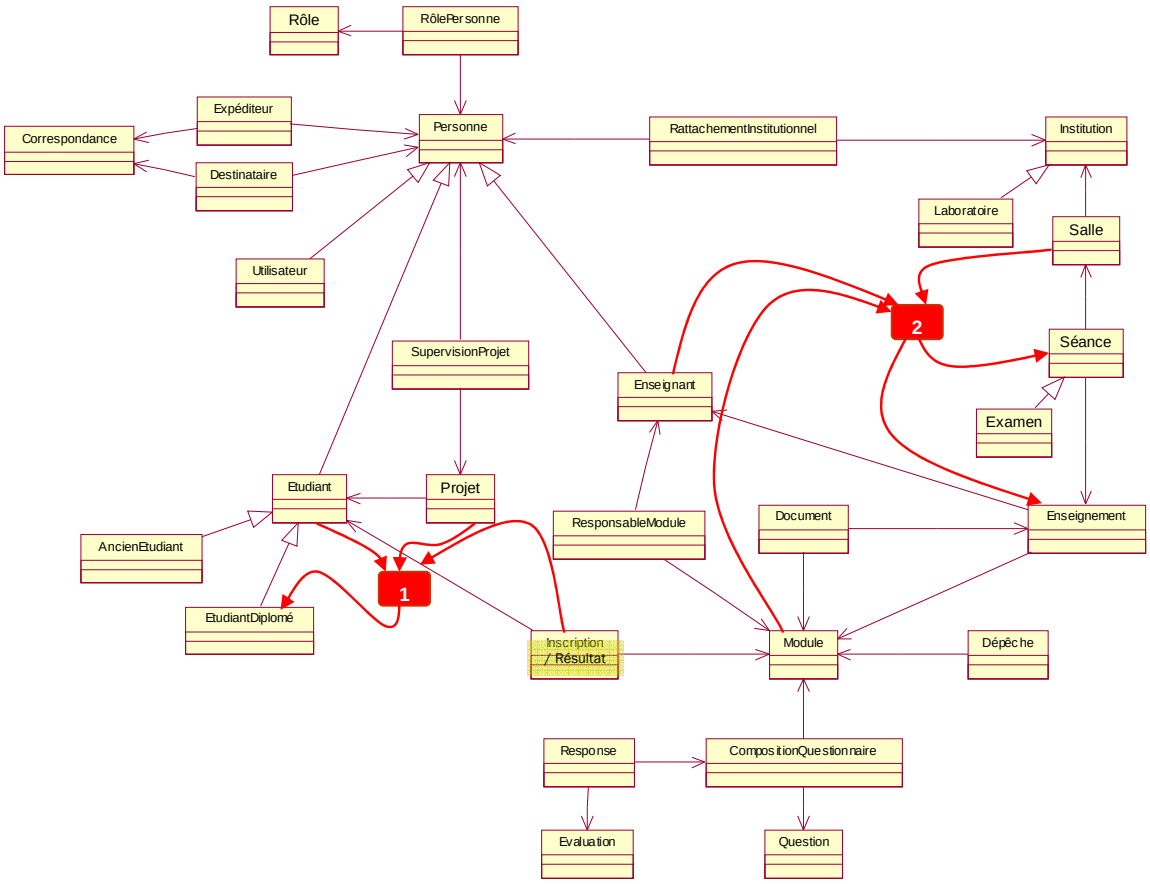

Figure 91 : Exemples de transactions dans *M@TIS*

La transaction « *1* » concerne l'analyse des résultats des étudiants en fin de formation : il s'agit de décider pour un *étudiant* donné, au vu de ses résultats aux modules auxquels il est inscrit ainsi que sa note de projet de recherche, si l'étudiant vérifie les conditions pour obtenir son diplôme et devenir un *étudiant diplômé*.

La transaction « *2* » permet à partit d'un *module*, un *enseignant* et une *salle* de définir une *séance* et un objet d'*enseignement*.

1.2.2. Opérations sur les graphes de Gavroche

Nous définissons un ensemble d'opérations pour la manipulation et l'évolution des graphes de *Gavroche*. Cet ensemble, complet, a été obtenu par l'application des trois opérateurs génériques (i) *créer*, (ii) *supprimer* et (iii) *mettre à jour* à chacun des éléments du modèle. Ces opérations n'ont pas d'impact sur les objets existants du SI.

Pour prendre en compte l'extension du méta-modèle *binex* avec les concepts des graphes de Gavroche, nous définissons également les actions supplémentaires à exécuter lors de la suppression d'objets des classes *Hyperméthode*, *Hyperclasse* et *Diagramme*.

1.2.2.1. Transaction

1. Créer une transaction

Paramètres :	▪ Le nom de la nouvelle transaction *tr* ; ▪ °*tr*, l'ensemble des classes de base de *tr* ; ▪ *tr*°, l'ensemble des classes de rajout de *tr* ; ▪ Les pré- et post-conditions de *tr* s'il y a lieu ; ▪ Le traitement associé à *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Le nom de la nouvelle transaction n'est pas déjà utilisé pour une autre transaction dans le même modèle ; ▪ °*tr* ≠Ø ; ▪ *tr*°≠Ø ; ▪ °*tr* ∩ *tr*°= Ø ; ▪ Le traitement associé à *tr* n'invoque que des méthodes des classes de *Domaine(tr)*.
Post-act. *Modél. :*	▪ Créer *tr ;* ▪ Créer les objets correspondants aux classes de°*tr* et de *tr*° dans *Ressource*, *Classe de base* et *Classe de rajout* ; ▪ En fonction des méthodes invoquées dans le traitement associé à *tr*, créer les objets correspondants dans *HyperméthodeDansTransaction*.

2. Supprimer une transaction

Paramètres :	▪ La transaction *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. *Modél. :*	▪ Supprimer tout objet de la classe *Ressource* lié à *tr* ; ▪ Supprimer tout objet de la classe *HyperméthodeDansTransaction* lié à *tr* ; ▪ Supprimer *tr.*

3. Renommer une transaction

Paramètres :	▪ La transaction *tr* ; ▪ Le nouveau nom de *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Le nouveau nom n'est pas déjà utilisé pour une autre transaction dans *D*.
Post-act. *Modél. :*	▪ Renommer *tr*.

4. Mettre à jour une pré-condition

Paramètres :	▪ La transaction *tr* ; ▪ La nouvelle pré-condition de *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. *Modél. :*	▪ Mettre à jour la pré-condition de *tr*.

5. Mettre à jour une post-condition

Paramètres :	▪ La transaction *tr* ; ▪ La nouvelle post-condition de *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modél. :	▪ Mettre à jour la post-condition de *tr*.

6. Mettre à jour un traitement

Paramètres :	▪ La transaction *tr* ; ▪ Le nouveau traitement associé à *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Le nouveau traitement associé à *tr* n'invoque que des méthodes des classes de *Domaine(tr)*.
Post-act. Modél. :	▪ Mettre à jour le traitement associé à *tr* ; ▪ Mettre-à-jour l'ensemble des objets d'*HyperméthodeDansTransaction* liés à *tr*.

1.2.2.2. Hyperméthode dans Transaction

1. Associer une hyperméthode à une transaction

Paramètres :	▪ La transaction *tr* ; ▪ L'hyperméthode *hm* de l'hyperclasse *hcl* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Le contexte de *hm* est inclus dans le domaine de *tr*.
Post-act. Modél. :	▪ Associer *hm* à *tr* dans *HyperméthodeDanstransaction*.

2. Dissocier une hyperméthode d'une transaction

Paramètres :	▪ La transaction *tr* ; ▪ L'hyperméthode *hm* de l'hyperclasse *hcl* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Modél. :	▪ Dissocier *hm* de *tr* dans *HyperméthodeDanstransaction*.

1.2.2.3. Ressource

1. Associer une ressource à la transaction

Paramètres :	▪ La transaction *tr* ; ▪ La classe *cl*, nouvelle classe ressource de *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ *cl* n'est pas déjà une classe ressource dans *tr*.
Post-act. Modél. :	▪ Associer *cl* à *tr* dans *Ressource*.

2. Dissocier une ressource de la transaction

Paramètres :	▪ La transaction *tr* ; ▪ La classe *cl*, ressource à dissocier de *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ $\{°tr\text{-}\{cl\}\} \neq Ø$; ▪ $\{tr°\text{-}\{cl\}\} \neq Ø$; ▪ Le traitement associé à *tr* n'invoque aucune méthode de *cl*.
Post-act. *Modél. :*	▪ Dissocier *cl* à *tr :* supprimer l'objet correspondant à *cl* et *tr* dans *Ressource*.

1.2.2.4. Classe de base

1. Associer une classe de base à la transaction

Paramètres :	▪ La transaction *tr* ; ▪ La classe *cl*, nouvelle classe de base de *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ $\{°tr \cup \{cl\}\} \cap tr° = Ø$.
Post-act. *Modél. :*	▪ Associer *cl* à $°tr$: créer un nouvel objet lié à *cl* et *tr* dans *Classe de base*.

2. Dissocier une classe de base de la transaction

Paramètres :	▪ La transaction *tr* ; ▪ La classe *cl* à dissocier de $°tr$; ▪ Le diagramme *D*.
Pré-cond. :	▪ $\{°tr\text{-}\{cl\}\} \neq Ø$; ▪ Le traitement associé à *tr* n'invoque aucune méthode de *cl*.
Post-act. *Modél. :*	▪ Dissocier *cl* de $°tr$: supprimer l'objet de *Classe de base* lié à *cl* et *tr*.

1.2.2.5. Classe de rajout

1. Associer une classe de rajout à la transaction

Paramètres :	▪ La transaction *tr* ; ▪ La classe *cl*, nouvelle classe de rajout de *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ $°tr \cap \{tr° \cup \{cl\}\} = Ø$.
Post-act. *Modél. :*	▪ Associer *cl* à $tr°$: créer un nouvel objet lié à *cl* et *tr* dans *Classe de rajout*.

2. Dissocier une classe de rajout de la transaction

Paramètres :	▪ La transaction *tr* ; ▪ La classe *cl* à dissocier de *tr*° ; ▪ Le diagramme *D*.
Pré-cond. :	▪ *{tr°-{cl}} ≠Ø* ; ▪ Le traitement associé à *tr* n'invoque aucune méthode de *cl*.
Post-act. Modél. :	▪ Dissocier *cl* de *tr*° : supprimer l'objet de *Classe de rajout* lié à *cl* et *tr*.

1.2.2.6. Hyperméthode

1. Supprimer une hyperméthode

Paramètres :	▪ L'hyperméthode *hm* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Revoir tout traitement d'une transaction invoquant *mth* ; ▪ Supprimer tous les objets d'*HyperméthodeDansTransaction* reliés à *mth*.

1.2.2.7. Hyperclasse

1. Supprimer une hyperclasse

Paramètres :	▪ L'hyperclasse *hcl*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Supprimer les classes *cl* spécialisation de *hcl* des ressources de toute transaction ; ▪ Supprimer toute classe de base ou de rajout spécialisation de *cl*.

1.2.2.8. Diagramme

1. Supprimer un diagramme

Paramètres :	▪ Nom du diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Supprimer toute transaction de *D*.

1.3. ASPECTS RÉGLEMENTAIRES DANS UN SI

Les règles d'intégrité (RIs) constituent une composante essentielle du SI. Leur rôle est de préserver sa cohérence et sa consistance durant son exploitation.

Les définitions concernant les RIs sont tirées de (Léonard, 1988)[93] et (Luu, 2003)[94].

Une RI est une condition qui doit être validée par l'information stockée et manipulée dans le SI. Elle est définie sur une ou plusieurs classes du diagramme de classes, et sa validation s'effectue de manière algorithmique. Lorsque l'ensemble des RIs est valide, le SI est dit *cohérent*.

En plus des contraintes définies sous forme de liens existentiels ou de spécialisation-généralisation dans le diagramme de classes, il existe deux autres types de RIs : (i) les *RIs statiques* et (ii) les *RIs de comportement* :

1) une *RI statique* s'applique à chaque objet d'une classe (*RI statique individuelle*) ou à un ensemble d'objets d'une classe (*RI statique ensembliste*).
2) une *RI de comportement* est relative à une opération de modification d'un ou plusieurs objets d'une classe. Elle exprime une condition logique entre l'état du SI avant la modification et son état après modification.

1.3.1. Expression

L'*expression* d'une RI est définie sous forme d'une expression mathématique ou d'une expression algorithmique terminée par une *condition*. Cette condition doit être vérifiée pour tout état du SI ou pour tout changement d'états de celle-ci.

1.3.2. Contexte

Le *contexte* d'une règle d'intégrité *ri*, noté *Contexte(ri)*, désigne les classes concernées par sa définition et sa validation. C'est l'ensemble de classes sur lesquelles la RI est définie.

1.3.3. Condition

La *condition* d'une RI doit être vérifiée pour tout état du SI (RI statique) ou pour tout changement de son état (RI de comportement) par les objets des classes du contexte.

---

[93] (Léonard, 1988) Léonard M., Structures des bases de données, ISBN 2-04-016407-3, DUNOD informatique.

[94] (Luu, 2003) Luu H-T-H., Une approche pour la gestion et l'évolution des règles d'intégrité d'un système d'information. Mémoire préliminaire, Université de Genève, Décembre 2003.

1.3.4. Portée

La *portée* d'une RI désigne l'ensemble des méthodes de classes qui doivent contenir un algorithme de validation de la RI, pour qu'un état cohérent du SI soit transformé par cette méthode en un autre état cohérent.

La portée montre l'ensemble des méthodes de classes qui peuvent transgresser la RI en s'exécutant. Ces primitives s'appellent les *risques* de la RI.

1.3.5. Réponse

La *réponse* d'une RI indique les actions à entreprendre lorsqu'elle est transgressée. Le SI peut simplement refuser les modifications incohérentes ou bien effectuer certaines modifications de compensation afin de conserver la cohérence des données.

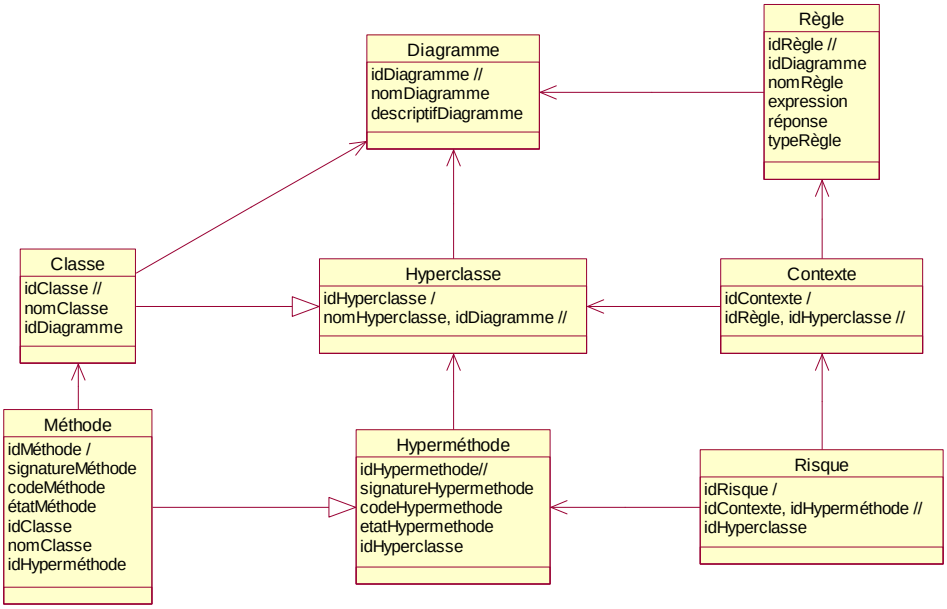

Figure 92 : Modèle des règles d'intégrité adapté aux hyperclasses à partir de (Luu, 2003))

Dans la Figure 92, la notion de *risque* indique les hyperméthodes (et les méthodes de classe) dont l'exécution peut transgresser la RI.

1.3.6. Exemple de règles d'intégrité

Voici quelques exemples de règles d'intégrité dans *M@TIS*, extraites et adaptées à partir de (Leung & Olivari, 2002)[95] :

[95] (Leung & Olivari, 2002) Leung K-F, Olivari A., Conception et développement d'une solution web pour la formation MATIS. Novembre 2002.

1) Pour créer un nouvel objet dans *Étudiant*, il est nécessaire que l'objet de *Personne* correspondant soit associé au rôle *Étudiant* via la classe *RôlePersonne*.

L'expression de cette RI est :

```
∀ rp ∈ ROLEPERSONNE ∀ e ∈ ETUDIANT ∀ p ∈ PERSONNE ∀ rl ∈ ROLE
(e.idPers = p.idPers) ∧ (p.idPers = rp.idPers) ∧ (rp.idRôle =
rl.idRôle) ∧ (rl.Rôle = 'Etudiant')
```

Les classes *Étudiant*, *Personne*, *RôlePersonne* et *Rôle* forment le contexte de cette RI.

L'exécution des méthodes *Etudiant.créer()*, *Personne.supprimer()*, *Rôle.supprimer()* et *RôlePersonne.supprimer()* constitue un risque de transgresser la RI ; elles constituent sa portée.

La réponse de la RI est : Refuser la spécialisation d'une personne en étudiant.

2) Un étudiant ne peut répondre au questionnaire d'évaluation d'un module donné que s'il est inscrit à celui-ci ;
3) Un étudiant ne peut remplir plus d'une fois le questionnaire d'évaluation d'un module donné ;
4) Le login d'une personne est unique ;
5) On ne peut ajouter une séance, un document, un examen ou une dépêche pour un module inactif ou archivé ;
6) Un étudiant ne peut modifier ou supprimer les informations sur son projet que s'il n'a pas encore reçu une note pour son travail de diplôme ;
7) Un étudiant ne peut s'inscrire qu'aux cours en tronc commun et aux cours de sa spécialisation ;
8) Un étudiant ne peut être considéré comme ayant réussi la formation et considéré comme étudiant diplôme que s'il obtient une moyenne supérieure à 10 pour les modules auxquels il est inscrit et une note supérieure à 10 pour son projet de recherche.

1.3.7. Opérations sur le modèle de règles d'intégrité

Le modèle de règles d'intégrité est muni d'un ensemble complet d'opérations d'évolution obtenu par par l'application des trois opérations génériques (i) *créer*, (ii) *supprimer* et (iii) *mettre à jour* à chacun des éléments. Ces opérations n'ont pas d'impact sur les objets existants du SI.

Les opérations de suppression d'objets des classes *Méthode*, *Classe* et *Diagramme* évoluent pour prendre en compte les règles d'intégrité. Nous définissons les actions supplémentaires à exécuter lors de la suppression d'objets de ces classes.

1.3.7.1. Règle

1. Créer une règle

Paramètres :	▪ Nom de la règle *rg* ; ▪ Expression de *rg* ; ▪ Réponse de *rg ;* ▪ Type de *rg* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Le nom de *rg* n'est pas utilisé pour une autre règle de *D* ; ▪ *rg* est validée dans l'état actuel du SI.
Post-act. *Modél. :*	▪ définir *rg* dans *D*.

2. Supprimer une règle

Paramètres :	▪ La règle *rg* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. *Modél. :*	▪ Supprimer tout risque lié à *rg* ; ▪ Supprimer tout élément du contexte de *rg* ; ▪ Supprimer *rg* de *D*.

3. Renommer une règle

Paramètres :	▪ La règle *rg* ; ▪ La nouveau nom de *rg* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Le nouveau nom de *rg* n'est pas utilisé pour une autre règle de *D.*
Post-act. *Modél. :*	▪ Mettre-à-jour l'expression de *rg*.

4. Mettre-à-jour l'expression d'une règle

Paramètres :	▪ La règle *rg* ; ▪ La nouvelle expression de *rg* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ La nouvelle expression de *rg* est validée dans l'état actuel du SI.
Post-act. *Modél. :*	▪ Mettre-à-jour l'expression de *rg*.

5. Mettre-à-jour la réponse d'une règle

Paramètres :	▪ La règle *rg* ; ▪ La nouvelle réponse de *rg* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. *Modél. :*	▪ Mettre-à-jour la réponse de *rg*.

1.3.7.2. Contexte

1. Ajouter une classe/hyperclasse au contexte d'une règle

Paramètres :	▪ La règle *rg* ; ▪ La classe *cl* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ La classe *cl* ne fait pas partie du contexte de *rg* ; ▪ *rg* et *cl* sont définies dans *D*.
Post-act. *Modél. :*	▪ Ajouter *cl* au contexte de *rg*.

2. Supprimer une classe/hyperclasse du contexte d'une règle

Paramètres :	▪ La règle *rg* ; ▪ La classe *cl* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ La classe *cl* fait partie du contexte de *rg*.
Post-act. *Modél. :*	▪ Supprimer *cl* du contexte de *rg*.

1.3.7.3. Risque

1. Ajouter une hyperméthode au risque d'une règle

Paramètres :	▪ La règle *rg* ; ▪ *hm*, hyperméthode de l'hyperclasse *hcl* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ *hcl* fait partie du contexte de *rg*.
Post-act. *Modél. :*	▪ Ajouter *hm* à l'ensemble des hyperméthodes à risque de *rg*.

2. Supprimer une hyperméthode du risque d'une règle

Paramètres :	▪ La règle *rg* ; ▪ *hm*, hyperméthode de l'hyperclasse *hcl* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. *Modél. :*	▪ Supprimer *hm* de l'ensemble des hyperméthodes à risque de *rg*.

1.3.7.4. Hyperméthode

1. Supprimer une méthode

Paramètres :	▪ L'hyperméthode *hm* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Supprimer *hm* des hyperméthodes à risque.

1.3.7.5. Hyperclasse

1. Supprimer une classe

Paramètres :	▪ L'hyperclasse *hcl*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Supprimer les classes *cl* spécialisation de *hcl* du contexte de toute règle d'intégrité.

1.3.7.6. Diagramme

1. Supprimer un diagramme

Paramètres :	▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Supprimer toute règle d'intégrité de *D*.

2. COMPOSANT DE SYSTÈME D'INFORMATION

Un composant de système d'information est un artefact qui regroupe des spécifications structurelles (statiques), des spécifications dynamiques et des spécifications réglementaires (sous formes de RIs) d'un SI ou d'une partie d'un SI.

Un composant *csi* est un (sous)-modèle autonome de SI, défini (Figure 93) sur une hyperclasse *hcl* (son espace statique *esc*), un ensemble de transactions (son espace dynamique *edc*) et un ensemble de règles d'intégrités (son espace réglementaire *erc*).

csi = <esc, edc, erc>

esc=hcl, edc=$\cup_k\{tr_k\}$ et erc=$\cup_i\{ri_i\}$

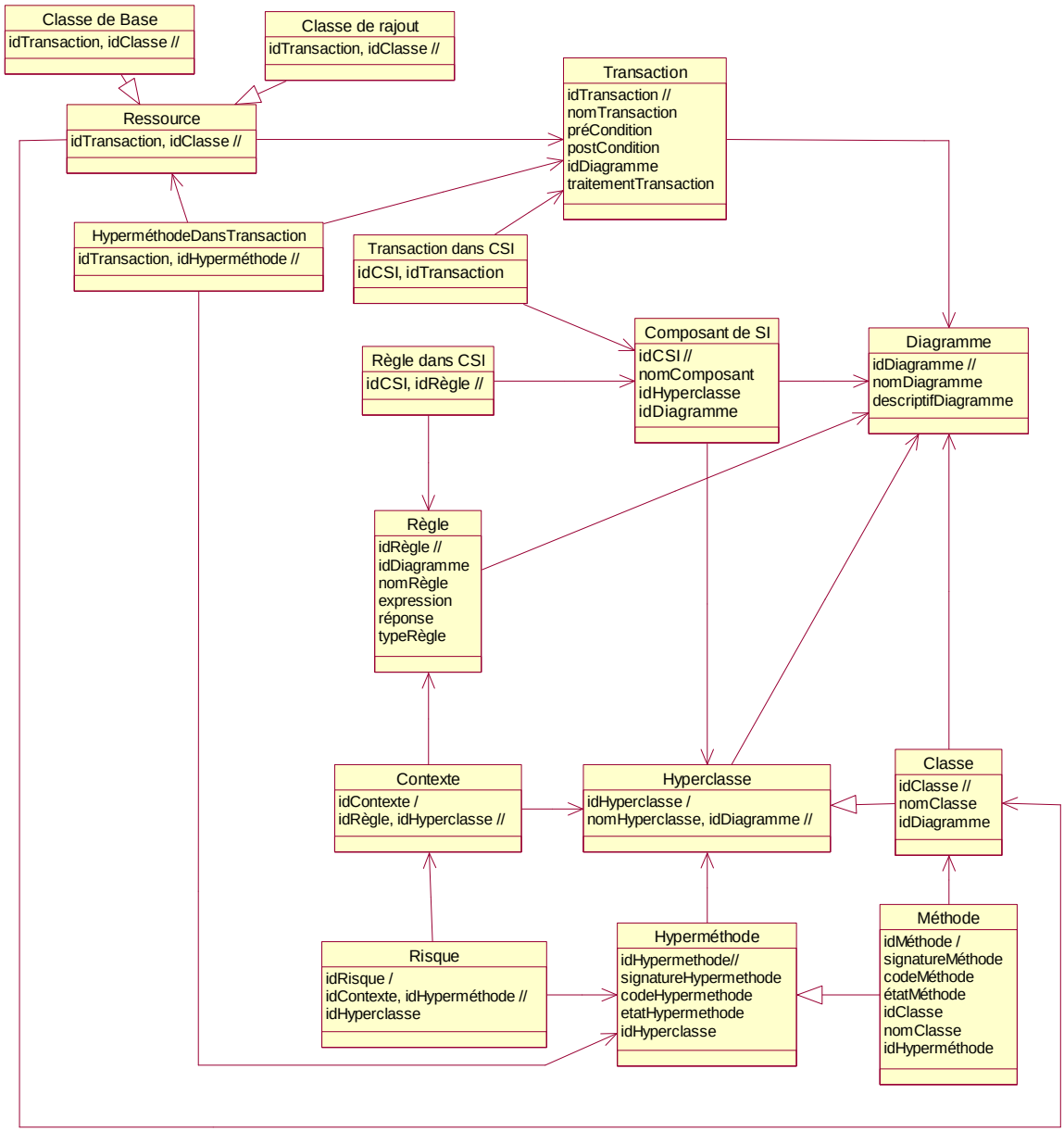

Figure 93 : Modèle des composants de SI

2.1. RÈGLES DE CONFORMITÉ D'UN CSI

Pour être valide, un composant de système d'information *csi* respecte un ensemble de cinq règles de conformité qui garantissent à la fois son autonomie et sa cohérence :

1. L'espace statique est une hyperclasse valide

Cette règle garantit la connexité et la complétude de l'espace statique *esc* de *csi* ; *esc* contient tous les éléments qui participent à sa définition.

2. Complétude des transactions

Si une transaction tr_k appartient à *csi*, toutes les classes appartenant au domaine *Domaine*(tr_k) de tr_k appartiennent à *esc*.

$$\forall tr_k \in erc,\ \forall cl_i \in Domaine(tr_k);\ cl_i \in esc$$

3. Complétude des traitements

Le traitement associé à une transaction tr_k de *csi* n'invoque que des hyperméthodes ou des méthodes de classes de l'hyperclasse de *csi*.

4. Cohérence des transactions

Toute transaction tr_k de *csi* valide toutes les règles d'intégrité de *csi* : aucune transaction de *csi* ne présente de risque, à travers les méthodes de classes qu'elle invoque, d'enfreindre une règle d'*erc*.

5. Complétude des règles d'intégrité

Toutes les classes appartenant au contexte *Contexte*(ri_l) d'une règle d'intégrité ri_l appartiennent à *esc*.

$$\forall ri_l \in erc,\ \forall cl_i \in Contexte(ri_l);\ cl_i \in esc$$

2.2. EXEMPLE DE COMPOSANT DE SI

La Figure 94 représente le composant de SI *csi_Inscription_Étudiant.*

csi_Inscription_Étudiant est construit à partir de :

1) l'hyperclasse *hcl_inscription_étudiant*, construite sur les classes *Personne*, *Étudiant*, *ÉtudiantDiplômé*, *Inscription*, *Module*, *Projet* et *SupervisionProjet*.
2) la transaction *Inscription*, ayant *Étudiant* et *Module* comme classes de base et *Inscription* comme classe de rajout ;
3) la transaction *Définition de Projet*, ayant *Personne* et *Étudiant* comme classes de base et *Projet* et *SupervisionProjet* comme classes de rajout ;
4) les règles d'intégrités *RI1* (« *Un étudiant ne peut s'inscrire qu'aux cours en tronc commun et aux cours de sa spécialisation* »), *RI2* (« *Un étudiant ne peut s'inscrire qu'aux modules actifs* ») et *RI3* (« *Un étudiant ne peut mettre à jour les informations sur son projet que s'il n'a pas encore reçu une note pour son projet* »).

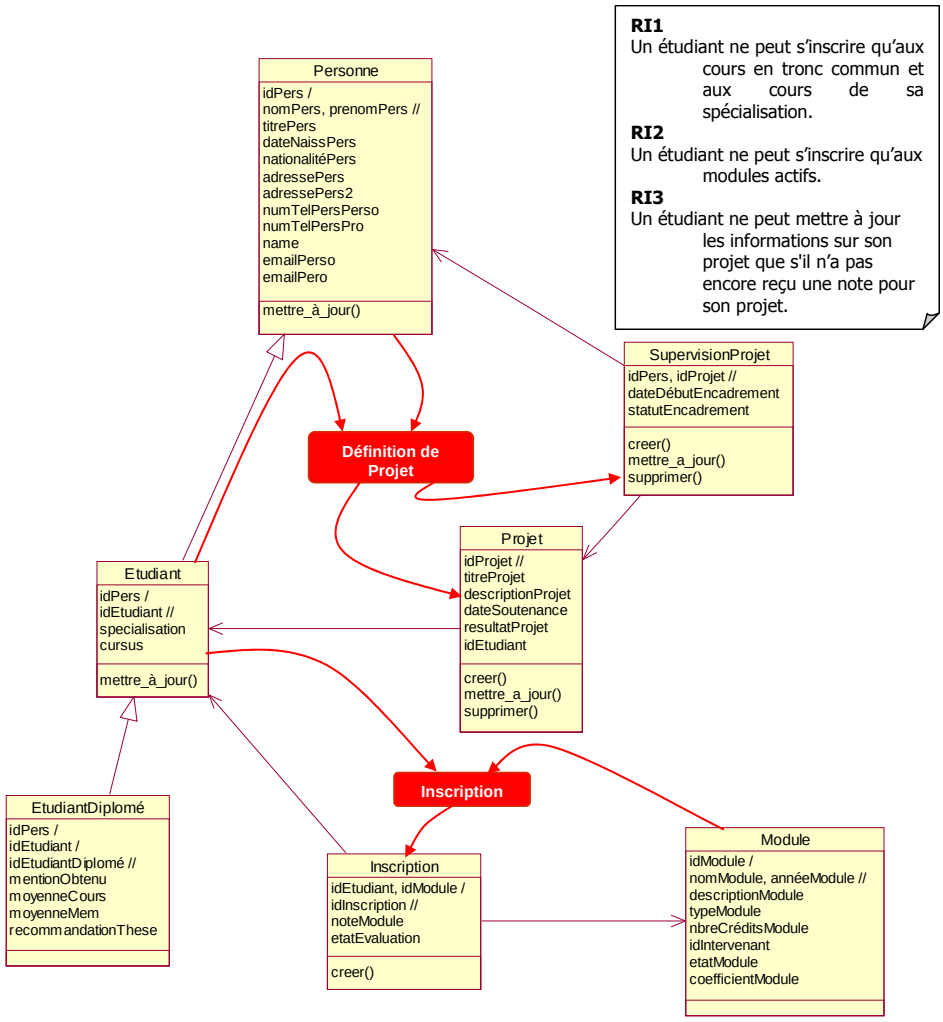

Figure 94 : Le composant de SI *csi_Inscription_Étudiant*

csi_Inscription_Étudiant représente le « poste de travail » d'un étudiant dans *M@TIS* pour la gestion de ses inscriptions aux modules et la déclaration de son projet de recherche : *hcl_inscription_étudiant* représente l'espace informationnel (l'univers sémantique) de ce poste ; les transactions *Inscription* et *Définition de Projet* représentent les opérations productrices d'informations attribuées aux étudiants ; et les règles d'intégrité

2.3. OPÉRATIONS SUR LES COMPOSANTS DE SI

Un composant de SI est formé d'une hyperclasse, d'un ensemble de transactions et d'un ensemble de règles d'intégrité. Une opération de manipulation et d'évolution de composant de SI consiste en l'ajout ou la soustraction d'un de ses éléments tout en respectant les règles de conformité.

Pour être valide, un composant de système d'information *csi* respecte un ensemble de cinq règles de conformité qui garantissent à la fois son autonomie et sa cohérence. Ces règles de conformité sont des règles d'intégrité au niveau du modèle de composant de SI :

1. *RI1* : L'espace statique est une hyperclasse valide ;
2. *RI2* : Complétude des transactions ;
3. *RI3* : Complétude des traitements ;
4. *RI4* : Cohérence des transactions ;
5. *RI5* : Complétude des règles d'intégrité.

La Table 2 montre les opérations atteignables au niveau d'un composant. Certaines de ces opérations constituent un risque pour ses règles de conformité ; l'indication *Ri* indique que l'opération constitue un risque pour la règle *RIi*.

	Créer()	***Supprimer()***	***Mettre-à-jour()***
Composant	*Créer un CSI*	*Supprimer un CSI*	*Renommer un CSI*
	R1-R2-R3-R4-R5	-	-
Transaction dans CSI	*Associer une transaction à un CSI*	*Dissocier une transaction d'un CSI*	-
	R2-R3-R4	-	-
Règle dans CSI	*Associer une règle d'intégrité à un CSI*	*Dissocier une règle d'intégrité à un CSI*	-
	R4-R5	-	-

Table 2 : Opérations sur un composant de SI

2.3.1. Composant de SI

2.3.1.1. Créer un CSI

Paramètres :	▪ Nom du nouveau composant *csi* ; ▪ Description de *csi* ; ▪ L'hyperclasse *hcl* ; ▪ Ensemble des RIs de *csi* ; ▪ Ensemble des transactions de *csi* ;
Pré-cond. :	▪ Le nom de *csi* n'est pas déjà utilisé pour un autre composant ; ▪ Toutes les transactions de *csi* valident toutes les RIs de *csi* ; ▪ Si une transaction *tr* appartient à *csi*, toutes les classes appartenant au domaine de *tr* appartiennent à *hcl* ; ▪ Toutes les classes appartenant au contexte d'une RI appartiennent à *hcl*.
Post-act. Modél. :	▪ Créer *csi* ; ▪ Associer *hcl* à *csi* ; ▪ Associer les transactions à *csi ;* ▪ Associer les RIs à *csi.*

2.3.1.2. Supprimer un CSI

Paramètres :	▪ Le composant *csi.*
Pré-cond. :	▪ Aucune
Post-act. *Modél. :*	▪ Dissocier hcl à *csi* ; ▪ Dissocier les transactions à *csi ;* ▪ Dissocier les RIs à *csi ;* ▪ Supprimer *csi*.

2.3.1.3. Renommer un CSI

Paramètres :	▪ Nom du nouveau composant *csi* ; ▪ Description de *csi* ; ▪ L'hyperclasse *hcl* ; ▪ Ensemble des RIs de *csi* ; ▪ Ensemble des transactions de *csi* ;
Pré-cond. :	▪ Le nom de *csi* n'est pas déjà utilisé pour un autre composant ; ▪ Toutes les transactions de *csi* valident toutes les RIs de *csi* ; ▪ Si une transaction *tr* appartient à *csi*, toutes les classes appartenant au domaine de *tr* appartiennent à *hcl* ; ▪ Toutes les classes appartenant au contexte d'une RI appartiennent à *hcl*.
Post-act. *Modél. :*	▪ Créer *csi* ; ▪ Associer *hcl* à *csi* ; ▪ Associer les transactions à *csi ;* ▪ Associer les RIs à *csi.*

2.3.2. Transaction dans CSI

2.3.2.1. Associer une transaction au CSI

Paramètres :	▪ Le composant *csi* ; ▪ La transaction *tr*.
Pré-cond. :	▪ *tr* valide toutes les RIs de *csi* ; ▪ Toutes les classes appartenant au domaine de *tr* appartiennent à *hcl.*
Post-act. *Modél. :*	▪ Associer *tr* à *csi.*

2.3.2.2. Dissocier une transaction du CSI

Paramètres :	▪ Le composant *csi* ; ▪ La transaction *tr*.
Pré-cond. :	▪ Aucune.
Post-act. *Modél. :*	▪ Dissocier *tr* à *csi.*

2.3.3. Règle dans CSI

2.3.3.1. Associer une règle d'intégrité à un CSI

Paramètres :	▪ Le composant *csi* ; ▪ La règle d'intégrité *ri.*
Pré-cond. :	▪ Toutes les transactions de *csi* valident *ri* ; ▪ Toutes les classes du contexte de *ri* appartiennent à *hcl*.
Post-act. *Modél. :*	▪ Associer *ri* à *csi.*

2.3.3.1. Dissocier une RI du CSI

Paramètres :	▪ Le composant *csi* ; ▪ La règle d'intégrité *ri.*
Pré-cond. :	▪ Aucune.
Post-act. *Modél. :*	▪ Dissocier *ri* de *csi.*

2.3.4. Opérations sur les éléments d'un composant

Nous avons défini plusieurs ensembles d'opérations d'évolution d'éléments qui interviennent dans la définition d'un composant. Ces opérations ne sont pas atteignables au niveau d'un composant, mais leur exécution peut avoir un impact sur sa définition et sur sa validité. La Table 3 regroupe l'ensemble de ses opérations.

	Créer()	***Supprimer()***	***Mettre-à-jour()***
Diagramme	-	*Supprimer un diagramme*	-
	-	-	-
Hyperclasse		*Supprimer une hyperclasse*	-
	-	-	-
Nœud dans une hyperclasse	*Insérer une classe dans une hyperclasse*	*Exclure une classe d'une hyperclasse*	-
	R1	*R1-R2-R3*	-
Transaction	-	*Supprimer une transaction*	*Mettre à jour le traitement*
	-	-	*R3-R4*
Ressource	*Associer une ressource à la transaction*		
	R2		
Règle d'intégrité	-	*Supprimer une règle*	*Mettre-à-jour l'expression d'une règle* *Mettre-à-jour la réponse d'une règle*
			R5

Contexte	*Ajouter une classe/hyperclasse au contexte d'une règle*		
	R4-R5		
Risque	*Ajouter une hyperméthode au risque d'une règle*		
	R4-R5		

Table 3 : Opérations sur les éléments d'un composant

Dans le cas d'une suppression de diagramme, d'hyperclasse, de transaction ou de règle d'intégrité d'un modèle de SI, la réponse est dicté par la nature existentielle de ces concepts dans le méta-modèle de SI :

Supprimer un diagramme

Paramètres :	▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Supprimer tout composant de *D*.

Supprimer une hyperclasse

Paramètres :	▪ L'hyperclasse *hcl* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Supprimer le composant défini sur *hcl*.

Supprimer une transaction

Paramètres :	▪ La transaction *tr* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ dissocier *tr* de tout composant de *D*.

Supprimer une règle d'intégrité

Paramètres :	▪ La règle d'intégrité *ri* ; ▪ Le diagramme *D*.
Pré-cond. :	▪ Aucune.
Post-act. Supplémentaires sur le Modèle :	▪ Dissocier *ri* de tout composant de *D*.

2.4. INTEROPÉRABILITÉ DES COMPOSANTS DE SI

Le concept de composant de SI regroupe dans un même module l'espace informationnel d'une zone de responsabilité (représenté par son hyperclasse) et ses possibilités opérationnelles sur cet univers (transactions et règles d'intégrité). Le traitement de l'information (Figure 95) est considéré du (i) *point de vue métier* (en rouge) et du (ii) point de vue informationnel (en jaune).

Grâce au concept de composant de SI, il est possible de définir quelle zone de responsabilité est responsable de qu'elle partie de l'évolution de l'information dans le SI, et nous disposons alors d'une base pertinente pour la compréhension et la coordination des activités autour des informations dans le SI.

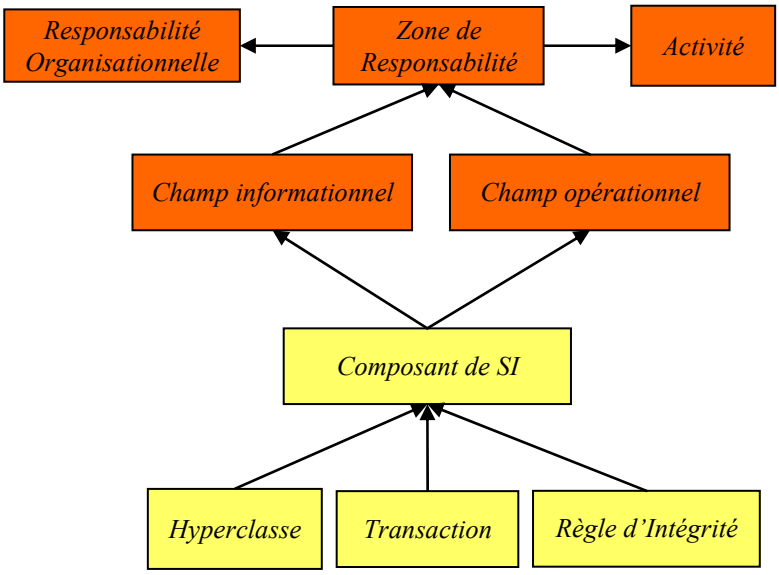

Figure 95 : Zones de responsabilité et composants de SI

Lorsque nous considérons plusieurs composants de SI, trois cas de figure peuvent se présenter (Léonard, 2002)[96] :

[96] (Léonard, 2002) adapté à partir de Léonard M., Composants de SI : Définition et Propriétés. Notes de cours, Université de Genève, 2002.

1. les composants de SI n'ont aucune classe, aucune transaction ni aucune règle d'intégrité commune ; ils mettent en évidence une *situation d'indépendance* ;

2. les composants de SI ont en commun des classes, mais aucune transaction ni aucune règle d'intégrité ; ils caractérisent une *situation de recouvrement uniquement structurel* (recouvrement de leurs hyperclasses respectives) ;

3. les composants de SI ont en commun des classes, des transactions et éventuellement des règles d'intégrité ; ils caractérisent une *situation d'interdépendance* ;

La *zone d'interdépendance* de plusieurs composants de SI est le sous-ensemble d'éléments du modèle de SI que les composants ont en commun.

L'existence d'une situation d'interdépendance indique alors que certaines problématiques que les zones de responsabilité mises en jeu ne doivent pas être envisagées de manière indépendante et qu'il faut préciser la nature des règles de coordination qui réglementent leurs interactions. De la même manière que pour le cas des hyperclasses, à partir du moment où il n'est pas possible de proposer de configuration du modèle de SI permettant d'éviter la présence de zones d'interdépendance, il devient nécessaire d'étudier et d'établir les protocoles adéquats de coordination des activités autour du SI.

Grâce au concept d'hyperclasse, il est possible de définir une partie de ces protocoles de coordination, notamment en spécifiant la visibilité des objets et des hyperobjets, des hyperobjets et des hyperattributs. En attribuant les composants de SI à des zones de responsabilité, nous finalisons les recouvrements et les protocoles de recouvrements. La répartition des transactions, responsables de l'évolution de l'information dans le SI, permet de réglementer la coordination des activités autour des informations exercées par les différentes zones de responsabilités utilisant les composants d'un SI.

En spécifiant un protocole de recouvrement, l'objectif consiste à permettre aux zones de responsabilités impliquées par une situation d'interdépendance d'assurer localement leurs activités.

3. MISE EN PRATIQUE DES COMPOSANTS DE SI

D'un point de vue pratique, nous travaillons, dans le cadre de la collaboration entre le groupe de recherche *MatisGe* et le *Centre des Technologies de l'Information*[97] (*CTI*) de l'*État de Genève*, sur l'ingénierie des SIs pour le e-gouvernement.

Le *CTI* gère environ 578 applications et projets de SI (CTI, 2004)[98] qui concernent les cinq piliers de l'*État de Genève* : l'éducation, la santé, l'économie, le social et la sécurité. Ces domaines et leurs SIs sont fortement interdépendants et exigent une grande rigueur dans les

[97] *http://www.geneve.ch/cti*

[98] (CTI, 2004) CTI, Rapport de Gestion 2004.

échanges et le partage de ressources d'information entre les services ; d'où la nécessiter de mettre en place des SIs transversaux pour supporter les différentes situations de recouvrement et maîtriser l'interopérabilité des différents SIs. Le *composant adresse* (Aïdonidis, 2004)[99] est un exemple de composant transversal à l'*État de Genève*. Actuellement, chaque service de l'état gère son propre fichier adresse. En plus des redondances et des incohérences qui existent dans ces fichiers, c'est au citoyen actuellement de contacter, un à un, chacun des services pour annoncer son changement de domicile. Le projet *composant adresse*, piloté par un groupe interdépartemental, vise à mettre en place un SI transversal centralisant les adresses des différents services. Le *service du cadastre* en sera l'institution responsable et en donnera accès aux autres services de l'état.

Le *composant adresse* est défini comme un composant de SI : (i) une hyperclasse décrit sa structure définie sur les classes Adresse, Adresse en Suisse, Adresse hors Suisse, Adresse du territoire genevois, Commune, Commune politique Bâtiment, Voie, etc. ; (ii) un ensemble de transactions indiquant les opérations possibles de recherche et de mise-à-jour possibles sur le composant ; et (iii) un ensemble de règles d'intégrités pour maintenir la cohérence des données. Les espaces informationnel et opérationnel dans le *composant adresse* de chaque service sont aussi représentés par des composants de SI. Des protocoles de recouvrement seront spécifiés et mis en place pour coordonner les activités autour du *composant adresse*.

Dans le domaine de l'ingénierie des SIs pour le e-gouvernement, on ne peut ignorer que les activités institutionnelles que supporte ces SIs sont définies par un cadre légal. C'est pour développer des SIs qui se conforment à se cadre que nous travaillons depuis plusieurs années sur l'ingénierie de SIs à base de composants de SI, extraits à partir des textes de lois (Turki & al., 2004)[100] (Khadraoui & al., 2004)[101]. L'Annexe K montre deux exemples de ces composants.

Cette démarche présente plusieurs avantages et s'inscrit dans une politique de développement durable. Elle rend possible l'ingénierie de SI en adéquation avec les activités de l'institution. En plus de posséder ses modèles, l'institution a la maîtrise de l'interopérabilité de ses SIs avec les différents niveaux de l'état : communes, cantons, confédération, union européenne, etc. Les lois induisent un consensus et leur existence n'est pas remise en cause par l'informatisation. Enfin, cette démarche a l'avantage d'expliciter les liens entre les lois et les SIs, et en particulier en cas d'évolution du cadre légal.

L'e-gouvernement est un domaine où les enjeux pour la société sont considérables et dont la portée va de la relation du citoyen avec l'administration jusqu'à la définition de la démocratie et de la citoyenneté. Les systèmes d'information y sont la clé de voûte et l'un des ses éléments les plus critiques. Il convient alors d'exiger la plus grande rigueur et la plus vaste maîtrise dans leur ingénierie.

---

[99] (Aïdonidis, 2004) Aïdonidis Ch., Le composant adresse à l'État de Genève. CTI, juin 2004.

[100] (Turki & al., 2004) Turki S., Aïdonidis C., Khadraoui A., Léonard M., Towards Ontology-Driven Institutional IS Engineering. EMOI-INTEROP 2004, CaiSE'04 Conf., Riga (Latvia), June 2004.

[101] (Khadraoui & al., 2004) Khadraoui A., Turki S., Aïdonidis C., Léonard M., Institutional Information Systems Alignment on Laws. AISTA 2004, IEEE, November 15-18, 2004.

CONCLUSION

Un composant de SI vérifie « (i) un critère de complétude logique : les composants d'un SI forment un tout cohérent du point de vue de la logique des informations qu'ils contiennent et traitent, et (ii) un critère d'interdépendance entre les composants : les situations de partage de l'information entre les composants (situations de recouvrements ou d'interdépendance) sont clairement identifiées et réglementées (par des protocoles de recouvrements) afin d'éviter au maximum les problèmes de coordination des activités autour des informations » (Léonard & Parchet, 1998)[102].

Ces propriétés des composants de SI ont rendu possible la construction du framework *ISIS*[103] proposé par (Le Dinh, 2005)[104] et dont le but est de gérer et de coordonner modèles informationnels utilisés dans les développements de SIs. Chaque SI est représenté par un composant de SI. Il est alors autonome et il est possible de gérer son interopérabilité avec d'autres SIs (voir l'Annexe J).

Dans le chapitre suivant, nous comparons nos travaux à d'autres travaux scientifiques proches.

---

[102] (Léonard & Parchet, 1998) Léonard M., Parchet O., Étude des situations de recouvrements de l'information pour la conception des SI. Actes d'INFORSID'98, Mai 1998, Montpellier, France.

[103] « *Information System upon Information System* » ou *SI sur SIs*.

[104] (Le Dinh, 2004) Le Dinh Th., Information System Upon Information Systems: A Conceptual Framework. Thèse de doctorat. Université de Genève, Novembre 2004.

CHAPITRE V

MISE EN PERSPECTIVE

INTRODUCTION

Ce chapitre est consacré au positionnement de nos travaux par rapport aux autres travaux scientifiques du même domaine. Cette comparaison se base sur les possibilités offertes à un responsable de SI d'obtenir des modèles informationnels de qualité, c'est-à-dire, rigoureux, non-ambigus, cohérents et capables d'évoluer.

Nous commençons par comparer le modèle *binex* au modèle binaire. Ensuite, nous comparons les diagrammes de classes dans *binex* aux diagrammes de classes d'*UML*[105]. Nous comparons brièvement le formalisme des graphes de *Gavroche* aux diagrammes d'états-transitions d'*UML* et aux réseaux de *Petri*. Nous comparons ensuite les concepts d'hyperclasse aux concepts de composant logiciel et de paquetage dans *UML*. Nous comparons également le concept d'hyperclasse aux concepts de relation universelle, de contexte sémantique, d'objet complexe et de vue. Enfin, nous présentons le modèle *ISIS* pour la gestion et la coordination de SIs en développement.

1. LE MODÈLE BINAIRE

« Le modèle relationnel binaire a connu ses heures de gloire en recherche au début des années 1970 par des contributions décisives de J.R. Abrial (Abrial, 1974)[106] et de M.E. Senko (Senko, 1973)[107]… » (Giraudin & Chabre-Pecoud, 1985)[108]. C'est « un formalisme de premier ordre qui combine les capacités des réseaux sémantiques, de l'intégrité logique et des règles de déduction » (Azmoodeh & al., 1984)[109]. Il représente « l'information d'un domaine comme une collection de faits élémentaires de deux types : (i) des faits unaires catégorisant les objets des faits réels du domaine et (ii) des faits binaires établissant les rapports de diverses sortes entre les paires d'objets » (Rishe, 1993)[110].

Les connexions entre les objets alors sont des « connexions bidirectionnelles (ou relations binaires) » (Abrial, 1974). Une relation binaire définit un lien atomique entre des paires d'objets appartenant à certaines catégories. Une relation binaire est définie par le nom de ses deux

---

[105] *Unified Modeling Language*, ou langage de modélisation objet unifié.

[106] (Abrial, 1974) Abrial J.R., Data semantics. J. W. Klimbie et K. L. Koffeman, IFIP TC2 Working Conference on Data Base Management, pages 1-59, Cargese (Corse), avril 1974. North Holland.

[107] (Senko, 1973) Senko M.E., Altman E.B., Astrahan M.m., Fehder P.l., Data Structures and Accessing in Data Base Systems. IBM Systems Journal, vol. 12, num. 1, 1973.

[108] (Giraudin & Chabre-Pecoud, 1985) Giraudin J.-P., Chabre-Pecoud M., Pour la réhabilitation du modèle relationnel binaire. Actes des journées Afcet-Informatique. Evry, octobre 1985.

[109] (Azmoodeh & al., 1984) Azmoodeh M., Lavington S. H., Standring M., The Semantic Binary Relationship Model of information. Proc. 7th ACM SIGIR conf. P.133-151, Cambridge, England., 1984. ISBN:0-521-26865-6.

[110] (Rishe, 1993) Rishe N., A Methodology and Tool for Top-Down Relational Database Design, Data Knowledge Engineering. Vol. 10", p. :259-291, 1993.

fonctions d'accès et les cardinalités minimales et maximales des ensembles qu'elle décrit. Une fonction d'accès est une fonction qui associe à une catégorie une partie d'une autre catégorie. « La pure approche binaire force à représenter les relations par la connexion d'objets deux à deux, même si plus d'objets sont impliqués. Ceci exploite le fait que les relations sont elles-mêmes des entités, qui peuvent alors à leur tour être reliées à d'autres » (Kent, 1984)[111].

Dans *binex*, les liens entre classes et les liens entre les objets de ces classes sont binaires : un lien entre classes relie une paire de classes ; un lien entre objets implique une paire d'objets. Toutefois, les liens dans *binex* sont de nature existentielle : un objet d'une classe dépendante reste lié au même objet d'une classe référencée durant toute son existence ; la suppression d'un objet d'une classe référencée entraîne la suppression de tout objet qui en dépend. Les liens de spécialisation-généralisation employé dans *binex* sont un cas particulier des liens existentiels binaires : ils portent la sémantique « *est un* ». Le méta-modèle *binex* est un modèle relationnel binaire où les liens sont de nature existentielle, et où il est possible de représenter des liens de spécialisation-généralisation.

« Ce « vieux » modèle (..) allie simplicité et puissance pour énoncer assez naturellement des aspects statiques de la sémantique des informations d'un système à étudier (..). La pratique de ce modèle qui associe une sémantique riche et une structure pauvre en fait un outil privilégié de spécification avant d'être aussi une aide ultérieure précieuse à la conception : (..) il laisse de nombreuses perspectives de solutions réalistes et réalisables selon le degré de détail (..). Le modèle relationnel binaire présente des qualités (..) : (i) flexibilité (..), (ii) neutralité (..), (iii) généralité (..) et (iv) simplicité (..) » (Giraudin & Chabre-Pecoud, 1985). « Cette structure présente beaucoup d'avantages : clarté conceptuelle, un niveau plus haut d'abstraction, mécanismes d'abstraction différents, proximité du langage naturel et raisonnement avec des spécifications formelles. Principalement, cette approche permet l'utilisation de bien plus de sémantique qu'une modélisation orientée objet « classique », sans sacrifier les fondations du paradigme objet » (Girow, 1996)[112]. « Par ailleurs, et ce n'est pas négligeable, le modèle binaire est le seul dont la conception autorise un passage automatique à un modèle relationnel normalisé (au sens de *CODD*) » (Sauge & Jeulin, 1988)[113].

2. *UML*: UNIFIED MODELING LANGUAGE

UML (*Unified Modeling Language*, ou *langage de modélisation objet unifié*) est un langage pseudo-formel pour la modélisation des systèmes orientés objet. *UML* se présente comme « un ensemble de modèles permettant de représenter un système informatique et son

[111] (Kent, 1984) Kent, W., Data and reality. 4ième edition, North-Holland, 1984 ; cité dans (Habrias, 1988).

[112] (Girow, 1996) Girow A., Binary Relations Approach to building Object Database Model. Object Currents Vol. 1, Issue 6, SIGS Publications, NY, 1996.

[113] (Sauge & Jeulin, 1988) Sauge P., Jeulin P., préface de (Habrias, 1988).

utilisation prévue dans l'entreprise » (Morley & al., 2000)[114]. Il propose une notation et permet de définir et de visualiser un modèle, à l'aide de diagrammes. Un diagramme *UML* est une représentation graphique, qui s'intéresse à un aspect précis du modèle.

« *UML* s'articule autour de neuf diagrammes différents, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel, par ailleurs, *UML* modélise le système suivant deux modes de représentation : l'un concerne la structure du système pris *au repos*, l'autre concerne sa dynamique de fonctionnement. Les deux représentations sont nécessaires et complémentaires pour schématiser la façon dont est composé le système et comment ses composantes fonctionnent entre elles » (Roques & Vallée, 2002)[115].

Le mode de représentation statique ou structurel s'appuie sur les cinq diagrammes suivants :

1. Diagrammes de cas d'utilisation ;
2. Diagrammes d'objets ;
3. Diagrammes de classes ;
4. Diagrammes de composants ;
5. Diagrammes de déploiement.

Le mode de représentation dynamique s'appuie sur les quatre diagrammes suivants :

1. Diagrammes de collaboration ;
2. Diagrammes de séquence ;
3. Diagrammes d'états-transitions ;
4. Diagrammes d'activités.

Dans la suite, nous positionnons les méta-modèles que nous avons adopté par rapport aux diagrammes de classes, aux diagrammes de composants et de déploiement, et aux diagrammes d'états-transitions.

3. DIAGRAMME DE CLASSES D'*UML*

« Le diagramme de classes représente le concept le plus important dans un développement orienté objet. Sur la branche fonctionnelle, ce diagramme est prévu pour développer la structure des entités connues des utilisateurs (..). En conception, le diagramme de classes représente la structure d'un code orienté objet, ou au mieux les modules du langage de

[114] (Morley & al., 2000) Morley Ch., Hugues J., Leblanc B., UML pour l'analyse d'un système d'information. Dunod, Paris, Février 2000. ISBN : 2-10-004826-0.

[115] (Roques & Vallée, 2002) Roques P., Vallée F., UML en action ; De l'analyse des besoins à la conception en Java. Eyrolles, Avril 2002. ISBN : 2-212-09127-3.

développement. » (Roques & Vallée, 2002)[116]. Il « exprime de manière générale la structure statique d'un système, en termes de classes et de relations entre classes » (Muller, 1998)[117].

Les concepts utilisés dans les diagrammes de classes d'*UML* sont détaillés dans l'Annexe G de ce document. Nous les comparons aux concepts utilisés dans les diagrammes de classes dans le modèle *binex* aux concepts des diagrammes de classe d'*UML*.

3.1. Visibilité des attributs et des opérations

Dans *UML*, le niveau de visibilité des attributs et des opérations est : (i) *Public*, qui rend l'élément visible à tous les clients de la classe, (ii) *Protégé*, qui rend l'élément visible aux sous-classes de la classe ou (iii) *Privé*, qui rend l'élément visible à la classe seule.

Dans *binex*, la visibilité des attributs et des opérations d'une classe n'est pas déterminée au niveau de la classe mais au niveau des zones de responsabilité (voir chapitre II, *les hyperclasses*) et des transactions (voir chapitre III, *les composants de SI*) qui utilisent cette classe. Par défaut, tous les attributs et toutes les opérations d'une classe sont disponibles aux autres éléments du diagramme de classes et le restent pour toutes les zones de responsabilité et les transactions qui utilisent cette classe. La décision de visibilité n'est pas ainsi prise de manière absolue au niveau de la définition d'une classe, mais de manière relative au niveau des zones de responsabilité et des transactions.

3.2. Les associations

Dans *binex*, le concept d'*association* n'est pas un concept particulier : il est assimilé au concept de *classe.* Les associations ont les mêmes propriétés que les classes (attribut, opération, lien existentiel, spécialisation-généralisation, cycles de vie, transaction, règle d'intégrité, etc.).

Dans *binex*, les associations sont fondamentalement de nature n-aire ; les associations binaires en sont un simple cas particulier. La notion de nommage d'association se confond avec celle de nommage de classe.

3.3. Multiplicité des associations

Dans *binex*, à travers un lien existentiel, un objet de son (sa classe) extrémité initiale est relié à un et un seul objet de son (sa classe) extrémité terminale. Par ailleurs, un objet de la classe référencée peut admettre zéro ou plusieurs objet de la classe dépendante qui lui sont reliés.

UML autorise toutes les combinaisons entre les six multiplicités possibles[118] à chaque extrémité d'une association. En particulier, *UML* autorise les associations de type *(N...M)* où

---

[116] (Roques & Vallée, 2002) Roques P., Vallée F., UML en action ; De l'analyse des besoins à la conception en Java. Eyrolles, Avril 2002. ISBN : 2-212-09127-3.

[117] (Muller, 1998) Muller P.-A., Modélisation objet avec UML. Eyrolles, Paris, Août 1998. ISBN : 2-212-08966-X.

[118] (i) *1*, (ii) *0..1*, (iii) *M..N*, (iv) *, (v) *0..** et (vi) *1..**.

chaque objet de l'une des extrémités de l'association peut être relié à plusieurs objets de l'autre extrémité, et vis-versa.

4. PAQUETAGE UML

UML propose la notion de paquetage[119] pour regrouper des éléments de modélisation et organiser les modèles.

« Les paquetages divisent et organisent les modèles de la même manière que les répertoires organisent les systèmes de fichiers. Chaque paquetage correspond à un sous-ensemble du modèle et contient, selon le modèle, des classes, des objets, des relations, des composants ou des nœuds, ainsi que les diagrammes associés. La forme générale du système (l'architecture du système) est exprimée par la hiérarchie de paquetages et par le réseau de relations de dépendance entre paquetages. Une relation de dépendance entre deux paquetages signifie qu'au moins une classe du paquetage client utilise les services offerts par au moins une classe du paquetage fournisseur. Toutes les classes contenues par un paquetage ne sont pas nécessairement visibles de l'extérieur du paquetage.

Un paquetage est un regroupement d'éléments de modélisation, mais aussi une encapsulation de ces éléments. À l'image des classes, les paquetages possèdent une interface et une réalisation. Chaque élément contenu par un paquetage possède un paramètre qui signale si l'élément est visible ou non à l'extérieur du paquetage. Dans le cas des classes, seules celles indiquées comme publiques apparaissent dans l'interface du paquetage qui les contient ; elles peuvent alors être utilisées par les classes membres des paquetages clients. Les classes de réalisation ne sont utilisables qu'au sein du paquetage qui les contient. Les relations de dépendance entre paquetages entraînement des relations d'arborescence entre les éléments de modélisation contenus dans ces paquetages » (Muller, 1998)[120].

« Un élément de modélisation n'appartient qu'à un seul paquetage. Il peut être en revanche être cité dans d'autres paquetages » (Morley & al., 2000)[121]. « Les éléments appartiennent aux packages où ils sont définis, mais ils peuvent être référencés dans d'autres packages. Une classe représentée dans un package étranger peut être modifiée par de nouvelles associations mais doit, pour le reste, demeurer inchangée » (Larman, 2003) [122].

---

[119] *Package* en anglais.

[120] (Muller, 1998) Muller P.-A., Modélisation objet avec UML. Eyrolles, Paris, Août 1998. ISBN : 2-212-08966-X

[121] (Morley & al., 2000) Morley Ch., Hugues J., Leblanc B., UML pour l'analyse d'un système d'information. Dunod, Paris, Février 2000. ISBN : 2-10-004826-0.

[122] (Larman, 2003) Larman C., UML et les Design Patterns. CampusPress, Collection Référence, juillet 2003. ISBN : 2744016233.

« Contrairement aux composants (*au sens UML*) qui se situent au niveau de l'implantation, un paquetage est un élément purement conceptuel qui existe uniquement durant le développement » (Booch & al., 1999)[123].

Bien que regroupant en une même entité plusieurs éléments d'un modèle, les concepts d'hyperclasse et de composant de SI, d'une part, et le concept de paquetage, d'autre part, sont très différents :

1. Tout d'abord, un paquetage est propriétaire de ses éléments, qu'il peut encapsuler. Une hyperclasse ou un composant est défini sur des éléments d'un modèle de SI (classes, liens, transactions ou règles d'intégrité) qui peuvent faire partie d'autres hyperclasses ou d'autres composants ;

2. un élément du modèle ne peut être modifié au niveau d'une hyperclasse ou au niveau d'un composant de SI ;

3. la relation de dépendance entre paquetages est basée sur l'échange de services entre leurs classes. Grâce aux propriétés de complétude des hyperclasses et aux règles de conformité des composants de SI, une hyperclasse ou un composant ne peut recourir aux « services » d'un élément du modèle de SI sur lequel il n'est pas défini. Si l'élément du modèle fait partie d'une autre hyperclasse ou d'un autre composant, c'est une situation de recouvrement ou d'interdépendance qui se révèle, et qui nécessite la mise en place des protocoles de coordination adéquats ;

4. enfin, une hyperclasse ou un composant de SI n'est pas simplement un élément conceptuel. Il est destiné à être implanté.

5. COMPOSANT LOGICIEL

Un composant logiciel est un logiciel, programme ou élément d'un logiciel ou d'un programme qui constitue un module indépendant utilisé comme élément d'un système plus complexe et qui est spécialement conçu pour fonctionner avec d'autres logiciels ou programmes. Ce sont des « entités logicielles autonomes qui présentent clairement les services qu'elles offrent et ceux qu'elles requièrent (chez d'autres composants) pour fonctionner. Ainsi un composant peut-être utilisé facilement sans qu'un développeur n'ait besoin d'en connaître son fonctionnement (...). Les composants offrent des services génériques afin de pouvoir être réutilisés simplement. Une application est construite en interconnectant plusieurs composants. Chacun d'entre-eux représente un aspect fonctionnel de l'application (un ensemble de service) » (Cariou, 2000)[124]. Les composants représentent des éléments physiques remplaçables ; Il peut s'agir par exemple de code source, d'un fichier binaire ou d'un exécutable, par exemple un

[123] (Booch & al., 1999) Booch, G., Rumbaugh, J., Jacobson, I., The unified modeling language user guide. Addison-Wesley, 1999.

[124] (Cariou, 2000) Cariou E., Spécification de Composants de Communication en UML. Objets, Composants, Modèles OCM'2000, Nantes (France), 18 Mai 2000.

navigateur ou un serveur http, une base de données, une *DLL* ou une archive *JAR* (comme pour un *EJB*) » (Larman, 2003) [125].

Dans la méthode *CATALYSIS* (D'Souza & Wills, 1998)[126], un composant est un module logique d'objets logiciels qui (i) peut être développé indépendamment ; (ii) qui dispose d'interfaces explicites pour les services qu'il fournit ; (iii) a les interfaces explicites pour les services qu'il attend des autres composants et (iv) qui peut se composer à partir d'autres composants, en adaptant quelques une de leurs propriétés, sans modifier les composants eux-mêmes. Un composant peut être indépendamment développé, fourni, et déployé comme unité.

Un composant de SI est un module défini à partir d'éléments du modèle de SI qui (i) peut être développé indépendamment ; (ii) qui dispose d'interfaces explicites pour les services qu'il fournit sous forme de transactions et d'hyperméthodes ; (iii) qui grâce aux règles de conformité et en particulier aux propriétés de complétude, contient tous les éléments du modèle dont il a besoin ; il n'attend aucun service des autres composants et (iv) qui peut se composer à partir d'autres composants, en intégrant leurs éléments. Un composant peut être indépendamment développé, fourni, et déployé comme unité. Lorsqu'un composant de SI partage un élément de modèle avec un autre composant, c'est une situation de recouvrement ou d'interdépendance qui se révèle, et qui nécessite la mise en place des protocoles de coordination adéquats.

Un composant logiciel est une partie physique et remplaçable d'un système qui fournit et se conforme à la réalisation d'un ensemble d'interfaces (..) tels que les composants COM+ ou des *Java Beans*, tout comme des composants qui sont des artefacts du processus de développement, tel que les fichiers de code source (..) » (Booch & al., 1999)[127]. Le modèle des *JavaBeans* est un modèle de composants logiciels basé sur le langage Java permettant de créer de manière visuelle dans un outil graphique des applications centralisées. Aujourd'hui apparaissent des modèles adaptés à la construction d'applications distribuées comme les *Entreprise Java Beans*, le *Corba Component Model* et *DCOM* » (Cariou, 2000).

« La notion de composant logiciel est très proche de celle d'objet logiciel dans le cadre de la programmation orientée objet. Il s'agit de modules logiciels distincts mais communiquant au moment de leur exécution pour concourir à l'application utilisée, sur une même machine, ou à travers un réseau » (Lumbroso, 2002)[128].

Un composant de SI n'est pas ni un objet ni un module logiciel ; il ne s'exécute pas. Il est utilisé dans la modélisation modulaire orientée objet d'un SI, et il est défini indépendamment de la méthode et de l'architecture logicielle ou matérielle dans laquelle le SI est implanté. Les concepts de composant logiciel et de composant de SI sont fondamentalement différents : le

---

[125] (Larman, 2003) Larman C., UML et les Design Patterns. CampusPress, Collection Référence, juillet 2003. ISBN : 2744016233.

[126] (D'Souza & Wills, 1998) D'Souza D., Wills A.C., Objects, Components and Framework with UML: The Catalysis Approach. Addison-Wesley, 1998.

[127] (Booch & al., 1999) Booch, G., Rumbaugh, J., Jacobson, I., The unified modelling language user guide. Addison-Wesley, 1999.

[128] (Lumbroso, 2002) Lumbroso L., Composant logiciel. 01net, Septembre 2002.

premier a vocation de modéliser des éléments techniques du système ; le second de modéliser la structuration et l'évolution de l'information dans le SI.

6. DIAGRAMME D'ÉTATS-TRANSITIONS & RÉSEAU DE PETRI

Le langage *UML* propose quatre types de diagrammes pour spécifier les vues dynamiques d'un système : (i) les *diagrammes de collaboration*, (ii) les *diagrammes de séquence*, (iii) les *diagrammes d'activités* et (iv) les *diagrammes d'états-transitions.* (Annexe H).

Un diagramme d'états-transitions est un automate d'états fini, où les états sont reliés par des arcs orientés qui décrivent les transitions. Il décrit les différents états possibles et les changements d'états d'un objet ou d'un composant *UML*, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs. Les constituants d'un diagramme d'états-transitions sont (i) les *événements* (stimuli externes ou internes), (ii) les *états* (valeurs des objets), (iii) les *changements d'états* (transitions) et (iv) les *opérations* (actions ou activités).

Les réseaux de Petri (décrits dans l'Annexe I) permettent, comme les diagrammes d'états-transitions, de représenter des événements, des états et des transitions. C'est un formalisme graphique très répandu dans la modélisation du comportement des systèmes. Il est fondé sur la représentation de relation d'accessibilité entre états comme les diagrammes d'états-transitions. Les constituants d'un réseau de Petri sont (i) les *places*, (ii) les *transitions,* (iii) les *arcs,* et (iv) la *fonction de marquage*. Les places et les transitions sont reliées par des arcs orientés. Un réseau de Petri est dit *graphe biparti alterné* : il y a alternance des types de nœuds ; tout arc, a obligatoirement avoir un nœud à chacune de ses extrémités, et relie soit une place à une transition soit une transition à une place. Un *arc* représente une fonction d'écoulement entre une place et une transition et vice versa.

Les *places* sont des endroits qui contiennent des jetons. A chaque place p_i correspondent ses transitions en entrée $°t_i$ et ses transitions en sortie $t_i°$. Elles mémorisent le nombre de jetons.

Les *transitions* représentent les endroits à travers lesquels le système change. Une transition t_j a des places en entrées $°p_i$ et des places en sorties $p_i°$, représentées par des arcs respectivement venant d'une ou plusieurs places et allant vers une ou plusieurs places.

Pour définir l'état d'un système modélisé par un réseau de Petri, il est complété par un *marquage* qui consiste à disposer un nombre entier (positif ou nul) de *marques* ou *jetons* dans chaque place du réseau de Petri. L'évolution l'état du réseau de Petri correspond à une évolution du marquage. Les jetons, qui matérialisent l'état du réseau à un instant donné, peuvent passer d'une place à l'autre par *franchissement* ou *tir* d'une transition. Le franchissement d'une transition consiste à retirer un jeton dans chacune des places en amont de la transition et à ajouter un jeton dans chacune des places en aval de celle-ci.

Les diagrammes d'états-transitions d'*UML* s'intéressent au comment des transitions en indiquant les événements qui les provoquent et les opérations à exécuter. Un diagramme d'états-

transitions est proche de l'implantation et ne permet pas une assez grande souplesse pour les manipulations des modèles dynamiques d'un SI.

« Si les conventions des réseaux de Petri sont bien adaptées aux modélisations de systèmes matériels, il nous semble que les conventions provenant des fonctions booléennes sont plus intéressantes pour la modélisation de SI » (Léonard, 1999)[129]. Contrairement aux réseaux de Petri, les conditions en entrée et en sortie d'une transaction ne sont pas de simples opérateurs booléens mais peuvent être des expressions booléennes complexes. Par ailleurs, le « ou » est exclusif dans les réseaux de Petri alors qu'il peut être inclusif ou exclusif dans les graphes de Gavroche selon la définition de la transaction.

Dans les graphes de *Gavroche*, « les transactions représentent des unités décisionnelles (que ce soit une activité, une tâche ou une décision) : les conditions sont internes aux transactions ce qui signifie qu'elles ne se trouvent pas sur le modèle du SI, ce qui laisse plus de place à l'évolution du schéma. Les conditions seront spécifiées avec la transaction et codées lors de l'implantation. En effet, pour adapter un schéma pour une entreprise, il est plus difficile de transformer un « et » en un « ou » en Petri qu'avec Gavroche. En utilisant les graphes de *Gavroche* les conditions étant internes, le modèle pour une application pourra être transmis tel quel pour plusieurs entreprises et il y aura uniquement les conditions à modifier » (Léonard, 1999).

Les graphes de Gavroche nous paraissent donc plus intéressants que les réseaux de Petri et les diagrammes d'états-transitions pour l'évolution et la manipulation des modèles dynamiques d'un SI. Par ailleurs, l'un des points forts des graphes de Gavroche est l'intégration dans un même modèle des modèles statiques et dynamiques d'un SI, traditionnellement séparés.

7. RELATION UNIVERSELLE

Le concept d'hyperclasse est inspiré du modèle de *relation universelle* introduit par *Ullman* au début des années 1980 ((Ullman, 1982)[130], (Maier & Ullman, 1984)[131]). L'objectif principal du modèle de relation universelle était d'atteindre l'indépendance complète des chemins d'accès dans les bases de données relationnelles, en déchargeant le concepteur du besoin d'une navigation logique à travers les relations.

Pour qu'une application puisse utiliser de manière adéquate le modèle de la relation universelle, il faut qu'elle satisfasse les trois conditions suivantes :

---

[129] (Léonard, 1999) Léonard M., Introduction aux systèmes d'information, notes de cours, Université de Genève, 1999.

[130] (Ullman, 1982) Ullman J.-D., Principles of Database Systems. Computer Science Press, Rockville, 1982.

[131] (Maier & al., 1984) Maier D., Ullman J.D., Vardi, M. Y., On the foundations of the Universal Relation Model. ACM Transactions on Database Systems, Vol. 9, No. 2, June 1984, Pages 283-308.

7.1. Hypothèse du schéma universel

Il existe un ensemble d'attributs formant un schéma de relation sur lequel on peut poser des interrogations. Ce schéma est appelé le *schéma universel de la base*. De plus. Chaque attribut joue un rôle unique : il ne peut représenter plusieurs ensembles de valeurs.

7.2. Hypothèse de l'association unique

Pour chaque ensemble d'attributs *X* il existe une unique association entre les éléments de *X* qui est celle que l'usager a en tête. Cela ne signifie pas qu'il n'existe pas d'autres associations mais que celle-ci est la plus « forte ».

7.3. Hypothèse de la saveur unique (« one flavour »)

Cette dernière hypothèse est essentielle pour comprendre comment un schéma de relation universelle doit être conçu. À moins que l'usager ne le spécifie explicitement, tous les chemins d'accès permettant de calculer la connexion *[X](d)* représentent la même saveur d'association entre les attributs de *X*. La signification du fait que le n-uplet *t* est dans la connexion de *X,* ne dépend pas des détails de construction dans *X*.

Selon le modèle de la relation universelle, « la base de données est perçue par l'usager comme une seule relation. On atteint ainsi un haut niveau d'indépendance logique ; l'usager n'a pas à connaître le schéma de la base pour pouvoir l'interroger » (Falquet, 1989)[132].

Il existe différentes formes de cette hypothèse qui conduisent à différentes interprétations de la relation universelle. L'idée essentielle de la relation universelle est que les chemins d'accès sont « embarqués » dans les noms des attributs. Ainsi, les noms d'attributs doivent jouer des « rôles » uniques. Plus encore, on suppose que pour chaque ensemble d'attributs, il existe une relation basique que le concepteur a en tête. Le concepteur se réfère à ces relations basiques pour ses requêtes plutôt qu'au schéma de base de données qui les supporte.

Cependant, « le modèle de relation universelle n'est pas toujours directement applicable sans perdre une partie de la sémantique portée par les schémas de la base. Ceci est particulièrement vrai lorsque la base ne satisfait pas les conditions énoncées (..). Lorsque le schéma de la base est complexe ou lorsqu'il atteint une taille importante, il devient très difficile d'admettre ces hypothèses. En d'autres termes, il n'est pratiquement plus possible de représenter toute la sémantique de la base dans une seule relation » (Falquet, 1989).

Les concepts sémantiques (Falquet, 1989) et les hyperclasses proposent des modèles dans lesquels différentes sémantiques sur les attributs peuvent être prises en compte. Dans ces modèles, une relation universelle locale est employée pour couvrir chaque sous-ensemble du schéma qui forme une unité sémantique. Les contextes sémantiques comme les hyperclasses sont ces parties du modèle de SI qui sont vues à travers une relation universelle. Chaque

---

[132] (Falquet, 1989) Falquet G., Interrogation de bases de données à l'aide d'un modèle sémantique. Thèse de doctorat, Université de Genève, Faculté des sciences, 1989

contexte ou chaque hyperclasse représente alors une et une seule sémantique pour l'association entre ses attributs.

8. LES CONTEXTES SÉMANTIQUES

Le concept de contexte sémantique a été introduit dans (Falquet, 1989)[133]. Il présente une solution pour assurer l'indépendance logique d'une base de données. Les fondements théoriques des contextes sémantiques trouvent leur origine dans la notion de relation universelle qui a été quelque peu transformée pour permettre la prise en compte de différentes sémantiques sur les attributs d'un schéma de base de données.

Un contexte sémantique est une abstraction sémantique du schéma. Un contexte inclut des caractéristiques structurelles et de comportement. Il est défini sur un groupe de classes liées par des fonctions (attributs) et représentant une sémantique particulière pour l'association entre les objets de ses classes. La sémantique du contexte est donnée par sa *fonction de connexion*. Il peut être représenté par un graphe dont les nœuds correspondent aux classes et les arêtes orientées correspondent aux attributs qui font le lien entre les nœuds.

Les concepts de contexte sémantique et d'hyperclasse sont proches. Les travaux de recherche menés dans le groupe *Matis Geneva Team* de l'*Université de Genève* dans le cadre de la construction du système de gestion de bases de données *F2*, constituent la base et le point de départ du concept d'hyperclasse. Voici quelques différences entre les deux concepts sont les suivantes (comparaison initiée dans (Neira, 1997)[134]) :

1. le concept d'hyperclasse et le concept de contexte utilisent le modèle de la relation universelle de manière locale ;

2. le concept de contexte sémantique se base sur le paradigme de la jointure, tandis que le concept d'hyperclasse et le concept sous-jacent d'hyperobjet se basent sur le paradigme de la navigation ;

3. dans un contexte sémantique, toutes les classes sont équivalentes ; elles jouent le même rôle. Une hyperclasse dispose d'une classe particulière qui est sa classe racine, point d'entrée de son graphe de navigation et dont les objets identifient les hyperobjets ;

4. le concept de contexte sémantique a été introduit dans le cadre des interrogations de bases de données avec des structures complexes. L'idée de contexte sémantique est de permettre au concepteur de montrer les différentes sémantiques existantes et leur utilité à l'intérieur d'une application. Chaque contexte représente une de ces sémantiques. Cette idée est préservée dans le concept d'hyperclasse : chaque hyperclasse présente une unité sémantique précise, sans ambiguïté ;

---

[133] (Falquet, 1989) Falquet G., Interrogation de bases de données à l'aide d'un modèle sémantique. Thèse de doctorat, Université de Genève, Faculté des sciences, 1989

[134] (Neira, 1997) Neira P., SGBDOO Extensible : Comment faire la spécialisation sans faire de l'héritage. Mémoire de Diplôme d'Études Supérieures en Systèmes D'information. Université de Genève, Septembre 1997.

5. pour interroger une base de données munie de contextes sémantiques l'usager devra (i) choisir un contexte dont la sémantique correspond à sa requête et (ii) interroger la relation universelle du contexte choisi.

Finalement, il faut remarquer que tous les avantages proposés par le concept de contexte sémantique sont préservés dans notre approche basée sur les hyperclasses.

10. LES OBJETS COMPOSITES

Un lien de composition relie une classe cl_1 à une classe cl_2 si les objets de cl_1, appelés *objets composites*, sont composés d'objets de cl_2, appelés *objets composants*. Ce type de lien est appelé lien de composition ou « *partie de* » (« *part-of* »). Les classes reliées par des liens « *partie de* » sont organisées en une hiérarchie appelée *hiérarchie composée de classe*. cl_1 est dite *classe composite* et cl_2 *classe composante*.

Les modèles de bases de données orientées objet existants proposent différentes variantes du lien de composition et parfois des contraintes d'intégrité qui peuvent lui être associées. Le concept d'objet composite a été défini dans le cadre du SGBD Orion (Kim & al., 1989)[135] :

1) un lien de composition peut être *exclusif* ou *partagé* : le lien est *exclusif* si un objet composant $cl_{i,1}$ de cl_1 ne peut faire partie que d'un seul objet composite $cl_{j,2}$ de cl_2 ; le lien est *partagé* si $cl_{i,1}$ peut fait partie à la fois de $cl_{j,2}$ et d'autres objets de cl_2 ;

2) une classe composante peut être, ou non, *dépendante* de sa (d'une de ses) classe composite. Si cl_1 est dépendante de cl_2, tout objet composant $cl_{i,1}$ de cl_1 est en dépendance existentielle de son objet composite $cl_{j,2}$ de cl_2 : la suppression de $cl_{j,2}$ entraîne automatiquement la suppression de $cl_{i,1}$;

3) certains systèmes de gestion de bases de données assurent la *contrainte inverse* : l'objet composite dépend de ses objets composants. Les contraintes de dépendance définissent l'ordre selon lequel les objets doivent être créés (et supprimés) : d'abord les composants ou d'abord le composite ;

4) des cardinalités peuvent être associées au lien de composition. La contrainte « cl_2 est une classe composante non partagée de la classe cl_1 », qui signifie qu'un objet de cl_2 ne peut être lié par un lien de composition qu'à un seul objet de la classe cl_1, s'exprime alors sous la forme d'une contrainte de cardinalité maximale à 1 sur le lien.

Les liens de composition peuvent servir comme une unité de groupement dans la base de données de façon à ce qu'une grande collection d'objets qui sont en relation les uns avec les autres, puisse être récupérée d'une façon efficiente de la base de données. Lorsqu'une

[135] (Kim & al., 1989) Kim W., Ballou N., Chou H-T., Garza J.F., Woelk D., Features of the ORION Object-Oriented Database System. Object-Oriented Concepts, Databases, and Applications. Ed. W. Kim and F.H. Lochovsky, ACM Press Frontier Series, New York, 1989.

application accède l'objet racine d'une hiérarchie de composition, il est possible d'accéder aussi à tous ses objets composants.

Bien qu'il présente quelques similitudes avec les concepts d'hyperclasse et d'hyperobjet, le concept d'objet composite présente des différences importantes :

1. un objet composite comme une hyperclasse fait partie du schéma de base de données ou de la spécification. Cependant, l'objet composite est défini au niveau d'une classe composite et d'un ensemble de classes composantes, tandis qu'une hyperclasse est définie sur un ensemble de classes connexe et complet reliées par les liens existentiels.
2. les hyperobjets et les objets complexes ont des identifiants qui sont les identifiants de leurs objets racine.
3. au niveau de la sémantique, les liens existants entre les classes d'une hyperclasse n'ont pas forcement la sémantique « partie de ». Dans l'objet composite, la sémantique de la relation qui lie la classe composite avec ses composants est exclusivement une relation de composition.
4. les hyperclasses permettent de montrer différents points de vue ou différentes perceptions d'un objet, tandis que l'objet composite donne une représentation unique d'un objet.

11. LES VUES

Une *vue* est une vision logique mise en forme des données contenues dans une ou plusieurs classes ou tables d'une base de données. Il existe plusieurs approches différentes pour définir et manipuler les *vues*.

À l'origine, une vue est seulement une *requête stockée* (une interrogation de la base de données) tel qu'elle est conçue dans le monde relationnel. Les données présentes dans une vue sont généralement définies grâce à une clause *SELECT*. Le résultat de la requête est considéré comme un *schéma externe*, un *schéma* ou une *classe dérivés* ou *virtuels*.

Un schéma externe, ou aussi *sous-schéma*, est défini par rapport à un groupe d'utilisateurs de la base de données. Il indique la partie du système à laquelle il a accès, structurée de façon à répondre à ses besoins spécifiques. Dans les SGBD actuels, le modèle de données employé pour décrire les schémas externes est le même que celui du schéma logique.

Une classe *dérivée* ou *virtuelle* offre une nouvelle interface pour accéder aux objets des classes de base et pour partager les données qui y sont stockées. Elle est définie par rapport à un ensemble de classes de base. Une classe de base est une classe du diagramme de classes qui possèdent ses propres objets. Seule la définition des classes dérivées est enregistrée. Elles n'emmagasinent jamais aucune donnée. Ainsi tous les objets d'une classe dérivée sont obtenus à partir de ses classes de base. Ces classes sont peuplées en sélectionnant des objets existants appartenant à d'autres classes.

Les objets d'une classe dérivée nécessitent un identifiant pour être semblables aux objets des classes de base. Si la vue a seulement une classe de base, les objets de la classe de base auront le même identifiant que les objets correspondants de la vue. S'il y a plus d'une classe de

base, il est possible de créer des identifiants pour les objets de la vue en fonction des identifiants des objets qui leur correspondent.

11.1. Mise à jour des vues

« Les aspects structurels et dynamiques ont été souvent séparés dans les SGBD et originellement les mécanismes de vues dans le modèle relationnel concernaient seulement les aspects statiques : les vues permettaient facilement une interrogation de la base, mais pas la mise à jour » (Date, 1986)[136]. Dans la plupart des SGBD (relationnels ou orientés objet) la modification des objets à travers les vues est limitée au seul cas des vues avec une seule classe de base. Autrement, les modifications permises sur les objets de la vue sont très limitées, et ceci à cause des ambiguïtés possibles.

Cependant, le SGBD *Oracle©* propose depuis sa version *Oracle9i*, la possibilité de réaliser des insertions, des mises à jour ou des suppressions sur des vues définies sur plusieurs tables (« *join views* ») mais avec plusieurs restrictions. La mise à jour de vue définie sur une seule table est possible seulement à certaines conditions:

1. Aucun attribut calculé dans le *SELECT*,
2. Aucun *DISTINCT*,
3. Aucun opérateur d'agrégation,
4. Aucun *GROUP BY*,
5. Aucun opérateur ensembliste: *UNION*, *INTERSECT*, *DIFFERENCE*,
6. Aucune sous-requête dans le *SELECT*,
7. Aucune jointure.

Pour les vues définies sur plusieurs tables, la mise à jour est possible si (i) elle n'affecte qu'une seule des tables de la jointure, et (i) si les colonnes modifiées appartiennent à une « *key-preserved table* ». Une « *Key-preserved table* » est une table dont l'identifiant (clé) primaire est aussi un identifiant (clé) primaire du résultat de la jointure. S'il y a l'option « *WITH CHECK OPTION* », la mise à jour d'une colonne utilisée dans la condition de jointure est impossible et seules les lignes vérifiant la vue sont mises à jour.

Dans une vue définie avec une jointure, la suppression est possible seulement si la jointure n'a qu'une seule « *key preserved-table* » et cette table doit apparaître une seule fois dans la vue.

L'insertion est possible si toutes les colonnes dans lesquelles sont insérées des valeurs proviennent d'une table dont l'identifiant (clé) primaire est préservé. S'il y a une *CHECK OPTION*, l'insertion n'est pas possible.

---

[136] (Date, 1986) Date C.J., An introduction to Database Systems. Addison Wesley Longman, 7th edition, 1999. ISBN: 0201385902

11.2. Vues versus Hyperobjets

1. Une vue, dans le modèle relationnel, peut être considérée comme une table virtuelle qui n'a pas d'existence réelle et qui est définie par une interrogation de la base. La vue est définie au-dessus du schéma de base de données. Une hyperclasse fait partie du schéma de la base, et peut être vu comme un sous-schéma ;

2. Une vue est un concept statique utile pour réaliser des consultations, des requêtes, mais pas des mises à jour. Les opérations d'insertion, de suppression et de mise à jour ne sont pas toujours possibles sur une vue. La propagation des mises à jour faites sur une vue à la base réelle n'est pas toujours permise, souvent pour des raisons d'ambiguïté. Par contre, un hyperobjet peut être consulté, mis à jour ou supprimé. Dans une hyperclasse, il est possible de créer, de supprimer ou de mettre à jour des objets des classes des hyperclasses, dans cadre du modèle *binex* ;

3. Il est possible de définir des hyperméthodes sur des hyperclasses, mais pas sur des vues.

4. Dans une vue, l'identifiant de la relation de base n'est pas toujours présent. Par contre, dans les hyperclasses, les attributs identifiants et de référence sont toujours accessibles.

5. « Dans les vues orientées objet, le problème des mises à jour existant dans les vues relationnelles disparaît, car les objets ont une identité indépendante des valeurs que prennent les objets » (Scholl & al., 1991)137. Dans une hyperclasse, ce problème ne se pose pas, car ce sont les objets réels du SI qui sont manipulés.

CONCLUSION

Les concepts d'hyperclasse et de composant de SI sont originaux par les possibilités qu'ils offrent pour l'ingénierie des SIs. Un composant de SI intègre, de manière cohérente et rigoureuse, des spécifications statiques, dynamiques et règlementaires d'un SI. Le concept hyperclasse, en tant que généralisation du concept de classe, permet de manipuler des ensembles de classes comme une seule classe, et de traiter leurs objets comme un seul objet. Enfin, la simplicité des éléments qui interviennent dans la définition d'une hyperclasse ou d'un composant rend maîtrisable leur évolution, leur intégration et leur interopérabilité, et assure la traçabilité d'un concept de l'espace des activités jusqu'à sa représentation informatique.

Dans le chapitre suivant, nous présentons l'ensemble des réalisations informatiques qui supportent actuellement notre démarche et les concepts que nous utilisons.

---

[137] (Scholl & al., 1991) Scholl M.H., Laasch C., Tresch M., Updatable Views in Object-Oriented Databases. Proc. Int. Conf. on Deductive and Object-Oriented Databases, DOOD, Munich 1991.

CHAPITRE VI

INGÉNIERIE & RÉALISATIONS

INTRODUCTION

Lors de ce travail de recherche, nous avons mené plusieurs projets de développement afin d'appuyer notre démarche. Ces développements ont été réalisés autour de la plate-forme *M7*, pour créer un environnement de modélisation des SIs supportant le modèle *binex*, les graphes de *Gavroche* et les concepts d'hyperclasse et de composant de SI.

Dans ces développements, nous avons dû tenir compte des prototypes et systèmes légués tout en intégrant les architectures technologiques et les langages émergeants.

Nous présentons la plate-forme *M7* et trois de ses extensions en rapport avec nos travaux : (i) un serveur de composant de SI, (ii) *Taïchi*, un générateur de poste de travail à partir de graphes de *Gavroche* et (iii) *Stella*, un navigateur de base de données générique.

1. LA PLATE-FORME *M7*

M7 est un outil CAISE[138] construit sur la base de sept espaces conceptuels : (i) *classes*, (ii) *hyperclasses*, (iii) *règles*, (iv) *cycles de vie*, (v) *périodes*, (vi) *évènements* et (vii) *traitements*.

Son développement a débuté en 1996 dans le cadre d'un projet de transfert de technologie entre l'*Université de Genève* et la société de services informatiques *Proconcept SA*, financé par la *Commission pour la Technologie et l'Innovation* du *Département Fédéral de l'Économie Publique, Suisse*. Le projet était baptisé *Noyau d'un Système d'Information Générique* et avait pour objectif de « développer un noyau générique actif et permanent qui rende possible et facilite la coordination des activités de conception, de développement, d'administration et d'évolution des SIs » (Léonard & al., 1998)[139].

Schématiquement, *M7* est un dictionnaire formé des sept espaces et permettant de piloter une base de données gérée par un SGBD commercialisé, *Oracle* en l'occurrence. Son développement est passé par plusieurs étapes.

La première version de *M7* a vu le jour en 1998. Elle a nécessité un peu plus de 21'200 lignes de code source *Delphi3*, un peu plus de 12'000 lignes de code source *Oracle* (*SQL* et *PL/SQL*), un référentiel de 50 tables, une centaine de vues et une vingtaine de packages de procédures stockées. Elle comprenait trios modules : (i) *M7 Navigator* (Figure 96) : pour la saisie des spécifications, (ii) *M7 Generator* : pour la génération du référentiel de spécification dans *Oracle 8i* et (iii) *M7 Monitor* : pour l'évaluation et le test de la cohérence de la spécification.

---

[138] *Computer Assisted Information System Engineering*

[139] (Léonard & al., 1998) Léonard M., Estier Th., Parchet O., Ramos F., Rimbert R., Schnegg P-A., Somers J., Noyau d'un Systèmes d'Information Générique. Rapport scientifique et technique final. Projet CTI nº 3326.1, Décembre 1998.

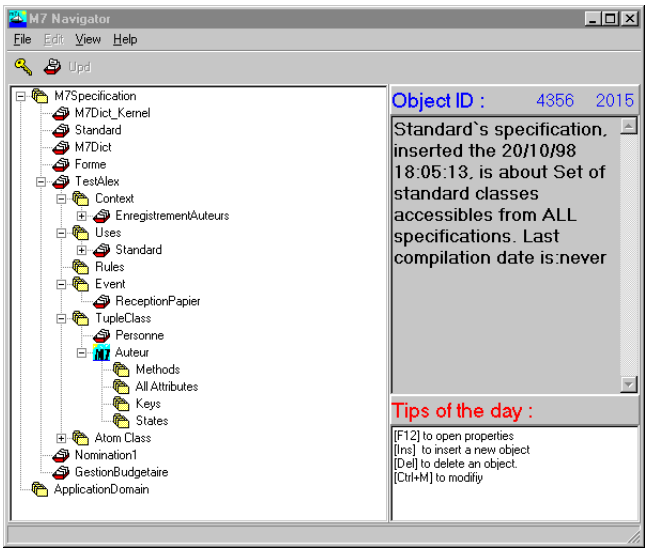

Figure 96 : Interface de *M7 Navigator*

En 1999, le concept d'hyperclasse est venu remplacer le concept de contexte, initialement utilisé dans *M7* ; ceci a nécessité l'adaptation notamment du référentiel et du générateur de *M7* (Fournier, 1999)[140]. Ensuite, des interfaces pour la définition des hyperclasses (Figure 97 et Figure 98) et les algorithmes de calcul des hyperobjets ont été implanté en Java dans *M7* (Buloz, 2000)[141].

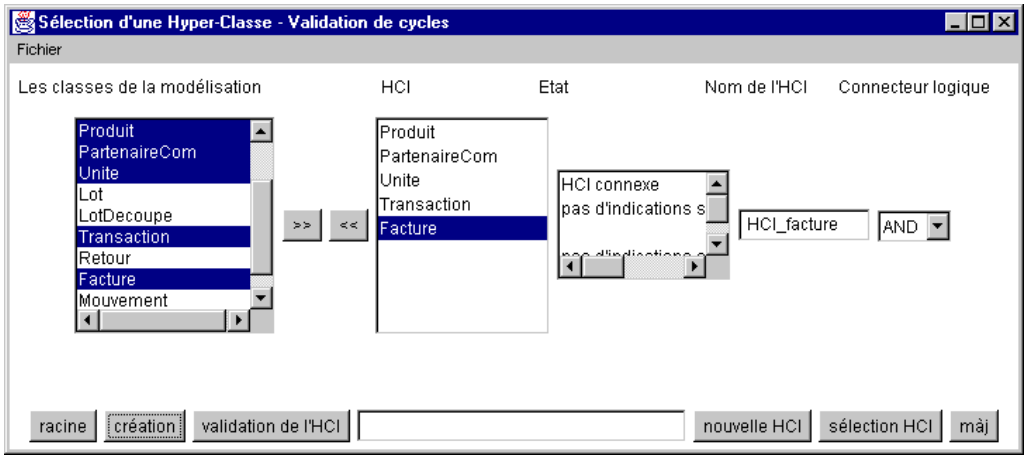

Figure 97 : Exemple d'interface pour la définition d'une hyperclasse

[140] (Fournier, 1999) Fournier N., Élaboration et Implémentation des Hyperclasses dans M7. Mémoire de Licence en Systèmes d'Information et de Communication. Septembre 1999, Université de Genève.

[141] (Buloz, 2000) Buloz d., Première mise en place des hyperclasses dans un SGBD commercialisé. Mémoire de Diplôme en Informatique. Université de Genève, Juillet 2000.

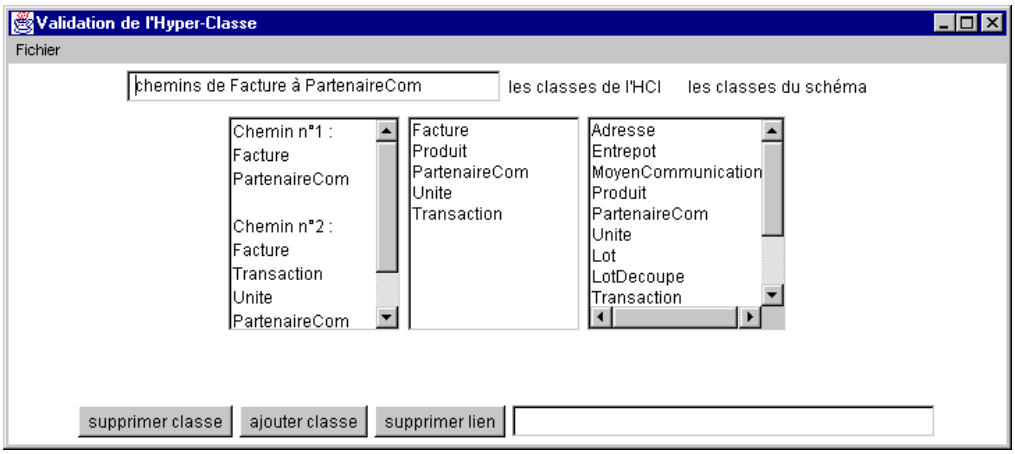

Figure 98 : Exemple d'interface pour la définition de la navigation dans une hyperclasse

Les premières primitives d'évolution d'hyperclasse, définies dans (Turki, 2001)[142], permettant de créer et de supprimer une hyperclasse, d'intégrer une hyperclasse dans une autre hyperclasse, d'exclure une classe d'une hyperclasse et de consolider un modèle de SI en cas de suppression et classe, ont été implantées dans M7 en Java. Une démonstration des possibilités d'évolution et de conservation des hyperméthodes a été organisée lors du module *Évolution des Systèmes d'information* de la formation *Interactive-Matis* en juin 2000.

En 2001, l'outil *M7* a été complètement revisité (Figure 99) et reconstruit autour d'une architecture distribuée à base de composants logiciels, de l'utilisation de la spécification *J2EE* de *Sun* et des composants *Enterprise Java Beans*. Ceci a permis de mettre en oeuvre une architecture coopérative, intéropérable, fiable et performante.

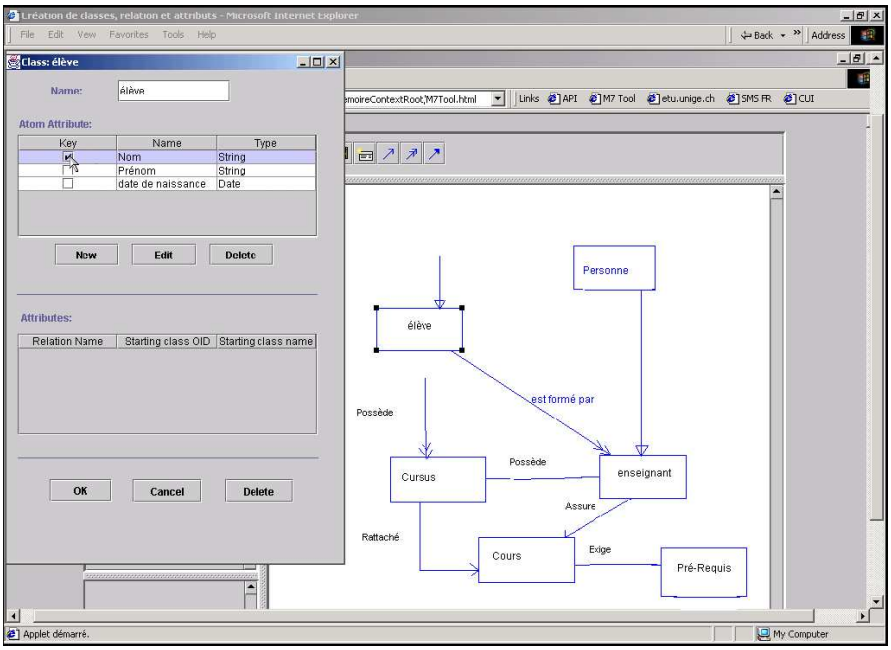

Figure 99 : Nouvelle interface de *M7*

[142] (Turki, 2001) Turki, S, Introduction aux hyperclasses. Proc. Inforsid'2001, P. 281-299, ISBN: 2-906855-17-0. Martigny, Suisse, Mai 2001.

L'utilisation de cet environnement distribué a induit une structuration de la plate-forme en plusieurs niveaux principaux (Figure 100) (Snene & Léonard, 2003)[143]. Ces niveaux sont :

1. le *niveau présentation*, contient les différentes interfaces applicative utilisateurs ;
2. le *niveau applicatif*, exécute les traitements relatifs aux données issues du niveau présentation ;
3. le *niveau service*, valide les différentes informations obtenues suite à leurs traitements ;
4. le *niveau domaine,* opère comme une mémoire transitoire avant la sauvegarde définitive des données tout en assurant leur validation sémantique, et
5. le *niveau persistance,* assure le stockage et la validation syntaxique des données.

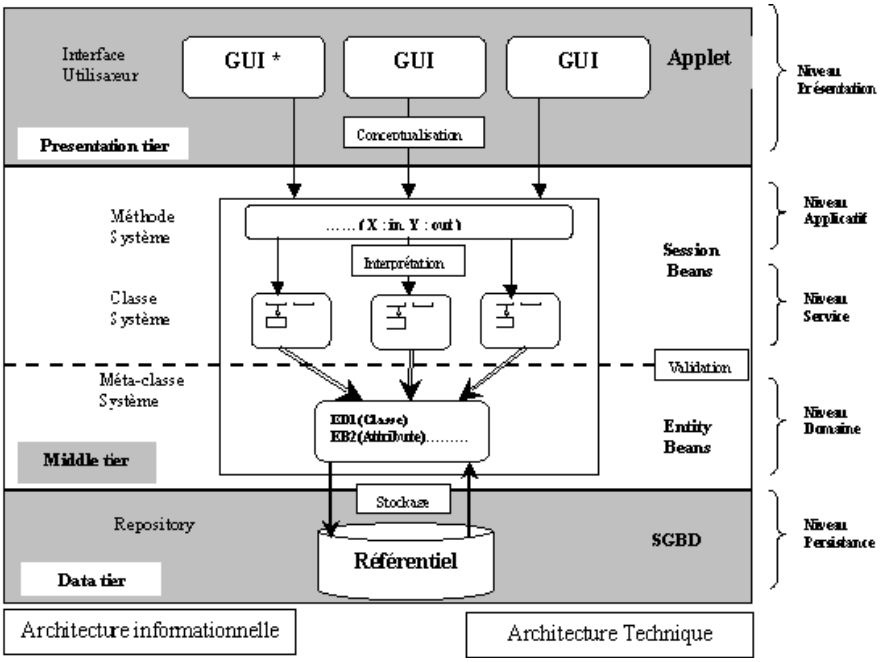

Figure 100 : Architecture distribuée de *M7*

2. SERVEUR DE COMPOSANTS

Le *Serveur de Composants* (Turrini & Gonçalves, 2002)[144] est une application web, construite autour de la plate-forme *M7*, qui permet de créer, de visualiser et de manipuler une spécification *M7* avec ses hyperclasses et ses composants de SI (Figure 101).

[143] (Snene & Léonard, 2003) Snene M., Léonard M., Distributed framework for real time web based collaboration: M7Tool case. Proc. ACS/IEEE Conf. Computer Systems and Applications. Tunis, Tunisia, July 2003.

[144] (Turrini & Gonçalves, 2002) Turrini G., Gonçalves R., Création d'un serveur de Composant de SI. Mémoire de Licence en Systèmes d'Information et de Communication. Université de Genève, Novembre 2002.

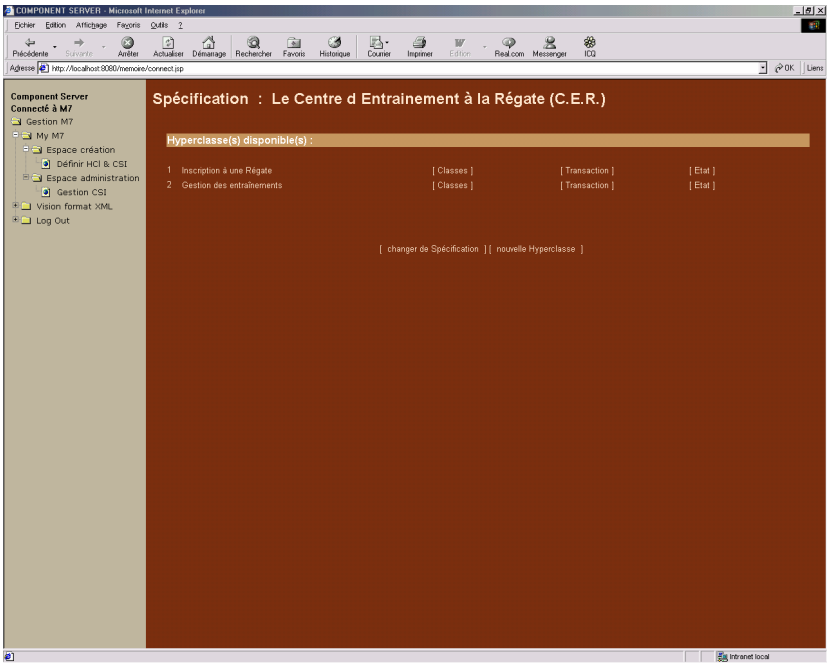

Figure 101 : Interface du *Serveur de Composants*

L'application a été développé en *JSP*[145] sur une plate-forme logicielle basée sur *Tomcat HTTP Server* (*Apache*) et *Oracle*. L'utilisation de *Java Beans* dans les pages *JSP* a notamment permis de maintenir une séparation entre les données de présentation (affichées à l'utilisateur) et l'implantation des traitements (le code). Le serveur de composant interagit avec le dictionnaire de *M7*, qui est un schéma particulier du serveur Oracle (SGBD) sur lequel est installé *M7*.

Le *Serveur de Composants* permet d'exécuter les opérations suivantes sur un modèle de SI :

Au niveau des transactions

- Créer une transaction
- Supprimer une transaction
- Renommer une transaction
- Mettre à jour la pré-condition d'une transaction
- Mettre à jour la post-condition d'une transaction
- Mettre à jour le traitement d'une transaction
- Associer une classe de base à une transaction
- Dissocier une classe de base d'une transaction
- Associer une classe de rajout à une transaction

[145] *Java Server Pages* : langage permettant le développement des pages web dynamiques, créée au moment où la page web est accédée.

- Dissocier une classe de rajout d'une transaction

Au niveau des composants de SI

- Créer un composant
- Associer une règle d'intégrité à un composant,
- Dissocier une règle d'intégrité d'un composant
- Renommer un composant
- Modifier la description d'un composant

Les règles de conformité des composants de SI ont été implantées comme des règles d'intégrité, vérifiées et validées à chaque exécution d'une opération qui constitue un risque pour la conformité du composant.

Nous avons aussi initié le développement d'un *SQL-like language* pour la manipulation d'un modèle de SI. Voici quelques exemples de commandes :

```
CREATE csi ON HYPERCLASS hcl
ALTER COMPONENT csi ADD TRANSACTION tr
ALTER COMPONENT csi DROP TRANSACTION tr
ALTER COMPONENT csi ADD INTEGRITY RULES ri
ALTER COMPONENT csi DROP INTEGRITY RULES ri
ALTER COMPONENT csi RENAME nouv_compoName VARCHAR
ALTER COMPONENT csi CHANGE DESCRIPTION nouv_compoDesc VARCHAR
```

3. TAÏCHI

Taïchi (Zheng, 2003)[146] est une application développé à partir des travaux de (Pham, 2003)[147]. Cette application permet, à partir d'un modèle de SI utilisant les liens existentiels et la spécialisation dynamique pour les aspects statiques, et le formalisme des graphes de Gavroche pour les aspects dynamiques, de générer un noyau de SI. Ce noyau comprend une base de données munie :

1. des interfaces permettant de (i) consulter, (ii) créer, (iii) supprimer et (iv) mettre à jour tout objet d'une classe du SI, et

[146] (Zheng, 2003) Zheng Y., TAÏCHI, vers un SGBD intégrants les aspects statiques et dynamiques. Mémoire de Diplôme d'Études Avancées en Systèmes d'Information. Université de Genève, Septembre 2003.

[147] (Pham, 2003) Pham T.-T.-T., Intégration des aspects statiques et dynamiques des Systèmes d'Information. Mémoire préliminaire, Université de Genève, 2003.

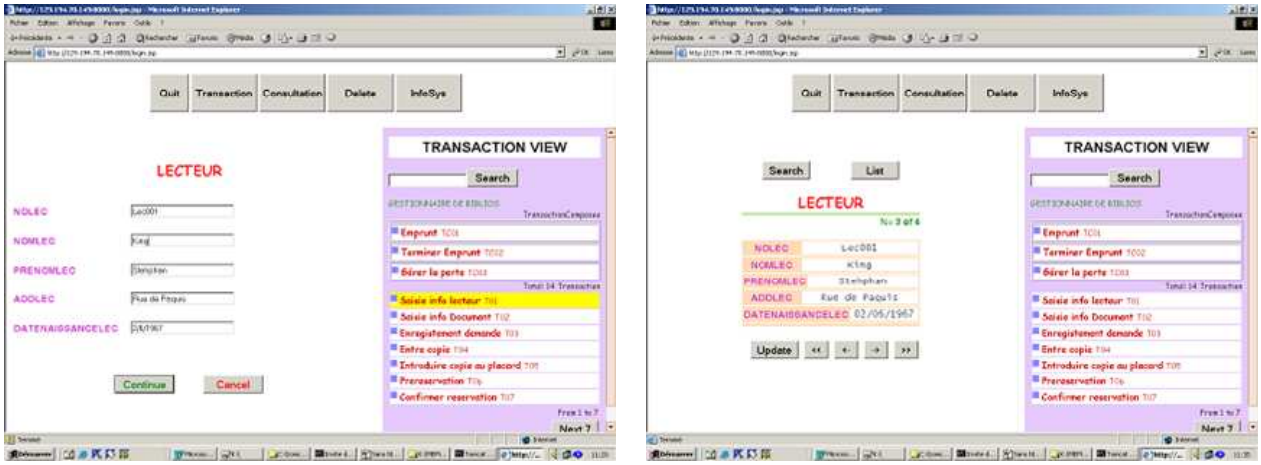

Figure 102 : Exemple d'interfaces de création et de consultation d'objet générées par *Taïchi*

2. d'une interface pour chacune transaction *tr* du SI proposant l'ensemble des objets des classes de base de *tr* vérifiant les pré-conditions de *tr*, et les boutons déclenchant les traitements liés à *tr*.

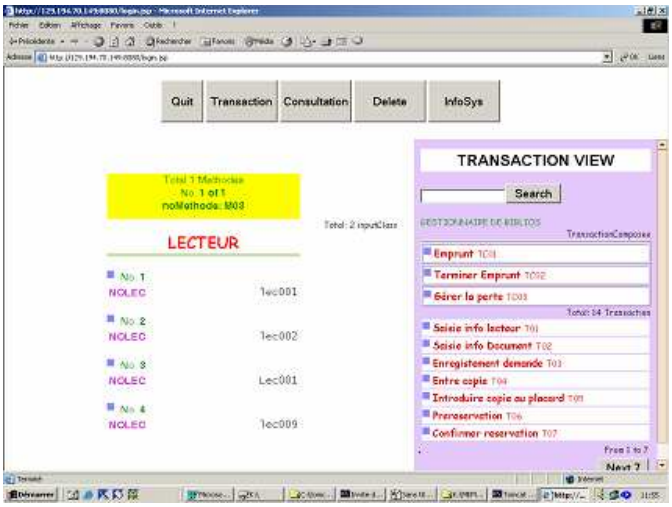

Figure 103 : Exemple d'interface d'une transaction

Taïchi interagit avec le dictionnaire *M7* pour lire les modèles de SI et générer les interfaces nécessaires.

L'application a été développée en *JSP* et tourne sur un serveur *TOMCAT*.

4. STELLA

Stella est une application web permettant de consulter, de manipuler et de naviguer entre les objets d'une base de données (Lopez & Mustabasic, 2004)[148]. Développée en *JSP* et tournant

[148] (Lopez & Mustabasic, 2004) Lopez L. R., Mustabasic R., Navigateur de Bases de Données *STELLA*. Mémoire de Licence en Systèmes d'Information et de Communication. Université de Genève, Septembre 2004.

sur un serveur *TOMCAT, Stella* peut être utilisée sur toute base de données disposant de driver *JDBC*[149], en particulier les *SGBD* Oracle et *MySQL*.

Pour garantir ses fonctionnalités sur plusieurs SGBD de manière générique, *Stella* interroge le dictionnaire de données du SGBD pour connaître la structure de la base (les classes et les liens qui existent entre elles), en utilisant les méthodes fournies par les *API* de *JDBC*.

Stella offre plusieurs fonctionnalités en relation avec la navigation entre les données :

1. *Stella* permet de visualiser toutes les classes d'un modèle (Figure 104) et d'en consulter le contenu. Lors de la sélection d'une classe, elle affiche toutes les informations qui lui sont rattachées comme son type, ses identifiants et ses liens avec d'autres classes.

Figure 104 : Affichage des informations relatives à une table

2. Pour chaque objet d'une classe sélectionnée, *Stella* affiche l'ensemble des objets qui lui sont reliés (Figure 105). À partir d'un objet sélectionné, il est possible d'atteindre par navigation et de consulter tout autre objet d'une classe adjacente qui lui est relié ;

[149] *Java Database Connectivity* : API de JAVA permettant à une application JAVA de se connecter à une base de données

Figure 105 : Affichage d'un objet et des objets qui lui sont liés

3. Au niveau de chaque classe, il est possible de créer de nouveaux objets (Figure 106) ou de supprimer (Figure 107) ou de mettre à jour (Figure 108) des objets existants dans la base.

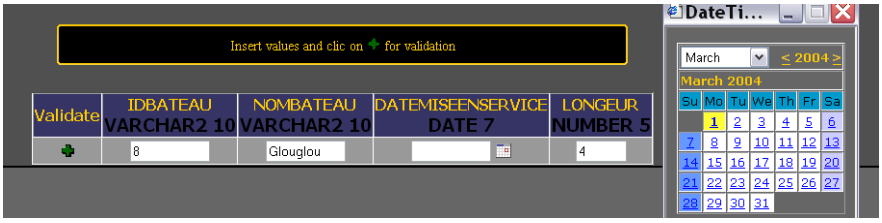

Figure 106 : Exemple de création d'objet dans *Stella*

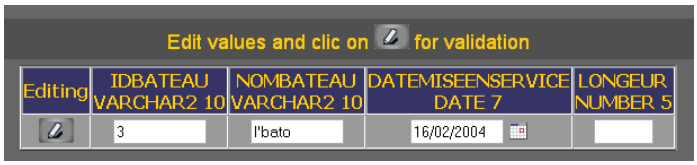

Figure 107 : Exemple d'édition d'objet pour mise-à-jour dans *STELLA*

Figure 108 : Exemple de suppression d'objet dans *Stella*

Stella est la solution que nous avons adoptée pour disposer d'une interface générique et portable pour la consultation et la manipulation à la fois des objets du méta-modèle de *M7* (dictionnaire de *M7*) et les objets des modèles générés à partir de spécifications dans *M7*. Ce choix nous assure une liberté dans l'évolution du méta-modèle dont les coûts de l'adaptation des interfaces existantes ou du développement de nouvelles interfaces nous privaient.

CONCLUSION

Depuis quelques mois, nous étudions la faisabilité de la reconstruction de la plate-forme *M7* en *Python*[150] et en utilisant *ZODB*[151] pour le stockage des objets. Le but est d'aboutir à un environnement d'ingénierie des SIs dont le dictionnaire de données nous serait complètement accessible ; en construisant nos applications autour du SGBD *Oracle*, nous sommes soumis aux limites de visibilité et d'évolution qu'impose *Oracle*.

Nous avons aussi fait le choix de privilégier les applications *Open Source* : nous adoptons tant que possible les logiciels libres et tous nos développement sont mis à la disposition du public.

Bien que ne disposant pas d'une plate-forme opérationnelle, nous avons réussi à employer la plupart des concepts développés dans cette thèse dans le cadre de plusieurs projets de SI. Les notions d'espace informationnel et d'espace opérationnel, la logique de navigation entre les objets atteignables d'un espace informationnel (hyperobjet) et le concept d'hyperméthode, ont été à la base de *M@TIS* et des SIs qui ont servi à piloter les comités de programmes des conférences *Inforsid'2000* et *Inforsid'2001*. Les composants de SI relatifs à l'inscription dans M@TIS et à l'évaluation des modules par les étudiants ont été adapté et intégrés à d'autres SIs académiques tels que le SI du *Département des Systèmes d'Information* de l'*Université de Genève*, et la SI de la formation continue *interactive-MATIS*.

[150] Langage de programmation orienté objet, *http://www.python.org/*

[151] *Zope Object Database*

CONCLUSION

CONTRIBUTIONS & PERSPECTIVES

« Les SIs actuels couvrent des domaines d'activités de plus en plus étendus et complexes, de sorte qu'il est devenu difficile de les appréhender dans leur globalité que ce soit du point de vue de leur conception, de leur utilisation ou même de leur administration » (Léonard & Parchet, 1998)[152]. « L'étendue des activités que doit supporter le SI d'une entreprise ou d'une institution fait qu'il est de plus en plus difficile d'obtenir une vue globale pertinente du SI, de distinguer ses différentes parties, et d'identifier les recouvrements entre ces parties » (Léonard, 2003)[153]. Dorénavant, il est indispensable de raisonner en termes de composants et en termes de recouvrement entre ces composants. Pour faire face à cette complexité, sans cesse croissante, nous avons construit un cadre conceptuel basé sur les concepts d'hyperclasse et de composant de SI.

1. CONTRIBUTIONS

Dans ce travail de recherche, nous avons proposé un cadre pour l'ingénierie des systèmes d'information par composants. Ce cadre, construit à partir des concepts d'hyperclasse et de composant de SI, est générique, rigoureux, non dédié à une technologie, une méthode ou un SI particulier.

Les hyperclasses et les composants de SI sont définis à partir des éléments de diagrammes de classes exprimés en *binex*, de graphes de Gavroche et de règles d'intégrité.

Binex est un méta-modèle pour la modélisation orientée objet de l'information dans un SI. Il a la particularité de n'autoriser que les liens existentiels et de spécialisation-généralisation entre ses classes.

Le formalisme des graphes de *Gavroche* permet de décrire le comportement des objets des classes de *binex* : un graphe de *Gavroche* est un graphe biparti où les nœuds correspondent un à un à des *classes* et les étoiles à des *transactions*. La notion de *transaction* utilisée est inspirée des concepts de « *transaction commerciale* » et de « *transaction bancaire* » et est associée à une activité productrice ou consommatrice d'informations dans un processus de prise de décision.

Le modèle règlementaire décrit les règles d'intégrité définies sur un SI pour garantir sa cohérence durant son exploitation.

Une hyperclasse est définie sur un sous-diagramme de classes connexe et complet. Un hyperobjet de l'hyperclasse est formé à partir des objets des classes de l'hyperclasse, atteints par navigation, à partir d'une classe particulière de l'hyperclasse qui est sa classe racine, et en

---

[152] (Léonard & Parchet, 1998) Léonard M., Parchet O., Étude des situations de recouvrements de l'information pour la conception des SI. Actes d'INFORSID'98, Mai 1998, Montpellier, France.

[153] (Léonard, 2003) Léonard M., IS Engineering Getting out of Classical System Engineering. In Proc. ICEIS'03, Angers, France, pp. 35-45, 2003.

suivant un graphe de navigation. Un hyperattribut de l'hyperclasse est un attribut de l'une de ses classes, atteint via un chemin d'accès précisé. Une hyperméthode est une méthode associée à l'hyperclasse qui peut avoir comme opérandes des hyperattributs, des hyperobjets, d'autres hyperméthodes de l'hyperclasse, les classes de l'hyperclasse, leurs objets et leurs méthodes de classes.

Le concept d'hyperclasse est une généralisation du concept de classe. Construite sur un ensemble de classes, une hyperclasse permet d'exprimer des concepts que le niveau de classe n'aurait permis d'exprimer, et se comporte comme une classe : elle dispose d'hyperobjets, d'hyperattributs et d'hyperméthodes, équivalents des concepts d'objet, d'attribut et de méthode pour une classe.

Le concept d'hyperclasse introduit aussi une forme d'indépendance entre la structure du SI et ses traitements, en particulier lors de situations d'évolution. Dans certaines conditions, il est possible de préserver les hyperméthodes des dysfonctionnements dus à l'évolution de la définition d'une hyperclasse : (i) les hyperméthodes restent opérationnelles ; nous parlons de *conservation d'hyperméthode,* et (ii) continuent à retourner le même résultat ; nous parlons de *conservation du résultat*.

Une hyperclasse décrit, au niveau d'un modèle de SI, l'espace informationnel nécessaire à une zone de responsabilité pour assurer ses activités. Un composant de SI fournit une représentation à la fois des espaces informationnel et opérationnel nécessaires à la zone de responsabilité.

Un composant de SI est construit à partir d'une hyperclasse, d'un ensemble de transactions et d'un ensemble de règles d'intégrité du SI. C'est une entité autonome dans un modèle de SI dont la cohérence est garantie par un ensemble de règles de conformité.

Les concepts d'hyperclasse et de composant de SI sont construits à partir d'éléments conceptuels simples, définis dans le modèle *binex*, le formalisme des graphes de *Gavroche* et le modèle des règles d'intégrité. La nature élémentaire de ces éléments rend sûres les étapes d'analyse, de conception et d'implantation du SI. Elle préserve aussi une traçabilité du concept métier jusqu'à aux objets informatiques. Munis d'ensembles complets d'opérations d'évolution, ces concepts permettent de maintenir l'alignement du SI avec le métier qu'il supporte.

Les concepts d'hyperclasse et de composant de SI constituent un moyen pour détecter et prendre en charge les situations de recouvrements entre zones de responsabilités autour du SI. La définition de protocoles de recouvrement permet à chaque zone de responsabilité de coordonner ses activités avec d'autres zones de responsabilité, et d'assurer localement ses propres activités.

2. PERSPECTIVES

Actuellement, de nombreuses possibilités de développement des concepts d'hyperclasse et de composant de SI sont offertes. Chacun de ces concepts peut encore être enrichi par la

définition formelle notamment des relations de spécialisation-généralisation et de version (ou variantes) d'hyperclasse, respectivement de composant de SI.

Les opérations d'extraction et d'intégration d'hyperclasses peuvent également être étendues aux composants de SI, et permettre à terme la mise en place d'une ingénierie de patrons de conception ou de domaine. D'une façon similaire à l'intégration d'hyperclasses, l'intégration de composant peut nécessiter d'adapter localement le composant au modèle dans lequel il est intégré : il peut s'agir d'ajouter, de supprimer ou de mette à jour certaines de ses classes, ses transactions ou ses règles d'intégrité, ou de faire évoluer son hyperclasse. L'adaptation globale du composant peut s'avérer nécessaire pour gérer les situations de recouvrement ou d'interdépendance que provoque son intégration. Grâce à la fois aux ensembles complets d'opérations d'évolution des hyperclasses et des composants de SI, et des mécanismes que ces concepts offrent pour la gestion de l'interopérabilité d'espaces informationnels et opérationnels, nous disposons des moyens nécessaires pour l'adaptation locale et globale des composants en vue de leur intégration.

Tels qu'ils ont été définis, les concepts d'hyperclasse et de composant de SI sont indépendants de la méthode et de l'architecture logicielle ou matérielle dans laquelle le SI est développé. Toutefois, nous envisageons la construction de fragments ou composants de méthode guidant les processus de développement de SIs construits à base de composants de SI. En effet, même s'il existe à ce jour plusieurs méthodes d'ingénierie de SI, ces méthodes ne sont pas universelles et elles ne peuvent pas prévoir toutes les situations possibles. La situation de développement de chaque SI étant différente, l'utilisation de blocs de base pour la construction des méthodes « à la volée » (Ralyté & al., 2003)[154], par assemblage de composants, nous semble la solution la plus intéressante. Un premier ensemble de composants de méthode peut déjà être défini pour le guidage de la définition et de l'évolution des hyperclasses et des composants de SI.

À terme, notre projet est d'aboutir à un environnement d'ingénierie des SIs complètement modulaire, tant au niveau produit qu'au niveau processus, basé sur les concepts de composants de SI et de composant de méthode.

[154] (Ralyté &al., 2003) Ralyté J., Deneckère R., Rolland C., Towards a Generic Model for Situational Method Engineering. Proc. of CAISE'03, Velden, Austria, 16 – 20 June 2003

DOCUMENTS ANNEXES

ANNEXE A

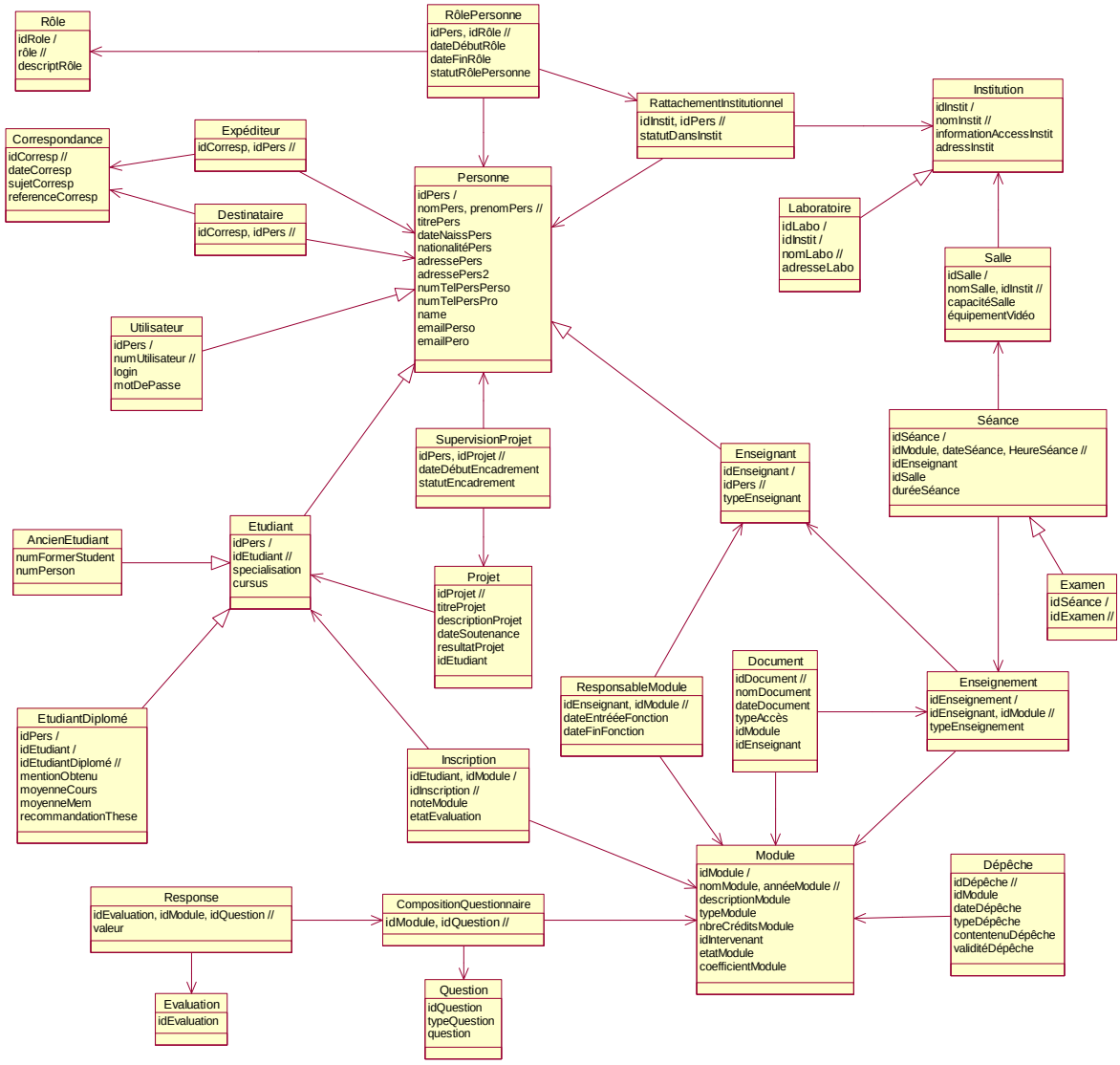

Figure 109 : Diagramme de classes de *M@TIS*

ANNEXE B

RAPPELS DE DÉFINITIONS DE LA THÉORIE DES GRAPHES

Ces définitions sont reprises et adaptées à partir de (Berge, 1970)[155] et de (Léonard, 1988)[156].

1. Graphe

Un graphe $G=(N, U)$ est le couple constitué par :

1. un ensemble de nœuds $N=\{n_1\ n_2 \ldots\ n_n\}$;
2. une famille $U=\{u_1\ u_2 \ldots\ u_m\}$ d'éléments du produit cartésien $N \times N$, où un élément (n_i, n_j) peut apparaître plusieurs fois.

Un graphe est dit *simple* si un élément de la famille U n'y apparaît qu'une seule fois.

Si le graphe n'est pas orienté, un élément $u = (n_i, n_j)$ de la famille U est appelé *arête* dont n_i et n_j sont les extrémités.

Si le graphe est *orienté*, un élément $u=(n_i, n_j)$ de la famille U est appelé *arc* dont n_i est l'extrémité initiale et n_j est l'extrémité terminale. Un arc est une arête particulière. Deux arcs (ou arêtes) sont *adjacents* s'ils ont au moins une extrémité commune.

2. Chaîne

Une *chaîne* est une séquence $ch = (u_1, u_2, \ldots , u_q)$ d'arêtes de G telle que chaque arc de la séquence a une extrémité en commun avec l'arc précédent, et l'autre extrémité en commun avec l'arc suivant.

On dit alors que ch est une chaîne reliant n_1, l'extrémité de u_1 qui n'est pas commune à u_2, et n_q, l'extrémité de u_q qui n'est pas commune à l'arête précédente. n_1 et n_q sont les extrémités de la chaîne.

Une chaîne est simple si elle ne rencontre pas deux fois le même nœud ; elle est élémentaire si elle n'utilise pas deux fois la même arête.

3. Chemin

Un chemin $ch = (u_1, u_2, \ldots , u_p)$ est une chaîne formée d'arcs, où pour tout arc u_i ($i<p$) l'extrémité terminale de u_i coïncide avec l'extrémité initiale de u_{i+1}.

---

[155] Berge C., Graphes et hypergraphes, ISBN 2-04-016906-7, Dunod informatique, 1970.

[156] Léonard M., Structures des bases de données, ISBN 2-04-016407-3, Dunod informatique, 1988.

n_1 étant l'extrémité initiale de u_1, n_p l'extrémité terminale de u_p, on dit alors que c'est un chemin de n_1 à n_p. n_1 et n_p sont respectivement les extrémités initiale et terminale du chemin.

4. Cycle

Un *cycle* est une chaîne $ch = (u_1, u_2, \ldots, u_p)$ telle que :

- la même arête (arc) ne figure pas deux fois dans la séquence ;
- les deux nœuds aux extrémités de la chaîne coïncident.

5. Circuit

Un *circuit* est un cycle qui est formé par un chemin.

Deux chaînes sont *différentes* si elles ne contiennent aucune arête en commun.

6. Bassin

Dans un graphe $G=(N, U)$, un bassin de n_1 est un ensemble $E = \{u_1, u_2, \ldots, u_n\}$ d'éléments de U tels que :

- ces éléments sont des arêtes ;
- et l'une d'entre elles admet n_1 comme extrémité ;
- et pour chaque extrémité de ces arêtes n_i, il existe une chaîne reliant n_1 et n_i, formée d'éléments de E.

7. Écoulement

Dans un graphe $G=(N, U)$, un *écoulement* du nœud n_1 est un ensemble d'éléments $E = \{u_1, u_2, \ldots, u_n\}$ de U tels que :

- ces éléments sont des arcs ;
- et l'un d'entre eux admet n_1 comme extrémité initiale ;
- et pour chaque extrémité n_i de chacun de ces arcs, il existe un chemin de n_1 à n_i, formé d'éléments de E.

Les extrémités des éléments du bassin de n_1 (ou d'un écoulement de n_1) sont les nœuds du bassin (ou de l'écoulement).

Nous introduisons les définitions suivantes :

8. Graphe connexe

Un graphe $G=(N, U)$ est dit *connexe* si et seulement si quels que soient les nœuds n_i et n_j de N, il existe un chemin de n_i à n_j. G ne doit pas contenir de nœud isolé ou de section du graphe isolée.

9. Nœuds adjacents

Dans un graphe $G=(N, U)$, deux nœuds n_i et n_j sont dits adjacents si $(n_i, n_j) \in U$ ou $(n_j, n_i) \in U$.

10. Nœud directement prédécesseur

Le nœud n_i est dit directement prédécesseur du nœud n_j dans G si et seulement s'il existe un arc de n_i vers n_j, en d'autres termes, si $(n_i, n_j) \in U$.

11. Nœud directement successeur

Le nœud n_j est dit directement successeur du nœud n_i dans G si et seulement s'il existe un arc de n_i vers n_j, en d'autres termes, si $(n_i, n_j) \in U$.

12. Nœuds prédécesseur et successeur

Le nœud n_j est dit *successeur* du nœud n_i dans G si et seulement s'il existe une séquence d'arcs partant de n_i vers n_j, telle que chaque arc de cette séquence a une extrémité en commun avec l'arc précédent, et l'autre extrémité en commun avec l'arc suivant.

Le nœud n_j est dit *successeur* du nœud n_i dans G si et seulement s'il existe un chemin partant de n_i vers n_j. Le nœud n_i est alors dit prédécesseur du nœud n_i dans G.

13. Nœud source

Dans un graphe orienté, un nœud est dit *source* s'il n'est l'extrémité terminale d'aucun arc.

14. Nœud puits

Dans un graphe orienté, un nœud est dit *puit* s'il n'est l'extrémité initiale d'aucun arc.

ANNEXE C

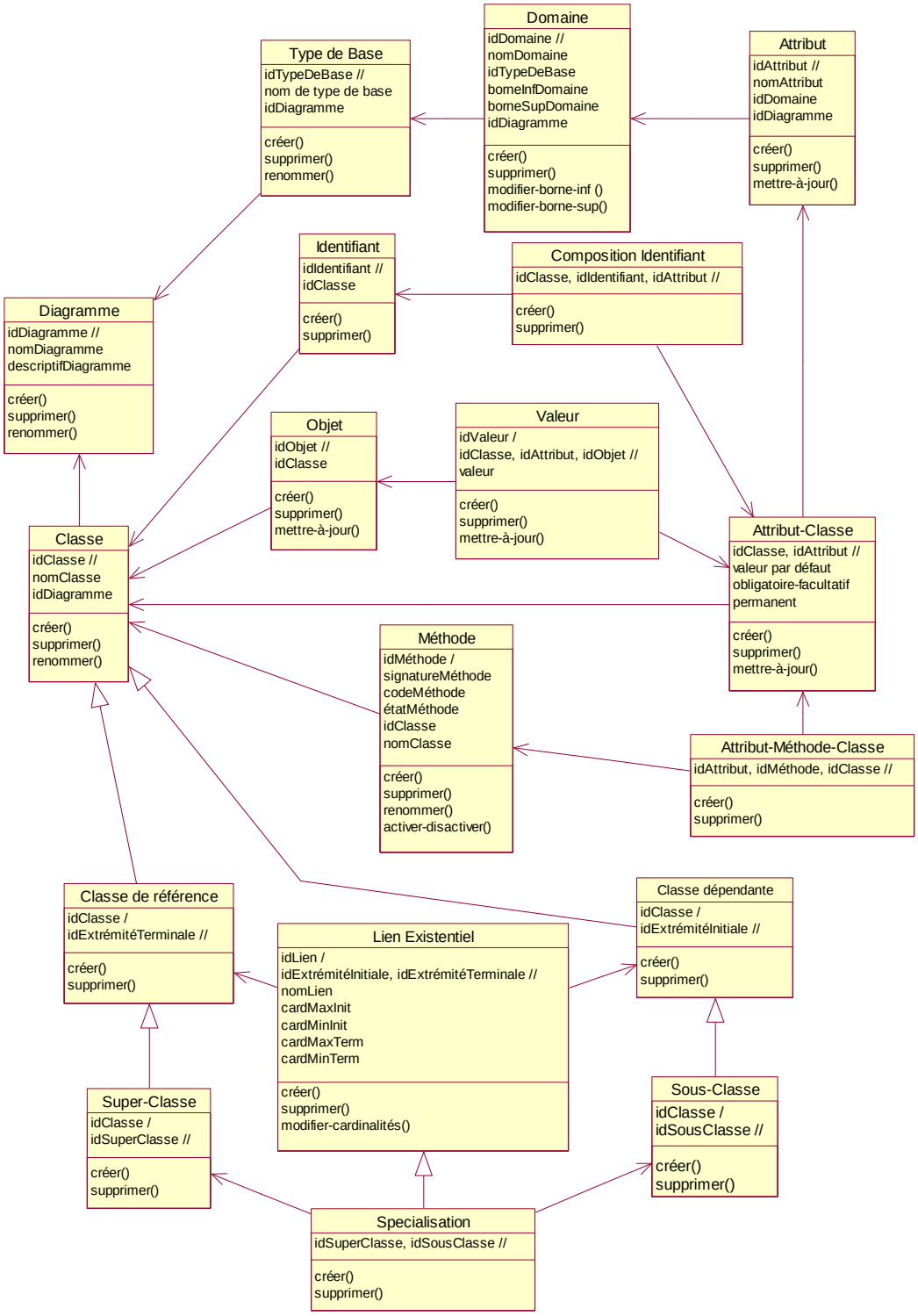

Figure 110 : Méta-modèle *binex*

ANNEXE D

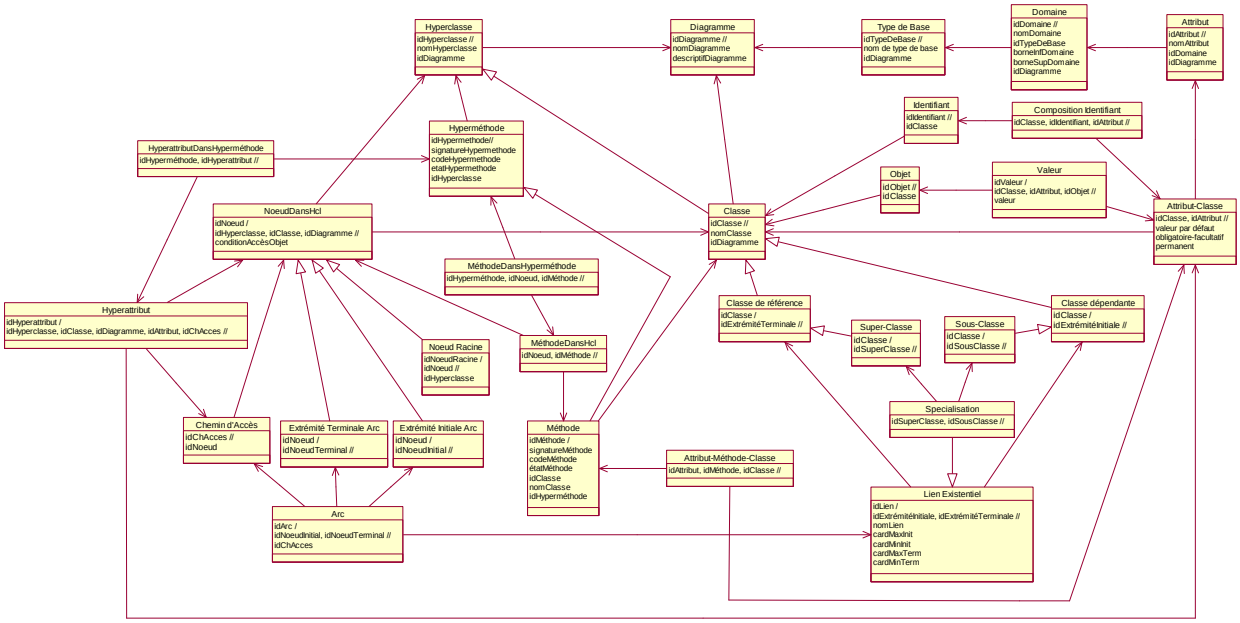

Figure 111 : Méta-modèle d'*hyperclasse*

ANNEXE E

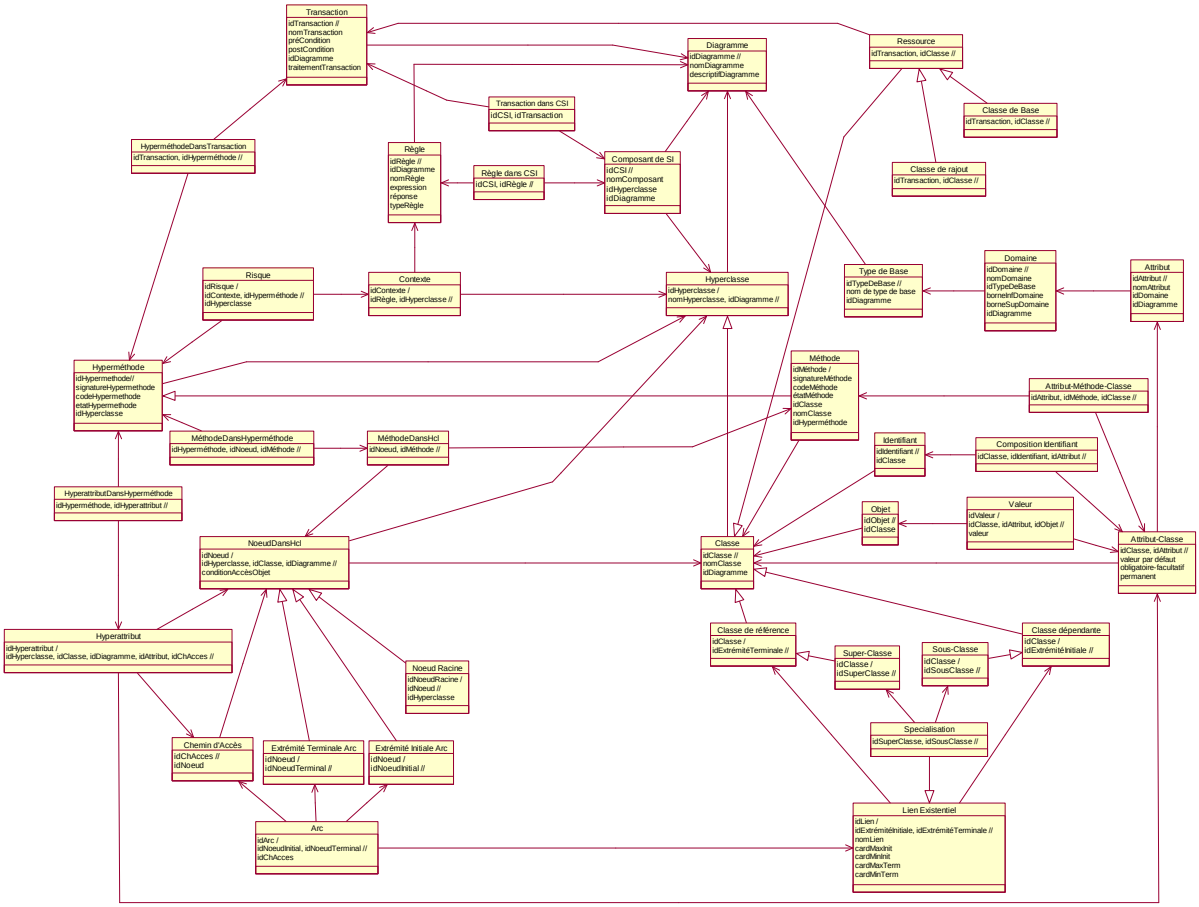

Figure 112 : Méta-modèle de SI

ANNEXE F

LES HYPERCLASSES DANS *M@TIS*

Autour de la formation *MATIS*, nous avons identifié cinq catégories d'acteurs : (i) *étudiant*, (ii) *intervenant*, (iii) *responsable de module*, (iv) *coordinateur* et (v) *technicien* (Giraudin & al., 2002)[157]. Ces cinq catégories d'acteurs sont les cinq sommets d'un pentagone fait apparaître les relations inter-acteurs dans un système d'information pédagogique (Figure 113).

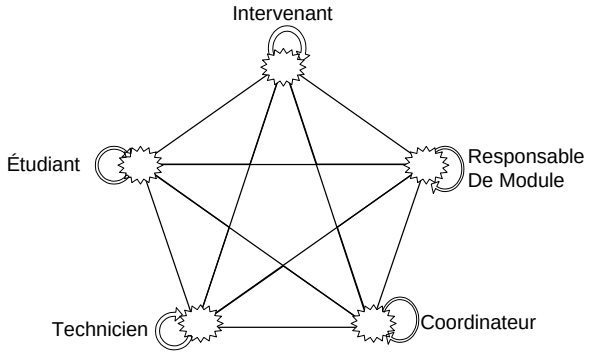

Figure 113 : *Pentagonie*, schéma des collaborations entre les acteurs de *M@TIS*

Chacune des cinq catégories d'acteurs de la formation dispose d'un espace informationnel qui décrit l'espace informationnel qui leur est attribué dans le cadre de leurs interactions avec le système. Ces espace informationnels sont représentés sous forme d'une ou plusieurs hyperclasses au niveau du diagramme de classes.

Dans M@TIS, chaque acteur est responsable du maintien à jour des informations personnelles et de contact qui le concernent. L'hyperclasse *hcl_acteur* (Figure 114) décrit l'espace informationnel représentant ces informations. Elle est construite sur les classes *Personne*, *RattachementInstitutionnel*, *Institution* et *Laboratoire*.

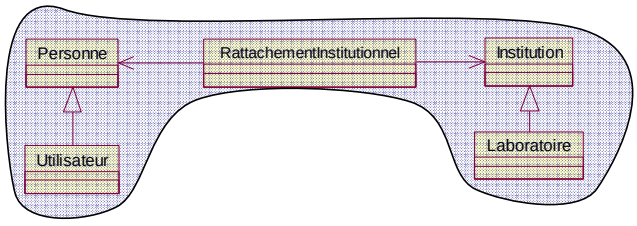

Figure 114 : Diagramme de l'hyperclasse *hcl_acteur*

[157] Giraudin J.-P., Freireire Junior J-C., Saïdane M., Turki S., Assessment of a difficult voyage in Pentagonie - Evaluation of a pedagocical intranet project. Proc. of the International Conference on Engineering Education 2002, Manchester, UK, August 2002.

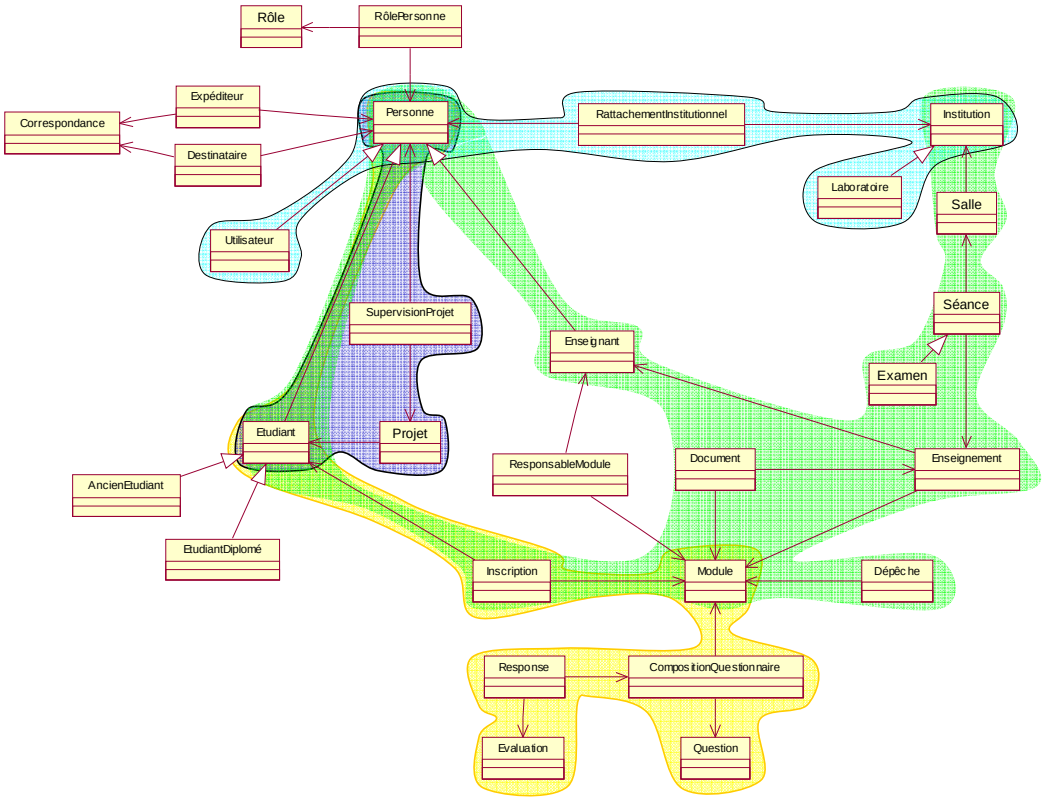

Figure 115 : Espace informationnels de l'étudiant dans *M@TIS*

En plus de l'hyperclasse *hcl_acteur*, l'espace informationnel d'un étudiant est modélisé par plusieurs hyperclasses (Figure 115) :

4. *hcl_projet_étudiant* : hyperclasse construite sur les classes *Étudiant*, *Personne*, *Projet* et *SupervisionProjet* ; elle est dédiée à la gestion du projet de recherche de l'étudiant et sa supervision.

3. *hcl_modules_étudiant* : construite sur les classes *Étudiant*, *Personne*, *Inscription*, *Module*, *Dépêche*, *Document*, *Enseignant*, *Enseignement*, *Séance*, *Examen*, *Salle* et *Institution*, cette hyperclasse permet à l'étudiant d'accèder aux informations relatives aux modules auxquels il est inscrit : planning des cours, documents mis en ligne, enseignants, dépêche de dernière minute, etc.

4. *hcl_évaluation_module* : cette hyperclasse est construite sur les classes *Étudiant*, *Personne*, *Inscription*, *Module*, *CompositionQuestionnaire*, *Question*, *Réponse* et *Évaluation* ; elle est utile pour l'évaluation des modules par les étudiants en répondant à un questionnaire.

Le *technicien* est responsable de l'entretien du système M@TIS. Bien qu'il dispose des pleins pouvoirs d'un point de vue technique, les prérogatives informationnelles de celui-ci se concentrent sur la gestion des comptes des utilisateurs : il est le responsable de la création de ces comptes, de l'attribution des rôles et de leur archivage.

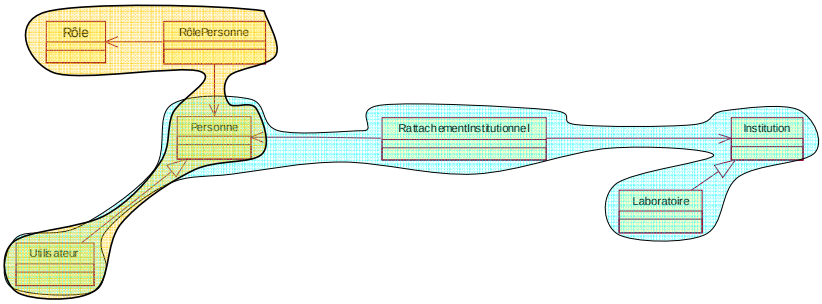

Figure 116 : Espace informationnel du *technicien* dans *M@TIS*

Une partie de l'espace informationnel du technicien est représenté par les hypeclasses *hcl_acteur* et *hcl_technicien* (Figure 116). *hcl_technicien* est construite sur classes *Personne*, *Utilisateur*, *Rôle* et *RôlePersonne* et permet au technicien de gérer les comptes des utilisateurs.

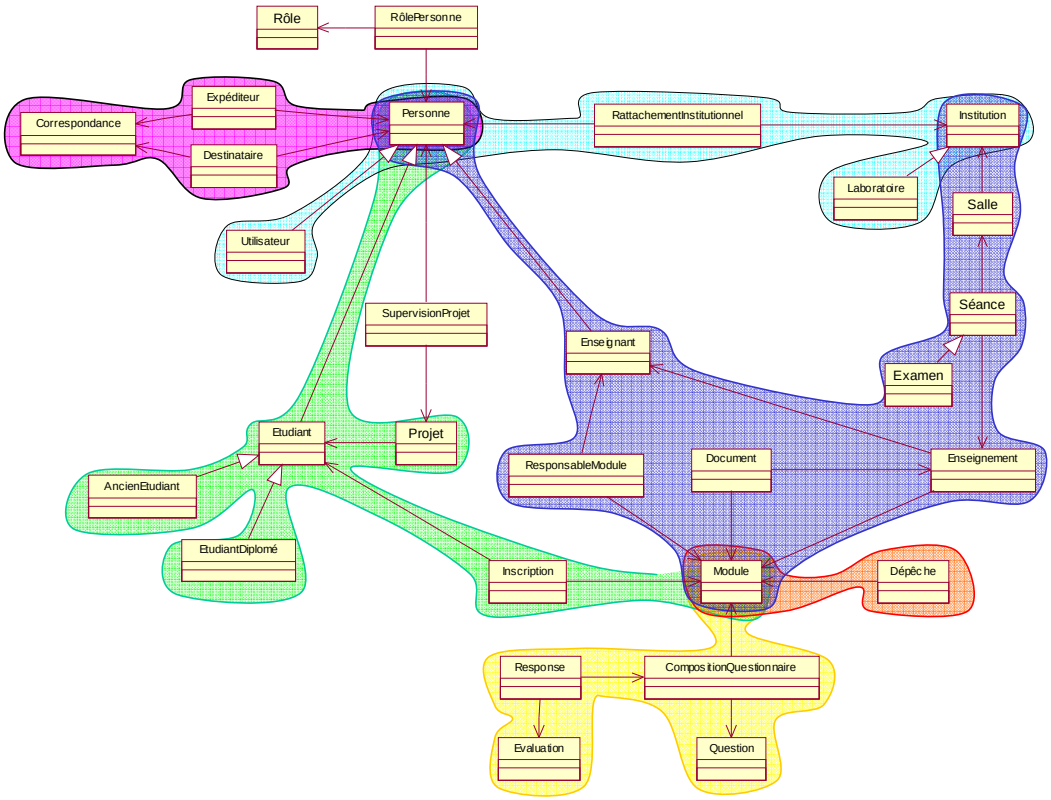

Figure 117 : Espace informationnel du coordinateur dans *M@TIS*

De la même manière, l'espace informationnel du coordinateur est représenté par plusieurs hyperclasses dont voici quelques exemples (Figure 117) :

1. *hcl_résultats_étudiant* : hyperclasse dédiée à la gestion des résultats et à la préparation des relevés de notes des étudiants de la formation. Elle est construite sur les classes *Étudiant*, *Personne*, *Inscription*, *Module*, *AncienÉtudiant* et *ÉtudiantDiplômé*.

2. *hcl_courrier* : construite sur les classes *Personne*, *Correspondance*, *Expéditeur* et *Destinataire*, cette hyperclasse permet au coordinateur de gérer la correspondance officielle avec les différents acteurs de la formation.

3. *hcl_dépêche* : construite sur les classes *Module* et *Dépêche*, cette hyperclasse permet au coordinateur de publier des dépêches de dernière minute pour informer les étudiants des changements dans le planning des cours.

4. *hcl_planning* : cette hyperclasse est construite sur les classes *Module*, *Enseignement*, *Séance*, *Examen*, *Salle*, *Institution*, *Enseignant*, *ResponsableModule*, *Document* et *Personne* ; elle permet au coordinateur de gérer les modules et de planifier les séances.

5. *hcl_évaluation* : cette hyperclasse est construite sur les classes *Module*, *CompositionQuestionnaire*, *Question*, *Réponse* et *Évaluation* ; elle permet au coordinateur de lancer des questionnaires et de gérer les évaluations des modules par les étudiants.

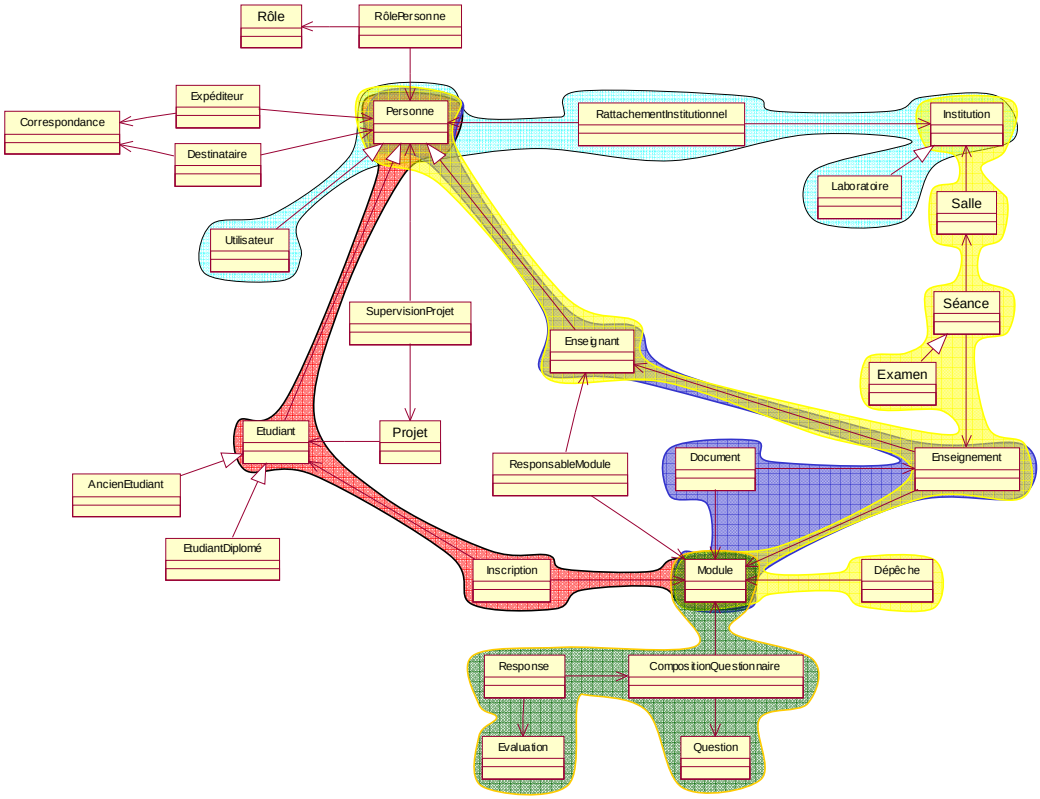

Figure 118 : Partie de l'espace informationnel d'un enseignant dans *M@TIS*

La Figure 118 montre une partie de l'espace informationnel d'un enseignant dans *M@TIS* représenté à travers plusieurs hyperclasses :

1. *hcl_étudiants_module* : cette hyperclasse est construite sur les classes *Personne*, *Étudiant*, *Inscription* et *Module* ; elle fournit à l'enseignant la liste des étudiants inscrit au module dans lequel il intervient.

2. *hcl_document_enseignement* : cette hyperclasse est construite sur les classes *Personne*, *Enseignant*, *Enseignement*, *Document* et *Module* ; elle permet à l'enseignant de gérer les documents qu'il met à disposition des étudiants dans le cadre d'un module.

3. *hcl_planning_enseignant* : cette hyperclasse est construite sur les classes *Personne*, *Enseignant*, *Enseignement, Module*, *Dépêche*, *Séance*, *Examen*, *Salle* et *Institution* ; elle fournit à l'enseignant le planning de sa participation à un module.

4. *hcl_évaluation_module* : cette hyperclasse est construite sur les classes *Module*, *CompositionQuestionnaire*, *Question*, *Réponse* et *Évaluation* ; elle permet à un enseignant d'accéder aux résultats des évaluations des modules par les étudiants.

Le responsable de module dispose d'un espace informationnel semblable à celui de l'enseignant (Figure 119). La différence réside dans les prérogatives élargies du responsable de module sur cet espace. Le responsable de module dispose aussi d'une hyperclasse *hcl_responsable_module*, construite sur les classes *Personne*, *Enseignant*, *ResponsableModule* et *Module*, qui regroupe les modules dont une personne est responsable.

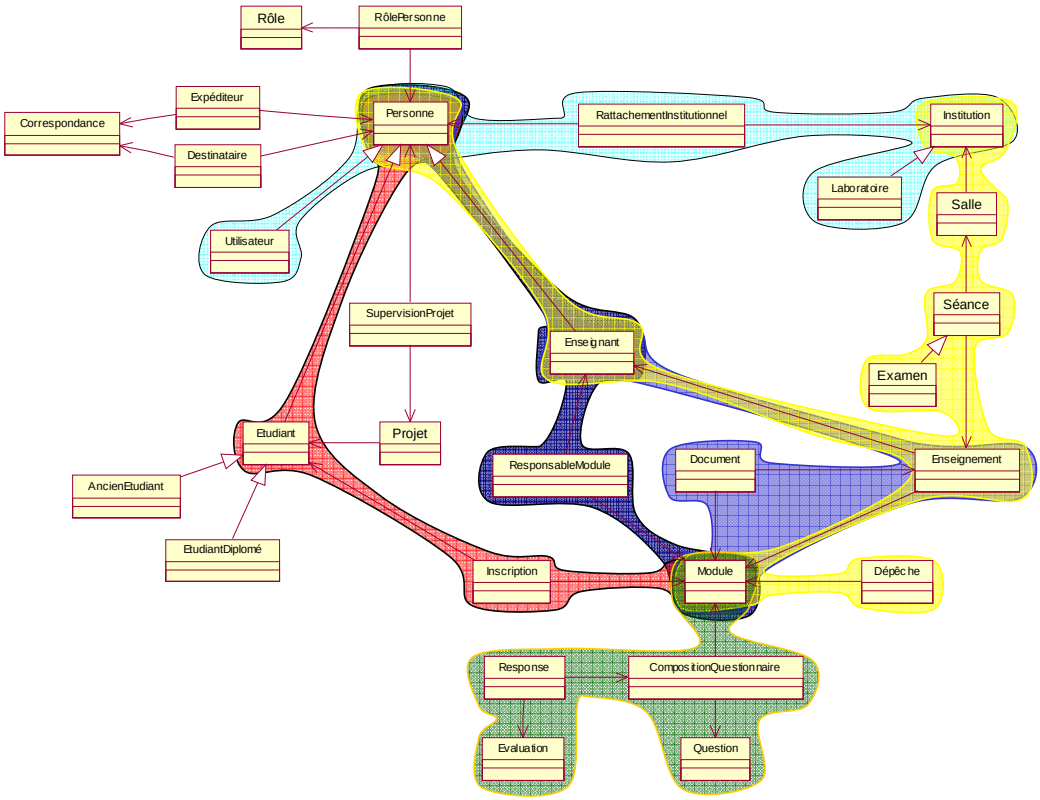

Figure 119 : Espace informationnel d'un responsable de module dans *M@TIS*

ANNEXE G

VUES STATIQUES DANS *UML*

1. Diagrammes de classes d'*UML*

« Un diagramme de classes *UML* est une collection d'éléments de modélisation statiques (classes, paquetages...), qui montre la structure d'un modèle. Il représente l'architecture conceptuelle du système : les classes que le système utilise, ainsi que leurs liens, que ceux-ci représentent un emboîtage conceptuel (héritage…) ou une relation organique (agrégation …) » (Volle, 2002)[158].

1.1. Classe

« Une classe n'est pas une fonction, une classe est une description abstraite - condensée – d'un ensemble d'objets du domaine de l'application… Une classe contient au moins son nom. Le rectangle qui symbolise la classe peut également contenir un stéréotype et des propriétés. *UML* définit les stéréotypes de classes suivants :

- `<< signal >>`, une occurrence remarquable qui déclenche une transaction dans un automate,
- `<< interface >>`, une description des opérations visibles,
- `<< métaclasse >>`, la classe d'une classe comme en Smalltalk,
- `<< utilitaire >>`, une classe réduite au concept de module et qui ne peut être instanciée.

Les propriétés désignent toutes les valeurs attachées à un élément de modélisation, comme les attributs, les associations et les étiquettes. Une étiquette est une paire (attribut, valeur) définie par l'utilisateur ; elle permet de préciser par exemple des informations de génération de code, d'identification ou de références croisées » (Muller, 1998)

1.1.1. Les classes paramétrables

« Les classes paramétrables sont des modèles de classes. Elles correspondent aux classes génériques d'Eiffel et aux templates de C++. Une classe paramétrable ne peut être utilisée telle quelle. Il convient d'abord de l'instancier, afin d'obtenir une classe réelle qui pourra à son tour être instanciée afin de donner des objets. Lors de l'instanciation, les paramètres effectifs personnalisent la classe réelle obtenue à partir de la classe paramétrable. Les classes paramétrables permettent de construire des collections universelles, typées par des paramètres effectifs.

---

[158] (Volle, 2002) Volle M., Langage de modélisation UML. Décembre 2002. sur *http://www.volle.com*

Ce type de classes n'apparaît généralement pas en analyse, sauf dans le cas particulier de la modélisation d'un environnement de développement. Les classes paramétrables sont surtout utilisées en conception détaillée, pour incorporer par exemple des composants réutilisables » (Muller, 1998).

1.2. Les attributs et les opérations

« Un attribut est une propriété nommée d'une classe qui décrit un domaine de valeurs possibles partagé par tous les objets de la classe. À tout instant, chaque objet d'une classe porte une valeur spécifique pour chaque attribut de sa classe. » (Roques & Vallée, 2002)[159].

« La syntaxe retenue pour la description des attributs est la suivante :

```
Nom_Attribut : Type_Attribut = Valeur_Initiale
```

Cette description peut être complétée progressivement… les attributs dérivés offrent une solution pour allouer des propriétés à des classes, tout en indiquant clairement que ces propriétés sont dérivées d'autres propriétés déjà allouées.

La syntaxe retenue pour la description des opérations est la suivante :

```
Nom_Opération
(Nom_Argument : Type_ Argument = Valeur_Par_Défaut, … )
: Type_Retourné
```

» (Muller, 1998).

1.2.1. Visibilité des attributs et des opérations

« *UML* définit trois niveaux de visibilité pour les attributs et les opérations :

- Public, qui rend l'élément visible à tous les clients de la classe,
- Protégé, qui rend l'élément visible aux sous-classes de la classe,
- Privé, qui rend l'élément visible à la classe seule. » (Muller, 1998).

1.2.2. Les interfaces

« Une interface utilise un type pour décrire le comportement visible d'une classe, d'un composant ou d'un paquetage. Une interface est un stéréotype d'un type. *UML* représente les interfaces au moyen de petits cercles reliés par un trait à l'élément qui fournit les services décrits par l'interface. Une interface fournit une vue totale ou partielle d'un ensemble de services offerts par un ou plusieurs éléments. Les dépendants d'une interface utilisent tout ou partie des services décrits dans l'interface. » (Muller, 1998).

---

[159] (Roques & Vallée, 2002) Roques P., Vallée F., UML en action ; De l'analyse des besoins à la conception en Java. Eyrolles, Avril 2002. ISBN : 2-212-09127-3.

1.3. Associations entre classes

Une association exprime une connexion sémantique bidirectionnelle entre deux classes. L'association est instanciable dans un diagramme d'objets ou de collaboration, sous forme de liens entre objets issus de classes associées.

« Les associations représentent des relations conceptuelles entre les classes… elles impliquent des responsabilités en termes de navigation. La navigation dans le modèle statique représente la capacité à obtenir des informations en parcourant les associations entre classes ». (Roques & Vallée, 2002).

« Les associations représentent des relations structurelles entre classes d'objets… La plupart des associations sont binaires, c'est-à-dire qu'elles connectent deux classes… Des arités supérieures peuvent cependant exister… Les associations n-aires peuvent généralement se représenter en promouvant l'association au rang de classe et en ajoutant une contrainte qui exprime que les multiples branches de l'association s'instancient toutes simultanément, en un même lien… Les associations peuvent être nommées.

L'extrémité d'une association est appelée rôle chaque association binaire possède deux rôles, un de chaque extrémité. Le rôle décrit comment une classe voit une autre classe au travers d'une association » (Muller, 1998).

1.3.1. Multiplicité des associations

Chaque rôle d'une association porte une indication de multiplicité qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe. La multiplicité est une information portée par le rôle, sous la forme d'une expression entière bornée.

1	Un et un seul
0..1	Zéro ou un
M..N	De M à N (entiers naturels)
*	De zéro à plusieurs
*0..**	De zéro à plusieurs
*1..**	D'un à plusieurs

Table 4 : valeurs des multiplicités dans UML

Les valeurs de multiplicité sont souvent employées pour décrire de manière générique les associations. Les formes les plus courantes sont les associations *1 vers 1*, *1 vers N* et *N vers N*.

1.3.2. Contraintes sur les associations

Les contraintes sont des expressions qui précisent le rôle ou la portée d'un élément de modélisation (elles permettent d'étendre ou préciser sa sémantique). Sur une association, elles

peuvent par exemple restreindre le nombre d'instances visées (ce sont alors des *expressions de navigation*).

« Toutes sortes de contraintes peuvent être définies sur une relation ou sur un groupe de relations. La multiplicité est une contrainte sur le nombre de liens qui peuvent exister entre deux objets.

La contrainte `{ordonnée}` peut être placée sur le rôle pour spécifier qu'une relation d'ordre décrit les objets placés dans la collection ; dans ce cas, le modèle ne spécifie pas comment les éléments sont ordonnés, mais seulement que l'ordre doit être maintenu durant l'ajout ou la suppression des objets par exemples.La contrainte `{sous-ensemble}` indique qu'une collection est incluse dans une autre collection. La contrainte `{ou-exclusif}` précise que, pour un objet donné, une seule association parmi un groupe d'associations est valide.

Les associations peuvent également relier une classe à elle-même… Ce type d'association est appelé association réflexive » (Muller, 1998).

Les contraintes peuvent s'exprimer en langage naturel. Graphiquement, il s'agit d'un texte encadré d'accolades.

1.3.3. Les classes-associations

« Une association peut être représentée par une classe pour ajouter, par exemple, des attributs et des opérations dans l'association. Une classe de ce type, appelée parfois classe associative ou classe-association, est une classe comme les autres et peut à ce titre participer à d'autres relations dans le modèle » (Muller, 1998).

1.3.4. Restriction des associations

« La restriction (UML parle de qualification) d'une association consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association » (Muller, 1998). La restriction de l'association est définie par une clé, qui permet de sélectionner les objets ciblés.

1.3.5. Les agrégations

« Une agrégation représente une association non symétrique dans laquelle une des extrémités joue un rôle prédominant par rapport à l'autre extrémité. Quelle que soit l'arité, l'agrégation ne concerne qu'un seul rôle d'une association » (Muller, 1998).

L'agrégation exprime un couplage fort et une relation de subordination. Elle représente une relation de type « *ensemble / élément* ». *UML* ne définit pas ce qu'est une relation de type « ensemble / élément », mais permet cependant d'exprimer cette vue subjective de manière explicite.

Une agrégation peut notamment (mais pas nécessairement) exprimer : (i) qu'une classe (un « élément ») fait partie d'une autre (« l'agrégat »), (ii) qu'un changement d'état d'une classe,

entraîne un changement d'état d'une autre, (iii) qu'une action sur une classe, entraîne une action sur une autre.

A un même moment, une instance d'élément agrégé peut être liée à plusieurs instances d'autres classes (l'élément agrégé peut être partagé). Une instance d'élément agrégé peut exister sans agrégat (et inversement) : les cycles de vies de l'agrégat et de ses éléments agrégés peuvent être indépendants.

1.3.6. Composition

La composition est une agrégation forte (agrégation par valeur). Les cycles de vies des éléments (les « composants ») et de l'agrégat sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi.

A un même moment, une instance de composant ne peut être liée qu'à un seul agrégat.

Les « objets composites » sont des instances de classes composées.

1.3.7. La navigation

« Les associations décrivent le réseau de relations structurelles qui existent entre les classes et qui donnent naissance aux liens entre les objets, instances de ces classes. Les liens peuvent être vus comme des canaux de navigation entre les objets.

Par défaut, les associations sont navigables dans les deux directions. Dans certains cas, seule une direction de navigation est utile. Une association navigable uniquement dans un sens peut être vue comme une demi-association » (Muller, 1998).

1.3.8. Association dérivée

Les associations dérivées sont des associations redondantes, qu'on peut déduire d'une autre association ou d'un ensemble d'autres associations. Elles permettent d'indiquer des chemins de navigation "calculés", sur un diagramme de classes. Elles servent beaucoup à la compréhension de la navigation (comment joindre telles instances d'une classe à partir d'une autre).

1.3.9. La généralisation

« *UML* emploie le terme généralisation pour désigner la relation de classification entre un élément plus général et un élément plus spécifique. Dans le cas des classes, la relation de généralisation exprime le fait que les éléments d'une classe sont aussi décrits par une autre classe. La relation de généralisation signifie *est un* ou *est une sorte de*… Les attributs, les opérations, les relations et les contraintes définies dans les super-classes sont héritées intégralement dans les sous-classes… Différentes contraintes peuvent être appliquées aux relations de généralisation, pour distinguer par exemple les formes de généralisation exclusives des formes inclusives. Les contraintes sur les relations de généralisation se représentent au moyen d'expressions entre parenthèses, directement attachées aux généralisations.

La contrainte `{Chevauchement}` ou `{Inclusif}` indique qu'une classe descendante d'une classe appartient au produit cartésien des sous-classes de la classe. Un objet concret est alors construit à partir d'une classe obtenue par mélange de plusieurs super-classes » (Muller, 1998).

Les hiérarchies de classes permettent de gérer la complexité, en ordonnant les objets au sein d'arborescences de classes, d'abstraction croissante.

- Spécialisation : Démarche descendante, qui consiste à capturer les particularités d'un ensemble d'objets, non discriminés par les classes déjà identifiées. Elle consiste à étendre les propriétés d'une classe, sous forme de sous-classes, plus spécifiques (permet l'extension du modèle par réutilisation).
- Généralisation : Démarche ascendante, qui consiste à capturer les particularités communes d'un ensemble d'objets, issus de classes différentes. Elle consiste à factoriser les propriétés d'un ensemble de classes, sous forme d'une super-classe, plus abstraite (permet de gagner en généricité).
- Classification : L'héritage (spécialisation et généralisation) permet la classification des objets.

1.4. Stéréotypes

Il s'agit d'un mécanisme d'extensibilité du méta-modèle d'*UML*. Les stéréotypes permettent d'étendre la sémantique des éléments de modélisation et de définir de nouvelles classes d'éléments de modélisation, en plus du noyau prédéfini par *UML*. *UML* propose de nombreux stéréotypes standards.

2. Diagramme d'objets

Les diagrammes d'objets, ou diagrammes d'instances, montrent des objets et des liens. Comme les diagrammes de classes, les diagrammes d'objets représentent la structure statique.

Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets. Ce type de diagramme s'utilise principalement pour montrer un contexte, par exemple avant ou après une interaction, mais également pour faciliter la compréhension des structures de données complexes, comme les structures récursies. Il montre des objets (instances de classes dans un état particulier) et des liens (relations sémantiques) entre ces objets.

3. Diagramme de composants

Dans *UML*, un composant représente une partie modulaire, déployable et remplaçable d'un système ; le composant encapsule l'implantation et expose un ensemble d'interfaces. » Les composants représentent des éléments physiques remplaçables ; Il peut s'agir par exemple de code source, d'un fichier binaire ou d'un exécutable, par exemple un navigateur ou un serveur

http, une base de données, une DLL ou une archive JAR (comme pour un EJB) » (Larman, 2003) [160].

« Les diagrammes de composants décrivent les éléments physiques et leurs relations dans l'environnement de réalisation (..). Ils montrent les choix de réalisation. Les modules représentent toutes les sortes d'éléments physiques qui entrent dans la fabrication des applications informatiques. Les modules peuvent être de simples fichiers, des paquetages Ada, ou encore des bibliothèques chargées dynamiquement.

Les relations de dépendance sont utilisées dans les diagrammes de composants pour indiquer qu'un composant fait référence aux services offerts par un autre composant. Ce type de dépendance est le reflet des choix de réalisation. Elles représentent généralement les dépendances de compilation » (Muller, 1998).

> « Un composant au sens *UML* représente un élément physique à partir duquel on construit le logiciel. Il peut s'agir d'un fichier de code tout comme d'un exécutable ou même d'un fichier de documentation… ce concept est trop vaste pour être exploitable » (Roques & Vallée, 2002)[161]

4. Diagramme de déploiement

« Un diagramme de déploiement représente la façon dont les instances de composants et les processus sont configurés pour s'exécuter sur des instances de nœuds de traitement. *UML* propose une notation à usage général appelée *package* ou *paquetage*, qui est une sorte d'icône de dossier à onglets. Les packages permettent de représenter des groupes logiques d'objets ; ils peuvent correspondre à des packages Java ou des espaces de nommage C++, ou encore à d'autres agrégats logiquement distincts et à des sous-systèmes » (Larman, 2003).

---

[160] (Larman, 2003) Larman C., UML et les Design Patterns. CampusPress, Collection Référence, juillet 2003. ISBN : 2744016233.

[161] (Roques & Vallée, 2002) Roques P., Vallée F., UML en action. 2ème édition, Eyrolles, 2002.

ANNEXE H

VUES DYNAMIQUES DANS *UML*

Le langage *UML* propose quatre types de diagrammes pour spécifier les vues dynamiques d'un système : (i) les *diagrammes de collaboration*, (ii) les *diagrammes de séquence*, (iii) les *diagrammes d'activités* et (iv) les *diagrammes d'états-transitions*.

1. Diagramme de collaboration

Les diagrammes de collaboration montrent des interactions entre objets (instances de classes et acteurs) et permettent de représenter le contexte d'une interaction, car on peut y préciser les états des objets qui interagissent.

2. Diagramme de séquence

Les diagrammes de séquences permettent de représenter des collaborations entre différentes entités et/ou acteurs selon un point de vue temporel, on y met l'accent sur la chronologie des envois de messages : par exemple des objets dans un modèle d'un logiciel, des sous-systèmes dans un modèle d'un système complet.

Contrairement au diagramme de collaboration, on n'y décrit pas le contexte ou l'état des objets, la représentation se concentre sur l'expression des interactions.

3. Diagramme d'activités

Les diagrammes d'activités sont une variante des diagrammes d'états-transitions. Ils permettent de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation.

Une activité représente une exécution d'un mécanisme, un déroulement d'étapes séquentielles. Le passage d'une activité vers une autre est matérialisé par une transition. Les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre.

4. Diagramme d'états-transitions

« L'état d'un objet est la valeur d'un ou plusieurs de ses attributs à un instant donné. Les états potentiels d'un objet peuvent, lorsqu'ils sont énumérables, être organisés en un diagramme dont les nœuds représentent les états et dont les arcs représentent les transitions chronologiques

d'un état à un autre. Le diagramme d'états d'un objet peut donc avoir un état initial et un ou plusieurs états finals » (Bouzeghoub & al. 1997)[162]. Les diagrammes d'états-transitions décrivent les changements d'états d'un objet ou d'un composant *UML*, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs.

Ils décrivent les différents états possibles des objets d'une classe ainsi que les changements d'états, en réponse aux interactions avec d'autres objets (comportement d'un objet ordonnancé par les événements) et les transitions possibles entre ces états.

Un diagramme d'états-transitions est un automate d'états fini, où les états sont reliés par des arcs orientés qui décrivent les transitions. Les attributs du diagramme d'états-transitions sont (i) les *événements* (stimuli externes ou internes), (ii) les *états* (valeurs des objets), (iii) les *changements d'états* (transitions) et (iv) les *opérations* (actions ou activités).

Un *état* se caractérise par sa durée et sa stabilité, il représente une conjonction instantanée des valeurs des attributs d'un objet. Un *état initial* est le point de commencement du cycle de vie d'objet. Un *état terminal* correspond à la fin de vie d'un objet.

Une *transition* représente le passage instantané d'un état vers un autre. La transition de l'*état source* à l'*état de destination*, est déclenchée par l'arrivée de l'*événement* qui conditionne la transition. Une transition peut être automatique lorsque l'événement qui la déclenche n'est pas spécifié. En plus des événements précis, il est aussi possible dans un diagramme d'états-transitions de conditionner une transition, à l'aide de "gardes" : expressions booléennes, exprimées en langage naturel).

Une *action* correspond à une opération disponible dans l'objet dont on représente les états. On peut aussi associer une action à l'événement qui déclenche une transition.

Dans un état donné, il est possible d'associer une action à l'événement qui déclenche une transition avec la syntaxe « *événement / action* » qui exprime que la transition (déclenchée par l'événement cité) entraîne l'exécution de l'action spécifiée sur l'objet, à l'entrée du nouvel état :

- *entry / action* : action exécutée à l'entrée de l'état ;
- *exit / action* : action exécutée à la sortie de l'état ;
- *on event / action* : action exécutée à chaque fois que l'événement cité survient ;
- *do / action* : action récurrente ou significative, exécutée dans l'état ;

Il est aussi possible de représenter l'échange de messages entre automates dans un diagramme d'états-transitions.

[162] (Bouzeghoub & al. 1997) Bouzeghoub M., Gardarin G., Valduriez P., Les objets. Paris : Eyrolles, 1997. ISBN 2212089570.

ANNEXE I

RÉSEAUX DE PETRI

Les réseaux de Petri permettent, comme les diagrammes d'états-transitions, de représenter des événements, des états et des transitions.

Les réseaux de Petri sont un formalisme graphique très répandu dans la modélisation du comportement des systèmes. Il est fondé sur la représentation de relation d'accessibilité entre états comme les graphes d'états.

Un réseau de Petri est un triplet $N=<P, T, F>$ où :

1. $P = \{p_i\} ; i \in [1, n]$, est l'ensemble fini et non vide des *places* ;
2. $T = \{t_j\} ; j \in [1, h]$, est l'ensemble fini et non vide des *transitions* ;
3. $F = \{a_{ij}\}$, est l'ensemble des *relations* (arcs orientés) entre les ensembles P et T. *F* est la relation d'écoulement du réseau.

Les places p_i et les transitions t_i d'un réseau de Petri, en nombre fini et non nul, sont reliées par des arcs orientés. Un réseau de Petri est dit *graphe biparti alterné*, c'est à dire qu'il y a alternance des types de nœuds : tout arc, qui doit obligatoirement avoir un nœud à chacune de ses extrémités, relie soit une place à une transition soit une transition à une place.

Un réseau de Petri $N=<P, T, F>$ vérifie donc les conditions suivantes :

- $P \cup T \neq \emptyset$
- $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$

Les *places* sont des endroits qui contiennent des jetons. A chaque place p_i correspondent ses transitions en entrée $^\circ t_i$ et ses transitions en sortie t_i°. Elles mémorisent le nombre de jetons.

Les *transitions* représentent les endroits à travers lesquels le système change. Une transition t_j a des places en entrées $^\circ p_i$ et des places en sorties p_i°, représentées par des arcs respectivement venant d'une ou plusieurs places et allant vers une ou plusieurs places.

Les *arcs* sont des flèches orientées représentant les fonctions d'écoulement entre les places et les transitions utilisées et vice versa. Un arc relie toujours une transition à une place ou une place à une transition.

Marquage du réseau de Petri

Pour définir l'état d'un système modélisé par un réseau de Petri, il est nécessaire de compléter le réseau de Petri par un marquage. Ce marquage consiste à disposer un nombre entier (positif ou nul) de *marques* ou *jetons* dans chaque place du réseau de Petri.

Le marquage est une fonction de P vers les entiers positifs, appelée *fonction de marquage* et donnée par un vecteur $M=(m_i)$, $i\in[1, p]$, tel que m_i représente le nombre de marqueurs (jetons) associés à la place p_i.

Les *jetons* sont placés sur les places et les jetons ne se déplacent pas : ils apparaissent ou disparaissent seulement sur les places.

Dans un réseau de Petri, on dispose d'un marquage initial M_0.

Évolution temporelle d'un réseau de Petri

L'évolution l'état du réseau de Petri correspond à une évolution du marquage. Les jetons, qui matérialisent l'état du réseau à un instant donné, peuvent passer d'une place à l'autre par *franchissement* ou *tir* d'une transition.

Le franchissement d'une transition consiste à retirer un jeton dans chacune des places en amont de la transition et à ajouter un jeton dans chacune des places en aval de celle-ci.

Les règles générales d'évolution des réseaux de Petri marqué simple sont les suivantes :

- une transition est *franchissable* ou *sensibilisée* ou *tirable* ou encore *validée* lorsque chacune des places en amont possède au moins un jeton,
- le réseau ne peut évoluer que par franchissement d'une seule transition à la fois, transition choisie parmi toutes celles qui sont validées à cet instant,
- le franchissement d'une transition est indivisible et de durée nulle.

Une transition est dite *tirée* si elle est validée et activée.

« Dans la première interprétation, le réseau représente une chaîne d'événements s'impliquant les uns les autres (chaîne causale). Dans la seconde interprétation, le réseau peut être assimilé à un diagramme d'états avec ses transitions. Mais la particularité d'un réseau de Petri est de représenter dans le même modèle les états successifs de plusieurs objets différents. Un des principaux intérêts des réseaux de Petri est leur capacité à décrire à la fois l'aspect dynamique et l'aspect fonctionnel. Le même modèle spécifie le « comment faire » et le « quand le faire ». Les résultats théoriques acquis sont très utilisables pour valider les différents modèles du système d'information. On peut par exemple s'assurer de l'atteignabilité de certains états, de la concentration de certains événements, du bon parallélisme de l'algorithme en général, etc. » (Bouzeghoub & al. 1997)[163].

[163] (Bouzeghoub & al. 1997) Bouzeghoub M., Gardarin G., Valduriez P., Les objets. Paris : Eyrolles, 1997. ISBN 2212089570.

ANNEXE J

ISIS : INFORMATION SYSTEM UPON INFORMATION SYSTEM

ISIS est l'acronyme de « *Information System upon Information System* » ou *SI sur SIs*. C'est un cadre de référence (Figure 120), proposé par (Le Dinh, 2005)[164], pour la gestion et la coordination des connaissances, sous forme de modèles, utilisées dans les développements de SIs. L'objectif d'*ISIS* est de gérer et de coordonner modèles informationnels utilisés dans ces développement, en les considérant comme les principales ressources d'information.

Un SI sur SIs est typiquement un SI qui supporte d'autres SIs « prêts à la coordination »[165] en développement.

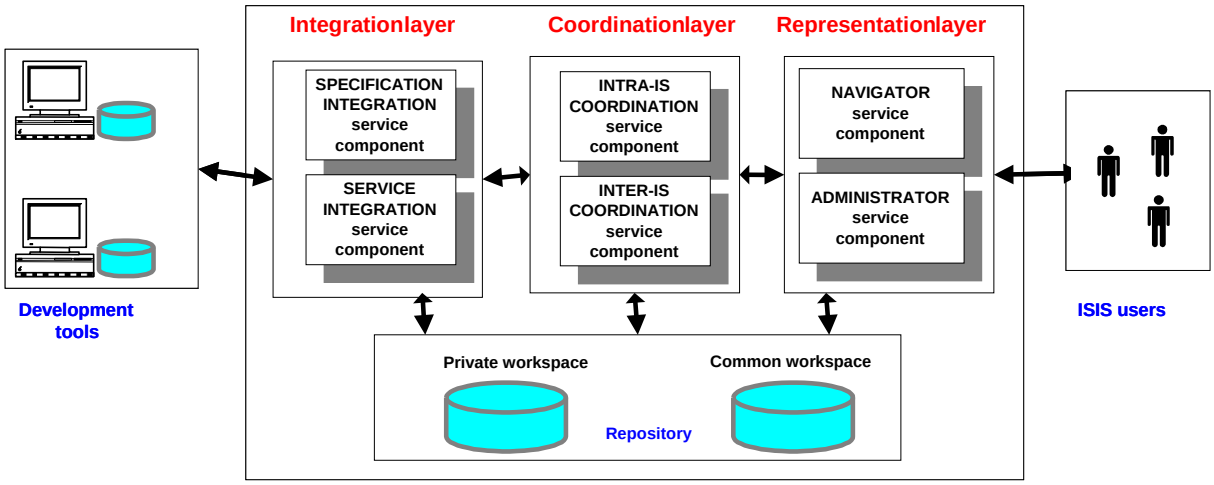

Figure 120 : Architecture globale d'*ISIS*

Dans *ISIS*, chaque SI en développement est représenté par un composant de SI. Ce choix garantit que chaque composant représentant un SI est autonome : il contient tous les éléments qui participent à sa définition ; et rend possible la prise en charge des situations de recouvrement informationnel entre plusieurs SIs et la mise en place des protocoles de coordination adéquats. *ISIS* se transforme en un outil de pilotage de l'interopérabilité des SIs en développement.

ISIS comprend trois niveaux ((i) *Intégration*, (ii) *Coordination* et (iii) *Représentation*) pour représenter, valider, gérer, intégrer et coordonner les modèles informationnels. Le niveau *Intégration* fournit les facilités nécessaires à l'intégration des modèles sauvegardés dans les

---

[164] (Le Dinh, 2004) Le Dinh Th., Information System Upon Information Systems: A Conceptual Framework. Thèse de doctorat. Université de Genève, Novembre 2004.

[165] *Coordination-ready information systems.*

outils de développement et les modèles stockés dans *ISIS*. Le niveau *Représentation* offre les moyens nécessaires pour qu'utilisateurs d'*ISIS,* développeurs et administrateurs puissent utiliser les modèles stockés. Enfin, le niveau *Coordination* permet la coordination et de modèles entre / au sein de plusieurs projets de développement.

ISIS identifie six rôles génériques ou zones de responsabilité qui interviennent dans l'ingénierie des SIs :

1. Méthodologue : responsable de la conception des ressources d'information (composant de SI) et des activités de développement de SI ;

2. Modélisateur de méta-modèle : conçoit et construit le méta-modèle, qui va cadrer les modèles informationnels et techniques ;

3. Intégrateur de modèles : responsable de l'extraction des modèles et des composant des SI à partir de leurs sources, de leur intégration, et de leur stockage dans *ISIS* ou vice-versa ;

4. Intégrateur de services : responsable de l'intégration des services des différentes plates-formes ;

5. Analyste de modèles : responsable de la spécification de la structure et du comportement, et de la cohérence des modèles ;

6. Coordinateur de modèles : responsable de la coordination entre les utilisateurs d'*ISIS* pour la prise en charge des situations de recouvrement ou d'interdépendance de composants de SI et la mise en place des protocoles de recouvrement.

ANNEXE K

EXEMPLES DE COMPOSANTS DE SI POUR LE E-GOUVERNEMENT

Exemple # 1 : Demande et réexamen de demandes RMCAS

Le composant de SI de la Figure 121 a été construit à partir des articles 10 et 12 de la loi genevoise sur prestations cantonales accordées aux chômeurs en fin de droit (J 2 25) (*Revenu Minimum de Réinsertion*). Cette loi permet à ces derniers de recevoir une aide financière non remboursable en échange d'une contre-prestation.

Art. 10 Demande[166]

[1] Les prestations d'aide sociale sont accordées sur demande écrite de l'intéressé ou de son représentant légal.

[2] Cette demande doit être remise à l'Hospice général.

[3] Toutes pièces utiles concernant l'état civil, le domicile, la résidence, les enfants à charge, les ressources et la fortune de l'intéressé doivent être fournies.

[4] L'intéressé doit s'engager par écrit à :

a) autoriser le propriétaire ou son représentant à communiquer à l'Hospice général toute notification de hausse de loyer;
b) donner mandat à l'Hospice général, en cas d'octroi de prestations, de le représenter en cas de procédure. L'Hospice général se réserve le droit d'engager la procédure.

Art. 12 Réexamen périodique

[1] Les prestations d'aide sociale sont accordées pour une période de 12 mois au maximum. Au-delà de cette période, une nouvelle demande doit être déposée.

[2] Pendant cette période, le bénéficiaire des prestations d'aide sociale doit poursuivre activement ses démarches afin de retrouver un emploi.

166

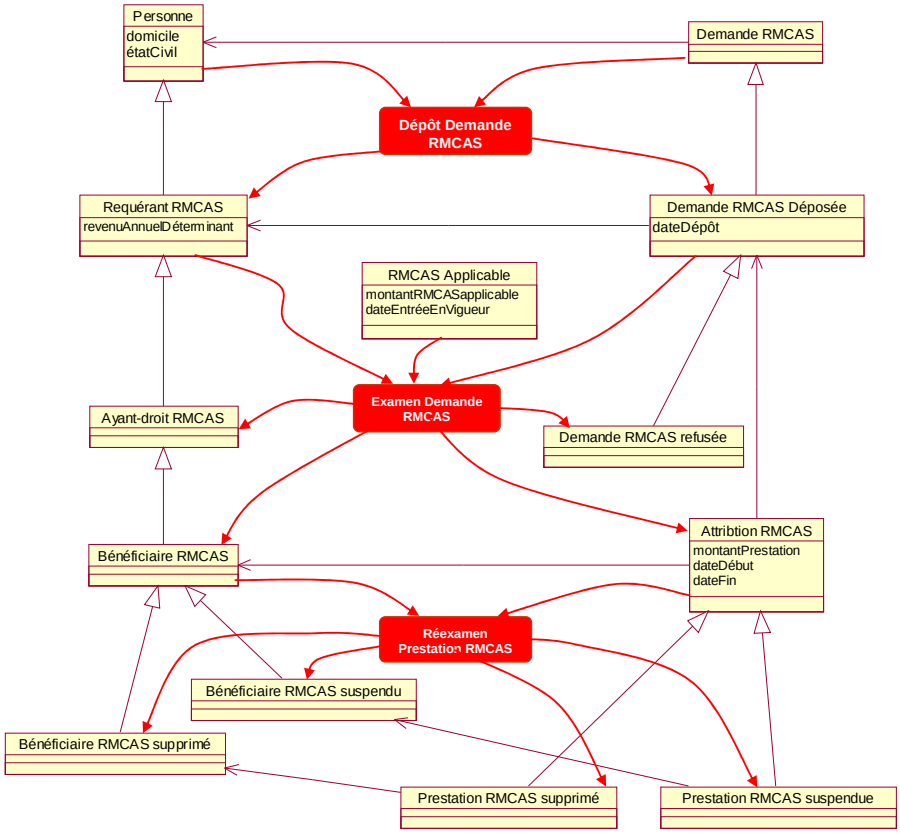

Figure 121 : Demande et réexamen de demandes RMCAS

Exemple # 2 : Allocation d'insertion

Cet exemple concerne l'attribution d'allocation d'insertion (AI). Le composant de la Figure 122 a été construit à partir des articles 28, 29, 30 et 31 de la loi (J 2 25).

Art. 28 Allocation d'insertion

Les personnes qui ont droit au revenu minimum cantonal d'aide sociale versé par l'Hospice général peuvent également recevoir une allocation d'insertion, unique, d'un montant variable, de 1 000 F au minimum et de 10 000 F au maximum.

Art. 29 Destination de l'allocation

L'allocation d'insertion est destinée à financer, totalement ou partiellement, des projets, réalistes et réalisables, inscrits dans la durée et concernant l'un des domaines suivants :

a) formation et recyclage professionnel;
b) création d'une activité lucrative;
c) réinsertion professionnelle et sociale.

Art. 30 Demande

[1] Le requérant présente par écrit une demande d'allocation d'insertion à l'Hospice général, accompagnée d'un descriptif et budget détaillés du projet envisagé.

[2] Les services sociaux compétents ou d'autres organismes peuvent prêter leur concours à l'élaboration du projet.

Art. 31 Commission d'attribution

[1] Les demandes d'allocation d'insertion sont examinées par une commission, nommée par le Conseil d'Etat, qui se compose :

a) du directeur général de l'Hospice général, qui la préside;(1)
b) d'un représentant de l'office de l'emploi;
c) d'un représentant de l'office d'orientation et de formation professionnelle;
d) de deux représentants des services sociaux privés;
e) de deux représentants des employeurs désignés par l'Union des associations patronales genevoises et de deux représentants des travailleurs désignés par la Communauté genevoise d'action syndicale.

[2] Les décisions de la commission sont notifiées par l'Hospice général, qui est lié par l'avis et les montants déterminés par celle-ci.

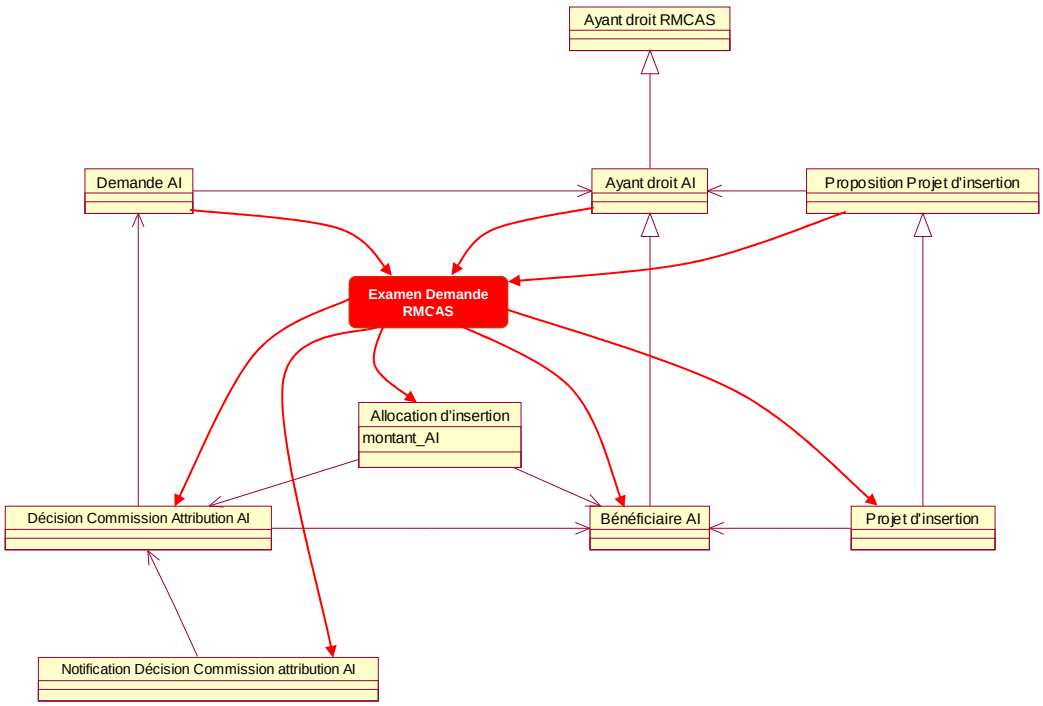

Figure 122 : Attribution de l'allocation d'insertion

BIBLIOGRAPHIE

(Al-Jadir & al., 1995) Al-Jadir L., Estier T., Falquet G., Léonard M., Evolution Features of the F2 OODBMS, Proc. Int. Conf. on Database Systems for Advanced Applications, DASFAA, Singapore 1995.

(Al-Jadir & Léonard, 1999) Al-Jadir L., Léonard M., If we refuse the Inheritance…, Proc. Int. Conf. on DEXA, Firenze, 1999.

(Al-Jadir, 1997) Al-Jadir L., Evolution-oriented database systems, PhD thesis, Uni. de Genève, 1997.

(Alter, 1992) Alter S., Information Systems. Prentice Hall, September 2001, ISBN: 0-13-043242-3

(Bachman & Daya, 1977) Bachman Ch. W., Daya M., The role concept in data models, VLDB 3rd, Tokyo, 1977.

(Bagui & Earp, 2003) Bagui S., Earp R., Database Design Using Entity-Relationship Diagrams. Auerbach Pub, June 2003. ISBN: 0849315484.

(Banerjee & al., 1987) Banerjee J. Kim W., Kim H-J., Korth H.F., Semantics and Implementation of Schema Evolution in object-oriented databases. Proc. Int. Conf. On Management of Data, ACM SIGMOD, San Francisco 1987.

(Batini, 1986) Batini C., Lenrerini M., Navathe S.B., A comparative analysis of methodologies for database schema integration. ACM Computing Surveys, Vol. 18, No. 4, December 1986.

(Berge, 1970) Berge C., Graphes et hypergraphes, ISBN 2-04-016906-7, Dunod informatique, 1970.

(Bodart & Pigneur, 1989) Bodart F., Pigneur Y., Conception assistée des systèmes d'information. 2ème édition, Masson, Paris, 1989.

(Booch & al., 1999) Booch, G., Rumbaugh, J., Jacobson, I., The unified modelling language user guide. Addison-Wesley, 1999.

(Booch & al., 2000) Booch, G., Rumbaugh, J., Jacobson, I., Le guide de l'utilisateur UML. Eyrolles, Paris, février 2000.

(Bouchy, 1994) Bouchy S., L'ingénierie des systèmes d'information évolutifs. Eyrolles. 1994. ISBN : 2-212-08790-X.

(Bouzeghoub & al. 1997) Bouzeghoub M., Gardarin G., Valduriez P., Les objets. Paris : Eyrolles, 1997. ISBN 2212089570.

(Bui, 2002) Bui Minh Tu D., Élaboration et Intégration des Spécifications de Workflow Coopératif dans la Conception des Systèmes d'Information, Mémoire de DEA en Systèmes d'information, Université de Genève, 2002.

(Buloz, 2000) Buloz d., Première mise en place des hyperclasses dans un SGBD commercialisé. Mémoire de Diplôme en Informatique. Université de Genève, Juillet 2000.

(Cauvet & Rosenthal-Sabroux, 2001) Cauvet C., Rosenthal-Sabroux C., Ingénierie des systèmes d'information. Hermes Sciences Publications, 2001. ISBN : 2746202190.

(Charbou-Pecou & al., 2002) Charbou-Pecou, M., Freire J., J. C., Front, A., Giraudin, J-P., Guetari, R., Ingénierie Objet. Chapitre 1 : Objets, langages et méthodes. Editions Dunod, 2002. ISBN : 2729606424.

(Chen, 1976) Chen. P., The entity-relationship model: Toward a unified view of data. ACM Transactions on Database Systems, 1(1):9-36, 1976.

(Dahchour & al., 2002) Dahchour M., Pirotte A., Zimanyi E., A generic role model for dynamic object, Proc. Int. Conf. CAISE 2002, Toronto 2002.

(Fournier, 1999) Fournier N., Élaboration et Implantation des Hyperclasses dans M7. Mémoire de Licence en Systèmes d'Information et de Communication. Septembre 1999, Université de Genève.

(Galliers, 1994) Galliers R.D., Relevance and Rigor in Information Systems Research: Community, in Business Process Reengineering - Information Systems Opportunities and Challenges. Proc. of the IFIP TC8 Open Conference on BPR, 1994.Elsevier, Holland.

(Giraudin & al., 2002) Giraudin J. P., Freireire J.-J.-C., Turki S., Assessment of a difficult voyage in Pentagonie - Evaluation of a pedagocical intranet project. Proceedings of ICEE 2002, Manchester, UK, August 2002.

(Giraudin & Freireire, 2001) Giraudin J. P., Freireire J.-J.-C., An educational intranet for a formation in management and technology of information systems. Proceedings of ICEE 2001, Oslo, Norway, August 2001.

(Giraudin, 2001) Giraudin, J-P., Chabre-Peccoud, M., Rieu, D., Saint-Marcel, Ch., Ingénierie des systèmes d'information, Modèles de spécification pour l'ingénierie des systèmes d'information. P 115-148, ISBN: 2-7462-0219-0, Hermes Sc. Europe Ltd, 2001.

(Gottlob & al., 1996) Gottlob G., Schrefl M., Röck B., Extending object-oriented systems with roles, ACM Transactions on Information Systems, Vol.14, No.3, July 1996.

(IEEE, 1990) IEEE Standard Computer Dictionary.

(Junet & al., 1986) Junet M., Falquet G., Léonard M., ECRINS/86: An Extended Entity-Relationship Data Base Management System and its Semantic Query Language, Proc. Int. Conf. on Very Large Data Bases, VLDB, Kyoto 1986.

(Khadraoui & al., 2004) Khadraoui A., Turki S., Aïdonidis C., Léonard M., Institutional Information Systems Alignment on Laws. AISTA 2004, IEEE, November 15-18, 2004.

(Lai, 2000) Lai M., UML La notation unifiée de modélisation objet. DUNOD, 2000.

(Laudon & Laudon, 2001) Laudon K. C., Laudon J. P., Management Information Systems. 7th Ed. Prentice Hall, 2001.

(Le Dinh, 2004) Le Dinh Th., Information System Upon Information Systems: A Conceptual Framework. Thèse de doctorat. Université de Genève, Novembre 2004.

(Le Dinh, 2005) Le Dinh Th., Towards a new infrastructure supporting interoperability of information systems in development: the Information System upon Information Systems. Proc. Interop-ESA Conf. Genève. Février 2005.

(Léonard & al., 1998) Léonard M., Estier Th., Parchet O., Ramos F., Rimbert R., Schnegg P-A., Somers J., Noyau d'un Systèmes d'Information Générique. Rapport scientifique et technique final. Projet CTI nº 3326.1, Décembre 1998.

(Léonard & Parchet, 1998) Léonard M., Parchet O., Étude des situations de recouvrements de l'information pour la conception des SI. Actes d'INFORSID'98, Mai 1998, Montpellier, France.

(Léonard, 1988) Léonard M., Structures des bases de données, ISBN 2-04-016407-3, Dunod informatique, 1988.

(Léonard, 1999a) Léonard M., Introduction aux systèmes d'information, notes de cours, Université de Genève, 1999.

(Léonard, 1999b) Léonard M., M7 : approche évolutive des systèmes d'information. Proc. Inforsid'99. La Garde, France, mai 1999.

(Léonard, 2002) adapté à partir de Léonard M., Composants de SI : Définition et Propriétés. Notes de cours, Université de Genève, 2002.

(Léonard, 2003a) Léonard M., IS engineering getting out of classical system engineering. Proc. ICEIS'2003. April 2003, Angers, France.

(Léonard, 2003b) Léonard M., Modélisation des Systèmes d'Information. Interactive-Matis. Genève, Juin 2003.

(Lesca & Lesca, 1995) Lesca H., Lesca E., Gestion de l'Information : qualité de l'information et performances de l'entreprise. Litec, 1995,

(Leung & Olivari, 2002) Leung K-F, Olivari A., Conception et développement d'une solution web pour la formation MATIS. Novembre 2002.

(Lopez & Mustabasic, 2004) Lopez L. R., Mustabasic R., Navigateur de Bases de Données STELLA. Mémoire de Licence en Systèmes d'Information et de Communication. Université de Genève, Septembre 2004.

(Luu, 2003) Luu H-T-H., Une approche pour la gestion et l'évolution des règles d'intégrité d'un système d'information. Mémoire préliminaire, Université de Genève, Décembre 2003.

(Maier & al., 1984) Maier D., Ullman J.D., Vardi, M. Y., On the foundations of the Universal Relation Model. ACM Transactions on Database Systems, Vol. 9, No. 2, June 1984, Pages 283-308.

(Moix, 2002) Moix E.,Conception et mise en place d'un système d'information pour la formation MATIS. Mémoire de licence en systèmes d'information à l'Université de Genève. Avril 2002.

(Morejon, 1994) Morejon J., Merise, vers une modélisation orientée objet. Les Editions d'Organisation 1994.

(Morley & al., 2000) Morley C., Hugues J., Leblanc B., UML pour l'analyse d'un système d'information. Dunod Informatiques, 2000.

(Muller, 1998) Muller P.-A., Modélisation objet avec UML. Eyrolles, Paris, Août 1998. ISBN : 2-212-08966-X.

(Mumford & al., 1984) Mumford E., Hirschheim R., Fitzgerald G., Wood-Harper T., Research Methods in Information Systems. Proc. of the IFIP WG 8.2 Colloquim, Manchester Business School, September 1984.

(Nguyen, 1997) Nguyen G. T., Objets et évolution, Chapitre 6, Ingénierie Objet : Objets, techniques, concepts (Sous la direction de Oussalah C.), P205-232. InterEditions, Collection Informatiques, 1997. ISBN : 2729606424.

(Nurcan, 1998) Nurcan, S., Main Concepts for Cooperative Work Place Analysis, Proceedings of Telecooperation Conference of the 15th, IFIP World Computer Congress 1998.

(Odberg, 1995) Odberg E., MultiPerspectives: Object Evolution and Schema Modification Management for Object-Oriented Databases, PhD Thesis, Norwegian Institute of Technology, 1995.

(Ott & Natansky, 1997) Ott M., Nastansky L., Groupware Based Organization – Design for dynamic Workflow Management and Office Systems. Workgroup Computing Competence Center Paderborn, Universität-GH Paderborn. 1997.

(Oussalah, 1999) Oussalah Ch., Génie Objet : analyse et conception de l'évolution. HERMES Sciences Publications, Paris 1999. ISBN 2-7462-0029-5.

(Ozkan, 2004) Ozkan S., Information Systems Quality versus Software Quality: A Process Based Assessment Model within the context of Organisation. Proc. of AIM'2004, May 2004, INT, Evry, Paris, France.

(Papazoglou & Krämer, 1997) Papazoglou M.P., Krämer B.J., A database model for object dynamics, VLDB Journal volume 6, 1997.

(Pham, 2003) Pham T., Intégration des aspects statiques et dynamiques des Systèmes d'Information, Mémoire préliminaire, Université de Genève, 2003.

(Probst & al., 1997) Probst G., Mercier J-Y., Bruggimann O., Rakotobarison A., Organisation et Management. Éditions d'Organisation, Paris, 1997.

(Roques & Vallée, 2002) Roques P., Vallée F., UML en action ; De l'analyse des besoins à la conception en Java. Eyrolles, Avril 2002. ISBN : 2-212-09127-3.

(Schmidt & al., 2001) Schmidt Ch., Nastasi S., Iseli B., Mise en place d'un système d'information: l'intranet M@TIS. Mémoire de licence en systèmes d'information à l'Université de Genève. Octobre 2001.

(Scholz-Reiter & Stickel, 1996) Scholz-Reiter B., Stickel E., Business Process Modelling. Springer Verlag Berlin, Heidelberg, 1996.

(Smith & Smith, 1977) Smith J.M., Smith D.C.P., Database Abstractions: Aggregation and Generalization. ACM TODS, Vol. 2, No. 2, 1977.

(Snene & Léonard, 2003) Snene M., Léonard M., Distributed framework for real time web based collaboration: M7Tool case. Proc. ACS/IEEE Conf. On Computer Systems and Applications. Tunis, Tunisia, July 2003.

(Soutou, 2002) Soutou C., De UML à SQL, conception de bases de données. Eyrolles, Paris, juin 2002.

(Turki & al., 2003) Turki S., Léonard M., Arni-Bloch N., From Hyperclasses to IS Components. Proc. of the 10th International Conference on Concurrent Engineering (CE'2003), July 2003, Madeira, Portugal, P. 235-242, R. Jardim-Goncalves, H. Cha, A. Steiger-Garcao (eds.), Balkema Publishers.

(Turki & al., 2004) Turki S., Aïdonidis C., Khadraoui A., Léonard M., Towards Ontology-Driven Institutional IS Engineering. EMOI-INTEROP 2004, CaiSE'04 Conf., Riga (Latvia), June 2004.

(Turki & Léonard, 2002a) Turki S., Léonard M., Hyperclasses: towards a new kind of independence of the methods from the schema. Proc. of the 4th International Conference on Enterprise Information Systems, ICEIS'2002, Vol.2, P. 788-794, ISBN: 972-98050-6-7. Ciudad Real, Spain, April 3-6, 2002.

(Turki & Léonard, 2002b) Turki, S., Léonard, M., IS Components with Hyperclasses. In the proceedings of the OOIS 2002 Workshops, Advances in Object-Oriented Information Systems, Montpellier, France, September 2, 2002, LNCS 2426, P. 132-141, ISBN: 3-540-44088-7. Springer, 2002.

(Turki, 2001) Turki, S, Introduction aux hyperclasses. Proc. Inforsid'2001, P. 281-299, ISBN: 2-906855-17-0. Martigny, Suisse, Mai 2001.

(Turrini & Gonçalves, 2002) Turrini G., Gonçalves R., Création d'un serveur de Composant de SI. Mémoire de Licence en Systèmes d'Information et de Communication. Université de Genève, Novembre 2002.

(Volle, 2001) Volle M., Économie des nouvelles technologies, Chapitre 11, Quelques indications sur les systèmes d'information. Economica, 2001.

(Volle, 2002) Volle M., Langage de modélisation UML. Décembre 2002. sur http://www.volle.com

(Von Hellens, 1997) Von Hellens L.A., Information systems quality versus software quality: a discussion from a managerial, an organisational and an engineering viewpoint. Information and Software Technology. 39, P. 801-808. 1997.

(Warboys & al., 1999) Warboys B., Kawalek P., Robertson I., Greenwood R., Business Information Systems: a Process Approach. McGraw-Hill. 1999.

(WordNet, 2.0) WordNet 2.0 by Princeton University.

(Zheng, 2003) Zheng Y., TAÏCHI, vers un SGBD intégrants les aspects statiques et dynamiques. Mémoire de Diplôme d'Études Avancées en Systèmes d'Information. Université de Genève, Septembre 2003.