UNIVERSITE JEAN MOULIN LYON 3

Thèse

Pour l'obtention du titre de

Docteur en Informatique

Présentée et soutenue publiquement le lundi 20 décembre 2004 par

Vincent COUTURIER

L'ingénierie des systèmes d'information coopératifs par réutilisation : une approche à base de patterns

devant le Jury composé de

Directeur de thèse : **Madame Danielle BOULANGER**
Professeur à l'Université Lyon 3

Rapporteurs : **Madame Thérèse LIBOUREL**
Maître de Conférences H.D.R. à l'Université Montpellier 2
Monsieur Jean-Pierre GIRAUDIN
Professeur à l'Université Grenoble 2

Suffragants : **Monsieur Robert LAURINI**
Professeur à l'INSA Lyon
Monsieur Gilles DUBOIS
Maître de Conférences à l'Université Lyon 3

L'université n'entend donner ni approbation, ni improbation aux opinions émises dans les thèses. Ces opinions doivent être considérées comme propres à leurs auteurs

REMERCIEMENTS

Nous tenons à remercier vivement notre Directeur de thèse, Madame Danielle Boulanger, pour la confiance qu'elle nous a témoignée en permanence, pour son encadrement efficace et son indéfectible soutien. Elle a su nous faire partager, dans son équipe de recherche MODEME, son enthousiasme pour la recherche en systèmes d'information.

Notre reconnaissance va également aux rapporteurs, Madame Thérèse Libourel, Maître de conférences H.D.R. à l'Université Montpellier 2 et Monsieur Jean-Pierre Giraudin, Professeur à l'Université Grenoble 2, dont les évaluations ont contribué à l'amélioration du document final et à l'enrichissement de nos futurs travaux de recherche.

Notre sincère reconnaissance s'adresse à Monsieur Robert Laurini, Professeur à l'INSA de Lyon, dont la présence dans ce jury est pour nous un grand honneur.

Nous remercions également tout particulièrement, Monsieur Gilles Dubois, pour son implication dans les propositions que nous avons effectuées et la rédaction de cette thèse. Ses remarques et conseils ont toujours été éclairés et judicieux et ont grandement facilité l'avancement de nos travaux.

Nous exprimons, en outre, notre gratitude à Monsieur Pascal Colin pour sa disponibilité et la vérification soigneuse de certaines parties du présent document.

Nous souhaitons, enfin, remercier très chaleureusement et amicalement diverses personnes.

Nous avons tout d'abord une pensée amicale pour Christophe Lafaye, avec qui nous avons partagé les mêmes bureaux de travail – et les périodes de doutes et d'incertitudes inhérentes – au cours des six derniers mois de rédaction de nos travaux respectifs.

Nous tenons à remercier à nouveau Gilles Dubois, ainsi qu'Eric Disson, pour leur amitié indéfectible qui a grandement facilité notre intégration dans l'équipe MODEME et exprimons notre gratitude aux membres actuels ou « anciens » de cette équipe – Guilaine Talens, Caroline Wintergerst, Muriel Perez, Magali Séguran et Nicolas Faure – ainsi qu'aux membres du Centre de Recherche de l'IAE que nous avons eu le plaisir et l'honneur de côtoyer : Martine Chorein, Geneviève Tabaret, François-Xavier de Vaujany, Olivier Brunel, Olivier Mérignac, François Bonicalzi, Magdalena Godek, Isabelle Gallois, Wafi Chtourou, Séverine Le Loarne, Séverine Ventolini,…

Enfin, nous terminerons ces remerciements par une pensée pour nos parents pour leurs encouragements et leur disponibilité sans faille tout au long de ce processus de thèse.

« Soit A un succès dans la vie. Alors A = x + y + z,
où x = travailler, y = s'amuser, z = se taire. »

Albert Einstein[1]

[1] Extrait du magazine The Observer, 15 janvier 1950.

Sommaire

SOMMAIRE **9**

INTRODUCTION GENERALE **11**

CHAPITRE 1 : L'INGENIERIE DES SYSTEMES D'INFORMATION COOPERATIFS **15**

1. INTRODUCTION 15
2. DE L'INGENIERIE DES SYSTEMES D'INFORMATION… 15
3. … A L'INGENIERIE DES SYSTEMES D'INFORMATION COOPERATIFS 30
4. VERS LE CHOIX D'UN TYPE D'AVOIR LOGICIEL ADAPTE A L'INGENIERIE DES SYSTEMES D'INFORMATION COOPERATIFS : LES PATTERNS 60
5. CONCLUSION 76

CHAPITRE 2 : PRESENTATION DES PATTERNS DE COOPERATION **77**

1. INTRODUCTION 77
2. TYPOLOGIE DES PATTERNS 77
3. DEMARCHE D'INGENIERIE DES PATTERNS DE COOPERATION 87
4. FORMALISME DES PATTERNS DE COOPERATION ET SUPPORT DE COOPERATION 96
5. VALIDATION ET REUTILISATION DES PATTERNS DE COOPERATION ET DE SUPPORT DE COOPERATION 112
6. TRAVAUX DU DOMAINE 116
7. CONCLUSION 134

CHAPITRE 3 : PATTERNS DEDIES A L'ANALYSE DES SYSTEMES D'INFORMATION COOPERATIFS **137**

1. INTRODUCTION 137
2. REFERENTIEL DU DOMAINE DE LA COOPERATION 137
3. PATTERNS D'ANALYSE DE COOPERATION 161
4. TRAVAUX FUTURS 193
5. CONCLUSION 194

CHAPITRE 4 : PATTERNS DEDIES A LA CONCEPTION ET A L'IMPLANTATION DES SYSTEMES D'INFORMATION COOPERATIFS **195**

1. INTRODUCTION 195
2. PATTERNS D'ARCHITECTURE DE COOPERATION 195
3. PATTERNS DE CONCEPTION DE COOPERATION ET PATTERNS DE SUPPORT TECHNIQUE 210
4. CONCLUSION 235

BILAN & PERSPECTIVES 237

1. BILAN 237
2. PERSPECTIVES 240

BIBLIOGRAPHIE 243

ANNEXE A : ARRP - ATELIER DE REPRESENTATION ET DE REUTILISATION DES PATTERNS DE COOPERATION 265

1. ETAT DE L'ART SUR LES OUTILS DE MISE EN ŒUVRE DE PATTERNS 265
2. PRESENTATION D'ARRP 267
3. CONCLUSION 278

INDEX DES FIGURES 279

INDEX DES TABLEAUX 281

TABLE DES MATIÈRES 283

Introduction générale

A l'heure où l'information n'est plus considérée comme une ressource opérationnelle mais aussi comme une ressource stratégique pour l'organisation, son système d'information devient un facteur de différenciation par rapport à ses concurrents. Un système d'information informatisé (SII) véhicule l'information à travers le temps, l'espace et les acteurs de l'organisation. Ce système, pour être conçu et maintenu, requiert l'application d'un processus d'Ingénierie des Systèmes d'Information (ISI). L'ISI est une approche interdisciplinaire permettant la construction de systèmes d'information opérationnels et porteurs d'une certaine valeur ajoutée pour l'organisation. Elle demande une expertise technique, i.e. des connaissances portant sur le développement logiciel mais aussi des connaissances métier relatives au domaine d'application. Le processus d'ingénierie des systèmes d'information se focalise sur la définition des besoins et des fonctionnalités requises dès le début du cycle de développement, sur la documentation de ces besoins et continuent avec la conception, l'implantation, la validation du système, la mise en production et la maintenance.

La complexification des problèmes économiques et les progrès réalisés dans les technologies de l'information ces dernières années sont à l'origine d'une importante évolution du concept de système d'information. Le développement des systèmes d'information est ainsi devenu une tâche de plus en plus complexe et il en ressort un besoin de plus en plus vital de codifier et gérer les connaissances modélisées dans les processus d'ingénierie des SI. Cette complexité croissante justifie l'adoption de modèles, de méthodes et d'outils qui seront utilisés pendant les phases d'analyse, de conception et d'implantation des SI. Traditionnellement, l'ingénierie des systèmes d'information s'organise par la mise en oeuvre, dans le cadre d'une méthode (MERISE, OMT[2], etc.), d'une succession de modèles (entité-association, orienté-objet, relationnel,..) afin de favoriser un continuum de la définition des besoins des clients jusqu'au système développé et exploité [RIEU, 1999]. Cependant, ces modèles et méthodes ne suffisent plus aujourd'hui pour prendre en compte la complexité des nouvelles applications. Depuis quelques années, une nouvelle approche basée sur la réutilisation de composants (ou avoirs logiciels) a émergé. Initialement cantonnée aux étapes d'implantation, la réutilisation s'impose de plus en plus dans les phases amont du développement des systèmes d'information (expression des besoins, analyse, conception). Des exemples de tels avoirs logiciels sont les bibliothèques de classes, les composants

[2] Object Modeling Technique

logiciels, les frameworks (cadriciels), les patterns logiciels (ou patrons) ou les objets métiers [RIEU, 1999] et ont pour objectif de faciliter le développement des systèmes d'information, mais aussi d'assurer leur qualité et leur pérennité. D'une manière générale, la réutilisation lors du développement des systèmes d'information vise trois objectifs : diminuer les coûts de développement et de maintenance, réduire les délais et améliorer la qualité du logiciel.

Cette nécessité de réutiliser est encore plus flagrante, dans le cadre de l'ingénierie de systèmes d'information coopératifs (SIC), systèmes complexes dont l'objectif est de faire coopérer des sources d'information et des applications pré-existantes, hétérogènes et conçues de manière autonome. De nombreux systèmes d'information coopératifs ont été proposés jusqu'alors - aussi bien dans le domaine de la recherche que dans la sphère des applications professionnelles - présentant leurs propres concepts permettant de construire des processus coopératifs entre applications et/ou sources d'information et de traiter les conflits inhérents. L'objectif de tels systèmes est d'obtenir une réelle interopérabilité entre les sources d'information assurant à la fois la coordination des échanges d'information et la prise en compte de la sémantique des données.

Nos travaux se situent dans le cadre de l'aide à l'ingénierie des systèmes d'information coopératifs et de l'amélioration de la réutilisabilité des SIC existants. Parmi les différentes techniques de réutilisation facilitant cette ingénierie, l'une des voies les plus pertinentes et prometteuses semble aujourd'hui celle des patterns. L'idée du concept de pattern consiste à capitaliser un problème récurrent d'un domaine et sa solution de manière à faciliter la réutilisation et l'adaptation de cette solution lors de l'apparition d'une nouvelle occurrence du problème.

Notre travail de recherche a plus précisément pour objectif de mettre en place un cadre méthodologique pour l'ingénierie des systèmes d'information coopératifs basés sur la réutilisation de patterns. Ce cadre méthodologique propose d'une part, la conception de patterns et, d'autre part, de réutiliser ceux-ci lors du développement de ces systèmes. Ainsi, deux activités du processus d'ingénierie de SIC peuvent être distinguées :

- l'activité d'*ingénierie des patterns* (développement d'un système réutilisable) consiste à identifier les connaissances réutilisables, à les spécifier et les organiser sous forme de patterns.
- l'activité d'*ingénierie des SIC* (développement d'un système par réutilisation), proprement dite, a pour objectif la recherche, la sélection, l'adaptation et l'intégration des patterns définis lors de l'étape précédente afin de spécifier et développer le SIC.

La première activité vise ainsi à définir des modèles de spécifications de différents niveaux d'abstraction qui couvrent la complexité des SIC et assurent un continuum de transformation dans les

spécifications. La deuxième consiste à mettre en place une démarche générale de développement des SIC, par réutilisation, couvrant l'ensemble des phases du processus d'ingénierie (analyse, conception, implantation) et permettant d'accélérer et de faciliter ce développement.

Le **premier chapitre** présente le cadre de notre travail et argumente le choix des patterns logiciels comme technique de réutilisation la plus adaptée pour accélérer et faciliter l'ingénierie des systèmes d'information coopératifs.

Le **deuxième chapitre** présente et positionne les types de patterns que nous proposons, leur démarche d'ingénierie et leur formalisme. Il permet, d'autre part, d'envisager la réutilisation des différents patterns afin de permettre l'ingénierie des systèmes d'information coopératifs par réutilisation. Enfin, il expose les travaux du domaine existants et permet de situer nos propositions par rapport à ces travaux.

Les patterns spécifiant un ensemble de fragments de modèles représentant les éléments du SIC à différents niveaux d'abstraction, les **troisième et quatrième chapitres** présentent, respectivement, les patterns dédiés à la phase d'analyse de l'ingénierie des SIC et les patterns dédiés aux phases de conception et d'implantation. Ces deux chapitres présentent ainsi le résultat de l'activité d'ingénierie des patterns.

Chapitre 1 :
L'ingénierie des systèmes d'information coopératifs

1. Introduction

Notre travail de recherche a pour cadre l'ingénierie des systèmes d'information coopératifs (SIC). Ce premier chapitre présente ce cadre et a pour objectif de sélectionner la ou les techniques de réutilisation permettant de faciliter cette ingénierie.

La première partie de ce chapitre (§ 2) présente la notion d'ingénierie des systèmes d'information, au sens large, et explicite notamment quelques techniques de réutilisation (ou avoirs logiciels) susceptibles de faciliter cette ingénierie. La deuxième partie (§ 3) expose le contexte de notre travail, i.e. les systèmes d'information coopératifs et l'ingénierie de ces systèmes. Enfin, la troisième et dernière partie (§ 4) développe notre choix d'une technique de réutilisation, parmi celles explicitées en section 2, compatible avec notre contexte, c'est-à-dire adaptée à l'ingénierie des systèmes d'information coopératifs : les patterns. La notion de pattern y est également détaillée afin de mieux cerner ce concept assez complexe que nous utilisons tout au long de cette thèse.

2. De l'ingénierie des systèmes d'information…

Cette section a pour objectif de présenter la notion d'ingénierie des systèmes d'information (§ 2.2), ainsi que quelques techniques de réutilisation favorisant cette ingénierie (§ 2.3). Nous définissons, cependant, préalablement le concept de « système d'information » (§ 2.1).

2.1. Les systèmes d'information

Les *systèmes d'information* (SI) sont aujourd'hui incontournables dans les organisations : ils leur donnent vie et permettent l'accomplissement coordonnée des activités. Au cours de ces dernières années, le système d'information est même devenu une ressource stratégique facteur de différenciation

pour l'organisation. La notion de système d'information a été très largement définie et commentée ; nous retiendrons la définition suivante mettant l'accent sur l'acception informatique du SI[3] [ROLLAND & AL., 1988] :

« Un système d'information est un artefact qui supporte un réseau de flux d'informations nécessaires pour organiser, mettre en oeuvre, gérer et maintenir les activités d'une organisation. C'est un instrument de communication qui sert aux échanges informationnels entre les partenaires de l'organisation et qui accroît leur efficacité. »

Un SI est utilisé pour collecter, stocker, manipuler et présenter l'information nécessaire à la coordination, au contrôle et à la prise de décision au sein d'une organisation [HICKS, 1993]. Les systèmes d'information aident les acteurs à analyser les problèmes, visualiser les sujets complexes et créer de nouveaux produits. Le rôle des systèmes d'information est devenu plus complexe et plus large au cours des décennies, allant du support d'une activité spécifique et d'un ensemble limité d'utilisateurs au support des processus métiers et des collaborations entre acteurs dans un environnement décentralisé et distribué [VERHAREN, 1997].

Les systèmes d'information devant être construits et maintenus, ce processus est appelé *ingénierie des systèmes d'information* (*Information Systems Engineering*). La maîtrise de cette ingénierie est devenue vitale avec l'accroissement de la complexité et de l'étendue des systèmes d'information. Nous précisons plus amplement cette notion d'ingénierie dans la section suivante.

2.2. L'ingénierie des systèmes d'information

Préambule : la notion de modèle

Ce préambule vise à préciser la notion de modèle, notion que nous employons tout au long de ce mémoire et dont la signification diffère en fonction de son contexte d'utilisation : génie logiciel ou ingénierie des systèmes d'information.

L'objectif principal du génie logiciel est de décrire à différents niveaux d'abstraction et de détails (analyse, conception, réalisation) et selon différents points de vue (statique, dynamique, fonctionnel) une application à réaliser [REVAULT, 1996]. Ces descriptions visent à représenter les données et traitements inhérents à cette application : elles constituent toutes des *modèles* de l'application. Les modèles, en génie logiciel, sont exprimés selon des formalismes ou langages de description plus ou moins graphiques, chacun étant adapté à un niveau d'abstraction et relatif à un point de vue spécifique.

[3] La vision informatique du SI est également qualifiée de Système d'Information Informatisé (SII).

L'ingénierie des systèmes d'information s'intéresse à l'automatisation de ces systèmes à travers principalement une vision orientée données. De nombreuses méthodes sont utilisées dans ce domaine et fournissent plusieurs contextes à l'élaboration de *modèles*. Les modèles les plus usités en ingénierie des SI sont ceux représentant les données. Ces modèles sont qualifiés de « statiques ». Au niveau le plus « bas » des représentations statiques d'information, i.e. au niveau de leur mise en oeuvre sous forme de bases de données, on peut noter l'utilisation fréquente du terme *schéma*, de préférence au terme *modèle*. La raison principale repose sur l'utilisation du terme *modèle* pour évoquer le langage et les structures informatiques dans lesquels les schémas sont exprimés (par exemple, le « modèle relationnel » ou le « modèle entité-association »). Ainsi, la notion de modèle évoque à la fois une expression dans un langage de modélisation et le langage lui-même. Cette polysémie n'est pas faite pour simplifier le dialogue ; elle a parfois cours en génie logiciel, mais c'est en ingénierie des SI qu'elle est sans doute la plus répandue.
Nous employons, dans le reste du document, le terme *modèle* dans son acception issue du génie logiciel. Nous utilisons également le terme *schéma*, dans le cas des bases de données, pour représenter leur structure. Le terme *schéma* constitue ainsi le synonyme de *modèle*.

* __ *

L'ingénierie des systèmes d'information a trait au système d'information et à l'ingénierie des systèmes. Pour Pressman [PRESSMAN, 1997], l'ingénierie des systèmes se focalise sur une variété d'éléments en analysant, concevant et organisant ces éléments en un système qui peut être un produit, un service, ou une technologie pour la transformation ou le contrôle de l'information.

Walsham [WALSHAM, 1993] présente une définition de l'ingénierie des SI (ISI) alliant ingénierie des systèmes et système d'information. L'ingénierie des systèmes d'information y est définie comme une approche interdisciplinaire permettant le développement de systèmes d'information opérationnels. L'ISI se focalise sur la définition des besoins des utilisateurs et des fonctionnalités requises dès le début du cycle de développement, i.e. dès l'analyse des besoins et se poursuit lors de la conception, du développement et de la validation du système. L'ISI comprend plusieurs domaines d'expertise qui doivent être intégrés et gérés afin de construire un système d'information et prend en compte à la fois les besoins organisationnels et les besoins techniques des utilisateurs avec l'objectif de fournir un produit de qualité qui soit en adéquation avec ces besoins.
Cette définition met principalement l'accent sur la vision orientée *produit* de l'ISI [PRESSMAN, 1997]. L'ISI possède cependant un aspect connexe, le *processus*. Le processus de construction des SI est appelé *processus d'ingénierie des systèmes d'information*. Ce processus comprend les différentes phases du cycle de vie d'un système d'information [AMBLER, 1998] [ALTER, 1999], à savoir, la phase

d'avant projet ou d'initialisation, la phase de développement[4] et la phase d'exploitation et de maintenance.

Selon, la vision française (par exemple, celle de [NANCI & AL., 2001]) ou américaine de l'ingénierie des systèmes d'information (par exemple, celle de [ZMUD & PRICE, 1999]), celle-ci couvre un champ disciplinaire plus ou moins vaste. Nous retiendrons la vision de Zmud & Price [ZMUD & PRICE, 1999] qui intègre sous l'appellation d'Ingénierie des Systèmes d'Information les champs disciplinaires suivants :

- La conception, l'implémentation et l'utilisation des technologies de l'information dans les organisations à des fins opérationnelles et stratégiques,
- Le management des technologies de l'information,
- Le génie logiciel[5],
- Les outils de l'utilisateur final,
- Les sciences du management et la recherche opérationnelle.

La vision de Zmud & Price intègre à la fois les acceptions organisationnelle et informatique des systèmes d'information, et considère que l'ingénierie des systèmes d'information fait appel à la discipline du génie logiciel. Remarquons, enfin, que de nombreux auteurs utilisent indifféremment les termes d'ingénierie des systèmes d'information et de génie logiciel.

Le développement des systèmes d'information devient une activité de plus en plus complexe et il est essentiel aujourd'hui de pouvoir codifier et gérer la connaissance dans et autour du processus d'ingénierie des SI.
Concernant l'aspect connaissance, les différents modèles créés dans un projet d'ingénierie des SI sont des exemples de connaissances codifiées sur ce que le système doit être. En résumé, un des principaux soucis dans l'ingénierie des SI est de gérer la quantité importante des connaissances associées au *processus* d'ingénierie, au *domaine* du projet en cours de développement et au *logiciel* développé lui-même. Trois domaines de connaissances dans l'ingénierie des SI peuvent ainsi être relevés [IIVARI, 2000] [BACKLUND, 2001] [BACKLUND, 2003] :

- La *connaissance du processus de développement*. Elle inclut la connaissance sur la mise en oeuvre d'un projet de développement, sur la façon d'appliquer des méthodes et des techniques

[4] La phase de développement du logiciel est composée de plusieurs phases, à savoir l'analyse, la conception, l'implantation et le test. Ces différentes phases constituent le *cycle de développement du logiciel*.

[5] Le génie logiciel regroupe l'ensemble des méthodes, techniques et outils de développement de logiciels.

pour le développement des SI, ainsi que les compétences relatives aux langages de développement.

- La *connaissance relative au domaine*. Elle inclut la connaissance sur les processus métiers qui devront être gérés par le système à développer (connaissances de domaine intrinsèques au système), ainsi que la connaissance sur la façon dont le nouveau système d'information affectera l'environnement. Ainsi, pour pouvoir construire un système, le développeur aura besoin de comprendre le domaine et l'environnement du système.
- La *connaissance relative à l'application*. Elle inclut les connaissances sur les différents aspects du produit logiciel, telles que les connaissances architecturales, de conception et les connaissances sur les ressources technologiques requises.

Selon Lyytinen & Robey [LYYTINEN & ROBEY, 1999], la complexité croissante des systèmes d'information et leurs incessantes évolutions rendent leur développement plus laborieux, plus coûteux et moins fiable et impliquent qu'un nombre toujours important de projets sont soumis à des échecs. Les raisons de ces échecs, qui ont amené à parler de *crise du logiciel* sont dues, d'après Lyytinen & Robey et Reel [REEL, 1999], au fait que les organisations n'ont pas suffisamment appris de leurs propres expériences. Un remède à ce problème consiste à encourager les développeurs de systèmes d'information à apprendre de leurs propres ou d'autres expériences et à utiliser cette connaissance, disponible dans un format explicite et codifié, dans leurs pratiques de développement. Wastell [WASTELL, 1999] suggère que l'ingénierie des SI soit vue comme un processus dans lequel sont engagés des spécialistes du domaine et des professionnels des systèmes d'information. En l'absence de l'une ou de ces deux compétences, il est indispensable que les acteurs impliqués dans le développement des systèmes d'information utilisent les connaissances du domaine et/ou des artefacts conceptuels et logiciels réutilisables afin de faciliter les développements. Dès lors, il émerge des organisations des besoins très forts en matière de *réutilisation*. En effet, la réutilisation s'inscrit dans l'ensemble des solutions en cours de recherche et de développement pour faire face à la crise du logiciel. Elle se définit comme une nouvelle approche d'ingénierie de systèmes selon laquelle il est possible de construire un système à partir d'éléments existants. La réutilisation notamment d'éléments de conception, ainsi que de code existant, est une technique efficace pour rendre les projets plus rentables et moins sujets à l'échec.

La problématique liée à la réutilisation est très vaste. Nous en explicitons les principes dans la section suivante.

2.3. La réutilisation, une aide à l'ingénierie des systèmes d'information

Depuis les années 1960, le génie logiciel essaie de résoudre les problèmes de la crise du logiciel, mentionnés précédemment.

Dans son article "*Mass-produced software components*" [MCILROY, 1976], présenté en 1968 à la conférence de Génie Logiciel de l'OTAN, McIlroy faisait naître un espoir. Il décrivait une bibliothèque de composants réutilisables dotés de fonctionnalités d'adaptation. L'aspect clé de cet article a été de permettre d'envisager le développement de logiciels comme une industrie capable de réaliser des produits à partir d'un ensemble d'avoirs logiciels[6] réutilisables. Dès lors, la réutilisation de logiciel a été considérée comme un des facteurs nécessaires pour passer de l'artisanat à l'ingénierie.

2.3.1. Définition de la réutilisation

Une des définitions majeures de la réutilisation a été donnée par Gamma & al. : « *capacité à construire systématiquement les nouveaux logiciels à partir de composants préexistants* » [GAMMA & AL., 1995]. Cette réutilisation lors de l'ingénierie des systèmes se manifeste sous deux formes : la réutilisation *opportuniste* (*opportunistic reuse*), qui survient accidentellement et la réutilisation *stratégique* (*strategic or systematic reuse*), qui est prévue dans le processus de développement du logiciel [TRACZ, 2002]. La plupart des organisations pratique la réutilisation opportuniste, réutilisation qui permet de gagner un peu de temps en modifiant les conceptions existantes afin de les adapter aux nouveaux besoins. Cependant, les nouvelles conceptions subissent le même processus coûteux en temps et en argent que les conceptions récupérées (test, déploiement, documentation, etc.). Cette réutilisation non structurée ni planifiée ne permet ainsi de réaliser que des gains marginaux. Les organisations pratiquant la réutilisation stratégique intègrent la réutilisation dans un processus bien défini. La réutilisation stratégique vise à améliorer la qualité de la conception logicielle et à réduire les délais et les coûts de développement.

Lorsqu'elle est centrée sur un domaine d'application donné, la réutilisation permet de mieux maîtriser la multiplication des connaissances et bénéficie ainsi d'un contexte plus favorable. Dans ce cadre, mettre en place techniquement la réutilisation concerne deux problématiques distinctes : le développement d'un système réutilisable (*design for reuse*) et le développement d'un système par réutilisation (*design by reuse*) [MILI & AL., 1995].

[6] D'après Rada & Moore [RADA & MOORE, 1997], un avoir logiciel (*asset*) est « n'importe quel produit du cycle de vie du logiciel », « l'utilisation d'un avoir logiciel dans plusieurs projets » étant qualifiée de réutilisation. Ainsi, la notion d'avoir logiciel est synonyme de la notion de composant ou de composant générique défini, par exemple, par [BARBIER & AL., 2002]. Ainsi, un composant ou avoir logiciel peut-être, par exemple, un pattern, un framework, un composant logiciel, un composant métier,… Afin d'éviter toute ambiguïté entre la notion de composant (générique) d'une part et, d'autre part, la notion de composant logiciel ou de composant métier, nous avons préféré utiliser le terme "avoir logiciel". Cependant, quand nous citerons ou nous référerons aux propos des auteurs, nous conserverons strictement le terme « Composant » s'il est usité.

2.3.2. Développement d'un système de réutilisation vs développement d'un système d'information par réutilisation

Le *développement d'un système de réutilisation* (*design for reuse*)[7] met en œuvre des tâches d'identification de ressources réutilisables, de spécification et d'organisation de composants, mais aussi d'implantation d'un ensemble de composants au sein d'un système de réutilisation [CAUVET & AL., 2001]. Les recherches concernent la définition de modèles et de langages pour la spécification d'avoirs logiciels réutilisables permettant la représentation de connaissances génériques. Des méthodes et des outils logiciels assistant l'identification de ressources réutilisables et la conception/réalisation de systèmes de réutilisation doivent également être développés. D'après Gross & al. [GROSS & AL., 1994], le développement d'un système de réutilisation correspond à l'ingénierie de domaine (*domain engineering*).

Le *développement d'un système d'information par réutilisation* (*design by reuse*)[8] nécessite une reformulation des processus usuels d'ingénierie des systèmes d'information, ainsi que des outils de développement intégrant de manière systématique la réutilisation d'avoirs logiciels dans le développement de systèmes d'information [CAUVET & AL., 2001].
D'après Gross & al. [GROSS & AL., 1994], le développement d'un système d'information par réutilisation correspond à l'ingénierie d'application (*application engineering*) qui utilise et personnalise les avoirs produits lors de l'ingénierie de domaine pour créer des produits logiciels spécifiques adaptés aux besoins des utilisateurs. Durant cette activité, les avoirs logiciels peuvent être adaptés (ou spécialisés) et combinés pour être intégrés dans un produit spécifique.

Ainsi, l'ingénierie d'application, également qualifiée d'*ingénierie logicielle conventionnelle* (*conventional software engineering*), se concentre sur la satisfaction des besoins pour un unique système, alors que l'ingénierie de domaine se concentre sur la fourniture de solutions réutilisables pour des familles de systèmes (Cf. Figure 1).

[7] Le "design for reuse" est parfois simplement qualifié de réutilisation en opposition à la réutilisabilité ou "design by reuse".

[8] Le "design by reuse" est parfois qualifié de "réutilisabilité". Un avoir logiciel est qualifié de réutilisable, par Calderia & Basili [CALDERIA & BASILI, 1991], si l'effort pour le réutiliser est infime par rapport à l'effort pour implanter un nouveau composant.

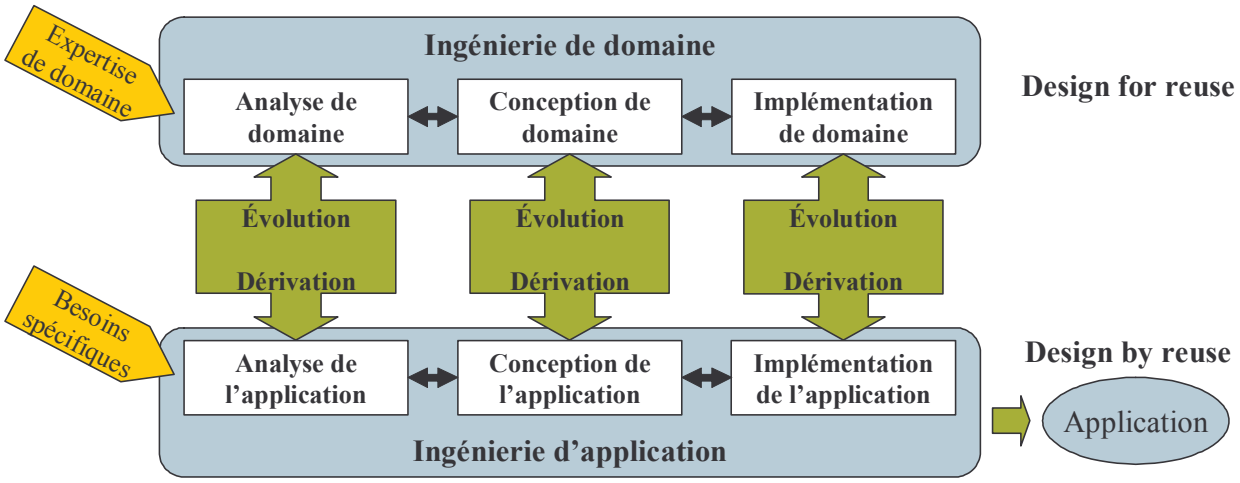


Figure 1 : Développement pour la réutilisation vs Développement par réutilisation

La réutilisation peut intervenir à différents niveaux au cours du cycle de vie du logiciel. Nous les détaillons dans la section suivante.

2.3.3. Les niveaux de réutilisation

La démarche de réutilisation peut être mise en oeuvre dans toutes les phases du cycle de développement d'un logiciel. Un premier niveau de réutilisation porte sur le processus de construction. De nombreux modèles ont été proposés : le modèle en cascade [ROYCE, 1970], le modèle en spirale [BOEHM, 1988], le modèle de la fontaine [HENDERSON-SELLERS & EDWARDS, 1990], le modèle orienté prototype [BISCHOFBERGER & POMBERGER, 1992], etc. La réutilisation consiste à reproduire ces processus de construction de logiciel, en identifiant les étapes et un ordre d'exécution.

Quel que soit le cycle de développement utilisé pour la construction d'un système, on y retrouve des activités communes : expression des besoins, analyse, conception et implémentation. Goldberg [GOLDBERG, 1990] considère que la réutilisation doit être mise en œuvre sur ces différentes activités et distingue la réutilisation au niveau du code source (implémentation) de la réutilisation au niveau des modèles.

La première possibilité consiste à structurer le code existant en modules, composants logiciels, classes ou fonctions et à réutiliser ces ensembles pour implémenter de nouvelles applications. Si les bibliothèques de composants logiciels ou de classes réutilisables constituent des techniques de réutilisation intéressantes pour la construction d'applications, elles n'apportent toutefois que des gains marginaux, le taux de réutilisation global restant en effet limité autour de 15 à 20% [FRONT, 1997].

La seconde possibilité, plus abstraite, consiste à utiliser des modèles réutilisables lors des phases d'analyse et de conception. L'objectif est de favoriser la réutilisation de connaissances acquises lors de

l'analyse ou de la conception logicielle et de permettre aux concepteurs de travailler à un niveau d'abstraction plus élevé que celui des classes ou composants logiciels. Cette seconde voie assure la mise en oeuvre d'une réutilisation plus systématique dans le processus de développement logiciel.

Un grand nombre d'avoirs logiciels, réutilisables lors des différentes phases du processus de développement du logiciel, ont été proposés dans le domaine du génie logiciel et des systèmes d'information. Tracz [TRACZ, 1994] énumère une vingtaine de technologies favorisant la réutilisation, essentiellement lors de la phase d'implantation du logiciel. Nous analysons, dans la section suivante, les types d'avoirs logiciels réutilisables qui nous semblent les plus intéressants pour faciliter l'ingénierie des systèmes d'information.

2.3.4. Les avoirs logiciels réutilisables

Nous présentons, dans les sous-sections suivantes, quatre types d'avoirs réutilisables - ou techniques d'ingénierie de SI par réutilisation - provenant du génie logiciel, à savoir les *modèles de domaine*, les *patterns*, les *composants logiciels* et les *frameworks*. Dans chaque cas, nous les définissons et précisons leur cadre d'applicabilité.

2.3.4.1. Les modèles de domaine

a) *Définition et principes*

Un *modèle de domaine* (*domain model*) exprime des connaissances de haut niveau réutilisables dans le développement de tous les systèmes d'un même champ d'application (appelé domaine) [SEMMAK, 1998]. Un modèle de domaine constitue un cadre de référence pour le développement de tous les systèmes appartenant à ce domaine. Différents types de modèles de domaine ont été proposés allant des modèles de domaine orientés objet, relevant du génie logiciel, aux abstractions de domaine issues du domaine de l'intelligence artificielle [NELLBORN, 1999].

Le principal objectif d'un modèle de domaine orienté-objet est d'être un véhicule pour la communication entre les praticiens d'un métier et les informaticiens, facilitant la perception mutuelle et la compréhension de certains aspects du métier [NELLBORN, 1999]. Les modèles de domaine orientés objet, tels que définis par Nellborn, sont des représentations particulières d'un ou plusieurs aspects d'un type de métier spécifique (par exemple, le transport, la production de biens de consommation,…). Un exemple simple d'un modèle de domaine, pour la gestion d'un hôtel, est présenté en Figure 2. Ce

modèle représente les aspects structurels du domaine et prend la forme d'un diagramme de classes UML[9].

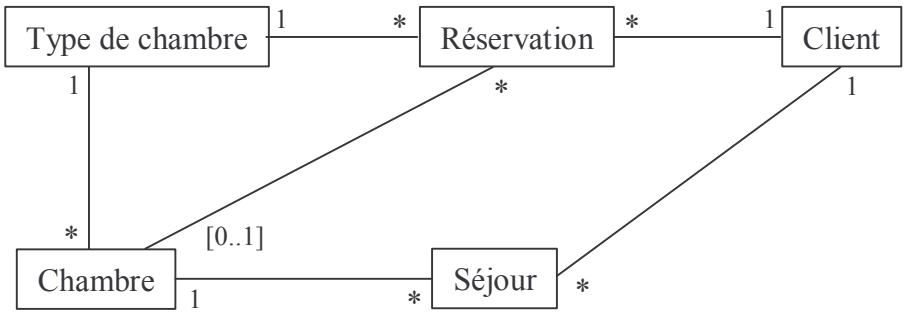


Figure 2 : Modèle de domaine de gestion d'un hôtel (d'après [SNOECK & POELS, 2000])

Les *abstractions de domaine* (ou *modèles de domaines génériques*) ont été proposées par Maiden & al. [MAIDEN & SUTCLIFFE, 1993A] [MAIDEN & SUTCLIFFE, 1993B] [MAIDEN & AL., 1994] dans le cadre d'une théorie sur la connaissance de domaine en ingénierie des besoins. Les abstractions de domaine sont des cas particuliers de modèles de domaine et ont pour vocation d'abstraire et de représenter les similarités entre modèles de domaine [SEMMAK, 1998]. Elles fournissent des structures réutilisables capturant les lois de fonctionnement d'un champ d'application. Leur représentation est basée sur un méta-modèle intégrant un ensemble de concepts et de liens entre ces concepts (notions de transition d'états et de structure d'objet) qui permettent de mettre en évidence les éléments discriminants entre des structures d'un domaine ou entre des domaines.

Maiden & al. proposent une méthode d'identification d'abstractions de domaine basée sur la recherche d'analogies, composée de cinq étapes :

- comparaison d'éléments selon quatre types de contraintes : contraintes structurelles, sémantiques, pragmatiques et d'abstraction,
- regroupement des éléments présentant des similarités,
- abstraction de ces groupes d'éléments et représentation à l'aide du modèle d'abstractions de domaine,
- classement de l'abstraction de domaine obtenue dans une hiérarchie des abstractions de domaine,
- détermination des différences entre le nouveau modèle et ceux auxquels il est lié.

[9] UML (*Unified Modeling Language)* est un langage de modélisation orientée-objet normalisé. Il propose des concepts, une syntaxe et une sémantique avec une notation sous forme graphique. Cf. [MULLER & GAERTNER, 2000] ou [BOOCH & AL., 1999] pour de plus amples détails.

b) *Applicabilité*

Les modèles de domaine sont utilisés en amont du processus de développement pour représenter les connaissances relatives à l'expression des problèmes et sont la base pour créer d'autres avoirs réutilisables, comme par exemple les architectures à base de composants logiciels [JOHN & DÖRR, 2003].

Les abstractions de domaine sont des structures génériques qui peuvent être réutilisées dans le processus d'ingénierie des besoins. Elles aident à caractériser le champ d'application du système d'information en cours de développement. Elles permettent la réutilisation d'architectures et réduisent ainsi le travail d'assemblage du concepteur d'applications. Cependant, les liens prédéfinis entre entités peuvent réduire la réutilisabilité de cette forme d'avoir logiciel.

2.3.4.2. Les patterns

a) *Définition et principes*

La notion de *pattern*[10] a été introduite en 1977 par Christopher Alexander [ALEXANDER & AL., 1977] [ALEXANDER, 1979] dans le domaine de l'architecture des bâtiments et fut ensuite étendue au domaine du génie logiciel et de la programmation orientée-objet par Beck & Cunningham [BECK & CUNNINGHAM, 1987]. Par la suite, des patterns dédiés à l'analyse orientée-objet et à la conception de logiciels ont été respectivement proposés par Coad [COAD, 1992] et Gamma & al. [GAMMA & AL., 1995]. Ces différents patterns, relevant du génie logiciel, ont été qualifiés de *patterns logiciels* (*software patterns*).

Chaque ouvrage ou même article de recherche propose sa propre définition du concept de pattern. Alexander définit un pattern en dehors du domaine de l'ingénierie informatique : « *un pattern décrit un problème se produisant très fréquemment dans un environnement et l'architecture de la solution à ce problème de telle façon qu'il est possible d'utiliser cette solution plus d'un million de fois sans jamais l'adapter deux fois de la même manière* ».

Un pattern logiciel relève de la même définition et a pour objectif de guider les développeurs en présentant et en explicitant une solution à un problème de construction logicielle dans un contexte spécifique. Le contexte décrit des situations récurrentes dans lesquelles le pattern est applicable. Le problème se réfère aux forces, i.e. aux buts et aux contraintes qui surviennent dans ce contexte. Généralement, le problème décrit quand le pattern doit être appliqué. La solution représente les éléments

[10] L'utilisation de l'expression en anglais étant fortement majoritaire et la traduction française non encore figée (patron, gabarit, motif, modèle), nous préférons garder le terme « pattern ». En outre, l'utilisation du terme anglo-saxon dans les articles de recherche francophones est très courante.

qui constituent le pattern, les relations entre ces éléments ainsi que les responsabilités et collaborations, éléments qui peuvent être appliqués pour résoudre ces forces. Un pattern constitue ainsi un moyen de structurer des descriptions de solution. Pour représenter et documenter le triplet <problème, contexte, solution>, un pattern possède une forme canonique comprenant des rubriques dont la nature et le nombre varie selon les auteurs [BUSCHMANN & AL., 1996] : un nom, des objectifs, une force, une solution, quelques exemples, le contexte résultant (quelles forces sont laissées non résolues ?), le choix de conception (d'où vient le pattern ?), les utilisations connues, les raisons pour lesquelles les experts l'utilisent, les patterns liés,…

Un pattern n'est pas seulement la description d'une solution à un problème dans un contexte spécifique, mais peut être aussi perçu comme un artefact cognitif. Différentes approches partagent la même perception de la connaissance comme étant un ensemble d'objets pouvant être stockés dans des référentiels et ainsi être disponibles pour la réutilisation. Un pattern peut dès lors être considéré comme une solution pour préserver, stocker et réutiliser la connaissance. Les patterns viennent de l'expérience d'experts concepteurs ou programmeurs et décrivent un savoir-faire qui peut être réutilisé. Les patterns permettent aussi de documenter les solutions proposées : il suffit au concepteur d'indiquer les patterns qu'il a utilisés pour qu'un "réutilisateur" averti comprenne la solution, sans avoir à reprendre chaque détail pour appréhender l'ensemble.

Les patterns ne sont pas une méthode de développement. Néanmoins, au cours du processus de développement logiciel, ils complémentent les méthodes et processus existants [SCHMIDT & AL., 1996]. De nombreux patterns ont été proposés traitant divers problèmes relatifs au développement logiciel. Selon que le problème traité apparaît lors de la capture des besoins, de l'analyse, de la conception ou de l'implantation du SI, différents types de patterns peuvent être respectivement distingués : les *patterns de capture des besoins* (*requirements patterns*) [ADAMS & AL., 1995], les *patterns d'analyse* (*analysis patterns*) [COAD, 1996] [FOWLER, 1997], les *patterns de conception* (*design patterns*) [GAMMA & AL., 1995] et les *patterns d'implantation* ou *idiomes* (*idioms*) [COPLIEN, 1992]. Les *patterns de capture des besoins* documentent les besoins des utilisateurs et le comportement générique du système à un haut niveau d'abstraction. Les *patterns d'analyse* ont pour but de faciliter et guider les étapes de la phase d'analyse en fournissant un cadre pour la modélisation du domaine d'application et des solutions réutilisables. Les *patterns de conception* donnent un nom, isolent et identifient les principes fondamentaux d'une structure générale pour en faire une ressource utile et réutilisable lors de la conception de systèmes [LARMAN, 2002]. Par exemple, lors de la conception d'un système orienté objet, un pattern de conception va décrire les classes et les instances intervenant dans la solution, leurs attributs, leurs rôles, les modalités de collaboration et la contribution de chaque élément décrit à l'obtention de la solution ciblée dans un contexte particulier. Les *patterns d'implantation* sont des

patterns de bas niveau qui décrivent comment implanter dans un langage particulier certaines caractéristiques généralement absentes.

Enfin, les patterns peuvent avoir différents niveaux de généralité. Certains peuvent être spécifiques à une application ou à un domaine d'application, alors que d'autres peuvent être des patterns généraux s'appliquant quels que soient les domaines d'application (par exemple, les patterns de conception proposés par Gamma & al. [GAMMA & AL., 1995]).

b) *Applicabilité*

Les patterns permettent la réutilisabilité au niveau des modèles - ce qui constitue le niveau d'abstraction le plus élevé - et peuvent ainsi être qualifiés de "hautement réutilisables". Ils couvrent le cycle de développement du logiciel allant de la capture des besoins, l'analyse et la conception de l'architecture globale de l'application à des détails très spécifiques de conception physique ou d'implantation. Ils peuvent être appliqués quel que soit le niveau d'abstraction ou l'étendue de l'application à développer, du moment que leurs critères d'utilisation sont satisfaits (contexte).

Certains patterns sont cependant difficiles à assimiler. Soit ils sont par nature conceptuellement difficiles à appréhender, soit certains noms ou descriptions fournis ne sont pas clairs. En outre, certains patterns prennent en compte les besoins d'experts, alors que d'autres sont dédiés à des débutants ; peu sont adaptés à ces deux publics. Les patterns sont classifiés en fonctions d'axes variés tels l'objectif, l'étendue ou la justification ; ces catégories sont utiles aux personnes qui maîtrisent déjà les concepts des patterns mais pas toujours compréhensibles par des débutants. Enfin, l'application des patterns est aussi basée sur l'expérience et les connaissances générales du concepteur ou développeur d'application et est d'autant moins évidente quand les patterns ne sont pas spécifiques à un domaine d'application.

2.3.4.3. Les composants logiciels

a) *Définition et principes*

Il existe plusieurs définitions parfois contradictoires d'un *composant logiciel* (*software component*) [BUTLER, 1995] [BUTLER & AL., 1995]. Pour Szyperski [SZYPERSKI & AL., 2002], il s'agit d'une unité de composition présentant des interfaces contractuelles et des dépendances explicites avec son contexte. Cette définition présente un composant logiciel comme étant un objet entièrement indépendant et pouvant communiquer avec son environnement par l'intermédiaire d'interfaces bien définies [PRAEHOFER & AL., 2000]. Le but principal d'un composant logiciel est de permettre de réutiliser une classe en facilitant son intégration dans le logiciel en cours de développement.

Pour qu'un composant logiciel soit utilisé dans un certain nombre d'applications, il faut qu'il soit interopérable et utile. L'interopérabilité désigne à la fois la faculté pour un composant de pouvoir communiquer avec d'autres composants résidants sur des plates-formes de types différents

(*interopérabilité de plate-forme*) et l'indépendance du composant par rapport à son langage d'implantation (*interopérabilité de langage*). L'interopérabilité de langage garantit la compatibilité des logiciels et facilite ainsi le processus d'assemblage et d'intégration. L'utilité du composant signifie qu'il comble un besoin de l'application en cours de développement.

Cette notion de composant logiciel intègre deux acceptions : les composants binaires et les composants définis au sein d'une architecture logicielle donnée. L'approche des composants binaires est celle de l'industrie et repose sur le concept de "boîte noire" : l'assembleur de composants a accès aux services fournis et requis par celui-ci, spécifiés au sein d'une interface, mais pas à son fonctionnement interne, i.e. les détails d'implantation. L'approche des architectures logicielles vient du monde académique, une architecture logicielle correspondant alors à une interconnexion de composants. La définition d'un composant logiciel y est plus floue et celui-ci peut ainsi être un ou plusieurs module(s) ADA, objet(s) C++, procédure(s) Pascal,… L'orientation est aujourd'hui vers la spécification de composants logiciels de forte granularité pour pouvoir appliquer la réutilisation aux phases amont du développement logiciel : « *si les premiers composants étaient de simples classes d'objets dépendantes d'un langage de programmation, la tendance aujourd'hui est de développer des composants de plus forte granularité et réutilisables au niveau de l'analyse des besoins et de la conception* » [CAUVET, 1998].

Trois technologies de composants logiciels sont actuellement sur le marché : Microsoft COM/COM+[11], JavaBeans / Enterprise JavaBeans (EJB) et CORBA[12] Component Model (CCM). Ces différentes technologies procurent notamment un degré d'interopérabilité des composants différent. Si le génie logiciel basé sur les composants est un domaine encore en évolution, des solutions techniques, comme les JavaBeans et les objets COM, existent déjà depuis longtemps sur le marché et permettent aux développeurs de réutiliser des composants logiciels [KOZACZYNSKI & BOOCH, 1998]. Ces solutions sont très répandues et sont utilisées quotidiennement.

b) *Applicabilité*

Les composants logiciels sont utilisables lors de la phase d'implantation d'une application. Ils ne permettent ainsi que de réutiliser du code source et ne répondent pas encore réellement à la problématique de la réutilisation dans les phases conceptuelles du développement logiciel. En outre, ils favorisent une appréhension morcelée du problème, limitant la perception globale des architectures techniques ou des organisations.

[11] COM : Component Object Model

[12] CORBA : Common Object Request Broker Architecture

2.3.4.4. Les frameworks

a) *Définition et principes*

Un *framework*[13] est une application semi complète, réutilisable, qui peut être spécialisée pour produire des applications adaptées [FAYAD & SCHMIDT, 1997A]. Un framework capture l'architecture d'une famille d'applications pour un domaine particulier et permet d'obtenir des applications de structures semblables favorisant ainsi la fabrication et la maintenance ; il exprime cette architecture avec un ensemble de classes abstraites et concrètes et leurs collaborations ; de plus, il implémente cette conception. En fonction du type de programmation, le développeur d'application complètera le framework, soit par instanciation et dérivation des classes le constituant (programmation objet), soit en écrivant le corps de certaines fonctions (programmation fonctionnelle). Le développeur écrit ainsi les fonctions manquantes qui particularisent son application et qui sont en fait des fonctions clientes.

Plus précisément, dans une programmation orientée objet, un framework est constitué d'un ensemble de classes, une bibliothèque de composants liés par des patterns de conception et dont les interactions sont prévues par le concepteur du framework. Ces composants comportent une partie codage : ils sont écrits à l'aide d'un langage de programmation.

Un framework n'est pas une simple bibliothèque de classes : il incorpore en plus des connaissances sur la conception de cette famille d'applications et impose un schéma de collaboration entre les classes. Les frameworks appliquent une stratégie dite "d'inversion de contrôle". Ainsi, contrairement aux bibliothèques de classes - où le programme développé instancie et donne le contrôle aux classes de la bibliothèque -, c'est le framework qui appelle les méthodes définies par le développeur d'application.

La conception d'un framework est basée sur l'analyse des aspects stables (ou *frozen spots*) et des aspects variables (ou *hot spots*) des applications à produire.

Des frameworks ont été conçus dans plusieurs domaines d'application : les logiciels graphiques [ROBBINS, 1998], les composants multimédias [POSNAK & AL., 1997], les réseaux et communications [SCHMIDT, 1998], les systèmes multi-agents [BRUGALI & SYCARA, 1998], le secteur bancaire [BÄUMER & AL., 1997], etc. Les plus connus sont ceux pour le développement d'applications dans des langages de programmation (AWT de Java, MFC de Microsoft,…), les produits pour le développement rapide d'applications (Visual C++, C++ Builder, etc.) et les frameworks d'interface graphique et d'application (par exemple, Gnome définit une architecture de communication entre composants/applications fondée sur CORBA).

[13] Charpente, canevas, architecture, cadriciel.

Pour que les frameworks soient utiles et utilisés, il est indispensable de disposer d'une bonne documentation et de processus de formation bien établis. Le développeur aura besoin de comprendre comment fonctionne le framework pour l'adapter. La présence de bons tutoriaux, permettant d'appréhender le framework par l'exemple, est notamment indispensable. En outre, l'apprentissage de l'utilisation efficace d'un framework exige du temps et des efforts. Fayad & Schmidt [FAYAD & SCHMIDT, 1997B] [FAYAD & SCHMIDT, 1997A] estiment que pour devenir un expert de frameworks tels que MFC, par exemple, il est nécessaire d'investir de 6 à 12 mois.

b) *Applicabilité*

Les frameworks permettent à la fois de réutiliser des éléments de conception – puisque l'architecture, les modes d'interaction et de communication sont conservés -, mais aussi du code. Ainsi, ils restent ancrés dans la mise en œuvre dans le sens où ils sont définis dans un langage de programmation spécifique. Le principal inconvénient des frameworks réside dans cet ancrage car certains concepts ne peuvent pas être exprimés correctement dans tous les langages et les frameworks deviennent de plus en plus spécifiques au langage dans lequel ils sont exprimés. Notons enfin que certains frameworks permettent plus une réutilisation de code source que de réels éléments de conception.

Nous avons présenté dans les sections précédentes le concept d'ingénierie des systèmes d'information et avons montré que cette ingénierie nécessitait l'usage de techniques de réutilisation efficaces afin de permettre un développement rapide et de meilleure qualité. Cette constatation est d'autant plus vraie quand elle s'applique à des systèmes d'information complexes tels que les systèmes d'information coopératifs, cadre dans lequel nos travaux se positionnent. Nous présentons plus en détail le concept d'ingénierie des systèmes d'information coopératifs en section suivante (§ 3) et explicitons en quoi l'utilisation de techniques de réutilisation est primordiale dans ce cadre.

3. … à l'ingénierie des systèmes d'information coopératifs

Avant d'exposer, en section 3.2, les principes de l'ingénierie des systèmes d'information coopératifs, nous précisons préalablement (§ 3.1) la notion de système d'information coopératif.

3.1. Les systèmes d'information coopératifs : définition et caractéristiques

L'informatisation a conduit au développement, dans les organisations, d'un nombre considérable de systèmes d'information chargés de stocker, d'organiser, d'interroger et de restituer des données utiles. Ces systèmes doivent collaborer afin de permettre une coopération à différents niveaux : coopération

des informations issues des SI nouveaux ou existants, coopération entre les acteurs de l'organisation ou coopération entre les organisations. Les solutions répondant à ces besoins reposent sur la notion de *système d'information coopératif*, dont l'objectif est d'assurer la coopération de systèmes d'information à différents niveaux. Nous définissons plus précisément le concept de système d'information coopératif en section suivante.

3.1.1. Définition

La définition littéraire du terme *coopérer* est *Agir conjointement avec quelqu'un* [LAROUSSE, 2004]. La *coopération* en informatique est une instanciation de cette définition en utilisant des moyens logiciels et matériels. Elle désigne une activité informatique menée par un groupe d'utilisateurs dans un espace informatique commun.

Issues principalement de deux courants de recherche, l'Intelligence Artificielle Distribuée (IAD) et les bases de données, la *coopération de systèmes d'information* et ses solutions, les *Systèmes d'Information Coopératifs* (SIC) constituent un domaine de recherche assez récent dont l'apparition date du début des années 1990 [BOULANGER, 1991] [PAPAZOGLOU & AL., 1992]. L'appel à communication de la conférence CoopIS'1994 introduit ainsi la notion de Système d'Information Coopératif [COOPIS'94, 1994] : *« Le paradigme pour la nouvelle génération de systèmes d'information implique un grand nombre de systèmes d'information distribués sur de vastes et complexes réseaux de communication (…). De tels systèmes d'information géreront et accèderont à de nombreuses sources d'information et services. Ils supporteront le travail humain individuel ou collaboratif. Les traitements seront réalisés de façon concurrente sur le réseau par des systèmes logiciels allant d'applications conventionnelles à des systèmes applicatifs avancés tels les systèmes experts et les systèmes multi-agents. L'information et les services seront disponibles sous de nombreuses formes à travers les applications existantes ou nouvelles (…). La communication entre les systèmes se fera de façon centralisée ou distribuée en utilisant des protocoles de communication conventionnels ou issus de l'Intelligence Artificielle Distribuée. Nous appellerons cette nouvelle génération de systèmes, les systèmes d'information coopératifs… ».*
Décrits en ces termes, les systèmes d'information coopératifs mettent l'accent sur les challenges technologiques à relever et reposent sur des avancées en terme de télécommunication, de matériels, de logiciels…
Plusieurs définitions plus contextualisées ont été assignées aux SIC. Dans l'approche des bases de données distribuées, un SIC « *intègre des sources d'informations distribuées - bases de données ou systèmes à bases de connaissances – pouvant utiliser des représentations de connaissances et de données hétérogènes* » [BRODIE & CERI, 1992]. La communauté de l'Intelligence Artificielle Distribuée définit un SIC comme « *un ensemble d'agents "computationnels" qui partagent continuellement des objectifs avec d'autres systèmes d'information, des agents humains aussi bien qu'avec l'organisation au*

sein de leur environnement opérationnel » [HERIN & AL., 2001]. Boulanger & Dubois [BOULANGER & DUBOIS, 1997] proposent une définition plus fédératrice dans laquelle un SIC est perçu comme *« un ensemble de composants plus ou moins autonomes, souvent préexistants qui travaillent de manière synergique en échangeant information, expertise et en coordonnant leurs activités* ». Ces définitions font apparaître trois facettes des SIC [DE MICHELIS & AL., 1997] [AGOSTINI & AL., 1998] :

- la facette "systèmes distribués". Cette facette inclut des types variés d'information, d'applications et de systèmes logiciels ou matériels existants, développés à l'aide de technologies conventionnelles telles que les langages de programmation, les systèmes de gestion de bases de données (SGBD),... La coopération doit traiter à ce niveau les problèmes d'hétérogénéité et donc d'incompatibilité entre les systèmes et doit ainsi permettre à tous les systèmes d'une organisation, ou de plusieurs organisations, d'échanger des données et d'utiliser des fonctionnalités d'autres systèmes. Cette facette considère ainsi que la coopération doit d'abord se situer au niveau des données des applications et nécessite la résolution des problèmes d'hétérogénéité des sources d'information (format de stockage des données, sémantique des données,…). Elle a trait aux recherches sur la coopération de sources de données et l'intégration de l'information.
- la facette "travail collaboratif" (workflow, processus métiers, CSCW[14],…). Cette facette concerne la manière dont des acteurs travaillant sur un processus métier ou un projet commun peuvent coordonner leurs activités, prendre en compte les contingences et changer leurs pratiques à travers les discussions et l'apprentissage. Le problème de coopération posé par cette facette est que la collaboration de groupes requiert un haut niveau de flexibilité des systèmes qui supportent le travail collaboratif. Cette facette a donné naissance aux recherches liées à la coopération de tâches et de processus, au groupware, à l'informatique organisationnelle,…
- la facette organisationnelle (processus métiers, objectifs stratégiques,…). Elle s'intéresse à la gestion du travail d'un point de vue organisationnel, sans se soucier des acteurs ou de la technologie. Elle adresse les aspects organisationnels globaux, incluant les objectifs métiers et organisationnels, les politiques,... Les "outils" de coopération, dans le cadre de cette facette, ont trait aux modèles de processus organisationnels, aux modèles de règles métier et aux modèles d'objectifs et d'interdépendances entre les agents organisationnels,…

Nos travaux se situent dans le cadre de la première facette et plus exactement des systèmes d'information coopératifs dédiés à la coopération de sources d'information hétérogènes (sources de données structurées, semi-structurées ou non structurées). Les principaux objectifs de tels systèmes sont :

[14] Computer Supported Cooperative Work.

- résoudre les différents types d'hétérogénéité (ou conflits) inhérents à la coopération de SI[15],
- traiter les requêtes soumises au SIC en respectant ses caractéristiques ou propriétés définis dans la section suivante (§ 3.1.2).

3.1.2. Caractéristiques et propriétés des SIC

Un système d'information coopératif repose sur un certain nombre de principes essentiels, à savoir l'*autonomie*, la *distribution* et l'*hétérogénéité* des systèmes composants[16] [SHETH & LARSON, 1990] [BREITBART, 1990]. D'autres propriétés, telles la *transparence*, l'*évolutivité* ou la *scalabilité* permettent d'évaluer la qualité du SIC [GOH & AL., 1994] [LIU & PU, 1995]. Ces caractéristiques et propriétés sont présentées dans les sections suivantes.

3.1.2.1. L'autonomie

L'*autonomie* permet à un système de devenir un composant du SIC sans que le partage de l'information ne remette en question le fonctionnement local de ce système. Nous distinguons trois types d'autonomie. L'*autonomie de conception* interdit tout changement dans la conception (modèle de données, langage d'interrogation, noms et contextes sémantiques des données, contraintes d'intégrité,..) et l'implantation (fichier, base de données,…) des composants du SIC, qui apparaissent ainsi comme des boîtes noires. Ce principe garantit la pérennité des applications héritées (*legacy systems*), c'est-à-dire préalablement existantes à la constitution du SIC. L'*autonomie d'exécution* assure la liberté du traitement des requêtes externes provenant de la coopération. Le système peut ainsi choisir l'ordre des requêtes ou refuser certaines d'entre elles, jugées incompatibles avec des contraintes particulières (sécurité, droits d'accès,…). L'*autonomie de communication* concerne la possibilité pour un composant de communiquer ou non avec d'autres composants et de choisir les modalités de cette communication. Enfin, l'*autonomie d'association* est la faculté donnée à un composant de choisir si il souhaite partager ses fonctionnalités ou ressources avec d'autres composants et lesquelles il souhaite partager.

[15] Par la suite, les termes « coopération de systèmes d'information » et « systèmes d'information coopératifs » s'appliqueront au cadre de la coopération de sources d'information.

[16] Nous considérons les termes « système composant », « système participant », « système d'information coopérant » ou « source d'information locale » comme synonymes.

3.1.2.2. L'hétérogénéité

L'*hétérogénéité* provient du développement autonome des systèmes et ainsi des différents choix faits au niveau de la structure et du type des systèmes composants, et du mode de représentation des informations dans chaque composant.

De nombreuses classifications ont été proposées[17] [KIM & SEO, 1991] [SHETH & KASHYAP, 1992] [GOH & AL., 1994] [BOULANGER, 1995] [SHETH, 1999] [JOUANOT, 2001]. La classification[18] présentée ci-après repose sur trois niveaux d'hétérogénéité : technique, structurelle et sémantique.
L'*hétérogénéité technique* se réfère à l'hétérogénéité des plates-formes, i.e. des systèmes d'exploitation et des systèmes matériels, sur lesquelles sont situés les systèmes composants.
L'*hétérogénéité structurelle* traduit entre les systèmes locaux une différence :

- de modèle de données (e.g., réseau, relationnel, ou orienté objet),
- de langage de requêtes (e.g., DML[19] Codasyl, SQL[20] ou OQL[21]),
- de représentation des informations entre composants coopérants, à savoir :
  - ⇨ Les *conflits schématiques* se réfèrent à l'utilisation de concepts différents pour représenter la même information. Par exemple, une information peut être représentée par un attribut dans un système composant ou par une table dans un autre.
  - ⇨ Les *conflits de généralisation/spécialisation* sont liés aux différences de hiérarchisation d'une même information (utilisation de plusieurs concepts pour représenter une information ou d'un concept unique plus général). Par exemple, un hôpital pourra définir plusieurs entités pour représenter ses types d'employés (infirmiers, médecins, personnel administratif), alors qu'un autre définira une unique entité (personne) pour gérer tous ses employés.
  - ⇨ Les *conflits d'agrégation* concernent les niveaux de granularité différents entre les systèmes. Ainsi la valeur d'un attribut d'un système composant pourra être équivalent à l'agrégation des valeurs de plusieurs attributs dans un autre système. Par exemple, l'attribut "adresse" dans un système pourra être décomposé en plusieurs attributs "numéro de rue", "nom de la rue", "code postal" et "ville" dans un autre système.

[17] Pratiquement chaque projet coopératif propose sa propre taxonomie des types d'hétérogénéité. Nous le verrons plus loin…

[18] Classification retenue par l'équipe de recherche MODEME dans le cadre de son projet de coopération nommé ACSIS (Agents pour la Coopération de Systèmes d'Information Sécurisés) [DUBOIS, 1997].

[19] Data Manipulation Language.

[20] Structured Query Language

[21] Object Query Language

⇨ Les *conflits de types* se réfèrent à des différences de types de données entre attributs porteurs de la même information. Par exemple, une date pourra être représentée sous la forme d'une chaîne de caractères dans un système composant et par un type "Date" dans un autre composant.

L'*hétérogénéité sémantique* représente les conflits de domaine de définition, mais aussi les différences dans la signification ou l'interprétation d'une même donnée. Elle englobe, plus précisément, les conflits suivants :

- Les *conflits de noms* relèvent de l'autonomie des systèmes à nommer les informations. Ces conflits de noms peuvent être des conflits de *synonymie* (des termes différents expriment la même information), d'*homonymie* (des informations différentes possèdent le même nom), de *polysémie* (un même terme change de signification en fonction du contexte), d'*hyperonymie* et d'*hyponymie* (un terme peut être plus général ou plus spécifique qu'un ou plusieurs autres termes).
- Les *conflits de valeurs* sont liés à des manières différentes de coder la même information dans des systèmes différents. Plusieurs conflits de valeurs ont été recensés. Les *conflits de représentation des données* résultent de l'utilisation de valeurs différentes pour représenter la même information. Par exemple, les valeurs de l'attribut "permis de conduire" pourront être {"oui", "non"} dans un système alors qu'elles seront {0,1} dans un autre. Les *conflits de précision des données* surviennent quand une même information a un degré de précision différent dans des systèmes composants (par exemple, une note sera représentée par une valeur allant de 0 à 20 dans un système alors qu'elle sera représentée par une lettre {A, B, C, D, E} dans un autre). Enfin, les *conflits d'échelle* relèvent de la représentation d'une même information en utilisant des unités ou mesures différentes (par exemple, une taille peut être représentée en centimètres et en pouces dans des systèmes différents).

3.1.2.3. La distribution

Un troisième problème, orthogonal aux deux problèmes précédemment exposés, est le problème de la *distribution physique* des sources d'information. Les données sont réparties entre plusieurs systèmes d'information localisés sur un même site ou géographiquement distribués, mais interconnectés par un système de communication. Le problème de distribution des données est résolu par des techniques nombreuses et disparates telles HTTP[22] ou CORBA/IIOP[23]. CORBA permet, par exemple, à des composants d'être utilisés tout en ignorant leur localisation physique.

[22] HyperText Transfer Protocol

[23] Common Object Request Broker Architecture / Internet Inter-ORB Protocol

3.1.2.4. La transparence

La transparence a trait à l'intégration : un système d'information parfaitement intégré doit donner l'illusion que les utilisateurs interagissent avec un SI centralisé, fonctionnant localement, homogène et cohérent. Trois types de transparence peuvent être distingués :

- La *transparence de localisation* permet à l'utilisateur de ne pas se soucier de la localisation physique de l'information.
- La *transparence au niveau de la modélisation des données* : les utilisateurs n'ont pas besoin de connaître les différents noms des entités ou attributs des sources d'information.
- La *transparence de langage* permet aux utilisateurs de ne pas être obligé d'apprendre un langage de requête spécifique pour requêter les données.

La transparence implique ainsi un découplage, i.e. la présence d'un niveau d'abstraction, entre l'utilisateur (i.e. l'énoncé des besoins) d'une part et les sources d'information locales d'autre part.

3.1.2.5. L'évolutivité

L'*évolutivité* est la capacité à prendre en charge les changements qui peuvent apparaître au sein d'une architecture coopérative. Ces changements interviennent à deux niveaux :

- au niveau du SIC (ajout / retrait d'un système participant),
- au niveau des systèmes composants (modification d'un schéma, de l'implantation ou de la sémantique d'un composant,…)

3.1.2.6. La scalabilité

La *scalabilité*[24] est définie comme la capacité d'un SIC à gérer un nombre important de systèmes composants. La scalabilité garantit un bon niveau de performance dans l'exécution d'une requête lorsque le nombre de sources d'information augmente.

3.1.3. Taxonomie des systèmes d'information coopératifs

De nombreuses solutions ont été proposées pour résoudre les différents types d'hétérogénéité et permettre ainsi la coopération de sources d'information. Elles permettent toutes, au moins, de résoudre

[24] Nous avons choisi d'utiliser le substantif « scalabilité » comme traduction du terme anglo-saxon « scalability », en l'absence de traduction convenable (la traduction "passage à l'échelle" employée en recherche d'information, ne nous satisfait pas).

certains problèmes liés à l'hétérogénéité structurelle - notamment les différences entre les modèles de données des systèmes composants - et garantissent l'autonomie des sources d'information locales.

Nous avons recensé plus d'une centaine de projets, anciens ou récents, ayant conçu ou développé des SIC. Ces derniers peuvent être classés[25] selon deux approches : l'*approche fédérée* et les *approches à base de médiation*. Ces deux approches sont présentées dans les sections suivantes (§ 3.1.3.1 et 3.1.3.2).

3.1.3.1. L'approche fédérée

Un système fédéré, selon la définition de Sheth & Larson [SHETH & LARSON, 1990], contrôle et coordonne la manipulation d'une collection de systèmes de gestion de bases de données autonomes. Chacun des systèmes de base de données participant à une fédération continue à être utilisé par un ensemble d'applications, de manière locale. L'approche fédérée est orientée vers l'intégration partielle des structures de données. Sheth & Larson distinguent deux types d'intégration : l'intégration sous forme d'un schéma fédéré (couplage fort) et l'intégration sous forme de vues fédérant plusieurs SI (couplage faible).

a) *Principes de l'approche*

L'approche fédérée est basée sur la présence de cinq niveaux de schémas [SHETH & LARSON, 1990] (Cf. Figure 3) :

- Le *schéma local* représente le schéma de chaque base de la fédération dans son modèle local.
- Le *schéma composant* résulte de la traduction d'un schéma local en un schéma exprimé à l'aide d'un modèle de données commun, appelé modèle canonique[26]. Cette transformation permet de résoudre les problèmes d'hétérogénéité des modèles de données.
- Le *schéma exporté* est une vue d'un schéma composant spécifiant les données qui seront mises à disposition de la fédération.
- Le *schéma fédéré* peut consister à intégrer statiquement des schémas d'export ou peut être une vue dynamique sur plusieurs schémas d'export. Il peut y avoir plusieurs schémas fédérés répartis en fonction des applications clientes ou des utilisateurs. Un schéma fédéré correspond à un domaine d'application.
- Le *schéma externe* consiste en une vue d'un schéma fédéré adaptée à une application cliente ou une catégorie d'utilisateurs. Les schémas externes permettent ainsi de limiter l'étendue et la complexité des schémas fédérés.

[25] … parfois avec difficulté…

[26] ou modèle pivot.

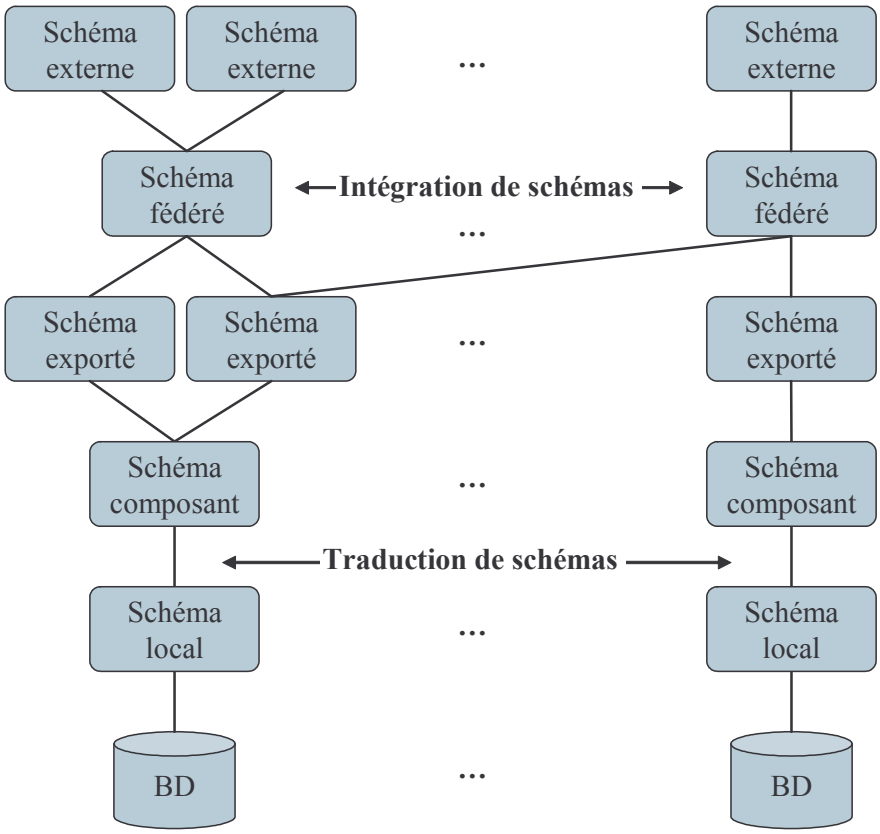


Figure 3 : Les cinq niveaux de schémas de l'approche fédérée

Les quatre premiers niveaux de schémas utilisent un modèle canonique permettant d'homogénéiser les schémas locaux représentés dans différents modèles de données (relationnel, objet, hiérarchique,…). Ce modèle canonique est en général un modèle orienté objet, en raison de ses capacités d'abstraction et de représentation. Le modèle canonique assure ainsi la résolution de certains problèmes d'hétérogénéité structurelle, telle que nous l'avons définie précédemment (§ 3.1.2.2). Cependant, les problèmes d'hétérogénéité sémantique restent implicites et doivent être pris en compte (en général, manuellement) lors de l'intégration des schémas.

Cette approche nécessite la présence d'un dictionnaire de données (répertoire des méta-données des schémas,…), appelé dictionnaire de la fédération, afin de pouvoir réaliser l'intégration.

b) Les systèmes fédérés fortement intégrés

Les systèmes fortement intégrés[27] permettent aux applications ou utilisateurs de formuler leurs requêtes sur un schéma fédéré "global" unifié. Les schémas exportés par les différentes sources de données sont ainsi intégrés en un schéma fédéré unique. La définition du schéma fédéré global demande, d'une part,

[27] Ou *systèmes fédérés fortement couplés*

l'analyse et la comparaison des différents schémas et, d'autre part, la résolution des conflits éventuels [KIM & AL., 1994]. Définir un schéma global est une tâche difficile, limitant les capacités du système à évoluer et à intégrer de nouvelles sources.
Les caractéristiques de tels systèmes sont :

- l'administration de la fédération a un contrôle total sur la création et la maintenance des schémas fédérés et des schémas exportés,
- la transparence de localisation des données et la distribution des données,
- les schémas fédérés sont statiques et donc rarement modifiés,
- l'autonomie des sources locales n'est pas pleinement respectée.

Les problèmes liés à l'hétérogénéité structurelle sont résolus par le processus d'intégration. Ce dernier résout également implicitement les problèmes liés à l'hétérogénéité sémantique du fait que le schéma fédéré global représente l'organisation des informations pour un domaine d'application particulier. La transparence est assurée, il suffit de poser une requête sur le schéma fédéré. Néanmoins l'extensibilité et la scalabilité sont les critères qui s'adaptent très mal à l'intégration. La fédération est une approche à préférer dans le cadre d'une coopération de quelques SI et peu évolutive.

Les projets coopératifs Mermaid [TEMPLETON & AL., 1986] [TEMPLETON & AL., 1987] et Mind [DOGAC & AL., 1995A] [DOGAC & AL., 1995B] [OZCAN & AL., 1996] relèvent par exemple de cette approche.

c) *Les systèmes fédérés faiblement intégrés*

Les systèmes fédérés faiblement intégrés[28] n'exigent pas la définition d'un schéma global et l'intégration des informations des différents systèmes composants est réalisée en fonction des besoins des utilisateurs. Ces systèmes permettent d'intégrer simplement de nouvelles sources de données.
Les caractéristiques de tels systèmes sont :

- l'utilisateur de la fédération est responsable de la création et de la maintenance de la fédération,
- l'utilisateur doit être bien informé sur la structure et l'information contenue dans les schémas exportés,
- les schémas fédérés peuvent être créés ou supprimés selon les besoins,
- les composants locaux sont autonomes mais uniquement disponibles en lecture seule,
- l'évolutivité des composants locaux.

[28] Ou *systèmes fédérés faiblement couplés*

Les SIC proposés dans le cadre des projets Omnibase [RUSINKIEWICZ & AL., 1988], Pegasus [AHMED & AL., 1991] et Myriad [CLEMENTS & AL., 1993] relèvent par exemple de cette approche.

Relèvent également de cette approche, les systèmes multibases (MRDSM [LITWIN, 1985], VIP-MDBS [KUHN & AL., 1988], INFOBASE [LYNGBAEK & MCLEOD, 1983],…). De tels systèmes n'utilisent pas de schéma partiel ou global prédéfini mais reposent sur l'utilisation d'un langage de manipulation multibase (en général, SQL étendu ou OQL) et, éventuellement, d'un langage de définition de données [LITWIN & AL., 1982] [LITWIN & AL., 1990]. Le langage de manipulation multibase comprend un langage de manipulation étendu et enrichi permettant d'exprimer les opérations des données au niveau multibase (capacités d'adressage des sources d'information, de préfixation des données manipulées par le nom de la base, de définition de vues multibases,…). Un langage de définition des données peut lui être adjoint et permet l'expression des structures de données et la définition des dépendances, i.e. contraintes d'intégrité, entre les bases composantes. L'approche multibase est évolutive mais n'offre pas de transparence à la localisation des données pour les utilisateurs ni une scalabilité importante. Les conflits schématiques et sémantiques, n'étant pas traités, restent à la charge des utilisateurs.

3.1.3.2. Les approches à base de médiation

a) *Principes*

Les approches de type médiation proposent une organisation modulaire des systèmes d'information de manière à tirer parti de l'accès à de nombreuses sources de données connectées par des réseaux. La médiation repose sur un composant central, appelé *médiateur*, qui simplifie, abstrait, réduit, combine et décrit les données [WIEDERHOLD, 1992] [WIEDERHOLD & GENESERETH, 1995] et est chargé des traitements permettant à l'application cliente d'obtenir les données extraites des sources de données locales. Pour cela, le médiateur encode les connaissances nécessaires aux transformations sur les données des sources. Un médiateur peut être spécialisé pour un ensemble de sources ou un domaine d'application. Plusieurs médiateurs peuvent être composés en une hiérarchie de modules entre l'application et un grand nombre de sources de données.

L'infrastructure de médiation a pour but de rassembler les connaissances permettant l'intégration des données dans des composants utilisables par plusieurs applications. D'autre part, l'infrastructure se doit d'être modulaire ; cette modularité permet à l'infrastructure de médiation d'évoluer facilement et notamment d'intégrer de nouvelles sources.

Une telle infrastructure de médiation reprend et généralise un certain nombre d'idées relatives aux systèmes de bases de données fédérés. Wiederhold [WIEDERHOLD, 1992] étend cependant les concepts de ces derniers, en décrivant notamment des techniques qui, selon lui, doivent être prises en compte

pour la définition des médiateurs. Il cite notamment le domaine de l'intelligence artificielle qui apporte la notion de déclarativité, les capacités d'explication et l'utilisation d'heuristiques pour explorer de grands espaces de recherche. Il cite le domaine de la logique appliquée aux bases de données qui permet une approche formelle des problèmes d'intégration. Il cite enfin les techniques qui permettent un accès efficace aux sources de données, notamment les vues matérialisées et l'optimisation sémantique.

Les idées et concepts présentés par Wiederhold ont été repris par la plupart des projets de médiation. Ainsi, les projets issus de ce domaine ont adopté une même architecture pour leur SIC. La figure suivante (Cf. Figure 4) décrit cette architecture basée sur trois niveaux : le premier niveau concerne les SI locaux, le deuxième niveau englobe les différents outils pour traiter les requêtes et résoudre les conflits et le troisième niveau est dédié aux applications et utilisateurs.

Médiateurs et *adaptateurs*[29] sont les composants du deuxième niveau du système de médiation. A chaque source de données est associé un adaptateur. Chaque adaptateur fournit la même interface d'accès vers les sources de données et masque ainsi l'hétérogénéité de la source associée. Le médiateur fournit un point d'entrée unique et une vue uniforme sur un ensemble de sources de données (il résout les problèmes d'hétérogénéité liés aux différences de modèles de données et de langages de requêtes en présentant les données dans le modèle de médiation[30]). L'application soumet des requêtes déclaratives au médiateur de manière à obtenir les données demandées par l'utilisateur.

Pour intégrer une nouvelle source de données, un adaptateur doit être développé. Cet adaptateur est ensuite enregistré auprès du médiateur. Lorsque l'application soumet une requête, celle-ci est optimisée par le médiateur, elle est ensuite exécutée. Le moteur d'exécution du médiateur envoie des sous-requêtes aux sources de données par l'intermédiaire des adaptateurs : chaque sous-requête envoyée par le médiateur est transformée par un adaptateur en une requête locale au format de la source de données. La réponse de la source est ensuite transformée par l'adaptateur au format du médiateur. Dans une dernière étape, le médiateur rassemble les résultats fournis par toutes les sources de données impliquées dans la requête, les combine et compose la réponse qu'il renvoie à l'application. Lors de cette étape, le médiateur résout certains conflits structurels et sémantiques.

L'intégration de nouvelles sources de données requiert le développement d'adaptateurs dont l'interface et les fonctionnalités sont précisément définies. Un système de médiation doit ainsi fournir les outils pour le développement rapide d'adaptateurs.

[29] ou *wrappers*.

[30] Un modèle de médiation est généralement défini comme une modèle canonique étendu à la représentation de la sémantique des informations. Il s'agit d'un modèle de représentation des méta-données des sources d'information locales, ainsi que des concepts sémantiques entre méta-données.

Notons enfin que médiateurs et adaptateurs sont souvent conçus et implantés, dans les projets récents, sous formes d'agents intelligents en raison de leurs fonctionnalités de distribution des connaissances et de l'exécution des tâches [CHAWATHE & AL., 1994] [GENESERETH, 1997].

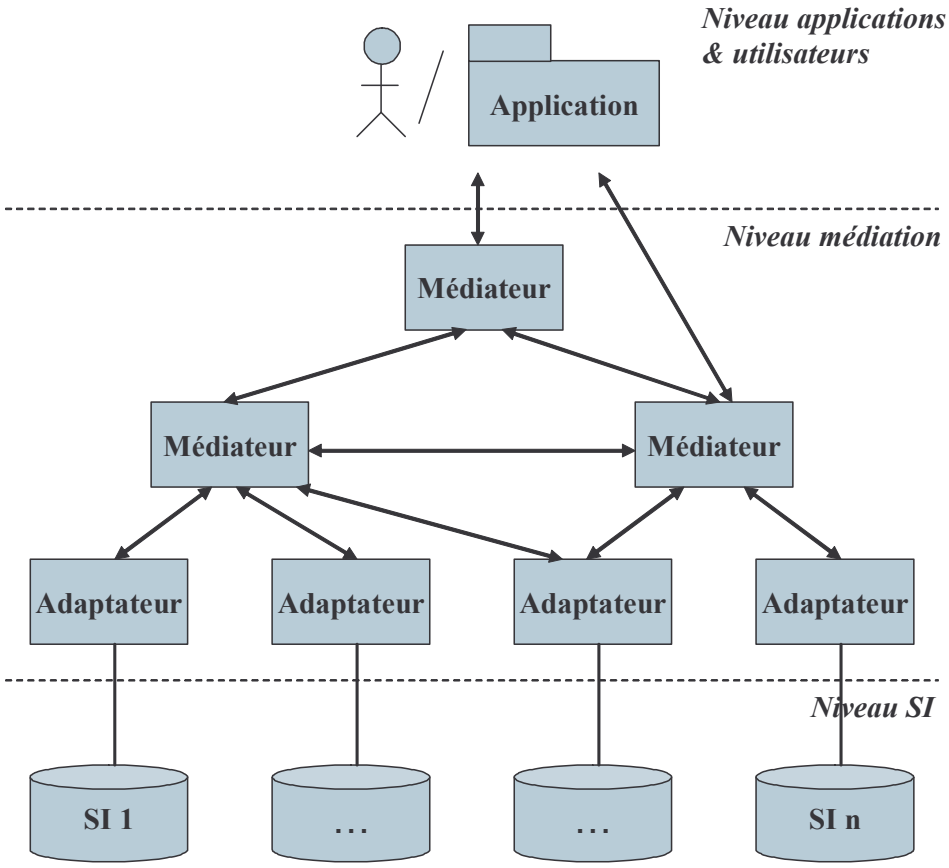


Figure 4 : Architecture basée sur un système de médiation

Deux types de médiation peuvent être distingués en fonction de la manière de gérer les requêtes et d'utiliser la sémantique dans le processus de médiation [JOUANOT, 2000] [JOUANOT, 2001] : la *médiation de schémas* et la *médiation de contextes*.

b) *La médiation de schémas*

La *médiation de schémas* construit au préalable une base d'informations prenant en compte les SI participants. Elle associe au médiateur un ensemble de connaissances qui localisent et intègrent a priori les informations pertinentes à un contexte d'utilisation. La résolution d'une requête et l'accès aux données distribuées suivent un plan d'exécution établi par des règles qui définissent où se trouvent les données pertinentes, comment elles sont transformées et combinées pour fournir un résultat exploitable. La notion de contexte est ici implicite: le schéma de médiation travaille au niveau de la structure des données et des outils simplifient la résolution à l'avance des nombreux conflits. Le schéma de médiation est formé d'interfaces qui peuvent être soit :

- Des *interfaces objet* définies sous la forme d'un ensemble de classes virtuelles. Le modèle choisi pour représenter ces interfaces est en général le modèle ODMG'93 [CATTEL & AL., 1993] étendu ou non. Les instances des classes virtuelles ne sont pas des objets stockés et gérés localement mais sont des instances d'objets distants attachés à un SI. Une base de connaissances met en correspondance ces classes virtuelles avec les éléments des schémas locaux. Les solutions mettant en œuvre des interfaces objet sont, par exemple, DIOM [LIU & PU, 1995] ou DISCO [TOMASIC & AL. 1995] qui utilisent un modèle objet basé sur ODMG'93 ou le projet ACSIS [BOULANGER & DUBOIS, 1997] [BOULANGER & DUBOIS, 2001] [DUBOIS & AL., 2002] qui repose sur l'application d'un modèle objet spécifique.
- Des *interfaces logiques* basées sur l'utilisation d'un modèle logique (logique terminologique ou logique déductive). Les solutions mettant en œuvre des interfaces logiques sont, par exemple, TSIMMIS [GARCIA-MOLINA & AL., 1997] ou YAT [SIMEON, 1999].
- Des *interfaces objet XML.* Le langage XML est utilisé comme modèle commun pour définir les interfaces de coopération et remplace petit à petit le modèle objet classique. MIX [LUDASCHER & AL., 1999] est un exemple de projet mettant en œuvre des interfaces XML : les wrappers fournissent un schéma d'export XML des sources d'information et permettent l'accès aussi bien à des sites web qu'à des SGBD ; le médiateur correspond à une vue sur les schémas XML.

Le modèle canonique choisi a un impact très important sur la capacité du SIC à pouvoir faire coopérer des systèmes intégrant des données structurées, semi-structurées ou non structurées.

La médiation de schémas est une extension directe de l'approche fédérée avec une meilleure évolutivité et souvent une meilleure scalabilité, notamment souvent permise par l'utilisation d'agents intelligents et de systèmes multi-agents.

c) La médiation de contextes

La *médiation de contextes* est basée sur une représentation explicite de la sémantique des données à travers la notion de contexte ; ce contexte peut être décrit à l'aide d'outils de type méta-données[31] [LIBOUREL & AL., 2003] ou ontologies[32]. Un contexte spécifie un ensemble de connaissances sur la structure, les valeurs ou les fonctionnalités d'un objet qui permet d'en saisir la sémantique.

La médiation de contextes repose sur l'intégration dynamique des informations en fonction du contexte de l'application ou de l'utilisateur. Chaque source d'information décrit le contexte de ses informations et doit se définir par rapport à un modèle de référence. Chaque système composant doit donc spécifier un contexte d'utilisation du modèle de référence ; à la charge du médiateur de réconcilier

[31] Les méta-données sont définies comme des données sur des données, i.e. des informations sur le stockage et la gestion des données. Elles décrivent le contenu de l'information des systèmes composants.

[32] Une ontologie est une description des concepts d'un domaine et des relations entre ces concepts. Les termes d'un vocabulaire liés par des relations sémantiques (synonymie, homonymie,…) forment une ontologie.

les différents contextes des composants. Dans la plupart des approches, le médiateur est associé à un contexte d'application qui utilise les concepts et le vocabulaire d'une ontologie, l'accès à un médiateur particulier définit alors le contexte d'une application. Deux types de médiation de contextes peuvent être distingués :

- l'*approche mono-domaine* repose sur un ensemble unique de connaissances (domaine unique) pour décrire la sémantique des informations. Un médiateur est alors associé à une ontologie unique représentant la connaissance du domaine d'application. La scalabilité est respectée mais reste limitée en raison du fait que chaque système composant doit définir le contexte de ses données par rapport à chaque domaine auquel il souhaite être rattaché. Les SIC basés sur la médiation de contextes et mettant en œuvre une approche mono-domaine sont, par exemple, SIMS [ARENS & AL., 1993] et COIN [GOH, 1997].
- l'*approche multi-domaines* repose sur la distribution des informations sur des domaines de connaissances différents, domaines en relation les uns avec les autres. Un médiateur prend ainsi en charge les ontologies liées aux différents domaines de connaissances et devra disposer d'outils spécifiques pour comparer et réconcilier des domaines distincts. Ces différents outils devront permettre de définir :
  - ⇨ des relations de synonymie, d'homonymie ou d'antonymie entre les termes désignant les concepts d'un domaine (termes d'une ontologie),
  - ⇨ des relations inter-ontologies explicitant les règles de mise en correspondance entre les concepts de domaines différents.

  Les solutions de médiation de contextes multi-domaines sont, par exemple, OBSERVER [MENA, 1999] et INFOSLEUTH [BAYARDO & AL., 1997]. Ces solutions utilisent différents moyens pour représenter et lier des domaines sémantiquement différents tels les treillis d'ontologies, les hiérarchies d'ontologies ou les agents intelligents et systèmes multi-agents.

La médiation de contextes assure une résolution dynamique des conflits et ne requiert pas la connaissance a priori des SI participants (pas de schéma de médiation).

3.1.3.3. Comparaison des approches

L'objectif de cette section est de comparer les différentes approches coopératives présentées dans les sections précédentes, d'étudier l'évolution de la coopération mais aussi de montrer la pluralité des concepts et techniques utilisés au sein des SIC inhérents à chaque approche. Chaque approche est évaluée en fonction des points suivants :

- modèle canonique utilisé,

- langage d'interrogation de la coopération,
- représentation de la sémantique du domaine d'application,
- respect des propriétés pour la coopération,
- types d'hétérogénéité prise en compte,
- type de proposition (projet de recherche, projet fournissant un prototype ou une application opérationnelle),
- modèle d'architecture.

Les différents points de comparaison cités ci-dessus ne se veulent pas exhaustifs, mais permettent d'illustrer les différences ou points communs entre les approches coopératives ou entre les SIC (techniques et architectures sous-jacentes,…) à l'intérieur de chaque approche (Cf. Tableau 1). Les approches, ou plus exactement les SIC, peuvent être comparés sur de nombreux autres points, comme par exemple, les techniques d'optimisation ou de traitement des requêtes (Cf. [OUZZANI & BOUGUETTAYA, 2004]), etc.

REMARQUE — La comparaison des approches a notamment été réalisée à partir de l'étude des SIC suivants : ACSIS [BOULANGER & DUBOIS, 1998], CIS-MDBMS [FLORESCU & AL., 1996], Coin [GOH, 1997], Déca [BENSLIMANE & AL., 1998], DILEMMA [JOUANOT, 2001], DIOM [LIU & PU, 1995], Disco [TOMASIC & AL. 1995], DOK [TARI & AL., 1996], Garlic [CAREY & AL., 1995], Hermès [ADALI & AL., 1996], InfoMaster [GENESERETH & AL., 1997], Information Manifold [LEVY & AL., 1996], Infosleuth [BAYARDO & AL., 1997], Kraft [PREECE & AL., 1999], Mermaid [TEMPLETON & AL., 1986], Mind [DOGAC & AL., 1995A], MIRO-Web[33] [FANKHAUSER & AL., 1998], MRDSM [LITWIN, 1985], OASIS [MESROBIAN & AL., 1996], Observer [MENA, 1999], On2Broker [FENSEL & AL., 1998], Picsel [ROUSSET & AL., 2002], Scope [OUKSEL & AHMED, 1999], Tsimmis [GARCIA-MOLINA & AL., 1997], WebFindit [BOUGUETTAYA & AL., 2000] et YAT [SIMEON, 1999].

[33] MIRO-Web est une extension du projet IRO-DB [GARDARIN & AL., 1995].

a) *Modèle de représentation des données (modèle canonique)*

Un modèle canonique est un modèle commun de représentation des données assurant l'homogénéisation des modèles locaux. Les schémas des sources de données locales (ou schémas locaux) sont traduits en schémas composants à l'aide de ce modèle. Ce modèle peut-être :

- relationnel,
- orienté-objet (modèles objet issus des standards ODL[34] de l'ODMG[35], OGM[36],…)
- logique (modèles basés sur la logique terminologique tels KL-ONE, LOOM ou KIF[37] ou modèles basés sur la logique déductive tels LDL[38], F-Logic[39] ou Prolog),
- objet XML
- propriétaire (extension d'un modèle objet ou logique avec des concepts spécifiques adaptés à la représentation de données particulières : par exemple, FDS[40] pour les données spatiales,…).

Remarquons que certains SIC, relatifs principalement aux approches à base de médiation de contextes, n'utilisent pas de modèles canoniques.

b) *Langage d'interrogation de la coopération*

Un langage d'interrogation de la coopération (ou langage global) est étroitement lié au modèle utilisé pour représenter les données (modèle canonique). Il permet l'accès uniforme aux informations distribuées (données et connaissances) et peut être :

- un langage de programmation (C, C++, Java,…), souvent associé à un modèle relationnel ou orienté-objet. Le langage de programmation est souvent utilisé en combinaison avec un langage déclaratif.
- un langage déclaratif (langage dérivé de SQL ou OQL), associé à un modèle relationnel (SQL), orienté-objet (OQL, GeoOQL[41]) voire logique (par exemple, SQL associé à un modèle de logique terminologique KIF ou à un modèle de logique déductive LDL).
- un langage logique, dérivé, par exemple, de Prolog pour un modèle déductif ou de KL-ONE, LOOM ou KIF pour un modèle terminologique,… Ce langage est également associé à un modèle relationnel dans certains projets.

[34] Object Definition Language

[35] Object Database Management Group

[36] Open Geodata Model

[37] Knowledge Interface Format

[38] Logical Data Language

[39] Frame Logic

[40] Formal Data Structure

[41] GeoPOM Query Language

Remarquons que certains SIC soit n'utilisent pas de langage d'interrogation de la coopération, soit combinent plusieurs types de langages (par exemple, un langage déclaratif et un langage logique).

c) Représentation de la sémantique du domaine d'application

La représentation de la sémantique des informations concerne notamment le domaine d'application, i.e. le domaine général dans lequel les données trouvent une signification. Quatre types de représentation du domaine d'application sont possibles :

- Le schéma. Un schéma décrit comment sont structurées les informations d'un domaine. De nombreux projets utilisent un schéma de données pour exprimer le domaine d'application ; le modèle de données utilisé pour décrire ce schéma est généralement le même que le modèle commun de représentation des données.
- L'interface objet. Une interface objet décrit la sémantique du domaine d'application et nécessite l'utilisation d'un langage spécifique. Cette interface sert de point d'accès au système pour l'interroger.
- L'annuaire. Un annuaire regroupe une description résumée des différents systèmes d'un environnement de coopération. Il permet de localiser un SI en fonction d'une thématique particulière.
- L'ontologie. Elle explicite les concepts d'un domaine d'application. Cette ontologie peut soit être limitée à un vocabulaire de référence, soit utiliser un schéma terminologique ou logique. Cette ontologie peut être hiérarchisée ou non (hiérarchie d'ontologies) ou être une ontologie universelle prédéfinie. Chaque projet, basé sur l'usage d'ontologies, utilise un langage de règles pour formaliser les concepts et relations entre ces concepts : F-Logic, KIF, ONTOLINGUA,…

d) Respect des propriétés pour la coopération

Les différents projets respectent, à des degrés différents, les propriétés requises pour la coopération, à savoir, l'autonomie, la distribution, l'hétérogénéité, la transparence, l'évolutivité et la scalabilité (Cf. section 2.2.2).

- Les propriétés de distribution et d'hétérogénéité sont bien respectées par la plupart des projets et notamment par les projets de type médiation. L'autonomie est également respectée par la plupart des solutions, et notamment, l'autonomie de conception en raison de l'utilisation de schémas d'export ou d'adaptateurs. L'autonomie d'association est assurée principalement dans les approches de type médiation de contextes.
- La transparence. La transparence doit permettre à l'utilisateur de la coopération de formuler une requête globale sans se soucier de la modélisation et du langage d'interrogation des données, ni de la localisation des informations pertinentes. Le schéma global, utilisé dans les approches fédérées à couplage fort, résout les problèmes de localisation et de modélisation des données et

permet l'utilisation d'un unique langage d'interrogation. La gestion de la transparence, dans ce cas, est à la charge de l'administrateur de la fédération. Dans le cas des approches fédérées à couplage faible, la gestion de la transparence est gérée par l'utilisateur et est donc moins bien respectée. Les solutions de type médiation de schémas fournissent généralement des outils nombreux et variés permettant d'assurer la transparence à la localisation et de respecter l'indépendance du modèle local de représentation des données. Les approches de type médiation de contextes assurent toutes un niveau de transparence à la localisation, à la modélisation et à l'interrogation des données (les requêtes sont spécifiées selon les termes définis dans l'ontologie).

- L'évolutivité. L'évolutivité est permise par une intégration dynamique des informations en fonction de la sémantique de la requête ou de la source d'information. L'évolution est ainsi une caractéristique de base des systèmes de médiation de contextes. Les autres approches demandent une modification des connaissances globales ; l'évolutivité sera ainsi respectée en fonction de la présence ou non d'outils pour simplifier la gestion de ces modifications. De manière générale, l'évolutivité dans les approches fédérées à couplage faible nécessite des mises à jour importantes au niveau local si les sources sont nombreuses ; dans les approches fédérées à couplage fort, l'administrateur de la fédération devra procéder à une mise à jour manuelle de la coopération qui pourra être facilitée par la présence d'outils.
- La scalabilité. La scalabilité est conditionnée par les caractéristiques de résolution dynamique des requêtes, et, notamment, par l'utilisation d'une ou plusieurs ontologies. La scalabilité est donc une propriété de base des systèmes à base de médiation de contextes. La scalabilité est, en outre, encore mieux gérée avec un petit nombre de relations terminologiques (utilisation de plusieurs ontologies) plutôt qu'avec une multitude de relations contenue dans une ontologie globale. Les approches fédérées faiblement couplées assurent également un degré de scalabilité correct du fait que l'ajout de nouvelles sources reste relativement aisé en raison de la simplicité de leur modèle de données. Par contre, les approches à base de médiation de schémas nécessitent lors de l'ajout de systèmes composant, la mise à jour d'un ou plusieurs médiateurs, opération rendue difficile par la complexité des modèles. L'implantation des médiateurs sous forme de hiérarchies ou la présence d'outils spécifiques permet cependant d'accroître la scalabilité de telles architectures.

e) *Types d'hétérogénéité pris en compte*

Les projets relevant de l'approche fédérée assurent principalement la résolution des conflits de modèles de données et de langages d'interrogation. Les systèmes d'information coopératifs de type médiation de schémas ou médiation de contextes traitent la plupart des conflits structurels et certains conflits sémantiques. La résolution de ce dernier type de conflit varie cependant selon les SIC. Ainsi, les

systèmes coopératifs de type médiation de contextes résolvent généralement des conflits sémantiques plus complexes que ceux relevant de l'approche à base de médiation de schémas.

f) *Type de proposition*

- Recherche : le projet propose des concepts, une méthode de résolution des conflits et une architecture coopérative permettant la coopération de systèmes d'information, mais ni prototype, ni application opérationnelle.
- Prototype : type de proposition "Recherche" et fourniture d'un prototype.
- Application : type de proposition "Recherche" et fourniture d'une application opérationnelle.

g) *Modèles d'architecture*

Les architectures proposées ont généralement suivi l'évolution des techniques informatiques. Plusieurs types d'architectures peuvent être rencontrés.

L'architecture 3-tiers

L'application cliente manipule un module applicatif qui est le seul à accéder aux sources d'information. Ce module intermédiaire traite la requête globale.

Les objets distribués

Les objets distribués sont une évolution des applications 3-tiers, où les fonctions du module applicatif sont réparties sur plusieurs plates-formes (utilisation d'un middleware CORBA, en général, pour assurer la communication entre les différents objets).

Les wrappers

Dans les systèmes à base de wrapper, l'application cliente accède directement aux sources d'information locales via une interface uniforme représentée dans un modèle commun. Ce type d'architecture n'est pas adapté à la résolution des problèmes d'hétérogénéité sémantique.

Les médiateurs

Le médiateur fournit un accès transparent aux différentes sources de données. Les architectures peuvent éventuellement être composées de plusieurs médiateurs regroupés par domaine d'application.

Les agents artificiels

Les agents artificiels sont de plus en plus fréquemment usités dans les projets de systèmes d'information coopératifs. Un agent artificiel[42] est défini, par Briot & Demazeau [BRIOT & DEMAZEAU, 2001], comme « une entité logicielle ou physique à qui est attribuée une certaine mission qu'elle est capable d'accomplir de manière autonome et en coopération avec d'autres agents ». Différents points de vue sur les agents artificiels[43] correspondant à différents types d'agents peuvent être rencontrés dans la littérature. Nous présentons les quatre points de vue suivants sur les dix proposés par Briot & Demazeau [BRIOT & DEMAZEAU, 2001] :

- *Les agents logiciels.* Les premiers et les plus simples sont les démons Unix (processus informatiques autonomes capables de se réveiller à certaines heures ou en fonction de certaines conditions). Les virus informatiques en sont des versions déjà plus sophistiquées (notamment douées de la capacité de reproduction) et malfaisantes. Notons qu'il est difficile de donner une définition actuelle (consensuelle) du terme agent logiciel. Cela sous-entend souvent aussi une encapsulation en agent (« agentification ») de programmes (par exemple, objets) ou de bases de données plus traditionnels, de manière à favoriser leur coopération. En tout cas, ce terme s'oppose au terme agent physique, tel un robot [BRIOT & DEMAZEAU, 2001].

- *Les agents mobiles.* L'idée des agents mobiles est de donner la capacité à un agent logiciel de se déplacer d'une machine à une autre en fonction des données et informations à traiter ainsi localement. Les motivations sont généralement la minimisation des communications distantes (il est en général moins coûteux de migrer le code de traitement que les données à traiter, qui peuvent être beaucoup plus volumineuses) et le cas de l'informatique nomade (l'agent mobile peut continuer ses traitements sur des serveurs pendant la déconnexion de la machine cliente) [BRIOT & DEMAZEAU, 2001].

- *Les agent cognitifs ou intelligents*. Un agent cognitif est un agent de forte granularité[44] capable de résoudre certains problèmes par lui-même [FERBER, 1995]. Chaque agent dispose d'une base de connaissances comprenant l'ensemble des règles et des procédures nécessaires à la réalisation d'objectifs visés et à la gestion des interactions avec les autres agents :

[42] Il n'existe pas encore un consensus sur la définition d'un agent artificiel. En plus de la relative jeunesse du domaine, une des raisons est que diverses communautés revendiquent ce terme avec des problématiques assez différentes.

[43] Notez que ces différents points de vue possèdent plus ou moins de recouvrements entre eux. On peut voir également qu'ils cadrent plus ou moins complètement avec la définition que nous avons donnée plus haut d'un agent artificiel, car cela dépend des caractéristiques (adaptabilité, raisonnement/apprentissage, autonomie, téléonomie/rationalité, collaboration/coopération, capacité sociale, inférence, mobilité/migration, personnalité, pro-activité, réactivité) qui sont mises en avant selon chaque point de vue. Pour plus de détails sur ces caractéristiques, nous renvoyons à [FERBER, 1995] [WOOLDRIDGE & JENNINGS, 1995] [NWANA, 1996] [BRADSHAW, 1997] et [SCHUBERT & AL., 1998].

[44] La granularité d'un agent est le degré de détail des connaissances de l'agent. Elle exprime la complexité des fonctionnalités de celui-ci.

- ⇨ Pour satisfaire un objectif, ils ont des comportements de négociation[45].
- ⇨ Ils sont proches du modèle humain et agissent selon un mode Perception/Décision/Action. Chaque agent agit en fonction de sa connaissance, de ses objectifs et de la perception qu'il a de son environnement.
- ⇨ Ils disposent de possibilités de communication[46], i.e. capacités à communiquer avec les utilisateurs et les autres agents du système, voulant se rapprocher du langage humain.

- *Les agents réactifs.* Cette approche est souvent basée sur l'émergence de comportements collectifs à partir de comportements individuels relativement simples (réagissant à des stimuli, d'où leur nom d'agents réactifs) et n'ayant pas ou très peu de capacités de représentation et de raisonnement [FERBER, 1995] [BRIOT & DEMAZEAU, 2001]. Chaque agent réactif est de faible granularité, de plus bas niveau qu'un agent cognitif, n'est pas individuellement intelligent mais participe à l'intelligence du système. Il ne dispose pas d'une représentation symbolique et explicite de son environnement à partir de laquelle il peut raisonner. Il est régi par ses capacités sensori-motrices et a ainsi un comportement « réflexe » [FERBER, 1995].

L'approche consistant à résoudre un problème en utilisant un seul agent suscite de nombreuses difficultés [NWANA, 1996]. Dans la plupart des cas, plusieurs agents autonomes se regroupent et communiquent dans l'objectif de résoudre un problème donné. Ils forment ainsi un système multi-agents. Un système multi-agents (avec son acronyme SMA, et MAS pour « multi-agent system » en anglais) est défini comme « un ensemble organisé d'agents » [BRIOT & DEMAZEAU, 2001]. Cela signifie que dans un système multi-agents, il existe une ou plusieurs organisations qui structurent les règles de cohabitation et de travail collectif entre agents (définition des différents rôles, partages de ressources, dépendances entre tâches, protocoles de négociation, de résolution de conflits, etc.).

Les paradigmes agents et SMA sont utilisés pour la modélisation des SIC. L'intérêt de l'agent pour les systèmes d'information coopératifs réside dans ses capacités collaboratives, une distribution forte des tâches et des connaissances et permet de construire des architectures ouvertes et évolutives. D'après Ferber [FERBER, 1995], l'utilisation des agents assure une reconfiguration simple, flexible et dynamique de l'ensemble des composants d'un système d'information coopératif et confère à ce dernier un grand niveau de décentralisation.
L'intégration des concepts agents dans les SIC s'est cependant faite de façon très disparate selon les projets. En effet, certains projets ont encapsulé en agents (« agentification ») des entités auparavant spécifiées à l'aide d'autres paradigmes (par exemple, la paradigme objet) et ont, ainsi, principalement

[45] La négociation est un processus de communication itératif lorsque chaque agent doit défendre certains points de vue, solutions, croyances, connaissances dans l'objectif d'atteindre un accord. Un exemple simple de protocole de négociation est le réseau contractuel (Contract-net Protocol [SMITH, 1980]).

[46] La communication fait partie de la définition de l'agent. Elle est constituée d'un ensemble de méthodes et de protocoles modélisés à l'aide d'un langage de communication.

conçu des agents logiciels, des agents mobiles ou des agents réactifs (par exemple, les projets DOK et Déca) ; d'autres ont intégré la notion d'agent dès le commencement de leurs recherches et ont généralement conçu des agents de type réactif ou cognitif[47] (par exemple, le projet InfoSleuth).
Les SIC basés sur des systèmes multi-agents spécifient des types d'agents différents. Ces agents disposent également de dénominations, tâches et rôles variés dans le cadre du traitement des requêtes globales et de la résolution des problèmes d'hétérogénéité. La diversité des dénominations et des tâches allouées aux agents est illustrée par les exemples suivants :

- *Agents wrappers*, *agents de tâches* et *agents de coordination* pour le projet coopératif DOK [TARI & AL., 1996] [TARI & STOKES, 1997] :

  Les agents de coordination, après transmission d'une requête globale, sont chargés de décomposer cette requête en sous-requêtes et de les transmettre aux agents de tâches dont l'objectif est l'optimisation des requêtes locales, la gestion des transactions et l'accès à l'information souhaitée par le biais des agents wrappers. Les agents wrappers disposent de fonctions de traduction entre langages de requête, de fonctions de ré-ingénierie permettant la modélisation de la sémantique des sources d'information locales dans le modèle de représentation commun et de fonctions de sécurité et de réplication des données.

- *Agents facilitateurs* et *agents wrappers* pour le projet InfoMaster [GENESERETH & AL., 1997] :

  Le composant principal d'InfoMaster est un agent facilitateur qui détermine dynamiquement la manière la plus efficace de répondre à une requête globale soumise par un utilisateur, i.e. en n'utilisant que les sources nécessaires et en résolvant les problèmes d'hétérogénéité entre ces sources. Des agents wrappers permettent l'accès aux bases de données ou sources web.

- *Agents d'ontologie*, *agents de courtage*, *agents utilisateurs*, *agents ressources*, *agents moniteurs*, *agents d'exécution de tâches*, *agents d'analyse de données* pour le projet Infosleuth [BAYARDO & AL., 1997] [FOWLER & AL., 1999] :

  Les agents d'ontologie gèrent les ontologies de domaine, i.e. la création, la mise à jour et l'interrogation de chaque ontologie. Les agents de courtage répertorient les services et la localisation de chaque agent du système et ont pour rôle d'assurer la mise en relation des agents fournisseurs et des agents clients, lesquels varient selon le processus de traitement des requêtes.

  L'agent utilisateur aide l'utilisateur à construire sa requête globale, puis récupère et affiche les résultats obtenus. Les agents d'exécution de tâches gèrent les requêtes exprimées sur l'ontologie (requêtes globales) en les décomposant et les transmettant aux agents ressources appropriés. Ces derniers identifient les informations d'une source de données par rapport à une ontologie,

[47] Remarquons que certaines propositions considèrent intégrer des agents cognitifs, alors que dans les faits, ces agents se révèlent être bien moins "intelligents".

traduisent la requête globale dans le format local de la source et retournent les résultats dans le modèle de l'ontologie après exécution. Un agent fournisseur ne pouvant communiquer qu'avec un agent client que s'il connaît son emplacement, cette information lui est fournie par l'agent de courtage. Les agents moniteurs sont utilisés pour guider les interactions entre les autres agents et pour enregistrer les traces des plans d'exécution des requêtes. Enfin, les agents d'analyse des données sont des agents ressources spécialisés dans les méthodes d'analyse et de fouille de données.

- *Agents wrappers*, *agents de coopération*, *agent d'ontologie* et *agent d'interface* pour le projet DECA [BENSLIMANE & AL., 1998] [BENSLIMANE & AL., 2000] :

  Les agents de coopération assurent le rôle de médiateur et sont chargés de décomposer la requête globale soumise en sous-requêtes. L'agent d'ontologie gère une ontologie commune composée de concepts et de relations entre les concepts du domaine ainsi que d'informations sur la localisation des agents de coopération qui ont agréé tout ou partie de ces concepts. Un agent d'interface constitue l'interface entre un utilisateur et un agent de coopération gestionnaire chargé du traitement de la requête globale. Cet agent de coopération sollicite l'agent d'ontologie pour connaître les agents de coopération susceptibles d'intervenir dans le traitement de cette requête. Chaque agent de coopération traduit la requête dans son propre vocabulaire et détecte s'il peut participer à son traitement. Si tel est le cas, cette requête sera transmise dans le langage de requête commun (OQL) aux agents wrappers chargés de la traduction de la requête dans le langage d'interrogation local et de la demande d'exécution. Les résultats envoyés par les différents agents sont rassemblés par l'agent de coopération gestionnaire de la requête et transmis à l'agent d'interface pour affichage.

- *Agents facilitateurs*, *agents médiateurs*, *agent utilisateur* et *agents wrappers* pour le projet Kraft [GRAY & AL., 1997] [PREECE & AL., 1999] :

  Les utilisateurs se connectent à un agent utilisateur via un agent wrapper pour accéder aux services des agents facilitateurs et médiateurs. L'agent facilitateur répertorie les propriétés et services des autres agents sous la forme d'une base de connaissances. Il permet également d'identifier les sources d'information, via les agents wrappers, ou d'autres agents facilitateurs capables d'intervenir dans le traitement de la requête globale. Cette requête est définie à l'aide de l'agent utilisateur. L'agent médiateur est utilisé pour adapter et décomposer les requêtes, puis filtrer et fusionner les réponses obtenues des différents agents wrappers. Les agents wrappers sont chargés de la traduction des langages d'interrogation des données et de demander l'exécution de chaque sous-requête sur la source d'information locale. Les résultats sont ensuite transmis à l'agent utilisateur pour affichage.

h) *Tableau comparatif des approches de coopération et des SIC*

Le tableau suivant (cf. Tableau 1) présente la comparaison des approches de coopération et des SIC inhérents sur les points présentés dans les sections précédentes.

	Approche fédérée		Approche à base de médiation	
Période	Années 80/90		Années 90/2000	
Approches	**Fortement couplée**	**Faiblement couplée**	**Médiation de schémas**	**Médiation de contextes**
Modèle canonique	Logique, relationnel ou orienté objet (ODMG)	Pas de modèle canonique ou modèle logique, relationnel ou orienté objet (ODMG, ODMG+,…)	Logique, relationnel, orienté objet (ODMG, ODMG+,…), XML, propriétaire ou pas de modèle canonique	Logique, relationnel, orienté objet (ODMG, ODMG+,…), XML, propriétaire ou pas de modèle canonique
Langage d'interrogation	Logique ou déclaratif (SQL, OQL)	Logique (Prolog), programmation (C++,…) et/ou déclaratif (SQL, SQL étendu, OQL).	Pas de langage d'interrogation ou langage logique et/ou déclaratif (SQL, OQL, geoOQL).	Pas de langage d'interrogation ou langage logique et/ou déclaratif (SQL, OQL).
Représentation du domaine d'application	Schéma	Schéma, vue sur un schéma.	Schéma, interface objet, annuaire ou ontologie (vocabulaire de référence)	Ontologie mono ou multi-domaines (vocabulaire de référence, ontologie terminologique, ontologie logique), hiérarchisée ou non, universelle ou non.
Transparence	Localisation : *** Modélisation : *** Interrogation : **	Localisation : Nulle (multibase) ou ** Modélisation : * Interrogation : * à **	Localisation : *** Modélisation : *** Interrogation : **	Localisation : **** Modélisation : **** Interrogation : ** à **** (variable)

Evolutivité	* (Intégration statique des informations)	* (intégration statique : évolutivité variable en fonction de la présence ou non d'outils d'aide à la gestion des modifications) ou *** (intégration dynamique)	* à *** (intégration principalement statique mais l'évolutivité est variable en fonction de la présence ou non d'outils d'aide à la gestion des modifications)	**** (intégration dynamique)
Scalabilité	*	**	* ou ** (variable en fonction de la disponibilité ou non d'outils d'aide à la mise à jour des médiateurs)	*** ou **** (**** si ontologies multiples)
Hétérogénéité prise en compte	Structurelle (modèles de données et langages de requête)	Structurelle (modèles de données et langages de requête)	Structurelle et sémantique (principalement, nommage)	Structurelle et sémantique

Type de proposition / Ex. de SIC / (modèle d'architecture)	**Recherche** : Mermaid (3-tiers) **Applications opérationnelles** : Mind (obj.)	**Recherche** : CIS-MDBMS (obj.) **Prototypes** : MIRO-Web (3-tiers), Information Manifold (3-tiers), MRDSM (3-tiers),… **Applications opérationnelles :** Garlic (wrap.), InfoMaster (agt., wrap.)	**Recherche** : DISCO, DOK (agt.), TSIMMIS (med., wrap.) **Prototypes :** DIOM (3-tiers), ACSIS (Agt.). **Applications opérationnelles :** OASIS (obj.), HERMES (3-tiers), YAT (wrap.)	**Recherche** : Scope, DILEMMA (med., wrap.) **Prototypes** : KRAFT (agt.), OBSERVER, WEBFINDIT (obj.), COIN (wrap., med.), Déca (agt.), Picsel (med., wrap.),… **Applications opérationnelles** : Infosleuth (agt.), OntoBroker (wrap., web).

Obj. : Architecture basée sur des objets distribuées / Agt : Architecture agent / Wrap. : Wrapper / Med. : Médiateur

* : capacité faible / ** : capacité moyenne / *** : capacité élevée / **** : capacité très élevée

Tableau 1 Comparaison des différentes approches de coopération

Nous pouvons constater que de très nombreux SIC ont été conçus voire implémentés au cours de ces dernières années. Ces SIC sont issus d'approches de coopération dissemblables. Cependant, même à l'intérieur de ces approches, les SIC diffèrent également sur de très nombreux points (hétérogénéité des concepts, des architectures sous-jacentes, des technologies utilisées et des types de conflits résolus, variabilité[48] du respect des propriétés des SIC,…). En outre, leur processus de conception ou de développement varie également. Ainsi, comme tout système d'information, les systèmes d'information coopératifs nécessitent un processus d'ingénierie. Dans la section suivante, nous définissons le concept d'ingénierie des SIC et analysons ses particularités.

3.2. L'ingénierie des systèmes d'information coopératifs

Conrad & al. [CONRAD & AL., 2002] proposent la définition suivante de la notion d'*ingénierie des systèmes d'information coopératifs* (de type sources d'information) : « *un domaine de recherche qui à trait au développement systématique de solutions interopérables pour les systèmes hétérogènes et autonomes comprenant à la fois des bases de données mais aussi d'autres types de sources d'information (...) provenant de domaines d'application variés* ». L'ingénierie des SIC vise ainsi à construire des solutions réutilisables dédiées à la coopération des sources d'information. Tout comme l'ingénierie des systèmes d'information, l'ingénierie des SIC utilisent des concepts ou techniques provenant du génie logiciel, mais aussi des techniques issues de l'intelligence artificielle distribuée telles que les ontologies, les agents ou systèmes multi-agents,…

Boughzala [BOUGHZALA, 2002] donne une définition de l'ingénierie de la coopération, dans le contexte du travail collaboratif, mais pouvant également s'appliquer à la coopération de sources d'information : « *l'étude des concepts, des méthodes et des techniques permettant d'acquérir, de modéliser et de formaliser la connaissance afin de la mobiliser* » lors du développement de SIC.

Ces deux définitions complémentaires de l'ingénierie des SIC mettent l'accent sur le développement de solutions d'interopérabilité et sur la nécessité de modéliser et de formaliser les connaissances portant sur ce développement.

Ainsi, comme nous venons de le voir (§ 3.1.3.3), de nombreuses solutions très hétérogènes ont été proposées, portant à la fois sur la définition de concepts et d'architectures réutilisables. Ces architectures conceptuelles et logicielles permettent la construction de processus coopératifs entre sources d'information et la résolution des conflits inhérents. Cependant, les SIC existants (et leurs

[48] Caractère de ce qui est variable, dissemblance [LAROUSSE, 2004].

concepts et architectures sous-jacents) souffrent de trois défauts rendant fastidieuse leur réutilisation :

- ils sont très complexes et donc difficilement réutilisables,
- d'après Fernandez & Zhao [FERNANDEZ & ZHAO, 2000], leur ingénierie ne s'appuie pas, pour la plupart des cas, sur les techniques de réutilisation évoquées précédemment et qui ont fait leur preuve : framework[49], pattern[50],…
- leur documentation est souvent éclatée entre de nombreux articles de recherche et/ou thèses. Ainsi, par exemple dans le cadre du projet de coopération ACSIS, la définition et la spécification de l'architecture et des concepts sous-jacents ont fait l'objet de plusieurs dizaines d'articles, de trois thèses et de deux prototypes distincts. C'est en fait en partie ces lacunes relatives à la documentation qui laissent penser que les plates-formes de coopération actuelles ne sont utilisables que par leurs concepteurs et qui entraînent ainsi le développement de nouvelles plates-formes *ad hoc*.

Face à ce constat, les recherches du domaine pointent deux directions à investiguer [FERNANDEZ & ZHAO, 2000] [HASSELBRING & AL., 2000] [WYSS & AL., 2003] :

- augmenter la réutilisabilité des SIC existants avec notamment des techniques de réutilisation ayant fait leurs preuves dans le monde objet : pattern, framework, composant logiciel,…
- promouvoir les recherches sur l'aide à l'ingénierie de nouveaux systèmes d'information coopératifs (en particulier sur les aspects conceptuels et architecturaux liés à cette ingénierie).

Ces deux directions de recherche pointent ainsi la nécessité d'utiliser des techniques éprouvées permettant d'une part de favoriser la réutilisation des SIC existants et d'autre part de faciliter l'ingénierie des systèmes d'information coopératifs. Se pose alors la question du choix du ou des avoirs logiciels adaptés. A partir des types d'avoir logiciel précédemment présentés (§ 2.3.4), nous effectuons et explicitons notre choix dans la section suivante (§ 4).

[49] Certaines architectures coopératives proposées sont mentionnées sous le nom de framework. Cependant, ce qualificatif est très fréquemment galvaudé et les "frameworks" coopératifs se résument dans la plupart des cas à des bibliothèques de classes.

[50] Pour les patterns dédiés aux systèmes d'information coopératifs, Cf. chapitre 2

4. Vers le choix d'un type d'avoir logiciel adapté à l'ingénierie des systèmes d'information coopératifs : les patterns

La section suivante est dédiée au choix du type d'avoir logiciel réutilisable adapté à l'ingénierie des systèmes d'information coopératifs (§ 4.1). La section 4.2 présente plus amplement le type d'avoir réutilisable lors de l'ingénierie des SIC que nous avons sélectionné et analyse notamment ses avantages.

4.1. Le choix du type d'avoir logiciel adapté à l'ingénierie des SIC

Le choix de l'avoir logiciel réutilisable adapté à l'ingénierie des systèmes d'information coopératifs est fonction de plusieurs critères. Nous nous basons sur les critères exposés dans [RIDEAU, 1999] et [RIEU, 1999] pour effectuer ce choix, à savoir la granularité (taille des briques réutilisables mesurée en nombre de classes ou d'objets), la portée (étape(s) du cycle de développement à laquelle l'avoir s'adresse), la couverture (avoir dédié à un domaine d'application ou indépendant d'un domaine) et la technique d'implémentation sous-jacente (dépendance ou non entre l'avoir logiciel et son cadre technique d'implémentation).

Granularité :

Les patterns et les modèles de domaine, ayant pour objectif de fournir une solution à un problème type restreint, proposent des diagrammes constitués de quelques objets ou classes. Par exemple, les patterns d'analyse de Fowler et les patterns de conception de Gamma n'exposent que des diagrammes de classes composés de deux à six classes. La granularité des composants logiciels est variable, allant de quelques objets ou classes à quelques dizaines. Au contraire, les frameworks sont des macro-architectures et une majorité est composée de plusieurs dizaines voire centaines de classes. En matière de réutilisation, les gains attendus sont d'autant plus importants que les avoirs logiciels ont une granularité forte, mais, comme nous venons de le présenter dans le cas des SIC, celle-ci pénalise aussi grandement la facilité d'utilisation. Compte tenu de leur taille, les frameworks peuvent s'avérer trop rigides et très complexes à utiliser (cf. section 2.3.4.4.a)). Ainsi, les patterns, modèles de domaine et composants logiciels, proposant des unités de raisonnement très modulaires, présentent un avantage indéniable, par rapport au framework.

Portée :

Les avoirs logiciels peuvent être utilisés lors des phases de capture des besoins (expression des besoins de l'utilisateur), d'analyse (pour décrire le domaine d'application), de conception (pour bâtir le logiciel à un haut niveau d'abstraction) et d'implantation (pour traduire la conception dans un

langage informatique). L'utilisation des différents types d'avoir logiciel dans le cycle de développement du logiciel est présentée en Figure 5.

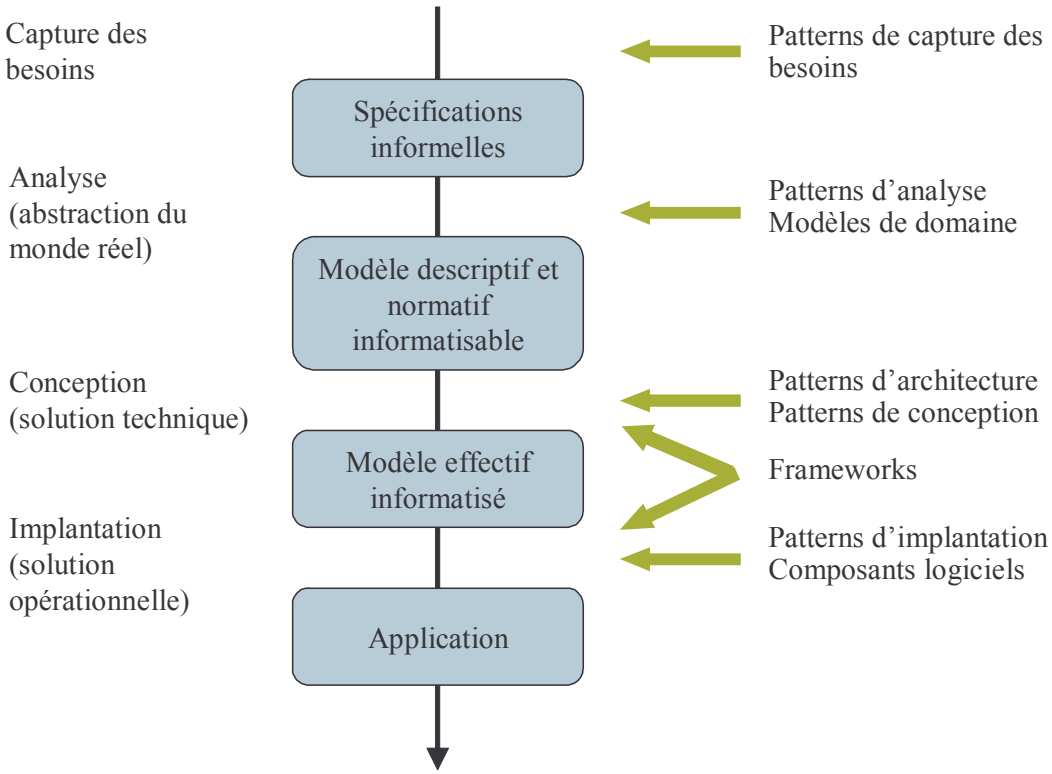


Figure 5 : Utilisation des avoirs logiciels dans le cycle de développement du logiciel

Les modèles de domaine sont réutilisables lors de la phase d'analyse uniquement. Les composants logiciels ne sont réutilisables que lors de la phase d'implantation. Les frameworks sont également réutilisables lors de cette phase, mais leur réutilisation offre une plus-value plus importante, puisqu'elle implique aussi une réutilisation de conception. Enfin, seuls les patterns sont réutilisables sur tout le cycle de développement du logiciel. Dans le cadre de l'ingénierie des systèmes d'information coopératifs, l'accent devra être mis sur les avoirs utilisables lors des phases d'analyse, de conception et d'implémentation.

Couverture :

Les modèles de domaines, ainsi que les composants logiciels généralement orientés sur un métier, sont liés à un domaine d'application. Les patterns d'analyse, de conception et d'architecture sont suffisamment abstraits pour être utilisés quelque soit le domaine, mais peuvent être également dédiés à un domaine d'application spécifique.

Technique d'implémentation sous-jacente

En dehors des patterns d'implantation liés à un langage de programmation, les patterns sont indépendants des techniques d'implantation, de même que les modèles de domaine. Leur potentiel à être réutilisés est donc plus large. Les frameworks et les composants logiciels sont par contre dépendants d'un langage et/ou d'une plate-forme logicielle.

Compte tenu des critères de comparaison envisagés, la réutilisation de patterns pour l'ingénierie des systèmes d'information coopératifs nous semble la forme de réutilisation la plus adaptée. Les patterns peuvent être utilisés pour toutes les étapes du cycle de développement d'un SIC (analyse, conception, implantation), sont de granularité adaptée et peuvent être appliqués sur un domaine d'application spécifique. Par ailleurs, l'intégration dans un même pattern d'un problème type et d'une solution constitue une aide à la recherche et à l'intégration. Enfin, les patterns ont une double fonction correspondant aux deux aspects de la réutilisation :

- ils constituent des avoirs logiciels réutilisables puisqu'ils capitalisent un savoir-faire utilisable à de nombreuses reprises lors du développement logiciel (domaine du « design by reuse »),
- ils permettent de guider un concepteur d'applications en favorisant des constructions modulaires, respectant le domaine d'application, favorisant la documentation, améliorant la fiabilité… (domaine du « design for reuse »).

Dans la section suivante (§ 4.2), nous présentons plus amplement le concept de pattern, donnons des exemples et analysons ses avantages pour le domaine de l'ingénierie des SIC.

4.2. Les patterns logiciels

Avant d'appliquer, dans le chapitre 2, les patterns au domaine de la coopération des systèmes d'information coopératifs, nous allons développer et étudier plus en détails les patterns, leurs avantages et comment ils peuvent être utiles aux concepteurs de SIC. Nous présentons tout d'abord une classification détaillée des différents types de patterns.

4.2.1. Classification des patterns

De nombreux travaux ont proposé des critères pour classifier les patterns logiciels : classification en fonction de l'utilisation du pattern lors des phases du cycle de développement du logiciel, pattern génératif vs. pattern non génératif, pattern lié à un domaine d'application vs. pattern indépendant

d'un domaine, patterns produits vs. patterns de processus[51], classification en fonction du degré de formalisation de la description (langage naturel ou modèles), classification selon l'utilisation du pattern dans une partie du SI,… Nous proposons de les classifier en fonction des critères les plus couramment usités : leur application en fonction des phases du cycle de développement du logiciel (§ 4.2.1.1) et leur caractère génératif (§ 4.2.1.2).

4.2.1.1. Classification des patterns en fonction de leur utilisation dans le cycle de développement du logiciel

Comme nous l'avons succinctement explicité précédemment (§ 2.3.4.2), les patterns logiciels sont notamment classés en fonction de leur application lors des phases du développement logiciel : patterns de capture des besoins [BRAGA & AL., 2000], patterns d'analyse [COAD, 1996] [FOWLER, 1997]) (phase d'analyse), patterns de conception [GAMMA & AL., 1995], idiomes (étape d'implantation) [COPLIEN, 1992] et les patterns de processus [AMBLER, 1998]. Remarquons que les patterns de processus d'Ambler sont les seuls patterns couvrant tout le cycle de développement du logiciel. Le degré de description des patterns de processus varie cependant en fonction de ces différentes étapes.

Comme illustré en Figure 6, nous adoptons cette taxonomie pour proposer une classification des patterns logiciels disponibles dans la littérature. Ainsi, comme d'autres auteurs, nous pensons que ces catégories peuvent classifier la grande variété de patterns logiciels disponibles pour chaque domaine d'application.

[51] Un pattern produit décrit la structure d'une solution. Un pattern de processus exprime la façon dont un pattern produit est engendré.

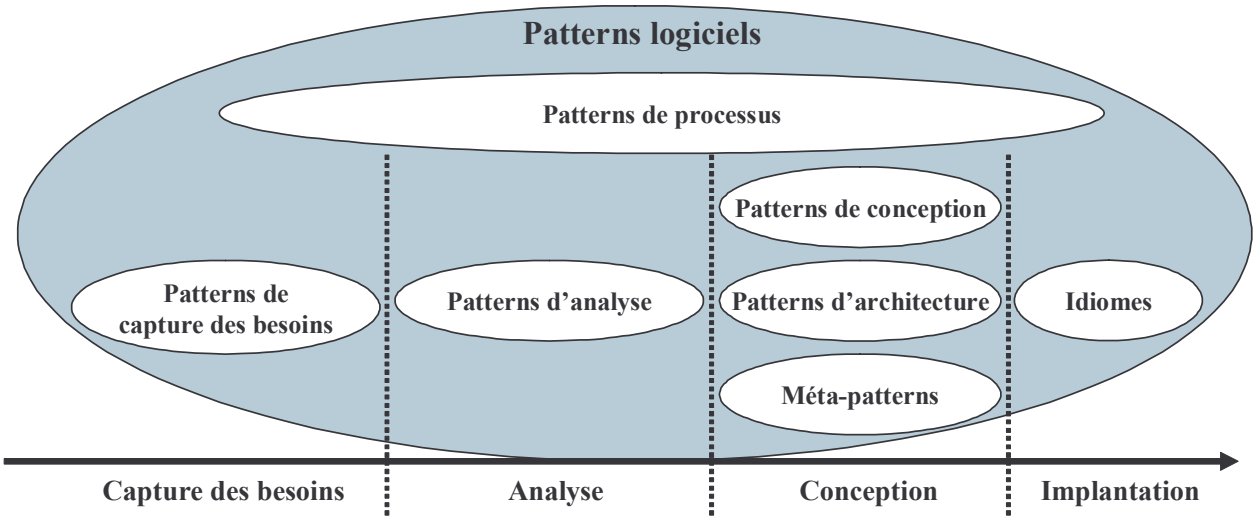


Figure 6 : Proposition d'une classification des patterns logiciels

Les sous-sections suivantes présentent de façon plus détaillée les patterns de capture des besoins, les patterns d'analyse, les patterns de conception comprenant également les patterns d'architecture et les méta-patterns, les idiomes et les patterns de processus.

a) Patterns de capture des besoins

Les *patterns de capture des besoins* documentent les besoins des utilisateurs et le comportement générique du système à un haut niveau d'abstraction. Ils se focalisent sur le comportement fonctionnel d'un système sans prendre en compte sa structure. Ces patterns permettent également, dans certains cas, de décrire des actions génériques que les développeurs peuvent accomplir pour améliorer les besoins logiciels non fonctionnels tels que la performance, la réutilisabilité, la fiabilité ou la maintenance. Comme présentés dans [RISING, 2000], les patterns dédiés aux systèmes de télécommunication tolérants aux pannes ("fault-tolerant telecommunication system patterns") [ADAMS & AL., 1995] appartiennent, par exemple, à cette classification et adressent les problèmes de facteurs humains et de fiabilité dans les logiciels de télécommunication.

Remarquons que certains auteurs considèrent que les patterns de capture des besoins font partie des patterns d'analyse présentés en section suivante.

b) Patterns d'analyse

Les patterns d'analyse ont été proposés pour compléter les patterns de conception déjà existants. Ces patterns s'appliquent durant la phase d'analyse du cycle de développement du logiciel.

Le but de la phase d'analyse est d'obtenir une description approfondie du domaine du problème, de manière à ce que les applications s'appuyant sur ce domaine puissent être bien comprises. Pour cela, le concepteur du logiciel doit créer un modèle conceptuel qui représente sa perception du monde réel, les patterns d'analyse facilitant ce travail de modélisation [FOWLER, 1997]. En nous aidant à

structurer notre vision d'un domaine, ils permettent ainsi d'obtenir un modèle conceptuel de qualité, à savoir clair, évolutif et plus facilement compréhensible. Deux types de patterns d'analyse peuvent être relevés dans la littérature : les *patterns d'analyse génériques* [COAD, 1992] et les *patterns d'analyse de domaine* [FOWLER, 1997].

- les patterns d'analyse génériques

Des *patterns d'analyse génériques* ont été proposés par Coad [COAD, 1992]. Ces patterns décrivent des structures génériques applicables et réutilisables quel que soit le domaine d'application.

- les patterns d'analyse de domaine ou patterns de domaine

Selon Riehle & Züllinghoven [RIEHLE & ZÜLLINGHOVEN, 1996], les patterns d'analyse doivent être décrits en utilisant les termes et concepts du domaine d'application. Ainsi, ils suggèrent que la portée d'un pattern d'analyse doit être restreinte à un domaine d'application bien défini, comme, par exemple, les logiciels temps réels.

Fowler fait la même critique et propose plusieurs *patterns d'analyse de domaine* ou *patterns de domaine* spécialisés sur un domaine d'application unique. Il considère ainsi que les patterns d'analyse doivent représenter la manière dont les gens abordent leur métier, indépendamment de tout système informatique [FOWLER, 1997]. La communication est ainsi facilitée entre les experts du domaine et les développeurs.

Nous présentons ci-après un exemple de pattern de domaine proposé par Fowler [FOWLER, 1997] et traduit par [CHIKHI & AL., 1999] (Cf. Tableau 2).

Nom : Contrat - commercial
Contexte : Transaction financière pour l'achat d'instruments entre deux parties
Problème : Un contrat est une transaction entre une partie primaire (la partie qui initie le contrat) et l'autre partie (la contrepartie). L'instrument est acheté ou vendu pour un montant et un prix donnés. Dans le jargon commercial, dans le cas d'une transaction de vente (resp. Achat), le contrat est considéré comme court (short) (resp. Long (long)).
Exemple : MegaBank vend 2 millions US Dollars (USD) pour 1 million British Pounds (GBP) pour British Railways. C'est un contrat long où la contrepartie est British Railways, le montant est d'1 million, le prix est de 2 et l'instrument est GBP/USD. De même, le contrat peut être court où le montant est de 2 millions, le prix de 0,5 et l'instrument est USD/GBP.
Solution : Plusieurs solutions sont possibles suivant l'information qu'on veut faire apparaître (la partie primaire et la contrepartie, ou le vendeur et l'acheteur). La solution finale se traduit par la représentation de toutes les associations possibles, en précisant celles qui sont dérivées.

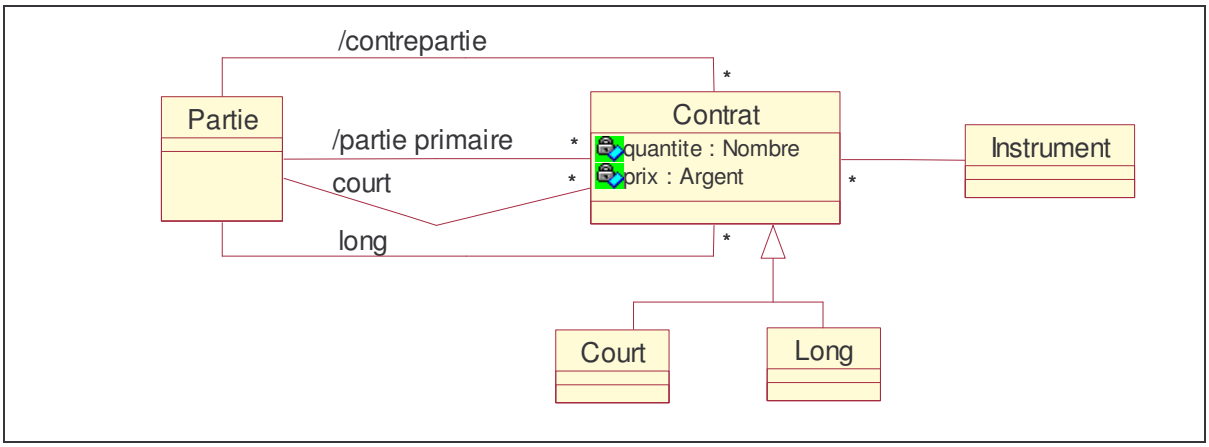


Tableau 2 : Exemple de pattern d'analyse de domaine de Fowler [FOWLER, 1997]

Notons que Fowler [FOWLER, 1997] propose un autre type de patterns, les *patterns de support*. Ces patterns traitent des problèmes qui interviennent quand un système informatique doit être construit autour des patterns d'analyse de domaine. Ils décrivent comment utiliser les patterns d'analyse de domaine et comment les appliquer pour les rendre concrets. Les patterns de support proposés par Fowler sont tout d'abord des *patterns d'architecture*, philosophiquement proches de ceux de Buschmann, dont il donne d'ailleurs une définition comparable [BUSCHMANN & AL., 1996]. Un pattern de support de type architectural décrit par exemple les différents niveaux requis d'une architecture client/serveur trois-tiers.

Fowler propose ensuite des *patterns de conception* comparables à ceux de Gamma & al. (par exemple, le pattern *Façade* également présenté dans [GAMMA & AL., 1995]). Enfin, Fowler présente des *patterns de transformation* (*patterns for type model design templates)* qui décrivent des principes permettant de transformer des modèles conceptuels en logiciel. Ils peuvent être utilisés pour construire des "templates" de conception qui décrivent comment transformer un modèle de spécification implicite en un modèle de spécification explicite et une implémentation. Ces "templates" de conception varient selon la méthode de conception utilisée, l'environnement d'implantation, les standards et les besoins de performance du système final. Ils peuvent être automatisés par un générateur de code. D'après Fowler, ces templates de conception doivent définir l'interface des composants logiciels et fournir un certain nombre d'implantations de ces composants.

c) *Patterns de conception, patterns d'architecture et méta-patterns*

La plupart des patterns disponibles dans la littérature sont dédiés aux phases d'analyse et de conception du cycle de développement du logiciel. Les méta-patterns et les patterns d'architecture sont également associés à la phase de conception (Cf. Figure 6).

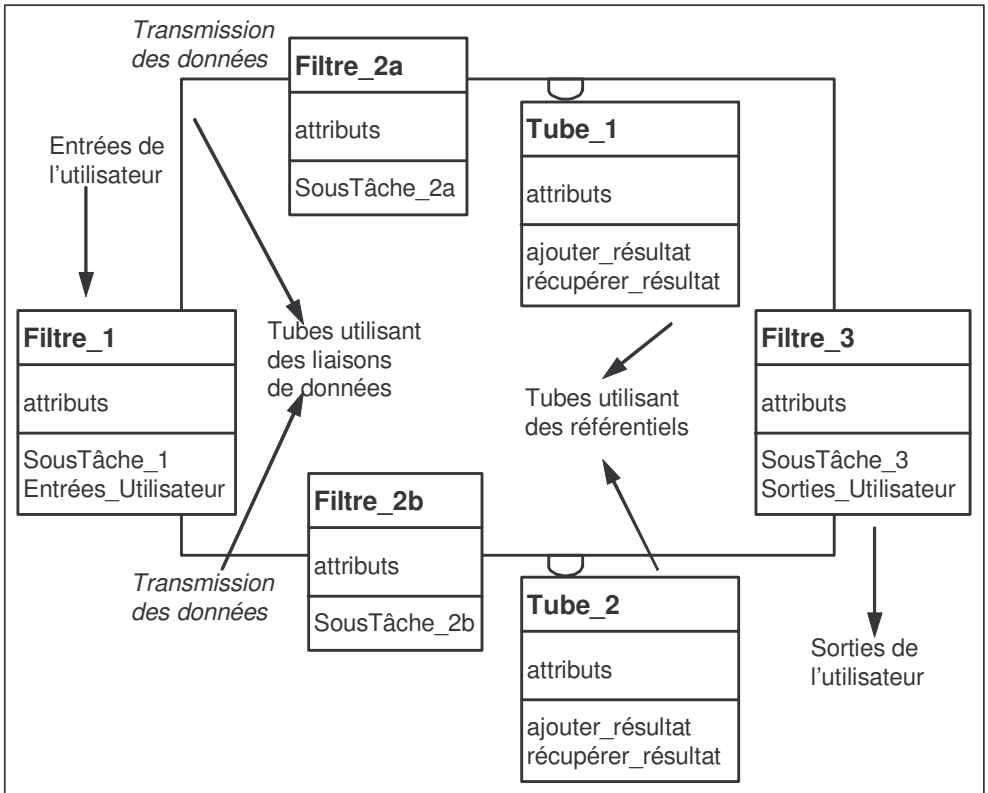
La conception objet a pour objectif de proposer une solution technique, à partir de l'analyse préalable. Elle débute par la détermination de l'architecture logicielle, tâche permettant l'élaboration

des structures statiques et dynamiques qui serviront de charpente pour l'ensemble du développement. Les patterns architecturaux aident à créer de telles architectures. Une fois l'architecture de l'application établie, la solution technique doit être définie plus en détail ; les patterns de conception favorisent l'élaboration de cette solution.

- Les patterns d'architecture

Un *pattern d'architecture* détermine la structure de base d'une application et peut être vu comme un "template" pour des architectures logicielles concrètes. Il fournit un ensemble de sous-systèmes prédéfinis et spécifie leurs responsabilités ainsi que les collaborations et communications entre ceux-ci [BUSCHMANN & AL., 1996]. Ces patterns décrivent ainsi les mécanismes globaux d'une application.

Un exemple de pattern d'architecture est le pattern *Pipes and Filters* présenté en figure suivante (Cf. Figure 7).



Applicabilité: L'architecture *Pipes and Filters* (tubes et filtres) est utile pour les systèmes logiciels dans lesquels différentes sous-tâches peuvent être identifiées et liées à des composants distincts qui : 1) sont complets, 2) sont indépendants des autres composants du système, 3) coopèrent uniquement avec un autre composant en utilisant le résultat de ce composant comme entrée et 4) sont exécutables séquentiellement ou parallèlement dans un ordre déterminé.

Figure 7 : Un exemple de pattern d'architecture, le pattern « Pipes and Filters »

- Les patterns de conception

Les *patterns de conception* sont dédiés au raffinement des composants du système ou de leurs relations. Gamma & al. [GAMMA & AL., 1995] définissent les patterns comme « *des descriptions d'objets et de classes communicants qui sont adaptés pour répondre à un problème général de conception dans un contexte particulier* ». Ils sont décrits au moyen des éléments de construction du logiciel (objets, classes, relations d'héritage, d'agrégation,…) et clarifient la structure mais aussi et surtout les relations dynamiques entre ces éléments [RIEHLE & ZÜLLINGHOVEN, 1996]. Les patterns de conception sont indépendants des langages de programmation et des plates-formes d'implantation logicielle.

Par exemple, le pattern de conception *Composite* ([GAMMA & AL., 1995]) comporte onze sections réparties sur dix pages. Les trois premières sections du pattern, *Intent*, *Motivation* et *Applicability*, présentent le problème : composer des classes d'objets sous la forme d'un arbre pour traiter uniformément un objet ou une composition d'objets. Les trois sections suivantes, *Structure*, *Participants* et *Collaborations*, proposent une solution à ce problème sous la forme de diagrammes de classes (Cf. Figure 8) et d'instances, d'une liste des classes participantes – liste présentant leurs rôles –, et d'une liste des collaborations entre les classes – liste présentant leurs interactions. La section *Consequences* énonce les effets de l'application de ce pattern de conception : simplification des clients et de l'implémentation. Enfin, les sections *Implementation*, *Sample Code*, *Known Uses* et *Related Patterns* fournissent des renseignements spécifiques à l'implémentation et à l'utilisation de la solution.

Exemple : Un survol du pattern de conception Composite
Le pattern de conception *Composite* construit des structures complexes d'objets en composant récursivement sous forme d'arbres des objets de même nature. Il permet de traiter uniformément les objets et les compositions d'objets. La structure générale du pattern de conception *Composite* est présentée ci-dessous.
La classe abstraite (ou interface) *Composant* définit une *operation()* qui peut être appliquée indifféremment sur un objet de type *Feuille* ou sur une composition d'objets de type *Composant*, *Composite*. Les instances de la classe *Composite* se chargent alors d'appliquer l'*operation()* sur l'ensemble de tous leurs fils.

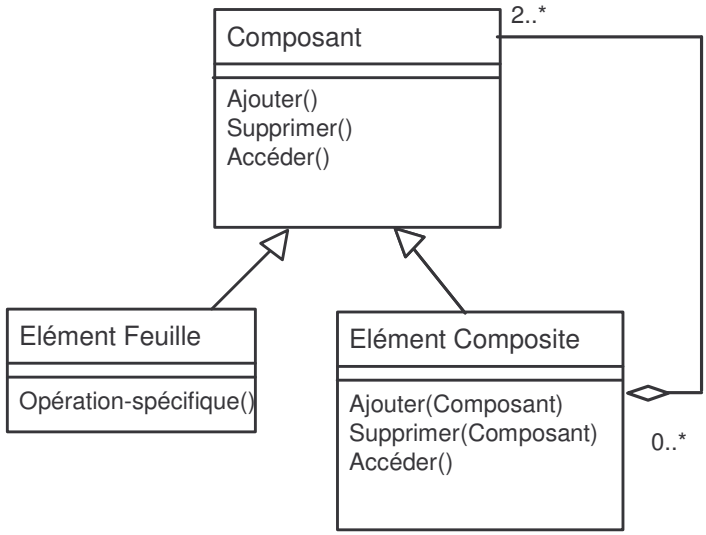


Figure 8 : Le pattern de conception "Composite" [GAMMA & AL., 1995]

Notons que les patterns de conception disponibles dans la littérature sont soit des patterns généraux, i.e. s'appliquant quel que soit le domaine d'application (patterns de Gamma & al. [GAMMA & AL., 1995],…), soit des patterns dédiés à un domaine d'application particulier (patterns de conception de workflow [VAN DER ALST & AL., 2003], patterns de conception dédiés à l'intégration d'applications d'entreprise [HOHPE & WOOLF, 2004], patterns de conception dédiés à l'ingénierie des systèmes d'information produit [GZARA, 2000],…).

- Les méta-patterns

Le terme *méta-pattern* est défini par Pree [PREE, 1995], comme « *un ensemble de patterns de conception qui décrivent comment construire des frameworks indépendants d'un domaine spécifique* ». Dans ce contexte, les frameworks sont décrits comme des « *blocs de construction semi-finis et prêts à l'emploi* » (pour plus de détails sur les frameworks, cf. section 2.3.4.4).
Au niveau des étapes de conception et d'implantation, les méta-patterns fournissent les principaux concepts orientés objets pour construire un framework (y compris les *hot spots* et *frozen spots*). Les méta-patterns sont complémentaires aux concepts et modèles orientés objet qui permettent l'implémentation de frameworks avec *hot spots*, ainsi qu'aux autres patterns logiciels décrivant comment concevoir et implanter ces *hot spots*.

Les méta-patterns ne capturent pas seulement la conception d'une application existante, mais garantissent également que tout domaine peut développer des frameworks basés sur des expériences antérieures.

Les méta-patterns peuvent également être considérés comme la base structurelle des patterns de conception, du fait qu'ils capturent l'essence de patterns de conception existants. D'après Pree [PREE, 1995], les meta-patterns permettent ainsi de décrire des patterns de conception à un méta-niveau : les patterns de conception étant structurellement similaires, les méta-patterns peuvent être utilisés pour factoriser ces similarités et permettent de créer de nouveaux patterns de conception.

d) Patterns d'implantation

Les *patterns d'implantation*, ou idiomes de programmation, sont liés à la phase d'implantation du cycle de développement du logiciel. Ce type de pattern couvre les particularités et astuces propres à un langage de programmation [COPLIEN, 1992]. Les idiomes permettent d'exploiter aux mieux le potentiel offert par un langage de programmation spécifique et ne sont généralement pas transposables d'un langage de programmation à un autre. Ils permettent souvent d'obtenir un code plus condensé ou lisible, ce qui facilite l'écriture ou la maintenance de l'application.

De nombreux idiomes ont été proposés : idiomes C++ [COPLIEN, 1992], Java [LANGR, 1999], Smalltalk [BECK, 1997],… Par exemple[52], la copie d'une chaîne de caractères en C++ peut se réaliser de la manière suivante [COPLIEN, 1992] :

```
While (*dest++=*source++) ;
```

Cette solution proposée par Coplien permet de simplifier l'ensemble des lignes de code suivantes :

```
While (dest != null)
{
      *dest = *source ;
      dest++ ;
      source++ ;
}
```

Par rapport au problème simple de recopie d'une chaîne de caractères (deuxième morceau de code), la solution proposée par Coplien est beaucoup plus condensée (mais est-elle réellement plus lisible ?!).

e) Patterns de processus

Ambler [AMBLER, 1998] considère que non seulement des mêmes types de problèmes peuvent survenir fréquemment et indépendamment des domaines d'application, mais également que les

[52] L'exemple présenté n'est qu'un exemple partiel d'idiome C++ proposé par Coplien. Le pattern complet devrait, en plus de la solution, expliquer le problème et le contexte.

stratégies pour les résoudre peuvent être récurrentes. Ces stratégies peuvent être décrites par les *patterns de processus* ou *process patterns*. Un pattern de processus est défini comme « *une collection de techniques, d'actions et/ou de tâches (activités) générales pour développer des logiciels orientés-objet* » [AMBLER, 1998]. Un pattern de processus est un moyen pour les organisations de donner forme à un processus logiciel ayant fait ses preuves. En fait, Ambler décrit un pattern de processus comme conceptuellement similaire à un pattern de conception ou un pattern d'analyse, dans le sens que chacun décrit une solution à un problème sans donner de détails sur l'implémentation.

Nous présentons dans la section suivante une autre classification des patterns logiciels, fonction de leur caractère génératif ou non.

4.2.1.2. Patterns génératifs vs patterns non génératifs

Tous les patterns logiciels peuvent être classés selon leurs caractéristiques génératives ou non génératives.

Les patterns qui sont observés dans un système existant sont considérés comme non génératifs (par exemple, les patterns de conception de Gamma & al. [GAMMA & AL., 1995]). Ces patterns sont descriptifs et passifs du fait qu'ils décrivent des phénomènes récurrents sans décrire nécessairement la façon de reproduire ces phénomènes. Ils se focalisent sur les solutions sans préciser quand les utiliser, et n'explicitent pas les raisons qui ont conduit à les spécifier.

Les patterns génératifs reposent sur une double acception : ils sont à la fois prescriptifs et actifs. Un pattern génératif exprime la (ou les) règle(s) expliquant comment créer la configuration résolvant le problème, i.e. comment parvenir à la solution, et ne se limite pas à décrire cette dernière. Ainsi, les patterns génératifs ne montrent pas seulement les caractéristiques d'un bon système, mais enseignent aussi comment construire celui-ci : « chaque pattern génératif est une règle qui décrit ce que vous devez faire pour générer l'entité qu'il définit » [ALEXANDER, 1979].

Ils sont conçus pour donner forme à une architecture et peuvent générer des systèmes ou des parties de systèmes. Des exemples de patterns génératifs sont les patterns d'analyse de Fowler [FOWLER, 1997] ou les patterns de conception génératifs de Beck & Johnson [BECK & JOHNSON, 1994].

Les patterns, quels que soient leur type et leur classification, ne sont pas isolés mais reliés les uns aux autres pour former une collection. Nous étudions les différentes collections possibles de patterns en section suivante.

4.2.2. Collection de patterns

Trois types de collections de patterns peuvent être distingués dans la littérature : le catalogue de patterns, le système de patterns et le langage de patterns. Ces trois types de collections se différencient par leur structuration et les interactions internes entre les patterns.

4.2.2.1. Catalogue de patterns

Un *catalogue de patterns* est une collection de patterns reliés, dans laquelle ces derniers sont regroupés dans un petit nombre de catégories variées, incluant généralement des références croisées entre les patterns [BUSCHMANN & AL., 1996]. Les catalogues visent à aider les développeurs ou concepteurs à trouver les patterns adaptés à leurs besoins spécifiques de conception, d'analyse, d'implantation ou de capture des besoins. Les patterns proposés par Gamma & al. [GAMMA & AL., 1995] sont, par exemple, proposés sous forme d'un catalogue composé de deux critères de sélection et de regroupement :

- l'essence du pattern ("purpose") reflète ce que le pattern fait,
- la portée du pattern ("scope") spécifie si le pattern s'applique principalement sur des classes ou des objets.

Rising [RISING, 2000] présente également une collection de patterns déjà publiés, organisée en un catalogue regroupé par domaines d'application.

4.2.2.2. Système de patterns

Un *système de patterns* est un ensemble cohésif de patterns reliés et ajoute une structure profonde, des interactions plus précises et une uniformité à un catalogue de patterns [BUSCHMANN & AL., 1996] [APPLETON, 2000]. Il décrit la plupart des relations entre les patterns, leurs regroupements et la façon dont ils peuvent être combinés et composés pour résoudre des problèmes plus complexes.

4.2.2.3. Langage de patterns

Un *langage de pattern*[53]*s* est une collection structurée de patterns pour transformer des besoins et des contraintes en une architecture [COPLIEN, 1998]. Un langage de patterns n'est pas un langage de programmation, mais plutôt un document dont l'objectif est d'informer et de guider le concepteur ou le développeur dans l'utilisation des patterns. Il intègre ainsi des règles et des guides qui expliquent

[53] Un langage de patterns n'est pas un langage formel.

comment et quand appliquer ces patterns pour résoudre un problème qu'un pattern, utilisé seul, ne pourrait résoudre. Ces règles et guides suggèrent l'ordre d'application des patterns du langage.

Alexander [ALEXANDER, 1979] considère qu'un langage de patterns est une collection structurée de patterns génératifs que les utilisateurs peuvent utiliser pour construire des systèmes logiciels. Cela signifie que les architectures sont générées à partir du langage de patterns utilisé dans cette architecture.

La principale différence entre un système et un langage de patterns réside dans la robustesse, la plus grande facilité de compréhension et la complétude de ce dernier. Les langages de patterns montrent dans l'idéal toutes les combinaisons possibles de patterns et leurs variations pour produire des architectures complètes. En pratique cependant, la différence entre un langage et un système de patterns est guère respectée par de nombreuses publications (par exemple, [HASSELBRING, 1999], [SCHUMACHER & ROEDIG, 2001],…).

Nous présentons dans la section suivante, la mise en œuvre - ou réutilisation - des patterns par l'utilisateur.

4.2.3. Mise en oeuvre des patterns

La mise en oeuvre d'un pattern, également qualifiée d'activité de développement logiciel à base de patterns, comprend trois activités distinctes :

(1) le choix du pattern adéquat,
(2) son adaptation au contexte d'utilisation ou sa recopie,
(3) et, éventuellement, la production du code correspondant.

La première activité passe soit par une analyse du contexte puis une tentative d'assimilation à un problème déjà référencé pour enfin suggérer le pattern le plus adéquat, soit par la mise en œuvre d'un système de navigation entre les différents patterns afin de faciliter le choix de l'utilisateur. Même si des systèmes - aux fonctionnalités plus ou moins complexes - de ce type existent déjà (par exemple [BUDINSKY ET AL., 1996], [MOTELET, 2000], [CONTE & AL., 2001B]…), nous en avons développé un spécifique. Les fonctionnalités de celui-ci sont succinctement présentées au chapitre 2 et sa description complète est effectuée en annexe A.

La deuxième activité consiste à adapter la solution du pattern au contexte d'utilisation (et à l'instancier) ou recopier celle-ci. La recopie consiste à utiliser la solution telle quelle ; elle est notamment usitée dans le cadre de solutions exprimées en langage naturel. Dans le cadre de

solutions de type modèle, l'adaptation consiste principalement à renommer, redéfinir, ajouter et/ou supprimer les propriétés statiques (attributs), dynamiques (méthodes) et relationnelles (associations) des entités spécifiées [RIEU & AL., 1999].

La troisième activité consiste à produire le code correspondant à la solution du pattern [ALBIN-AMIOT, 2003]. Elle concerne certains patterns de conception et patterns d'implantation. Plusieurs approches de génération de code à partir de patterns ont été proposées. Nous citons en particulier les travaux de Sunyé [SUNYE, 1999] pour l'implantation de patterns de conception.

Nous présentons dans la section suivante les avantages des patterns pour l'ingénierie des systèmes d'information - et donc aussi pour l'ingénierie des SIC[54] - et comment ils peuvent être utiles aux concepteurs lors du développement de logiciels.

4.2.4. Avantages des patterns pour l'ingénierie des SI

Riehle & Züllinghoven [RIEHLE & ZÜLLINGHOVEN, 1996] considèrent que les patterns ont d'indéniables qualités dont les plus importantes sont :

- l'aide à la conception et à l'implémentation de logiciels,
- le transfert de la connaissance,
- une aide à la compréhension des systèmes et à la gestion de leur complexité,
- la définition ou l'enrichissement d'un vocabulaire commun, ainsi qu'une meilleure compréhension du domaine d'application,
- et enfin, leur complémentarité avec les méthodes et outils existants.

Ces qualités sont détaillées dans les sous-sections suivantes.

4.2.4.1. Aide à la conception de logiciels

La conception de logiciels est une tâche difficile, surtout si elle a pour objectif la production d'entités ou d'objets réutilisables et d'une maintenance aisée. En effet, pour atteindre cet objectif, le concepteur doit surmonter différents problèmes tels que : trouver les bons objets ou entités et les bonnes abstractions, identifier le bon niveau de granularité, envisager les besoins futurs, obtenir flexibilité et modularité,… Comme le souligne [SCHMIDT & AL., 1996], les patterns aident à obtenir une bonne conception plus rapidement. Ils permettent également de déterminer où sont les points

[54] L'intérêt des patterns pour l'ingénierie des SIC est notamment explicité dans les publications [COUTURIER & SEGURAN, 2003] et [SEGURAN & COUTURIER, 2003].

variables de la conception, les choix qui risquent d'être à la source de la reconception du logiciel, ou ceux qui permettent au logiciel d'évoluer sans heurt lors de la modification des contraintes ou des besoins.

4.2.4.2. Transfert de la connaissance

Les patterns permettent de transférer le savoir et le savoir-faire des concepteurs ou développeurs expérimentés ayant rencontré ou résolu plusieurs fois les mêmes problèmes. Ils capturent explicitement la connaissance que les développeurs expérimentés possèdent déjà implicitement [SCHMIDT, 1995]. Cette capitalisation du savoir-faire offre ainsi un gain de temps et d'efficacité en réduisant les tâtonnements fastidieux nécessaires à l'élaboration d'un logiciel [GAMMA & AL., 1995].

4.2.4.3. Aide à la compréhension des systèmes

Les patterns apportent également une meilleure compréhension des systèmes et permettent ainsi d'aborder des systèmes complexes de manière plus simple. Ils permettent par exemple de comprendre le fonctionnement d'un framework sans avoir à se pencher sur les détails de son implémentation [PREE, 1995]. Cette réduction de la complexité est assurée par les qualités d'abstraction des patterns : ils capturent les niveaux essentiels d'une architecture en occultant les détails non pertinents et permettent d'en obtenir une description à un haut niveau d'abstraction [SCHMIDT, 1995].

Un système utilisant des patterns de façon visible, c'est-à-dire en recourant à leur utilisation dans sa documentation, est aussi plus facile à appréhender. Les arguments que les patterns utilisés apportent aux décisions prises lors du développement du système sont très convaincants et évitent de longues explications pour le concepteur du logiciel (car elles sont déjà intégrées dans la description des patterns). Ces arguments sont également très pratiques pour le réutilisateur car cela lui évite d'avoir à entrer dans les détails souvent complexes d'une conception pour comprendre la structure de la construction du système.

4.2.4.4. Définition et enrichissement d'un vocabulaire commun

Les patterns fournissent un supplément de vocabulaire commun aux concepteurs, modélisateurs et développeurs, qui peuvent faire référence à des abstractions d'un niveau plus élevé que celles qu'ils utilisent habituellement [FOWLER, 1997] [GAMMA & AL., 1995]. Une collection de patterns décrit un véritable langage abstrait facilitant la communication autours des abstractions, des concepts et des techniques que décrivent ou portent ces patterns. La documentation s'en trouve ainsi enrichie et simplifiée. Les patterns facilitent également la manipulation, le passage et la communication des

spécifications, des modèles et des concepts, notamment avec le personnel non technique, qui n'est pas apte à comprendre les détails d'implantation mais est, par contre, plus disposé à saisir les abstractions qu'ils véhiculent [SCHMIDT, 1995].

4.2.4.5. Complémentarité avec les outils et méthodes existants

Les patterns n'empêchent en aucun cas le concepteur ou le développeur d'utiliser d'autres moyens de conception ou d'implantation. Ils sont utilisables parallèlement à ces moyens, que ce soit des méthodes d'analyse, de conception ou des outils de développement.
Les méthodes d'analyse ou de conception n'ont pas réussi à capturer les bonnes pratiques des experts. Si l'on prend pour exemple, la conception objet, l'abord en a toujours été difficile et la compréhension des concepts inhérents toujours fastidieuse. Les patterns comblent cette insuffisance en montrant l'utilisation des techniques primitives du paradigme utilisé et en présentant les bonnes pratiques de conception ou d'analyse.

5. Conclusion

Ce chapitre a présenté le cadre de notre travail de recherche, à savoir l'ingénierie des systèmes d'information coopératifs. Il nous a également permis de motiver notre choix d'avoir logiciel dans l'objectif de faciliter cette ingénierie. Les patterns permettent de décrire l'existant, de formaliser l'expertise, de la communiquer et de l'appliquer, et ce, à tous les niveaux, pendant toutes les phases de la construction du logiciel. Les patterns sont donc une technique de génie logiciel d'une grande valeur. D'autant plus quand les systèmes inhérents, comme c'est le cas des systèmes d'information coopératifs, sont d'une grande complexité. Ces raisons nous ont conforté dans notre choix d'utiliser des patterns pour l'ingénierie des SIC.

L'ingénierie des systèmes d'information coopératifs est basée sur la réutilisation d'une collection de patterns. L'essentiel du travail réalisé consiste ainsi à définir des patterns pour le domaine des SIC. Le chapitre suivant présente les types de patterns que nous avons développés, la démarche d'ingénierie des patterns, leur formalisme, leur réutilisation afin de construire des applications ainsi que les travaux du domaine existants (Cf. Chapitre 2). Les patterns spécifiant un ensemble de fragments de modèles représentant les éléments du SIC à différents niveaux d'abstraction, les chapitres 3 et 4 présentent, respectivement, les patterns dédiés à la phase d'analyse de l'ingénierie des SIC et les patterns dédiés aux phases de conception et d'implantation.

Chapitre 2 :
Présentation des patterns de coopération

1. Introduction

Le chapitre précédent nous a permis de justifier l'adoption d'une approche de réutilisation à base de patterns dans le but de faciliter l'ingénierie des systèmes d'information coopératifs. Le chapitre 2 a pour objectif, d'une part, de présenter succinctement et positionner les patterns que nous proposons et, d'autre part, d'exposer les travaux existants du domaine destinés à améliorer cette ingénierie. Nous nous situerons par rapport à ces travaux et montrerons qu'un certain nombre de problèmes n'ont pas été traités jusqu'alors.

Nous présentons tout d'abord la typologie des patterns conçus (§ 2) et décrivons la démarche d'ingénierie de ces patterns (§ 3). La section 4 expose le formalisme de pattern adéquat supportant les éléments gérés dans le SIC et favorisant le dialogue entre les acteurs. La section 5 définit les moyens pour assurer une gestion efficace des patterns afin de faciliter leur réutilisation. Enfin, la dernière partie (§ 6) présente les travaux du domaine et établit une comparaison de ces travaux avec nos propositions.

2. Typologie des patterns

Les patterns de coopération proposés pour l'ingénierie des systèmes d'information coopératifs[55] permettent d'élaborer, après réutilisation, l'ensemble des modèles du SIC, à différents niveaux d'abstraction. Ce sont des patterns de domaine[56], i.e. portant sur un domaine spécifique, celui de la coopération de systèmes d'information.

[55] Nous les appellerons par la suite "Patterns de coopération" ou "Patterns de coopération de systèmes d'information"

[56] Notre définition d'un pattern de domaine étend celle de Fowler [FOWLER, 1997] seulement limitée aux patterns d'analyse liés à un domaine d'application. Nous qualifions ainsi de pattern de domaine tout pattern (d'analyse, de conception,…) lié à un domaine d'application spécifique.

Aux patterns de coopération s'ajoutent des patterns de support de coopération qui s'appliquent quel que soit le domaine d'application et traitent des problèmes de construction de systèmes informatiques à partir des patterns de coopération. Ces patterns décrivent comment utiliser les patterns de coopération et comment les assembler afin d'obtenir l'architecture ciblée [FOWLER, 1997].

L'ensemble des patterns conçus forme un système de patterns dénommé *système de patterns de coopération*.

Nous présentons ces différents types de patterns dans les sections suivantes.

2.1. Patterns de coopération

Les patterns de coopération sont des patterns de domaine centrés sur les phases amont du processus de développement logiciel (phases de spécification), i.e. les phases d'analyse et de conception.

Adaptés à un domaine spécifique, nos patterns de coopération permettent de limiter le problème de la « généricité et de l'abstraction excessives de la plupart des patterns d'analyse et de conception existants » [PORT, 1998]. Les patterns de coopération sont des collections de classes qui représentent des partitions bien définies (encapsulées, aux frontières claires) d'un domaine particulier. Port considère que des patterns trop abstraits et trop génériques ne sont pas utilisés et, ainsi, que le niveau de spécificité des patterns de domaine est rendu nécessaire. Cependant, il relève le fait qu'il est important que ces patterns soient les plus généraux et invariants possibles à l'intérieur de ce domaine. Plus précisément, les principales idées sous-jacentes aux patterns de domaine sont [PORT, 1998] :

- ils permettent de découvrir les principes d'un domaine et les utilisent comme des partitions afin de spécifier ou construire des systèmes dans ce domaine,
- les partitions des domaines sont des parties inhérentes au domaine et fournissent des contraintes applicables à n'importe quelle application dans ce domaine,
- les partitions sont de faible granularité, bien définies et gérables,
- les rubriques - et connaissances inhérentes - telles que les relations entre les patterns ont du sens dans le contexte du domaine,
- les partitions sont faciles à identifier à partir d'exemples d'utilisation.

Les patterns de coopération aident le concepteur à mieux structurer sa représentation d'un environnement et guident sa perception du monde réel. Ils aident à aborder les systèmes d'informations coopératifs complexes d'une manière simple sans qu'il soit nécessaire d'en étudier les détails d'implantation.

Nous détaillons les types de patterns utilisés lors de la phase d'analyse (§ 2.1.1) et ceux utilisés lors de la phase de conception (§ 2.1.2) dans les sections suivantes.

2.1.1. Patterns de coopération dédiés à l'analyse des SIC

Les patterns de coopération dédiés à la phase d'analyse des SIC sont les *patterns d'analyse de coopération.* Ces derniers sont comparables aux patterns d'analyse de domaine proposés par Fowler [FOWLER, 1997] (Cf. section 4.2.1.1-b) du chapitre 1), mais appliqués à la coopération de SI. Ils mettent en jeu les connaissances de domaine détenues par les concepteurs de SIC (i.e. principalement les chercheurs du domaine de la coopération de systèmes d'information). Ils permettent de définir un vocabulaire commun propre au domaine de la coopération, classifient et organisent les connaissances du domaine afin de favoriser leur réutilisation.

Les patterns d'analyse de coopération permettent d'identifier et de spécifier les aspects fonctionnels du SIC, i.e. les entités du SIC en termes de classes et de relations entre les classes et fournissent des fragments de modèle d'analyse (modèle de domaine) pour représenter les SIC. Le concepteur du SIC pourra alors réaliser le modèle du système en adaptant le modèle de domaine fourni dans la solution du pattern (Cf. section 4.2.3 du chapitre 1). L'ensemble des patterns d'analyse de coopération représente ainsi un modèle d'analyse global décrivant toutes - ou si ce n'est toutes, une grande partie - les entités du SIC (structure) et leurs relations. Les solutions des patterns d'analyse doivent ainsi permettre de constituer le modèle d'analyse de tout SIC et sont donc indépendantes d'un système d'information coopératif spécifique.

Nous proposons deux classes de patterns d'analyse de coopération :

- Des patterns d'analyse de coopération qui sont à la fois utilisables pour représenter certains concepts des SIC et ceux d'une (ou plusieurs) approche(s) de coopération (approche fédérée, à base de médiation de schémas ou à base de médiation de contextes[57]). Chaque pattern fournit alors un fragment de modèle d'analyse permettant de représenter certains concepts

[57] Compte tenu de l'ancienneté et de l'importance des points communs entre les systèmes fédérés fortement couplés et les systèmes fédérés faiblement couplés, nous regroupons ces systèmes sous la dénomination « approche fédérée » (Cf. section 3.1.3.1 du chapitre 1). Au contraire et en raison de leurs spécificités saillantes, nous distinguons les systèmes à base de médiation de schémas des systèmes à base de médiation de contextes.

spécifiques d'un SIC ou ceux spécifiques à une (ou plusieurs) approche(s) de coopération. Ainsi, ces patterns peuvent s'utiliser à deux niveaux : pour modéliser les concepts d'un SIC ou pour modéliser les concepts d'une approche de coopération spécifique. Des patterns d'analyse raffinant les patterns précédents (intégration de concepts spécifiques à une ou plusieurs approches coopératives au sein de la solution) ont parfois également été conçus et ne sont utilisables que pour modéliser les SIC issus de cette (ou ces) approche(s).

- Des patterns d'analyse de coopération qui ne permettent de représenter que certaines entités spécifiques aux SIC mais ne sont pas applicables aux approches de coopération. Chaque solution des patterns fournit ainsi un fragment de modèle d'analyse utilisable uniquement pour représenter certains concepts des SIC.

L'ensemble des patterns d'analyse de coopération forme un sous-système de patterns, appelé *système de patterns d'analyse de coopération*. Ce système est composé de six patterns : "Modélisation des types de conflit", "Modélisation des sources d'information gérées", "Modélisation du référentiel de coopération", "Modélisation des tâches de coopération effectuées par les agents", "Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas" et "Modélisation des processus de coopération".

Dans le chapitre 3, nous présentons les patterns de coopération relevant de ce type.

2.1.2. Patterns de coopération dédiés à la conception des SIC

Les patterns de coopération dédiés à la conception des SIC sont les *patterns d'architecture de coopération* et les *patterns de conception de coopération.* Nous présentons ces deux types de patterns dans les sections suivantes (§ 2.1.2.1 et 2.1.2.2).

2.1.2.1. Patterns d'architecture de coopération

Les *patterns d'architecture de coopération* traitent des différents styles[58] d'architecture logicielle possibles d'un SIC, d'un point de vue structurel et organisationnel et aident ainsi à déterminer sa structure de base. Ces styles d'architecture sont dépendants des approches de coopération. Ainsi, en fonction de l'approche de coopération retenue pour concevoir le SIC, différents niveaux ou couches seront spécifiés dans le pattern associé et faciliteront la détermination de l'architecture logicielle du

[58] Un style est une aide générique à l'élaboration de structures architecturales. Il véhicule des propriétés spécifiques (modèle et sa sémantique) à ces architectures [SHAW & GARLAN, 1996].

SIC (début de la phase de conception). En ce sens, les patterns d'architecture de coopération ont la même philosophie, par exemple, que les patterns d'architecture « Architecture 2-tiers », « Architecture 3-tiers » ou « Présentation et logique d'application » de Fowler [FOWLER, 1997].

L'ensemble des patterns d'architecture de coopération forme un sous-système de patterns, appelé *système de patterns d'architecture de coopération*. Ce système est composé de trois patterns : "Architecture d'un SIC issu de l'approche fédérée", "Architecture d'un SIC issu de l'approche à base de médiation de schémas" et "Architecture d'un SIC issu de l'approche à base de médiation de contextes".

2.1.2.2. Patterns de conception de coopération

Les *patterns de conception de coopération* décrivent les éléments conceptuels nécessaires à la mise en œuvre d'un SIC et définissent les traitements informatiques associés. Ils permettent plus précisément de compléter la description des entités d'analyse identifiées par les patterns d'analyse de coopération – notamment en terme de comportement (d'opérations sur les classes) et de collaboration entre entités – et éventuellement de spécifier de nouvelles entités dédiées à la conception de SIC. Cette description intermédiaire, permettant de passer d'un modèle d'analyse à un modèle de conception, nécessite la décomposition et l'enrichissement des entités métier, mises en exergue par les patterns d'analyse, pour aboutir aux traitements et entités informatiques associés ainsi que la mise en évidence des collaborations entre les entités informatiques du SIC.

Nous avons vu que les entités (granularité, définition,…) des SIC diffèrent (Cf. section 3.1.3.3 du chapitre 1) et que leur comportement est également variable : les traitements réalisés (décomposition, granularité, définition,…) varient non seulement en fonction de l'approche de coopération dont est issu le SIC, mais aussi et surtout des propres choix et objectifs du concepteur du SIC. Ainsi, chaque SIC définit ses propres entités et ses propres traitements. A l'instar de Ganguly & al. [GANGULY & AL., 2001] et Fernandez & Zhao [FERNANDEZ & ZHAO, 2000] (Cf. travaux du domaine en section 6), nous pensons que les patterns de conception de coopération ne peuvent être que dépendants du SIC à concevoir et ne peuvent représenter au sein de leur solution un modèle conceptuel abstrait qu'il serait possible d'adapter pour représenter les entités informatiques du SIC et leur traitement. Ainsi comme la plupart des travaux existants du domaine, nos patterns de conception de coopération dédiés à la représentation des entités de conception du SIC et de leurs traitements sont directement dérivés d'un projet coopératif spécifique, en l'occurrence du SIC ACSIS. Ils ne permettent de ne réutiliser que les entités et traitements informatiques de ce SIC.

En ce qui concerne la spécification des interactions entre les entités d'un SIC, nous avons conçu un pattern de conception générique permettant de modéliser ces interactions et applicable à la conception de tout SIC. Ce pattern, après adaptation, permet de modéliser les interactions entre les composants fonctionnels du SIC ACSIS.

Ainsi, la plupart des patterns de conception de coopération que nous proposons sont dépendants du système d'information coopératif ACSIS et ne permettent donc pas de concevoir tout SIC, mais servent à capitaliser et réutiliser plus facilement les éléments de conception de ce SIC lors de la construction de processus coopératifs entre sources d'information. Cette approche répond à l'un des objectifs assignés aux recherches sur l'ingénierie des SIC, à savoir l'amélioration de la réutilisabilité des SIC existants (Cf. section 3.2 du chapitre 1).

En général, les patterns de conception constituent une base d'expériences pour construire des composants réutilisables, et ils agissent comme des blocs de construction, à partir desquels des systèmes plus complexes peuvent être construits [FRONT-CONTE & AL., 1999]. Certains de nos patterns de conception de coopération sont génératifs (au sens défini précédemment, Cf. section 4.2.1.2 du chapitre 1) et fournissent des composants logiciels réutilisables, spécifiés dans un langage normalisé, l'*OMG Interface Definition Language (OMG IDL)* [OMG, 2002], et le savoir-faire associé, en utilisant la structure du pattern comme documentation de ces composants. Certains patterns de conception doivent ainsi nous permettre de capitaliser les connaissances du domaine de la coopération et également de générer les composants logiciels qui constitueront les futurs systèmes.
A l'instar de Rideau [RIDEAU, 1999], certains de nos patterns de conception établissent ainsi une bijection entre le modèle et son implantation et sont composés de deux parties :

- Un modèle de connaissances capturant le savoir-faire,
- Un composant (binaire) exécutable et réutilisable. L'application finale correspondant à la combinaison de plusieurs composants.

L'obligation de définition OMG IDL de tout élément réutilisable permet par ailleurs de contraindre la réutilisation. Cette définition claire, explicite et contraignante facilite la réutilisation du composant. Il reste cependant à décliner les composants spécifiés dans les patterns de coopération en des composants CORBA/CCM, COM/COM+ ou EJB en fonction de l'environnement choisi par le développeur d'application. Notre travail étant accès sur la spécification des patterns dédiés à l'ingénierie des SIC, nous n'avons pas implanté ces composants et n'avons, jusqu'à présent, que spécifié les interfaces des composants en OMG IDL, spécifications intégrées au sein de certains patterns de conception de coopération.

L'ensemble des patterns de conception de coopération forme un sous-système de patterns, appelé *système de patterns de conception de coopération*. Ce système est composé de six patterns : "Objet Descriptif des Données", "Lien sémantique", "Traduction ACSIS", "Création des liens sémantiques", "Contract-net protocol" et "Interaction d'agents".

Le chapitre 4 présente les patterns dédiés à la conception des SIC, i.e. les patterns d'architecture et patterns de conception de coopération.

2.2. Patterns de support de coopération

Les *patterns de support de coopération* proposés sont philosophiquement très proches des patterns de support définis par Fowler [FOWLER, 1997] (Cf. section 4.2.1.1-b) du chapitre 1). Nos patterns de support de coopération traitent des problèmes qui interviennent quand un système informatique doit être construit à partir des patterns de coopération. Ils décrivent comment utiliser les patterns de coopération et comment les assembler afin d'obtenir l'architecture ciblée.

Nous avons spécifié deux types originaux de patterns de support de coopération, les patterns de support technique (§ 2.2.1) et les patterns de support d'utilisation (§ 2.2.2).

2.2.1. Patterns de support technique

Les *patterns de support technique* sont des patterns qui décrivent comment construire une application à partir des patterns de conception. Ces patterns s'utilisent lors des phases de conception (en fin de phase) et d'implantation des SIC. Ainsi, ce sont soit des patterns dédiés à l'implémentation des entités spécifiées dans les patterns de conception de coopération soit des patterns dédiés à leur implantation.

Les patterns de support technique dédiés à l'implémentation sont, généralement, des patterns de conception spécifiques à un paradigme (objet, agent artificiel,…). Dans le cas, par exemple, d'un SIC spécifiant des agents artificiels en charge de la réalisation des traitements coopératifs, ces patterns vont permettre de construire un système multi-agents, de spécifier les interactions entre les agents (spécification des modèles de coopération ou collaboration),… D'autres patterns, explicitant par exemple les communications (synchrone, asynchrone,…) entre objets, peuvent être utilisés en fonction des paradigmes inhérents au SIC.

Les patterns de support technique dédiés à l'implantation sont spécifiques aux plates-formes d'exécution ou aux langages de programmation. Ces patterns peuvent être des patterns de

conception, mais sont la plupart du temps des idiomes (idiomes Java, idiomes CORBA,…). Ils permettent notamment de générer et d'assembler les composants spécifiés dans les patterns de conception de coopération génératifs dans l'environnement technique souhaité afin de construire une micro-architecture[59] (i.e., un ensemble de composants) destiné à faciliter la création de processus coopératifs entre sources d'information. Ces patterns sont fonction de l'environnement de développement choisi par le développeur d'applications. Nous avons conçu et présentons ultérieurement (Cf. chapitre 4) deux patterns de support technique assurant la transformation des modèles de conception, intégrés dans les patterns de conception de coopération et indépendants des plates-formes en des modèles dépendants des plates-formes techniques.
Notre démarche est ainsi compatible avec la spécification Model Driven Architecture (MDA) de l'Object Management Group (OMG) [OMG, 2003] basée sur la séparation des modèles d'implantation, dépendants des plates-formes techniques, des modèles de conception indépendants des systèmes d'exécution ciblés. Les principaux points de convergence concernent :

- la construction d'architectures logicielles par ingénierie de modèles de conception spécifiés dans un langage de description standard,
- la génération des composants logiciels implémentant les fonctionnalités du système par des générateurs de code liés à la plate-forme d'exécution et exploitant directement les modèles de conception,
- l'explicitation de la sémantique des liens entre les modèles pour exploiter pleinement les relations de transformation et d'intégration entre les modèles.

L'ensemble des patterns de support technique forme un sous-système de patterns, appelé *système de patterns de support technique*. Ce système est notamment composé de deux patterns que nous avons conçus : "Démarche de transformation de modèle" et "Traduction des interfaces spécifiées en OMG IDL vers Microsoft IDL".

2.2.2. Patterns de support d'utilisation

Chaque pattern de coopération et de support technique ne donnant qu'une vision atomique des éléments réutilisables (et ce, même s'il propose une rubrique listant ses relations avec d'autres patterns), le concepteur n'aura pas de vision globale de la collection et ne saura pas non plus comment se guider à travers cet ensemble. Plusieurs solutions ont été proposées permettant de résoudre ce problème.

[59] Comme indiqué précédemment, cette génération est pour l'instant seulement limitée à celle des interfaces spécifiées dans les patterns de conception de coopération.

Une manière possible de présenter un langage de patterns est l'utilisation d'exemples ou d'études de cas fonctionnant à travers la collection de patterns. Bien que de récentes et nombreuses collections de patterns procèdent de cette façon ([BERGIN, 2002], [MEIRA & AL., 2001], [SCHMIDT & AL., 2001], [VÖLTER & AL., 2002],…) et qu'un exemple concret semble un bon moyen pour relier des patterns, la plupart des collections ne fournissent qu'un seul et unique exemple. Le danger est que cet exemple soit utilisé littéralement par le concepteur et que les autres relations entre les patterns ne soient pas usitées.

D'autres collections de patterns utilisent une forme abstraite de séquences : elles ne sont pas basées sur l'usage d'un exemple concret mais indiquent à l'utilisateur quels patterns doivent être appliqués dans une séquence pour résoudre un problème (par exemple, la collection proposée par [FRICKE & VÖLTER, 2001]). Ces collections utilisent une forme textuelle pour décrire cette séquence ou, beaucoup plus rarement, une représentation schématique[60]. La présentation de cette séquence est également externe à la collection de patterns spécifiée.

Nous avons opté pour cette deuxième solution. Notre approche ne décrit cependant pas une séquence de patterns à utiliser pour résoudre un problème donné sous une forme textuelle, mais sous la forme d'un diagramme et l'intègre dans un pattern spécifique appelé, *pattern de support d'utilisation*. Ainsi, les patterns de support d'utilisation décrivent l'utilisation d'un système de patterns, et font partie intégrante de ce système. Un pattern de support d'utilisation décrit la séquence de patterns devant être appliquée pour résoudre un problème. Ce dernier est défini au sein des rubriques liées à la description du problème et du contexte et la solution du pattern présente, sous la forme d'un diagramme d'activités, la séquence de patterns à appliquer pour résoudre ce problème. Ainsi, lors de l'utilisation d'un système de patterns, les utilisateurs traversent les différents chemins de ce diagramme d'activités dont les graphes représentent les patterns à appliquer, en choisissant à chaque fois un chemin spécifique au sous-problème à résoudre. Un pattern de support d'utilisation constitue donc un point d'entrée d'un système de patterns et a pour objectif de guider l'utilisateur dans la réutilisation de celui-ci.

Un ou plusieurs patterns de support d'utilisation sont associés à chaque système de patterns conçu (système de patterns d'analyse de coopération, système de patterns d'architecture de coopération, système de patterns de conception de coopération, système de patterns de support technique). Chacun d'eux est détaillé après le système de patterns auquel il s'applique (Cf. chapitres 3 et 4).

[60] Par exemple, Deneckere & Souveyet [DENECKERE & SOUVEYET, 2001] utilisent le concept de carte de processus, i.e. graphe composé de nœuds et d'arcs, pour organiser les patterns d'un système ou d'un langage.

2.3. Positionnement des patterns conçus

Le positionnement des différents types de patterns que nous proposons est présenté en figure suivante (cf. Figure 9). Cette figure indique également les noms des patterns conçus pour chaque type.

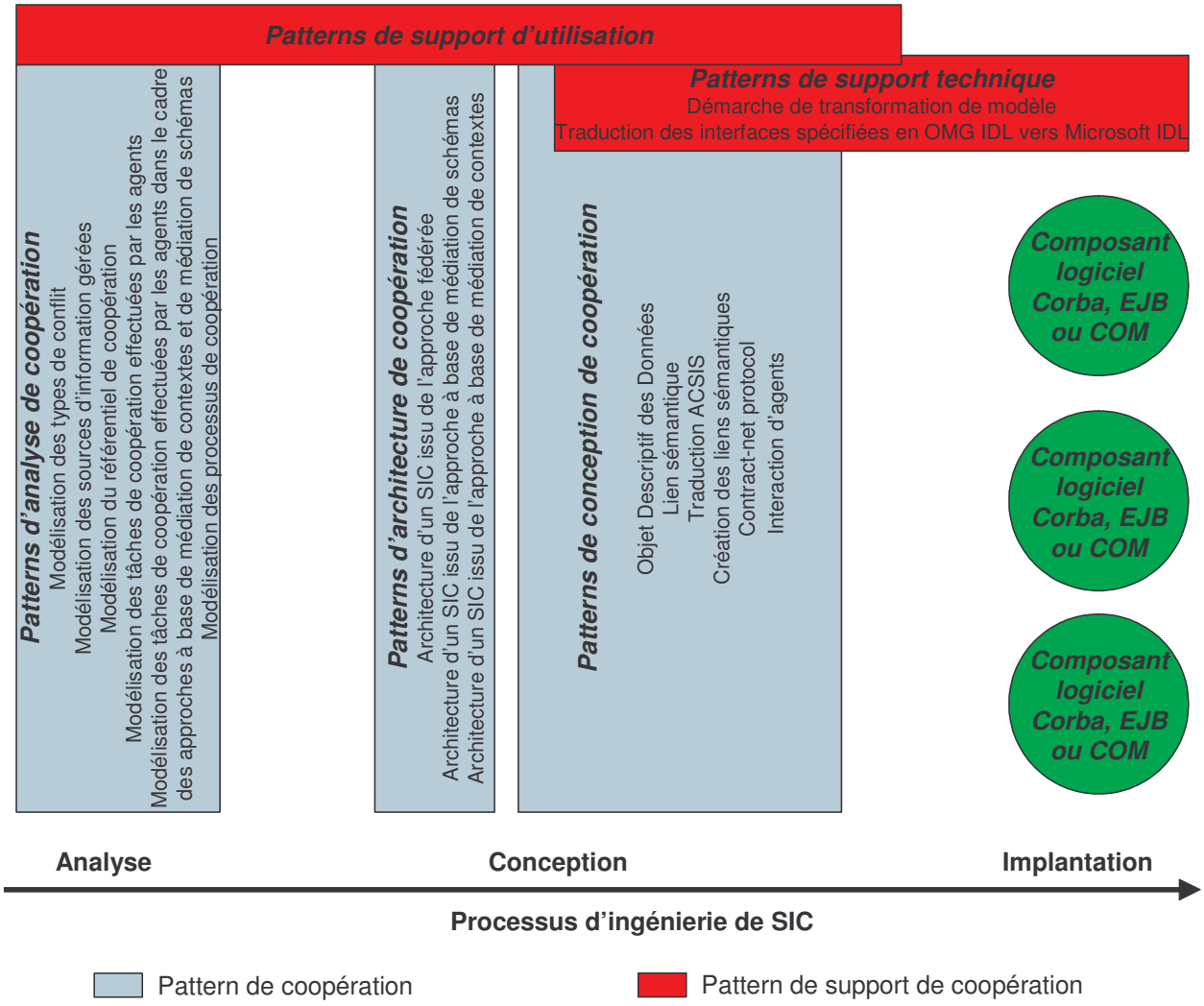


Figure 9 : Positionnement des patterns conçus en fonction des étapes du processus d'ingénierie de SIC

REMARQUE — Comme indiqué précédemment, nous focalisons nos travaux sur les patterns, et non sur les composants logiciels présentés en Figure 9.

Après avoir précisé la nature et le type des patterns conçus, nous explicitons la démarche d'ingénierie des patterns de coopération dans la section suivante (§ 3).

3. Démarche d'ingénierie des patterns de coopération

Cette section a pour objectif de présenter la démarche d'ingénierie des patterns de coopération que nous avons conçus à partir des connaissances du domaine de la coopération de SI.

3.1. Introduction

Comme présenté au chapitre 1, l'ingénierie de systèmes d'information coopératifs par réutilisation nécessite, d'une part, la production d'avoirs logiciels réutilisables et, d'autre part, de réutiliser ceux-ci lors du développement de ces systèmes. Ainsi deux activités du processus d'ingénierie de SIC peuvent être distinguées :

- l'activité d'*ingénierie des avoirs logiciels* (développement d'un système réutilisable - *design for reuse*) consistant à identifier, spécifier et organiser les avoirs logiciels à utiliser lors de la spécification des SIC,
- l'activité d'*ingénierie des SIC* (développement d'un système par réutilisation - *design by reuse*), proprement dite, ayant pour objectif la recherche, la sélection, l'adaptation et l'intégration des avoirs conçus lors de l'étape précédente afin de spécifier et développer le SIC.

Plus précisément, dans le cadre de notre approche basée sur la conception et la réutilisation de patterns, l'activité d'ingénierie des avoirs logiciels consiste à identifier les problèmes souvent rencontrés lors de l'ingénierie de systèmes d'information coopératifs, puis à fournir des solutions aux problèmes identifiés et à formaliser l'ensemble des problèmes/solutions sous forme de patterns. L'activité d'ingénierie des SIC consiste à offrir des moyens pour identifier les problèmes à résoudre, sélectionner les patterns qui résolvent ces problèmes et enfin réutiliser (après adaptation ou recopie) les solutions proposées.

L'identification et la conception des patterns sont donc essentielles dans ce processus puisqu'elles ont pour objectif de produire les ressources réutilisables. La suite de cette section est structurée en fonction de ces deux tâches. La section 3.2 présente la démarche retenue pour identifier les différents éléments réutilisables (problèmes), ainsi que les sources d'information dont sont issus ces éléments ; la section 3.3 décrit la manière de spécifier les patterns (solutions) associés aux problèmes identifiés.

3.2. Identification des éléments réutilisables (problèmes)

Préalablement à la conception des patterns, l'ingénierie des patterns nécessite l'identification des éléments susceptibles d'être réutilisables. Cette section présente les techniques d'identification des éléments réutilisables que nous avons utilisées, ainsi que les sources de connaissances à partir desquelles ces éléments ont été identifiés. Ces sources de connaissances et techniques varient selon les différents types de patterns de coopération conçus. Avant de présenter la technique et les sources de connaissances associées à chaque type de pattern de coopération, nous faisons un bref aperçu des techniques d'identification existantes.

3.2.1. Préambule : techniques d'identification des éléments réutilisables

Deux approches d'identification des éléments réutilisables, pouvant être complémentaires, sont distinguées dans la littérature : l'approche de *rétro-ingénierie* et l'approche d'*ingénierie de domaine*.

La *rétro-ingénierie* provient de l'ingénierie des systèmes matériels [REKOFF, 1985]. Dans ce domaine, la rétro-ingénierie consiste à développer les spécifications d'un système matériel complexe produit à grande échelle en examinant des exemplaires de ce produit (par exemple, des microprocesseurs). Dans le domaine du génie logiciel, les objectifs de la rétro-ingénierie[61] sont différents mais le principe est le même. Il s'agit d'une démarche ascendante visant à abstraire des éléments réutilisables à partir de l'analyse des avoirs existants tels les logiciels, schémas conceptuels, documents de spécification technique, etc. Cette approche est basée sur l'utilisation de comparaisons permettant de rechercher les similarités et différences entre avoirs afin d'en dégager les éléments réutilisables.
Trois types de rétro-ingénierie peuvent être distingués [SEMMAK, 1998] [CAUVET & AL., 2001] :

- Les *modèles de réutilisation* sont des ensembles d'attributs (attributs de réutilisation) permettant de mesurer la réutilisabilité d'un élément [CALDERIA & BASILI, 1991]. Ce type de modèle fournit aussi pour les attributs, les valeurs ou les plages de valeurs acceptables pour qualifier un élément d'avoir logiciel candidat à la réutilisation. Les attributs de réutilisation sont choisis pour mesurer l'écart entre le coût de développement et le coût de réutilisation d'un composant. Cette approche a pour principal intérêt d'automatiser le processus d'analyse des avoirs à partir desquels les composants sont extraits ; en revanche, elle ne permet pas d'augmenter le degré d'abstraction des composants, ni d'identifier les composants de haut niveau. Cette approche est notamment utilisée dans le cadre du projet

[61] Selon certains, la rétro-ingénierie correspondrait à "reconstruire le cochon à partir des saucisses et du jambon...".

Esprit F3 [BELLINZONA & AL., 1994] pour identifier les entités réutilisables dans la construction de schémas conceptuels Entité/Association.

- L'*analyse des similarités* est basée sur la comparaison d'éléments provenant de différents produits d'un même domaine dans le but de dégager des éléments candidats à la réutilisation [CASTANO & DE ANTONELLIS, 1992]. Elle met en oeuvre des techniques de recherche de similarités entre éléments permettant de comparer et de hiérarchiser ces derniers, ainsi que des techniques de généralisation qui visent à produire des entités génériques factorisant les propriétés d'éléments similaires.
- La *recherche d'analogies* identifie, comme l'approche précédente, les éléments réutilisables en comparant les éléments communs à plusieurs systèmes [MAIDEN, 1991]. Cependant, la différence essentielle réside dans la nature des éléments comparés. Alors que l'analyse des similarités évalue la proportion d'attributs partagés (similitudes syntaxiques), la recherche d'analogies évalue les structures relationnelles abstraites partagées par les systèmes (similitudes sémantiques). Ainsi, l'idée est de rechercher les structures similaires si les instances concernées sont issues d'une même structure générique. En outre, la recherche d'analogies peut être utilisée sur des systèmes appartenant à des domaines différents.

L'approche d'*ingénierie de domaine* peut être définie simplement comme l'activité d'identification des objets et des opérations d'un champ d'application [PRIETO-DIAZ, 1990] [ARANGO, 1994]. Plus précisément, elle consiste d'une part à analyser le domaine, i.e. à identifier les connaissances d'un champ d'application, les classifier, les structurer et les abstraire, et d'autre part à fournir un environnement qui en permette l'utilisation dans le développement de tous les systèmes appartenant à ce champ d'application [SEMMAK, 1998]. Ainsi, l'ingénierie de domaine permet d'identifier les connaissances génériques liées à un domaine d'application spécifique afin de pouvoir les réutiliser et produire un modèle de domaine[62] générique constituant un cadre de référence pour le développement de tous les systèmes appartenant à ce domaine.

Semmak [SEMMAK, 1998] distingue deux stratégies de mise en oeuvre de l'ingénierie de domaine :

- une *stratégie descendante* où l'ingénierie de domaine est centrée directement sur la modélisation des problèmes et des décisions qui conduisent le processus de résolution de ces problèmes. Un domaine est vu comme une classe de problèmes pour lesquels différentes solutions sont envisageables. Elle met en avant la définition de problèmes pour lesquels plusieurs systèmes d'information différents peuvent être décrits. Les modèles de domaine ainsi obtenus sont orientés problèmes [CAUVET & AL., 2001].

[62] La littérature utilise très souvent les termes « modèle de domaine » et « référentiel de domaine » de façon interchangeable.

- une *stratégie ascendante* où l'ingénierie de domaine utilise et analyse des familles de systèmes existants pour produire les objets et les opérations communs qui caractérisent ces systèmes. En partant d'une famille de systèmes de fonctionnalités similaires, l'objectif est de produire un modèle de domaine exprimant les ressemblances et les différences entre les systèmes. Un domaine est alors vu comme un champ d'application pour lequel des systèmes d'information existent déjà. Contrairement à l'approche descendante où le modèle de domaine obtenu est orienté problème, dans l'approche ascendante le modèle de domaine est orienté solution [CAUVET & AL., 2001].

Comme l'ingénierie des SI, l'ingénierie de domaine est organisée selon des méthodes. Plusieurs méthodes d'ingénierie de domaine [NEIGHBORS, 1984] [KANG & AL., 1990] [WARTIK & PRIETO-DIAZ, 1992] [PETERSON & STANLEY, 1994] [SEMMAK, 1998] [WEISS & CHI TAU, 1999] [CZARNECKI & EISENECKER, 2000] présentant de fortes disparités quant aux outils utilisés, aux processus mis en oeuvre et aux résultats produits ont été proposées. Ainsi, Peterson & Stanley [PETERSON & STANLEY, 1994] considèrent que la démarche d'ingénierie de domaine est constituée de trois phases :

- La phase d'*analyse de domaine*[63] (*domain analysis*) est centrée sur l'acquisition et la description des connaissances du domaine d'application. Elle peut aller de la simple définition des frontières du domaine jusqu'à la conceptualisation et la classification détaillée des connaissances et permet d'obtenir un *référentiel de domaine.* Ce référentiel contient des taxonomies d'objets, des modèles de contexte, des modèles fonctionnels, etc.

- La *conception de domaine* (*domain design*) est le processus de construction d'un modèle de conception (*design model*) à partir des produits de l'analyse de domaine et des connaissances acquises lors de l'étude des conceptions logicielles et architectures génériques existantes. Nilson & al. [NILSON & AL., 1994] considèrent que l'objectif de la conception de domaine est de « (…) spécifier et documenter une architecture qui soit suffisamment générique pour la plupart des systèmes du domaine ». Le modèle de conception représente l'architecture générique développée pour le domaine analysé et les composants logiciels sous-jacents. Cette architecture définit la plate-forme logicielle pour toutes les applications du domaine et est basée sur les similarités et différences entre les applications de ce domaine ; elle est représentée par le style d'architecture le plus approprié à la famille d'applications du domaine. Enfin, la conception de domaine permet également de préciser

[63] La littérature utilise très souvent les termes « analyse de domaine » et « ingénierie de domaine » de façon interchangeable. Cependant, la plupart des publications récentes font la distinction entre ces deux termes, l'analyse de domaine étant considérée comme une phase de l'ingénierie de domaine.

les outils permettant de fournir les applications finales. Un environnement constitué de tels outils est qualifié d'*environnement d'ingénierie d'applications* [WEISS & CHI TAU, 1999].

- La phase d'*implantation de domaine* couvre l'implantation de l'architecture, des composants sous-jacents et des outils conçus lors de la phase précédente.

Notons que si la stratégie de mise en oeuvre adoptée est ascendante, des techniques de rétro-ingénierie peuvent être utilisées pour capturer la connaissance du domaine. Les approches de rétro-ingénierie et d'ingénierie de domaine peuvent ainsi être utilisées de manière complémentaire.

Les techniques d'identification des éléments réutilisables, ainsi que les sources de connaissances contenant ces éléments, varient en fonction des types de patterns de coopération que nous avons conçus. Ainsi, l'analyse de domaine a été utilisée pour identifier les éléments réutilisables associés aux patterns d'analyse de coopération ; la spécification des patterns d'architecture de coopération a préalablement nécessité la réalisation d'une phase de conception de domaine ; enfin, la modélisation des patterns de conception de coopération a été permise par une phase préalable de rétro-conception de l'architecture ACSIS. Les sections suivantes (§ 3.2.2 à § 3.2.4) présentent la technique d'identification ainsi que les sources de connaissances utilisées pour chaque type de pattern de coopération.

3.2.2. Les patterns d'analyse de coopération et l'analyse de domaine

L'analyse de domaine constitue la première phase du processus d'ingénierie de domaine. Pour Prieto-Diaz [PRIETO-DIAZ, 1990], l'objectif de l'analyse de domaine est de capturer l'information liée à un domaine d'application afin d'être réutilisée pour développer de nouvelles applications inhérentes à ce domaine. Elle est usitée depuis de nombreuses années pour appréhender et comprendre un domaine, pour expliquer ses aspects statiques et dynamiques ou plus récemment pour produire des avoirs logiciels réutilisables. L'analyse de domaine a pour objectif d'acquérir les connaissances sur un champ d'application et de les structurer dans un modèle de domaine suffisamment générique pour constituer un cadre de référence pour les développements de tous les systèmes de ce domaine. Ce modèle de domaine peut être composé de différents éléments pouvant se compléter :

- La *portée du domaine* (*domain scoping*), permet de fixer les frontières du champ d'application. Cette activité fournit des exemples d'applications appartenant ou situées en dehors du domaine.

- L'*analyse des similarités* (*commonality analysis*) considère à la fois les similarités et variabilités des applications du domaine. Cette analyse permet de produire un document de similarités.
- Le *dictionnaire de domaine* (*domain lexicon*) fournit et définit les termes du domaine.
- Les *notations* (*notations*, *concept modeling*) fournissent un moyen uniforme de représenter les concepts utilisés dans la modélisation du domaine (diagrammes d'objet, d'état-transition, entité-association,…). Lam & McDermid [LAM & MCDERMID, 1997] préconisent l'utilisation de diagrammes UML pour représenter ces concepts en raison de son utilisation généralisée.
- L'*ingénierie des besoins* (*requirements engineering*) comprend le recueil, la définition, la documentation, la vérification et la gestion des besoins qui spécifient les applications du domaine [CLEMENTS & NORTHROP, 2002]. De tels besoins sont également appelés *Features* et représentent les caractéristiques d'un système. Ces caractéristiques sont décrites par des modèles spécifiques tels que FODA[64] [KANG & AL., 1990] permettant de représenter les caractéristiques obligatoires, optionnelles ou alternatives d'un système. Ainsi, ces modèles représentent schématiquement les variabilités documentées au niveau de l'analyse des similarités [CZARNECKI & EISENECKER, 2000].

Les patterns d'analyse de coopération s'adressent à des problèmes qui apparaissent lors de la phase d'analyse des SIC. Il s'agit des concepts et fonctionnalités que les concepteurs souhaitent gérer dans le SIC. Ainsi, l'analyse de domaine revient à explorer les connaissances relatives aux différents concepts manipulés dans les SIC ou les approches de coopération, i.e. les connaissances du domaine. Plus précisément, l'objectif de l'analyse de domaine vise, dans notre contexte, à :

- proposer un cadre terminologique et sémantique des connaissances du domaine de la coopération (ou *dictionnaire de domaine*) permettant de classifier et de définir l'ensemble des concepts ou entités gérés dans les SIC,
- préciser les variabilités et similarités entre les SIC (*analyse des similarités*),
- proposer un modèle de référence[65] (*notations*) qui modélise les entités des SIC (et notamment leurs relations).

Ces trois points sont documentés dans un référentiel de domaine (Cf. section 2 du chapitre 3). Le référentiel de domaine exprime en terme de solutions les besoins fonctionnels et concepts pris en

[64] Feature-Oriented Domain Analysis

[65] Nous qualifierons, par la suite, le modèle de référence de "modèle de domaine". Le référentiel de domaine sera ainsi composé de ce modèle, du dictionnaire du domaine et de l'analyse des similarités.

compte dans les différents systèmes ou approches de coopération existants. Ainsi, le référentiel obtenu constitue une source de connaissances susceptible de contenir des éléments réutilisables.

Notre démarche d'analyse de domaine, i.e. de constitution du référentiel de domaine, est ascendante et a consisté à identifier des solutions (des produits d'ingénierie) communes. Cette analyse a été menée à partir de l'étude des SIC et approches coopératives existants portant principalement sur leurs aspects conceptuels et logiciels : analyse des concepts utilisés, de leurs relations, des modèles de données et des fonctionnalités présentés aussi bien dans les publications de recherche (articles de recherche ou thèses) que dans les prototypes ou applications opérationnelles. Nous avons ainsi dégagé l'ensemble des entités partagées par l'ensemble des systèmes mais également les différences entre les systèmes. Notre analyse est donc menée selon un aspect expérimental et non selon un aspect théorique basé sur l'étude de modèles génériques de SIC proposés dans la littérature, en raison de l'absence, à notre connaissance, de contributions qui explicitent et modélisent tout ou partie des concepts récurrents ou variables gérés dans les systèmes d'information coopératifs.

3.2.3. Les patterns d'architecture de coopération et la conception de domaine

La *conception de domaine* consiste à concevoir une architecture et ses composants pour une famille d'applications. Elle comprend la sélection du style d'architecture [BUSCHMANN & AL., 1996] [SHAW & GARLAN, 1996]. Cette architecture doit permettre de prendre en compte les similarités et variabilités entre les applications du domaine. Elle est définie à partir du modèle spécifié lors de l'analyse de domaine et des connaissances acquises lors de l'étude des conceptions logicielles et des architectures génériques existantes. La conception de domaine décrit également comment les applications concrètes doivent être dérivées de cette architecture et des composants réutilisables fournis.

Dans notre cas, la phase de conception de domaine vise à produire uniquement un (ou éventuellement plusieurs) modèle d'architecture "générique" s'appliquant aux SIC et non à spécifier des composants réutilisables permettant de concevoir des SIC en raison des importantes disparités de concepts et de traitements entre ces systèmes (Cf. section 2.1.2.2). Ce (ou ces) modèle d'architecture est lié aux approches coopératives existantes. Ainsi les connaissances utilisées pour spécifier ce modèle sont de deux ordres :

- Des connaissances théoriques basées sur l'étude des modèles d'architectures génériques existants explicitant les différents niveaux requis d'un SIC. Les différents modèles proposés sont dépendants des approches de coopération. Par exemple, les travaux de Wiederhold

[WIEDERHOLD, 1992], de Jouanot [JOUANOT, 2001] ou de Boucelma & Lacroix [BOUCELMA & LACROIX, 2001], nous ont permis de spécifier deux modèles d'architecture destinés, respectivement, aux SIC de type médiation de schémas et aux SIC de type médiation de contextes ; les travaux de Bouguettaya [BOUGUETTAYA, 1994] [BOUGUETTAYA & AL., 1998] nous ont servi pour proposer un modèle d'architecture destiné aux SIC relevant de l'approche fédérée.

- Des connaissances pragmatiques basées sur l'étude des SIC existants. Ces connaissances nous ont notamment servi pour valider les modèles d'architectures conçus.

3.2.4. Les patterns de conception de coopération et la rétro-conception

Les patterns de conception de coopération étant, pour la plupart, spécifiques au système d'information coopératif ACSIS, les connaissances ont directement été extraites de ce dernier. L'extraction des connaissances a été réalisée d'une part en étudiant les principes et concepts du projet ACSIS issus des thèses et articles publiés et d'autre part par rétro-conception des prototypes existants.

La rétro-conception est une phase du processus de rétro-ingénierie et peut être définie comme un traitement sur un produit connu fonctionnant en arrière pour définir la manière dont il a été conçu [PERCIVAL, 1999]. C'est un processus d'extraction des connaissances d'un système ou d'un logiciel. Ces connaissances sont ensuite conceptualisées avec l'objectif de les utiliser pour le développement de nouveaux systèmes ou logiciels. La rétro-conception a ainsi permis de partir des deux prototypes existants pour aboutir aux modèles inhérents (identification des composantes du SIC ACSIS et de leurs relations). Ces modèles ont ensuite été intégrés *in fine* au sein des patterns de conception de coopération.

Les patterns sont des avoirs réutilisables orientés problème. Or, les éléments réutilisables identifiés ne sont pas toujours orientés problème ni forcément de la granularité adaptée à celle des patterns, il est donc parfois nécessaire de réorganiser ces éléments réutilisables.

3.2.5. Réorganisation des éléments réutilisables

L'analyse de domaine, la conception de domaine et la rétro-conception du SIC ACSIS ont produit des connaissances, qui présentées telles quelles, sont difficilement réutilisables pour un concepteur de SIC souhaitant travailler par réutilisation. Il est ainsi nécessaire d'organiser les connaissances produites ou extraites pour former *in fine* les patterns. L'objectif des patterns de coopération étant de

résoudre un problème spécifique, il est nécessaire de réorganiser certaines connaissances selon une vision « problème » (par exemple, le modèle de domaine obtenu à partir de l'analyse de domaine est orienté solution). Il est, en outre, nécessaire de s'intéresser à la granularité des patterns conçus et de rendre les patterns les moins dépendants les uns des autres.
La démarche de réorganisation des éléments réutilisables revient ainsi à repérer les blocs[66] d'entités, issus des modèles conçus, liés sémantiquement et d'associer à chaque bloc un problème et un pattern. Les modèles sont alors fragmentés de telle sorte que chaque fragment corresponde à un problème récurrent.

Par exemple, à l'issue de l'analyse de domaine, dont les résultats seront développés dans le chapitre suivant (Cf. chapitre 3), le référentiel de domaine élaboré met en évidence l'existence de besoins fonctionnels et de concepts souvent gérés dans les SIC. A partir de ce modèle, nous avons repéré divers blocs ou fragments dont les entités sont sémantiquement proches et peuvent constituer une solution à un problème donné. Chaque fragment est une constante des SIC, c'est-à-dire qu'il est présent dans tous (ou si ce n'est tous, un grand nombre) les SIC existants et peut ainsi être adapté et réutilisé pour en modéliser de nouveaux. Ainsi, les fragments déterminés concernent les types de conflit définis, les sources d'information gérées, le ou les référentiels de coopération conçus, les tâches de coopération spécifiées par les SIC et le regroupement de ces tâches. Des problèmes correspondant à ces fragments ont alors été spécifiés.

3.3. Spécification des solutions des patterns

Une fois l'identification des problèmes à résoudre, et donc des patterns associés, effectuée, se pose le problème de la spécification des solutions proposées dans ces patterns. Cette phase peut être basée sur l'utilisation de solutions existantes et donc sur des patterns existants. Dans ce cas, le problème considéré est comparé au(x) problème(s) traité(s) dans les collections existantes. Plusieurs scénarios[67] peuvent être rencontrés lors de la spécification de la solution [GZARA, 2000] :

(1) le problème est déjà traité dans un pattern dédié à la coopération des SI existant : la solution correspondante est alors adaptée et intégrée dans le pattern nouvellement conçu.

(2) le problème est un raffinement d'un problème déjà traité dans un pattern de coopération existant : une nouvelle solution (spécialisée) est alors définie en se basant sur la solution existante résolvant le problème général.

[66] ou partitions

[67] Remarquons que les scénarios proposés par [GZARA, 2000] sont analogues à ceux spécifiés dans le cycle du raisonnement à partir de cas ou cycle d'Aamodt & Plaza [AAMODT & PLAZA, 1994].

(3) le problème est complexe et peut être ramené à des problèmes plus simples traités dans des patterns de coopération existants : le problème est alors divisé en des sous-problèmes traités dans les patterns existants et plusieurs patterns correspondant à ces sous-problèmes sont conçus. La conception de la solution de chaque pattern est alors identique au point (1).

(4) le problème est nouveau : une nouvelle solution est alors proposée. Cette nouvelle solution correspond, par exemple dans le cadre des patterns d'analyse de coopération, à des fragments du modèle de domaine.

Après avoir explicité les types de patterns conçus et la façon de les concevoir, nous présentons, dans la section suivante, le formalisme utilisé pour représenter les patterns de coopération et support de coopération.

4. Formalisme des patterns de coopération et support de coopération

De nombreux formalismes de patterns ont été proposés. Un formalisme est la structure documentaire adoptée par l'auteur des patterns pour représenter ces derniers. Deux grandes classes de formalismes peuvent être distinguées :

- les formalismes *narratifs*, tels ceux d'Alexander & al. [ALEXANDER & AL., 1977] et de Portland [PORTLAND, 1994], permettent de représenter le traditionnel triplet <problème, solution, contexte> de façon informelle, i.e. non structurée sous forme de rubriques.
- Les formalismes *structurés*, tels ceux de Gamma & al. [GAMMA & AL., 1995], de Coad [COAD, 1992] ou de Fowler [FOWLER, 1997], proposent une représentation du problème, de la solution et du contexte inhérents au pattern sous forme de rubriques. Les formalismes structurés offrent une meilleure représentation des patterns, qui sont souvent complexes et nécessitent une description généralement longue. Même si l'utilisation de formalismes structurés est aujourd'hui de rigueur et a permis d'accroître l'échange d'information dans la "communauté pattern", le nombre de rubriques proposées diffèrent d'un formalisme à l'autre, ainsi que le degré de détail de celles-ci.

Ces formalismes sont souvent fonction du type de patterns conçus (patterns d'analyse, patterns de conception, idiomes,…), du domaine d'application, mais aussi et surtout des auteurs des patterns eux-mêmes. Ainsi, pour une même phase du développement logiciel, par exemple, la phase d'analyse, de nombreux formalismes de patterns ont été proposés ([COAD, 1992], [FOWLER, 1997], [CHIKHI, 1998], [GZARA, 2000], [GEYER-SCHULZ & HAHSLER, 2001],…). Avant de présenter le formalisme retenu pour décrire les patterns de coopération et patterns de support de coopération, il

est tout d'abord important de se focaliser sur le degré de formalisation des connaissances représentées dans ceux-ci.

4.1. Quel degré de formalisation pour les patterns ?

Parmi de nombreuses options, nous considérons trois approches de modélisation[68] qui peuvent être combinées ou utilisées seules au sein des patterns de coopération et patterns de support de coopération : les langages formels (notations mathématiques), les langages semi-formels (représentations de type diagramme ou code) et le langage naturel.

4.1.1. Les notations mathématiques ou langages formels

Les ordinateurs exécutent des opérations numériques, et le comportement des applications logicielles peut en grande partie être modélisé en utilisant des langages mathématiques tels que Z [SPIVEY, 1989] [SPIVEY, 1992], B [ABRIAL & AL., 1991] ou VDM[69] [JONES, 1990]. Ces langages ont pour avantage d'avoir une sémantique précise et vérifiable. Les notations mathématiques sont par contre peu adaptées à la formalisation de connaissances destinées à des développeurs ou à des utilisateurs de systèmes logiciels. Pour être explicite à un développeur, une modélisation doit ainsi être moins complexe que le code de l'application lui-même. Nous ne sommes ainsi pas sûr qu'une représentation formelle aide le lecteur à comprendre le pattern : « les programmeurs ont besoin d'une information concrète qu'ils peuvent comprendre et transférer directement en leur propre code, et non d'une formalisation (…). Les méthodes formelles ont leur place dans le développement logiciel – nous pensons juste qu'elles ne peuvent s'appliquer aux patterns » [BUSCHMANN & AL., 1996]. En outre, du fait que les connaissances portant sur les systèmes d'information coopératifs sont complexes et étendues, « l'adéquation, la consistance et l'exactitude des spécifications peuvent être incertaines » [ABELSON & AL., 1985]. Outre ces problèmes de complexité, ces langages formels sont également difficiles à apprendre (et la plupart des développeurs et concepteurs ne les connaissent pas). Ainsi, les notations mathématiques ne semblent pas un langage de modélisation adéquat pour notre objectif : fournir des patterns de coopération à des concepteurs ou développeurs afin qu'ils puissent (aisément) les réutiliser lors de l'ingénierie de SIC.

[68] [WINOGRAD, 1972] considère que le langage naturel est un langage de modélisation.

[69] Vienna Development Method

4.1.2. Le langage naturel (expressions textuelles)

Les langages naturels, tels que le français, « partagent tous le même avantage pour la modélisation d'être la méthode de communication disponible la plus expressive, mais aussi l'énorme inconvénient d'être la méthode de communication la plus expressive » [WINOGRAD, 1972], et ainsi d'être par nature ambiguë. Un autre avantage réside dans le fait que les langages naturels ne nécessitent pas d'effort d'apprentissage (pour une personne adulte). D'autres inconvénients concernent le fait qu'ils sont plus verbeux que les autres langages de modélisation et qu'il peut-être difficile d'imposer une structure ou d'adhérer à des conventions. De nombreux patterns ne proposent souvent qu'une représentation des connaissances sous la forme d'expressions d'un langage naturel. C'est également le cas de nombreux patterns de coopération proposés jusqu'alors (Cf. section 6).

4.1.3. Les langages semi-formels

Nous nous limitons ici aux langages semi-formels basés sur des représentations de type diagrammes, qui permettent d'imposer une structure de représentation et réduisent les ambiguïtés par l'application d'une notation stricte, et plus particulièrement au langage de modélisation UML. UML est un langage de modélisation orienté objet utilisé pour spécifier, construire, visualiser et documenter les artefacts d'un système logiciel et qui a pour principal intérêt d'être unifié - i.e. unifie les représentations sous formes de diagrammes des principales méthodes à objet : « Un des objectifs derrière l'introduction d'UML (…), est de faire d'UML un outil fournissant une base pour une méthode de développement orientée-objet commune, stable et expressive » [BOOCH & AL., 1999]. UML est même devenu aujourd'hui un standard *de facto*. Partager le même langage de modélisation entre les acteurs intervenant dans la conception des patterns et ceux intervenant dans la réutilisation des modèles développés rend leur réutilisation plus aisée. UML permet, en outre, la traduction des modèles en code et la plupart des outils de modélisation assurent même une traduction dans les deux sens, afin de pouvoir aisément modifier le code (« modifie le modèle et l'outil modifiera automatiquement le code correspondant ») ou le modèle (« modifie le code et l'outil modifiera automatiquement le modèle correspondant »). Il y a des concepts qu'UML peut exprimer qu'un langage de programmation ne peut pas et vice-versa, mais néanmoins, l'utilisation d'UML permet de réduire le temps dédié à l'implémentation et permet d'assurer la cohérence entre le modèle UML d'une part et le code qui l'implante d'autre part.

4.1.4. Approches de modélisation retenues

Notre volonté n'est pas de répondre à la question « Comment exprimer un pattern ? », mais de savoir quel degré de formalisation est nécessaire à nos patterns de coopération. L'objectif de ces patterns

étant de modéliser les connaissances du domaine de la coopération, il nous semble utile de représenter ces connaissances au sein des patterns sous forme de diagrammes UML, mais aussi d'y associer des représentations sous forme de langage naturel. Les expressions en langage naturel sont cependant structurées grâce à l'utilisation de rubriques au sein des patterns, ce qui permet de réduire certaines ambiguïtés inhérentes. Schmidt [SCHMIDT, 1995] considère que pour faciliter leur réutilisation, les patterns ne doivent pas être trop difficiles à appréhender par les concepteurs et développeurs. Ainsi, nous n'avons pas souhaité intégrer des spécifications formelles, difficiles à comprendre en général pour ce type de public, dans les patterns proposés.

Nous présentons plus précisément dans la section suivante la structure documentaire de nos patterns, structure qui allie notamment représentations textuelles (langage naturel) et diagrammes UML (langage semi-formel).

4.2. Structure documentaire des patterns de coopération et de support de coopération

La structure documentaire (Cf. Tableau 3, Tableau 4 et Tableau 5) de nos patterns est dérivée de celle élaborée par Gamma & al. [GAMMA & AL., 1995], du formalisme P-Sigma [CONTE & AL., 2001A] et de celui de Rideau [RIDEAU, 1998].

Les différentes rubriques proposées sont regroupées en trois parties : Interface, Solution et Relation. La partie « Interface » contient tous les éléments qui permettent la sélection d'un pattern et facilitent ainsi sa réutilisation. La partie « Solution » exprime la solution d'un pattern en terme de solution modèle ou de solution démarche. Enfin, la partie « Relation » permet d'organiser les relations entre les patterns et donc d'organiser le système de patterns. Comme indiqué précédemment, chaque rubrique est constituée d'un ou plusieurs champs de type diagramme UML et/ou description en langage naturel, et éventuellement de champs de type expression logique ou spécification d'interfaces IDL. La structure retenue permet ainsi d'améliorer la formalisation de l'interface des patterns, interface servant à leur sélection, mais aussi de faciliter l'organisation de notre système de patterns par l'expression de relations inter-patterns variées.
Nous explicitons plus précisément ces différentes parties et rubriques inhérentes en section suivante (§ 4.2.1) puis présentons les rubriques utilisées en fonction de chaque type de pattern conçu (§ 4.2.2).

4.2.1. Rubriques du formalisme

REMARQUE — Les rubriques du formalisme utilisées varient en fonction du type de pattern conçu ; cette variabilité est présentée dans chaque tableau.

4.2.1.1. Partie Interface

La partie *interface* assure une meilleure formalisation de l'interface de sélection des patterns et facilite ainsi la sélection d'un ou plusieurs patterns (Cf. Tableau 3). Elle permet d'identifier que le problème de l'utilisateur correspond effectivement à celui résolu par le pattern. Les patterns de coopération étant associés à un domaine d'application spécifique, cette partie utilise le vocabulaire du domaine favorisant ainsi la compréhension des problèmes couverts.

Rubrique	Description	Champs	Patterns d'analyse	Patterns d'architecture	Patterns de conception	Patterns de support technique	Patterns de support d'utilisation
Nom	Définit le couple problème/solution à partir duquel le pattern pourra être référencé.	Champ textuel	✓	✓	✓	✓	✓
Synonymes	Synonymes ou métaphores qui définissent aussi le problème.	Champ textuel	Op.	Op.	Op.	Op.	Op.
Classification	Définit le type du pattern et ainsi sa cible d'utilisation (pattern d'analyse de coopération, pattern d'architecture de coopération, pattern de conception de coopération, pattern de support technique ou pattern de support d'utilisation).	Expression logique	✓	✓	✓	✓	✓

Contexte	Décrit la pré-condition pour l'application du pattern. Cette pré-condition est obtenue en appliquant la solution d'un ou plusieurs patterns ; les noms des patterns correspondants constituent alors l'expression logique	Champ textuel et/ou expression logique	Op.	Op.	Op.	Op.	Op.
Objectif	Décrit en quelques mots quel problème le pattern adresse, reflétant ainsi ce que le pattern fait.	Champ textuel	✓	✓	✓	✓	✓
Applicabilité	Décrit les situations dans lesquelles le pattern peut être appliqué afin d'aider le concepteur ou développeur à l'utiliser.	Expression logique	✓	✓	✓	✓	✓

Tableau 3 : Rubriques de la partie « Interface »

REMARQUE — ✓: rubrique obligatoire ; Op. : rubrique optionnelle.

4.2.1.2. Partie Solution

Les rubriques de la partie *Solution* sont présentées dans la tableau ci-après (Cf. Tableau 4). Une fois identifié un certain nombre de patterns potentiellement réutilisables à partir de leur interface, les rubriques « Exemples d'utilisation » et « Conséquences » permettent de limiter le choix du réutilisateur et le forcent ainsi à exprimer et valider son choix de patterns.

Rubrique	Description	Champs	Patterns d'analyse	Patterns d'architecture	Patterns de conception	Patterns de support technique	Patterns de support d'utilisation
Modèle *ou* *Démarche*	Description de l'organisation statique du pattern Décrit la solution du problème en terme de démarche à suivre	- Champ textuel - Diagramme de classes (Op.) ou graphique divers (Op.) - Champ textuel - Diagramme d'activités (Op.) ou graphique divers (Op.)	✓	✓	✓ *ou* ✓	✓ *ou* ✓	✓
Participants	Décrit le rôle de chaque classe impliquée dans l'organisation statique du pattern (modèle).	- Champ textuel - Diagramme d'états-transitions (Op.)	✓	✓	Op.		
A un participant commun avec	Les patterns X et Y impliquent un participant commun P.	Expression logique	Op.	Op.	Op.	Op.	Op.
Implémentation	Définition de l'interface de chaque élément du modèle, selon le langage de définition d'interface OMG IDL. L'objectif est de contraindre la réutilisation effective par la manipulation de types d'éléments réutilisables conformes à ceux exposés dans cette rubrique.	Champ textuel (descriptions IDL)			Op.		

Exemples d'utilisation	Montre des exemples concrets d'utilisation du pattern afin de faciliter sa sélection ou la compréhension de sa solution.	- Champ textuel - Diagramme de classes et/ou de séquence (Op.)	✓	✓	✓	Op.	
Conséquences	Présente les forces et faiblesses de la réutilisation du pattern pour guider le concepteur dans son choix de pattern.	Champ textuel	Op.	Op.	Op.	Op.	Op.

Tableau 4 : Rubriques de la partie « Solution »

4.2.1.3. Partie Relation

La coordination entre les patterns est assurée par l'utilisation de patterns de support d'utilisation, patterns de type processus intégrant une démarche spécifiant l'utilisation et la coordination des patterns (Cf. section 2.2.2), mais aussi par la formalisation de relations inter-patterns. Les rubriques de la partie *relation* (Cf. Tableau 5) permettent ainsi d'organiser les patterns en systèmes ou langages en définissant les relations qu'entretient un pattern avec d'autres patterns. Dans un système ou langage de patterns, les patterns sont généralement liés les uns aux autres, ce qui permet de rendre le processus d'ingénierie par réutilisation beaucoup plus explicite que dans un catalogue classique. Le système ou langage constitue alors une formalisation du processus d'ingénierie.

La partie *Relation* permet d'organiser nos systèmes de patterns selon des relations clairement établies : utilisation (rubrique *utilise*), ressemblance (*ressemble*), conflit (*conflit*), raffinement (*raffine*), contrainte d'application (*requiert*). Les relations que nous proposons pour organiser les patterns de coopération et patterns de support de coopération constituent, pour certaines, une extension des relations primaires[70] proposées par [ZIMMER, 1995] ou [RIEU & AL., 1999].

[70] Relations de base offertes par la plupart des formalismes

Rubrique	Description	Champs	Patterns d'analyse	Patterns d'architecture	Patterns de conception	Patterns de support technique	Patterns de support d'utilisation
Utilise	Représente la relation suivante : le pattern X utilise le pattern Y dans sa solution.	Expression logique	Op.	Op.	Op.	Op.	Op.
Ressemble	Représente la relation suivante : le pattern X est une variante du pattern Y (les différents éléments composant le pattern X ressemblent à ceux du pattern Y).	Expression logique	Op.	Op.	Op.	Op.	Op.
Conflit	Représente la relation suivante : le pattern X a un problème similaire au pattern Y mais fournit une solution différente.	Expression logique	Op.	Op.	Op.	Op.	Op.
Raffine	Représente la relation suivante : un pattern X est un raffinement d'un pattern Y si Y peut résoudre les problèmes posés par X.	Expression logique	Op.	Op.	Op.	Op.	Op.
Requiert	Le pattern X mentionne le pattern Y dans son contexte. Cela signifie que Y sera appliqué avant X.	Expression logique	Op.	Op.	Op.	Op.	Op.

Tableau 5 : Rubriques de la partie « Relation »

4.2.2. Formalisme des différents types de patterns conçus

4.2.2.1. Formalisme des patterns d'analyse de coopération

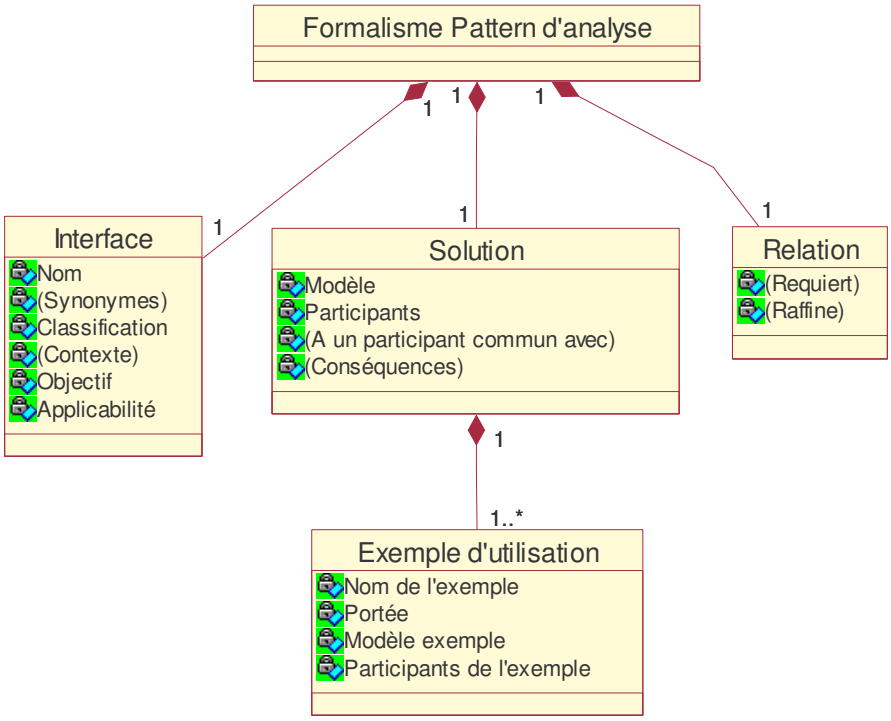


Figure 10 : Rubriques du formalisme associé aux patterns d'analyse de coopération

REMARQUE — Les attributs figurant entre parenthèses permettent de représenter les rubriques optionnelles.

Le formalisme associé aux patterns d'analyse de coopération est constitué des trois parties explicitées précédemment : *Interface*, *Solution* et *Relation*.

La partie *Interface* est composée des rubriques *Nom*, éventuellement *Synonymes*, *Classification*, *Contexte*, *Objectif* et *Applicabilité*. La rubrique *Classification* détermine le type du pattern (*classification = Pattern d'analyse de coopération*). La rubrique *Applicabilité* détermine si le pattern d'analyse permet de modéliser un SIC et éventuellement une approche de coopération (Cf. section 2.1.1) :

- "*Applicabilité = SIC*" signifie que le pattern d'analyse permet de modéliser tout type de SIC relevant d'une approche de coopération existante (fédéré, médiation de schémas, médiation de contextes) ou non (cas de la création d'une nouvelle approche).
- "*Applicabilité = Approche ^ SIC*" signifie que le pattern s'applique à la représentation des concepts inhérents à une approche de coopération (existante ou nouvelle), ainsi qu'à la représentation des concepts d'un SIC.
- Enfin, l'applicabilité peut être restreinte à un SIC d'un type particulier. Par exemple, le Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » (Cf. section 3.2.5) s'applique uniquement pour modéliser les SIC de type médiation de schémas et médiation de contextes (*applicabilité = SIC médiation de schémas ^ SIC médiation de contextes*).

La partie *Solution* est constituée des rubriques *Modèle* (un pattern d'analyse propose une solution sous la forme d'un diagramme de classes UML), *Participants* (explicitation du rôle de chaque type d'objet du modèle) et éventuellement *A un participant commun avec* et *Conséquences*. La rubrique *A un participant commun avec* permet de chaîner les participants communs aux différents patterns d'analyse. Chaque relation entre un participant et un pattern lié décrit le rôle du participant dans le pattern lié et est spécifiée de la façon suivante : *Participant : rôle [pattern lié]*. Jusqu'à présent, les différents formalismes proposés privilégiaient uniquement les relations inter-patterns. Nous proposons ainsi la création de relations entre les artefacts constituant les patterns, permettant une meilleure structuration, et ainsi une utilisation facilitée, du système de patterns.
La partie *Solution* est également constituée d'un ou plusieurs exemples d'utilisation, chacun étant décrit par :

- son nom (*Nom de l'exemple*).
- sa *portée* (exprimée sous forme d'une expression logique). L'exemple fourni est il relatif à un type de SIC spécifique (*"Portée = SIC médiation de schémas"* ou *"Portée = SIC médiation de contextes"* ou *"Portée = SIC fédéré"*) ou à une ou plusieurs approche(s) de coopération (*"Portée = Approche médiation de schémas et/ou Approche médiation de contextes et/ou Approche fédérée"*) ?
- son modèle *(Modèle exemple)*. Le modèle spécifié au sein de l'exemple constitue une adaptation du modèle de la solution.
- et les participants inhérents à ce modèle (*Participants de l'exemple*). Il s'agit de décrire ici explicitement le rôle de chaque type d'objet spécifié dans le modèle de l'exemple (SIC ou approche de coopération). Le réutilisateur du pattern devra, de la même façon, expliciter de

façon détaillée les types d'objets inhérents à son SIC ou à son approche de coopération et pourra s'aider des descriptions fournies en exemple.

Cette structuration des exemples d'utilisation a pour objectif d'améliorer l'exhaustivité de la description des connaissances liées à un SIC ou à une approche de coopération (décrit dans l'exemple) et ainsi de faciliter sa compréhension. Elle facilite également la recherche d'un exemple particulier, correspondant à un SIC existant. La description d'un SIC est ainsi mise en exergue car elle est indispensable pour pouvoir appréhender les connaissances qui lui sont associées.

La partie *Relation* des patterns d'analyse de coopération est constituée des rubriques *Raffine* et *Requiert*. Nos patterns d'analyse de coopération proposent des solutions conceptuelles suffisamment génériques et abstraites pour modéliser les concepts sous-jacents à tout (ou, en tout cas, de nombreux) SIC, mais aussi aux approches de coopération. Ainsi, à notre connaissance, ces patterns sont originaux et ne ressemblent à aucun autre pattern spécifié jusqu'alors (les relations *Conflit* et *Ressemble* sont ainsi non usitées). Les patterns proposés n'utilisant pas, non plus, d'autres patterns, la relation *Utilise* n'est pas conservée.

4.2.2.2. Formalisme des patterns d'architecture de coopération

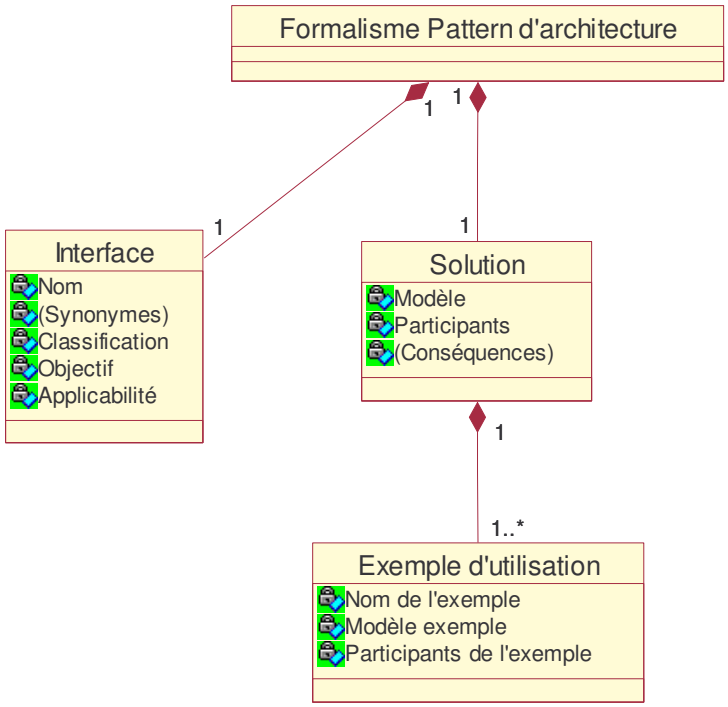


Figure 11 : Rubriques du formalisme associé aux patterns d'architecture de coopération

REMARQUE — Les attributs figurant entre parenthèses permettent de représenter les rubriques optionnelles.

Le formalisme associé aux patterns d'architecture de coopération est uniquement constitué de deux parties : *Interface* et *Solution*.

La partie *Interface* est composée des mêmes rubriques que celles du formalisme associé aux patterns d'analyse de coopération (la rubrique *Contexte* n'est cependant pas usitée). La rubrique *Classification* détermine le type du pattern (*classification = Pattern d'architecture de coopération*). La rubrique *Applicabilité* détermine si le pattern permet de représenter l'architecture d'un SIC basé sur une approche fédérée, sur une approche de type médiation de schémas ou sur une approche de type médiation de contextes (*applicabilité = SIC médiation de schémas, applicabilité = SIC médiation de contextes* ou *applicabilité = SIC fédéré)*.

La partie *Solution* est constituée des rubriques *Modèle* (un pattern d'analyse propose une solution sous la forme d'un schéma représentant un modèle d'architecture), *Participants* (explicitation des différents niveaux ou couches requis du modèle d'architecture) et éventuellement *Conséquences*. La partie *Solution* est également constituée d'un ou plusieurs exemples d'utilisation, chacun étant décrit par :

- son nom (*Nom de l'exemple*).
- son modèle *(Modèle exemple)*. Le modèle d'architecture spécifié au sein de l'exemple constitue une adaptation du modèle de la solution.
- et les participants inhérents à ce modèle (*Participants de l'exemple*). Il s'agit ici de décrire explicitement le rôle de chaque couche (ainsi que les entités à l'intérieur de ces couches) spécifiées dans le modèle de l'exemple. Le réutilisateur du pattern devra, de la même façon, expliciter de façon détaillée les entités inhérentes à l'architecture de son SIC et pourra s'aider des descriptions fournies en exemple.

Notons que la rubrique *Portée* présente dans le formalisme des patterns d'analyse de coopération n'a pas été conservée ici, car les SIC fournis en exemple dans les patterns d'architecture sont du même type que celui défini au niveau de la rubrique *Applicabilité*.

Enfin, la partie *Relation* n'a pas été usitée du fait que les patterns d'architecture que nous proposons n'ont pas, à notre connaissance, d'équivalent. En outre, bien qu'il soit fortement conseillé de les appliquer après les patterns d'analyse de coopération, ils n'utilisent ni ne requièrent l'application d'autres patterns.

4.2.2.3. Formalisme des patterns de conception de coopération

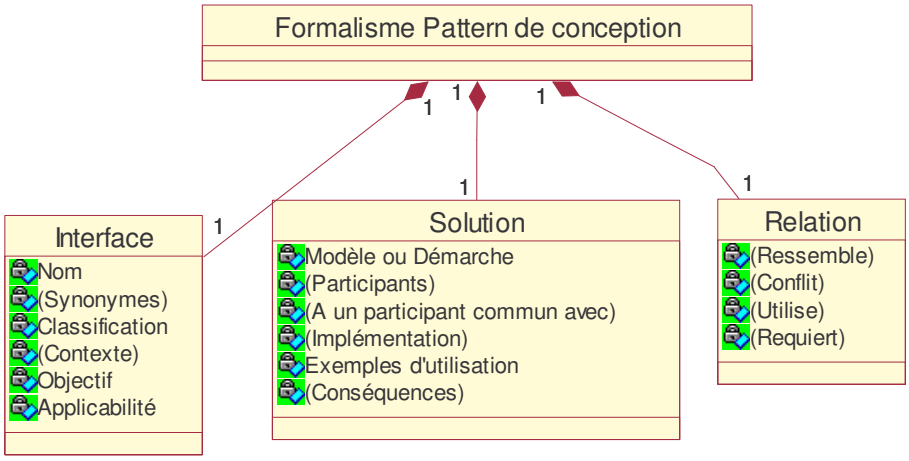


Figure 12 : Rubriques du formalisme associé aux patterns de conception de coopération

REMARQUE — Les attributs figurant entre parenthèses permettent de représenter les rubriques optionnelles.

Le formalisme associé aux patterns de conception de coopération est constitué des parties *Interface*, *Solution* et *Relation*.

La partie *Interface* est composée des mêmes rubriques que celles du formalisme associé aux patterns d'analyse de coopération. Notons que la rubrique *Classification* a pour valeur *"Pattern de conception de coopération"*. La rubrique *Applicabilité* détermine si le pattern peut être appliqué à la conception de tout SIC (*applicabilité = SIC*) ou s'il ne concerne que le SIC ACSIS (*applicabilité = SIC ACSIS*).

La partie *Solution* est constituée des rubriques :

- *Modèle* ou *Démarche* : un pattern de conception de coopération propose une solution sous la forme d'un diagramme de classes UML (Modèle) ou sous la forme d'un diagramme d'activités (Démarche).
- Eventuellement *Participants* dans le cas d'une solution modèle (explicitation du rôle de chaque type d'objet du diagramme de classes UML),
- Eventuellement *A un participant commun avec.* Cette rubrique permet de chaîner les participants communs aux différents patterns de conception.

- Eventuellement *Implémentation*. Cette rubrique permet de définir l'interface de chaque élément du modèle de classes UML, selon le langage de définition d'interface OMG IDL. Elle contraint ainsi le réutilisateur à devenir producteur de l'application et permet de respecter le caractère génératif de certains patterns de conception de coopération. Elle associe un ou plusieurs objets, sous forme de composant binaire, au pattern. Ainsi, la solution des patterns de conception, qui peut être difficile à saisir du fait de son niveau d'abstraction, se matérialise concrètement, dans notre proposition, sous forme de code. Le réutilisateur sera, en outre, obligé de respecter les interfaces définies pour réutiliser le code du pattern.
- *Exemples d'utilisation*. Nous n'avons pas jugé utile de décomposer cette rubrique en sous-rubriques comme nous l'avons fait pour les patterns d'analyse ou d'architecture de coopération. En effet, il nous a semblé inutile de décrire les participants des modèles fournis en exemples. Notons, en outre, que ces exemples ne concernent que le SIC ACSIS.
- Eventuellement *Conséquences*.

La partie *Relation* des patterns de conception de coopération est constituée des quatre rubriques suivantes : *Ressemble, Conflit, Utilise* et *Requiert*. En effet, certains de nos patterns utilisent ou requièrent l'application préalable d'autres patterns de conception. Enfin, certains patterns de conception de coopération ressemblent ou rentrent en conflit avec des patterns conçus par d'autres auteurs.

4.2.2.4. Formalisme des patterns de support technique

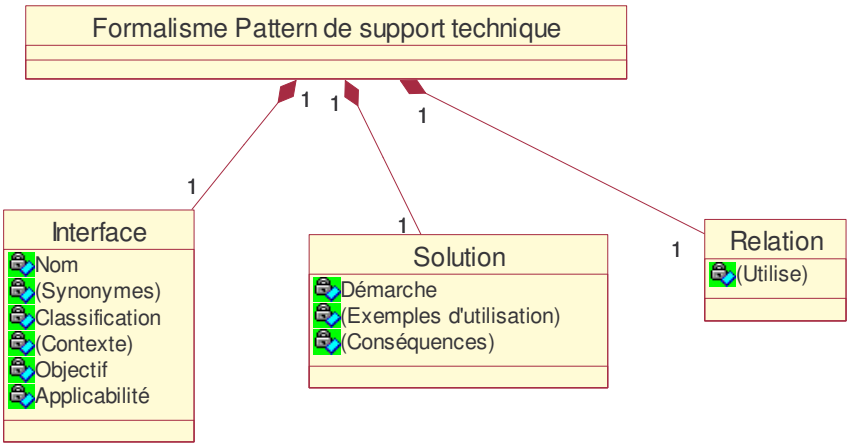


Figure 13 : Rubriques du formalisme associé aux patterns de support technique

REMARQUE — Les attributs figurant entre parenthèses permettent de représenter les rubriques optionnelles.

Le formalisme associé aux patterns de support technique est constitué des parties *Interface*, *Solution* et *Relation*.

La partie *Interface* est composée des rubriques précédemment explicitées. Notons que la rubrique *Classification* a pour valeur *"Pattern de support technique"*. Les deux patterns de support technique proposés ne concernant que le SIC ACSIS, la rubrique *Applicabilité* a pour valeur "*applicabilité = SIC ACSIS*".

La partie *Solution* est constituée des rubriques :

- *Démarche* : les deux patterns proposés spécifient une solution de type démarche.
- Eventuellement *Exemples d'utilisation*.
- Eventuellement *Conséquences*.

Les deux patterns de support technique présentés sont, à notre connaissance, originaux et ne ressemblent, ni n'entrent en conflit avec d'autres patterns. Ainsi, seule la rubrique *Utilise* est conservée au sein de la partie *Relation*.

4.2.2.5. Formalisme des patterns de support d'utilisation

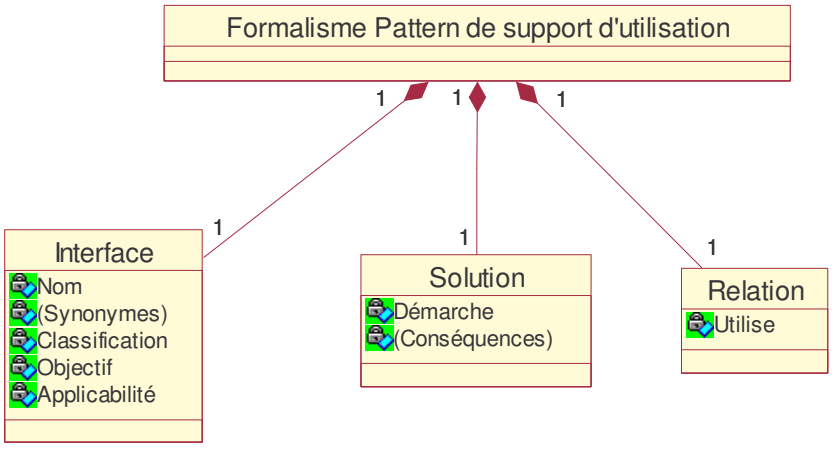


Figure 14 : Rubriques du formalisme associé aux patterns de support d'utilisation

REMARQUE — Les attributs figurant entre parenthèses permettent de représenter les rubriques optionnelles.

Le formalisme associé aux patterns de support d'utilisation est constitué des trois parties : *Interface*, *Solution* et *Relation*.

La partie *Interface* est composée des rubriques précédemment explicitées (la rubrique *Contexte* n'est cependant pas usitée). La rubrique *Classification* a pour valeur "*Pattern de support d'utilisation"*. L'*Applicabilité* détermine si le pattern s'applique à l'utilisation du système de patterns d'analyse (*applicabilité = système de patterns d'analyse*), à celle du système de patterns d'architecture (*applicabilité = système de patterns d'architecture*) ou à celle du système de patterns de conception et/ou du système de patterns de support technique (*applicabilité = système de patterns de conception* ou *applicabilité = système de patterns de conception et système de patterns de support technique*).

La partie *Solution* est constituée des rubriques *Démarche* (un pattern de support d'utilisation propose une solution sous la forme d'un diagramme d'activités UML) et, éventuellement, *Conséquences*.

La partie *Relation* est uniquement constituée de la rubrique *Utilise*. Cette rubrique permet de lister les patterns représentés dans la solution du pattern de support d'utilisation et indique ainsi les différents patterns à appliquer.

Dans la section suivante, nous envisageons le processus de validation et de réutilisation des patterns conçus (§ 5).

5. Validation et réutilisation des patterns de coopération et de support de coopération

5.1. Introduction

La recherche dans le domaine de l'ingénierie des systèmes d'information est différente des recherches dans les autres champs de l'informatique. La principale différence réside dans la présence d'un contexte social et organisationnel [MYERS, 1997] et donc d'un facteur humain. Il est, en effet, insuffisant de considérer uniquement le côté technique d'un problème sans considérer comment le problème et la solution proposée affectent le processus d'ingénierie, sans considérer comment intégrer la solution à ce problème dans la pratique de l'ingénierie des SI et, enfin, sans considérer la validation de la solution, i.e. qu'une solution technique particulière est qualifiée de bonne solution par les praticiens. La figure suivante (Cf. Figure 15) présente notre vision de l'intégration de l'ingénierie des SI dans les autres champs de l'informatique.

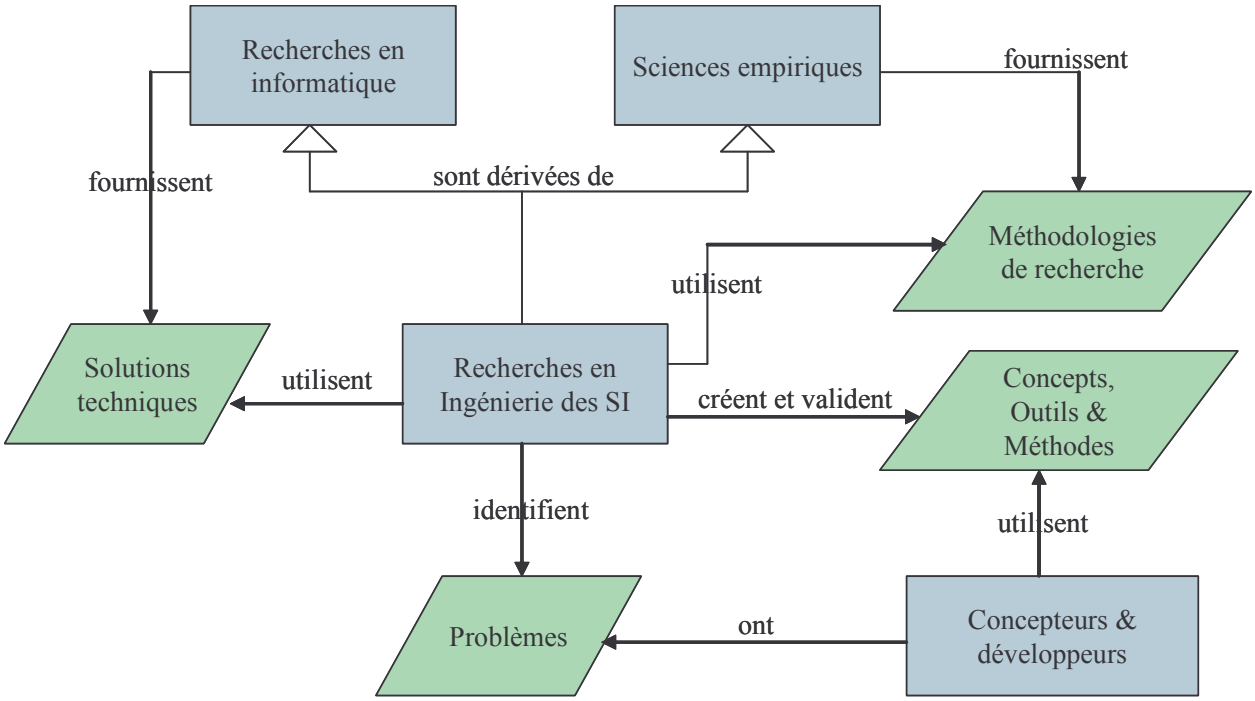


Figure 15 : Positionnement des recherches en ingénierie des systèmes d'information

Les problèmes sont identifiés en observant et analysant les pratiques des concepteurs et développeurs logiciels et le contexte d'ingénierie des SI. Les solutions à ces problèmes consistent en des outils et des techniques pour appliquer ces derniers effectivement. Pour pourvoir fournir de telles solutions, les solutions techniques provenant de la recherche en informatique peuvent être utilisées. Cependant, une preuve mathématique de l'exactitude de ces solutions n'est pas suffisante pour convaincre les concepteurs et développeurs d'adopter les solutions proposées. Ces derniers ont en effet besoin d'être convaincus qu'une solution particulière résout leur problème, n'en crée pas de nouveaux et que la solution est effectivement applicable dans leur contexte. Certes, les patterns proposent une solution à un problème dans un contexte donné et présentent souvent des exemples d'application. Cependant, il semble également intéressant d'utiliser des méthodologies de recherche telles que des études de cas ou la recherche-action pour valider ces propositions. Ces méthodologies de recherche sont plus communes dans les sciences empiriques, telles que la sociologie ou la psychologie que dans les sciences dites « dures » telles que l'informatique [CARDON, 2000].

L'application de méthodologies de recherche empirique est cependant de plus en plus populaire dans l'ingénierie des SI. Ainsi, en 1996, Basili [BASILI, 1996] faisait un plaidoyer pour l'utilisation d'études empiriques pour valider les théories et modèles conçus. Seaman [SEAMAN, 1999], lui, propose d'utiliser des études empiriques de type qualitatif afin de valider des solutions à des problèmes spécifiques et, parmi celles-ci, Darke & al. [DARKE & AL., 1998] vantent les mérites des études de cas.

5.2. Validation des patterns

Les patterns d'analyse et d'architecture de coopération, en l'absence d'un réel projet de recherche empirique aboutissant à la construction d'un nouveau système d'information coopératif, n'ont pu être totalement validés. Cependant, les exemples d'utilisation fournis permettent à la fois de valider leur utilité, mais aussi partiellement les problèmes identifiés et solutions proposées (Cf. chapitres 3 et 4).

Les patterns de conception spécifiés sont principalement issus de l'architecture ACSIS reposant sur les connaissances et les compétences de l'équipe de recherche MODEME, équipe qui travaille depuis 1991 dans le domaine de la coopération de systèmes d'information. Là encore, les exemples d'utilisation fournis permettent de valider l'utilité de ces patterns, ainsi que des solutions proposées. Il est cependant à noter que la plupart des patterns de conception spécifiés dans le domaine de la coopération de SI (Cf. section 6) n'ont pas été validés empiriquement, mais reposent en grande partie, comme dans notre cas, sur des architectures coopératives opérationnelles.

5.3. Réutilisation des patterns

L'activité d'ingénierie des SIC (développement d'un système par réutilisation), telle que nous la proposons est basée sur la réutilisation des patterns précédemment présentés. Cette activité a pour objectif la recherche, la sélection, l'adaptation et l'intégration des patterns afin de spécifier et développer le SIC. Elle consiste ainsi à mettre en place une démarche générale de développement des SIC, par réutilisation de patterns, couvrant l'ensemble des phases du processus d'ingénierie (analyse, conception, implantation) et permettant d'accélérer et de faciliter ce développement.

Afin de faciliter la réutilisation des patterns de coopération et des patterns de support de coopération conçus, et donc l'activité d'ingénierie des SIC, un environnement de représentation et de support de réutilisation des patterns est fourni et décrit en annexe (Cf. Annexe A). Cet environnement permet notamment de représenter, stocker, rechercher les patterns correspondant à un ou plusieurs problèmes dans une base spécifique et de présenter les relations entre ceux-ci.

Nos travaux ont abouti à la spécification de cinq types de patterns regroupés en deux grandes classes :

- les *patterns de coopération* regroupant les patterns d'analyse de coopération, les patterns d'architecture de coopération et les patterns de conception de coopération,

- les *patterns de support de coopération* regroupant les patterns de support technique et les patterns de support d'utilisation.

Les *patterns d'analyse de coopération* s'appliquent lors de la phase d'analyse du processus d'ingénierie de SIC ou, éventuellement, d'approche de coopération. Ils permettent d'identifier et de représenter les aspects fonctionnels du SIC ou de l'approche de coopération à concevoir, c'est-à-dire les concepts sous-jacents et les entités métier. Ces patterns ont un deuxième intérêt non lié à l'ingénierie de SIC ou d'approche de coopération : les fragments de modèles qu'ils contiennent permettent de représenter les concepts des SIC ou des approches de coopération existants et ainsi d'avoir une vision plus homogène de leurs spécificités ou points communs. Ils peuvent ainsi servir, par exemple, à comparer voire classifier les SIC existants en fonction de critères plus précis que leur seule appartenance à une approche de coopération spécifique.

Les *patterns d'architecture de coopération* s'appliquent au début de la phase de conception du processus d'ingénierie de SIC. Ils proposent des modèles d'architecture "génériques" (liés aux approches de coopération) permettant de représenter l'architecture globale - d'un point de vue structurel et organisationnel - du SIC à concevoir. Tout comme les patterns d'analyse, les patterns d'architecture sont indépendants d'un SIC spécifique.

Les *patterns de conception de coopération* sont principalement liés à une architecture particulière, l'architecture ACSIS en raison du fait que les traitements (granularité, définition) dépendent des SIC. Un pattern de conception ne peut, en effet, représenter de façon abstraite et générique les traitements associés aux entités conceptuelles de tout SIC. Tout comme la plupart des patterns de conception du domaine, nos patterns de conception de coopération sont donc issus et dépendants d'un système d'information coopératif particulier. Un pattern original permet cependant de spécifier les interactions entre les composants de tout SIC.

Les *patterns de support technique* sont des patterns génériques qui décrivent comment construire une application à partir des patterns de conception de coopération. Ces patterns constituent ainsi des structures réutilisables dédiées à l'implémentation et à l'implantation des entités spécifiées dans les patterns de conception.

Les *patterns de support d'utilisation* permettent de guider le concepteur lors de la réutilisation des différents patterns précédents et constituent un point d'entrée d'un système de patterns. Chaque pattern de support d'utilisation décrit une séquence de patterns à appliquer pour résoudre un problème spécifique.

L'application de l'ensemble de ces patterns permet de prendre en compte les différents types d'hétérogénéité inhérents à la coopération de sources d'information. Les patterns d'analyse et d'architecture permettent de spécifier un SIC particulier capable de résoudre une ou plusieurs de ces hétérogénéités. Les patterns de conception de coopération permettent principalement de concevoir le SIC ACSIS assurant la résolution des conflits structurels et sémantiques. Enfin, les patterns de support technique résolvent eux les problèmes d'hétérogénéité technique en permettant la génération de composants homogènes dépendants des technologies et langages d'implantation choisis par le concepteur du SIC.

Nous présentons dans la section suivante (§ 6), les patterns issus d'autres travaux de recherche liés à la coopération de systèmes d'information. Nous essayons à chaque fois de positionner nos propositions par rapport à ces différents travaux.

6. Travaux du domaine

Rieu & al. [RIEU & AL., 1999] considèrent « qu'il est rare aujourd'hui qu'un problème ne puisse être résolu, du moins en partie, par des patterns existants ». Certains patterns existants peuvent effectivement s'appliquer à notre domaine. Ces patterns sont soit directement dédiés à la coopération de systèmes d'information, soit relatifs aux principaux courants de recherche dont est issue la coopération, à savoir l'intelligence artificielle distribuée et les bases de données (Cf. section 3.1.1 du chapitre 1). Nous présentons d'abord des patterns spécifiés dans ces courants de recherche et montrons en quoi et comment ils peuvent s'appliquer à la coopération de systèmes d'information (§ 6.1) puis analysons les patterns de coopération existants proprement dits (§ 6.2).

6.1. Patterns issus des domaines de l'Intelligence Artificielle Distribuée et des Bases de données

6.1.1. Patterns issus de l'Intelligence Artificielle Distribuée

Ces patterns peuvent s'appliquer contextuellement en fonction des différentes technologies ou concepts de l'IAD utilisés dans les systèmes d'information coopératifs. Ainsi, si le concepteur de SIC désire intégrer une ontologie dans son SIC, il pourra, par exemple, s'aider de patterns de structuration d'ontologies. De même, s'il souhaite intégrer un système multi-agents, il pourra s'aider des patterns existants pour structurer ce SMA. Nous présentons dans les sections suivantes les patterns relatifs aux agents et systèmes multi-agents (§ 6.1.1.1) ainsi que les patterns dédiés aux ontologies (§ 6.1.1.2).

6.1.1.1. Patterns relatifs aux agents et systèmes multi-agents

De nombreux patterns ont été proposés aussi bien pour la structuration d'agents que pour la conception et la réalisation de SMA. Ces patterns sont plus particulièrement dédiés à la conception et à l'implémentation d'agents ou de systèmes multi-agents [SCHELFTHOUT & AL., 2002]. Ils s'appliquent aux agents logiciels, mobiles, réactifs ou cognitifs.

Les patterns agents ou SMA proposés se focalisent sur la représentation des connaissances des agents, sur les modèles de coopération ou collaboration, sur la communication entre les agents ou sur l'infrastructure sous-jacente [SILVA & DELGADO, 1998]. Ainsi, ils ne définissent pas d'agents spécifiques à la coopération de sources d'information, mais sont notamment particulièrement intéressants pour spécifier les collaborations entre agents ou les architectures d'agents (ou *systèmes de support d'agents* [SILVA & DELGADO, 1998]) quel que soit leur type. Nos patterns de conception de coopération utilisent d'ailleurs certains patterns de conception agent (Cf. chapitre 4). Nous présentons les principaux travaux sur les patterns agents dans les sections suivantes (§ a) à i)).

a) *Hung & Pasquale : Agent Usage Patterns*

Les patterns d'usage agents (*agent usage patterns*) proposés par Hung & Pasquale [HUNG & PASQUALE, 1999] ont pour objectif de faciliter le développement d'applications basées sur des agents. Les patterns proposés sont ainsi présentés comme des « modèles basiques pour les programmes basés sur les agents ». Ils ont été conçus afin de permettre la construction de programmes, fiables sûrs et performants, basés sur des agents. Ils mettent notamment l'accent sur l'infrastructure d'exécution de l'agent, ainsi que sur la communication entre l'agent et son environnement.

b) *Deugo & al. : Patterns as a Means for Intelligent Software Engineering*

Deugo & al. [DEUGO & AL., 1999A] décrivent les problèmes du développement d'applications basées sur les agents et propose cinq patterns de conception permettant de faciliter ce développement. D'après les auteurs, les problèmes liés au développement d'applications basées sur les agents sont : le manque d'accord sur les définitions, des efforts dupliqués, une incapacité à satisfaire les exigences de normalisation des industriels, une difficulté à identifier et spécifier les abstractions communes au-delà des agents, un manque de vocabulaire commun, la complexité du paradigme et du développement d'applications basées sur les agents, la rareté des modèles et l'absence de méthodes orientés agents. Les patterns qu'ils proposent ont pour objectif de réduire ces problèmes ou manques et de faciliter le développement d'applications orientées agents et plus particulièrement de spécifier des techniques de communication entre agents performantes :

- Le pattern « *Direct coupling* » assure une communication directe et à couplage faible entre des agents distants, et, ce, même lors de leurs déplacements (utilisation de mécanismes d'enregistrement/notification de la nouvelle adresse de l'agent),
- Le pattern « *Proxy agent* » s'applique si la communication directe entre deux agents n'est pas possible pour des raisons, par exemple, de performance (Cf. pattern précédent). Il permet la communication entre deux agents grâce à l'utilisation d'un agent proxy créé à l'endroit ou se trouve l'agent avant que celui-ci ne change de position.
- Le pattern « *Communication Sessions* » permet de gérer des communications multiples entre agents, d'une façon simple et efficace, grâce à la notion de session.
- Le pattern « *Badges* » gère l'identification des rôles des agents par des marqueurs. Il permet ainsi d'envoyer un message en fonction du badge porté et donc du rôle.
- Le pattern « *Event Dispatcher* » spécifie un mécanisme de communication événementielle de type publication/abonnement entre les agents, basé sur la présence d'un objet intermédiaire servant de diffuseur de messages.

c) *Silva & Delgado : The Agent Pattern*

Silva & Delgado [SILVA & DELGADO, 1998] proposent de voir l'ensemble du paradigme agent comme un unique pattern de conception, dénommé « *Agent Pattern* ». Ce pattern a pour objectif de faciliter le développement d'applications basées sur des agents et de permettre à ces applications d'avoir les caractéristiques suivantes : autonomie, hétérogénéité, ouverture, dynamicité, robustesse, sécurité. Un agent y est défini comme un objet actif[71], autonome, social, réactif et mobile. Ce

[71] Un objet actif (ou acteur) est un objet ayant son propre fil d'exécution et un comportement propre. Il repose sur les notions de parallélisme et d'asynchronisme. Le parallélisme permet aux objets actifs de continuer leurs calculs indépendamment les uns des autres. L'asynchronisme permet de laisser aux acteurs le « choix » du moment où un message qui leur est adressé est effectivement traité. Remarquons que cette notion est parfois définie de manière similaire à celle d'agent logiciel (cf. section 3.1.3.3 g) du chapitre 1).

pattern est orienté vers la spécification d'une infrastructure d'application et prend en compte les problèmes de synchronisation, de gestion des processus, de communication, de gestion de la sécurité ou de persistance. Cet unique pattern propose une intégration et une vision « agentifiée » de patterns de conception orientés objet existants : les patterns Proxy, Strategy et Factory Method de Gamma & al. [GAMMA & AL., 1995] et Active Object de Lavender & Schmidt [LAVENDER & SCHMIDT, 1995].

d) Aridor & Lange : Agent Design Patterns

Aridor & Lange [ARIDOR & LANGE, 1998] proposent un catalogue de patterns[72] de conception orientés agents mobiles classés en trois catégories : mobilité (*traveling*), tâche (*task*) et interaction (*interaction*). Ces patterns fournissent des solutions à des problèmes pragmatiques qui peuvent survenir lors de la conception et l'implémentation d'applications basées sur des agents mobiles. Certains de ces patterns sont cependant utilisables pour la conception d'applications à base d'agents cognitifs ou réactifs.

Les patterns dédiés à la mobilité spécifient les concepts nécessaires pour gérer la mobilité des agents (routage, qualité de service,…) :

- *Itinerary* : spécifie les itinéraires et le routage des agents. Ce pattern gère une liste de destinations, définit un plan de routage et gère des cas spéciaux (absence de site de destination,…)
- *Forwarding* : fournit un moyen de déplacer automatiquement des agents nouvellement arrivés sur un autre site.
- *Ticket* : spécifie des adresses de destination et encapsule la qualité de service (délai,…) et les permissions nécessaires pour acheminer un agent vers un autre site et l'exécuter à distance.

Les patterns de tâches concernent la division et l'ordonnancement des tâches allouées à un ou plusieurs agents :

- *Master/Slave* : permet la délégation de tâches. Un agent maître peut déléguer une tâche à un ou plusieurs agents esclaves.
- *Plan* : ce pattern permet de coordonner plusieurs tâches exécutées, en parallèle ou en séquence, par différents agents (concept de workflow). Ce pattern favorise la réutilisabilité des tâches, leur composition et leur allocation dynamique aux agents.

Les patterns d'interaction permettent de localiser les agents et de faciliter leurs interactions :

[72] La plupart des patterns proposés par ces auteurs ne sont décrits qu'en une ou deux phrases…

- *Meeting* : permet à plus de deux agents d'initier une interaction. Ce pattern spécifie notamment la gestion de la synchronisation spatiale et temporelle nécessaire à une interaction locale.
- *Locker* : permet à un agent de stocker temporairement ses données dans un endroit sûr et privé, le « placard ». Cela peut lui permettre d'abandonner un peu de "poids" lors de ses déplacements, réduisant ainsi le trafic réseau et améliorant l'intégrité des données.
- *Messenger* : spécifie un agent nommé « agent messager » en charge de transporter les messages intervenant entre deux agents situés sur des sites distants.
- *Facilitator* : décrit un service de nommage et de localisation destiné aux agents. Un annuaire de noms symboliques est spécifié afin de faciliter les communications entre les agents sans connaître leur emplacement.
- *Organized Group* : permet de composer plusieurs agents dans un groupe. Tous les membres de ce groupe peuvent ainsi se déplacer ensemble.

e) *Kendall & al : The Layered Agent Patterns*

Kendall & al. [KENDALL & AL., 1996] présentent une collection de patterns constituant une amorce d'un langage de patterns dédié à la conception orientée agents. Les auteurs décrivent clairement le contexte de leur travail : ils utilisent un seul modèle d'agent reposant sur les caractéristiques de complexité, d'intelligence, de cognition et de mobilité. Un pattern d'architecture général, nommé « *Layered Agent* » permet d'organiser et de structurer le comportement des agents. Il propose de décomposer l'agent en sept couches reflétant ses différentes dimensions et dont les communications bidirectionnelles inter-couches permettent de traiter les messages provenant d'autres agents ou d'en initier :

- mobilité (couche recevant les messages délivrés par les autres agents ou permettant de transporter l'agent sur d'autres sites),
- communication (couche traduisant le message reçu dans le dialecte local de l'agent ou formulant un message destiné à un autre agent),
- collaboration (détermine si un message entrant doit être traité ou vérifie le message sortant),
- action (sélectionne les actions appropriées faisant suite aux requêtes entrantes acceptées ou enregistre les intentions assumées par l'agent),
- raisonnement (l'agent détermine ce qu'il doit faire en fonction de ses objectifs, plans, capacités et modèles),
- croyances (modèle de représentation interne de l'agent et de son environnement),

- sensibilité (les sens changent en fonction de l'environnement de l'agent et notamment des messages reçus).

Pour chacune des couches, différents patterns sont spécifiés : *Active Object* (décrit comment gérer les problèmes de synchronisation et de concurrence d'accès entre les agents), *Adaptor* (gère les interactions entre les agents et des objets externes), *Mediator* (permet la collaboration entre des agents sans connaissance à priori de leur existence), *Negociator* (mise en œuvre d'une collaboration directe entre agents), *Broker* (permet à des agents d'accéder à d'autres agents ou à des ressources situés sur des sites distants), *Reasoner* (permet à un agent de choisir des réactions ou réponses appropriées en fonction d'un stimulus). Le bénéfice majeur de cette proposition réside dans le fait qu'elle propose un modèle conceptuel permettant de structurer des systèmes basés sur les agents et que les patterns présentés permettent de construire une architecture pour les systèmes multi-agents.

f) *Deugo & al. : Agent Patterns*

Les patterns proposés par Deugo & al. [DEUGO & AL., 1999B] sont classés en quatre catégories : architecture, communication, mobilité et coordination.

Les patterns architecturaux :

- *Layered Agent* (Cf. point e)).
- *Mobile Agent* : ce pattern permet d'encapsuler les aspects de sécurité et de distribution des agents. Il spécifie un agent *proxy*, un gestionnaire de sécurité et un agent générique responsable de la réalisation de la mobilité.
- *Resource Manager* : permet de minimiser les dépendances entre un agent et ses ressources en spécifiant un gestionnaire de ressources chargés d'encapsuler et de contrôler les ressources, ou une partie de celles-ci.
- *Jurisdiction* : facilite le partage de ressources en spécifiant des agents de haut niveau chargés de gérer les droits d'accès aux ressources.

Les patterns de communication :

- *Meeting* : les agents se retrouvent sur un site spécial pour communiquer plus facilement.
- *Ambassador* : spécifie le concept d'agent « ambassadeur ». Un agent ambassadeur est envoyé par un autre agent sur un site distant afin de se tenir informé des événements survenant sur ce site.
- *Proxy* : idem point b).

- *Finder* : Ce pattern spécifie un agent « découvreur » fixe permettant de localiser les agents mobiles (enregistrement du nom symbolique et de la localisation de chaque agent). Il permet ainsi de retrouver un agent en utilisant son nom symbolique.
- *Badges* : idem point b).
- *Whiteboard*[73] : spécifie une aire sur chaque site où les agents peuvent poster et lire des messages. Ce pattern permet à un agent d'échanger des données spécifiques avec d'autres agents.
- *Translator* : spécifie un agent traducteur permettant à deux agents avec des dialectes différents de communiquer (traduction des messages entre les agents).

Les patterns de mobilité (Travelling) :

- *Itinerary* : spécifie le concept d'itinéraire, gérant une liste de destinations possibles et les plans de déplacement associés, d'un agent mobile. Un itinéraire permet également de gérer les exceptions (site distant non accessible,…).
- *Ticket* : spécifie le concept de « ticket ». L'agent envoie un ticket sur un site distant avant de réellement se déplacer pour s'assurer de la disponibilité de ce site.
- *Router* : permet à un agent de sélectionner la destination la plus appropriée pour réaliser une tâche. Il s'agit d'ajouter des propriétés à l'agent (qualité de service,…), propriétés qui seront par la suite utilisées par un routeur pour le diriger vers la destination la plus appropriée.
- *Relocator* : permet de gérer les liens existants entre un agent mobile et d'autres agents, ou ses ressources, pour qu'ils subsistent après un déplacement (réplication des ressources,…).

Les patterns de coordination :

- *Master-Slave* : idem point d).
- *Marketplace* : spécifie un système de passation de contrat entre des agents fournisseurs de ressources et des agents consommateurs (contract-net protocol [SMITH, 1980]). Un agent consommateur recherche des ressources respectant certaines propriétés. Les agents fournissant des ressources compatibles avec ces propriétés font une offre. L'agent consommateur évalue ces offres et retient celle remplissant le mieux ces propriétés.

[73] Le tableau blanc (whiteboard) [BOITET & SELIGMAN, 1994] est une architecture dérivée du tableau noir (blackboard) de Erman & Lesser [ERMAN & LESSER, 1980]. La principale différence réside dans le fait que les composants n'accèdent pas directement au tableau et n'ont ainsi pas besoin de connaître son existence. L'idée de base est similaire à celle de la communication par forum modéré [ANDREOLI & PARESCHI, 1991] : tout passe par un coordinateur en charge de la centralisation des flux de communication.

- *Specialist* : permet de coordonner les actions collectives d'agents spécialisés en utilisant un tableau noir.
- *Negotiating Agents* : permet de détecter et résoudre les actions conflictuelles entre agents par l'introduction d'une négociation entre ceux-ci. La négociation passe par une phase préalable d'échange des intentions de chaque agent.
- *Subsumption* : permet de composer un agent à partir de comportements spécifiés indépendamment et pouvant se révéler conflictuels. Le comportement adéquat d'un agent est sélectionné par un arbitre en fonction de priorités.

g) Meira & al. : Agent Patterns for a More Expressive Approach

Meira & al. [MEIRA & AL., 2001] présentent un ensemble de patterns de conception agent, dont seuls quelques uns sont réellement nouveaux. Ces patterns sont dédiés à la spécification d'une hiérarchie spécialisée d'agents (mobiles ou cognitifs) ou de mécanismes de communication inter-agents. Ils proposent des solutions à des problèmes tels que : comment les agents communiquent avec d'autres agents sans savoir quels agents sont disponibles, ni comment les contacter ; comment contacter ou maintenir le contact avec des agents pouvant être mobiles.

- Le pattern « *Receptionist* » définit un agent spécifique servant de pages jaunes et référençant les services que les différents agents d'un système peuvent fournir. Il reprend le pattern *Facilitator* d'Aridor & Lange [ARIDOR & LANGE, 1998] (Cf. point d)).
- Le pattern « *Secretary* » spécifie un agent « secrétaire », dont le rôle est de conserver en permanence la localisation d'un agent. Les messages entre les agents transitent ainsi par l'agent « secrétaire », ce dernier ayant pour tâche de transmettre le message à l'agent destinataire. Ce pattern permet ainsi de contacter et maintenir le contact avec des agents pouvant migrer. Ce pattern reprend le pattern *Proxy* de Deugo & al. [DEUGO & AL., 1999A] (Cf. point b)).
- Le pattern « *Session* » introduit le concept de « session ». Une session est une relation de communication ouverte entre deux agents et représentée par un objet session dans chaque agent. Ce pattern assure une communication directe et simple entre agents ce qui assure une meilleure performance du système. Ce pattern reprend le pattern *Communication Sessions* de Deugo & al. [DEUGO & AL., 1999A] (Cf. point b)).
- Le pattern « *Mobile Session* » introduit les concepts de session mobile et d'antenne (Cf. pattern suivant). La session mobile est un lien de communication entre deux agents, représenté par un objet « session mobile » dans chaque agent et où les messages transitent par l'antenne en charge de transmettre les messages. Ce pattern permet ainsi à un agent de

continuer à converser tout en se déplaçant. Il est inspiré du pattern *Direct coupling* de Deugo & al. [DEUGO & AL., 1999A] (Cf. point b)).

- Le pattern « *Antenna* » est couplé au pattern précédent et permet, grâce à la présence d'un agent spécifique « Antenne » sur chaque site (en général, chaque serveur d'agent), de conserver le lien de communication entre les agents.
- Le pattern « *Private Session* » permet de rendre la conversation sûre en la rendant privée (utilisation d'un objet « Session privée »).
- Le pattern « *Meeting with Moderator* » assure une communication sur le même sujet entre plusieurs agents en utilisant un intermédiaire commun. A chaque fois qu'une conversation sur un sujet spécifique est nécessaire, un agent « Meeting with moderator » est invoqué et a pour fonction d'initier une session, de synchroniser les messages entre les agents puis de terminer les sessions de communication. Il est inspiré du pattern *Meeting* de Aridor & Lange [ARIDOR & LANGE, 1998] (Cf. point d)).

h) *Lind : Patterns in Agent-Oriented Software Engineering*

Lind [LIND, 2002] présente principalement, dans cet article, la structure d'un catalogue de patterns orientés-agents, ainsi qu'un formalisme de description de pattern. Il applique le formalisme spécifié à deux exemples de patterns : « InteRRap architecture » et « Contract-net protocol ». Le premier pattern présenté fait partie des patterns d'architecture d'agents ; le second, des patterns de protocole d'interaction.

Le pattern « *InteRRap architecture* »[74] définit une infrastructure d'exécution basée sur trois couches :

- la couche basée sur le comportement (*Behavior Based Layer*) implémente le comportement réactif de l'agent (suite à une demande externe),
- la couche de planification locale (*Local Planning Layer*) réalise le processus de planification d'un agent individuel et est responsable du contrôle des plans d'exécution des agents,
- la couche de planification sociale (*Social Planning Layer*) est responsable de la coordination avec les autres agents du système (utilisation de protocoles de négociation).

Le pattern « *Contract-net protocol* » décrit, sous la forme d'un pattern, le protocole de passation de contrats entre agents défini par Smith [SMITH, 1980] et est sensiblement analogue au pattern Marketplace de Deugo & al. [DEUGO & AL., 1999B] (Cf. point f). Ce pattern permet d'allouer une tâche spécifique à un des contractants potentiels afin de minimiser les coûts locaux. L'idée est que les participants calculent leur coût local afin de réaliser cette tâche puis annoncent le coût résultant à

[74] L'architecture InteRRaP a été spécifiée par [MULLER, 1997].

un gestionnaire qui décide de l'allocation finale de la tâche à un agent. Cela assure une allocation de ressources optimale dans le cas de tâches indépendantes.

i) *Aarsten & al. : Patterns for cooperation*

Aarsten & al. [AARSTEN & AL., 1996] proposent trois patterns orientés agent :

- *Compatible heterogeneous agents* : permet la collaboration entre agents hétérogènes grâce à la négociation d'interface (recherche des méthodes fournies par l'autre agent).
- *Visibility and Communication between agents*. Ce pattern rend visibles les ressources fournies par les agents : chaque agent annonce les ressources qu'il fournit à son arrivée dans le système.
- *Leader/Collaborator/Collaboration* : définit comment gérer une collaboration concurrente (partage de ressources et de capacités de traitement) entre plusieurs agents, sur le modèle Client–Serveur–Service.

Ces patterns, tout en restant d'assez haut niveau, proposent une mise en oeuvre orientée objet.

6.1.1.2. Patterns dédiés aux ontologies

Différents patterns principalement dédiés à la structuration d'ontologies ou servant à la traduction entre langages de représentation d'ontologies ont été proposés. Nous présentons trois propositions ayant conçu de tels patterns[75]. Ces derniers peuvent notamment s'appliquer à la conception d'ontologies dédiées au domaine de la coopération de systèmes d'information.

a) *Reich : Ontological Design Patterns*

Les six *patterns de conception ontologique* (*Ontological Design Patterns- ODP*), spécifiés par Reich [REICH, 1999], sont dédiés à la structuration d'ontologies et permettent d'abstraire et d'identifier les structures de conception ontologique, les termes, les expressions et les contextes sémantiques. Ils permettent de séparer la construction et la définition d'expressions complexes de leur représentation afin de pouvoir les modifier indépendamment. Cette méthode a été appliquée, par l'auteur, dans l'intégration d'informations de biologie moléculaire. Ces six patterns sont :

- *Definition-encapsulation ODP* : ce pattern permet de gérer les groupes terminologiques d'une ontologie pouvant avoir des significations différentes (en raison de définitions différentes de termes identiques). Il spécifie des assertions conditionnelles permettant de sélectionner les définitions et significations adéquates.

[75] Certains travaux utilisent le concept de pattern de façon parfois hasardeuse (le triplet <problème, contexte, solution> est parfois très difficile à cerner). Nous ne présentons pas ici les patterns dédiés à la formalisation des connaissances (par exemple, ceux de Puppe [PUPPE, 2000]), dont l'objectif est beaucoup plus étendu que celui de la formalisation des connaissances dans une ontologie.

- *Expression-composer ODP* : ce pattern sépare la construction d'un terme ou d'une expression complexe de sa représentation de façon que le même processus de construction puisse créer différentes représentations.
- *Chain-of-expressions ODP* : ce pattern permet de découpler les contextes et les termes en donnant à des expressions multiples la possibilité de remplir ce contexte. Les expressions forment une chaîne, et le contexte est passé entre les différentes expressions jusqu'à ce qu'il soit rempli.
- *Terminological-Hierarchy ODP* : ce pattern permet la composition récursive de composants (groupes terminologiques) pour former des composants plus larges et décrit comment utiliser cette composition.
- *Update-Dependencies ODP* : ce pattern permet de maintenir la cohérence entre des termes liés lors du partitionnement d'une ontologie en une collection de groupes terminologiques collaborant.
- *Terminology-organizer ODP* : ce pattern définit une interface pour créer et organiser les termes ou expressions.

b) Staab & al. : Engineering Ontologies using Semantic patterns

Staab & al. [STAAB & AL., 2001] font le constat qu'un grand nombre de langages de représentation (OIL[76], F-Logic,…) - et donc de modèles formels - des ontologies ont été proposés. La transformation directe d'un langage de représentation vers un autre n'est pas non plus considérée, par les auteurs, comme viable. Ils proposent ainsi le concept de pattern sémantique assurant la transformation et la réutilisation de la sémantique vers différents langages cibles permettant de faire un lien entre les différentes représentations et les différentes façons de modéliser la connaissance. Les patterns sémantiques permettent de modéliser et de décrire les primitives sémantiques à un méta-niveau (haut niveau d'abstraction), d'instancier la description du méta-niveau et de compiler la représentation finale en un langage de représentation particulier (OIL, F-Logic, logiques de description…).

c) Savonnet & al. : Patterns d'architecture d'ontologies

Savonnet & al. [SAVONNET & AL., 2002] ont pour objectif de fournir aux concepteurs d'ontologies un guide, spécifié sous forme de patterns, permettant de produire des ontologies de meilleure qualité. D'après les auteurs, ces patterns prennent en compte trois dimensions : la nature de la connaissance, la structuration de la connaissance dans l'ontologie et le lien entre l'ontologie et les systèmes

[76] Ontology Inference Layer.

d'information locaux. Quatre "ébauches" de patterns d'ontologie[77], appréhendant ces différentes dimensions, ont été proposées.

6.1.2. Patterns dédiés au domaine des bases de données

Les patterns conçus dans le cadre des bases de données sont des patterns peu voire non applicables au domaine de la coopération de systèmes d'information. Ils appréhendent, en effet, les problèmes et solutions liés à l'accès aux bases de données relationnelles [KELLER & COLDEWEY, 1998], à la gestion de la persistance d'objets dans les bases de données relationnelles [BROWN & WHITENACK, 1996] [LEA, 1999] [PANG, 1999], aux bases de données actives [FRONT, 1997],…

Nous présentons dans la section suivante les patterns dédiés à la coopération de sources d'information (§ 6.2).

6.2. Patterns de coopération de systèmes d'information

Nous présentons dans cette section les principaux travaux liés à la conception de patterns dédiés à la coopération de systèmes d'information. Tout comme les SIC, ces travaux, peu nombreux et récents, peuvent être classés selon deux approches :

- l'approche *interopérabilité de procédés et de processus* (patterns d'interopérabilité de tâches, de procédés, de processus,…),
- l'approche *interopérabilité de données* (patterns de coopération de sources d'information, d'intégration d'information).

Les patterns que nous proposons, étant dédiés à la coopération de sources d'information (Cf. section 3.1.1 du chapitre 1), s'inscrivent dans la seconde approche. Trois travaux de recherche s'inscrivent également dans ce courant : les travaux de l'équipe SIGMA [SAIDANE & GIRAUDIN, 2002], les travaux de [FERNANDEZ & AL., 2003] et, enfin, ceux de [GANGULY & AL., 2001]. Nous présentons également les patterns de coopération conçus dans le cadre du projet ECOO[78], bien que sémantiquement très différents de nos patterns car faisant partie de la première approche (interopérabilité de procédés), en raison de l'intérêt de l'approche retenue partiellement similaire à celle que nous proposons. Remarquons que ces quatre projets spécifient principalement des patterns de conception dédiés à la coopération de SI. Il est à noter, enfin, que la plupart des propositions sont en cours de développement et donc les solutions souvent partielles.

[77] La description des patterns en trois ou quatre lignes ne permet pas de se faire une idée précise de leur contenu, ni réellement du problème résolu.

[78] Environnement pour la COOpération

6.2.1. Patterns dédiés à l'interopérabilité des données

Trois projets relatifs à la conception de patterns dédiés à la coopération de sources d'information ont été menés. Le premier a permis de créer un catalogue de patterns dédiés à la conception d'une architecture fédérée (§ 6.2.1.1) ; le deuxième a permis de concevoir un pattern dédié à la résolution des conflits sémantiques des sources d'information (§ 6.2.1.2) ; et le troisième a abouti à la spécification de patterns de conception génériques (§ 6.2.1.3).

6.2.1.1. Patterns de conception d'une architecture fédérée de Fernandez & al.

Fernandez & al. [FERNANDEZ & ZHAO, 2000] [FERNANDEZ & AL., 2003] définissent un langage de patterns architecturaux dédié à la spécification et à la conception d'une architecture fédérée facilitant l'échange de données entre systèmes. Trois patterns ont ainsi été proposés :

- Le pattern « *Federation* » décrit la structure globale de l'architecture fédérée (basée sur l'utilisation d'un système multi-agents).
- Le pattern « *Dependency separation* », utilisé par le pattern précédent, permet de résoudre les dépendances entre applications en regroupant ces applications dans des domaines sémantiques, limitant les échanges d'information inter-domaines.
- Le pattern « *Interface connection* » connecte les domaines d'applications afin d'assurer l'échange et l'intégration de l'information. Ce pattern permet de spécifier les interactions entre les composants de l'architecture fédérée.

REMARQUES & POSITIONNEMENT – *Bien que qualifiés de « design patterns », les patterns proposés décrivent principalement la fédération du point de vue des besoins de coopération entre sources d'information ou de sa structure. En outre, ils sont principalement composés de descriptions textuelles, sans aucun modèle conceptuel fourni, ce qui les rend difficilement réutilisables. Enfin, à l'instar de nos patterns de conception, ils sont spécifiques à une architecture développée antérieurement [FERNANDEZ & WIJEGUNARATNE, 1996] [WIJEGUNARATNE & AL., 2000]. Cependant ils n'adressent en grande partie que les problèmes d'hétérogénéité sémantique des sources d'information et n'envisagent pas l'architecture fédérée du point de vue de son implémentation. D'après les auteurs, ce dernier point est un travail en cours de réalisation.*

6.2.1.2. Pattern d'interopérabilité sémantique de Ganguly & al.

Ganguly & al. [GANGULY & AL., 2001] proposent un pattern de conception nommé « *Semantic Interpreter Pattern* » et dédié à la résolution des conflits sémantiques entre bases de données. Ce

pattern spécifie et décrit succinctement les principaux composants d'une architecture nécessaires à la résolution des problèmes d'hétérogénéité sémantique :

- une ontologie de domaine décrivant les concepts et termes du domaine et leurs relations,
- un système multi-agents composé de cinq types d'agents :
  - ⇨ un *agent de coopération* (« *cooperation agent* ») est chargé de gérer les informations sur une base de données locale ou sur un utilisateur du système coopératif. Chaque agent de coopération associé à une base de données est en charge du traitement des requêtes locales.
  - ⇨ un *agent routeur sémantique* (« *semantic router agent* ») stocke les "contrats ontologiques" (« ontological agreements »), i.e. les liens entre les termes de l'ontologie et les bases de données locales.
  - ⇨ un *agent d'ontologie* (« *ontology agent* ») gère l'accès à l'ontologie de domaine et répond aux requêtes de l'agent routeur sémantique sur les liens entre les concepts de l'ontologie afin que ce dernier puisse déterminer quelle base de données est impliquée dans une requête.
  - ⇨ un *agent de traitement* (« *processing agent* ») qui à partir d'une requête globale interroge l'agent routeur sémantique pour déterminer quelle(s) base(s) de données est (sont) impliquée(s) dans la requête. A partir de cette information, cet agent établit des plans d'exécution à destination des agents de coopération concernés par cette requête.
  - ⇨ un *agent d'interface* (« *interface agent* ») constituant l'interface entre l'utilisateur et l'agent de coopération en charge de la gestion des informations concernant cet utilisateur.

REMARQUES & POSITIONNEMENT – *Le pattern de conception proposé par Ganguly & al. a été conçu à partir d'une architecture coopérative décrite dans [GANGULY & AL., 2000]. Ce pattern, cependant proposé « seul », ne permet pas en lui-même de concevoir un système d'information coopératif opérationnel pour les raisons suivantes :*

- *Le pattern s'intéresse à l'aspect sémantique de la coopération, sans prendre en compte les problèmes d'hétérogénéité technique et structurelle des bases de données. Il ne permet pas d'apporter, par exemple, une réponse à la question suivante : « Comment l'architecture homogénéise-t-elle les modèles de données ? ».*
- *Le pattern ne décrit pas les mécanismes internes (protocoles,…) de traitement de la requête globale (restructuration des résultats, transformation d'une requête globale en requêtes*

locales,…), ni la résolution inhérente des différents types de conflits sémantiques (conflits de nommage, de valeur, d'échelle,…),

- *Le pattern ne décrit pas non plus la structuration de l'ontologie, or c'est principalement celle-ci qui permet de résoudre les conflits sémantiques.*

Enfin, la structure de ce pattern ne permet pas la génération de composants dédiés à la coopération de systèmes d'information.

6.2.1.3. Les patterns de coopération de l'équipe SIGMA

Saidane & al. [SAIDANE & GIRAUDIN, 2002] proposent un catalogue de cinq patterns destiné à modéliser les systèmes d'information coopératifs. Ce catalogue est dérivé des patterns proposés par Gamma & al. [GAMMA & AL., 1995]. Il combine quatre patterns de conception et un pattern de type démarche dont l'objectif est de guider le concepteur vers le pattern de conception approprié à la résolution de son problème, en fonction des critères de coopération suivants : transparence / non transparence des données, recouvrement / non recouvrement des données, communication unidirectionnelle / bidirectionnelle, mode d'accès des systèmes composant le SI coopératif.
Les quatre patterns de conception proposés sont :

- Le pattern « *Façade SICO* », qui permet de fournir une interface unifiée de chaque système composant le SI coopératif.
- Le pattern « *Médiateur SICO* » formalise la coopération de deux systèmes composants transparents au concepteur et qui communiquent en mode bidirectionnel. Ces systèmes composants sont perçus au travers de leur façade. Ce pattern nécessite ainsi l'application du pattern précédent.
- Les deux autres patterns, juste mentionnés dans [SAIDANE & GIRAUDIN, 2002], concernent d'autres situations « de type mise à jour automatique (pattern *Observateur SICO*) et une forte indépendance entre les systèmes (pattern *Chaîne de responsabilités SICO*) ».

REMARQUES & POSITIONNEMENT – *Les patterns de conception proposés par l'équipe SIGMA sont génériques et s'appliquent à la fois à l'interopérabilité de processus et à l'interopérabilité de sources d'information. De ce fait, les patterns proposés sont très abstraits et difficilement utilisables par un concepteur de SIC dédiés à la coopération de sources d'information, car requérant de nombreuses adaptations pour être opérants. En outre, ils ne prennent en compte, ni n'utilisent les connaissances typiques du domaine de la coopération de sources d'information (problèmes d'hétérogénéité structurelle et sémantique,…). Non génératifs, ils ne fournissent pas non plus un cadre d'aide (sous forme de composants) à la création de systèmes coopératifs. Notons, enfin, que le*

pattern de démarche proposé est similaire, dans notre typologie, à un pattern de support d'utilisation.

Nous présentons dans la section suivante les patterns de coopération issus du projet ECOO et dédiés à l'interopérabilité de procédés et de processus.

6.2.2. Patterns dédiés à l'interopérabilité de procédés et de processus : le projet ECOO

Le projet ECOO (*Environnement pour la COOpération*) définit un ensemble de patterns de coopération génériques dédiés à la modélisation des interactions entre entités se répétant fréquemment d'une application à une autre [TATA & AL., 1999] [TATA, 2002] [BAÏNA & AL., 2003]. L'objectif est la mise en évidence d'un noyau représentatif de pratiques courantes de coopération. Ces patterns mettent notamment en évidence des règles d'échange de documents et sont classés en fonction de différents critères (dépendances lecture/écriture, cohérence synchrone ou asynchrone,…). Une démarche de développement de logiciels de support de coopération basé sur l'assemblage de patterns de coopération est également proposée [CANALS & AL., 1998]. Ces patterns sont cependant très spécifiques au domaine du travail collaboratif car s'appliquant à la coopération entre applications de type workflow.

REMARQUES & POSITIONNEMENT – *Les patterns conçus dans le cadre du projet ECOO sont très spécifiques au travail collaboratif et ne s'appliquent pas à la coopération de sources d'information, les notions de conflits structurels et sémantiques n'apparaissant pas. La démarche de développement de logiciels de support de coopération basé sur l'assemblage de patterns de conception, bien que différente de celle que nous proposons est cependant proche de la problématique inhérente à nos patterns de support technique.*

6.2.3. Positionnement de nos patterns par rapport aux patterns de coopération spécifiés par les travaux du domaine

Notre proposition s'inscrit dans le cadre de la conception de patterns dédiés à l'interopérabilité de données. Notre approche propose des patterns d'analyse et d'architecture génériques permettant de spécifier tout SIC, des patterns de conception ayant notamment pour objectif de concevoir un SIC spécifique dédié à la coopération de sources d'information hétérogènes et des patterns de support technique qui décrivent comment construire une application à partir des patterns de conception. Nos patterns adressent, en outre, chaque type d'hétérogénéité des sources d'information (hétérogénéités

technique, structurelle et sémantique). En ce sens, notre proposition est plus étendue que celles de Fernandez & al. (Cf. section 6.2.1.1) et Ganguly & al. (Cf. section 6.2.1.2) limitées à la modélisation de patterns essentiellement destinés à concevoir une architecture spécifique dédiée à la coopération de systèmes d'information et focalisés sur les problèmes d'hétérogénéité sémantique.

Nous effectuons une comparaison plus détaillée des différents projets dans le tableau suivant (Cf. Tableau 6). Cette comparaison s'effectue sur les points suivants :

- *Approche* : indique si le projet concerne l'approche *interopérabilité de procédés et de processus* (patterns d'interopérabilité de tâches, de procédés, de processus,…) ou l'approche *interopérabilité de données* (patterns de coopération de sources de données, d'intégration d'information).
- *Champ d'application* : les patterns proposés sont ils des patterns dédiés à la phase de capture des besoins, d'analyse, d'architecture, de conception ou d'implantation de systèmes d'information coopératifs ? Il s'agit également de préciser, ici, si les patterns proposés sont dépendants ou non d'un projet de coopération spécifique (dépendant vs. générique).
- *Types d'hétérogénéité pris en compte* : hétérogénéité technique, hétérogénéité structurelle, hétérogénéité sémantique. L'hétérogénéité technique se réfère à l'hétérogénéité des plates-formes, i.e. des systèmes d'exploitation et des systèmes matériels, sur lesquelles sont situées les sources d'information locales. L'hétérogénéité structurelle traduit entre les systèmes locaux une différence de modèles de données, de langages de requêtes et/ou de représentation des informations. L'hétérogénéité sémantique représente les conflits de domaine de définition, mais aussi les différences dans la signification ou l'interprétation d'une même donnée.

	Approche	**Champ d'application**				**Types d'hétérogénéité pris en compte**		
		Capture des besoins	Analyse	Conception	Implantation	Technique	Structurelle	Sémantique
Patterns du projet ECOO [TATA & AL., 1999] [CANALS & AL., 1998]	Procédés / processus	✘	✘	Génériques	Génériques	NA	NA	NA
Patterns de l'équipe SIGMA [SAIDANE & GIRAUDIN, 2002]	Procédés/processus & Données	✘	✘	Génériques	✘	✓	✘	✘
Patterns de Fernandez & al. [FERNANDEZ & ZHAO, 2000] [FERNANDEZ & AL., 2003]	Données	Dépendants	✘	Dépendants (architecture seulement)	✘	✘	✘	✓
Pattern de Ganguly & al. [GANGULY & AL., 2001]	Données	✘	✘	Dépendants	✘	✘	✘	✓
Notre proposition	Données	✘	Génériques	Génériques ou dépendants	Génériques	✓	✓	✓

Légende — ✓ : Oui ; ✘ : Non ; NA : *Non applicable*. Les patterns de coopération du projet ECOO étant dédiés au domaine du travail collaboratif, la typologie en fonction des types de conflit pris en compte n'est pas applicable.

Tableau 6 : Comparaison des patterns de coopération proposés par divers travaux de recherche

7. Conclusion

Nous avons présenté l'architecture de patterns conçue et basée sur cinq types de patterns. Les *patterns d'analyse de coopération*, utilisables lors de la phase d'analyse du processus d'ingénierie des SIC, permettent d'identifier et de représenter les entités métier du SIC à concevoir. Les *patterns d'architecture de coopération* s'appliquent au début de la phase de conception du processus d'ingénierie des SIC et proposent des modèles d'architecture "génériques" permettant de représenter les caractéristiques structurelles et organisationnelles de l'architecture globale d'un SIC. Les *patterns de conception de coopération* sont principalement liés à une architecture particulière, le SIC ACSIS et permettent de raffiner les entités métier spécifiées à l'aide des patterns d'analyse et de représenter leur comportement, ainsi que leurs collaborations. Les *patterns de support technique* sont des patterns génériques dédiés à la phase d'implémentation et d'implantation des SIC et décrivent comment construire une application à partir des patterns de conception de coopération. Enfin, les *patterns de support d'utilisation* permettent de guider le concepteur lors de la réutilisation des différents patterns précédents et favorisent ainsi cette réutilisation. Les chapitres 3 et 4 présentent ces différents patterns de façon détaillée.

Nous avons également présenté la démarche d'ingénierie des patterns de coopération que nous avons spécifiés. Les patterns d'analyse et d'architecture de coopération ont été conçus en utilisant des techniques d'ingénierie de domaine (abstraction des connaissances de domaine), alors que les patterns de conception de coopération ont été spécifiés en utilisant des techniques de rétro-ingénierie.

La structure des différents patterns proposés est basée sur un formalisme dont l'objectif est de fournir des rubriques spécifiques et appropriées aux différents types de patterns et principalement constituées de descriptions en langage naturel et de représentations sous formes de diagrammes ou de schémas (schémas d'architecture,…). Le formalisme présente cependant une interface cohérente et généralisée à tous les patterns. Cette interface permet de chercher et sélectionner le pattern adéquat plus facilement. Outre les patterns de support d'utilisation qui favorisent la réutilisation du système de patterns, le formalisme spécifie également des relations permettant de lier les différents patterns conçus. Cette réutilisation est en outre, en partie, automatisée par la réalisation d'un environnement logiciel de support de réutilisation (Cf. Annexe A) permettant notamment de rechercher un pattern et d'afficher ses relations avec d'autres patterns du système ou éventuellement des patterns conçus par d'autres travaux de recherche. Cet outil permet ainsi d'automatiser partiellement le processus d'ingénierie des SIC par réutilisation de patterns.

Enfin, une explicitation des travaux du domaine a permis de positionner nos patterns par rapport à ceux existants et a notamment mis en exergue l'originalité de nos propositions et, nous semble-t-il, le bien fondé de notre démarche.

Le chapitre 3 présente en détail nos patterns de coopération dédiés à la phase d'analyse du processus d'ingénierie des systèmes d'information coopératifs.

Chapitre 3 :
Patterns dédiés à l'analyse des systèmes d'information coopératifs

1. Introduction

Ce chapitre détaille le système de patterns s'appliquant lors de la phase d'analyse de l'ingénierie des SIC, i.e. les patterns d'analyse de coopération, ainsi que les patterns de support d'utilisation facilitant la réutilisation de ce système. Les patterns d'analyse de coopération ont pour objectif d'identifier et de spécifier les aspects structurels et fonctionnels d'un système d'information coopératif, i.e. les entités de ce système en termes de classes et de relations entre les classes et fournissent des fragments de modèle d'analyse (modèle de domaine) pour représenter les SIC. La structure de ces différents patterns repose sur le formalisme présenté précédemment (cf. section 4.2 du chapitre 2).

Préalablement à l'explicitation détaillée des patterns d'analyse de coopération que nous avons conçus (Cf. section 3), nous présentons le référentiel du domaine de la coopération, résultat de l'analyse de domaine, à partir duquel nous avons spécifié les patterns. Enfin, la dernière section (§ 4) présente les futurs travaux d'extension du système de patterns d'analyse proposé.

2. Référentiel du domaine de la coopération

L'analyse du domaine de la coopération a permis d'acquérir les connaissances sur ce domaine et de les structurer dans un référentiel de domaine générique permettant de constituer un cadre de référence pour les développements de tous les systèmes de ce domaine. Le référentiel du domaine, ainsi conçu et présenté dans cette section, expose les aspects communs et disparités conceptuels des systèmes d'information coopératifs et des approches de coopération de SI, et donc de la coopération au sens large. Il a plus précisément pour vocation de :

- délimiter et classifier l'ensemble des concepts ou entités (connaissances du domaine) gérés dans les SIC et dans les approches coopératives, et, ce, en mettant en exergue leurs similarités et variabilités.
- préciser la terminologie et la définition utilisées pour chaque concept ou entité identifié,
- proposer un modèle de domaine qui résout les problèmes de modélisation en structurant et reliant ces entités.

La classification des entités et concepts des SIC et approches de coopération, ainsi que les différentes terminologies et définitions utilisées sont exposées en section suivante (§ 2.1). Nous présentons, par la suite, le modèle de domaine résultant (§ 2.2).

REMARQUES — Les différents concepts et entités présentés, i.e. les caractéristiques des SIC et des approches de coopération, sont modélisés à l'aide du langage UML. Nous aurions également pu utiliser le modèle FODA permettant de représenter sous formes de diagrammes les caractéristiques obligatoires, optionnelles ou alternatives d'un système (et ainsi ses similarités et variabilités). Lam & McDermid [LAM & MCDERMID, 1997] préconisent, cependant, l'utilisation de diagrammes UML pour représenter ces concepts en raison de son utilisation généralisée et du fait qu'il est possible de représenter ces similarités et variabilités par l'utilisation notamment de classes abstraites, de spécialisations et de commentaires UML. Pour éviter toute surcharge des modèles, ces commentaires ont été intégrés au niveau des explications. De la même façon, UML est le langage de modélisation utilisé pour représenter les solutions et exemples d'utilisation des patterns proposés.

2.1. Entités et concepts caractéristiques du domaine de la coopération

REMARQUE — les exemples fournis dans les sections suivantes, ainsi que les exemples d'utilisation des patterns d'analyse sont issus de l'étude de projets coopératifs existants. Les concepts de ces travaux pouvant évoluer ou être représentés différemment en fonction des publications (articles, thèses,…), nous avons cité celles à partir desquelles nous les avons modélisés. Remarquons enfin que les modélisations proposées sont subjectives et pourraient ainsi être (ou auraient pu être) représentées différemment par les auteurs de ces projets.

2.1.1. Source d'information

Le domaine étudié étant la coopération de sources d'information, les sources d'information à faire coopérer sont les éléments essentiels du référentiel du domaine. Chaque source d'information est

caractérisée par son type et éventuellement le modèle sous-jacent utilisé (Cf. section 2.1.6) (Cf. Figure 16).

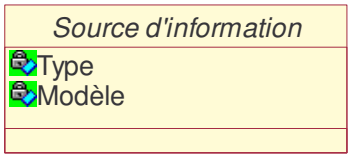


Figure 16 : Structure de l'entité « Source d'information »

Le type des sources d'information varie selon les projets de coopération (Cf. Figure 17). En effet, certains projets n'assurent la coopération que de sources d'information structurées telles les bases de données (MRSDM, Multibase,…) ; d'autres projets acceptent également des sources d'information semi-structurées telles les ressources Web (Infomaster, Infosleuth…) ; enfin, quelques SIC peuvent également traiter les conflits provenant de sources d'information non structurées, i.e. stockant, par exemple, du texte libre ou des données brutes (Tsimmis, Picsel, InfoSleuth…).

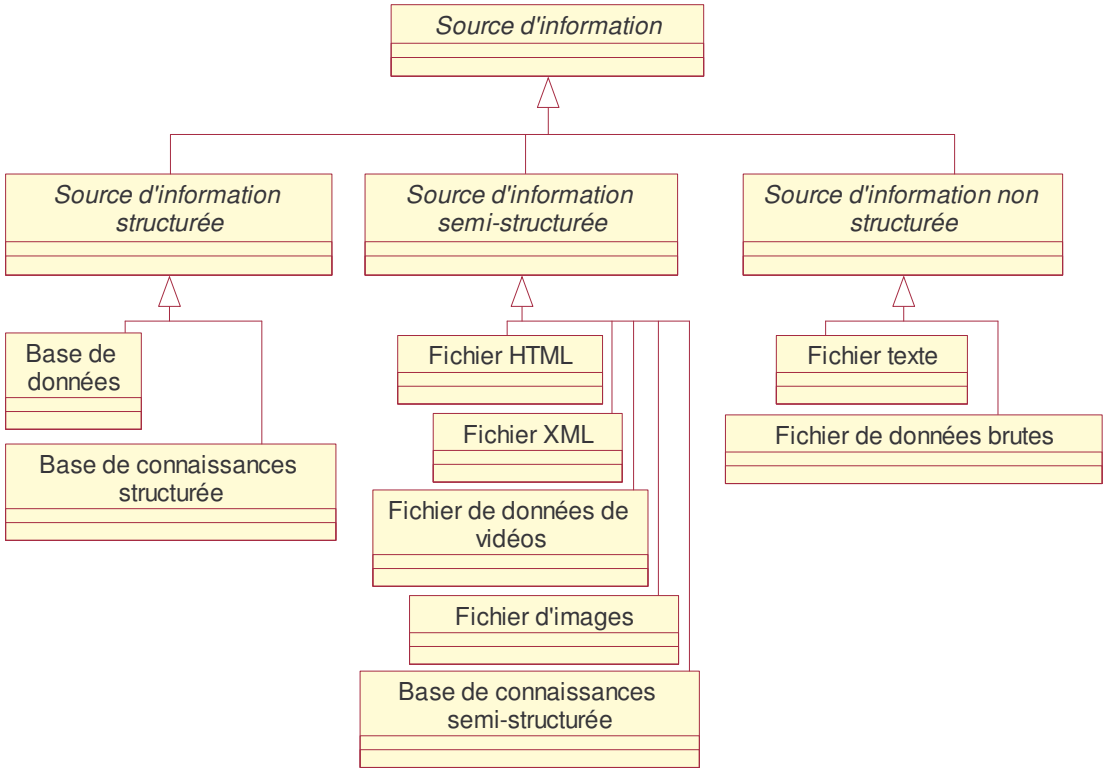


Figure 17: Les différents types de sources d'information locales gérées par les projets de coopération

REMARQUE — Certains projets (InfoSleuth, DILEMMA,…) proposent une autre typologie des sources d'information et considèrent notamment les fichiers contenant des données de vidéos, les

fichiers HTML ou les fichiers d'images comme des sources d'information non structurées. La typologie que nous proposons est cependant la plus partagée.

2.1.2. Conflit

La notion de conflit est sous-jacente à chaque projet de coopération, l'objectif de ce dernier étant de traiter ces conflits (un conflit est considéré comme un événement survenant lors de la résolution d'une requête). Des conflits peuvent intervenir entre les informations issues de la même ou de plusieurs sources d'information coopérantes.

Un conflit possède un type particulier caractérisé par son intitulé, sa définition et son éventuel traitement (attribut booléen « Traité ») par le SIC (chaque projet coopératif propose en général ses propres classification et définition des types de conflit, mais ces derniers ne sont pas toujours tous traités) (Cf. Figure 18).

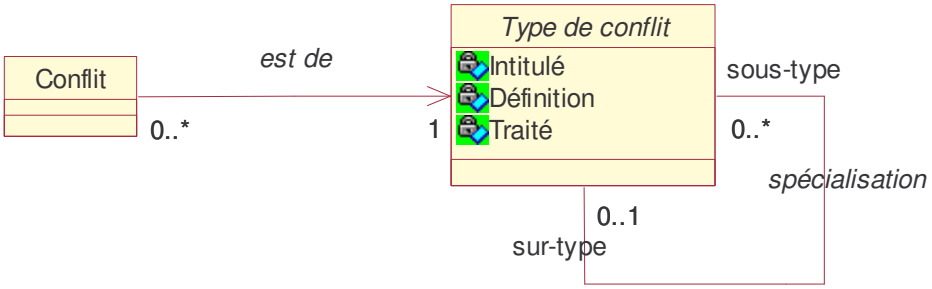


Figure 18 : Modélisation des entités « Conflit » et « Type de conflit »

La taxonomie des conflits (type, définition, traitement) variant très fréquemment d'un projet à l'autre, la hiérarchie des types de conflits est fonction des SIC. Les deux premiers exemples portant sur les projets coopératifs ACSIS et DILEMMA permettent d'illustrer le fait que des types de conflit dont les définitions sont relativement proches peuvent être regroupés de façon divergente. Le troisième projet, le projet COIN, propose une taxonomie assez différente des types de conflits.

Types de conflit définis dans le cadre du projet de coopération ACSIS (Cf. section 3.1.2.2 du chapitre 1) :

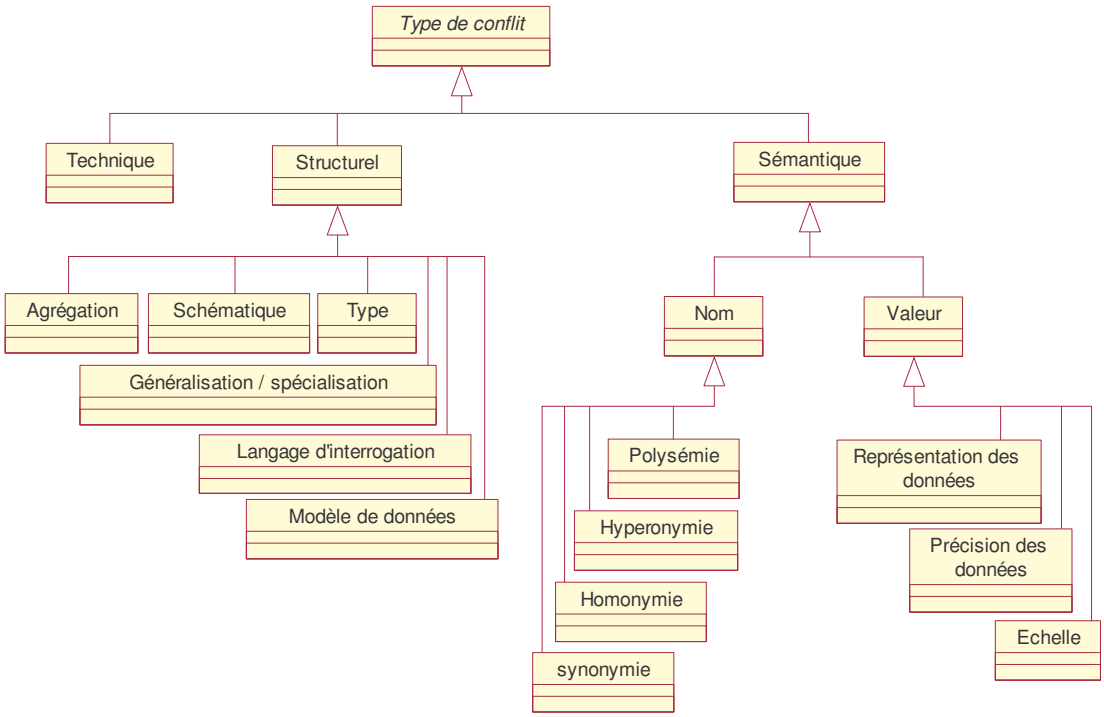


Figure 19 : Types de conflit spécifiés dans le projet de coopération ACSIS

Types de conflit définis dans le cadre du projet DILEMMA [JOUANOT, 2001]:

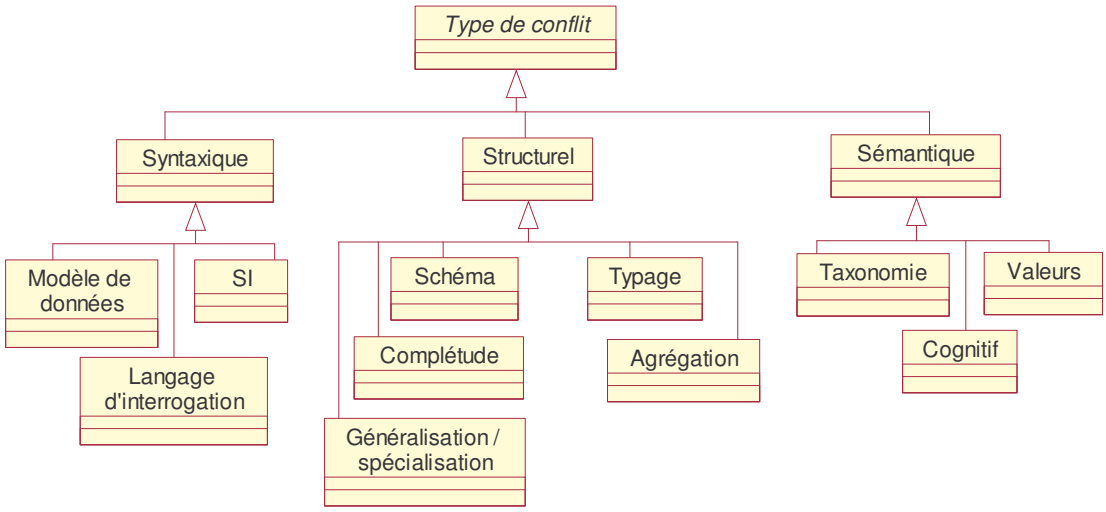


Figure 20 : Types de conflit spécifiés dans le projet de coopération DILEMMA

- *Conflit syntaxique :* lié à l'hétérogénéité des modèles de données et des langages d'interrogation des sources d'information. Une même information peut être représentée par différents modèles avec une syntaxe et des concepts différents.
- *Conflit structurel :* relatif aux choix de conception établis par chaque composant coopérant pour représenter les informations. Une même information peut être structurée différemment sur deux SI.
    - ⇨ *Conflit de schémas :* résulte de l'utilisation d'éléments de modélisation différents pour représenter la même information.
    - ⇨ *Conflit de généralisation/spécialisation :* résulte des différences de hiérarchisation des informations. Dans un système, plusieurs concepts peuvent être utilisés pour représenter des entités voisines du monde réel alors qu'un autre système n'utilise qu'un seul concept plus général.
    - ⇨ *Conflit d'agrégation :* issu d'un niveau de granularité différent des données entre deux systèmes. La valeur d'un attribut sur un système est l'agrégation des valeurs de plusieurs attributs d'une on plusieurs instances sur un autre système.
    - ⇨ *Conflit de complétude :* résulte de la mise en relation d'informations équivalentes mais dont les structures sont incomplètes, c'est-à-dire dont une partie des données ne se retrouvent pas d'un système à l'autre.
    - ⇨ *Conflit de typage :* se résume à des différences dans le typage des données entre systèmes coopérants.
- *Conflit sémantique :* provient de l'hétérogénéité de description des informations.
    - ⇨ *Conflit de taxonomie :* lié aux hétérogénéités de vocabulaire.
    - ⇨ *Conflit de valeur :* lié à la manière d'utiliser et de coder la valeur d'une donnée dans différents systèmes.

REMARQUE — les conflits cognitifs et les conflits de SI bien que faisant partie de la typologie proposée par DILEMMA ne sont pas définis.

Types de conflits définis dans le cadre du projet COIN [GOH, 1997] :

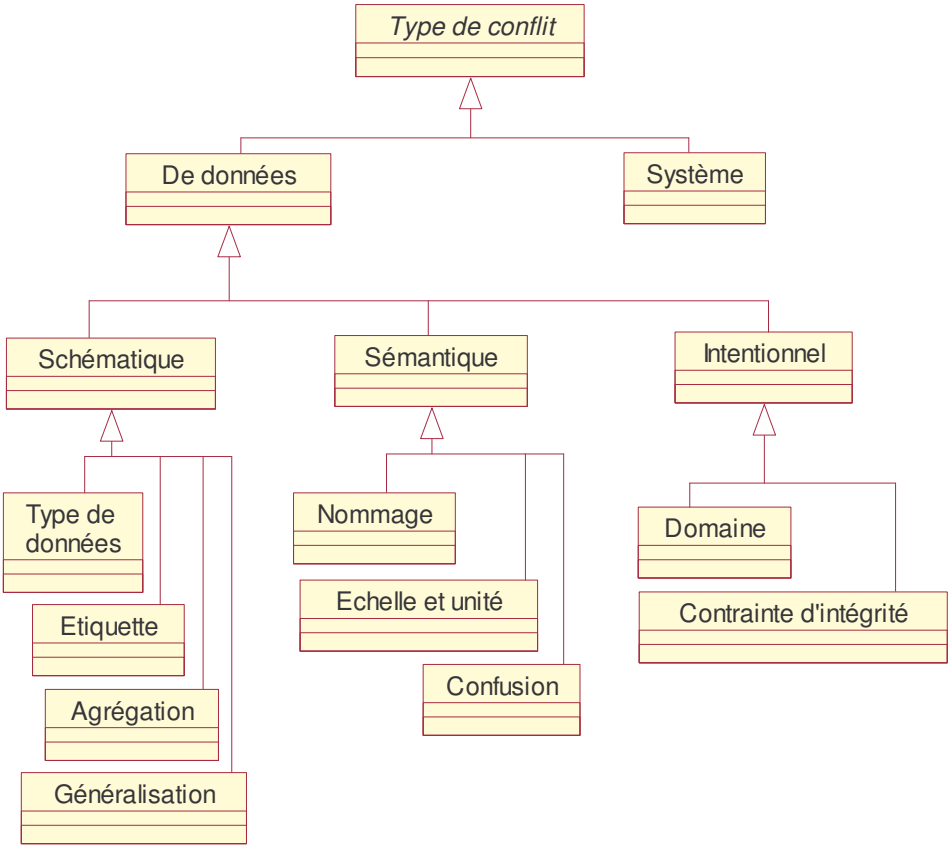


Figure 21 : Types de conflit spécifiés dans le projet de coopération COIN

- *Conflit système :* concerne les différences entre les modèles de données, les langages de manipulation de données et les mécanismes de contrôle de la concurrence.
- *Conflit de données :* se réfère aux différentes façons dont les données sont organisées ou interprétées dans des systèmes hétérogènes.
    - ⇨ *Conflit schématique :* correspond à des différences au niveau des schémas de données.
        - *Conflit de type de données :* se réfère à l'utilisation de types différents pour représenter la valeur d'une même donnée.
        - *Conflit d'étiquette :* se réfère aux synonymes et homonymes survenant parmi les éléments du schéma de données : le même attribut peut être associé à des étiquettes différentes ou la même étiquette peut être utilisée pour identifier des attributs différents.

- *Conflit d'agrégation :* concerne des choix de conception différents portant sur la façon dont les données sont regroupées pour former les attributs d'une entité.
- *Conflit de généralisation :* fait référence à la façon dont les données sont classées entre elles les unes par rapport aux autres.

⇨ *Conflit sémantique :* se réfère au fait que des données issues de systèmes différents peuvent être assujetties à des interprétations différentes, même si les schémas de données correspondants sont identiques.

- *Conflit de nommage* : consiste en la présence de synonymes et d'homonymes parmi les attributs.
- *Conflit d'échelle et d'unité :* se réfère à l'adoption d'unités de mesure ou d'échelles différentes pour représenter deux attributs proches sémantiquement.
- *Conflit de confusion :* se réfère à la confusion de concepts qui sont en fait distincts.

⇨ *Conflit intentionnel :* fait référence aux différences de contenu informationnel présent dans les sources d'information ou attendu par l'utilisateur.

- *Conflit de domaine :* se réfère aux désaccords entre les domaines, c'est-à-dire à l'univers du discours modélisé par chaque système composant. Prenons l'exemple de deux sources d'information ayant pour but de fournir le résultat financier de compagnies américaines. La première fournit l'information "toutes les fortunes des 500 compagnies américaines du secteur manufacturier". L'autre source liste "toutes les compagnies américaines cotées en bourse dont les avoirs cumulés avoisinent le milliard de dollars". Les deux éléments peuvent être identiques, disjoints ou bien un des éléments peut englober l'autre.
- *Conflit de contrainte d'intégrité :* concerne les disparités existantes entre les contraintes d'intégrité selon les systèmes.

2.1.3. Tâche de coopération

Une tâche de coopération définit un événement de coopération à traiter par l'association entre un ensemble d'entrées, c'est-à-dire de données à traiter, et un ensemble de sorties, c'est-à-dire de résultats produits, et par l'indication de leurs caractéristiques respectives. Ainsi, une tâche utilise des

ressources, soit de type source d'information (Cf. section 2.1.1), soit de type référentiel de coopération (Cf. section 2.1.5). Elle associe également une stratégie de traitement à cet événement. Une stratégie de traitement décrit comment la tâche concernée peut être décomposée en (sous-)tâches de plus en plus élémentaires et indique l'ordre de leur exécution (utilisation des concepts de pré-tâche et de post-tâche). Elle conduit finalement à un enchaînement de tâches élémentaires, dont la réalisation et l'exécution par les modules fonctionnels et/ou les acteurs mène au traitement complet de l'événement correspondant.

Une tâche de coopération est définie par (Cf. Figure 22) :

- son intitulé,
- sa description (type,…),
- ses flux entrants (attribut « Entrées »),
- ses flux sortants (attribut « Sorties »),
- la condition déclenchante de la tâche (attribut « Pré-condition »). Une pré-condition constitue une condition qui doit être vérifiée pour pouvoir exécuter une tâche.
- la condition d'arrêt de la tâche (attribut « Post-condition »). Une condition de sortie ou post-condition distingue des conditions d'abandon (l'objectif de la tâche ne pourra pas être atteint) et des conditions de terminaison normale.
- Pré-tâches : liste des tâches devant être exécutées avant de pouvoir exécuter le corps de la tâche elle-même. Les pré-tâches servent en général à rendre vraies les pré-conditions.
- Post-tâches : liste des tâches devant être exécutées après l'exécution d'une tâche donnée et qui permettent en particulier d'évaluer les résultats de cette dernière.

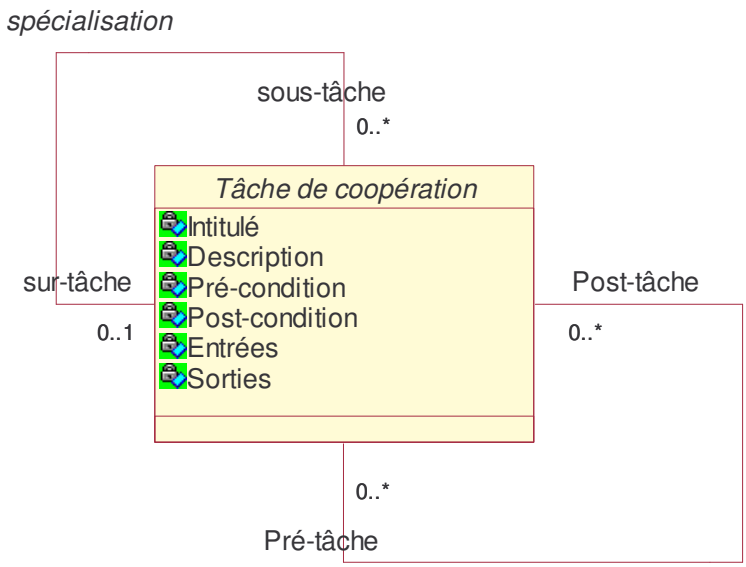


Figure 22 : Structure de l'entité « Tâche de coopération »

Notre approche permet de décrire à la fois des tâches abstraites ou des taches concrètes.
Les tâches sont modélisées par des classes d'objet. Comme toute classe d'objet, une classe modélisant une tâche fait partie d'une hiérarchie de classes. La classe racine d'une hiérarchie de tâches modélise l'événement correspondant à un niveau très général. La description de l'événement, c'est-à-dire l'ensemble de ses entrées / sorties et leurs caractéristiques, est de plus en plus raffinée dans les sous-classes successives. Si différents cas plus concrets peuvent être distingués pour un événement modélisé par une classe, ils sont modélisés par autant de sous-classes. Cette modélisation permet de représenter et de décrire les différents niveaux (ou granularité) de tâches de coopération spécifiés dans les SIC.

Remarquons que les tâches de coopération spécifiées, leurs intitulés et descriptions, ainsi que leur hiérarchie diffèrent selon les projets de coopération. Les deux exemples suivants illustrent cette variabilité.

Tâches de coopération spécifiées dans le cadre du projet MRDSM [LITWIN, 1985] [LITWIN, 1988] (Cf. Figure 23) :

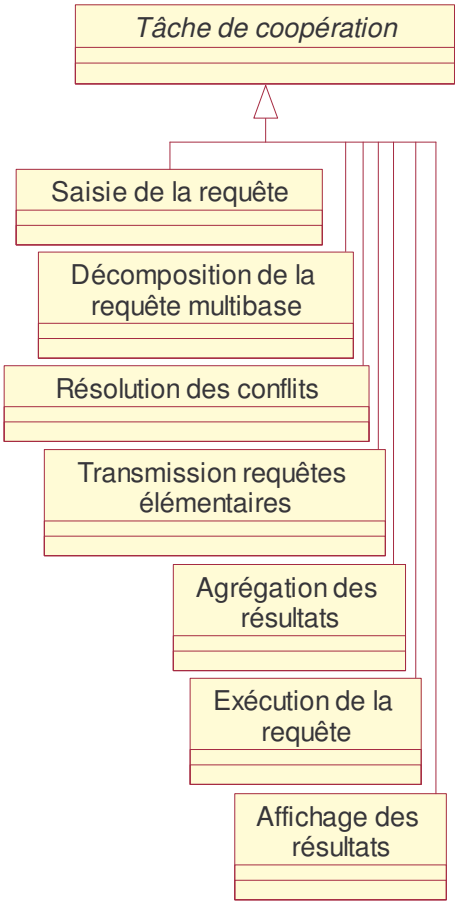


Figure 23 : Tâches de coopération spécifiées dans le projet MRDSM

- *Saisie de la requête :* tâche de création de la requête utilisateur soumise à la multibase.
- *Décomposition de la requête multibase :* décomposition si nécessaire de la requête multibase (si elle concerne plusieurs bases de données).
- *Résolution des conflits :* consiste à utiliser et définir manuellement des concepts et techniques ("variables sémantiques", "attributs dynamiques",…), qui seront ensuite pris en compte par le système lors du traitement des conflits.
- *Transmission requêtes élémentaires :* transmission des requêtes élémentaires, obtenues après décomposition de la requête multibase, au gestionnaire d'exécution.
- *Exécution de la requête :* traitement de la requête sur la base de données locale.
- *Agrégation des résultats :* consiste à appliquer les clauses « where » et les jointures définies dans les requêtes et à restructurer les résultats.

- *Affichage des résultats :* tâche permettant l'affichage des résultats "agrégés".

Tâches de coopération spécifiées dans le cadre du projet DILEMMA [JOUANOT, 2001] (Cf. Figure 24) :

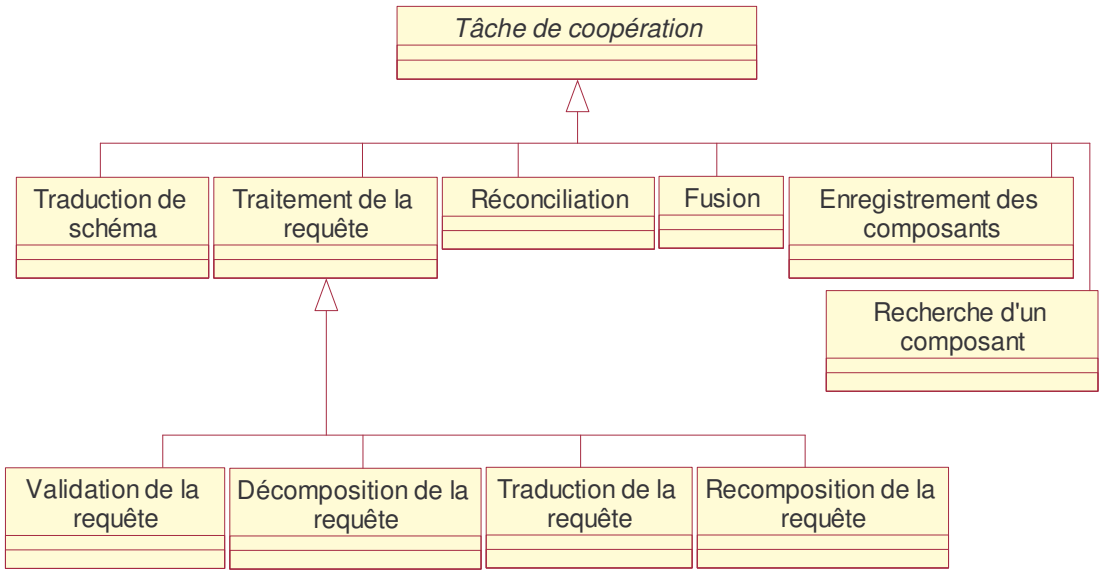


Figure 24 : Tâches de coopération spécifiées dans le projet DILEMMA

- *Traduction de schémas :* consiste à construire un schéma de données dans le modèle de médiation à partir des informations locales.
- *Traitement de la requête globale :*
  - ⇨ *Validation de la requête :* vérification de la syntaxe et de la cohérence d'une requête exprimée dans le modèle de médiation.
  - ⇨ *Décomposition de la requête :* décomposition de la requête globale en sous-requêtes.
  - ⇨ *Traduction de la requête :* permet de traduire une requête exprimée sur le schéma de données dans le modèle de médiation en une interrogation du système existant pour extraire des données.
  - ⇨ *Recomposition de la requête :* rassemble les résultats intermédiaires pour construire un résultat correspondant à la requête initiale posée par l'utilisateur.
- *Réconciliation :* permet de découvrir dynamiquement les données pertinentes et de les adapter au contexte de l'application. Elle compare les contextes des SI locaux en utilisant une mesure de distance sémantique et identifie les informations utiles dans une coopération.

- *Fusion d'objets :* construit des classes d'objets virtuels donnant naissance à des vues combinant et transformant des objets locaux et assurant un accès transparent aux informations d'une coopération.
- *Enregistrement des composants :* permet l'ajout d'un composant du système à un référentiel. Chaque composant doit s'enregistrer pour pouvoir être utilisé comme fournisseur.
- *Recherche d'un composant :* identifie les systèmes contenant les informations dont le consommateur a besoin.

2.1.4. Processus de coopération

Une tâche de coopération est employée pour définir une partie d'une activité coopérative. Cette activité coopérative est également appelée processus de coopération. Un processus de coopération peut ainsi être défini comme un ordonnancement partiel de tâches.
Un processus de coopération est caractérisé par les attributs suivants (Cf. Figure 25) :

- Intitulé
- Description (objectifs, type…)
- Alphabet : série de tâches ordonnancées du processus
- EvDéclencheur : événement ou liste d'événements déclenchant le processus. Par exemple, la soumission d'une requête, l'ajout d'une source d'information,…

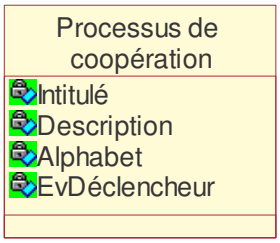


Figure 25 : Structure de l'entité « Processus de coopération »

Dans certains SIC, un processus de coopération est instancié à chaque soumission d'une requête à la coopération ; un processus est ainsi associé à une requête et est constitué d'une liste de tâches ordonnancées permettant de résoudre cette requête. Dans d'autres projets, la notion de processus est plus ou moins indépendante de la soumission d'une requête. Par exemple, dans le cadre du projet ACSIS, deux types de processus sont définis : le processus d'enregistrement des sources d'information locales (instancié lors de l'ajout d'une nouvelle source d'information) et le processus de traitement d'une requête globale (instancié lors de la soumission d'une requête à la coopération).

2.1.5. Référentiel de coopération

Un référentiel de coopération est une ressource, utilisée par une tâche de coopération, assurant le stockage de toutes les informations utiles à la coopération (schéma global, extensions du langage multibase, concepts ontologiques,…).
Un référentiel de coopération est caractérisé par (Cf. Figure 26) :

- son intitulé.
- sa description (contenu,…).
- son type.
- le modèle sous-jacent (Cf. section 2.1.6).

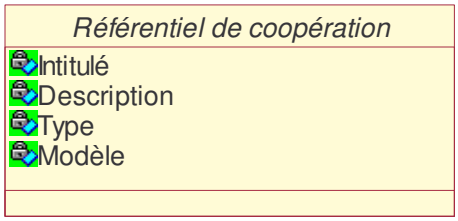


Figure 26 : Structure de l'entité « Référentiel de coopération »

Trois types de référentiels, correspondant à chacune des approches de coopération, peuvent être relevés dans la littérature (Cf. Figure 27) :

- *Dictionnaire de la fédération* (référentiel commun aux approches fédérées faiblement et fortement couplées) *:* référentiel, utilisé dans le cadre des projets issus de l'approche fédérée, contenant la description globale de la fédération, les informations concernant l'interface du système fédéré, le ou les protocoles utilisés dans la coopération (protocole de négociation entre les sources locales qui coopèrent pendant la phase de manipulation des données,…) et les liens inter-bases. Ce dictionnaire assure la cohérence du système fédéré et l'intégrité des données accessibles via la fédération.
- *Schéma de médiation :* référentiel utilisé dans le cadre des projets de type médiation de schémas spécifié à l'aide d'un modèle de médiation et représentant les méta-données des sources d'information locales, ainsi que des concepts sémantiques entre méta-données.
- *Ontologie :* référentiel, utilisé dans le cadre des projets de type médiation de contextes, stockant le contexte des informations des sources composantes (informations littérales et/ou attributs décrivant les propriétés intrinsèques d'un élément ou d'une valeur).

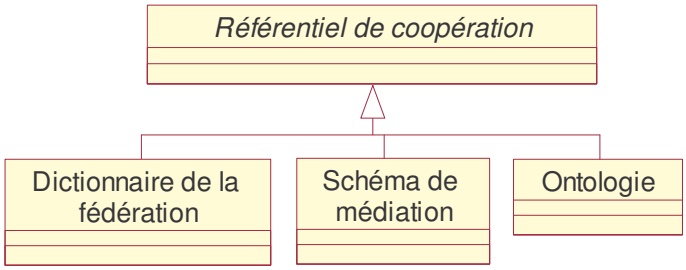


Figure 27 : Les différents types de référentiel de coopération

La description du référentiel (contenu, rôle), ainsi que le nombre de référentiels utilisés sont fonction du projet. Un projet peut par exemple utiliser plusieurs ontologies pour stocker les concepts du ou des domaine(s).

2.1.6. Modèle

Un modèle contient des instances d'éléments de modélisation[79] pour représenter les abstractions de certaines sources d'information et référentiels de coopération. Les instances d'éléments de modélisation constituant le modèle sont notamment utiles pour définir les spécificités des modèles propriétaires.

Un modèle possède les attributs suivants (Cf. Figure 28) :

- Intitulé (ex. : KIF, LDL,…)
- Type : modèle relationnel, modèle orienté-objet, modèle XML, modèle propriétaire, modèle de logique terminologique, modèle de logique déductive,… (Cf. section 3.1.3.3)
- Langage d'interrogation associé : indique le ou les langages d'interrogation associé(s) au modèle permettant de manipuler les données contenues dans la source d'information ou le référentiel de coopération (SQL, OQL, KIF,…).
- Type de rôle : modèle local, modèle canonique, modèle d'ontologie, modèle de médiation,…

[79] Un élément de modélisation est par exemple, une classe, un type d'objet, un attribut, une opération, une entité,… Ainsi, une instance d'un élément de modélisation sera par exemple, classe X, attribut Z,… [MULLER & GAERTNER, 2000]

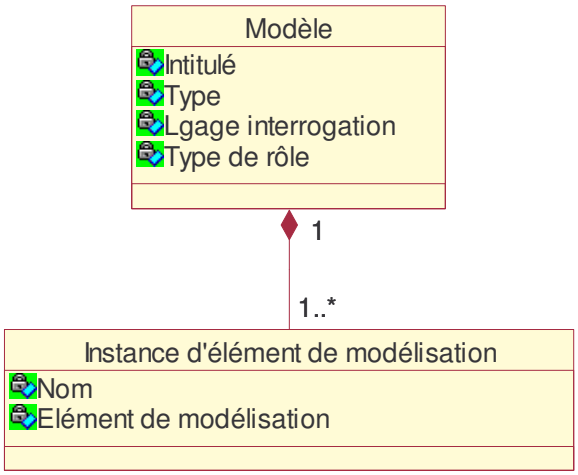


Figure 28 : Structure de l'entité « Modèle »

Le type de rôle du modèle varie selon la ressource modélisée. Dans le cas de la modélisation d'une source d'information locale, le type de rôle sera toujours "modèle local ". Ce type de rôle est cependant variable selon le type et le contenu de chaque référentiel de coopération. Ainsi, un référentiel contenant le schéma de données de la coopération exprimé dans le modèle canonique aura pour modèle associé un modèle dont le type de rôle sera "modèle canonique". L'exemple suivant illustre les différents modèles utilisés dans le cadre du projet InfoSleuth (Cf. Figure 29).

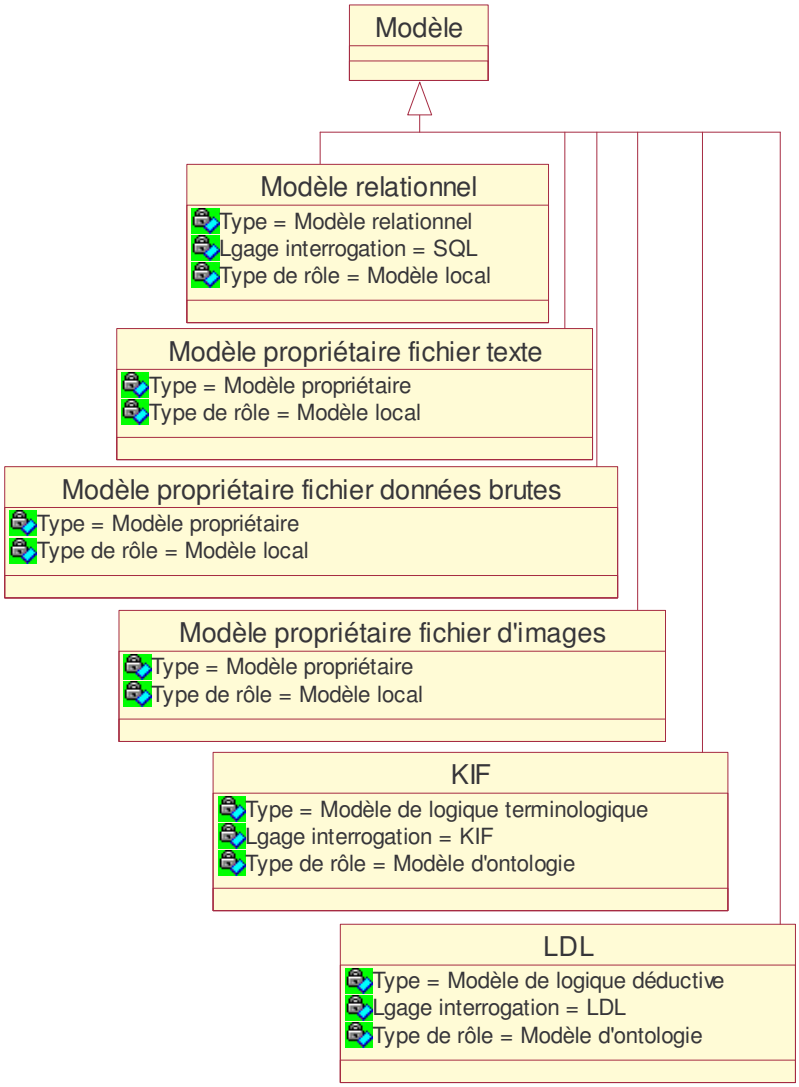


Figure 29 : Les différents modèles utilisés dans le projet de coopération InfoSleuth [BAYARDO & AL., 1997] [NODINE & AL., 2000]

2.1.7. Acteur

Personne ou individu intervenant dans une tâche de coopération. Un acteur peut avoir un ou plusieurs rôles dans un processus de coopération. Ce rôle distinct est fonction de la tâche de coopération à laquelle il participe (Cf. section 2.1.9). Ainsi, l'attribut « Type de rôle » contient la liste des types de rôles que peut prendre l'acteur (Cf. Figure 30).

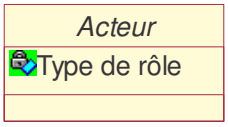


Figure 30 : Structure de l'entité « Acteur »

Chaque rôle correspond à un ensemble de permissions [FERRAIOLO & KUHN, 1992] sur le système et est représenté par une spécialisation de l'entité "acteur" : expert ou utilisateur (Cf. Figure 31). La plupart des SIC nécessite la participation d'un utilisateur (même si cet acteur est souvent décrit implicitement), intervenant notamment, lors de la saisie de la requête soumise à la coopération. En fonction du degré d'automatisation du traitement des événements, un expert peut également intervenir dans certaines tâches de coopération, comme, par exemple, pour créer manuellement des liens de correspondance sémantique entre certaines informations, afin de traiter certains types de conflits (projets DILEMMA, ACSIS, MRDSM…). Cet expert est parfois qualifié d'expert du domaine.

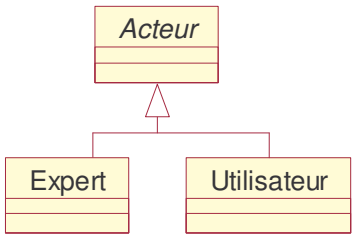


Figure 31 : Les entités « Acteur », « Expert » et « Utilisateur »

2.1.8. Module fonctionnel

Un module fonctionnel est un lieu de computation et d'état. Chaque module fonctionnel réalise et/ou contrôle une ou plusieurs tâche(s) de coopération identifiée(s), soit de façon autonome soit en coopération avec un (ou plusieurs) autre(s) module(s) et/ou acteur(s). Un module fonctionnel peut être composé de plusieurs modules fonctionnels.

Un module fonctionnel est caractérisé par les attributs suivants (Cf. Figure 32) :

- Intitulé : intitulé du module
- Description : ensemble des propriétés décrivant le module
- Type de rôle : liste de types de rôle que peut prendre le module fonctionnel. Un module fonctionnel pouvant réaliser différentes tâches de coopération (ou parties de tâches), il

dispose donc d'un rôle spécifique en fonction de la tâche (ou groupe de tâches) à accomplir (comportement). Ce rôle peut-être par exemple, médiateur, adaptateur,… Comme pour l'entité "Acteur", chaque rôle correspond à un ensemble de permissions sur le système et est représenté par une spécialisation de l'entité "module fonctionnel".

- Type : conceptuel ou logiciel. Un module fonctionnel est soit un module fonctionnel conceptuel, soit un module fonctionnel logiciel en fonction de l'association ou non d'un prototype ou d'une application opérationnelle aux spécifications conceptuelles (Cf. section 3.1.3.3 du chapitre 1).
- Représentation : indique la représentation sous-jacente au module fonctionnel. Par exemple, un objet, un agent artificiel,…

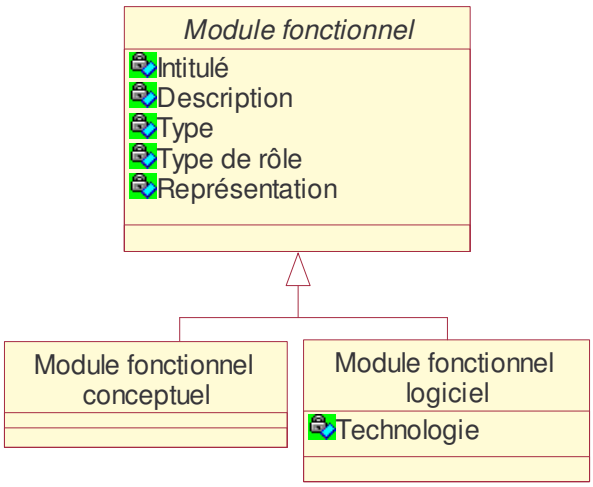


Figure 32 : Les entités « Module fonctionnel », « Module fonctionnel conceptuel » et « Module fonctionnel logiciel »

Un module fonctionnel logiciel dispose d'un attribut supplémentaire intitulé « Technologie » permettant de lister les technologies ou plates-formes sur lesquelles reposent ce module.

Notons que modules fonctionnels logiciel et conceptuel ont parfois la même représentation associée ; dans d'autres cas, un module conceptuel peut-être implanté différemment (par exemple, un agent artificiel conceptuel implanté sous la forme d'un objet).

Le nombre, la représentation, l'intitulé, la description, la granularité et la hiérarchie des modules fonctionnels varient d'un SIC ou d'une approche de coopération à l'autre.
Ainsi, les approches fédérées ne définissent pas de modules fonctionnels spécifiques, alors que les approches de types médiation de schémas et médiation de contextes définissent chacune deux modules fonctionnels :

- médiateur de schémas et adaptateur pour la médiation de schémas,
- médiateur de contextes et adaptateur pour la médiation de contextes.

Notons que les définitions assignées aux médiateurs de schémas et médiateur de contextes sont proches voire identiques dans certains cas, mais que les tâches de coopération qu'ils sont censés réaliser peuvent varier.

Modules fonctionnels spécifiés dans le cadre de l'approche à base de médiation de schémas (Cf. Figure 33) :

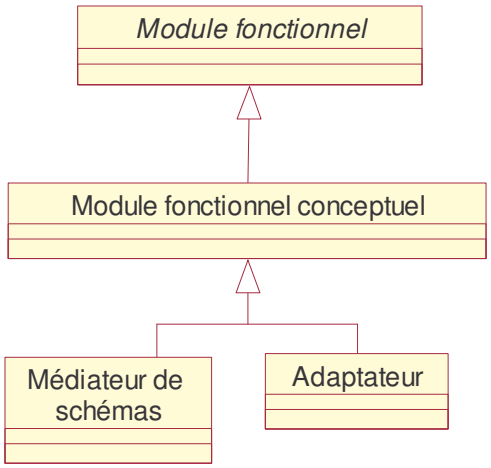


Figure 33 : Modules fonctionnels spécifiés dans l'approche à base de médiation de schémas

L'approche de coopération de type médiation de schémas définit deux modules fonctionnels conceptuels : le médiateur de schémas et l'adaptateur (Cf. section 3.1.3.2 du chapitre 1).

Modules fonctionnels spécifiés dans le cadre du projet DILEMMA [JOUANOT, 2001] :

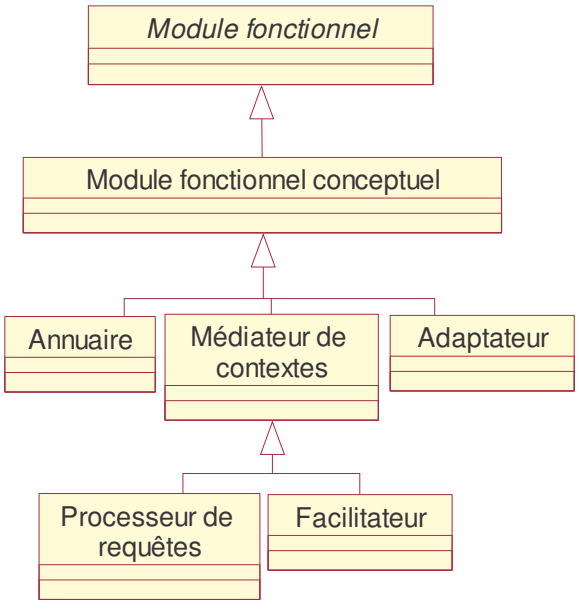


Figure 34 : Modules fonctionnels spécifiés dans le projet DILEMMA

Le projet DILEMMA définit cinq modules fonctionnels conceptuels (absence de prototype et d'application opérationnelle) (Cf. Figure 34) :

- *Médiateur de contextes :* est constitué de deux composants dynamiques qui réalisent la médiation, le facilitateur et le processeur de requêtes.
  - ⇨ *Facilitateur :* traite les conflits entre les informations du système local et les informations fournies par d'autres composants.
  - ⇨ *Processeur de requêtes :* reçoit la requête utilisateur exprimée dans le modèle de médiation et portant sur les informations du schéma de données. Il permet l'interrogation des différentes sources d'informations utiles à la résolution de cette requête et renvoie un résultat qui intègre les résultats des sources d'informations interrogées dans un format homogène adapté aux besoins de l'utilisateur.
- *Adaptateur* : permet d'interfacer le système existant avec sa description exprimée sur le modèle de médiation dans le schéma de données.
- *Annuaire :* a pour fonction de répertorier les composants disponibles (médiateurs, sources d'information) et de faciliter la recherche des systèmes fournisseurs qui peuvent répondre aux besoins d'un système consommateur. L'annuaire enregistre l'identité et la définition simplifiée du domaine d'application de chaque système participant.

REMARQUE — DILEMMA est un projet hybride de types médiation de schémas et médiation de contextes. Cependant nous considérons que ce SIC est plutôt représentatif de la médiation de contextes. Il continue d'ailleurs d'évoluer en ce sens [CULLOT & AL., 2003].

Modules fonctionnels spécifiés dans le cadre du projet InfoSleuth [BAYARDO & AL., 1997] [FOWLER & AL., 1999] :

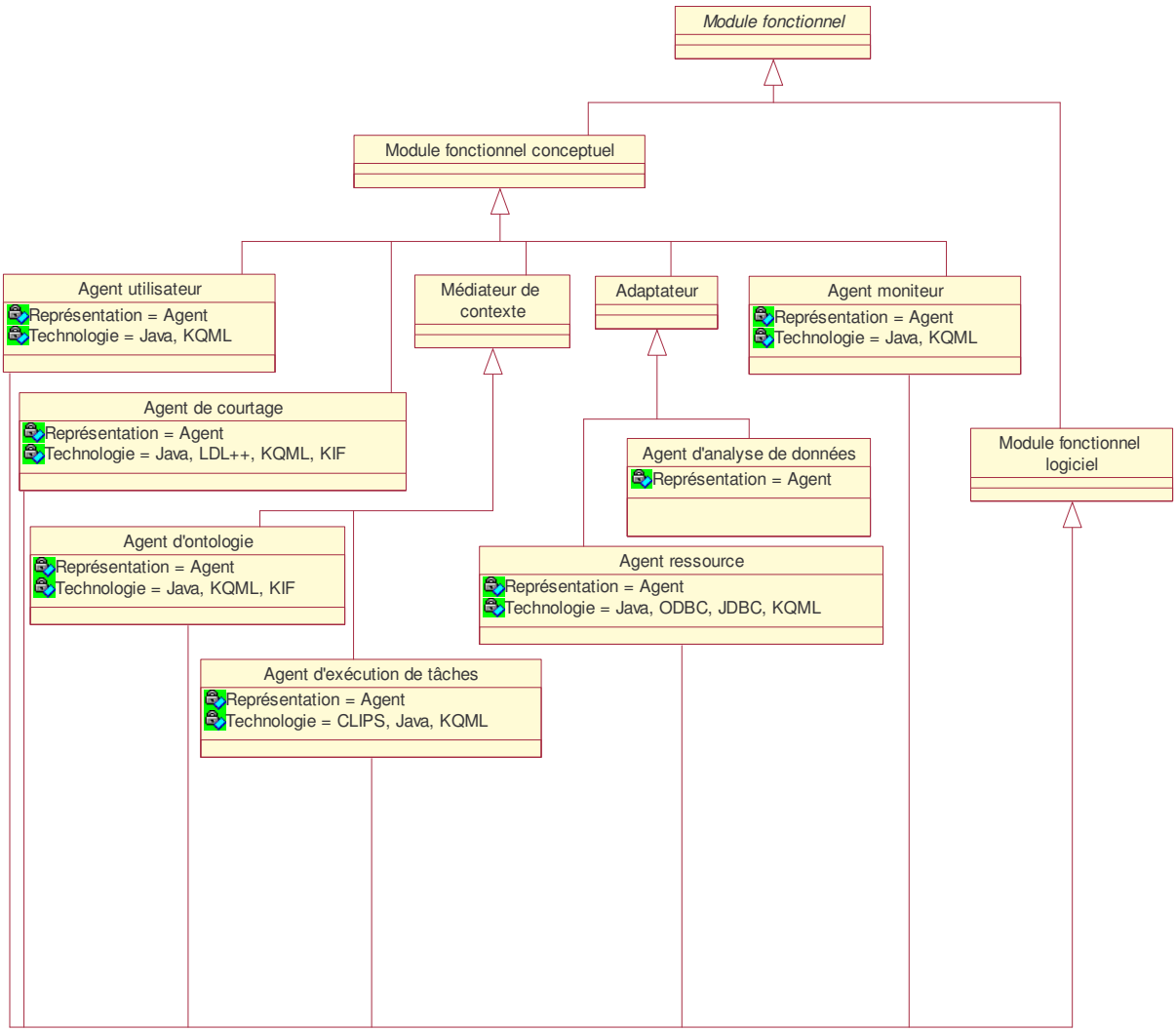


Figure 35 : Modules fonctionnels spécifiés dans le projet InfoSleuth

REMARQUE — LDL++ est un langage de base de données déductif ; CLIPS est un outil de construction de système expert [RILEY, 1991].

Le projet InfoSleuth définit sept modules fonctionnels à la fois conceptuels et logiciels (hormis le module « Agent d'analyse de données » non implanté dans l'application opérationnelle disponible) (Cf. Figure 35) :

- *Agent d'ontologie :* procure une connaissance générale des ontologies et gère chaque ontologie de domaine, i.e. la création, la mise à jour et l'interrogation de l'ontologie.
- *Agent de courtage :* répertorie les services et la localisation de chaque agent du système et a pour rôle d'assurer la mise en relation des agents fournisseurs et des agents clients.
- *Agent utilisateur :* utilise la connaissance des ontologies (modèles de domaine partagés) pour assister l'utilisateur dans la formulation de sa requête, puis récupère et affiche les résultats obtenus.
- *Agent ressource :* identifie les informations d'une source de données par rapport à une ontologie, traduit la requête globale dans le format local de la source, l'exécute et retourne les résultats dans le modèle de l'ontologie.
- *Agent d'analyse de données :* correspond aux agents ressource spécialisés dans les méthodes d'analyse et de fouille de données.
- *Agent d'exécution de tâches :* gère les requêtes exprimées sur l'ontologie (requêtes globales). Il utilise l'information fournie par l'agent de courtage pour identifier les ressources possédant l'information recherchée, décompose et transmet les requêtes aux agents ressource appropriés et rassemble les résultats.
- *Agent moniteur :* utilisé pour enregistrer les interactions entre les agents et les plans d'exécution des requêtes.

2.1.9. Rôle

Fonction, responsabilité, autorisation ou contrôle assignée à un module fonctionnel ou à un acteur pour l'exécution d'une tâche de coopération. Un rôle peut être attribué à un ou plusieurs acteur(s) et/ou module(s) fonctionnel(s) pour une période donnée et est fonction de la tâche (et non de l'acteur ou du module fonctionnel).

Un rôle est caractérisé par son intitulé et sa description (propriétés, type,…) (Cf. Figure 36).

Figure 36 : Structure de l'entité « Rôle »

Le référentiel du domaine de la coopération obtenu à l'issue de l'analyse de domaine organise l'ensemble des connaissances dans un modèle de référence. Ce modèle, assurant ainsi une structuration d'ensemble des entités et concepts de coopération précédemment explicités, est présenté en section suivante (§ 2.2).

2.2. Modèle du domaine de la coopération de systèmes d'information

Le modèle du domaine de la coopération, présenté ci-dessous (Cf. Figure 37), a été conçu à partir de la découverte et l'analyse des entités caractéristiques du domaine précédemment explicitées (Cf. sections 2.1.1 à 2.1.9). Il a été plus précisément bâti à partir des modélisations partielles que nous avons proposées et met en évidence les liens entre ces fragments de modèles. Lorsque des entités de modèles partiels se partagent des connaissances, nous définissons une super-classe à ces entités à laquelle sont attachées ces connaissances partagées. Deux classes abstraites ont été, de ce fait, ajoutées :

- la classe « Ressource », super-classe des classes « Source d'information » et « Référentiel de coopération ». Une ressource qualifie toute entité qui peut être créée, transformée ou utilisée par une tâche de coopération.
- la classe « Agent[80] », super-classe des classes « Acteur » et « Module fonctionnel », permet de représenter tout élément participant ou réalisant une tâche de coopération.

[80] Le terme « Agent » est fréquemment usité dans le domaine du travail collaboratif pour qualifier à la fois un acteur, un groupe d'acteurs, une machine ou un ordinateur, une application ou un programme, un automate,…

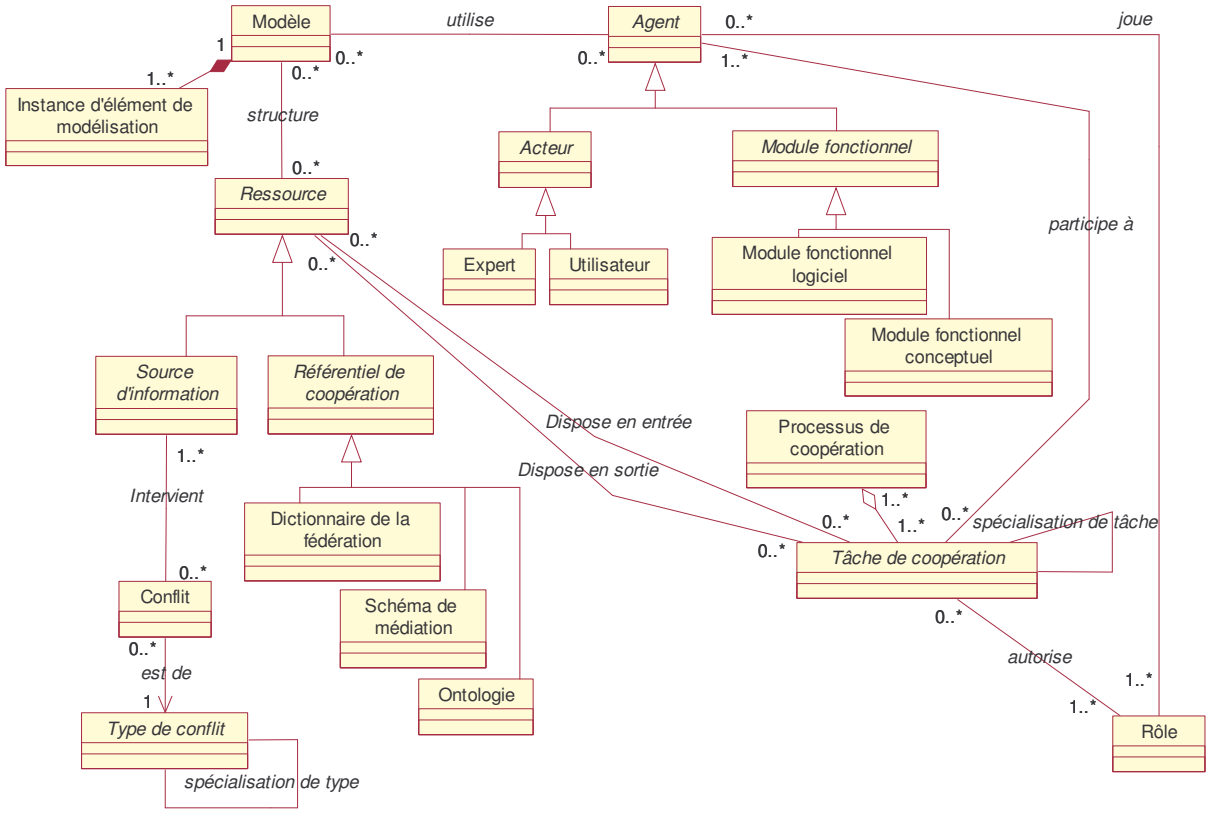


Figure 37 : Modèle du domaine de la coopération de systèmes d'information

Le modèle ci-dessus proposé permet de représenter le domaine de la coopération de systèmes d'information. Par conséquent, il permet également de représenter et catégoriser les concepts et principes des systèmes d'information coopératifs et des approches coopératives existants et ainsi de pouvoir notamment faire ressortir leurs points communs ou particularismes. Ce modèle exprime également des connaissances de haut niveau réutilisables dans le développement de nouveaux SIC ou de nouvelles approches coopératives et constitue ainsi un cadre de référence pour le développement de ces systèmes ou approches. Nous explicitons comment le modèle proposé est réutilisé lors du développement de systèmes d'information coopératifs dans la section suivante (§ 3).

3. Patterns d'analyse de coopération

3.1. Introduction

Le modèle de domaine présenté est général et abstrait. Les propriétés des SIC doivent ainsi être représentées en adaptant celui-ci. Ce modèle n'est pas facilement exploitable par le concepteur de

SIC, car il ne sait pas forcément comment parfaire cette adaptation. Notre objectif consiste à garder le modèle général et abstrait obtenu et à fournir ensuite un ensemble de patterns d'analyse de coopération, chacun permettant de guider le concepteur, à l'aide d'exemples d'utilisation, dans son adaptation et ainsi de représenter un fragment du modèle spécifique du SIC. Les patterns d'analyse de coopération permettent ainsi de documenter tout ou partie de ce modèle et de fournir des exemples d'utilisation à partir des systèmes existants afin de faciliter la tâche du concepteur du SIC (Cf. Figure 38).

Les patterns d'analyse de coopération permettent de faciliter l'analyse du système d'information coopératif à concevoir. Ils se focalisent sur la représentation du comportement fonctionnel du système et sur la description approfondie du domaine du problème. Ils permettent d'aider le concepteur du système à représenter sa perception du monde réel.

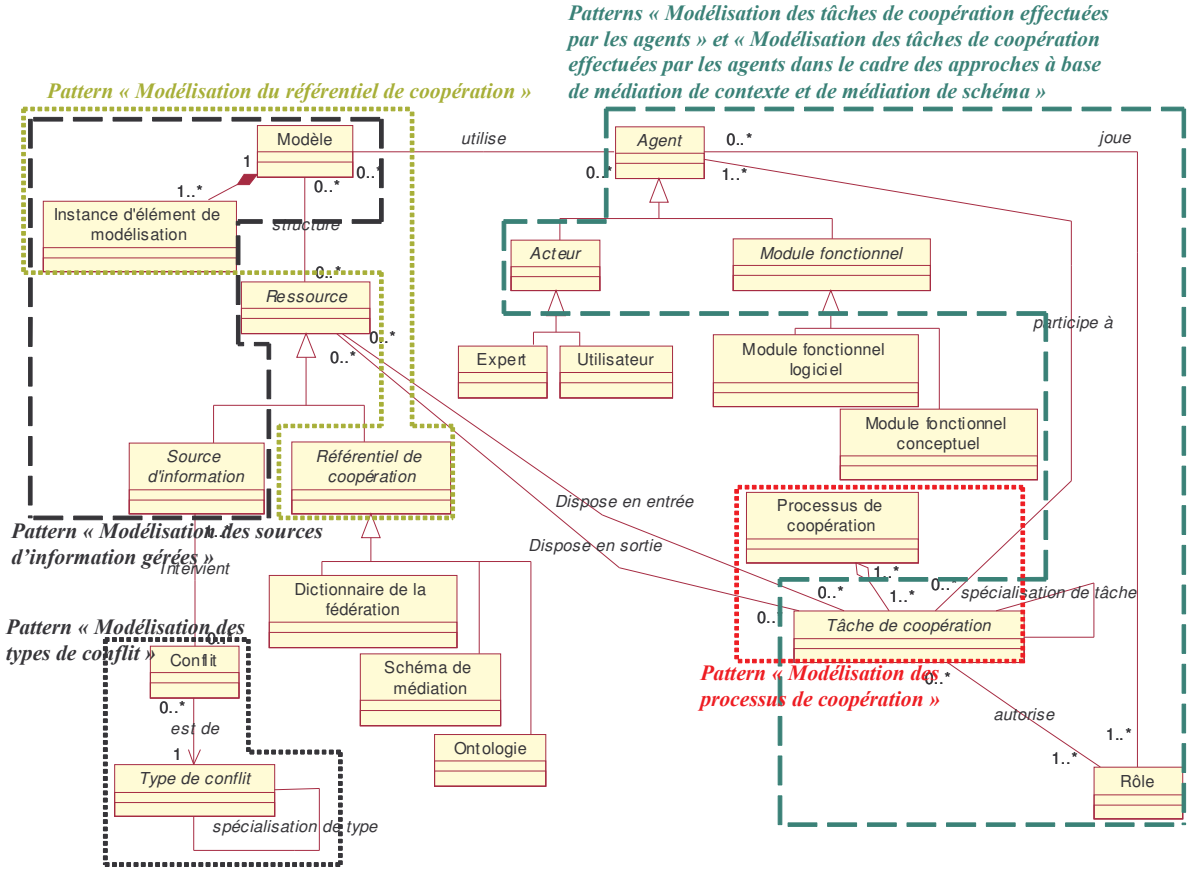


Figure 38 : Patterns d'analyse dérivés du modèle du domaine de la coopération de systèmes d'information

Nous avons conçu six patterns d'analyse, à savoir :

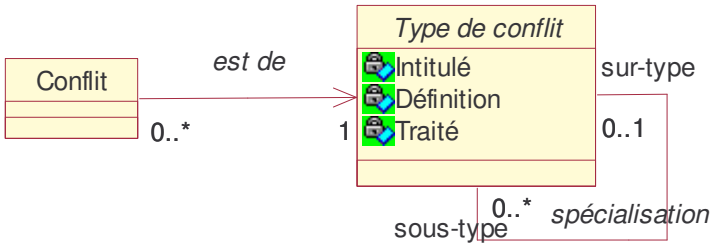
- Le pattern *"Modélisation des types de conflit"* (Cf. Tableau 7) catégorise et précise les définitions des différents conflits usités dans le système d'information coopératif à développer. Il permet également d'indiquer quels types de conflits seront traités par le SIC.
- Le pattern *"Modélisation des sources d'information gérées"* (Cf. Tableau 8) permet de représenter le modèle et le type des sources d'information composant le SIC.
- Le pattern *"Modélisation du référentiel de coopération*" (Cf. Tableau 9) permet de représenter le référentiel de coopération (modèle et type) utilisé au sein du SIC ou défini par une approche de coopération spécifique.
- Le pattern *"Modélisation des tâches de coopération effectuées par les agents"* (Cf. Tableau 10) permet de représenter les tâches de coopération que le système d'information coopératif ou l'approche de coopération effectue et, éventuellement, les différents agents intervenant lors de ces tâches. Remarquons que ce pattern n'intègre pas la dimension technologique inhérente à notre référentiel du domaine (entité "Module conceptuel logiciel"), du fait que cette dimension ne peut intervenir lors de la phase d'analyse du SIC.
- Le pattern *"Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas"* (Cf. Tableau 11) est un raffinement du pattern "Modélisation des tâches de coopération effectuées par les agents" dans le cadre du choix de conception d'un SIC de type médiation de schémas ou médiation de contextes.
- Le pattern *"Modélisation des processus de coopération"* (Cf. Tableau 12) permet de représenter les processus de coopération, c'est-à-dire les regroupements ordonnés de tâches de coopération spécifiés dans le SIC.

Ces six patterns sont présentés dans la section suivante (§ 3.2). Nous précisons, ensuite, leur processus de réutilisation et introduisons, à cet effet, deux patterns de support d'utilisation (§ 3.3).

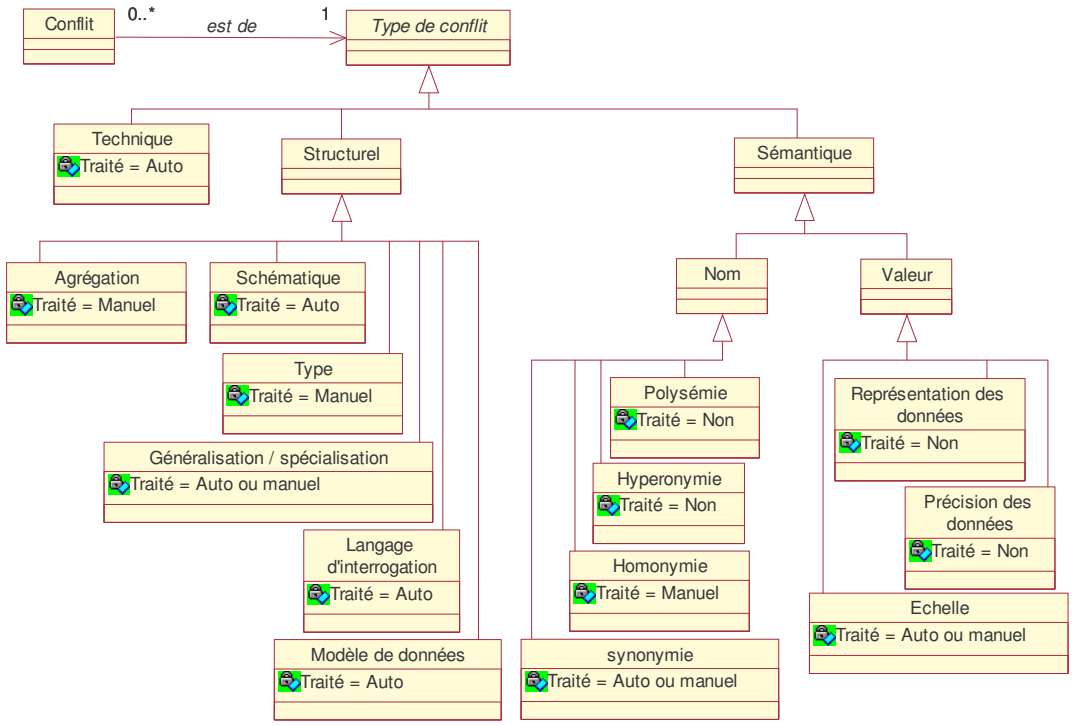
3.2. Présentation des patterns d'analyse de coopération

Les différents patterns d'analyse sont présentés de façon détaillée dans les sections suivantes (§ 3.2.1 à 3.2.6).

3.2.1. Pattern d'analyse de coopération « Modélisation des types de conflit »

Interface
Nom
Modélisation des types de conflit
Synonymes
Conflit_Type de conflit
Classification
Pattern d'analyse de coopération
Objectif
Ce pattern permet de catégoriser et de préciser les définitions des différents conflits spécifiés dans un système d'information coopératif. Il permet également de déterminer quels types de conflits sont traités par le SIC.
Applicabilité
SIC
Solution
Modèle

Participants
Conflit : hétérogénéité entre sources d'information participant au système coopératif. *Type de conflit :* précise le type de conflit défini au sein du SIC, et plus particulièrement son intitulé, sa description, et s'il est traité ou non par le SIC. Un conflit peut être traité automatiquement (« Traité = Auto »), peut nécessiter une intervention humaine (intervention d'un expert et/ou d'un utilisateur, par exemple) (« Traité = Manuel ») ou non pris en compte par le SIC (« Traité = Non »). Un conflit traité signifie que le SIC apporte des concepts et/ou techniques permettant la prise en compte de ce conflit.
Exemples d'utilisation
Exemple 1
Nom de l'exemple
Conflits liés au projet ACSIS [DUBOIS, 1997] [SEGURAN, 2003]

Portée
SIC médiation de schémas
Modèle



Participants
Conflit technique : est dû à l'hétérogénéité des plates-formes, i.e. des systèmes d'exploitation et des systèmes matériels, sur lesquelles sont situés les systèmes composants. *Conflit structurel :* traduit entre les systèmes locaux une différence de modèle de données, de langage de requêtes ou de représentation des informations entre composants coopérants. *Conflit de modèle de données :* est dû à une différence de modèle de données entre les systèmes locaux (e.g. codasyl, relationnel, ou orienté objet). *Conflit de langage d'interrogation :* se réfère à une différence de langages de requêtes entre les systèmes locaux (e.g. DML Codasyl, SQL ou OQL). *Conflit schématique :* se réfère à l'utilisation de concepts différents pour représenter la même information. Par exemple, une information peut être représentée par plusieurs attributs d'une table dans un composant ou par une table dans un autre. *Conflit de généralisation/spécialisation :* est lié aux différences de hiérarchisation d'une même information (utilisation de plusieurs concepts pour représenter une information ou d'un concept unique plus général). Par exemple, un hôpital pourra définir plusieurs entités pour représenter ses types d'employés (infirmiers, médecins, administratifs), alors qu'un autre définira une unique entité pour gérer tous ses employés. *Conflit d'agrégation :* concerne les niveaux de granularité différents entre les systèmes. Ainsi la

valeur d'un attribut d'un système composant pourra être équivalent à l'agrégation des valeurs de plusieurs attributs dans un autre système. Par exemple, l'attribut « adresse » dans un système pourra être décomposé en plusieurs attributs « numéro de rue, nom de la rue, code postal, ville » dans un autre système.

Conflit de type : se réfère à des différences de types de données entre attributs porteurs de la même information. Par exemple, une date pourra être représentée sous la forme d'une chaîne de caractères dans un système composant et par un type « Date » dans un autre composant.

Conflit sémantique : représente les conflits de domaine de définition, mais aussi les différences dans la signification ou l'interprétation d'une même donnée.

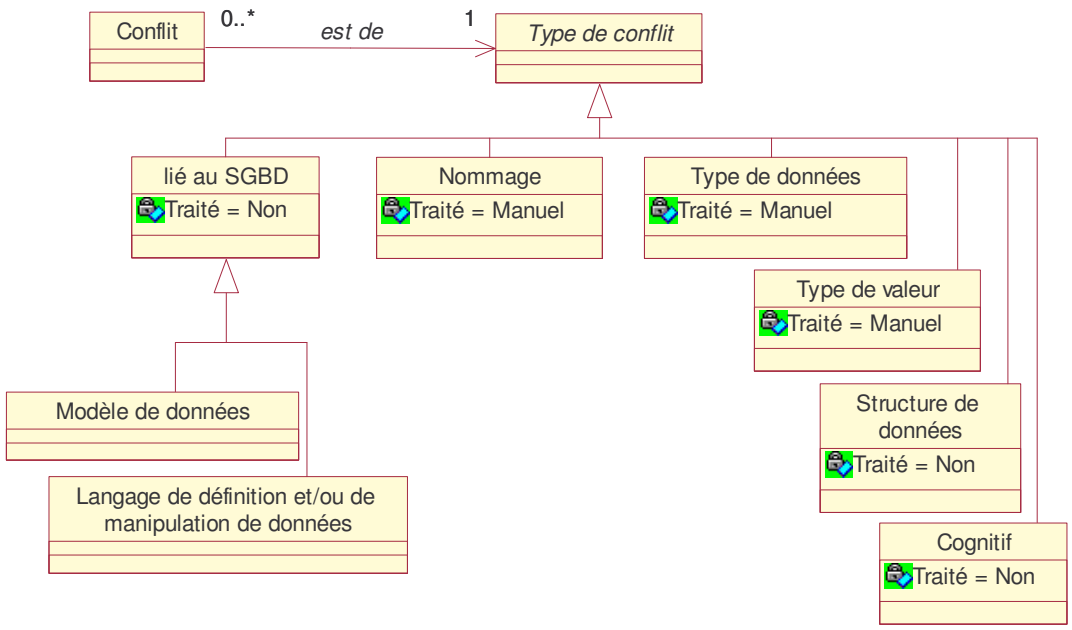
Conflit de nom : relève de l'autonomie des systèmes à nommer les informations. Les conflits de noms peuvent être des conflits de *synonymie* (des termes différents expriment la même information), d'*homonymie* (des informations différentes possèdent le même nom), de *polysémie* (un même terme change de signification en fonction du contexte), d'*hyperonymie* et d'*hyponymie* (un terme peut être plus général ou plus spécifique qu'un ou plusieurs autres termes).

Conflit de valeur : est lié à des manières différentes de coder la même information dans des systèmes différents. Plusieurs conflits de valeurs ont été recensés. Les *conflits de représentation des données* résultent de l'utilisation de valeurs différentes pour représenter la même information. Par exemple, les valeurs de l'attribut « permis de conduire » pourront être {« oui », « non »} dans un système alors qu'elles seront {0,1} dans un autre. Les *conflits de précision des données* surviennent quand une même information a un degré de précision différent dans des systèmes composants (par exemple, une note sera représentée par une valeur allant de 0 à 20 dans un système alors qu'elle sera représentée par une lettre {A, B, C, D, E} dans un autre). Enfin, les *conflits d'échelle* relèvent de la représentation d'une même information en utilisant des unités ou mesures différentes (par exemple, une taille peut être représentée en centimètres et en pouces dans des systèmes différents).

Les conflits de généralisation, ainsi que les conflits de synonymie et d'échelle sont soit traités automatiquement, soit nécessitent une intervention humaine (d'un utilisateur ou d'un expert).

Exemple 2
Nom de l'exemple
Conflits liés au projet MRDSM [LITWIN & AL., 1982] [LITWIN, 1985]
Portée
SIC fédéré

Modèle



Participants

Conflit lié au SGBD : lié aux différences de modèles de données et de langages d'interrogation entre les SGBD locaux.

Conflit de modèle de données : différence de modèle de données entre les SGBD.

Conflit de langage de définition et/ou de manipulation de données : se réfère aux différences de langages de définition et de manipulation des données entre les SGBD locaux.

Conflit de nommage : utilisation de noms différents pour représenter le même concept.

Conflit de type de données : se réfère à des différences de types de données entre attributs porteurs de la même information.

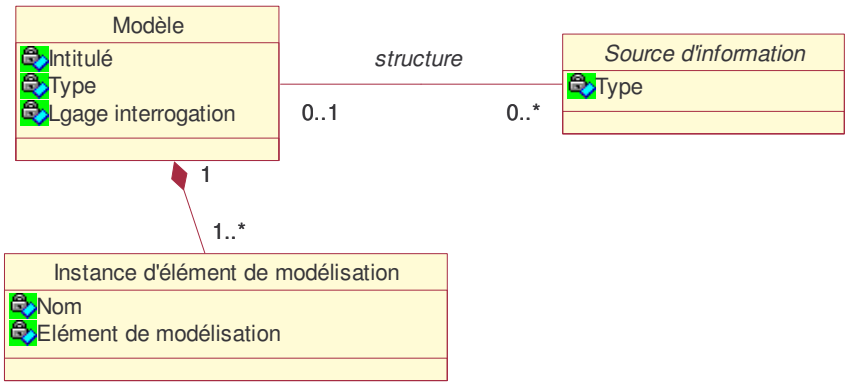
Conflit de type de valeur : est lié à des manières différentes de coder la même information dans des systèmes différents (conflit d'unité ou de mesure, conflit de précision des données).

Conflit de structure de données : se réfère à l'utilisation de concepts différents pour représenter la même information. Par exemple, une information peut être représentée par plusieurs attributs d'une table dans un composant ou par une table dans un autre.

Conflit cognitif : hétérogénéité due à des appréciations subjectives. « Un même concept ou une valeur peut relever de plusieurs vérités » [LITWIN & AL., 1982].

Tableau 7 : Pattern d'analyse de coopération « Modélisation des types de conflit »

3.2.2. Pattern d'analyse de coopération « Modélisation des sources d'information gérées »

Interface
Nom
Modélisation des sources d'information gérées
Synonymes
Source d'information_Modèle_Instance d'élément de modélisation
Classification
Pattern d'analyse de coopération
Objectif
Ce pattern permet de représenter les sources d'information (modèle et type) prises en compte par le système d'information coopératif.
Applicabilité
SIC
Solution
Modèle

Participants
Source d'information : entité participant au système coopératif. Elle est caractérisée par son type (structuré, semi-structuré ou non structuré). *Modèle :* Un modèle définit un langage pour spécifier les informations du domaine. Il contient des instances d'éléments de modélisation pour représenter les abstractions des sources d'information. Un modèle est caractérisé par son type (objet, relationnel, propriétaire,…) et, éventuellement, le langage d'interrogation associé (SQL, OQL, KIF,…). *Instance d'élément de modélisation :* permet de représenter les abstractions du système en cours de modélisation. Un élément de modélisation est par exemple, une classe, un type d'objet, un attribut, une opération, une entité,… Les instances d'éléments de modélisation constituant le modèle servent à définir les spécificités des modèles propriétaires.

A un participant commun avec

Modèle : Modélise le modèle (et le type) des référentiels de coopération [Modélisation du référentiel de coopération]

Instance d'élément de modélisation : Modélise les spécificités des modèles propriétaires associés aux référentiels de coopération [Modélisation du référentiel de coopération]

Exemples d'utilisation

Exemple 1

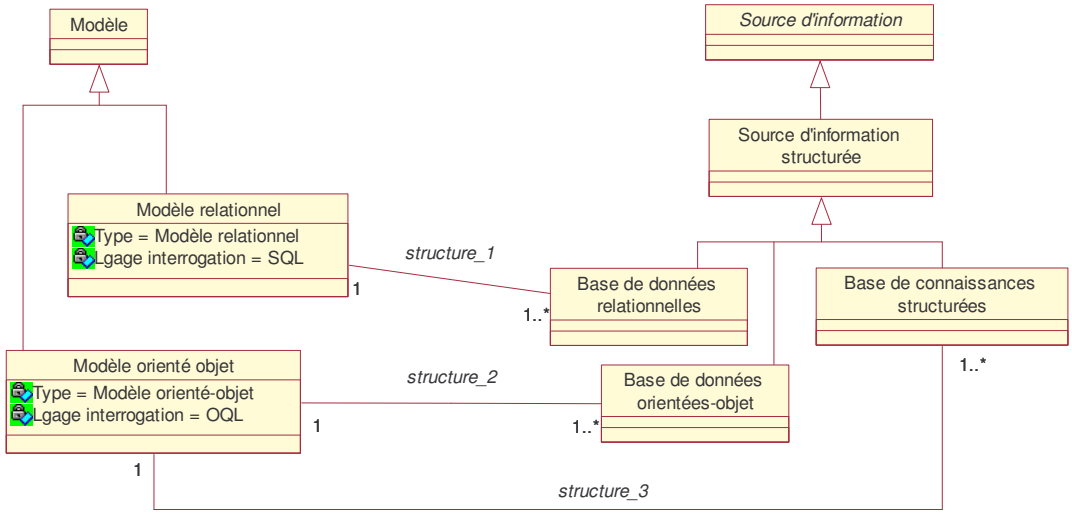
Nom de l'exemple

Sources d'information gérées par le projet ACSIS [DUBOIS, 1997]

Portée

SIC médiation de schémas

Modèle



Participants

Le projet coopératif ACSIS ne gère que les sources d'information structurées de type base de connaissances (dont les connaissances sont représentées par un modèle orienté-objet) ou base de données (bases de données relationnelles et orientées-objet).

Source d'information structurée : source dont les informations sont structurées par un modèle.

Base de données relationnelles : base de données structurée selon un modèle permettant de dresser des relations entre les différentes entités.

Base de données orientée-objet : base dont les données sont représentées sous formes d'objets.

Base de connaissances : collection d'information exprimée avec un langage formel de représentation.

Modèle relationnel : Modèle basé sur une organisation des données sous forme de tables et de relations entre ces tables.

Modèle orienté-objet : Modèle basé sur une organisation des données sous forme d'objets.

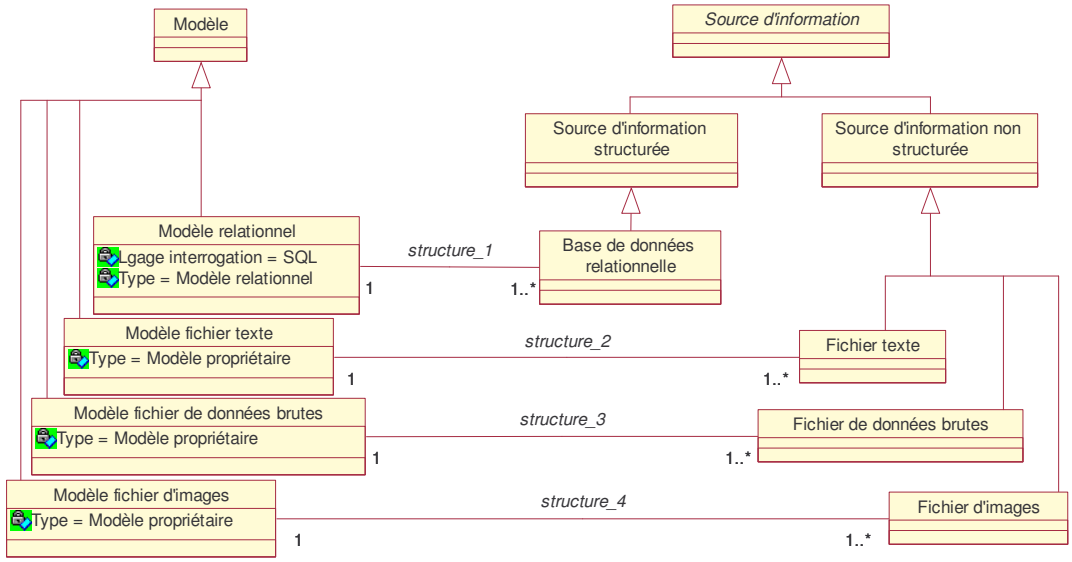
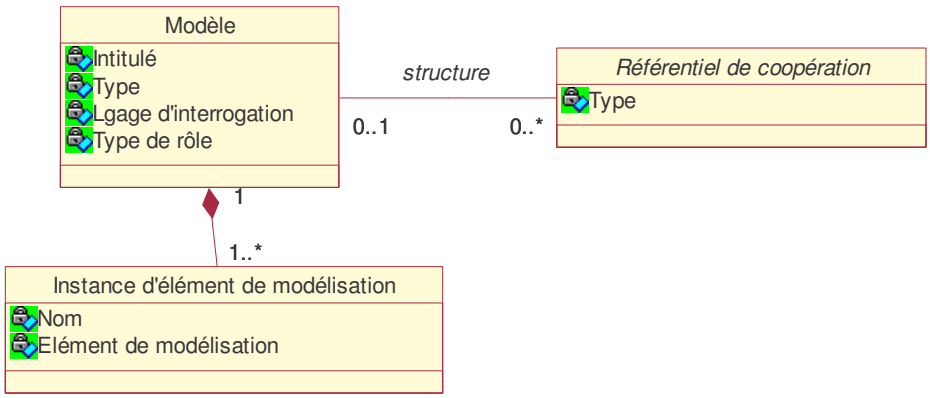
Exemple 2
Nom de l'exemple
Sources d'information gérées par le projet Infosleuth [NODINE & AL., 2000]
Portée
SIC médiation de contextes
Modèle

Participants
Le projet InfoSleuth gère les bases de données relationnelles, ainsi que les sources d'information non structurées de types fichier texte, fichier d'images[81] et fichier de données brutes. Un modèle est défini pour chaque source d'information. Dans le cadre des sources d'information non structurées, un modèle propriétaire est défini a posteriori permettant de créer un schéma de données spécifique (utilisation d'outils d'indexation, de techniques de parsing,…). REMARQUE — les instances d'éléments de modélisation des modèles propriétaires n'ont pas été explicitées dans le cadre du projet Infosleuth.

Tableau 8 : Pattern d'analyse de coopération « Modélisation des sources d'information gérées »

[81] InfoSleuth considère les fichiers d'images comme des sources d'information non structurées.

3.2.3. Pattern d'analyse de coopération « Modélisation du référentiel de coopération »

Interface
Nom
Modélisation du référentiel de coopération
Synonymes
Référentiel de coopération_Modèle_Instance d'élément de modélisation
Classification
Pattern d'analyse de coopération
Objectif
Ce pattern permet de représenter le référentiel de coopération (modèle et type) utilisé au sein du SIC ou défini par une approche de coopération spécifique.
Applicabilité
SIC ^ Approche
Solution
Modèle

Participants
Référentiel de coopération : ressource assurant le stockage de toutes les informations utiles à la coopération (schéma global, extensions du langage multibase, concepts ontologiques,…). Un référentiel de coopération est éventuellement caractérisé par son type ({dictionnaire de la fédération, schéma de médiation, ontologie}). *Modèle :* contient des instances d'éléments de modélisation pour représenter les abstractions des sources d'information. Un modèle est caractérisé par son type, éventuellement, par son type de rôle (modèle canonique, modèle d'ontologie, modèle de médiation,…) et par le langage d'interrogation associé (SQL, OQL, KIF,…). *Instance d'élément de modélisation :* permet de représenter les abstractions du système en cours de modélisation. Un élément de modélisation est par exemple, une classe, un type d'objet, un attribut,

une opération, une entité,… Les instances d'éléments de modélisation constituant le modèle servent à définir les spécificités des modèles propriétaires.

A un participant commun avec

Modèle : Modélise le modèle (et le type) des sources d'information coopérantes [Modélisation des sources d'information gérées]

Instance d'élément de modélisation : Modélise les spécificités des modèles propriétaires associés aux sources d'information coopérantes [Modélisation des sources d'information gérées]

Exemples d'utilisation

Exemple 1

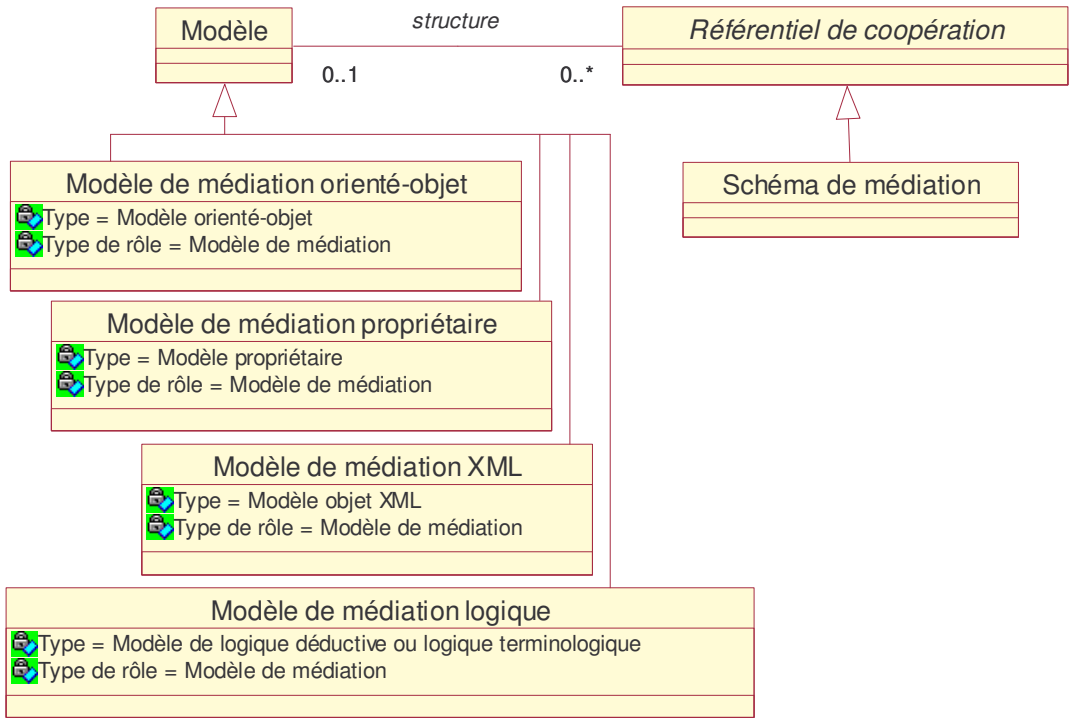
Nom de l'exemple

Référentiel de coopération dans le cadre de l'approche de type médiation de schémas

Portée

Approche médiation de schémas

Modèle



Participants

Schéma de médiation : référentiel, utilisé dans le cadre des projets de type médiation de schémas, spécifié à l'aide d'un modèle de médiation et représentant les méta-données des sources d'information locales, ainsi que des concepts sémantiques entre méta-données.

Modèle de médiation : modèle permettant d'homogénéiser les modèles de données locaux, mais aussi de prendre en compte la sémantique des données. Ce modèle peut-être un modèle orienté-

objet, objet XML, logique (terminologique ou déductif) ou un modèle propriétaire.

REMARQUE — le modèle relationnel n'a pas été retenu comme modèle de médiation car il ne permet pas d'exprimer la sémantique des informations. Cependant, le modèle de médiation peut être un modèle relationnel étendu. Dans ce cas, il sera considéré comme un modèle propriétaire.

Exemple 2

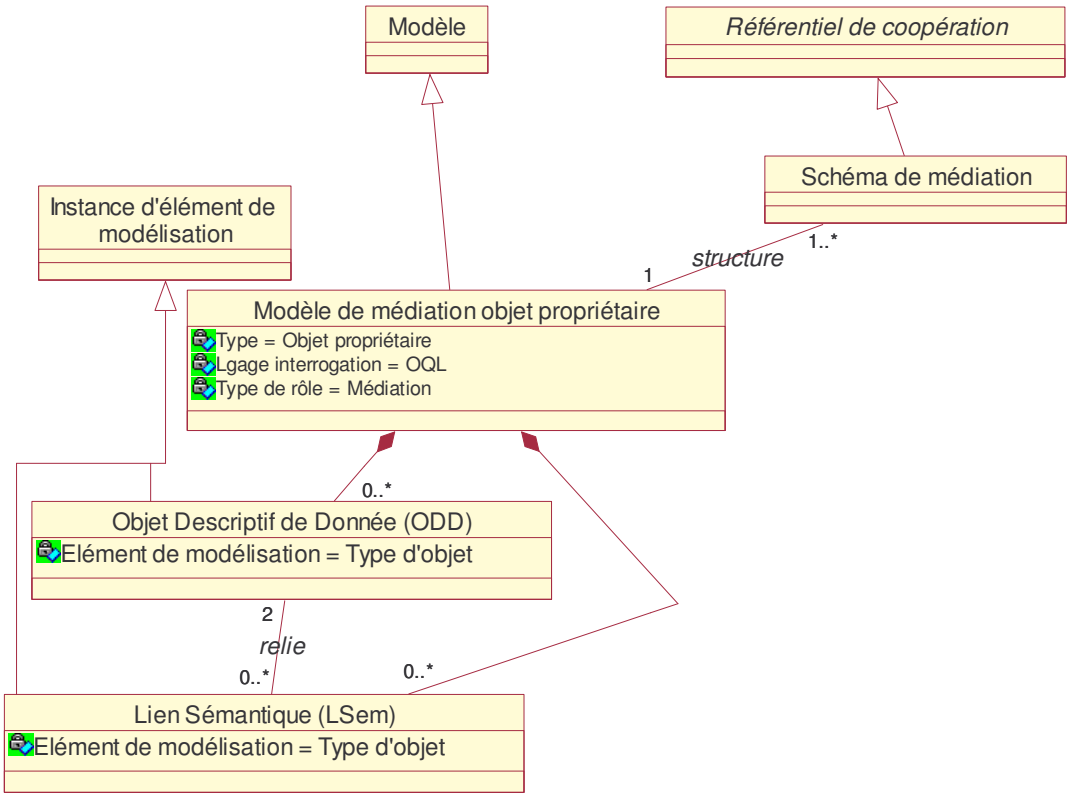
Nom de l'exemple

Référentiel de coopération dans le cadre du projet ACSIS [DUBOIS, 1997] [SEGURAN, 2003]

Portée

SIC médiation de schémas

Modèle



Participants

Schéma de médiation : référentiel, utilisé dans le cadre des projets de type médiation de schémas, spécifié à l'aide d'un modèle de médiation et représentant les méta-données des sources d'information locales, ainsi que des concepts sémantiques entre méta-données.

Modèle de médiation objet propriétaire : modèle permettant d'homogénéiser les schémas des sources d'information locales, mais aussi de prendre en compte la sémantique des données. Ce modèle définit deux types d'objets particuliers, les objets descriptifs des données et les liens sémantiques.

Objet Descriptif de Donnée (ODD) : type d'objet décrivant un élément du modèle de données local (par exemple, pour le modèle relationnel : une table, un attribut, un attribut clé primaire,…). L'ensemble des ODD fournit une représentation homogène des données accessibles via la coopération. *Lien sémantique (LSem) :* type d'objet permettant d'exprimer une relation sémantique orientée entre deux ODD (un ODD "source" et un ODD "cible"). Les LSem assurent ainsi l'appariement des objets descriptifs des données en fonction de caractéristiques structurelles ou sémantiques (exemples de liens sémantiques : généralisation, spécialisation, homonymie, synonymie,…).

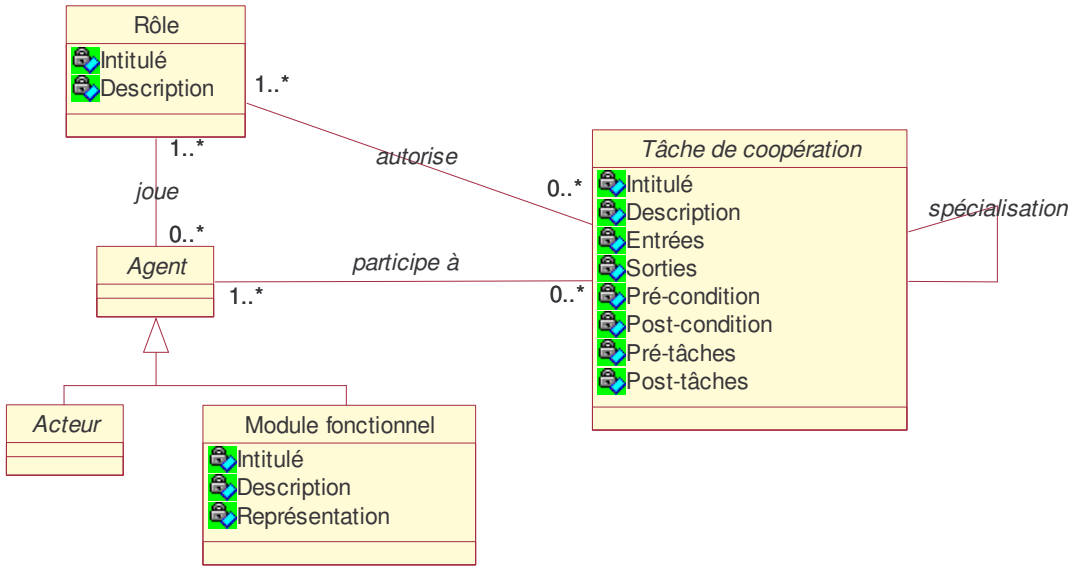
Tableau 9 : Pattern d'analyse de coopération « Modélisation du référentiel de coopération »

3.2.4. Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents »

Interface
Nom
Modélisation des tâches de coopération effectuées par les agents
Synonymes
Agent_Tâche de coopération_Rôle
Classification
Pattern d'analyse de coopération
Objectif
Ce pattern permet de modéliser les tâches de coopération que le système d'information coopératif ou l'approche de coopération effectue et, éventuellement, les différents agents intervenant lors de ces tâches.
Applicabilité
SIC ^ Approche

Solution

Modèle



Participants

Tâche de coopération : définit un événement de coopération à traiter par l'association entre un ensemble d'entrées, c'est-à-dire de données à traiter, et un ensemble de sorties, c'est-à-dire de résultats produits, et par l'indication de leurs caractéristiques respectives. Une tâche de coopération peut-être décomposée en sous-tâches représentant ainsi les différents événements ou sous-événements à traiter. Elle est caractérisée par un intitulé, une description et éventuellement les flux entrants et sortants (vis-à-vis des sources d'information ou du référentiel de coopération), les pré et post-conditions, ainsi que les pré et post-tâches.

Agent : assure la réalisation ou le contrôle d'une ou plusieurs tâches de coopération. Un agent peut-être composé ou responsable d'un ou plusieurs agents. Un agent peut être un individu (ou acteur) du fait que certaines tâches de coopération peuvent nécessiter des interventions manuelles ou un module fonctionnel réalisant une ou plusieurs tâches de coopération. Il peut avoir un ou plusieurs buts ou rôles dans un processus de coopération.

Acteur : individu intervenant dans une tâche coopérative. Cet individu est en général soit un expert, soit un utilisateur.

Module fonctionnel : entité (conceptuelle) spécifiée dans le cadre d'une approche coopérative ou d'un SIC réalisant une tâche de coopération. Un module fonctionnel est caractérisé par un intitulé, une description et éventuellement sa représentation (agent artificiel, objet,…).

Rôle : fonction, responsabilité, autorisation ou contrôle assigné à un agent pour l'exécution d'une ou plusieurs tâches de coopération. Un rôle peut être attribué à un ou plusieurs agents pour une période donnée et est fonction de la tâche de coopération.

A un participant commun avec

Tâche de coopération : représentation de la liste ordonnancée des tâches d'un processus de coopération [Modélisation des processus de coopération]

Exemples d'utilisation

Exemple 1

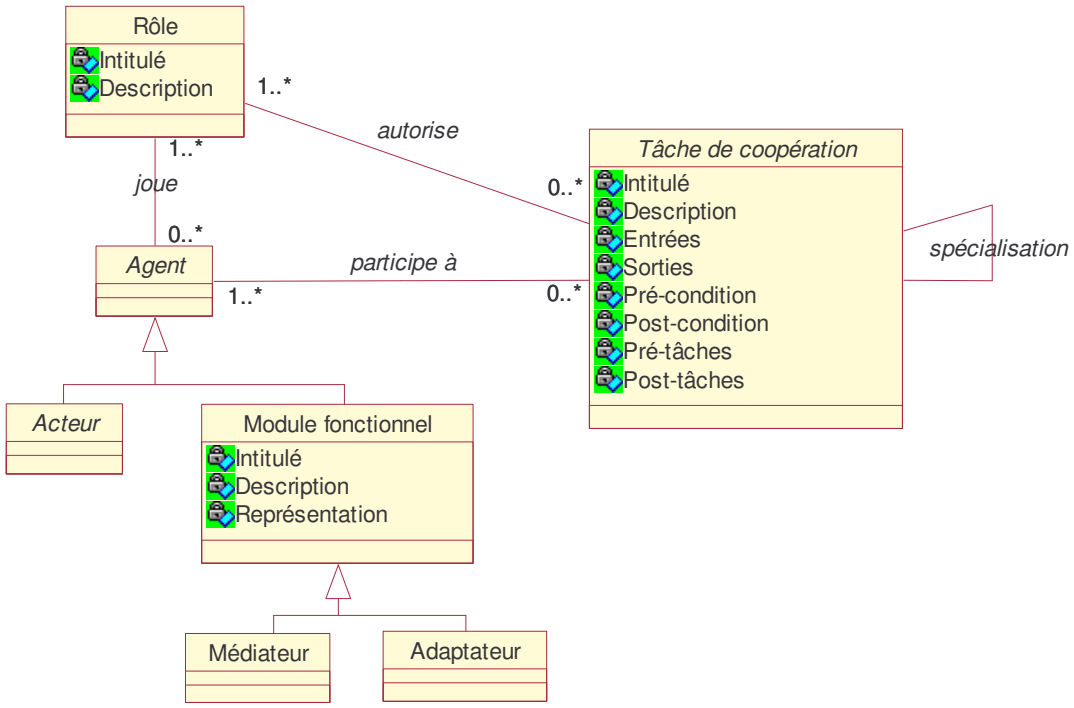
Nom de l'exemple

Tâches de coopération effectuées par les agents dans le cadre des approches de type médiation de schémas et médiation de contextes.

Portée

Approche médiation de schémas ^ Approche médiation de contextes

Modèle



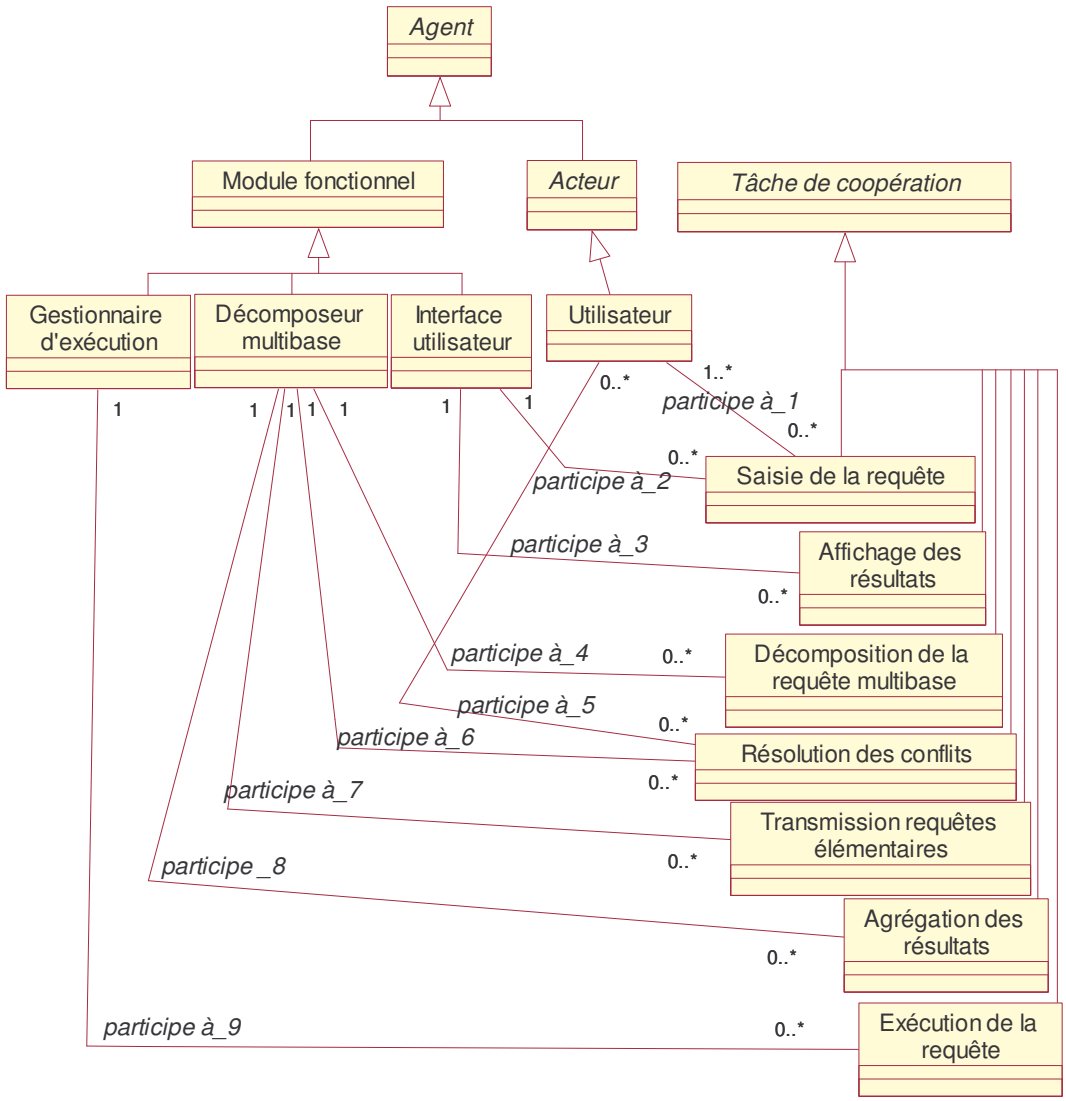
Participants

Les approches coopératives de type médiation de schémas et médiation de contextes ne définissent pas explicitement de tâches de coopération particulières, ni de type d'acteur mais spécifient deux modules fonctionnels assurant tout ou partie des tâches coopératives.

Médiateur : composant chargé de répondre à des besoins à partir de connaissances mises à sa disposition. Le médiateur permet de localiser l'information et de traiter certains conflits structurels et sémantiques.

Adaptateur : composant servant d'interface avec les sources d'information. L'adaptateur traite les conflits de modèle de données et de langage d'interrogation en présentant les données dans le modèle de médiation.

Exemple 2
Nom de l'exemple
Tâches de coopération effectuées par les agents dans le cadre du projet MRDSM [LITWIN, 1985] [LITWIN, 1988].
Portée
SIC fédéré
Modèle



Participants
<u>*Utilisateur :*</u> Acteur à l'origine de la saisie de la requête multibase et intervenant lors de la résolution des conflits. <u>*Interface utilisateur :*</u> composant permettant la création de la requête multibase à l'aide du langage de manipulation multibase dont la description est contenue dans le dictionnaire de la fédération (également appelé référentiel multibase).

Décomposeur multibase : détermine les noms des bases de données composantes qui doivent être ouvertes (grâce au dictionnaire de la fédération), redirige les requêtes n'impliquant qu'une seule base vers les bases correspondantes, décompose la requête multibase en requêtes élémentaires qui seront adressées aux bases locales respectives et résout les conflits de nommage, de type de données et de type de valeurs. *Gestionnaire d'exécution :* requiert et contrôle l'exécution de la requête élémentaire. *Saisie de la requête :* tâche de création de la requête utilisateur soumise à la multibase. *Décomposition de la requête multibase :* décomposition de la requête multibase (i.e., concernant plusieurs bases de données). *Résolution des conflits :* consiste principalement à utiliser et définir manuellement des concepts et techniques (« variables sémantiques », « attributs dynamiques »,…), stockées dans le référentiel de coopération, qui seront ensuite pris en compte par le système lors de la résolution des conflits. Cette résolution concerne les conflits de nommage, de type de données et de type de valeurs. *Transmission requêtes élémentaires :* transmission des requêtes élémentaires, obtenues après décomposition de la requête multibase, au gestionnaire d'exécution. *Exécution de la requête :* traitement de la requête sur la base de données locale. *Agrégation des résultats :* consiste à appliquer les clauses « where » et les jointures définies dans les requêtes et à restructurer les résultats. *Affichage des résultats :* tâche permettant l'affichage des résultats « agrégés » à l'utilisateur.

Tableau 10 : Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents »

3.2.5. Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas »

Interface
Nom
Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas.
Classification
Pattern d'analyse de coopération
Objectif
Ce pattern permet de modéliser les tâches de coopération que le système d'information coopératif (de type médiation de schémas ou médiation de contextes) effectue, ainsi que les différents agents

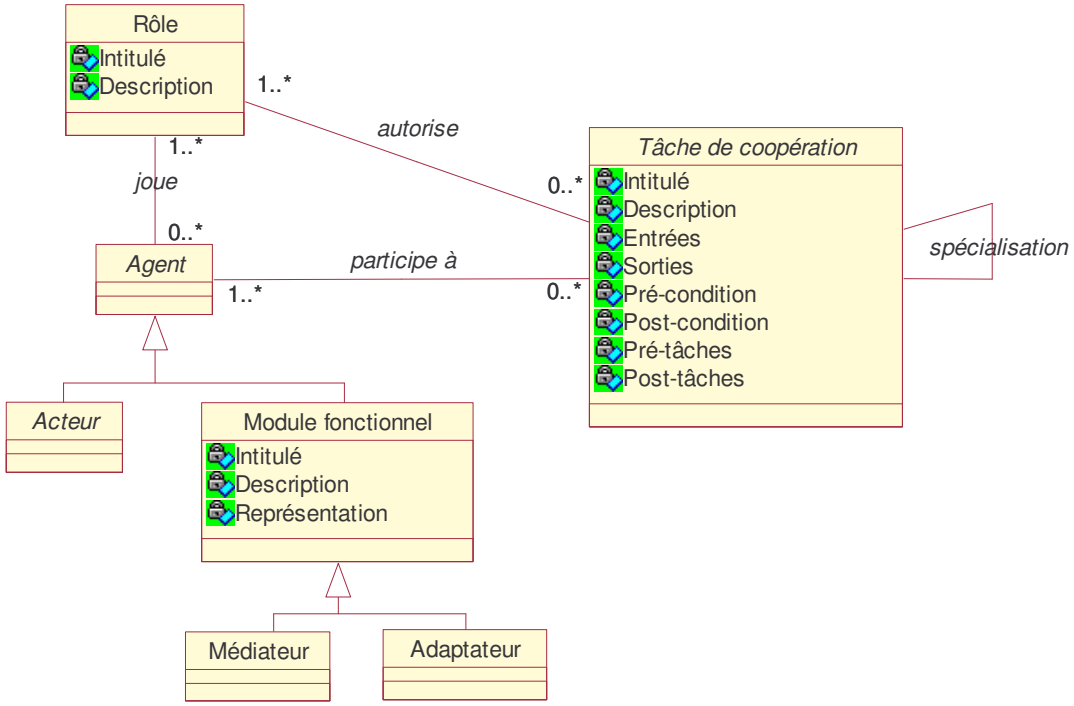
intervenant lors de ces tâches.

Applicabilité

SIC médiation de schémas ^ SIC médiation de contextes

Solution

Modèle



Participants

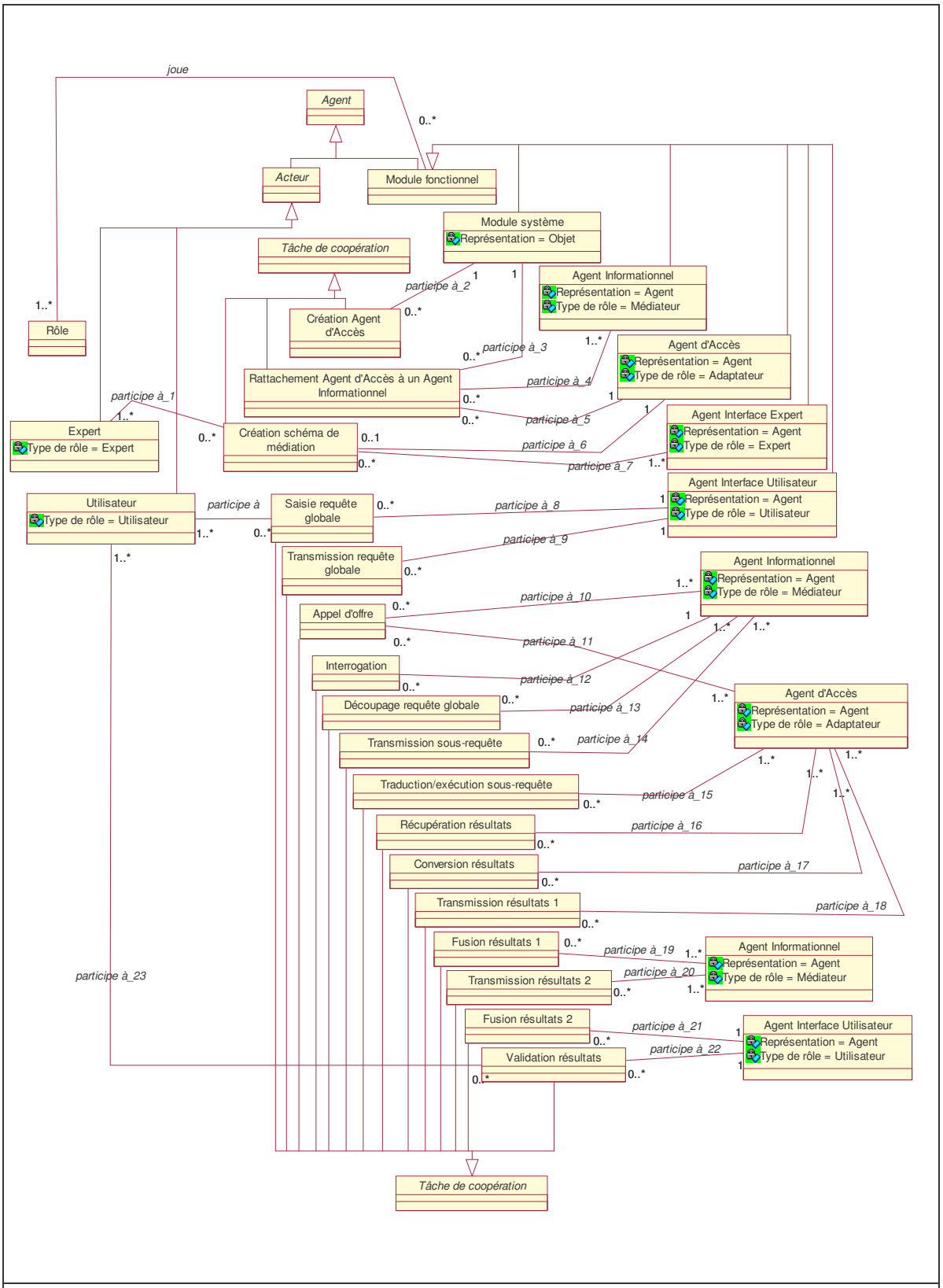
Tâche de coopération : définit un événement de coopération à traiter par l'association entre un ensemble d'entrées, c'est-à-dire de données à traiter, et un ensemble de sorties, c'est-à-dire de résultats produits, et par l'indication de leurs caractéristiques respectives. Une tâche de coopération peut-être décomposée en sous-tâches représentant ainsi les différents événements ou sous-événements à traiter. Elle est caractérisée par un intitulé, une description et éventuellement les flux entrants et sortants (vis-à-vis des sources d'information ou du référentiel de coopération), les pré et post-conditions, ainsi que les pré et post-tâches.

Agent : assure la réalisation ou le contrôle d'une ou plusieurs tâches de coopération. Un agent peut être composé ou responsable d'un ou plusieurs agents. Un agent peut être un individu (ou acteur) du fait que certaines tâches de coopération peuvent nécessiter des interventions manuelles ou un module fonctionnel réalisant une ou plusieurs tâches de coopération. Il peut avoir un ou plusieurs buts ou rôles dans un processus de coopération.

Acteur : individu intervenant dans une tâche coopérative. Cet individu est en général soit un expert, soit un utilisateur.

Module fonctionnel : entité (conceptuelle) spécifiée dans le cadre d'une approche coopérative ou

d'un SIC réalisant une tâche de coopération. Un module fonctionnel est caractérisé par un intitulé, une description et éventuellement sa représentation (agent artificiel, objet,…). *Médiateur :* composant chargé de répondre à des besoins à partir de connaissances mises à sa disposition. Le médiateur permet de localiser l'information et de traiter certains conflits structurels et sémantiques. *Adaptateur :* composant servant d'interface avec les sources d'information. L'adaptateur traite les conflits de modèle de données et de langage d'interrogation en présentant les données dans le modèle de médiation. *Rôle :* fonction, responsabilité, autorisation ou contrôle assigné à l'agent pour l'exécution d'une ou plusieurs tâches de coopération. Un rôle peut être attribué à un ou plusieurs agents pour une période donnée et est fonction de la tâche de coopération.
A un participant commun avec
Tâche de coopération : représentation de la liste ordonnancée des tâches d'un processus de coopération [Modélisation des processus de coopération]
Exemples d'utilisation
Exemple 1
Nom de l'exemple
Tâches de coopération effectuées par les agents dans le cadre du projet de médiation de schémas ACSIS [DUBOIS, 1997] [SEGURAN, 2003].
Portée
SIC médiation de schémas
Modèle



Participants

Agents :

Acteur : deux types d'acteur (rôles) interviennent dans le processus de coopération : l'expert du domaine et l'utilisateur.

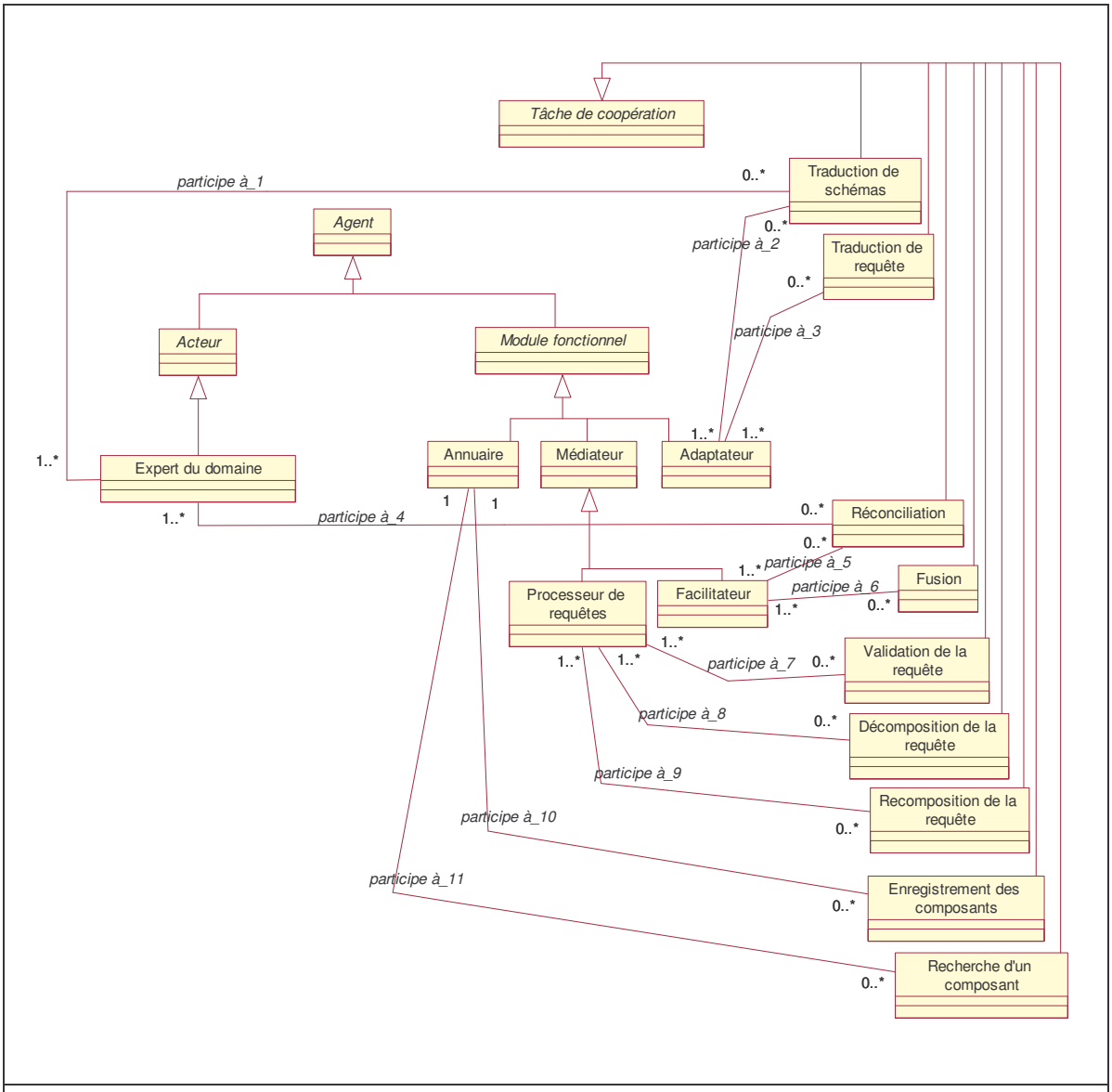
Agent Informationnel : agent multi-domaines (regroupant des domaines proches sémantiquement) découpant les requêtes globales en sous-requêtes et récupérant les résultats provenant des agents d'accès pour obtenir une solution globale.
Agent d'accès : assure la participation des données locales au sein des processus coopératifs. Il traduit une requête exprimée dans le langage global d'interrogation en requête exprimée dans le langage propre à la base locale dont il a la charge, exécute cette dernière, réceptionne et renvoie les résultats de la base locale.
Agent Interface (rôle d'Utilisateur ou d'Expert) : sert d'intermédiaire entre les utilisateurs ou les experts du domaine et les autres agents du système. Chaque rôle correspond à un ensemble de permissions sur le système et est représenté par une spécialisation de l'*Agent Interface*. L'*Agent Interface Utilisateur* est gestionnaire de la requête et l'exécute sans se préoccuper de la localisation des données. Il permet la validation des résultats et peut déclencher la ré-exécution de la requête. L'*Agent Interface Expert* a une expertise sur le domaine de la base locale (définition de certains liens sémantiques permettant le traitement des conflits).

Tâche de coopération :

L'enregistrement d'une source d'information locale consiste à créer un Agent d'Accès (AA - agent jouant le rôle d'adaptateur) chargé d'encapsuler cette source, à créer le schéma de médiation associé à l'AA , i.e. à créer un schéma de médiation constitué du graphe d'Objets Descriptifs des Données et des Liens Sémantiques (LSem) qui permettront de traiter les conflits (les liens sémantiques sont constitués automatiquement ou définis manuellement par l'utilisateur) et à rattacher l'agent d'accès à un agent informationnel (AI) existant ou à un AI nouvellement créé.

Le traitement d'une requête globale comprend plusieurs tâches :
- *Saisie requête globale* : saisie de la requête soumise à la coopération par l'utilisateur à l'aide de l'agent d'interface (type de rôle : utilisateur).
- *Transmission requête globale* : l'Agent d'Interface (type de rôle : utilisateur) transmet la requête globale à un Agent Informationnel (AI).
- *Appel d'offre* : l'agent informationnel transmet le profil sémantique de la requête globale aux agents d'accès qui lui sont rattachés. Chaque AA évalue les objets descriptifs des données et les liens sémantiques du schéma de médiation qui lui est associé et détermine ainsi sa capacité à traiter la requête et en avertit l'AI associé.
- *Interrogation :* dans le cas de l'incapacité des AA rattachés à l'AI de répondre à la requête, l'AI contacte les autres AI disponibles et la tâche d'appel d'offre est à nouveau effectuée pour chaque AI contacté.
- *Découpage requête globale :* l'AI découpe la requête globale en fonction des AA ayant la

capacité de répondre à celle-ci. - *Transmission sous-requête :* l' (ou éventuellement les) AI transmet chaque sous-requête à un AA capable de la traiter. - *Traduction et exécution sous-requête* : chaque AA traduit et exécute la sous-requête sur la source d'information associée. - *Récupération des résultats* par l'AA. - *Conversion résultats :* conversion des résultats dans le cas de l'identification de conflits d'échelle en fonction de l'unité de mesure précisée dans la requête. - *Transmission résultats 1* : transmission des résultats à l'AI associé. - *Fusion résultats 1 :* chaque AI fusionne les données. - *Transmission résultats 2 :* envoie des résultats à l'agent d'interface (rôle : utilisateur). - *Fusion résultats 2 :* fusion des données obtenues de chaque AI par l'agent d'interface. - *Validation résultats :* l'utilisateur valide les résultats qui lui sont présentés par l'agent d'interface (rôle : utilisateur).
Exemple 2
Nom de l'exemple
Tâches de coopération effectuées par les agents dans le cadre du projet de médiation de contextes DILEMMA [JOUANOT, 2000] [JOUANOT, 2001] [CULLOT & AL., 2003].
Portée
SIC médiation de contextes
Modèle



Participants

Acteur : un type d'acteur (rôle) intervient dans le processus de coopération : l'expert du domaine.

Médiateur : ce module est constitué de deux composants dynamiques qui réalisent la médiation, le facilitateur et le processeur de requêtes.

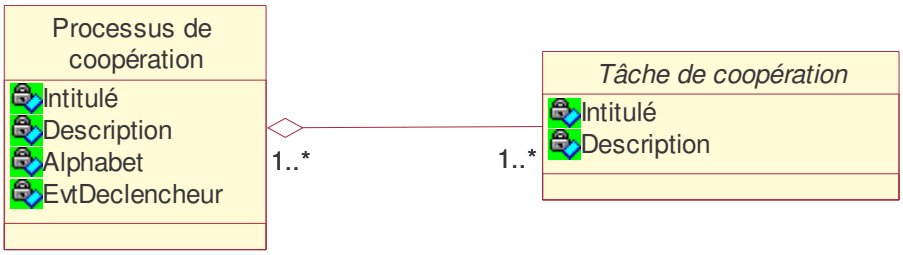
Facilitateur : traite les conflits entre les informations du système local et les informations fournies par d'autres composants. Il assure également les fonctionnalités de réconciliation de contextes, de fusion d'objets et met à jour l'ontologie.

Processeur de requêtes : reçoit la requête utilisateur exprimée dans le modèle de médiation et portant sur les informations du schéma de données. Il permet l'interrogation des différentes sources d'informations utiles à la résolution de cette requête et renvoie un résultat qui intègre les résultats des sources d'informations interrogées dans un format homogène adapté aux besoins de l'utilisateur. Il réalise les fonctionnalités de validation syntaxique et sémantique d'une requête, de décomposition de la requête et de recomposition des résultats.

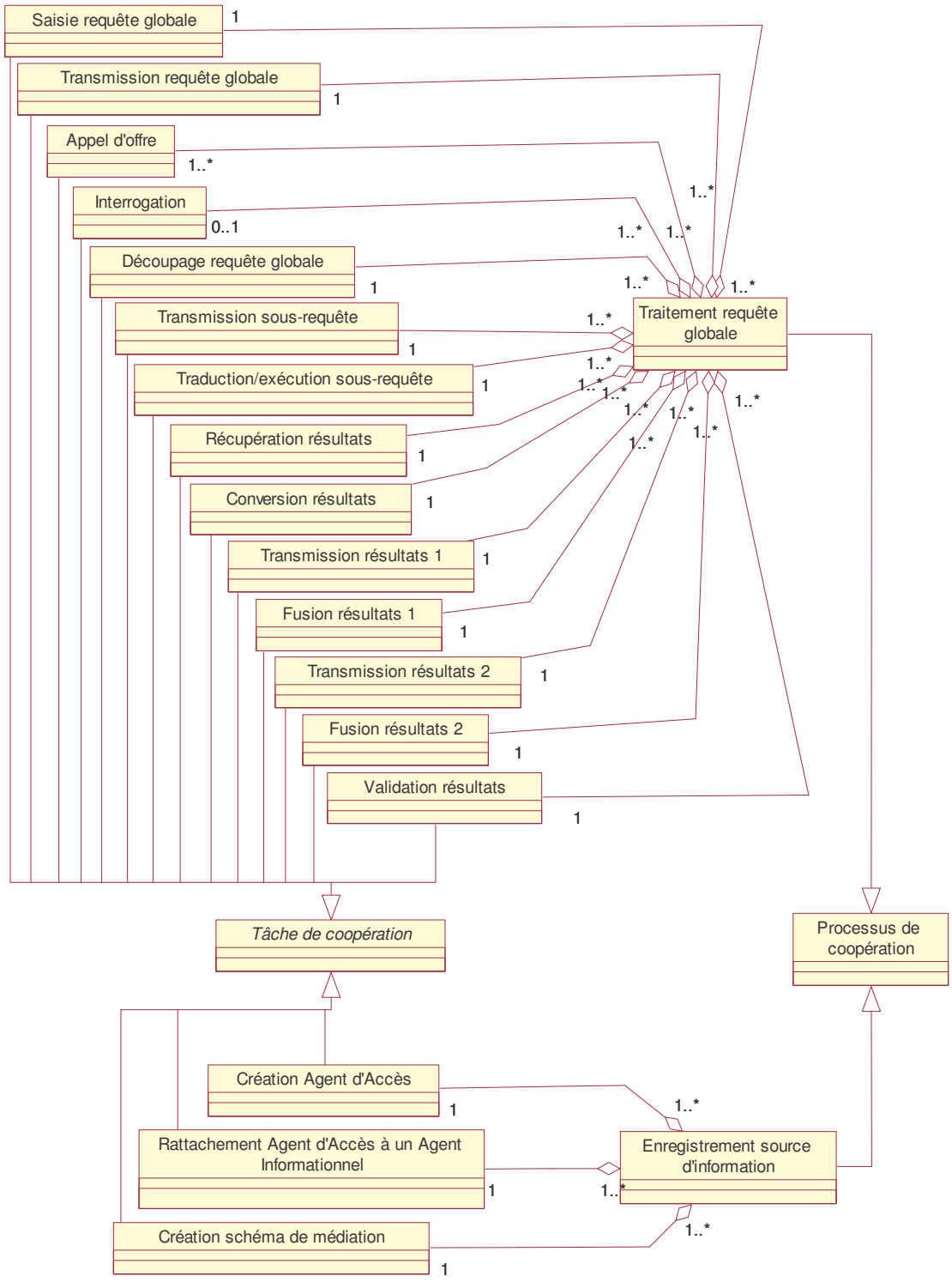
Adaptateur : permet d'interfacer le système existant avec sa description exprimée sur le modèle de médiation dans le schéma de données. Il accède à une ontologie qui spécifie les informations de mise en correspondance entre ces deux descriptions des informations. Il réalise les fonctionnalités de traduction de schéma et de traduction de requête. *Annuaire :* a pour fonction de répertorier les composants disponibles (médiateurs, sources d'information) et de faciliter la recherche des systèmes fournisseurs qui peuvent répondre aux besoins d'un système consommateur. L'annuaire enregistre l'identité et la définition simplifiée du domaine d'application de chaque système participant. *Traduction de schémas :* consiste à construire un schéma de données dans le modèle de médiation à partir des informations locales. Un dictionnaire de mapping contenu dans l'ontologie est constitué durant cette phase de traduction. *Traduction de requêtes :* permet de traduire une requête exprimée sur le schéma de données en une interrogation du système existant pour extraire des données. *Validation de la requête :* vérification de la syntaxe et de la cohérence d'une requête exprimée dans le modèle de médiation. *Décomposition de la requête :* décomposition de la requête globale en sous-requêtes. *Recomposition de la requête :* utilise les résultats intermédiaires fournis par les adaptateurs pour reconstruire un résultat qui correspond à la requête initiale posée par l'utilisateur. *Réconciliation :* permet de découvrir dynamiquement les données pertinentes et de les adapter au contexte de l'application. Elle compare les contextes des SI locaux en utilisant une mesure de distance sémantique et identifie les informations utiles dans une coopération. *Fusion d'objets :* construit des classes d'objets virtuels donnant naissance à des vues combinant et transformant des objets locaux et assurant un accès transparent aux informations d'une coopération. *Enregistrement des composants :* permet l'ajout d'un composant du système à un référentiel. Chaque composant doit s'enregistrer pour pouvoir être utilisé comme fournisseur. *Recherche d'un composant :* identifie les systèmes contenant les informations dont le consommateur a besoin. REMARQUE — l'utilisateur n'est pas explicitement décrit dans le cadre de ce projet.
Relations
Raffine
« Modélisation des tâches de coopération effectuées par les agents ».

Tableau 11 : Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas »

3.2.6. Pattern d'analyse de coopération « Modélisation des processus de coopération »

Interface
Nom
Modélisation des processus de coopération
Classification
Pattern d'analyse de coopération
Contexte
Ce pattern ne peut être utilisé que si le pattern « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » ou « Modélisation des tâches de coopération effectuées par les agents » a été précédemment appliqué.
Objectif
Ce pattern permet de représenter les processus de coopération, i.e. regroupements ordonnés de tâches de coopération spécifiés dans le SIC.
Applicabilité
SIC
Solution
Modèle

Participants
Tâche de coopération : définit un événement de coopération à traiter par l'association entre un ensemble d'entrées, c'est-à-dire de données à traiter, et un ensemble de sorties, c'est-à-dire de résultats produits, et par l'indication de leurs caractéristiques respectives. *Processus de coopération :* ordonnancement partiel de tâches de coopération.
A un participant commun avec
Tâche de coopération : modélisation de la hiérarchie des taches de coopération inhérentes au SIC [Modélisation des tâches de coopération effectuées par les agents] Tâche de coopération : modélisation de la hiérarchie des taches de coopération inhérentes au SIC de type médiation de schémas ou médiation de contextes [Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de

médiation de schémas]
Exemples d'utilisation
Exemple 1
Nom de l'exemple
Processus de coopération spécifiés dans le projet ACSIS [DUBOIS, 1997] [SEGURAN, 2003].
Portée
SIC médiation de schémas
Modèle



Participants

Enregistrement source d'information : processus de coopération assurant l'enregistrement d'une nouvelle source d'information auprès de la coopération. Ce processus est instancié à chaque ajout d'une source d'information. EvtDeclencheur = Ajout d'une source d'information. Alphabet :

1. Création Agent d'Accès 2. Création schéma de médiation 3. Rattachement Agent d'Accès à un Agent Informationnel *Traitement requête globale :* processus de coopération assurant le traitement d'une requête globale soumis à la coopération. Ce processus est instancié à chaque soumission d'une requête. EvtDeclencheur = soumission d'une requête globale. Alphabet : 1. Saisie requête globale 2. Transmission requête globale 3. Appel d'offre Si *incapacité des agents d'accès à traiter requête* alors 4. Interrogation 3. Appel d'offre Fin Si 5. Découpage requête globale 6. Transmission sous-requête 7. Traduction et exécution sous-requête 8. Récupération des résultats par l'AA. 9. Conversion résultats 10. Transmission résultats 1 11. Fusion résultats 1 12. Transmission résultats 2 13. Fusion résultats 2 14. Validation résultats
Relations
Requiert
« Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » ou « Modélisation des tâches de coopération effectuées par les agents »

Tableau 12 : Pattern d'analyse de coopération « Modélisation des processus de coopération »

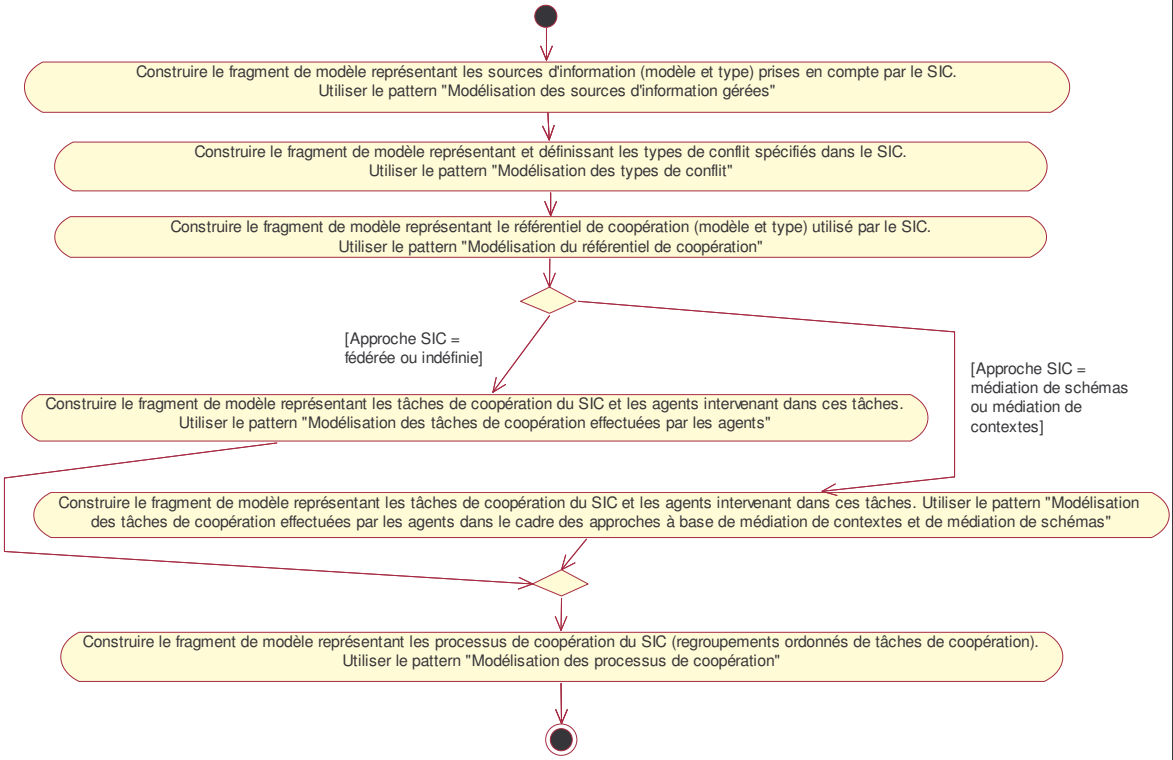
3.3. Réutilisation des patterns d'analyse de coopération

La réutilisation des patterns d'analyse de coopération consiste à appliquer ces patterns durant la phase d'analyse du processus d'ingénierie du SIC. Plus précisément, elle consiste à adapter tout ou

partie du modèle fourni dans la solution de chaque pattern pour représenter les éléments caractéristiques du SIC (ou de l'approche de coopération) à concevoir, et ce, en fonction du niveau de détail voulu par le concepteur (utilisation ou non des attributs des classes, de certaines classes…). Ces patterns permettent ainsi de préciser les entités métier (modèle) du SIC à concevoir.

La réutilisation est facilitée par deux patterns de support d'utilisation permettant de modéliser les contraintes d'application des patterns. Le premier (cf. Tableau 13) représente les patterns d'analyse à utiliser dans le cadre de la spécification d'un SIC ; le second (cf. Tableau 14) entre dans le cadre de la spécification d'une approche de coopération.

Interface
Nom
Utilisation des patterns d'analyse de coopération pour la spécification d'un SIC
Classification
Pattern de support d'utilisation
Objectif
Ce pattern permet de préciser les différents patterns à appliquer dans le cadre de l'utilisation du système de patterns d'analyse de coopération afin de modéliser les principes et fonctionnalités d'un SIC.
Applicabilité
Système de patterns d'analyse
Solution
Démarche



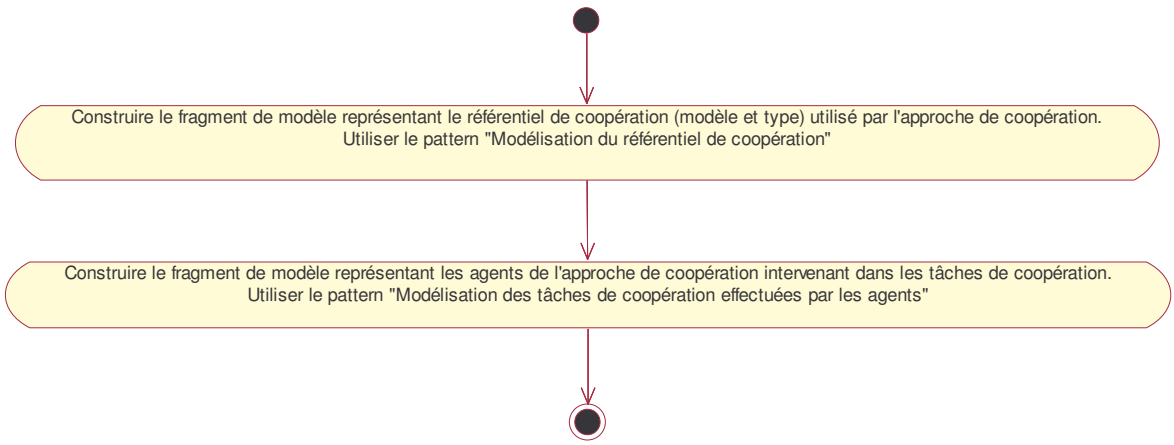
Ce pattern propose d'utiliser d'abord les patterns « Modélisation des sources d'information gérées », « Modélisation des types de conflit », « Modélisation du référentiel de coopération » et « Modélisation des tâches de coopération effectuées par les agents » ou « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » pour construire le modèle d'analyse du SIC. Il n'impose pas un ordre quant à l'application de ces patterns. Il nous semble néanmoins souhaitable de respecter l'ordre proposé impliquant une complexité croissante. Par ailleurs, ce pattern suppose l'application préalable soit du pattern « Modélisation des tâches de coopération effectuées par les agents », soit du pattern « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » pour appliquer le pattern « Modélisation des processus de coopération ».

Le choix entre les patterns « Modélisation des tâches de coopération effectuées par les agents » et « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » s'effectue en fonction du type de SIC à concevoir. Si le SIC à concevoir est de type médiation, il nécessitera l'utilisation du pattern « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas ». Les approches de type médiation de schémas et médiation de contextes offrent une meilleure garantie des propriétés de la coopération

(transparence, évolutivité, scalabilité) (Cf. section 3.1.3.3 du chapitre 1).
Relations
Utilise
« Modélisation des types de conflit » « Modélisation des sources d'information gérées » « Modélisation du référentiel de coopération » « Modélisation des tâches de coopération effectuées par les agents » « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » « Modélisation des processus de coopération »

Tableau 13 : Pattern de support d'utilisation « Utilisation des patterns d'analyse de coopération pour la spécification d'un SIC »

Interface
Nom
Utilisation des patterns d'analyse de coopération pour la spécification d'une approche de coopération
Classification
Pattern de support d'utilisation
Objectif
Ce pattern permet de préciser les différents patterns à appliquer dans le cadre de l'utilisation du système de patterns d'analyse de coopération afin de modéliser les principes d'une approche de coopération.
Applicabilité
Système de patterns d'analyse
Solution
Démarche



Ce pattern propose d'utiliser d'abord le pattern « Modélisation du référentiel de coopération » avant le pattern « Modélisation des tâches de coopération effectuées par les agents » pour construire le modèle d'analyse de l'approche de coopération. Il n'impose pas un ordre quant à l'application de ces patterns. Il nous semble néanmoins souhaitable de respecter l'ordre proposé impliquant une complexité croissante.

Relations
Utilise
« Modélisation du référentiel de coopération » « Modélisation des tâches de coopération effectuées par les agents »

Tableau 14 : Pattern de support d'utilisation « Utilisation des patterns d'analyse de coopération pour la spécification d'une approche de coopération »

4. Travaux futurs

Les travaux futurs concernent la validation des patterns d'analyse de coopération. Les exemples d'utilisation fournis dans les patterns et modélisant les concepts de systèmes d'information coopératifs existants permettent de valider leur utilité, mais aussi partiellement les problèmes identifiés et solutions proposées. Les patterns d'analyse de coopération seront cependant prochainement validés et appliqués dans le cadre du projet TRAFIC[82] de la région Rhône-Alpes [TRAFIC, 2004], projet auquel l'équipe de recherche MODEME contribue. L'objectif de ce projet est de définir un cadre méthodologique et une plate-forme expérimentale destinés à capitaliser et diffuser des connaissances de spécification, de conception et de mise en œuvre de systèmes

[82] TRAFIC est l'acronyme de Transports : Réutilisation, Adaptation, Fiabilité, Intermodalité et Coopération inter-systèmes.

d'information dédiés en particulier à l'intermodalité[83] des transports dans un contexte régional. Les résultats de ce projet, permettront de faciliter l'incubation et la diffusion d'approches innovantes centrées sur la réutilisation, l'adaptation, la fiabilité et la coopération de systèmes et de composants de façon à privilégier une conception et un assemblage rapides, cohérents et traçables de systèmes d'information et de logiciels multiples interdépendants dans le domaine de l'intermodalité des transports et de la mobilité des acteurs. Dans le cadre de ce projet, les patterns d'analyse proposés seront appliqués lors de l'ingénierie d'une architecture dédiée à la coopération des systèmes d'information de transport et permettront, notamment, de modéliser les modalités de coopération inter-composants métier et inter-systèmes de transport.

5. Conclusion

Ce chapitre a permis de représenter le référentiel du domaine de la coopération obtenu à partir de l'analyse de domaine et organisant l'ensemble des connaissances dans un modèle de domaine. Des fragments de ce modèle ont été intégrés au sein de patterns d'analyse de coopération, patterns répondant à des problèmes précis de spécification de SIC ou d'approches de coopération. L'ensemble des patterns d'analyse de coopération vise à guider un concepteur de SIC ou d'approche de coopération dans la représentation et la réutilisation des concepts gérés. Ils permettent de construire le modèle statique du SIC ou de l'approche, en fixant les objets métiers gérés et en les structurant en terme de classes et de relations entre les classes. Ces patterns sont à la fois utilisables pour représenter les concepts de nouveaux SIC ou approches de coopération, mais peuvent également s'appliquer pour modéliser ceux de SIC ou d'approches existants. Deux patterns de support d'utilisation, fonctions de la spécification d'un SIC ou d'une approche de coopération, sont également proposés et ont pour objectif de guider l'utilisation du système de patterns d'analyse et constituent ainsi des points d'entrée de ce système.

Le chapitre suivant (Cf. chapitre 4) présente les patterns dédiés aux phases suivantes du processus d'ingénierie des systèmes d'information coopératifs, à savoir les phases de conception et d'implantation.

[83] L'intermodalité est l'usage, l'exploitation et la gestion coopératives de deux ou plusieurs modes de transport pour un seul déplacement visant à créer entre autres, une complémentarité entre les modes de transport.

Chapitre 4 : Patterns dédiés à la conception et à l'implantation des systèmes d'information coopératifs

1. Introduction

L'objectif de ce chapitre est de détailler les systèmes de patterns dédiés aux phases de conception et d'implantation de l'ingénierie des SIC, i.e. les patterns d'architecture de coopération, les patterns de conception de coopération et les patterns de support technique ainsi que les patterns de support d'utilisation facilitant la réutilisation de ces systèmes. Les *patterns d'architecture de coopération* s'appliquent au début de la phase de conception et fournissent des modèles d'architecture "génériques" facilitant la détermination de l'organisation et de la structure de base d'un SIC spécifique. Les *patterns de conception de coopération* permettent de compléter la description des entités d'analyse identifiées par les patterns d'analyse de coopération et spécifient notamment leur comportement (opérations sur les classes) et leurs collaborations. Les *patterns de support technique* décrivent comment construire une application à partir des patterns de conception de coopération : ce sont des patterns dédiés à l'implémentation et à l'implantation des entités spécifiées dans ces derniers. La structure de ces différents patterns est basée sur le formalisme présenté précédemment (cf. section 4.2 du chapitre 2).

Nous détaillons tout d'abord les patterns d'architecture de coopération (§ 2) puis les patterns de conception de coopération et de support technique (§ 3). Enfin, la dernière section (§ 4) conclut ce chapitre et présente les futurs travaux d'extension des systèmes de patterns présentés.

2. Patterns d'architecture de coopération

Cette section a pour objectif de détailler les patterns d'architecture de coopération dédiés au début de la phase de conception des SIC. Préalablement à l'explicitation des patterns d'architecture de

coopération que nous avons conçus (Cf. section 2.2), nous présentons les résultats de la conception de domaine (Cf. section 2.1). Enfin, la dernière partie (Cf. section 2.3) explicite la réutilisation du système de patterns d'architecture proposé et introduit un pattern de support d'utilisation facilitant celle-ci.

2.1. La conception de domaine et ses résultats

La conception de domaine est le processus de construction d'un modèle de conception à partir des produits de l'analyse de domaine et des connaissances acquises lors de l'étude des conceptions logicielles et architectures génériques existantes. Elle vise à spécifier une architecture et ses composants pour une famille d'applications.

Nous avons vu précédemment (Cf. section 3.2.3 du chapitre 2) que nous limitions la phase de conception de domaine à la spécification d'un (ou, si nécessaire, plusieurs) modèle d'architecture "générique" s'appliquant aux systèmes d'information coopératifs. Ce modèle d'architecture doit permettre de représenter les similarités et variabilités structurelles et organisationnelles des SIC et ainsi de faciliter la détermination de l'architecture d'un SIC particulier. Il permet, plus précisément, de représenter les sous-systèmes et/ou couches caractéristiques des systèmes d'information coopératifs. D'après de nombreux auteurs [BOUGUETTAYA, 1994] [BOUGUETTAYA & AL., 1998] [JOUANOT, 2001] [BOUCELMA & LACROIX, 2001], ce modèle d'architecture "générique" n'est pas unique, mais est fonction de l'approche de coopération (approche fédérée, approche à base de médiation de schémas ou approche à base de médiation de contextes) dont dépend le SIC à concevoir. Ainsi, un ou plusieurs modèles d'architecture "génériques" facilitant la définition de l'architecture d'un SIC ont été spécifiés pour chaque approche. A l'intérieur même d'une approche de coopération, la plupart des modèles "génériques" d'architecture de SIC proposés par les auteurs sont cependant sensiblement similaires. Nous n'explicitons pas, en détail, les différents modèles d'architecture "génériques" - résultant de la phase de conception de domaine - que nous avons retenus car ceux-ci sont présentés dans chaque pattern proposé dans la section suivante (§ 2.2). Nous illustrons cependant ces résultats par l'explicitation du modèle d'architecture "générique" spécifié dans le cadre des SIC issus de l'approche fédérée. Ainsi, Bouguettaya & al. [BOUGUETTAYA & AL., 1998] considèrent que les SIC de type fédéré sont constitués de trois couches : une première couche contenant les sources d'information participantes, une deuxième contenant le (ou les différents) module(s) fonctionnel(s) du SIC chargés de résoudre les requêtes globales et traiter les conflits inhérents ainsi que le dictionnaire de la fédération et une troisième couche correspondant aux utilisateurs chargés de formuler des requêtes sur la coopération de systèmes d'information.

Chaque modèle d'architecture "générique" correspondant à chaque approche a été intégré dans un pattern d'architecture de coopération. Des exemples d'utilisation issus de l'étude des architectures conçues dans différents projets de coopération sont également présentés dans chaque pattern et constituent ainsi des exemples d'adaptation des modèles d'architecture "génériques" fournis. Ces patterns sont présentés dans la section suivante.

2.2. Présentation des patterns d'architecture de coopération

Un pattern d'architecture exprime l'organisation et le schéma d'une application : il décompose l'application en un ensemble de sous-systèmes tout en spécifiant pour chacun d'eux ses responsabilités et interactions avec les autres sous-systèmes [BUSCHMANN & AL., 1996]. Dans le cadre des systèmes d'information coopératifs, un pattern d'architecture a pour objectif de présenter les sous-systèmes et/ou couches caractéristiques des SIC, i.e. de proposer un modèle d'architecture "générique" réutilisable afin de faciliter la définition de l'architecture d'un SIC spécifique. Comme nous venons de le voir, ce modèle d'architecture "générique", qui sera par la suite adapté en fonction des entités métier spécifiques (acteurs et référentiels spécifiés,…) au SIC, mises en exergue lors de la phase d'analyse (Cf. chapitre 3), dépend de l'approche de coopération choisie (approche fédérée, approche à base de médiation de schémas et approche à base de médiation de contextes). Ainsi, le système de patterns d'architecture de coopération est composé de trois patterns :

- Le pattern « Architecture d'un SIC issu de l'approche fédérée » présente le modèle d'architecture "générique" s'appliquant aux SIC issus de l'approche fédérée (Cf. Tableau 15).
- Le pattern « Architecture d'un SIC issu de l'approche à base de médiation de schémas » vise à déterminer l'architecture des SIC de type médiation de schémas (Cf. Tableau 16).
- Le pattern « Architecture d'un SIC issu de l'approche à base de médiation de contextes » facilite la définition de l'architecture des SIC issus de l'approche à base de médiation de contextes (Cf. Tableau 17).

Ces trois patterns sont présentés dans les sections suivantes (§ 2.2.1 à 2.2.3).

2.2.1. Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche fédérée »

Interface
Nom
Architecture d'un SIC issu de l'approche fédérée.

Classification

Pattern d'architecture de coopération

Objectif

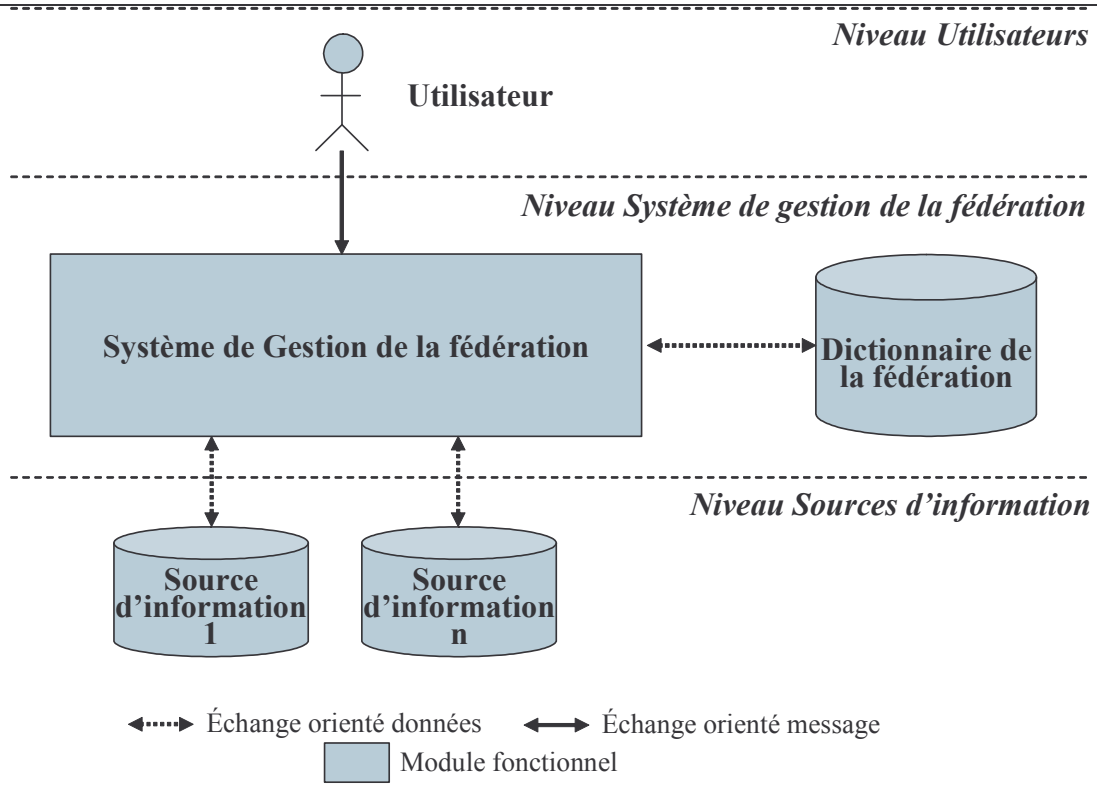
Ce pattern a pour objectif de représenter l'architecture des SIC issus de l'approche fédérée.

Applicabilité

SIC fédéré

Solution

Modèle



Participants

Un système d'information coopératif de type fédéré est constitué de trois niveaux (ou couches) :

Niveau Sources d'information (1er niveau) : contient les sources d'information locales, i.e. les entités participant à la coopération. Ces sources d'information peuvent être de différents types (sources structurées, semi-structurées ou non structurées).

Niveau Système de gestion de la fédération (2ème niveau) : contient le module fonctionnel en charge de la résolution de la requête globale et du traitement de certains conflits structurels et sémantiques inhérents. Il accède à la base de connaissances appelée *dictionnaire de la fédération* contenant la description globale de la fédération, les informations concernant l'interface du système fédéré, le ou les protocoles utilisés dans la coopération (protocole de négociation entre les sources locales qui coopèrent pendant la phase de manipulation des données,…) et les liens inter-bases.

Niveau Utilisateurs (3ème niveau) : utilisateurs soumettant une requête à la coopération.

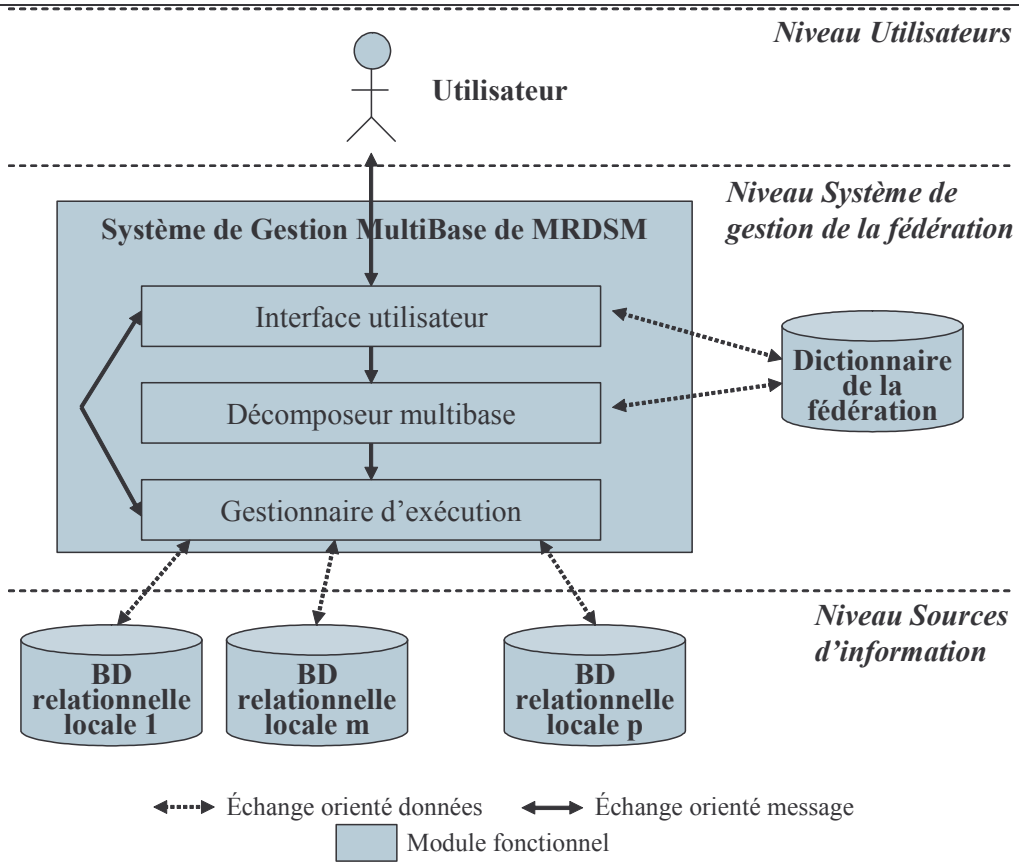
Exemples d'utilisation

Exemple 1

Nom de l'exemple

Architecture du SIC MRDSM [LITWIN, 1985] [LITWIN, 1988].

Modèle



Participants

Niveau Sources d'information (1er niveau) : contient les sources d'information locales, i.e. les entités participant à la coopération. Ces sources d'information sont des bases de données relationnelles reposant sur le système de gestion de bases de données MRDS.

Niveau Système de gestion de la fédération (2ème niveau) : contient le module fonctionnel en charge de la résolution de la requête globale et du traitement de certains conflits structurels et sémantiques inhérents. Ce module contient trois sous-modules fonctionnels : l'*interface utilisateur*, le *décomposeur multibase* et le *gestionnaire d'exécution*. L'interface utilisateur permet la création de la requête multibase à l'aide du langage de manipulation multibase dont la description est contenue dans le dictionnaire de la fédération (ou référentiel multibase). Le décomposeur multibase détermine les noms des bases de données composantes qui doivent être ouvertes (grâce au dictionnaire de la fédération), redirige les requêtes n'impliquant qu'une seule base vers les bases correspondantes, décompose la requête multibase en requêtes élémentaires qui seront adressées aux bases locales respectives et résout les conflits de nommage, de type de données et de type de

valeur. Le gestionnaire d'exécution requiert et contrôle l'exécution de la requête élémentaire.

Le dictionnaire de la fédération contient notamment la sémantique du langage multibase, i.e. les extensions et enrichissements des langages de manipulation et de définition des données (capacités d'adressage des sources d'information, de préfixation des données manipulées par le nom de la base, de définition de vues multibases,…) et les liens inter-bases.

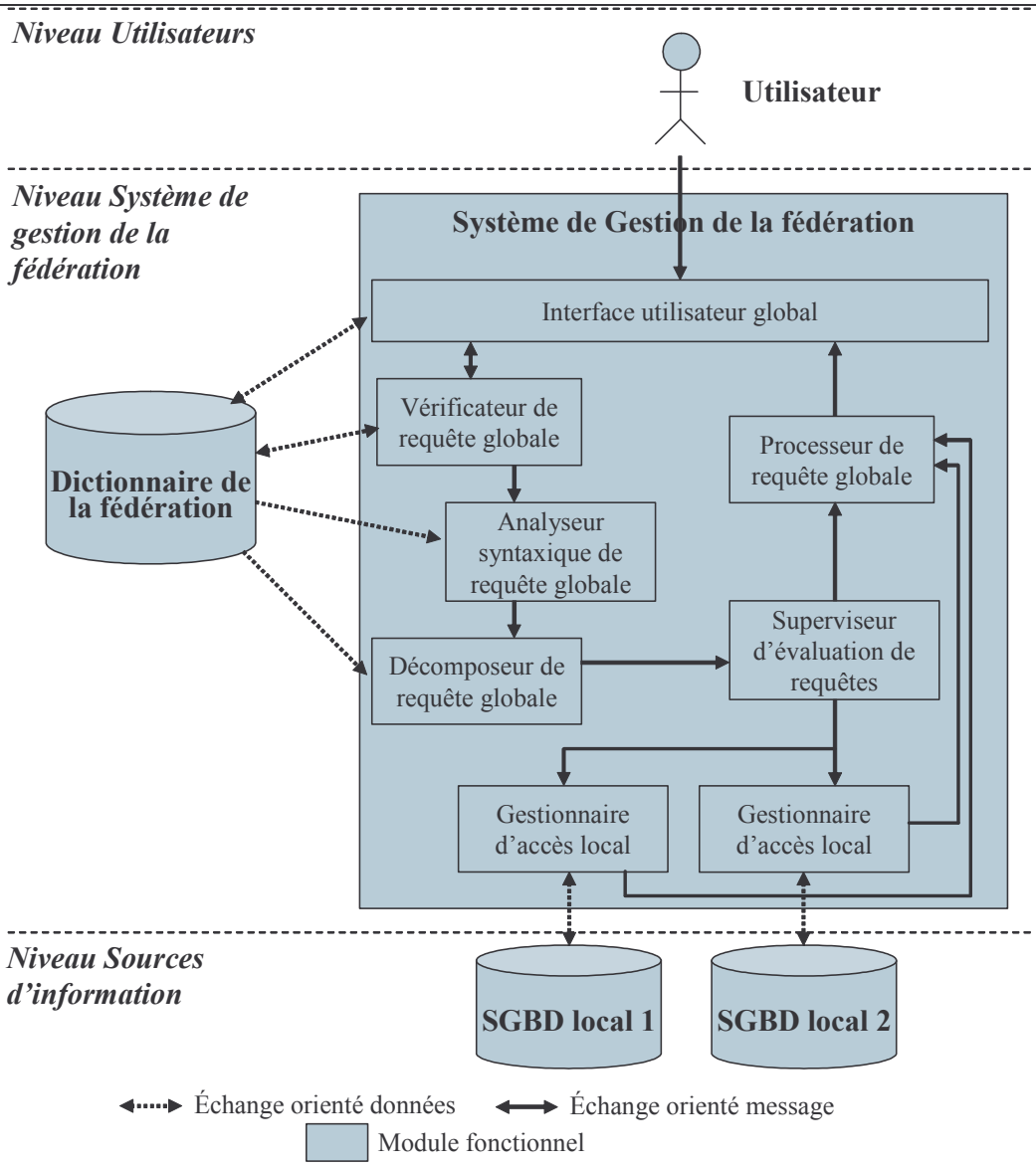
Niveau Utilisateurs (3ème niveau) : utilisateurs soumettant une requête à la coopération.

Exemple 2

Nom de l'exemple

Architecture du SIC Omnibase [RUSINKIEWICZ & AL., 1988]

Modèle



Participants

Niveau Sources d'information (1er niveau) : contient les sources d'information locales, i.e. les

entités participant à la coopération. Ces sources d'information sont des bases de données. *Niveau Système de gestion de la fédération (2ème niveau) :* contient le module fonctionnel en charge de la résolution de la requête globale et du traitement des conflits liés aux modèles de données, aux langages de manipulation et aux systèmes de gestion différents. L'architecture du système repose sur des composants spécialisés accessibles via une interface globale et capables de traiter les requêtes adressées à la fédération. La requête est d'abord prise en compte par le dictionnaire de la fédération couplé à un outil de vérification (utilisation du sous-module fonctionnel "Vérificateur de requête globale") qui rectifie les éventuelles ambiguïtés structurelles ou sémantiques. La requête est ensuite analysée (utilisation du sous-module fonctionnel "Analyseur syntaxique de requête globale") et décomposée (utilisation du sous-module fonctionnel "Décomposeur de requête globale") en sous-requêtes adressées aux systèmes de gestion de bases de données locaux. Le traitement des sous-requêtes est spécifié grâce à un langage de gestion des accès distribués appelé DOL (Distributed Operations Language). La réponse finale est obtenue par combinaison des réponses des sous-requêtes grâce au superviseur d'évaluation de requêtes. Enfin, le processeur de requête globale transmet cette réponse finale à l'interface globale pour affichage. Le recours au dictionnaire de la fédération permet de résoudre les ambiguïtés concernant les noms d'entités et les incompatibilités de représentation des données (utilisation d'une représentation canonique commune à toute les bases impliquées dans la fédération). *Niveau Utilisateurs (3ème niveau) :* utilisateurs soumettant une requête à la coopération.

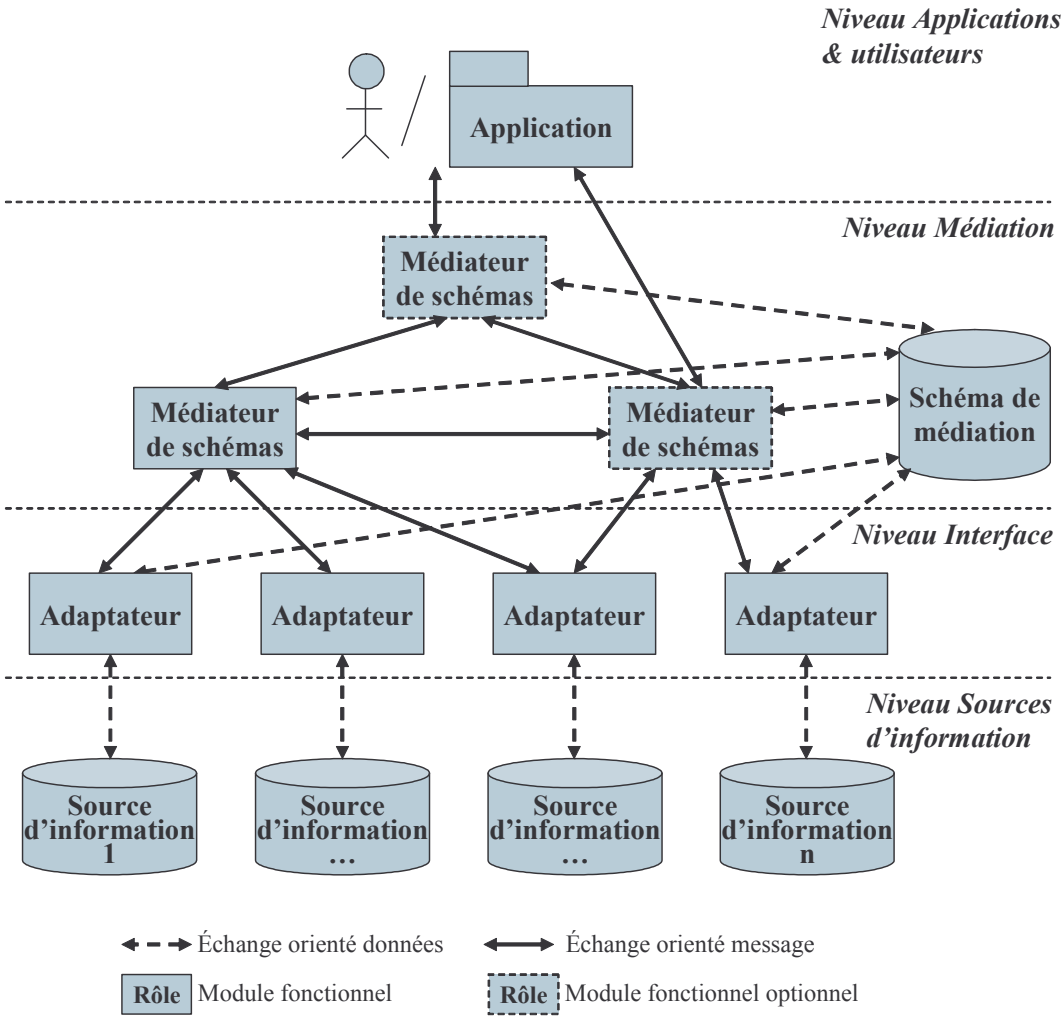
Tableau 15 : Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche fédérée »

2.2.2. Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche à base de médiation de schémas »

Interface
Nom
Architecture d'un SIC issu de l'approche à base de médiation de schémas.
Classification
Pattern d'architecture de coopération
Objectif
Ce pattern a pour objectif de représenter l'architecture des SIC issus de l'approche de coopération de type médiation de schémas.
Applicabilité
SIC médiation de schémas.

Solution

Modèle



Participants

Un système d'information coopératif de type médiation de schémas est constitué de quatre niveaux (ou couches) :

Niveau Sources d'information (1er niveau) : contient les sources d'information locales, i.e. les entités participant à la coopération. Ces sources d'information peuvent être de différents types (sources structurées, semi-structurées ou non structurées).

Niveau Interface (2ème niveau) : contient le (ou les) module(s) fonctionnel(s) (rôle = adaptateur) constituant la (ou les) interface(s) d'accès aux sources d'information. Chaque module fonctionnel de cette couche est chargé d'homogénéiser les modèles de données locaux, en présentant les données dans le modèle de médiation (modèle du schéma de médiation), et ainsi de résoudre les conflits de modèle de données et de langages d'interrogation. Il nécessite ainsi un accès à la base de connaissances chargée de stocker ce schéma. Ce(s) module(s) est (sont) parfois appelé(s) adaptateur(s) selon les SIC.

Niveau médiation (3ème niveau) : contient le (ou les) module(s) fonctionnel(s) chargés de réaliser la médiation des informations. Ce(s) module(s) fonctionnel(s) (rôle = médiateur de schémas) permettent de localiser l'information et de traiter certains conflits structurels et sémantiques. Chaque module accède à la base de connaissances chargée de stocker le schéma de médiation. Ce(s) module(s) est (sont) parfois appelé(s) médiateur(s) de schémas selon les SIC.

Niveau Applications & utilisateurs (4ème niveau) : utilisateurs et/ou programmes d'application soumettant une requête à la coopération.

Ces différentes couches peuvent également contenir d'autres modules fonctionnels (optionnels) permettant, par exemple, de localiser les composants de l'architecture (annuaire), d'interagir avec l'utilisateur (module d'interface avec l'utilisateur),…

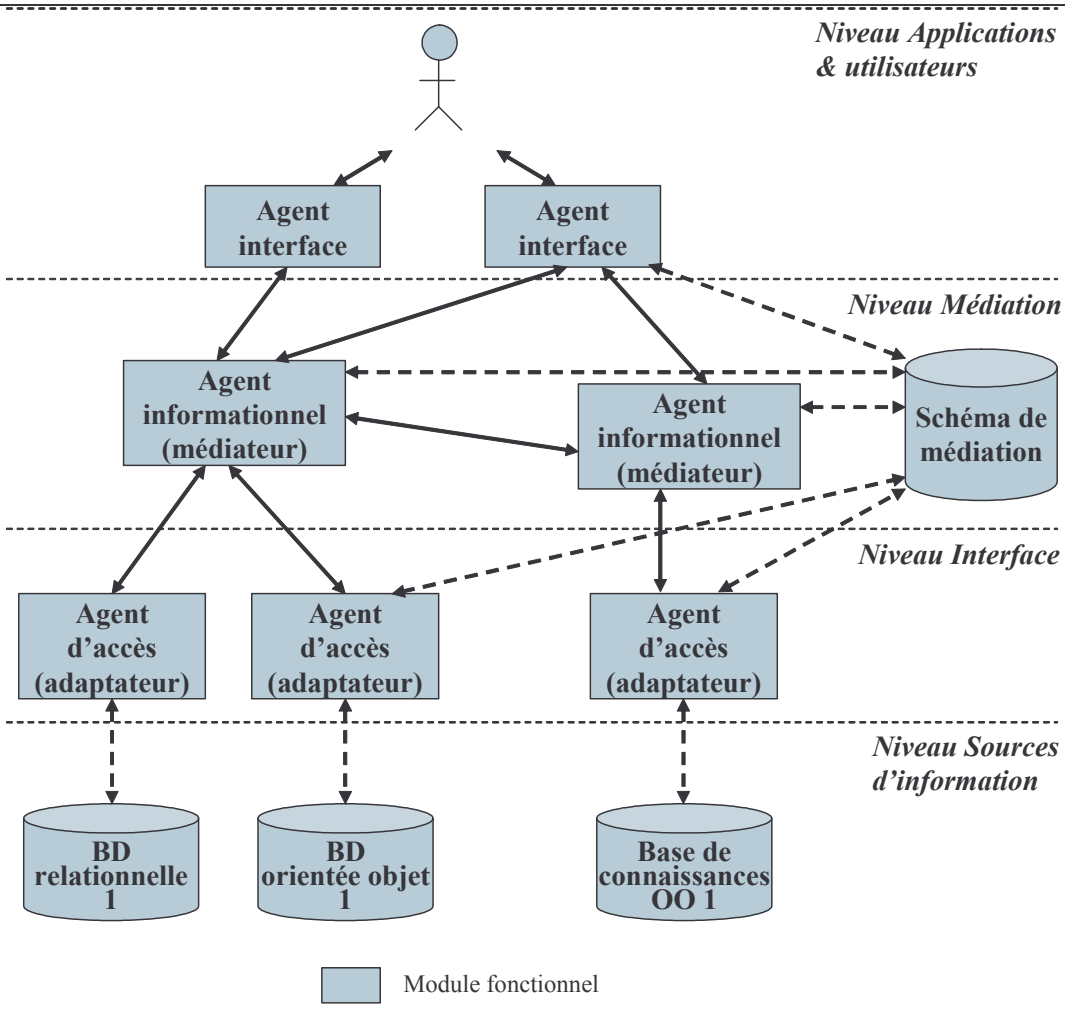
Exemples d'utilisation

Exemple 1

Nom de l'exemple

Architecture du SIC ACSIS [DUBOIS, 1997] [SEGURAN, 2003].

Modèle



Participants
Le SIC ACSIS est constitué de quatre niveaux (ou couches) : *Niveau Sources d'information (1er niveau) :* contient les sources d'information locales, i.e. les entités participant à la coopération. Ces sources d'information sont soit des bases de données relationnelles ou orientées-objet, soit des bases de connaissances orientées-objet. *Niveau Interface (2ème niveau) :* contient les modules fonctionnels chargés d'encapsuler les sources d'information locales (rôle = adaptateur). Ces modules sont dénommés « agents d'accès » au sein du SIC ACSIS. Un agent d'accès assure la participation des données locales au sein des processus coopératifs (il a la charge de constituer le schéma de médiation). Il traduit une requête exprimée dans le langage global d'interrogation en requête exprimée dans le langage propre à la base locale dont il a la charge, exécute cette dernière, réceptionne et renvoie les résultats de la base locale. *Niveau médiation (3ème niveau) :* contient les modules fonctionnels (rôle = médiateur de schémas) chargés de réaliser la médiation des informations. Ces modules sont dénommés « Agents informationnels » dans le SIC ACSIS. Ce sont des agents multi-domaines (regroupant des domaines proches sémantiquement) en charge du découpage des requêtes globales en sous-requêtes et de la récupération des résultats provenant des agents d'accès afin d'obtenir une solution globale. Ils accèdent au schéma de médiation et permettent le traitement de certains conflits structurels et sémantiques. *Niveau Applications & utilisateurs (4ème niveau) :* utilisateurs et/ou programmes d'application soumettant une requête à la coopération. Cette couche est également constituée d'agents interface. Un agent interface sert d'intermédiaire entre les utilisateurs ou les experts du domaine et les autres agents du système. Chaque rôle correspond à un ensemble de permissions sur le système et est représenté par une spécialisation de l'*Agent Interface*. L'*Agent Interface Utilisateur* est gestionnaire de la requête et l'exécute sans se préoccuper de la localisation des données. Il permet la validation des résultats et peut déclencher la ré-exécution de la requête. L'*Agent Interface Expert* a une expertise sur le domaine de la base locale et accède au schéma de médiation.

Tableau 16 : Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche à base de médiation de schémas »

2.2.3. Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche à base de médiation de contextes »

Interface
Nom
Architecture d'un SIC issu de l'approche à base de médiation de contextes.

Classification

Pattern d'architecture de coopération

Objectif

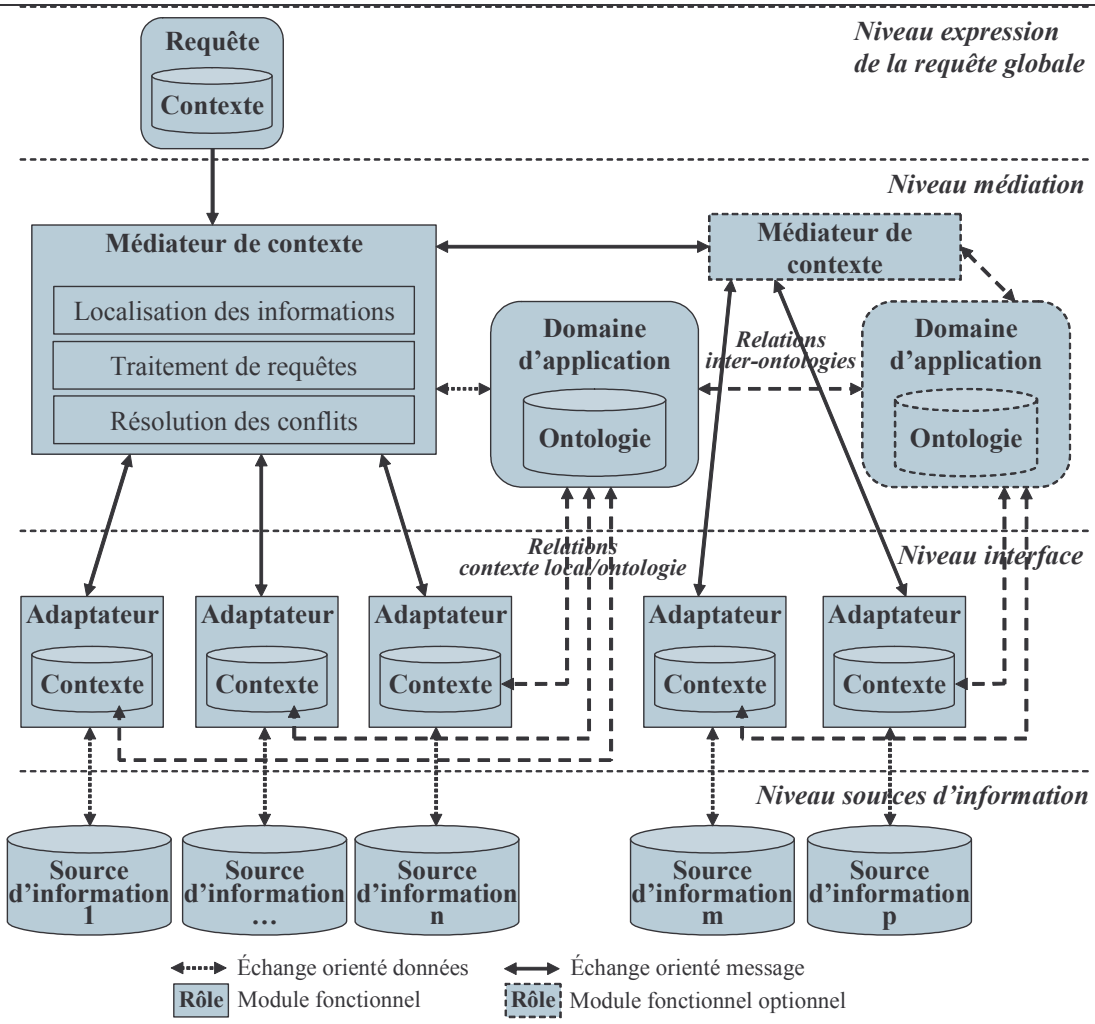
Ce pattern a pour objectif de représenter l'architecture des SIC issus de l'approche de coopération de type médiation de contextes.

Applicabilité

SIC médiation de contextes.

Solution

Modèle



Participants

Un système d'information coopératif de type médiation de contextes est constitué de quatre niveaux (ou couches) :

Niveau Sources d'information (1er niveau) : contient les sources d'information locales, i.e. les entités participant à la coopération. Ces sources d'information peuvent être de différents types (sources structurées, semi-structurées ou non structurées). Chaque source d'information décrit le contexte de ses informations et doit se définir par rapport à un modèle de référence. Chaque

système composant doit donc spécifier un contexte d'utilisation du modèle de référence.

Niveau Interface (2ème niveau) : contient le (ou les) module(s) fonctionnel(s) (rôle = adaptateur) constituant la (ou les) interface(s) d'accès aux sources d'information. Les tâches exactes de ces modules varient d'un SIC à l'autre. Ce(s) module(s) est (sont) parfois appelé(s) adaptateur(s) selon les SIC.

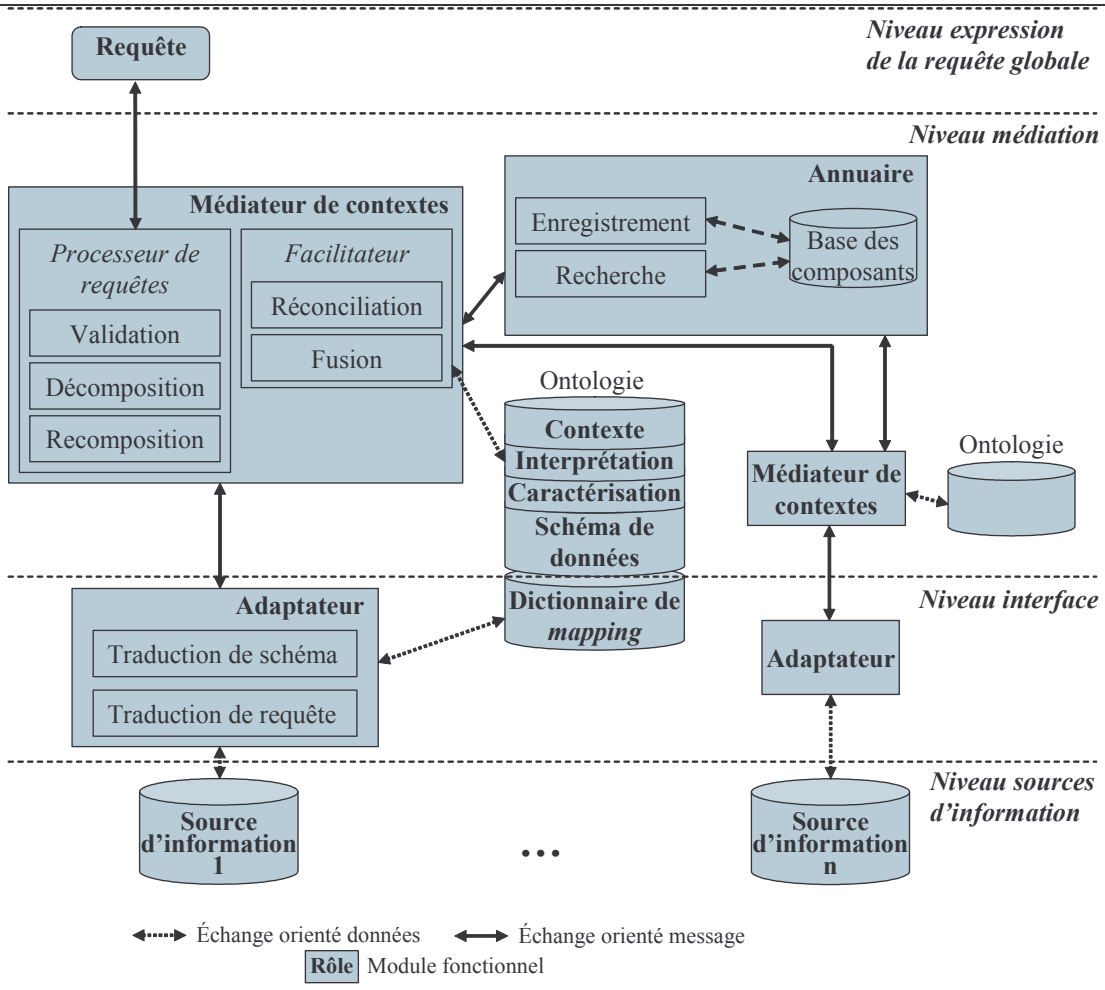
Niveau médiation (3ème niveau) : contient le (ou les) module(s) fonctionnel(s) chargés de réconcilier les différents contextes des sources d'information locales. La médiation de contextes repose sur l'intégration dynamique des informations en fonction du contexte des sources d'information ou de la requête. Ce(s) module(s) fonctionnel(s) (rôle = médiateur de contextes) permettent de traiter certains conflits structurels et sémantiques. Ce(s) module(s) est (sont) parfois appelé(s) médiateur(s) de contextes selon les SIC. Le module fonctionnel est associé à un contexte d'application qui utilise les concepts et le vocabulaire d'une ontologie. Deux approches peuvent être distinguées : l'approche mono-domaine et l'approche multi-domaines. L'approche mono-domaine repose sur un ensemble unique de connaissances (domaine unique) pour décrire la sémantique des informations ; un module fonctionnel (rôle = médiateur de contextes) est alors associé à une ontologie unique représentant la connaissance du domaine d'application. L'approche multi-domaines repose sur la distribution des informations sur des domaines de connaissances différents, domaines en relation les uns avec les autres ; un module fonctionnel (rôle = médiateur de contextes) prend en charge les ontologies liées aux différents domaines de connaissances et dispose d'outils spécifiques pour définir des relations inter-ontologies explicitant les règles de mise en correspondance entre les concepts de domaines différents.

Niveau Expression de la requête globale (4ème niveau) : cette couche à trait à la soumission d'une requête globale à la coopération. La résolution de cette requête dépend du contexte inhérent.

Ces différentes couches peuvent également contenir d'autres modules fonctionnels (optionnels) permettant, par exemple, de localiser les composants de l'architecture (annuaire),…

Exemples d'utilisation
Exemple 1
Nom de l'exemple
Architecture du SIC DILEMMA [JOUANOT, 2000] [JOUANOT, 2001] [CULLOT & AL., 2003].

Modèle



Participants

Le SIC DILEMMA est constitué de quatre niveaux (ou couches) :

Niveau Sources d'information (1er niveau) : contient les sources d'information locales, i.e. les entités participant à la coopération. Ces sources d'information sont soit des bases de données, soit des ressources web (documents XML). Chaque source d'information décrit le contexte de ses informations.

Niveau Interface (2ème niveau) : contient les modules fonctionnels (rôle = adaptateur) dénommés adaptateurs au sein du projet DILEMMA. Chaque adaptateur permet d'interfacer le système existant avec sa description exprimée sur le modèle de médiation dans le schéma de données. Il accède à une ontologie qui spécifie les informations de mise en correspondance entre ces deux descriptions des informations. Il réalise les fonctionnalités de traduction de schémas et de traduction de requête.

Niveau médiation (3ème niveau) : contient les modules fonctionnels (rôle = médiateur de contextes) dénommés médiateurs de contextes dans le SIC DILEMMA. Chaque médiateur est constitué de deux composants dynamiques qui réalisent la médiation, le *facilitateur* et le *processeur de*

requêtes. Le facilitateur traite les conflits entre les informations du système local et les informations fournies par d'autres composants. Il assure également les fonctionnalités de réconciliation de contextes, de fusion d'objets et met à jour l'ontologie. Le processeur de requêtes reçoit la requête utilisateur exprimée dans le modèle de médiation et portant sur les informations du schéma de données. Il permet l'interrogation des différentes sources d'informations utiles à la résolution de cette requête et renvoie un résultat qui intègre les résultats des sources d'informations interrogées dans un format homogène adapté aux besoins de l'utilisateur. Il réalise les fonctionnalités de validation syntaxique et sémantique d'une requête, de décomposition de la requête et de recomposition des résultats. La couche médiation contient également un unique composant appelé *Annuaire*. L'annuaire a pour fonction de répertorier les composants disponibles (médiateurs, sources d'information) et de faciliter la recherche des composants fournisseurs qui peuvent répondre aux besoins d'un consommateur. *Niveau Expression de la requête globale (4ème niveau) :* cette couche a trait à la soumission d'une requête globale à la coopération. La résolution de cette requête dépend du contexte inhérent.

Tableau 17 : Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche à base de médiation de contextes »

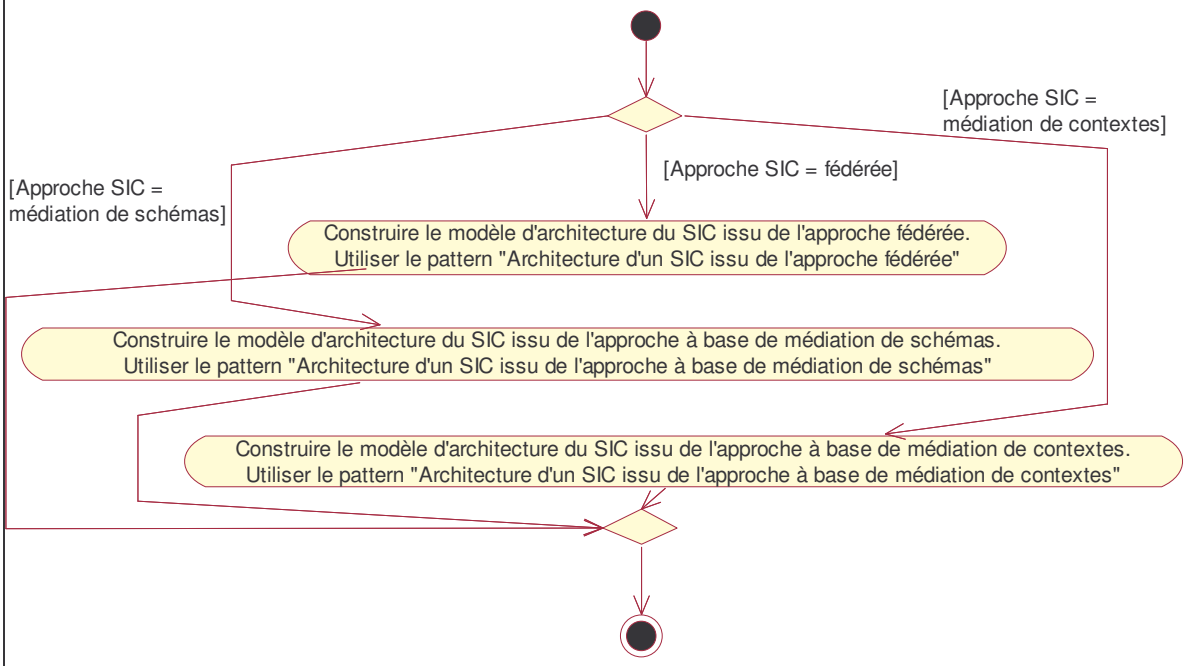
Nous précisons dans la section suivante (§ 2.3) le processus de réutilisation du système de patterns d'architecture de coopération et introduisons, à cet effet, un pattern de support d'utilisation.

2.3. Réutilisation des patterns d'architecture de coopération

La réutilisation des patterns d'architecture de coopération consiste à appliquer ces patterns au début de la phase de conception du cycle de développement du SIC. Plus précisément, elle consiste à adapter tout ou partie du modèle d'architecture "générique" fourni dans la solution de chaque pattern pour représenter les éléments architecturaux – ainsi que leurs responsabilités et types d'interactions – du SIC à concevoir, et ce, en fonction du niveau de détail voulu par le concepteur. Ces patterns ne peuvent cependant pas être appliqués sans avoir déterminé au préalable les entités métiers du SIC et en particulier ses modules fonctionnels et référentiels. Ainsi, ils requièrent l'application préalable des patterns d'analyse de coopération.

Comme pour les patterns d'analyse, la réutilisation des patterns d'architecture est facilitée par un pattern de support d'utilisation représentant les patterns d'architecture à utiliser dans le cadre de la spécification d'un SIC (Cf. Tableau 18).

Interface
Nom
Utilisation des patterns d'architecture de coopération.
Classification
Pattern de support d'utilisation
Objectif
Ce pattern permet de préciser les patterns à appliquer dans le cadre de l'utilisation du système de patterns d'architecture de coopération afin de représenter la structure et l'organisation des éléments architecturaux d'un SIC.
Applicabilité
Système de patterns d'architecture de coopération
Solution
Démarche



Ce pattern propose d'utiliser le pattern « Architecture d'un SIC issu de l'approche fédérée » ou « Architecture d'un SIC issu de l'approche à base de médiation de schémas » ou « Architecture d'un SIC issu de l'approche à base de médiation de contextes » pour représenter les éléments architecturaux (ou modèle d'architecture) du SIC à concevoir. Le choix entre ces trois patterns s'effectue en fonction de l'approche dont est issu le SIC à concevoir. Le choix de l'approche dépend notamment du degré de respect des propriétés de la coopération (transparence, évolutivité, scalabilité) (Cf. section 3.1.3.3 du chapitre 1).

Relations
Utilise
« Architecture d'un SIC issu de l'approche fédérée » « Architecture d'un SIC issu de l'approche à base de médiation de schémas » « Architecture d'un SIC issu de l'approche à base de médiation de contextes »

Tableau 18 : Pattern de support d'utilisation « Utilisation des patterns d'architecture de coopération »

Nous présentons dans la section suivante (§ 3) les patterns dédiés à la conception détaillée et à l'implantation des systèmes d'information coopératifs (patterns de conception de coopération et patterns de support technique).

3. Patterns de conception de coopération et patterns de support technique

Cette section a pour objectif de présenter en détail les patterns de conception de coopération (§3.1) et patterns de support technique (§ 3.2). La dernière partie (§ 3.3) présente la réutilisation des systèmes de patterns de conception et patterns de support technique proposés et introduit deux patterns de support d'utilisation facilitant celle-ci.

3.1. Patterns de conception de coopération

3.1.1. Introduction

Les patterns de conception de coopération ont, a priori, pour objectif de préciser les traitements et interactions entre les entités définies après adaptation des modèles fournis dans les patterns d'analyse de coopération et d'y ajouter des entités spécifiques nécessaires à la conception des SIC. Ils doivent ainsi permettre de modéliser les traitements et interactions entre les entités d'un SIC spécifique, après adaptation des modèles fournis au sein de leur solution. Ainsi, les modèles fournis dans les patterns de conception de coopération se doivent d'être suffisamment abstraits et génériques de façon à pouvoir être appliqués quel que soit le SIC à concevoir.

Nous avons vu précédemment (Cf. patterns d'analyse de coopération au chapitre 3) que la granularité et la définition des différents traitements (i.e. les tâches de coopération) varient selon les SIC. A l'instar de [FERNANDEZ & ZHAO, 2000] [GANGULY & AL., 2001], il nous semble difficile de spécifier des traitements génériques dans des patterns qui pourraient, après adaptation, s'appliquer à

la modélisation des traitements de tout SIC. Nos patterns de conception de coopération dédiés à la spécification des traitements ne permettent ainsi de décrire de façon détaillée que les traitements spécifiques au SIC ACSIS et ainsi de faciliter sa réutilisation.
Comme la plupart des travaux du domaine ayant spécifié des patterns de conception (Cf. section 6.2 du chapitre 2), nos patterns permettant de modéliser, de façon plus ou moins exhaustive, les entités informatiques et les traitements sont donc spécifiques à un SIC. Ainsi, le concepteur de SIC ne pourra pas utiliser directement certains de nos patterns de conception pour modéliser les entités de conception de son SIC spécifique ou les traitements inhérents. Il pourra cependant s'aider des patterns de conception existants – les nôtres ou ceux de [GANGULY & AL., 2001] ou [FERNANDEZ & AL., 2003] – pour définir notamment les traitements et pourra, s'il le désire, spécifier ces derniers en créant ses propres patterns de conception.

En ce qui concerne la spécification des interactions entre les composants d'un SIC, nous avons conçu un pattern de conception générique permettant de modéliser ces interactions et applicable à la conception de tout SIC. Ainsi, ce pattern après adaptation permet de modéliser les interactions entre les composants – et plus particulièrement entre les agents – de l'architecture ACSIS. Contrairement aux travaux de [FERNANDEZ & ZHAO, 2000] ayant conçu des patterns spécifiant les interactions typiques entre les composants de leur architecture coopérative, notre pattern est générique et permet de spécifier les interactions inhérentes à tout SIC.

La section suivante présente les caractéristiques fonctionnelles du système d'information coopératif ACSIS dont nous avons dérivé certains patterns de conception de coopération.

3.1.2. Le système d'information coopératif ACSIS

Notre objectif n'est pas ici de présenter en détail les résultats de la phase de rétro-conception de l'architecture ACSIS (Agents pour la Coopération de Systèmes d'Information Sécurisés[84]), les modèles résultant de cette phase étant, en effet, intégrés et explicités dans les patterns de conception de coopération présentés en section suivante (§ 3.1.3).

Les caractéristiques de l'architecture ACSIS ont déjà été décrites dans les exemples d'utilisation inhérents aux patterns d'analyse et d'architecture de coopération. La figure suivante[85] (Cf. Figure

[84] Les aspects sécurité de l'architecture ACSIS n'ont pas été abordés dans notre travail.

[85] Cette figure représente l'architecture ACSIS selon quatre couches. En réalité, les ODD et LSem sont encapsulés dans les agents d'accès et les agents informationnels. Notre objectif étant de positionner, ici, un exemple se focalisant sur les deux premières couches.

39) illustre à l'aide d'un exemple simplifié les niveaux ACSIS et leurs interactions et introduit un exemple d'utilisation qui sera par la suite utilisé dans les patterns de conception de coopération.

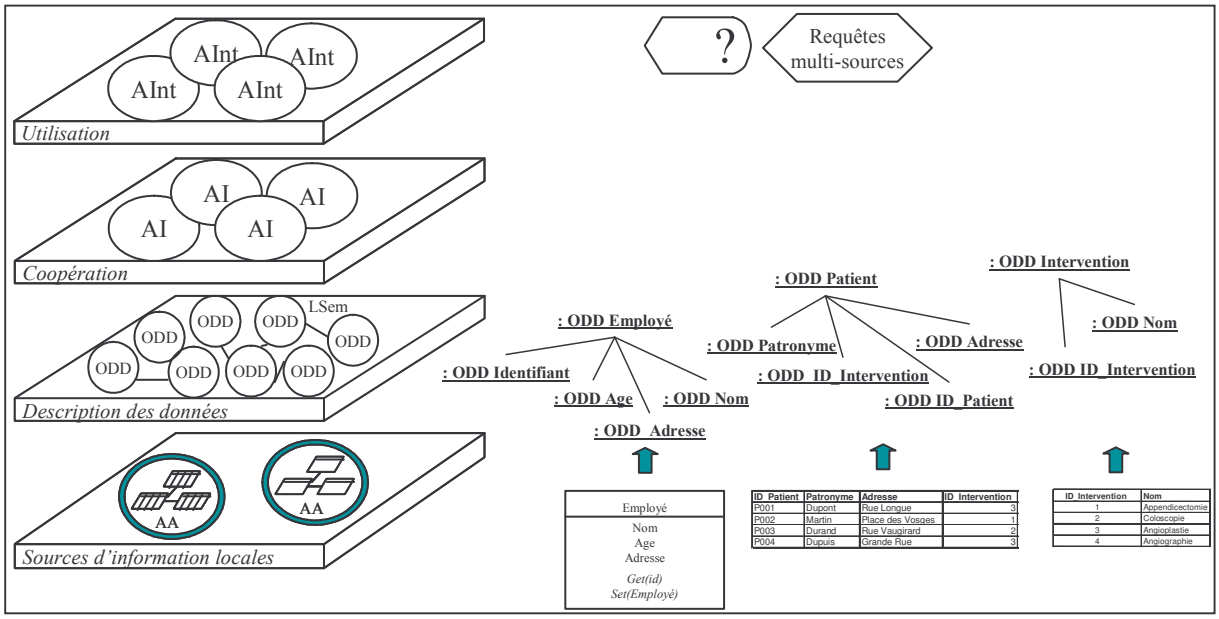


Figure 39 : Les niveaux ACSIS et la description des données locales.

Nous détaillons dans la section suivante (§ 3.1.3) les patterns de conception de coopération.

3.1.3. Présentation des patterns de conception de coopération

Notre volonté n'est pas ici de présenter tous les patterns de conception[86] dérivés de l'architecture ACSIS. A l'instar de [GANGULY & AL., 2001] et [FERNANDEZ & AL., 2003] qui ont conçu leurs patterns à partir des caractéristiques saillantes de leur architecture respective, nous présentons quelques patterns de conception typiques issus du SIC ACSIS.

Deux patterns permettent de spécifier plus amplement les concepts d'Objet Descriptif des Données (ODD) et de Liens Sémantiques (LSem) déjà décrits dans les exemples d'utilisation du pattern d'analyse « Référentiel de coopération » (Cf. section 3.2.3 du chapitre 3). En effet, si les patterns d'analyse ont permis de définir les entités caractéristiques du SIC ACSIS, les patterns de conception ont pour objectif de détailler certaines d'entre elles. Ces patterns sont le pattern « Objet Descriptif des Données » (Cf. Tableau 19) et le pattern « Lien sémantique » (Cf. Tableau 20). Le pattern « Objet Descriptif des Données » décrit le modèle d'objet descriptif, c'est-à-dire le modèle canonique dans lequel sont traduits les différents schémas locaux. Le pattern « Lien sémantique »

[86] Les patterns de conception de coopération proposés sont notamment présentés dans [COUTURIER, 2002] [COUTURIER, 2004] et [COUTURIER & DUBOIS, 2004].

décrit le modèle de lien sémantique, c'est-à-dire le modèle de représentation des liens schématiques, structurels et sémantiques entre objets descriptifs des données permettant ainsi de constituer le schéma de médiation[87].

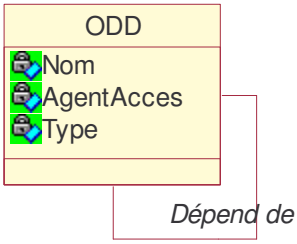
Deux patterns génératifs sont également proposés et permettent d'utiliser les patterns descriptifs précédents. Ces deux patterns sont le pattern « Traduction ACSIS » (Cf. Tableau 21) et le pattern « Création des liens sémantiques » (Cf. Tableau 22). Le pattern « Traduction ACSIS » concerne la traduction des schémas de données locaux dans le modèle canonique orienté objet et la génération des Objets Descriptifs des Données. Le pattern « Création des liens sémantiques » assure la génération des liens sémantiques permettant l'appariement des ODD en fonction de relations sémantiques, structurelles ou schématiques.

Les patterns de conception les plus importants permettent de décrire la dynamique entre les différents composants du SIC ACSIS. Nous avons conçu un pattern générique, appelé « Interaction d'agents » (Cf. Tableau 24) permettant de définir des protocoles d'interaction - et les messages inhérents - entre des agents (acteurs et modules fonctionnels) et des ressources (sources d'information, référentiel de coopération) non seulement du SIC ACSIS, mais aussi de tout autre SIC. Ce pattern modélise, dans l'exemple d'utilisation proposé, les interactions entre les agents artificiels, les sources d'information d'ACSIS et l'utilisateur. Un autre pattern est décrit, le pattern « contract-net protocol » (Cf. Tableau 23), utilisé dans le cadre d'ACSIS pour modéliser la tâche de coopération « Appel d'offre » (Cf. pattern « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » présenté en section 3.2.5 du chapitre 3), cette tâche nécessitant l'adaptation du protocole de passation de contrat spécifié par Smith [SMITH, 1980]. Notons que ce pattern peut-être appliqué pour modéliser certaines interactions (basées sur ce protocole) entre les composants d'autres SIC qu'ACSIS.

Ces patterns sont détaillés dans les sections suivantes (§ 3.1.3.1 à 3.1.3.6).

[87] Dans notre cas, le modèle de médiation est constitué des types d'objets ODD (modèle canonique) et LSem.

3.1.3.1. Pattern de conception de coopération « Objet Descriptif des Données »

Interface
Nom
Objet Descriptif des Données
Synonymes
ODD
Classification
Pattern de conception de coopération
Objectif
L'objectif de ce pattern est de représenter la structure d'un Objet Descriptif des Données. Ces objets servent à représenter les éléments des schémas locaux. REMARQUE — Ce pattern est non génératif.
Applicabilité
SIC ACSIS.
Solution
Modèle

Participants
ODD : Un ODD décrit un élément du modèle de données local (par exemple, pour le modèle relationnel : une relation, un attribut, un attribut clé primaire…).
A un participant commun avec
ODD : création des ODD et encapsulation dans un agent d'accès [Traduction ACSIS]
Implémentation

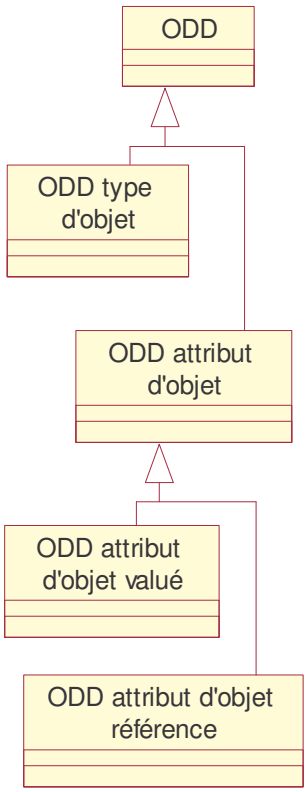
L'objectif de cette rubrique est de définir les interfaces des participants en IDL. Pour réutiliser le mécanisme de résolution des conflits de modèles de données (ODD), le développeur devra ainsi se conformer à ces interfaces pour définir ses propres objets. Nous présentons ci-dessous l'interface (partielle) du type d'objet ODD correspondant à ce pattern.
`interface ODD {`
`attribute string nom;`
`attribute string type;`
`attribute ODD oddpere;`

```
        attribute AgentAccess aaref;
        } ;
```

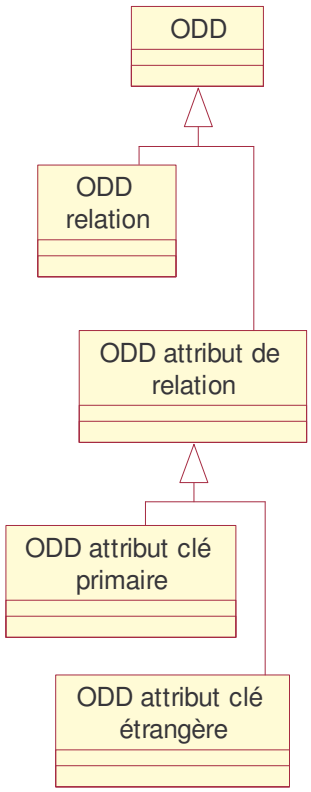
Exemples d'utilisation

Ce pattern est utilisé lors du traitement des conflits de modèle de données provenant de bases de données hétérogènes.

Modèle dans le cas d'une base de données orientée objet :



Modèle dans le cas d'une base de données relationnelle :



Conséquences

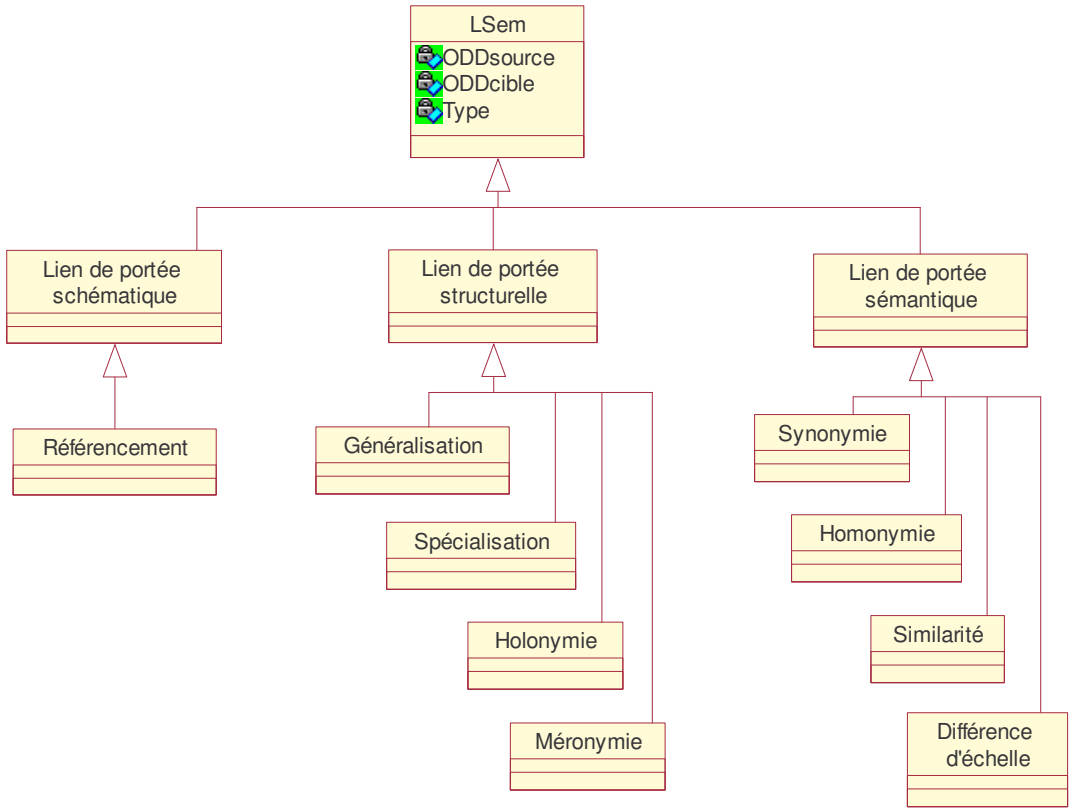
Pour pouvoir utiliser ce pattern et notamment sa partie « Implémentation », les différents éléments définis devront être conformes à l'interface IDL du pattern.

Tableau 19 : Pattern de conception de coopération « Objet Descriptif des Données »

3.1.3.2. Pattern de conception de coopération « Lien sémantique »

Interface
Nom
Lien sémantique

Synonymes
LSem
Classification
Pattern de conception de coopération
Objectif
L'objectif de ce pattern est de représenter la structure d'un lien sémantique. Un lien sémantique définit une relation structurelle ou sémantique entre deux ODD. REMARQUE — Ce pattern est non génératif.
Applicabilité
SIC ACSIS.
Solution
Modèle



Participants
LSem : Un lien sémantique est établi entre un ODD source et un ODD cible. Un lien peut intervenir entre deux ODD provenant de la même source d'information (liens sémantiques intra-bases) ou de sources d'informations différentes (liens sémantiques inter-bases). Les liens sémantiques assurent l'appariement des objets descriptifs des données en fonction de caractéristiques structurelles ou sémantiques et matérialisent ainsi la coopération des différentes sources d'information. Différents types de liens sont spécifiés dans l'architecture ACSIS et permettent de résoudre les

conflits de généralisation/spécialisation, d'agrégation, de synonymie, d'homonymie et d'échelle.

Type de lien	Portée	Signification
Référencement	Schématique	Décrit la relation entre un ODD de référencement et un ODD référencé.
Généralisation	Structurelle	Décrit un lien de spécialisation -> généralisation entre deux ODD (du terme spécialisé vers le terme général).
Spécialisation	Structurelle	Décrit un lien de généralisation -> spécialisation entre deux ODD (du terme général vers le terme spécialisé).
Holonymie	Structurelle	Décrit un lien de méronymie -> holonymie entre deux ODD (d'un élément agrégé vers un agrégat). Ex. : lien d'holonymie (roue, voiture).
Méronymie	Structurelle	Décrit un lien d'holonymie -> méronymie entre deux ODD (d'un agrégat vers un élément agrégé). Ex. : lien de méronymie (voiture, roue).
Synonymie	Sémantique	Décrit un lien d'équivalence sémantique entre deux ODD alors qu'ils ne disposent pas du même nom.
Homonymie	Sémantique	Décrit un lien de différence sémantique entre deux ODD disposant du même nom.
Similarité	Sémantique	Décrit un lien d'équivalence sémantique entre deux ODD.
Différence d'échelle	Sémantique	Décrit un lien de différence au niveau de l'échelle de deux ODD.

A un participant commun avec

LSem : création des Lsem et encapsulation dans un agent d'accès ou un agent informationnel [Création des liens sémantiques]

Implémentation

Nous présentons ci-dessous l'interface (partielle) du type d'objet LSem correspondant à ce pattern.

```
interface LSem {
            attribute ODD ODDsource;
            attribute ODD ODDcible;
            attribute string type;
            } ;
```

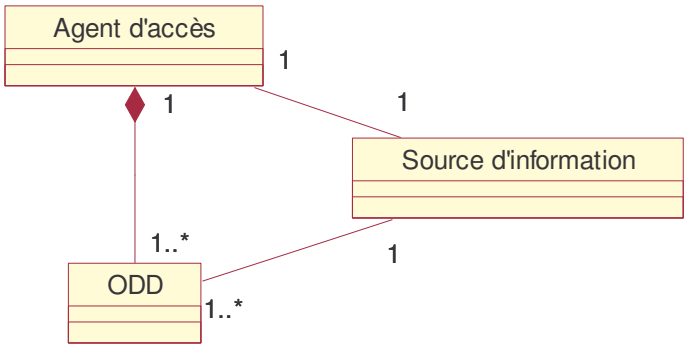
Exemples d'utilisation

Ce pattern est utilisé lors du traitement des conflits structurels et sémantiques provenant de bases de données hétérogènes.

Conséquences
Pour pouvoir utiliser ce pattern et notamment sa partie « Implémentation », les différents éléments définis devront être conformes à l'interface IDL du pattern.

Tableau 20: Pattern de conception de coopération « Lien sémantique »

3.1.3.3. Pattern de conception de coopération « Traduction ACSIS »

Interface
Nom
Traduction ACSIS
Classification
Pattern de conception de coopération
Objectif
L'objectif de ce pattern est de permettre la représentation du schéma exporté d'une source d'information sous la forme d'Objets Descriptifs des Données et, ainsi, de résoudre les conflits de modèles de données (homogénéisation des modèles de données locaux). Ce pattern est génératif.
Applicabilité
SIC ACSIS.
Solution
Modèle

Participants
Agent d'Accès: Chaque agent d'accès (AA) se charge de la traduction du schéma de la source d'information associée en ODD. La traduction d'un schéma exporté en un graphe d'ODD est réalisée en spécifiant des règles de traduction entre les éléments du modèle de données local et les types d'objet ODD. Chaque AA encapsule les ODD correspondant à sa source. *ODD :* Un ODD décrit un élément du schéma de données local. Les ODD assurent ainsi la description des données ou connaissances des sources d'information. Ce pattern utilise ainsi le pattern « Objet Descriptif des Données ».

A un participant commun avec

ODD : Présentation de la structure d'un ODD [Objet Descriptif des Données]

Implémentation

Nous présentons ci-dessous l'interface (partielle) de l'agent d'accès correspondant à ce pattern.

```
interface AgentAcces {
   attribute string nom;
   attribute string nom_source;
   attribute string path_source;
   // autres attributs…
   ODD oddtranslation (in string nom_concept,  // nom du concept
            in string type_concept,  // type du concept : relation,
                                     attribut de relation
            in AgentAcces aa_link,
            in ODD oddp)
      raises (UserException);
   // autres méthodes…
};
```

Exemples d'utilisation

Ce pattern est utilisé lors du traitement des conflits de modèles de données provenant de bases de données hétérogènes. Un exemple est donné dans le cas de la transformation de la structure d'un schéma relationnel en un graphe d'ODD.

Exemple :

Schéma relationnel :

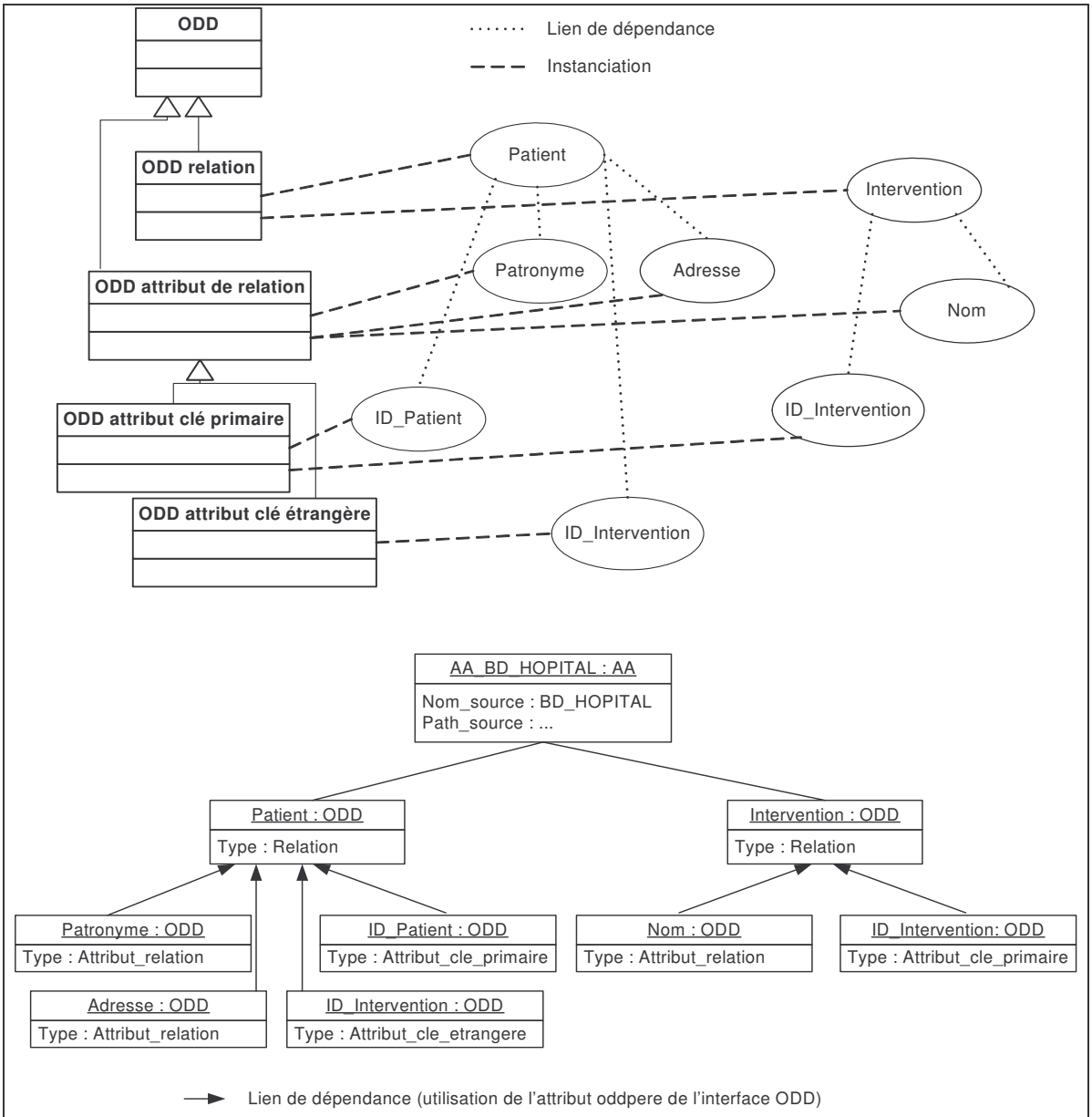
Relation Intervention (ID_Intervention, Nom)

Relation Patient (ID_Patient, Patronyme, Adresse, ID_Intervention)

Règles de traduction :

- Règle 1 : Table -> ODD Relation
- Règle 2 : Attribut de table -> ODD Attribut de relation
- Règle 3 : Clé étrangère -> ODD Attribut clé étrangère
- Règle 4 : Clé primaire -> ODD Attribut clé primaire

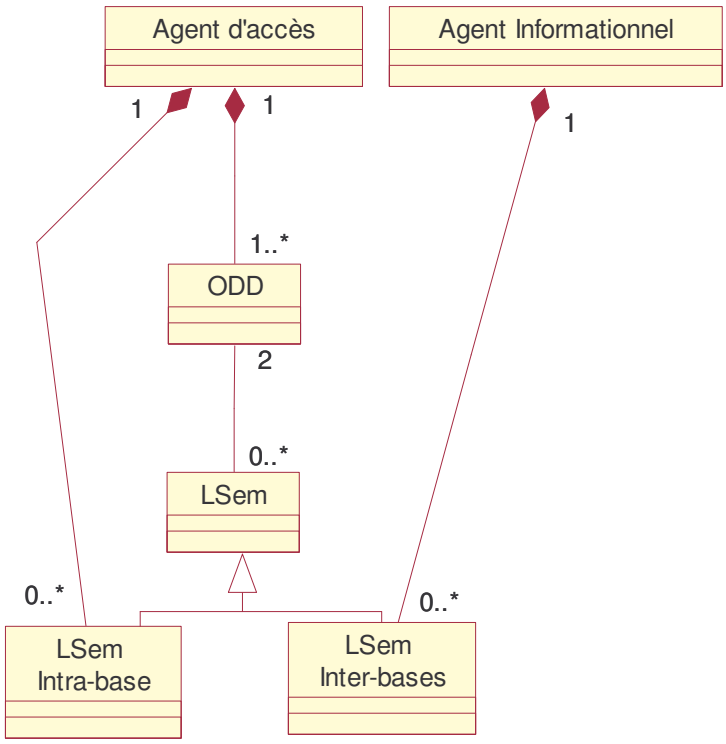
Graphes d'ODD :



Conséquences
Pour pouvoir utiliser ce pattern et notamment sa partie « Implémentation », les différents éléments définis devront être conformes à l'interface IDL du pattern.
Relations
Utilise
« Objet Descriptif des Données »

Tableau 21 : Pattern de conception de coopération « Traduction ACSIS »

3.1.3.4. Pattern de conception de coopération « Création des liens sémantiques »

Interface
Nom
Création des liens sémantiques
Classification
Pattern de conception de coopération
Contexte
Ce pattern ne peut être utilisé que si le pattern « Traduction ACSIS » a été précédemment appliqué.
Objectif
L'objectif de ce pattern est de permettre de constituer les liens sémantiques permettant de représenter les relations structurelles et sémantiques entre les Objets Descriptifs des Données et, ainsi, de résoudre certains conflits structurels et sémantiques. Ce pattern est génératif.
Applicabilité
SIC ACSIS.
Solution
Modèle

Participants
LSem : Un lien sémantique est établi entre un ODD Source et un ODD cible. Un lien peut intervenir entre deux ODD provenant de la même source d'information (liens sémantiques intra-

bases) ou de sources d'informations différentes (liens sémantiques inter-bases). Les liens sémantiques assurent l'appariement des objets descriptifs des données en fonction de caractéristiques structurelles ou sémantiques et matérialisent ainsi la coopération des différentes sources d'information. Ce pattern utilise ainsi le pattern « Lien sémantique ».

Agent d'Accès (AA): Chaque agent d'accès (AA) se charge de la création des liens sémantiques (LSem) intra-bases. Ces liens peuvent soit être directement détectés soit requérir l'intervention d'un expert du domaine qui aura pour tâche de les définir.

Agent Informationnel (AI): Chaque agent informationnel contient les liens sémantiques inter-bases concernant les sources d'information dont il a la charge. Les liens inter-bases sont définis manuellement par l'expert du domaine.

A un participant commun avec

LSem : Présentation de la structure d'un lien sémantique [Lien sémantique]

Implémentation

Nous présentons ci-dessous les interfaces (partielles) de l'agent d'accès et de l'agent informationnel correspondant à ce pattern.

```
interface AgentAcces {
   attribute string nom;
   attribute string nom_source;
   attribute string path_source;
   // autres attributs…
   LSem LSemCreation (in ODD ODDCi, in ODD ODDSo, in string type_lien)
      raises (UserException);
   // autres méthodes…
};

interface AgentInfo {
   attribute string nom;
   // autres attributs…
   LSem LSemCreation (in ODD ODDCi, in ODD ODDSo, in string type_lien)
      raises (UserException);
   // autres méthodes…
};
```

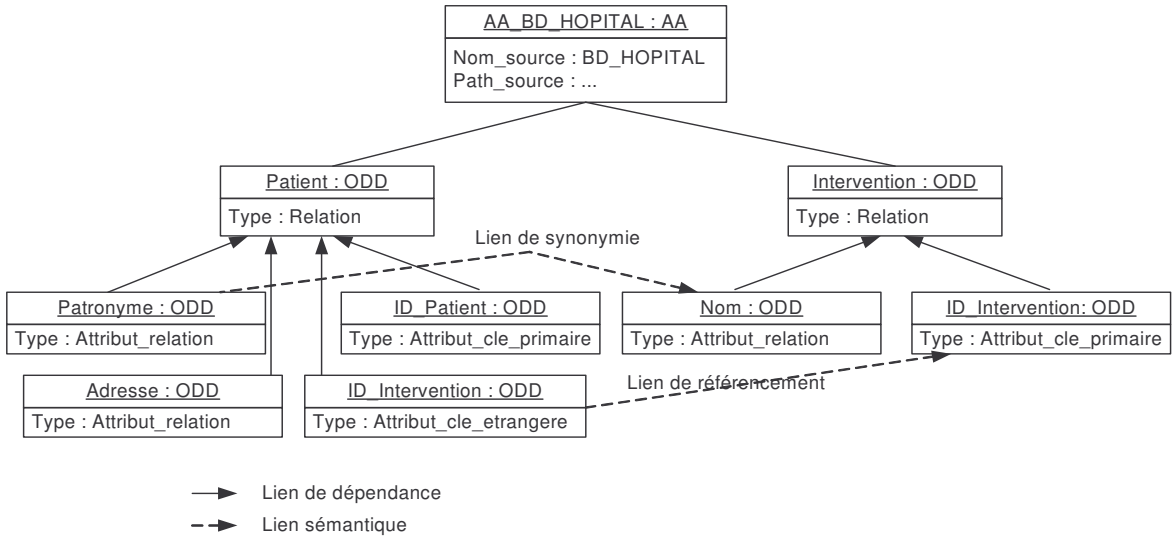
Exemples d'utilisation

Un exemple est donné dans le cas de la création de liens sémantiques intra-bases.

Schéma relationnel de l'exemple :

Relation Intervention (ID_Intervention, Nom)

Relation Patient (ID_Patient, Patronyme, Adresse, ID_Intervention)



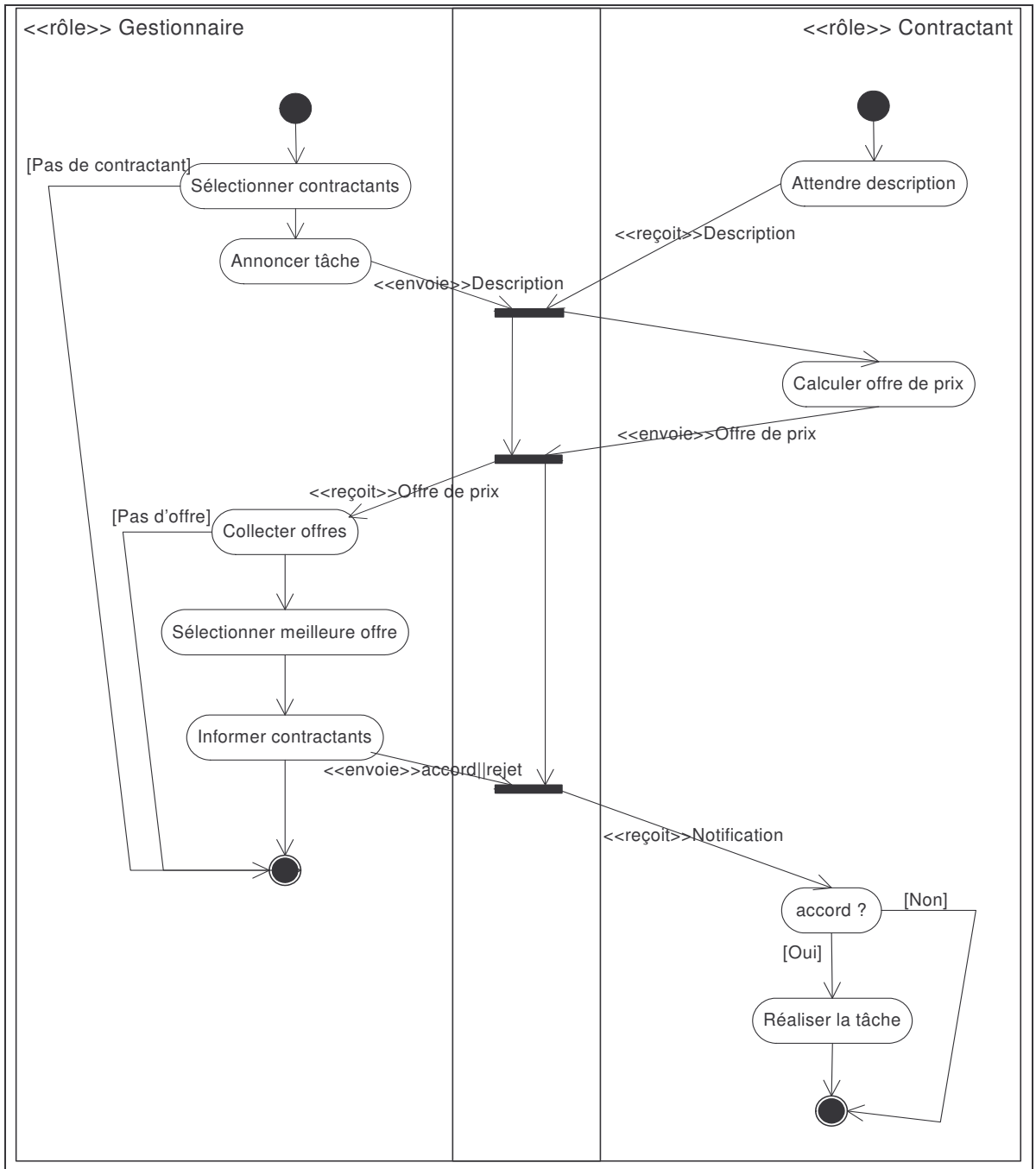
Conséquences
Pour pouvoir utiliser ce pattern et notamment sa partie « Implémentation », les différents éléments définis devront être conformes à l'interface IDL du pattern.
Relations
Utilise
« Lien sémantique »
Requiert
« Traduction ACSIS »

Tableau 22 : Pattern de conception de coopération « Création des liens sémantiques »

3.1.3.5. Pattern de conception de coopération « Contract-Net Protocol »

Interface
Nom
Contract-Net Protocol
Classification
Pattern de conception de coopération
Objectif
Ce pattern permet d'allouer une tâche spécifique à un des contractants potentiels afin de minimiser les coûts locaux. L'idée est que les participants calculent leur coût local de réalisation d'une tâche puis l'indiquent à un gestionnaire qui décide de l'allocation finale de la tâche à un contractant. Cela assure une allocation de ressources optimale dans le cas de tâches indépendantes.
Applicabilité
SIC ACSIS

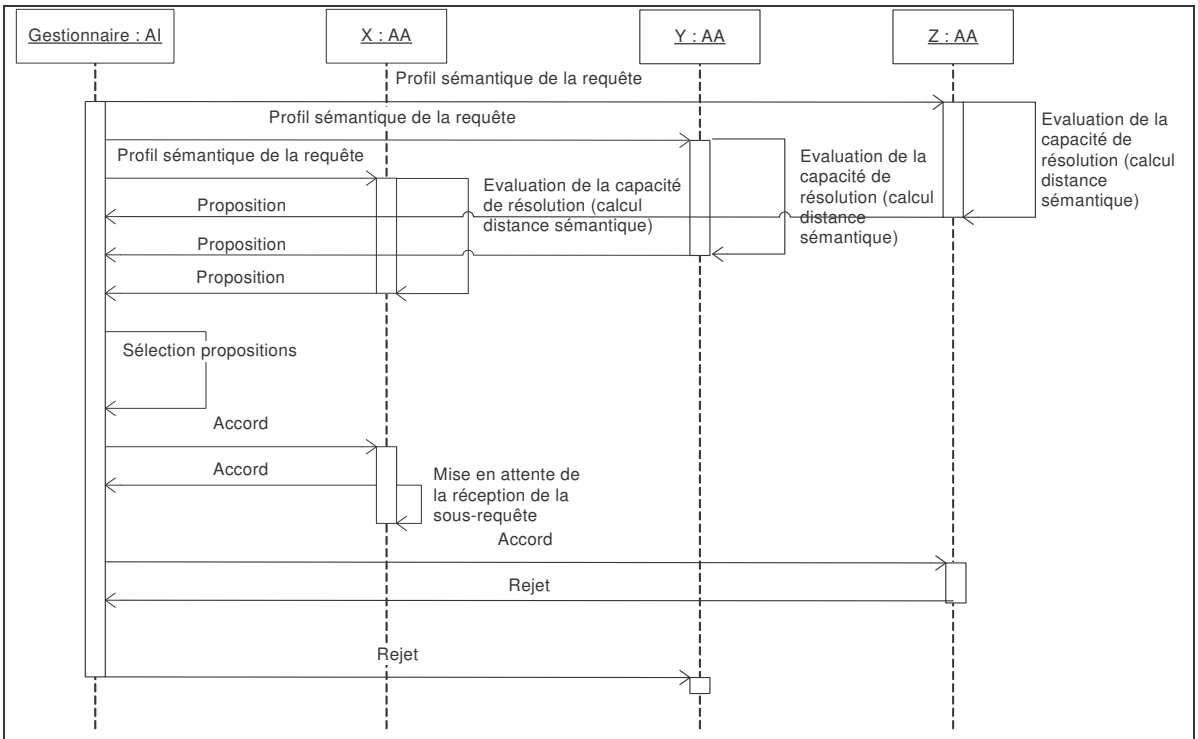
Solution
Démarche
Les principales étapes de ce protocole sont les suivantes : (1) Le gestionnaire ou demandeur formule son appel d'offre auprès de potentiels contractants ou fournisseurs. Une description de la tâche à exécuter est nécessairement jointe à l'appel d'offre. La diffusion de l'appel d'offre peut être générale (concerner tous les contractants accessibles) ou partielle. (2) Chaque contractant ayant été contacté détermine sa compétence (calcul de l'offre de prix) face à l'appel d'offre et envoie une proposition au demandeur. Cette réponse doit quantifier la compétence du fournisseur selon des règles unifiées pour permettre au demandeur d'évaluer de façon homogène les différentes propositions. (3) Le gestionnaire évalue les propositions et sélectionne la meilleure pour accorder le marché. Le fournisseur ainsi choisi est le bénéficiaire du contrat ; ce choix lui est signifié par un message émanant du demandeur. Le gestionnaire notifie également les autres fournisseurs du rejet de leur offre. (4) La dernière étape permet au fournisseur d'accepter le contrat ou de le refuser. Cette éventualité peut être due à un changement intervenu pendant le protocole au niveau du fournisseur. Ces étapes sont précisées dans le diagramme d'activités (d'après [LIND, 2002]) suivant :



Exemples d'utilisation

Ce pattern est utilisé (et adapté), dans le cadre de l'architecture ACSIS, préalablement au découpage de la requête globale en sous-requêtes, afin de lancer un appel d'offre dans le but de déterminer quel(s) fournisseur(s) d'information (agent d'accès) est (sont) capable(s) de participer à la résolution de la requête.

Lors de cet appel d'offre, se manifestent des interactions entre des agents informationnels et des agents d'accès. Nous représentons ces interactions à l'aide d'un diagramme de séquence.



Relations
Ressemble
« Contract-Net Protocol » [LIND, 2002] (Cf. section 6.1.1.1 du chapitre 2).

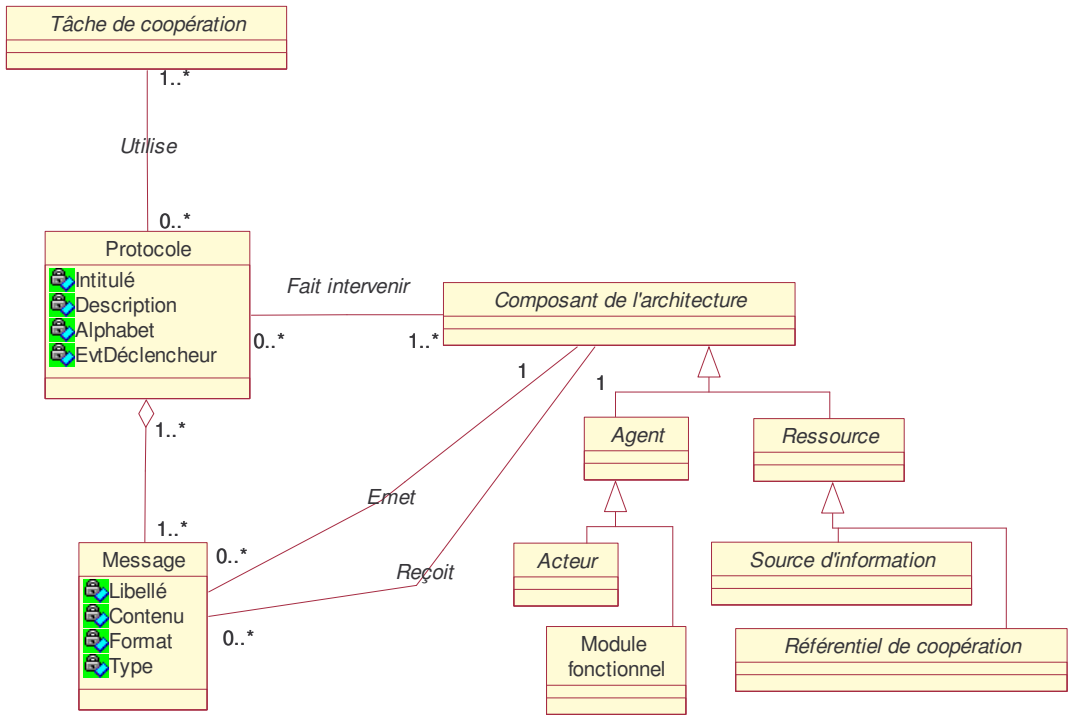
Tableau 23 : Pattern de conception de coopération « Contract-Net Protocol »

3.1.3.6. Pattern de conception de coopération « Interaction d'agents »

Interface
Nom
Interaction d'agents
Classification
Pattern de conception de coopération
Objectif
Ce pattern permet de représenter les protocoles d'interaction faisant intervenir les agents (acteurs et modules fonctionnels) et les ressources (référentiels de coopération et sources d'information) d'un SIC. Il permet également de représenter les messages constituant ce protocole d'interaction.
Applicabilité
SIC

Solution

Modèle



Ce diagramme de classes permet de représenter la structure statique des interactions entre les agents et ressources du SIC. Un diagramme de séquence permettant de spécifier les interactions entre des instances de classes peut également être utilisé pour décrire l'aspect chronologique de la communication et les scénarii de dialogue entre les agents et ressources. Ce diagramme de séquence, permettant de représenter les interactions entre objets, sera donc dépendant du contexte d'utilisation (typologie des agents ou des ressources, messages spécifiques,…).

Participants

Tâche de coopération : définit un événement de coopération à traiter par l'association entre un ensemble d'entrées, c'est-à-dire de données à traiter, et un ensemble de sorties, c'est-à-dire de résultats produits, et par l'indication de leurs caractéristiques respectives. Une tâche de coopération peut-être décomposée en sous-tâches représentant ainsi les différents événements ou sous-événements à traiter. Elle est caractérisée par un intitulé, une description et éventuellement les flux entrants et sortants (vis-à-vis des sources d'information ou du référentiel de coopération), les pré et post-conditions, ainsi que les pré et post-tâches.

Agent : participe à la réalisation ou au contrôle d'une ou plusieurs tâches de coopération. Un agent peut être un individu (ou acteur) ou un module fonctionnel (ex. : agent artificiel, objet,…) réalisant une ou plusieurs tâches de coopération.

Ressource : Une ressource qualifie toute entité qui peut être créée, transformée ou utilisée par une tâche de coopération. Une ressource est soit une source d'information coopérantes, soit un

référentiel de coopération.

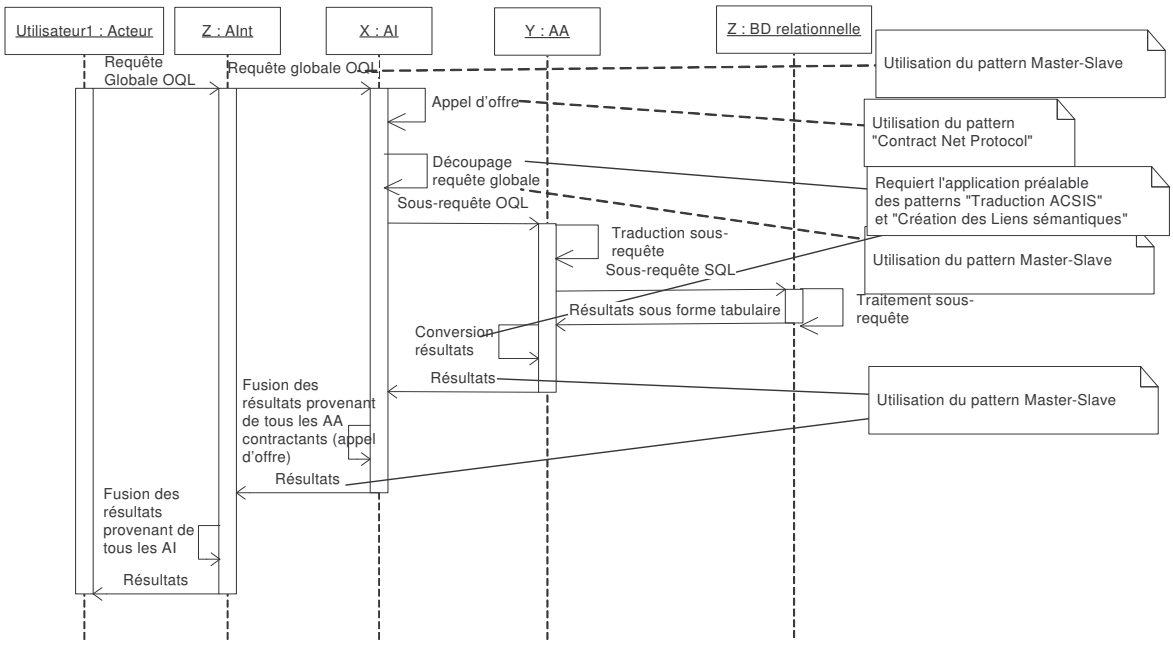
Composant : caractérise toute entité de type agent ou ressource du SIC intervenant dans des interactions.

Message : un message est constitué par le contenu des informations transmises. Il est caractérisé par son libellé, son contenu (l'information transmise), son format (message KQML, par exemple), son type (synchrone, asynchrone,…).

Protocole : un protocole est la spécification d'un dialogue impliquant des agents et des ressources. Il permet d'organiser les séquences de messages échangés en de véritables conversations structurées. Un protocole est caractérisé par son intitulé, sa description (règles,…), son alphabet (série de messages ordonnancée du protocole) et l'événement déclencheur (événement ou liste d'événements déclenchant le protocole).

Exemples d'utilisation

Nous présentons, à l'aide du diagramme de séquence suivant, un exemple modélisant les interactions entre les composants du SIC ACSIS lors du traitement d'une requête globale. Ce diagramme est basé sur les agents, ressources (source d'information) et tâches de l'architecture ACSIS définis lors de l'application des patterns d'analyse « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » et « Sources d'information gérées » (Cf. chapitre 3)



Certaines interactions entre les agents nécessitent l'utilisation du pattern Master-Slave (délégation de tâches) de Aridor & Lange [ARIDOR & LANGE, 1998] (Cf. section 6.1.1.1 du chapitre 2). Le découpage de la requête globale nécessite que les patterns « Traduction ACSIS » et « Création des liens sémantiques » aient été appliqués afin de constituer le schéma de médiation.

Relation
Conflit
« Interface connection » [FERNANDEZ & ZHAO, 2000] (Cf. section 6.2.1.1 du chapitre 2). REMARQUE — les patterns « Interaction d'agents » et « Interface connection » reposent sur un même problème (communication entre les composants d'un SIC), mais fournissent des solutions différentes, la nôtre étant indépendante d'un système d'information coopératif spécifique.

Tableau 24 : Pattern de conception de coopération « Interaction d'agents »

3.2. Patterns de support technique

Les patterns de support technique sont des patterns qui décrivent comment construire une application à partir des patterns de conception de coopération. Ce sont soit des patterns dédiés à l'implémentation des entités spécifiées dans les patterns de conception soit des patterns dédiés à leur implantation.

Les patterns dédiés à l'implémentation sont, généralement, des patterns de conception spécifiques à un paradigme (objet, agent artificiel,…). Dans le cas, par exemple, d'un SIC spécifiant des agents artificiels en charge de la réalisation des traitements coopératifs, ces patterns vont permettre de construire un système multi-agents, de spécifier les interactions entre les agents (spécification des modèles de coopération ou collaboration),… Les patterns agents de ce type ont été présentés précédemment (Cf. section 6.1.1.1 du chapitre 2). Nous utilisons, par exemple, dans le cadre du pattern de conception de coopération « Interaction d'agents », le pattern « Master-Slave » d'Aridor & Lange [ARIDOR & LANGE, 1998] pour définir les mécanismes de délégation de tâches entre agents (Cf. section 3.1.3.6). Cependant, d'autres patterns agent ou des patterns objet explicitant, par exemple, les communications (synchrone, asynchrone,…) entre objets peuvent être utilisés en fonction des paradigmes inhérents au SIC.

Les patterns dédiés à l'implantation sont spécifiques aux plates-formes d'exécution ou aux langages de programmation. Ces patterns peuvent être des patterns de conception, mais sont la plupart du temps des idiomes (idiomes Java, idiomes CORBA,…). Ils sont fonction de l'environnement de développement choisi par le développeur d'applications. Deux exemples de patterns de support technique de ce type sont présentés dans les sections suivantes. Ces patterns assurent la transformation des modèles de conception indépendants des plates-formes et spécifiés dans certains patterns de conception de coopération en des modèles dépendants des plates-formes techniques. Le premier pattern de support technique (§ 3.2.1) permet de définir la démarche de transformation des modèles liés aux patterns de conception de coopération. Le deuxième (§3.2.2) explicite la traduction

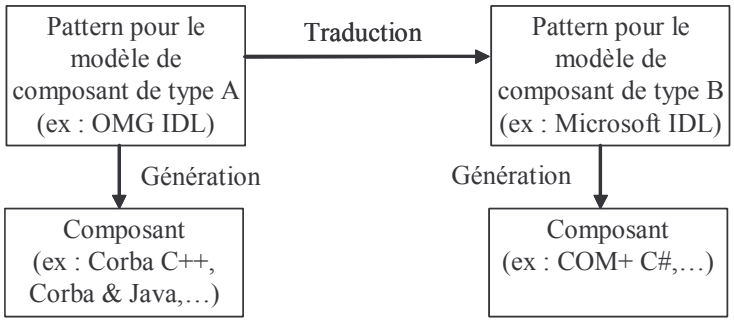
des définitions d'interface OMG IDL[88] vers Microsoft IDL[89] (MIDL) afin de pouvoir générer l'interface des composants spécifiés dans certains patterns de conception en MIDL.

3.2.1. Pattern de support technique « Démarche de transformation de modèle »

Interface
Nom
Démarche de transformation de modèle
Classification
Pattern de support technique
Objectif
L'objectif de ce pattern est de définir une démarche pour traduire la rubrique « Implémentation » des patterns de conception de coopération, spécifiée en OMG IDL en une description d'interface relative à un autre modèle de composant logiciel.
Applicabilité
SIC ACSIS
Solution
Démarche

[88] L'OMG IDL est devenu un langage de description d'interface normalisé (norme ISO 14750). Il est utilisable pour spécifier l'interface d'un composant ou d'un objet quel que soit son modèle sous-jacent.

[89] Langage de description d'interface de Microsoft. Pour de plus amples détails sur Microsoft IDL, nous renvoyons à [PLATT, 2000].



Cette traduction réalisée au niveau du pattern permet de traduire la rubrique « implémentation », spécifiée en OMG IDL, de la solution des patterns de coopération en un autre langage de définition d'interface ou un autre modèle de composant. Le pattern ainsi traduit associe toujours solution et documentation de cette solution.

REMARQUE — Ce pattern ne définit pas (mais utilise) de règles de transformation entre modèles de composants ou entre langages de définition d'interface (OMG IDL vers Microsoft IDL, par exemple).

Relation
Utilise
« Traduction des interfaces spécifiées en OMG IDL vers Microsoft IDL »

Tableau 25 : Pattern de support technique « Démarche de transformation de modèle »

3.2.2. Pattern de support technique « Traduction des interfaces spécifiées en OMG IDL vers Microsoft IDL »

Interface
Nom
Traduction des interfaces spécifiées en OMG IDL vers Microsoft IDL
Classification
Pattern de support technique
Objectif
Ce pattern décrit le mécanisme de traduction de la sémantique de l'OMG IDL en Microsoft IDL (règles de mapping). Ce pattern ne présente que les différences de sémantique entre les deux langages de définition d'interfaces.
Applicabilité
SIC ACSIS.
Solution
Démarche

- Règles de mapping pour les types de données de base et les chaînes de caractères (OMG IDL -> MIDL) : octet->unsigned char, Typedef string->Typedef [string, unique] char *
- Mapping d'un attribut OMG IDL en un attribut MIDL :
  - Un attribut en lecture seule se transforme en une propriété ayant une unique méthode pour récupérer sa valeur.
  - Un attribut en lecture/écriture se transforme en une propriété ayant deux méthodes, l'une permettant de définir sa valeur et l'autre de la lire.
- Mapping des opérations des interfaces OMG IDL en MIDL :
  - Opération -> Méthode
  - Direction des paramètres des opérations : in -> [in], out -> [out], inout -> [in,out], type "return" -> paramètre [out] (dernier paramètre de la signature).
- Mapping des hiérarchies d'interface :
  - chaque interface OMG IDL est précédée par la lettre "I" dans la définition MIDL,
  - Si l'interface est liée à un module IDL en utilisant "::", ces caractères sont remplacés par un underscore (par exemple, mymodule::myinterface -> Imymodule_myinterface),
  - Interface OMG IDL sans parent (pas d'héritage) -> interface MIDL dérivée de l'interface "IUnknown",
  - Interface OMG IDL héritant d'une interface parent unique -> interface MIDL dérivée du mapping de l'interface parent,
  - Interface IDL héritant de plusieurs interfaces parent -> interface dérivée de l'interface "IUnknown".
- Mapping des types complexes : struct -> typedef struct, union -> typedef union, sequence -> struct, any -> (pas de correspondance).
- Mapping des modules : Interface OMG IDL intégrée dans un module -> préfixation du nom de l'interface par le nom du module et un underscore. L'interface MIDL est dérivée de l'interface "IUnknown".

Tableau 26: Pattern de support technique «Traduction des interfaces spécifiées en OMG IDL vers Microsoft IDL »

3.3. Réutilisation des patterns de conception de coopération et des patterns de support technique

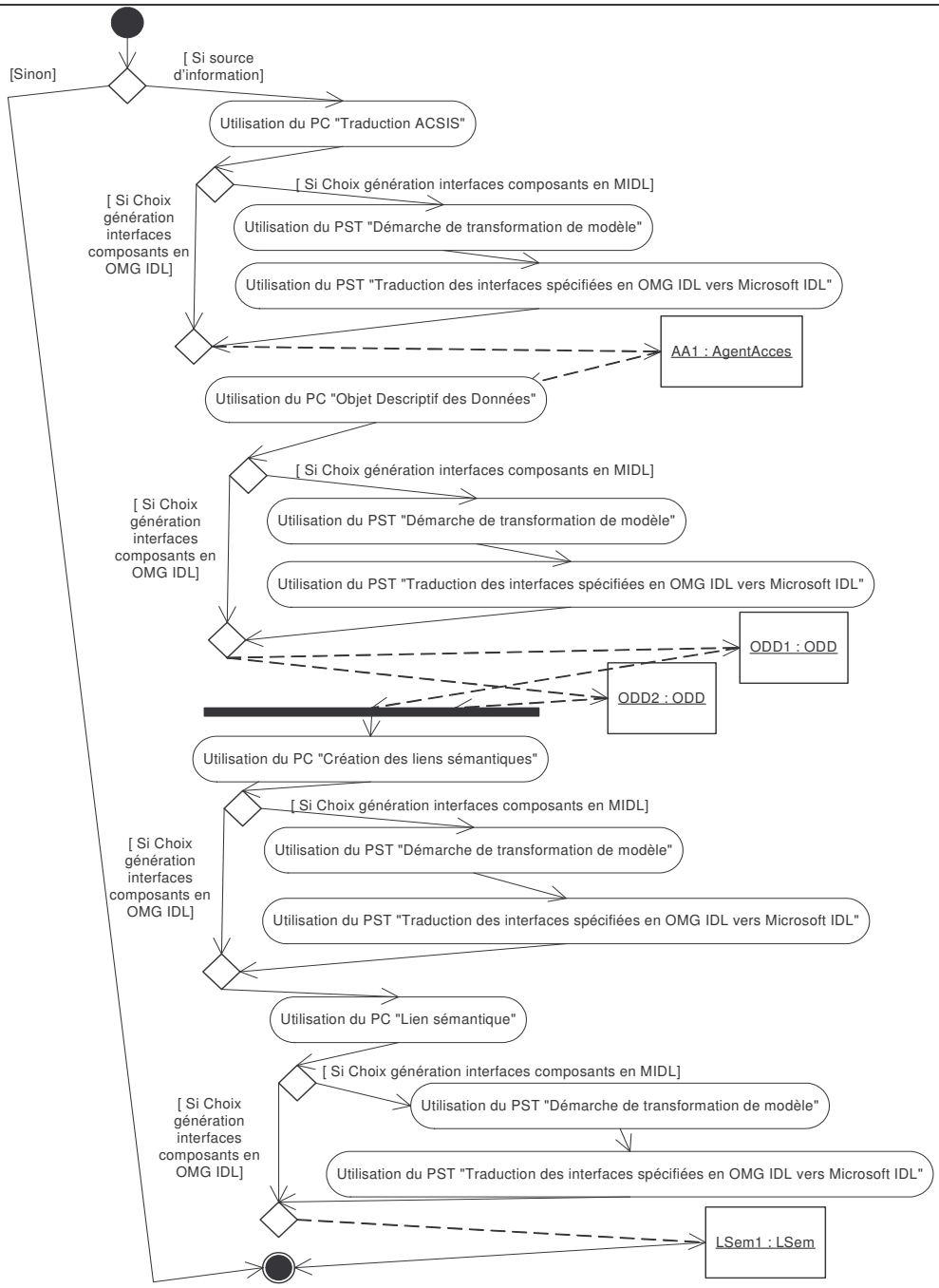
Les patterns de conception de coopération que nous avons conçus servant principalement à faciliter la réutilisation du SIC ACSIS, ceux-ci sont réutilisables lors du développement de ce système d'information coopératif. Il est peut alors être utile voire nécessaire de combiner la réutilisation des patterns de conception liés à ACSIS avec celle des patterns de support technique.

Notons cependant que le pattern de conception « Interaction d'agents » est également utilisable seul pour modéliser les interactions entre les entités (agents ou ressources) spécifiées dans tout SIC. Ainsi, ce pattern peut être réutilisé pour faciliter la conception (partielle, i.e. limitée à la modélisation des interactions) d'autres SIC. Il nécessite l'application préalable de la plupart des patterns d'analyse de coopération ("Modélisation des sources d'information gérées"et "Modélisation du référentiel de coopération" et "Modélisation des tâches de coopération effectuées par les agents" ou "Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas") (Cf. chapitre 3).

La réutilisation de ces patterns s'effectue par adaptation et/ou instanciation de tout ou partie du modèle fourni dans la solution de chaque pattern.

Comme pour les patterns d'analyse et les patterns d'architecture de coopération, la réutilisation des patterns de conception de coopération et des patterns de support technique est facilitée par la spécification de deux patterns de support d'utilisation représentant, d'une part, les patterns de conception et les patterns de support technique à utiliser pour créer la couche de description des données ACSIS (les objets descriptifs des données et les liens sémantiques) (Cf. Tableau 27) et, d'autre part, les patterns de conception à utiliser pour avoir une représentation des interactions entre les composants de l'architecture ACSIS (Cf. Tableau 28).

Interface
Nom
Création de la couche de description des données ACSIS.
Classification
Pattern de support d'utilisation.
Objectif
Ce pattern permet de préciser les patterns à utiliser dans le cadre de la création de la couche de description des données (ou schéma de médiation) ACSIS (ODD et Liens sémantiques).
Applicabilité
Système de patterns de conception de coopération ^ Système de patterns de support technique
Solution
Démarche



REMARQUE — PC : pattern de conception de coopération ; PST : pattern de support technique.

Relations
Utilise
- Patterns de conception de coopération : « Traduction ACSIS », « Objet Descriptif des Données », « Création des liens sémantiques » et « Lien sémantique ». - Patterns de support technique : « Traduction entre modèles de composants », « Traduction OMG IDL vers Microsoft IDL ».

Tableau 27: Pattern de support d'utilisation « Création de la couche de description des données ACSIS »

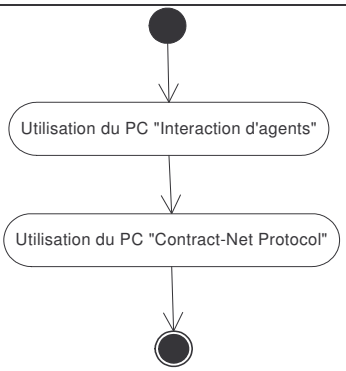
Interface
Nom
Représentation des interactions entre les composants de l'architecture ACSIS.
Classification
Pattern de support d'utilisation.
Objectif
Ce pattern précise les patterns à utiliser pour d'obtenir une représentation des interactions entre les composants de l'architecture ACSIS.
Applicabilité
Système de patterns de conception de coopération
Solution
Démarche

REMARQUE — PC : Pattern de conception de coopération
Relations
Utilise
« Interaction d'agents » « Contract-Net protocol »

Tableau 28 : Pattern de support d'utilisation « Représentation des interactions entre les composants de l'architecture ACSIS »

4. Conclusion

Ce chapitre a permis de présenter les patterns dédiés aux phases de conception et d'implantation des systèmes d'information coopératifs, à savoir les patterns d'architecture de coopération, les patterns de conception de coopération et les patterns de support technique.

Les *patterns d'architecture de coopération* s'appliquent au début de la phase de conception du processus d'ingénierie de SIC. Ils proposent des modèles d'architecture "génériques" permettant de

représenter l'architecture globale d'un SIC à concevoir, i.e. de positionner les différents acteurs, référentiels et sources d'information spécifiés à l'aide des patterns d'analyse de coopération. Remarquons que les patterns d'architecture sont à la fois utilisables pour représenter les sous-systèmes et leurs interactions inhérents à un nouveau SIC à concevoir ou ceux d'un SIC existant.

Les *patterns de conception de coopération* s'appliquent pour définir la conception détaillée d'un système d'information coopératif. Ils permettent de raffiner les entités métier spécifiées à l'aide des patterns d'analyse de coopération et plus spécialement de préciser leurs traitements et interactions. Ces patterns sont, pour la plupart, spécifiques au SIC ACSIS et permettent ainsi de faciliter la réutilisation de ce système d'information coopératif. Un pattern de conception générique permet cependant de spécifier les interactions entre les composants d'un système d'information coopératif.

Les *patterns de support technique* sont des patterns génériques qui décrivent comment construire une application à partir des patterns de conception de coopération. Ces patterns constituent ainsi des structures réutilisables dédiées à l'implémentation et à l'implantation des entités spécifiées dans les patterns de conception.

Enfin, des *patterns de support d'utilisation* sont également proposés et permettent de réutiliser les systèmes de patterns d'architecture, de conception de coopération et de support technique. Ils constituent des points d'entrée pour la mise en œuvre de ces systèmes.

Notre travail de recherche concernant les patterns présentés dans ce chapitre n'est cependant pas totalement abouti. Il nous reste à valider les patterns d'architecture et de conception de coopération génériques. Les exemples d'utilisation fournis dans ces patterns et modélisant les systèmes d'information coopératifs existants ont permis de valider leur utilité, mais aussi partiellement les problèmes identifiés et solutions proposées. Tout comme les patterns d'analyse de coopération, ces patterns seront prochainement appliqués et validés dans le cadre du projet TRAFIC de la région Rhône-Alpes. Une autre voix d'amélioration concerne également la validation et l'extension des patterns de conception de coopération liés à l'architecture ACSIS à d'autres contextes ou systèmes d'information coopératifs (généralisation de certaines entités de conception ou traitements applicables à quelques SIC) afin d'améliorer la généricité et la portée des solutions proposées.

Bilan & perspectives

1. Bilan

Ce mémoire de thèse a permis de mettre en place un cadre méthodologique pour l'ingénierie des systèmes d'information coopératifs dédiés à la coopération de sources d'information.

Les constats que nous avons formulés et à partir desquels nous avons défini notre problématique étaient :

- la grande variété des systèmes d'information coopératifs proposés jusqu'alors et présentant leurs propres concepts permettant de construire des processus coopératifs entre applications et/ou sources d'information et de traiter les conflits inhérents,
- l'absence d'aide au développement des systèmes d'information coopératifs,
- la difficulté de réutiliser les SIC existants (tout ou partie).

Partant de ces constats, notre travail de recherche a eu plus précisément pour objectif de mettre en place un cadre méthodologique pour l'ingénierie des systèmes d'information coopératifs basée sur la réutilisation de patterns. En effet, une étude des différents types d'avoirs logiciels réutilisables susceptibles de répondre à notre problématique nous a permis d'opter pour une démarche d'ingénierie de SIC basée sur la réutilisation d'avoirs de ce type.
Ce cadre méthodologique a proposé d'une part, la conception de patterns et, d'autre part, de réutiliser ceux-ci lors du développement des SIC. Ainsi deux activités du processus d'ingénierie de SIC ont pu être distinguées :

- l'activité d'*ingénierie des patterns* (développement d'un système réutilisable) a consisté à identifier les connaissances réutilisables, à les spécifier et les organiser sous formes de patterns. Cette activité a ainsi permis de définir des modèles de spécifications de différents niveaux d'abstraction couvrant la grande variété et la complexité des concepts gérés dans les SIC et d'assurer un continuum de transformation dans les spécifications.

- l'activité d'*ingénierie des SIC* (développement d'un système par réutilisation), proprement dite, ayant pour objectif la recherche, la sélection, l'adaptation et l'intégration des patterns définis lors de l'étape précédente afin de spécifier et développer le SIC. Cette activité a consisté à mettre en place une démarche générale de développement des SIC, par réutilisation de patterns, couvrant l'ensemble des phases du processus d'ingénierie (analyse, conception, implantation) et permettant d'accélérer et de faciliter ce développement en réutilisant les savoir et savoir-faire mis en oeuvre dans des SIC existants.

L'activité d'ingénierie des patterns, qui a constitué la part la plus importante de notre travail de recherche, nous a permis de spécifier un ensemble de patterns, regroupés en systèmes, dédiés à l'ingénierie des systèmes d'information coopératifs. Ces patterns sont de cinq types :

- Des *patterns d'analyse de coopération*, pour identifier et représenter les aspects fonctionnels du SIC à concevoir, c'est-à-dire les concepts sous-jacents et les entités métier. Ils permettent de construire le modèle d'analyse du SIC. Ils peuvent également servir, par exemple, à comparer voire classifier les SIC existants en fonction de critères précis.
- Des *patterns d'architecture de coopération* proposant des modèles d'architecture "génériques" permettant de représenter l'architecture globale - d'un point de vue structurel et organisationnel - du SIC à concevoir.
- Des *patterns de conception de coopération*, pour représenter les entités informatiques – ainsi que leurs traitements et interactions – associés aux diverses entités métiers identifiées à l'aide des patterns d'analyse de coopération. Ces patterns sont cependant pour la plupart liés au SIC ACSIS développé par l'équipe MODEME.
- Des *patterns de support technique* décrivant comment construire une application à partir des patterns de conception de coopération. Ces patterns constituent des structures réutilisables dédiées à l'implémentation et à l'implantation des entités spécifiées dans les patterns de conception de coopération.
- Des *patterns de support d'utilisation* permettant de guider le concepteur ou le développeur de SIC lors de la réutilisation des différents patterns précédents et constituant le point d'entrée de chaque système de patterns. Ils forment aussi des guides d'aide au développement de SIC par réutilisation de patterns.

Les patterns d'analyse, d'architecture et de conception modélisent notamment des fragments de modèles permettant de représenter les divers aspects du SIC à concevoir. Les modèles de spécification du SIC seront ainsi obtenus par adaptation et assemblage de fragments de modèles, et,

ce, par réutilisation des divers patterns présentés. Par ailleurs, le formalisme que nous avons proposé pour représenter les patterns favorise leur sélection, leur recherche et leur imbrication (rubriques Relation) et permet de respecter le caractère génératif (génération de composants) de certains patterns de conception de coopération.

L'activité d'*ingénierie des SIC* est basée sur une démarche générale de développement de ces systèmes, par réutilisation des patterns précédents, ces patterns couvrant l'ensemble des phases du processus d'ingénierie (analyse, conception, implantation). Un outil support de réutilisation permettant de faciliter partiellement la réutilisation et la mise en œuvre des patterns lors du développement de SIC a également été proposé.

En résumé, les principaux apports de notre travail de recherche sont :

- La spécification de patterns pour l'ingénierie des SIC qui capitalisent les savoir et savoir-faire mis en oeuvre lors de l'ingénierie des SIC et permettent de structurer, gérer et réutiliser ces connaissances.
- La conceptualisation de nouveaux types de patterns, les patterns de support d'utilisation et les patterns de support technique dérivés du concept de pattern de support de Fowler [FOWLER, 1997] qui traitent des problèmes qui interviennent quand un système informatique doit être construit à partir des patterns d'analyse, d'architecture et de conception de coopération. Ils décrivent comment utiliser ces patterns, comment les assembler et les mettre en oeuvre afin d'obtenir l'architecture ciblée.
- Une contribution à la structuration des connaissances relatives au domaine de la coopération de SI en un référentiel. Ce référentiel de domaine délimite et classifie l'ensemble des concepts ou entités gérés dans les SIC, précise la terminologie et la sémantique associées à ces concepts et identifie leurs problèmes de structuration et de gestion.
- La spécification d'un environnement permettant de faciliter – certes, partiellement – le développement de SIC par réutilisation des patterns de coopération.
- Et plus globalement, la contribution - avec d'autres auteurs - à l'instauration d'une vision, qui nous semble novatrice, pour concevoir et développer les systèmes d'information coopératifs. L'approche que nous proposons repose sur un ensemble de patterns dédiés à la coopération de SI et sur la présence d'un atelier facilitant la réutilisation de ces patterns lors de l'ingénierie de SIC.

2. Perspectives

Les perspectives de notre travail de recherche peuvent être classées selon deux axes : un axe d'extension du travail proposé et un axe d'élargissement de notre domaine de recherche.

Concernant le premier axe, il serait intéressant dans un premier temps d'étendre les systèmes de patterns que nous proposons par d'autres patterns soit les raffinant, soit venant en complément. Cette complémentation permettrait ainsi de transformer nos systèmes de patterns en de véritables langages. Un langage de patterns dédié à la coopération de systèmes d'information se doit, en effet, d'être le plus complet possible si l'on veut pouvoir s'en servir comme aide à la conception et au développement de tout type de SIC. Il serait notamment intéressant d'ajouter des patterns (de type patterns de support technique) permettant de faciliter la mise en œuvre des patterns de conception que nous avons conçus.
Comme nous l'avons déjà indiqué, nos patterns nécessitent également un travail de validation et de réutilisation sur plusieurs projets. En outre, les patterns que nous avons décrits ne nous semblent pas tous aboutis et nécessitent d'être affinés et confrontés à l'usage de la communauté. En effet, un pattern n'est jamais vraiment achevé : dans le sens où il s'agit d'une forme déclarative de l'expertise et que cette expertise s'affine et se modifie avec le temps.
Enfin, il nous semble important de compléter l'environnement de support de réutilisation proposé (ARRP) en lui intégrant des fonctionnalités liées à l'adaptation et l'instanciation des patterns. Notre objectif, à terme, est, en effet, de spécifier et développer un atelier de génie logiciel (AGL) assistant la création de systèmes d'information coopératifs, sous forme de composants, à partir de l'application de patterns d'analyse et de conception génériques et indépendants des plates-formes de développement. Notre AGL permettrait ainsi d' "appliquer" les principes de la démarche MDA à la construction de SIC (transformation des modèles d'analyse et de conception génériques et indépendants des systèmes d'exécution ciblés spécifiés dans les patterns en des modèles d'implantation dépendants des architectures techniques).

Nos pistes d'élargissement pourraient tout d'abord consister à modéliser des patterns dédiés à la sécurité des accès dans les systèmes d'information coopératifs. Ce futur travail pourrait ainsi décrire les modèles d'accès conçus au cours de travaux de recherche antérieurs de l'équipe MODEME [DISSON 2001] sous formes de patterns de conception dédiés à la sécurité des SIC (patterns de domaine). Il s'agirait, là aussi, d'associer à ces patterns de domaine des patterns de support technique génériques existants et dédiés aux aspects techniques de la sécurité des systèmes d'information (par exemple, des patterns relatifs au modèle de sécurité discrétionnaire RBAC[90]

[90] Role-Based Access Control

[FERNANDEZ & PAN, 2001] ou des patterns relatifs à l'authentification des utilisateurs [BLAKLEY & HEATH, 2004]).

En outre, nous pensons qu'il serait intéressant, et ce, même si des travaux existent déjà (par exemple, les patterns de conception liés au projet ECOO [TATA & AL., 1999] [TATA, 2002] [BAÏNA & AL., 2003] ou ceux de [LOPISTEGUY & AL., 2003],…) de spécifier des patterns (dont certains ne seraient que des raffinements ou des extensions des patterns que nous avons déjà conçus) relatifs aux deux autres facettes des systèmes d'information coopératifs, à savoir la facette "travail collaboratif" (workflow, CSCW,…) et la facette organisationnelle (processus métiers, objectifs stratégiques,…).

Enfin, d'un point de vue plus théorique, il nous semblerait intéressant de revisiter et approfondir la problématique de la génération de composants à partir de patterns.

Bibliographie

[AAMODT & PLAZA, 1994] AAMODT A., PLAZA E. — Case-based reasoning: Foundational issues, methodological variations, and system approaches, *Artificial Intelligence Communications*, IOS Press, march (1994).

[AARSTEN & AL., 1996] AARSTEN A., BRUGALI D., MENGA G. — Patterns for Cooperation, *3rd Conference on Pattern Languages of Programming (Plop'96)*, Monticello, Illinois, USA, September (1996).

[ABELSON & AL., 1985] ABELSON H., SUSSMAN G., SUSSMAN J. — *The Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, USA, (1985).

[ABRIAL & AL., 1991] ABRIAL J.R., LEE M.K.O., NEILSON D.S., SCHARBACH P.N., SORENSEN I.H. — The B-method (software development), *Formal Software Development Methods, 4th International Symposium of VDM Europe Proceedings*, eds. S. Prehn, S. Toetenel, Springer-Verlag, Berlin, Germany, p. 398-405, October (1991).

[ADALI & AL., 1996] ADALI S., CANDAN K.S., PAPAKONSTANTINOU Y., SUBRAHMANIAN V.S. — Query caching and optimization in distributed mediator systems, in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June (1996).

[ADAMS & AL., 1995] ADAMS M., COPLIEN J., GAMOKE R., HANMER R., KEEVE F., NICODEMUS K. — Fault-Tolerant Telecommunication System Patterns, AT&T Bell Laboratories, 1995.

[AGOSTINI & AL., 1998] AGOSTINI A., DE MICHELIS G., JARKE M., MATTHES F., MYLOPOULOS J., POHL K., SCHMIDT J., WOO C., YU E. — A Three-Faceted View of Information Systems: The Challenge of Change, *Communications of the ACM*, p. 64-71, December (1998).

[AHMED & AL., 1991] AHMED R., SMEDT P., DU W., KENT W., KETABCHI M., LITWIN W., RAFII A., SHAN M. — The Pegasus heterogeneous multidatabase system, *IEEE Computer*, Vol. 24, N°12, p.19-27, (1991).

[ALBIN-AMIOT, 2003] ALBIN-AMIOT H. — Idiomes et patterns Java: Application à la synthèse de code et à la détection, Thèse de doctorat, Université de Nantes, Février (2003).

[ALEXANDER, 1979] ALEXANDER C. — *The timeless way of building,* Oxford University Press, New York, (1979).

[ALEXANDER & AL., 1977] ALEXANDER C., SHIKAWA S., SILVERSTEIN M., JACOBSON M., FIKSDAHL-KING I., ANGEL S. — *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, (1977).

[ALTER, 1999] ALTER S. — *Information Systems: a management perspective,* Addison-Wesley, Massachusetts, (1999).

[AMBLER, 1998] AMBLER S.W. — *Process Patterns: Building Large-Scale Systems Using Object Technology*, Cambridge University Press, (1998).

[ANDREOLI & PARESCHI, 1991] ANDREOLI J.-M., PARESCHI R. — Communication as Fair Distribution of Knowledge, *Proc. OOPSLA'91*, ACM Press ed., vol. 1, n°1, p. 212-229, (1991).

[APPLETON, 2000] APPLETON B. — Patterns and Software: Essential Concepts and Terminology, http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html, dernière modification le 14 février (2000).

[ARANGO, 1994] ARANGO G. — Domain analysis methods, in *Software Reusability*, W. Schaefer, R. Prieto-Diaz & M. Matsumoto Editors, Ellis Horwood, New York, (1994).

[ARENS & AL., 1993] ARENS Y., CHEE C.Y., HSU C., KNOBLOCK C.A. — Retrieving and integrating data from multiple information sources, *International Journal of Intelligent and Cooperative Information Systems*, Vol. 2, N°2, p. 127-157, (1993).

[ARIDOR & LANGE, 1998] ARIDOR Y., LANGE D.B. — Agent Design Patterns: Elements of Agent Application Design, *Proceedings of the Second International Conference on Autonomous Agents (Agents'98)*, P. Sycara & M. J. Wooldridge Editors, ACM Press, may (1998).

[BACKLUND, 2001] BACKLUND P. — The Use of Patterns in Information Systems Engineering, Masters Thesis, Department of Computer Science, Högskolan, Suède, (2001).

[BACKLUND, 2003] BACKLUND P. — Knowledge Transfer in Information Systems Engineering, Licentiate Thesis, Department of Computer and Systems Sciences, Stockholm University, Suède, (2003).

[BAÏNA & AL., 2003] BAÏNA K., TATA S., BENALI K. — A Model for Process Service Interaction, *Business Process Management, International Conference (BPM 2003)*, LNCS 2678, Eindhoven, The Netherlands, June 26-27 (2003).

[BARBIER & AL., 2002] BARBIER F., CAUVET C., OUSSALAH M., RIEU D., BENNASRI S., SOUVEYET C. — Composants dans l'Ingénierie des Systèmes d'Information : Concepts clés et techniques de réutilisation, *Actes Assises Nationales GDR I3, Information Interaction Intelligence*, Nancy, p. 95-117, 4-6 décembre (2002).

[BASILI, 1996] BASILI V. — Editorial, *Journal of Empirical Software Engineering*, Vol. 1, n° 2, (1996).

[BÄUMER & AL., 1997] BÄUMER D., GRYCZAN G., KNOLL R., LINIENTHAL C., RIEHLE D., ZÜLLIGHOVEN H. — Framework development for large systems, *Communications of the ACM*, 40(10), p. 53-59, October (1997).

[BAYARDO & AL., 1997] BAYARDO R., BOHRER W., BRICE R., CICHOCKI A., FOWLER G., HELAL A., KASHYAP V., KSIEZYK T., MARTIN G., NODINE M., RASHID M., RUSINKIEWICZ M.,

SHEA R., UNNIKRISHNAN C., UNRUH A., WOELK D. — InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, USA, p. 195-206, May (1997).

[BECK, 1997] BECK K. — *Best Practice Patterns, Volume 1: Coding*, Prentice Hall, NJ, 1997.

[BECK & CUNNINGHAM, 1987] BECK K., CUNNINGHAM W. — Using pattern languages for object-oriented programs, Technical Report CR-87-43, Tektronix Inc., September (1987).

[BECK & JOHNSON, 1994] BECK K., JOHNSON R. — Patterns Generate Architectures, In *Proceedings of the 8th European Conference on Object-Oriented Programming*, Bologna, Italy, p.139-149, (1994).

[BELLINZONA & AL., 1994] BELLINZONA R., CASTANO S., DE ANTONELLIS V., FUGINI M.G., PERNICI B. — Requirements Reuse in the F3 Project, *Ingénierie des Systèmes d'Information*, vol. 2, n°6, Hermès, p.699-717, (1994).

[BENSLIMANE & AL., 1998] BENSLIMANE D., YETONGNON K., CHRAIBI S., ABDELWAHED H. — DECA : une architecture multi-agents pour l'interopérabilité de bases de données, *Conférence Internationale CARI'98*, Dakar, Sénégal, 12-15 octobre (1998).

[BENSLIMANE & AL., 2000] BENSLIMANE D., YETONGNON K., CHRAIBI S., LECLERCQ E., ABDELWAHED E. — Les Systèmes d'information coopératifs : le projet DECA, *Technique et Science de l'Informatique (TSI)*, Edition Hermès, Vol. 19, N°7/2000, p. 1009-1043, (2000).

[BERGIN, 2002] BERGIN J. — Coding at the Lowest Level - Coding Patterns for Java Beginners, In *Proc. of EuroPLoP 2001*, Rüping A., Eckstein J. and Schwanninger C. (Ed.), Konstanz, Germany, p. 251-285, (2002).

[BISCHOFBERGER & POMBERGER, 1992] BISCHOFBERGER W., POMBERGER G. — Prototyping-Oriented Software Development, Springer-Verlag, (1992).

[BLAKLEY & HEATH, 2004] BLAKLEY B., HEATH C. — Security Design Patterns, Technical Guide G031, The Open Group, April (2004).

[BLUEPRINT TECHNOLOGIES, 2004] BLUEPRINT TECHNOLOGIES — Site web du logiciel Framework Studio 1.5, http://www.blueprinttech.com/html/Products_products.asp?Product=2, Blueprint Technologies, (2004).

[BOEHM, 1988] BOEHM B.W. — A spiral model for software development and enhancement, *IEEE Computer*, p. 61-72, May (1988).

[BOITET & SELIGMAN, 1994] BOITET C., SELIGMAN M. — The "Whiteboard" Architecture: A Way to Integrate heterogeneous components of NLP Systems, *Proc. COLING-94*, Makoto Nagao & ICCL ed., vol. 1/2, Kyoto, Japan, p. 426-430, August (1994).

[BOOCH & AL., 1999] BOOCH G., RUMBAUGH J., JACOBSON I. — *The Unified Modelling Language User Guide*, Addison-Wesley, Reading, MA, (1999).

[BORNE & REVAULT, 1999] BORNE I, REVAULT N. — Comparaison d'outils de mise en oeuvre de Design Patterns, *l'Objet*, vol. 5, n° 2, p. 243-266, (1999).

[BOSCH, 1997] BOSCH J. —Design Patterns & Frameworks: On the Issue of Language Support, *Workshop on Language Support for Design Patterns and Object-Oriented Frameworks (LSDF)*, ECOOP '97, p.133-136, June 09-13 (1997).

[BOUCELMA & LACROIX, 2001] BOUCELMA O., LACROIX Z. — InterMed : Interface de Médiation pour les Systèmes d'Information, *Ingénierie des Systèmes d'Information*, Volume 16, p. 33-60, Hermès, (2001).

[BOUGHZALA, 2002] BOUGHZALA I. — Methodology for designing interenterprise cooperative information system, *6th World Multi Conference on Systemics, Cybernetics and Informatics SCI'2002*, Orlando, USA, july 14-18 (2002).

[BOUGUETTAYA, 1994] BOUGUETTAYA A. — On the Scalability of Interoperable Autonomous and Heterogeneous Databases, *Proceedings of the International Symposium on Advanced Database Technologies and Their Integration (ADTI)*, Eds. S. Uemura and M. Yoshikawa, Nara, Japan, (1994).

[BOUGUETTAYA & AL., 1998] BOUGUETTAYA A., BENATALLAH B., ELMAGARMID A. K. — Multidatabase Systems: Past and Present, *In Distributed and Heterogeneous Database Systems*, Editors: A.K. Elmagarmid, M. Rusinkiewicz and A. Sheth, Morgan Kauffmann, (1998).

[BOUGUETTAYA & AL., 2000] BOUGUETTAYA A., BENATALLAH B., HENDRA L., OUZZANI M., BEARD J. — Supporting Dynamic Interactions among Web-Based Information Sources, *IEEE Transactions on knowledge and data engineering*, Vol. 12, N° 5, p. 779-801, september/november (2000).

[BOULANGER, 1991] BOULANGER D. — Coopération de systèmes d'information, Dossier scientifique pour l'habilitation à diriger des recherches, Université Jean Moulin, Lyon, 13 mars (1991).

[BOULANGER, 1995] BOULANGER D. — Coopération d'objets dans les systèmes d'information : problématique, *Congrès RPO, Représentation Par Objets*, Montpellier, France, 29 juin (1995).

[BOULANGER & DUBOIS, 1997] BOULANGER D., DUBOIS G. — Objets et coopération de systèmes d'information, *Ingénierie objet : concepts, techniques et méthodes*, Chap. 10, C. Oussalah & al, InterEditions, mai (1997).

[BOULANGER & DUBOIS, 1998] BOULANGER D., DUBOIS G. — An object approach for information system cooperation, *Information Systems*, vol. 23, n° 6, p. 383-399, (1998).

[BOULANGER & DUBOIS, 2001] BOULANGER D., DUBOIS G. — Advanced Object Approach for Information System Cooperation, dans Allen Kent & James Williams Executive Editeurs, *Encyclopedia of Microcomputers*, vol. 27, supplément n° 6, New-York, Marcel Dekker Inc., p. 15-35, (2001).

[BRADSHAW, 1997] BRADSHAW J.M. — An Introduction to Software Agents, In *Software Agents*, J.M. Bradshaw Editor, AAAI Press/The MIT Press, Menlo Park, California, (1997).

[BRAGA & AL., 2000] BRAGA A.M., RUBIRA C.M.F., DAHAB R. — Tropyc: A Pattern Language for Cryptographic Software, *Pattern Languages of Program Design 4 (PLoPD4)*, Harrison N.D., Foote B. and Rohnert H. eds., Addison-Wesley, p. 337-371, (2000).

[BREITBART, 1990] BREITBART Y. — Multidatabase Interoperability, *ACM SIGMOD Record*, Vol. 19, N°3, p. 53-60, (1990).

[BRIOT & DEMAZEAU, 2001] BRIOT J.-P., DEMAZEAU Y. — Introduction aux agents, In *Principes et architectures des systèmes multi-agents*, Jean-Pierre Briot et Yves Demazeau éditions, Collection IC2, Hermès Science Publications, Paris, France, décembre (2001).

[BRODIE & CERI, 1992] BRODIE M., CERI S. — On intelligent and cooperative information systems, *International Journal of Intelligent and Cooperative Information Systems,* 1(2), p. 1-35, September (1992).

[BROWN & WHITENACK, 1996] BROWN K., WHITENACK B. — Crossing Chasms: a pattern language for object-RDBMS integration, in *Pattern Languages of Program Design Vol. II*, Jim Coplien, Douglas Schmidt and Norman Kerth Editors, Addison-Wesley, (1996).

[BRUGALI & SYCARA, 1998] BRUGALI D., SYCARA K. — Agent technology: A new frontier for the development of application frameworks?, In M.E. Fayad, D.C. Schmidt and R.E. Johnson, editors, *Object Oriented Application Frameworks*, Wiley, (1998).

[BUDINSKY ET AL., 1996] BUDINSKY F.J., FINNIE M.A., VLISSIDES J.M., YU P.S. — Automatic code generation from design patterns, *IBM Systems Journal*, 35(2), p.151-171, (1996).

[BUSCHMANN & AL., 1996] BUSCHMANN F., MEUNIER R., ROHNERT H., SOMMERLAD P., STAL M. — *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, New York, (1996).

[BUTLER, 1995] BUTLER G. — Object-Oriented Application Frameworks, (1995).

[BUTLER & AL., 1995] BUTLER G., LI L., TJANDRA, I. — Reusable Object-Oriented Design, submitted, (1995).

[CALDERIA & BASILI, 1991] CALDERIA G., BASILI R.V. — Identifying and qualifying reusable software components, *IEEE computer*, February (1991).

[CANALS & AL., 1998] CANALS G., GODART C., MUNIER M., TATA S. — Supporting cooperation in project-entreprises, *International Conference on Enterprise Networking and Computing (ENCOM'98)*, Atlanta, Géorgie, juin (1998).

[CARDON, 2000] CARDON A. — L'informatique, science ou technologie, in *Bulletin de l'AFIA*, juillet (2000).

[CAREY & AL., 1995] CAREY M.J., HAAS L.M., SCHWARZ P.M., ARYA M., CODY W.F., FAGIN R., FLICKNER M., LUNIEWSKI A., NIBLACK W., PETKOVIC D., THOMAS J., WILLIAMS J.H., WIMMERS E.L. — Towards heterogeneous multimedia information systems: the Garlic

approach, *In RIDE-DOM '95, Fifth Int. Workshop on Research Issues in Data Engineering - Distributed Object Management*, Taipei, Taiwan, p. 124-131, (1995).

[CASTANO & DE ANTONELLIS, 1992] CASTANO S., DE ANTONELLIS V. — A model for reusable requirements, Polytechnico di Milano, F3 Project Internal Report, (1992).

[CATTEL & AL., 1993] CATTEL R., BARRY D., BARTELS D., EDS. — *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann, (1993).

[CAUVET, 1998] CAUVET C. — Ingénierie des systèmes d'information : de l'approche objet à l'approche composant, *L'Objet*, éditions Hermès, vol. 4, n°2, (1998).

[CAUVET & AL., 2001] CAUVET C., RIEU D., FRONT-CONTE A., RAMADOUR PH. — Réutilisation en ingénierie des systèmes d'information, in *Ingénierie des systèmes d'information*, Hermès, p. 115-148, (2001).

[CHAWATHE & AL., 1994] CHAWATHE S., GARCIA-MOLINA H., HAMMER J., IRELAND K., PAPAKONSTANTINOU Y., ULLMAN J., WIDOM J. — The TSIMMIS Project: Integration of Heterogeneous Information Sources, *Proceedings of the 16th Meeting of the Information Processing Society of Japan*, Tokyo, Japan, p. 7-18, October (1994).

[CHIKHI, 1998] CHIKHI Y. — Réutilisation de structures de données dans le domaine des réseaux électriques, Thèse de doctorat, Université Paris 6, (1998).

[CHIKHI & AL., 1999] CHIKHI Y., BOUNAAS F., MOUSTAFIADIES J. — Patterns pour la conception de réseaux électriques, L'objet, vol. 5, n°2, p. 177-202, (1999).

[CLEMENTS & AL., 1993] CLEMENTS D., GANESH M., HWANG S., LIM E., MEDIRATTA K., SRIVASTAVA J. — Myriad: Design and implementation of a federated database, Technical Report TR 93-76, Dept. of Computer Science, University of Minnesota, November (1993).

[CLEMENTS & NORTHROP, 2002] CLEMENTS P., NORTHROP L. — *Software Product Lines: Practices and Patterns*, Addison-Wesley, (2002).

[COAD, 1992] COAD P. — Object-oriented patterns, *Communications of the ACM*, vol. 35, n° 9, (1992).

[COAD, 1996] COAD P. — *Object Models – Strategies, Patterns and Applications*, Yourdon Press Computing Series, (1996).

[CONRAD & AL., 2002] CONRAD S., HASSELBRING W., JAMES A.E., KAMBUR D., KUTSCHE R., THIRAN PH. — Report on the EFIS 2001 Workshop, *Computer Journal*, 45(2), p. 249-251, (2002)

[CONTE & AL., 2001A] CONTE A., FREDJ M., GIRAUDIN J.P., RIEU D. — P-Sigma : un formalisme pour une représentation unifiée de patrons, *Congrès Inforsid*, France, (2001).

[CONTE & AL., 2001B] CONTE A., HASSINE I., GIRAUDIN J.P., RIEU D. — AGAP : un atelier de gestion et d'application de patrons, *Congrès Inforsid,* France, (2001).

[COOPIS'94, 1994] Second International Conference on Cooperative Information Systems (CooPIS-94), Toronto, Canada, May 17-20, (1994).

[COPLIEN, 1992] COPLIEN J.O. — *Advanced C++: Programming Styles and Idioms*, Addison-Wesley, (1992).

[COPLIEN, 1998] COPLIEN J.O. — *The Patterns Handbook: Techniques, Strategies, and Applications*, Cambridge University Press, New York, (1998).

[COUTURIER, 2002] COUTURIER V. — Des patterns et des composants pour la coopération de systèmes d'information : application à l'architecture coopérative ACSIS, In *Workshop Objets Composants et Modèles dans l'ingénierie des Systèmes d'Information (OCM-SI 2002)*, Nantes, 1er juin (2002).

[COUTURIER, 2004] COUTURIER V. — Patterns de coopération de systèmes d'information, In *INFORSID 2004*, Biarritz, p. 495-510, 25-28 mai (2004).

[COUTURIER & DUBOIS, 2004] COUTURIER V., DUBOIS G. — Une Approche orientée Patterns de la Coopération de Systèmes d'Information, *Technique et science informatiques*, n°10 (à paraître), (2004).

[COUTURIER & SEGURAN, 2003] COUTURIER V., SEGURAN M. — Patterns and Components to Capitalyze and Reuse a Cooperative Information System Architecture, *ICEIS'2003 International Conference on Enterprise Information Systems*, Angers, p. 225-231, 23-26 avril (2003).

[CULLOT & AL., 2003] CULLOT N., JOUANOT F., YETONGNON K. — Une méthode de réconciliation sémantique pour l'extraction de connaissances, *Revue des sciences et technologie de l'information (RSTI) - série RIA-ECA*, 17(1-2-3), p. 481-493, (2003).

[CZARNECKI & EISENECKER, 2000] CZARNECKI K., EISENECKER U. — *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, (2000).

[DARKE & AL., 1998] DARKE P., SHANKS G., BROADBENT M. — Sucessfully completing case study research: combining rigour, relevance and pragmatism, *Information Systems Journal*, vol. 8, n°4, p. 273-289, October (1998).

[DE MICHELIS & AL., 1997] DE MICHELIS G., DUBOIS E., JARKE M., MATTHES F., MYLOPOULOS J., PAPAZOGLOU M., POHL K., SCHMIDT J., WOO C., YU E. — Cooperative information systems: A manifesto, *Cooperative Information System: Trends and Directions*, (1997).

[DENECKERE & SOUVEYET, 2001] R. DENECKERE, C. SOUVEYET — Organising and selecting patterns in pattern languages with Process Maps, *7th International Conference on Object-Oriented Information Systems (OOIS'01)*, August 27-29 (2001).

[DEUGO & AL., 1999A] DEUGO D., OPPACHER F., KUESTER J., VON OTTE I. — Patterns as a Means for Intelligent Software Engineering, in *Proceedings of The International Conference on Artificial Intelligence (IC-AI'99)*, CSREA Press, p. 605–611, (1999).

[DEUGO & AL., 1999B] DEUGO D., KENDALL E.A., WEIß M. — Agent Patterns, http://www.scs.carleton.ca/~deugo/Patterns/Agent/Presentations/AgentPatterns/index.htm, 21 novembre (1999).

[DISSON 2001] DISSON E. — Sécurité des accès dans les systèmes d'information coopératifs, Thèse de doctorat, Université Jean Moulin, Lyon, (2001).

[DOGAC & AL., 1995A] DOGAC A. & AL. — METU Interoperable Database System, Technical Report 6-1, Juin (1995).

[DOGAC & AL., 1995B] DOGAC A. & AL. — Experiences in using CORBA for a multidatabase implementation, *6th Conference on Database and Expert Systems Applications Workshop*, Londres, Septembre (1995).

[DUBOIS, 1997] DUBOIS G. — Apport de l'intelligence artificielle à la coopération de systèmes d'information automatisés, Thèse de doctorat, Université Jean Moulin, Lyon, (1997).

[DUBOIS & AL., 2002] DUBOIS G., BOULANGER D., COUTURIER V. — Coopération d'objets pour la résolution de requêtes multibases, *In INFORSID 2002*, Nantes, p. 199-214, 1er juin – 4 juin (2002).

[ERMAN & LESSER, 1980] ERMAN L. D., LESSER V. R. — The Hearsay-II Speech Understanding System: A Tutorial, In *Trends in Speech Recognition*, W. A. Lea ed., Prentice Hall, p. 361-381, 1980.

[FANKHAUSER & AL., 1998] FANKHAUSER P., GARDARIN G., LOPEZ M., MUÑOZ J., TOMASIC A. — Experiences in Federated Data-bases From IRO-DB to MIRO-Web, In *24th Very Large Database International Conference (VLDB'98)*, New-York, États-Unis, p. 655-658, 24-27 août (1998).

[FAYAD & SCHMIDT, 1997A] FAYAD M., SCHMIDT D.C. — Object-oriented application frameworks, *Communications of the ACM*, 40(10), p. 32-38, October (1997).

[FAYAD & SCHMIDT, 1997B] FAYAD M., SCHMIDT D.C. — Lessons learned building reusable OO frameworks for distributed software, *Communications of the ACM*, 40(10), p. 85-87, October (1997).

[FENSEL & AL., 1998] FENSEL D., DECKER S., ERDMANN M., STUDER R. — Ontobroker: or how to enable intelligent access to the WWW., *Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'98)*, Banff, Canada, (1998).

[FERBER, 1995] FERBER J. — *Les systèmes multi-agents : vers une intelligence collective*, InterEditions, (1995).

[FERNANDEZ & AL., 2003] FERNANDEZ G., ZHAO L., WIJIEGUNARATNE I. — Design Patterns for a Federation Architecture, *Journal of Object Technology*, vol. 2, n° 3, p. 135-149, May-June (2003).

[FERNANDEZ & PAN, 2001] FERNANDEZ E., PAN R. — A pattern language for security models, In *Conference on Pattern Languages of Programs (PloP 2001),* Allerton Park, Monticello, Illinois, USA, September 11-15 (2001).

[FERNANDEZ & WIJEGUNARATNE, 1996] FERNANDEZ G., WIJEGUNARATNE I. — A Cooperative Approach to Distributed Applications Engineering, *Proc. Asian'96 WORKSHOP,*

Coordination Technology for Collaborative Applications: Organisations, Processes and Agents, p. 39-48, 5 décembre (1996).

[FERNANDEZ & ZHAO, 2000] FERNANDEZ G., ZHAO L. — A Pattern Language for Federated Architecture, *Proceedings of KoalaPLoP 2000*, Melbourne, Australie, p.21-31, 24-26 mai (2000).

[FERRAIOLO & KUHN, 1992] FERRAIOLO D., KUHN R. — Role-based access control, *15th National Computer Security Conference*, Baltimore, October (1992).

[FLORESCU & AL., 1996] FLORESCU D., RASHID L., VALDURIEZ P. — A Methodology for Query Reformulation in CIS Using Semantic Knowledge*, IJCIS*, Vol 5, N°4, p. 431-468, (1996).

[FOWLER, 1997] FOWLER M. — *Analysis Patterns*, Addison Wesley, (1997).

[FOWLER & AL., 1999] FOWLER J., PERRY B., NODINE M., BARGMEYER B. — Agent-Based Semantic Interoperability in InfoSleuth**,** *In SIGMOD Record*, Vol. 28, N° 1, p. 60-67, March (1999).

[FRICKE & VÖLTER, 2001] FRICKE A., VÖLTER M. — Seminars - A Pedagogical Pattern Language about teaching seminars effectively, in *Proc. of EuroPLoP 2000*, Devos M. and Rüping A. (Ed.), Konstanz, Germany, p. 87-128, (2001).

[FRONT, 1997] FRONT A. — Développement de systèmes d'information à l'aide de patrons – Applications aux bases de données actives, Thèse de Doctorat, Université Grenoble 1, décembre (1997).

[FRONT-CONTE & AL., 1999] FRONT-COMTE A., GIRAUDIN J.P., RIEU D., SAINT-MARCEL C. — Réutilisation et patrons d'ingénierie, *Génie objet : analyse et conception de l'évolution*, Chap. 3, Editeur M. Oussalah, Hermès, (1999).

[GAMMA & AL., 1995] GAMMA E., JOHNSON R., HELM R., VLISSIDES J. — *Design patterns, elements of reusable object-oriented software*, Addison-Wesley, (1995).

[GANGULY & AL., 2000] GANGULY P., MAZZI C., RAY P. — Software Agent Based Personal Assistant for Tele-Homecare in Diabetes Treatment, *Proceedings of MAMA 2000*, Wollongong, Australie, 11-15 décembre (2000).

[GANGULY & AL., 2001] GANGULY P., RABHI F.A., RAY P.K. — The Semantic Interoperability Pattern, *Proceedings of The Second Asian-Pacific Conference of Pattern Languages of Program Design (KoalaPLoP 2001)*, James Noble and Brian Wallis Editions, Nouvelle-Zélande, (2001).

[GARCIA-MOLINA & AL., 1997] GARCIA-MOLINA H., PAPAKONSTANTINOU Y., QUASS D., RAJARAMAN A., SAGIV Y., ULLMAN J., VASSALOS V., WIDOM J. — The TSIMMIS approach to mediation: Data models and Languages, *In Journal of Intelligent Information Systems*, vol. 8, p. 117-132, (1997).

[GARDARIN & AL., 1995] GARDARIN G., GANNOUNI S., FINANCE B., FANKHAUSER P., KLAS W., PASTRE D., LEGO R. — IRO-DB: Making relational and object-oriented database

systems interoperable, *In Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*, Prentice Hall, p. 449-468, (1995).

[GENESERETH, 1997] GENESERETH M.R. — An Agent-based Framework for Interoperability, *In Software Agents*, J.M. Bradshaw (editor), AAAI Press/The MIT Press, Menlo Park, California, (1997).

[GENESERETH & AL., 1997] GENESERETH M.R., KELLER A.M., DUSCHKA O.M. — Infomaster : an information integration system, *In Proceedings of the ACM SIGMOD Conference*, p. 539-542, May (1997).

[GEYER-SCHULZ & HAHSLER, 2001] GEYER-SCHULZ A., HAHSLER M. — Software engineering with analysis patterns, Technical Report 01/2001, Institut für Informationsverarbeitung und -wirtschaft, Wirschaftsuniversität Wien, Wien, November (2001).

[GOH, 1997] GOH C. — Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems, PhD thesis, MIT Sloan School of Management, USA, (1997).

[GOH & AL., 1994] GOH C., MADNICK S., SIEGEL M. — Context Interchange: Overcoming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment, in *Proceedings of the Third International Conference on Information and Knowledge Management*, ACM Press, Gaithersburg, Maryland, USA, p. 337-346, December (1994).

[GOLDBERG, 1990] GOLDBERG A. — Reusing Software Development, *Publications of ACM*, p. 107-119, (1990).

[GRAY & AL., 1997] GRAY P.M.D., PREECE A., FIDDIAN N.J., GRAY W.A., BENCH-CAPON T.J.M., SHAVE M.J.R., AZARMI N., WIEGAND M., ASHWELL M., BEER M., CUI Z., DIAZ B., EMBURY S.M., HUI K., JONES A.C., JONES D.M., KEMP G.J.L., LAWSON E.W., LUNN K., MARTI P., SHAO J., VISSER P.R.S. — KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases, *Database and Expert System Applications (DEXA' 97)*, Toulouse, France, (1997).

[GROSS & AL., 1994] GROSS D., STUCKEY L., JONES G., LANG R. — An Instance of the Air Vehicle Training Systems Domain: A STARS Demonstration Project, *Proceedings of the 1994 Military, Government and Aerospace Simulation Multiconference*, Simulation Councils, (1994).

[GZARA, 2000] GZARA L. — Les patterns pour l'ingénierie des systèmes d'information produit, Thèse de doctorat, Institut National Polytechnique de Grenoble, 12 décembre (2000).

[HASSELBRING, 1999] HASSELBRING W. — Design of a Communication Framework for Interoperable Information Systems, In *Transactions of the SDPS: Journal of Integrated Design and Process Science*, 3(3), p. 15-27, September (1999).

[HASSELBRING & AL., 2000] HASSELBRING W., VAN DEN HEUVEL J., HOUBEN G., KUTSCHE R., RIEGER B., ROANTREE M., SUBIETA K. — Research and Practice in Federated Information Systems, Report of the EFIS '2000 International Workshop, *ACM SIGMOD Record*, 29(4), p.16-18, December (2000).

[HENDERSON-SELLERS & EDWARDS, 1990] HENDERSON-SELLERS B., EDWARDS J. M. — The Object-oriented Systems Life-Cycle, *Communications of the ACM*, 09, (1990).

[HERIN & AL., 2001] HERIN D., ESPINASSE B., ANDONOFF E., HANNACHI C. — Des systèmes d'informations coopératifs aux agents informationnels, chapitre 8 du livre *ingénierie des systèmes d'information*, Hermès, (2001).

[HICKS, 1993] HICKS J. — *Management Information Systems: A user perspective*, Wes Publishing Company, St. Paul, (1993).

[HOHPE & WOOLF, 2004] HOHPE G., WOOLF B. — Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions, Addison Wesley, USA, 2004.

[HUNG & PASQUALE, 1999] HUNG E., PASQUALE J. — Agent Usage Patterns: Bridging the Gap Between Agent-Based Application and Middleware, rapport technique CS1999-0638, Department of Computer Science and Engineering, University of California, San Diego, Californie, 17 novembre (1999).

[IIVARI, 2000] IIVARI J. — Information Systems Development as Knowledge Work: The body of systems development process knowledge, In *Information Modelling and Knowledge Bases XI,* Eds Kawaguchi E., Hamid I. A., Jaakkola H. et Kangassalo H., IOS Press, p. 41-56, (2000).

[JOHN & DÖRR, 2003] JOHN I., DÖRR J. — Elicitation of Requirements from User Documentation, *Ninth International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ '03*, Klagenfurt/Velden, Austria, 16 -17 June (2003).

[JONES, 1990] JONES C. — *Systematic Software development using VDM*, Prentice Hall, New-York, (1990).

[JOUANOT, 2000] JOUANOT F. — Un modèle sémantique pour l'interopérabilité de systèmes d'information, *In INFORSID 2000*, Lyon, p. 347-364, 16 – 19 mai (2000).

[JOUANOT, 2001] JOUANOT F. — DILEMMA : vers une cooperation de systèmes d'informations basée sur la médiation sémantique et la fusion d'objets, Thèse de doctorat, Université de Bourgogne, France, novembre (2001).

[KANG & AL., 1990] KANG K., COHEN S., HESS J., NOVAK W., PETERSON S. — Feature-Oriented Domain Analysis (FODA), Feasability study CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, (1990).

[KELLER & COLDEWEY, 1998] KELLER W., COLDEWEY J. — Accessing Relational Databases: A Pattern Language, in *Pattern Languages of Program Design 3,* Robert Martin, Dirk Riehle, Frank Buschmann (Eds.), Addison-Wesley, (1998).

[KENDALL & AL., 1996] KENDALL E.A., MALKOUM M.T., JIANG C.H. — The Layered Agent Pattern Language, in *3rd Conference on Pattern Languages of Programming (Plop'96)*, Monticello, Illinois, USA, septembre (1996).

[KIM & AL., 1994] KIM W., CHOI I., GALA S., SCHEEVEL M. — On resolving schematic heterogeneity in multidatabase systems, *In Modern database systems : the object model, interoperability, and beyond*, Addison-Wesley, p. 649-663, (1994).

[KIM & SEO, 1991] KIM W., SEO J. — Classifying Schematic and Data Heterogeneity in Multidatabase Systems, *Journal of the ACM*, vol. 24, p.12-18, (1991).

[KOZACZYNSKI & BOOCH, 1998] KOZACZYNSKI W., BOOCH G — Component Based Software Engineering, *IEEE Software*, Vol. 15, N° 5, septembre / octobre (1998).

[KRÄMER & PRECHELT,1996] KRÄMER C., PRECHELT L. — Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software, *Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96)*, November 08-10 (1996).

[KUHN & AL., 1988] KUHN E., LUDWIG T. — VIP-MDBMS: A logic multidatabase system, *In Proc. of the International Symposium on Databases in Parallel and Distributed Systems*, (1988).

[LAM & MCDERMID, 1997] LAM W., MCDERMID J. A. — A summary of domain analysis experience by way of heuristics, *Software Engineering Notes*, 22(3), p. 54-64, (1997).

[LANGR, 1999] LANGR J. — *Essential Java Style: Patterns for Implementation,* Prentice Hall PTR, (1999).

[LARMAN, 2002] LARMAN C. — *UML et les Design Patterns*, Campus Press, 2002.

[LAROUSSE, 2004] LE PETIT LAROUSSE 2004 — Dictionnaire de la langue française, Ed. Larousse, Paris, (2004).

[LAVENDER & SCHMIDT, 1995] LAVENDER R., SCHMIDT D.C. — Active Object - An Object Behavioural Pattern for Concurrent Programming, *Proceedings of the Second Pattern Languages of Programs Conference*, Monticello, Illinois, September 6-8 (1995)

[LEA, 1999] LEA D. — *Concurrent Programming in Java: Design Principles and Patterns*, Second Edition, Addison Wesley, November (1999).

[LEVY & AL., 1996] LEVY A., RAJARAMAN A., ORDILLE J.J. — Querying Heterogeneous Information Sources Using Source Descriptions, *In Proceedings of Twenty-second International Conference on Very Large Databases*, Bombay, India, p. 251-262, September (1996).

[LIBOUREL & AL., 2003] LIBOUREL TH. & AL. — Médiation via les méta-données, Synthèse de l'Action spécifique 97 du département STIC du CNRS, RTP 9, sous l'animation de Th. Libourel, http://www.lirmm.fr/%7Elibourel/MM/RA-AS.doc, (2003).

[LIND, 2002] LIND J. — Patterns in Agent-Oriented Software Engineering, *Third International Workshop on Agent-Oriented Software Engineering held at Autonomous Agents & Multi-Agent Systems (AAMAS 2002)*, Faculty of Engineering, Bologna, Italy, July 15, (2002).

[LITWIN, 1985] LITWIN W. — Implicit joins in the multidatabase system MRDSM, In *Ninth International Computer Software & Applications Conference (IEEE-COMPSAC'85)*, Chicago, USA, October 9-11 (1985).

[LITWIN, 1988] LITWIN W. — From Database Systems to Multidatabase Systems: Why and How, *in Proc. of the British National Conference on Databases (BNCOD'1988)*, Cardiff, UK, p. 161-188, July (1988).

[LITWIN & AL., 1982] LITWIN W., BOUDENANT J., ESCULIER C., FERRIER A., GLORIEUX A.M., LA CHIMIA J., KABBAJ K., MOULINOUX C., ROLIN P., STANGRET C. — SIRIUS System for Distributed Data Management, *in Distributed Database*, p. 311-366, (1982).

[LITWIN & AL., 1990] LITWIN W., MARK L., ROUSSOPOULOS N. — Interoperability of multiple autonomous databases, *ACM Computing Surveys*, Vol. 22, N°3, p. 267-293, (1990).

[LIU & PU, 1995] LIU L., PU C. — The Diom approach to large-scale interoperable information systems, Technical report, TR95-16, Department of Computing Science, University of Alberta, Edmonton, Alberta, March (1995).

[LOPISTEGUY & AL., 2003] LOPISTEGUY PH., ETCHEVERRY P., DAGORRET P. — Etude de la coordination dans un processus, In *INFORSID 2003*, Nancy, p. 169-184, 3-6 juin (2003).

[LUDASCHER & AL., 1999] LUDASCHER B., PAPAKONSTANTINOU Y., VELIKHOV P. — A Framework for Navigation-Driven Lazy Mediators, *Proceedings of WebDB'99*, p. 85-90, Juin (1999).

[LYNGBAEK & MCLEOD, 1983] LYNGBAEK P., MCLEOD D. — An Approach to Object Sharing in Distributed Database Systems, *Proceedings of the 9th International Conference on Very Large Data Bases*, p. 364-375, October 31-November 02 (1983).

[LYYTINEN & ROBEY, 1999] LYYTINEN K., ROBEY D. — Learning failure in information systems development, *Information Systems Journal,* 9, p. 85-101, (1999).

[MAIDEN, 1991] MAIDEN N. — Analogy as a paradigm for specification reuse, *Software engineering journal,* 6(1), p. 3-15, january (1991).

[MAIDEN & AL., 1994] MAIDEN N., SUTCLIFFE A., TAYLOR C., TILL D. — A set of formal problem abstractions for reuse during requirements engineering, dans *Ingénierie des systèmes d'information*, vol. 2, n°6, (1994).

[MAIDEN & SUTCLIFFE, 1993A] MAIDEN N., SUTCLIFFE A. — The Domain Theory: Object System Definition, Nature Report CU-93-OOE, (1993).

[MAIDEN & SUTCLIFFE, 1993B] MAIDEN N., SUTCLIFFE A. — The Domain Theory: Domain Specialisation Hierarchy, Nature Report CU-93-OOE, (1993).

[MCILROY, 1976] MCILROY M.D. — Mass-produced software components, *Software Engineering Concepts and Techniques* (1968 NATO Conference on Software Engineering), in J.M Buxton, P. Naur, and B. Randell editors, p. 88–98, (1976).

[MEIJERS, 1996] MEIJERS M. — Tool Support for Object-Oriented Design Patterns, Master's Thesis, INF-SCR-96-28, CS Dept, Utrecht University, August (1996).

[MEIRA & AL., 2001] MEIRA N., SILVA I.C., SILVA, A. — An Agent Pattern for a More Expressive Approach, in *Proc. of EuroPLoP 2000,* Devos M. and Rüping A. (Ed.), Konstanz, Germany, p. 331-346, (2001).

[MENA, 1999] MENA E. — OBSERVER: An approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies, PHD Thesis, University of Zaragoza, Spain, February (1999).

[MESROBIAN & AL., 1996] MESROBIAN E., MUNTZ R., SHEK E., NITTEL S., LA ROUCHE M., KRIGUER M. — OASIS: An Open Architecture Scientific Information System, *In Proc. of 6th International Workshop on Research Issues in Data Engineering: Interoperability of Non Traditional Database Systems (RIDE-96)*, New Orleans, Louisiana, p. 107-116, February (1996).

[MILI & AL., 1995] MILI H., MILI F., MILI A. — Reusing Software: Issues and Research Directions, *IEEE Transactions on Software Engineering*, vol. 21, n°6, June (1995).

[MOTELET, 2000] MOTELET O. — A contextual help system for assisting OO software designers in using design patterns, Master's thesis, Vrije Universiteit Brussel, (2000).

[MULLER, 1997] MULLER J.P. — The Design of Intelligent Agents: A Layered Approach, *Lecture Notes in Computer Science*, vol. 1177, Springer Verlag, (1997).

[MULLER & GAERTNER, 2000] MULLER P.A., GAERTNER N. — *Modélisation objet avec UML*, 2ème édition, Eyrolles, (2000).

[MYERS, 1997] MYERS M. — Interpretative Research in Information Systems, *In Information Systems: An Emerging Discipline?,* Eds Mingers J. et Stowell F., McGraw-Hill, London, p. 239-266, (1997).

[NANCI & AL., 2001] NANCI D., ESPINASSE B., COHEN B., ASSELBORN J.C., HECKENROTH H. — Ingénierie des systèmes d'information : Merise deuxième génération, 4ème édition, Vuibert, Paris, (2001).

[NEIGHBORS, 1984] NEIGHBORS J.M — The DRACCO approach to constructing software from reusable components, *IEEE transactions on software engineering*, 10(5), p.564-574, (1984).

[NELLBORN, 1999] NELLBORN C. — Business and Systems Development: Opportunities for an Integrated Way-of-Working, In Nilsson A.G., Tolis C., Nellborn C. (eds.), *Perspectives on Domain modeling: understanding and Changing Organisations*, Springer Verlag, Berlin, (1999).

[NILSON & AL., 1994] NILSON R., KOGUT P., JACKELEN G. — Component Provider's and Tool Developer's Handbook Central Archive for Reusable Defense Software (CARDS), STARS Informal Technical Report STARS-VC-B017/001/00, Unisys Corporation, March (1994).

[NODINE & AL., 2000] NODINE M., FOWLER J., KSIEZYK T., PERRY B., TAYLOR M., UNRUH A. — Active Information Gathering in InfoSleuth, *In International Journal of Cooperative Information Systems*, 9:1/2, p. 3-28, (2000).

[NWANA, 1996] NWANA H.S. — Software Agents: An Overview, *Knowledge Engineering Review*, Vol.11, N°3, p. 205-244, (1996).

[OMG, 2002] OMG — CORBA 3.0.3 Specification - IDL Syntax and Semantics, document de spécification, OMG, 7 juin (2002).

[OMG, 2003] OMG — MDA Guide version 1.0, Miller J. et Mukerji J. éditeurs, OMG, 1er mai (2003).

[OUKSEL & AHMED, 1999] OUKSEL A.M., AHMED I. — Ontologies are not the Panacea in Data Integration: A Flexible Coordinator to Mediate Context Construction, *Distributed and Parallel Databases*, 7(1), p. 7-35, (1999).

[OUZZANI & BOUGUETTAYA, 2004] OUZZANI M., BOUGUETTAYA B. — Query Processing and Optimization on the Web, *Distributed and Parallel Databases Journal*, 15(3), Kluwer Academic Publishers, p. 187-218, May (2004).

[OZCAN & AL., 1996] OZCAN F. & AL. — Dynamic query optimisation on a distributed object management platform, *In Proc. of Fifth International Conference on Information and Knowledge Management*, Maryland, USA, Novembre (1996).

[PANG, 1999] PANG C.H. — A Pattern of Inheritance and Polymorphism for Persistent Objects Stored in a Relational Database, *Journal of Object-Oriented Programming*, Vol. 11, N° 9, p. 41-44, February (1999).

[PAPAZOGLOU & AL., 1992] PAPAZOGLOU M.P., LAUFMANN S., SELLIS T.K. — An organizational framework for cooperating intelligent information systems, *International Journal of Cooperative Information Systems*, Vol. 1, N°1, March (1992).

[PERCIVAL, 1999] PERCIVAL R. — Standing on the Shoulders of Giants: The Reverse Engineering of Computer Software and the Law of Copyright In Canada, Rapport de l'Université du Maryland, USA, Octobre (1999).

[PETERSON & STANLEY, 1994] PETERSON A.S., STANLEY J.L. — Mapping a Domain Model and Architecture to a Generic Design, Technical Report CMU/SEI-94-TR-008, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, (1994).

[PLATT, 2000] PLATT D. — *The essence of COM, a programmer's workbook*, 3ème edition, Prentice Hall, (2000).

[PORT, 1998] PORT D. — Derivation of Domain Specific Design Patterns, USC Center for software engineering, (1998).

[PORTLAND, 1994] PORTLAND — About the Portland Form, http://c2.com/ppr/about/portland.html, (1994).

[POSNAK & AL., 1997] POSNAK E.J., LAVENDER R.G., VIN H.M. — An adaptive framework for developing multimedia software components, *Communications of the ACM*, 40(10), p. 43–47, octobre (1997).

[PRAEHOFER & AL., 2000] PRAEHOFER H., SAMETINGER J., STRITZINGER A — Building Reusable Simulation Components, dans *Proceedings of WEBSIM2000, Web-Based Modelling & simulation*, San Diego, USA, (2000).

[PREE, 1995] PREE W. — Design Patterns for Object-Oriented Software Development, Addison-Wesley Publishing, ACM Press, (1995).

[PREECE & AL., 1999] PREECE A.D., HUI K.-Y., GRAY P.M.D. — Kraft: Supporting Virtual Organisations through Knowledge Fusion, In T. Finin and B. Grosof (editors), *Artificial Intelligence for Electronic Commerce: Papers from the AAAI-99 Workshop*, AAAI Press, p. 33-38, (1999).

[PRESSMAN, 1997] PRESSMAN R.S. — *Software Engineering A Practitioner's Approach*, McGraw-Hill, New York, (1997).

[PRIETO-DIAZ, 1990] PRIETO-DIAZ R. — Domain analysis: an introduction, *ACM SIGSOFT Software Engineering Notes*, vol.15, n°2, p. 47-54, April (1990).

[PUPPE, 2000] PUPPE F. — Knowledge Formalization Patterns, *Proceedings of PKAW 2000*, Sydney Australia, (2000).

[RADA & MOORE, 1997] RADA R., MOORE J. — Standardizing Reuse, *Communications of the ACM*, Vol. 40, N° 3 , p. 19-23, March (1997).

[REEL, 1999] REEL J.S. — Critical Success Factors In Software Projects, *IEEE Software*, p. 18-23, May/June (1999).

[REICH, 1999] REICH J.R. — Ontological Design Patterns for the Integration of Molecular Biological Information, *Proceedings of the German Conference on Bioinformatics GCB'99*, Hannover, Germany, p. 156-166, 4-6 October (1999).

[REKOFF, 1985] REKOFF M.G. — On Reverse Engineering, in *IEEE Trans. on Systems, Man, and Cybernetics*, March-April (1985).

[REVAULT, 1996] REVAULT N. — Principes de méta-modélisation pour l'utilisation de canevas d'applications à objets (MétaGen et les frameworks), Thèse de Doctorat, Université Paris 6, novembre 1996.

[RIDEAU, 1998] RIDEAU S. — Les patterns métiers bancaires : un concept pour réutiliser le composant et le savoir-faire, *Congrès Inforsid*, France, (1998).

[RIDEAU, 1999] RIDEAU S. — Réutilisation objet : une approche technique et organisationnelle dans le monde bancaire, Thèse de doctorat, Université de Nantes, (1999).

[RIEHLE & ZÜLLINGHOVEN, 1996] RIEHLE D., ZÜLLINGHOVEN H. — Understanding and Using Patterns in Software Development, in *Theory and Practice of Object Systems*, vol. 2, n°1, p. 3-13, (1996).

[RIEU, 1999] RIEU D. — Ingénierie des Systèmes d'Information, Mémoire d'Habilitation à Diriger les Recherches, Institut National Polytechnique de Grenoble, (1999).

[RIEU & AL., 1999] RIEU D., GIRAUDIN J.P., SAINT-MARCEL C., FRONT-CONTE A. — Des Opérations et des Relations pour les Patrons de Conception, *XVIIe Congrès INFORSID*, Toulon, juin (1999).

[RILEY, 1991] RILEY G. — CLIPS: An Expert System Building Tool, *Proceedings of the Technology 2001 Conference*, San Jose, CA, December (1991).

[RISING, 2000] RISING L. — *The Pattern Almanac 2000*, Software Pattern Series, Addison-Wesley, (2000).

[ROBBINS, 1998] ROBBINS J. — Overview of the Graph Editing Framework (GEF), Arcadia project, University of California, Irvine, (1998).

[ROLLAND & AL., 1988] ROLLAND C., FOUCAULT O., BENCI G. — *Conception des systèmes d'information, la méthode REMORA,* Eyrolles, (1988).

[ROUSSET & AL., 2002] ROUSSET M.C., BIDAULT A., FROIDEVAUX C., GAGLIARDI H., GOASDOUE F., REYNAUD C., SAFAR B. — Construction de médiateurs pour intégrer des sources d'information multiples et hétérogènes : le projet PICSEL, *Revue I3 Information-Interaction-Intelligence*, (2002).

[ROYCE, 1970] ROYCE W. — Managing the development of large software systems, *in Proc. IEEE Western Computer Conf.*, Los Alamitos, CA, August (1970).

[RUSINKIEWICZ & AL., 1988] RUSINKIEWICZ M., ELMASRI R., CZEJDO B., GEORGAKOPOULOS D., KARABATIS G., JAMOUSSI A., LOA K., LI Y. — Omnibase: Design and Implementation of a Multidatabase System, *Distributed Technical Commitee Newsletter*, Vol.10, N°2, November (1988).

[SAIDANE & GIRAUDIN, 2002] SAIDANE M., GIRAUDIN J-P. — Des patrons pour l'ingénierie de la coopération des systèmes d'information, *Ingénierie des Systèmes d'Information*, numéro spécial "Connaissances métier dans l'Ingénierie des Systèmes d'Information", volume 7, numéro 4, Hermès, (2002).

[SAVONNET & AL., 2002] SAVONNET M., LECLERCQ E., TERRASSE M.N. — Patterns d'architecture d'ontologies, *Actes du 20ème Congrès Informatique des Organisations et Systèmes d'Information et de Décision Inforsid'2002*, Nantes, p. 291-305, 4-7 juin (2002).

[SCHELFTHOUT & AL., 2002] SCHELFTHOUT K., CONINX T., HELLEBOOGH A., HOLVOET T., STEEGMANS E., WEYNS D. — Agent Implementation Patterns, *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, Debenham J., Henderson-Sellers B., Jennings N. and Odell J. eds., p. 119-130, (2002).

[SCHMIDT, 1995] SCHMIDT D. C. — Using Design Patterns to Develop Reusable Object-Oriented Communication Sofware, *Communications of the ACM*, Vol. 38, N° 10, (1995).

[SCHMIDT, 1998] SCHMIDT D. — An Architectural Overview of the ACE Framework, USENIX Login Magazine, edited by Peter Salus, (1998).

[SCHMIDT & AL., 1996] SCHMIDT D. C., JOHNSON R. E., FAYAD M. — Software Patterns, in *Communications of the ACM*, Special Issue on Patterns and Pattern Languages, Vol. 39, N° 10, October (1996).

[SCHMIDT & AL., 2001] SCHMIDT D. C., STAL M., ROHNERT H., BUSCHMANN F. — *Pattern-Oriented Software Architecture - Patterns for Concurrent and Networked Objects*, UK, John Wiley & Sons, (2001).

[SCHUBERT & AL., 1998] SCHUBERT C., BRENNER W., ZARNEKOW R. — A methodology for classifying intelligent software agents, *Proceedings of the 6th European Conference on Information Systems*, Aix-En-Provence, France, p. 304-316, (1998).

[SCHUMACHER & ROEDIG, 2001] SCHUMACHER M., ROEDIG U. — Security Engineering with Patterns, In *Proceedings of the 8th Conference on Pattern Languages of Programs (PLoP 2001)*, September (2001).

[SEAMAN, 1999] SEAMAN C.B. — Qualitative Methods in Empirical Studies of Software Engineering, IEEE Transactions of Software Engineering, 25(4), p. 557-572, (1999).

[SEGURAN, 2003] SEGURAN M. — Résolution des conflits sémantiques dans les systèmes d'information coopératifs, Thèse de doctorat, Université Jean Moulin, Lyon, (2003).

[SEGURAN & COUTURIER, 2003] SEGURAN M., COUTURIER V. — A Reusable Architecture for the Cooperation of Information Systems, *In 7th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida, USA, July 27-30th (2003).

[SEMMAK, 1998] SEMMAK F. — *Réutilisation de composants de domaine dans la conception des systèmes d'information,* thèse de doctorat, Université Paris I, (1998).

[SHAW & GARLAN, 1996] SHAW M., GARLAN D. — *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, (1996).

[SHETH, 1999] A. SHETH — Changing Focus on Interoperability in Information Systems: from System, Syntax, Structure to Semantics, In *Interoperating Geographic Information Systems*, M. F. Goodchild, M.J. Egenhofer, R. Fegeas and C.A. Koffman Editors, Kluwer Academic Publishers, (1999).

[SHETH & KASHYAP, 1992] SHETH A., KASHYAP V. — So Far (Schematically) Yet So Near (Semantically), *in proceedings of the IFIP DS-5, Conference on Semantics of Interoperable Database Systems*, Lorne, Australia, November (1992).

[SHETH & LARSON, 1990] SHETH A., LARSON J. — Federated database systems for managing distributed, heterogeneous, and autonomous database, *ACM computing Survey*, Vol. 22, N°3, p. 183-190, (1990).

[SILVA & DELGADO, 1998] SILVA A., DELGADO J. — The Agent Pattern for Mobile Agent Systems, *3rd European Conference on Pattern Languages of Programming and Computing (Europlop'98)*, Irsee, Allemagne, (1998).

[SIMEON, 1999] SIMEON J. — Intégration de sources de données hétérogènes – ou comment marier simplicité et efficacité, Thèse de doctorat, Université Paris XI, 18 janvier (1999).

[SMITH, 1980] SMITH R.G. — The Contract Net Protocol: High level Communication and Control in a Distributed Problem Solver, *IEEE transactions on Computers*, Vol. 29, (1980).

[SNOECK & POELS, 2000] SNOECK M., POELS G. — Analogical Reuse of Structural and Behavioural Aspects of Event-Based Object-Oriented Domain Models, *Domain Engineering Workshop, Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, London, p. 802-806, 4-8 Sept. (2000).

[SPIVEY, 1989] SPIVEY J. — *The Z Notation: A Reference Manual*, Prentice Hall, New York, (1989).

[SPIVEY, 1992] SPIVEY J. — *The Z Notation: A Reference Manual,* Prentice Hall, New York, 2nd edition, (1992).

[STAAB & AL., 2001] STAAB S., ERDMANN M., MAEDCHE A. — Engineering ontologies using semantic patterns, In *Proceedings of. IJCAI-01 Workshop on E-Business & the Intelligent Web*, (2001).

[SUNYE, 1999] SUNYE G. — PatternGen, un outil de mise en oeuvre de patterns de conception, Thèse de doctorat, Université Paris 6, (1999).

[SZYPERSKI & AL., 2002] SZYPERSKI C., GRUNTZ D, MURER S. — *Component Software : Beyond Object-Oriented Programming*, 2nd edition, Addison-Wesley, (2002).

[TARI & AL., 1996] TARI Z., CHENG W., YETONGNON K., SAVNICK I. — Toward Cooperative Database: The Distributed Object Kernel (DOK) Approach, *Proceedings of Conference on Parallel and Distributed Computing Systems (PDCS'96)*, Dijon, France, p. 595-600, (1996).

[TARI & STOKES, 1997] TARI Z., STOKES J. — *Designing a Reengineering Service for the DOK Federated Database System*, *Int. Conf. on Data Engineering (ICDE'97)*, Birmingham, p. 465-475, (1997).

[TATA, 2002] TATA S. — Policies for Cooperative Virtual Teams, *Proceedings of the 5th International Conference on Coordination Models and Languages*, York, UK, p. 340-347, April (2002).

[TATA & AL., 1999] TATA S., CANALS G., GODART C. — Specifying interactions in cooperative applications, *Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99)*, Kaiserslautern, Allemagne, 17-19 juin (1999).

[TEMPLETON & AL., 1986] TEMPLETON M., BRILL D., CHEN A., DAO S., LUND E. — Mermaid: experiences with network operation, *Second Data Engineering Conference Proceedings*, p. 292-300, (1986).

[TEMPLETON & AL., 1987] TEMPLETON M., BRILL D., DAO S., LUND E., WARD P., CHEN A., MACGREGOR R. — Mermaid: a front-end to distributed heterogeneous databases, *Proceedings of the IEEE*, Vol. 75, N°5, p. 695-708, (1987).

[TOMASIC & AL. 1995] TOMASIC A., RASCHID L., VALDURIEZ P. — Scaling heterogeneous databases and the design of Disco, Technical report, number 2704, INRIA, Rocquencourt, France, (1995).

[TRACZ, 1994] TRACZ W. — Reuse State of the Art and State of the Practice Report Card, *proceedings of the 3rd International Conference on Software Reuse*, Rio de Janeiro, Brazil, p 193-194, November 1-4 (1994).

[TRACZ, 2002] TRACZ W. — Software Reuse - Trick or Treat?, in *Flashline Newsletter*, October (2002).

[TRAFIC, 2004] TRAFIC — Site Web du projet TRAFIC (Transports : Réutilisation, Adaptation, Fiabilité, Intermodalité et Coopération inter-systèmes) de la région Rhône-Alpes, http://www-lsr.imag.fr/Les.Personnes/Vianney.Darmaillacq/Site_TRAFIC/, (2004).

[VAN DER ALST & AL., 2003] VAN DER AALST W.M.P., HOFSTEDE A.H.M., KIEPUSZEWSKI B., BARROS A.P. — Workflow Patterns, *Distributed and Parallel Databases*, 14(1), p. 5-51, (2003).

[VERHAREN, 1997] VERHAREN E. — A Language-Action Perspective on the Design of Cooperative Information Systems, Ph.D. thesis, University of Tilburg, 1997.

[VÖLTER & AL., 2002] VÖLTER M. — Server-Side Components - A Pattern Language, in *Proc. of EuroPLoP 2001*, Rüping A., Eckstein J. and Schwanninger C. (Ed.), Konstanz, Germany, p. 87-128, (2002).

[WALSHAM, 1993] WALSHAM G. — *Interpreting Information Systems in Organisations*, Wiley, Chichester, (1993).

[WARTIK & PRIETO-DIAZ, 1992] WARTIK S., PRIETO-DIAZ R. — Criteria for comparing domain analysis approaches, *International Journal of Software Engineering and Knowledge Engineering*, 2(3), p.403-431, (1992).

[WASTELL, 1999] WASTELL D.G. — Learning Dysfunctions in Information Systems Development: Overcoming the Social Defences with Transitional Objects, *MIS Quarterly,* 23, p. 581-600, (1999).

[WEISS & CHI TAU, 1999] WEISS D.M., CHI TAU R.L. — *Software Product-Line Engineering: A Family-Based Software Development Process*, Addison-Wesley, (1999).

[WIEDERHOLD, 1992] WIEDERHOLD G. — Mediators in the Architecture of Future Information Systems, *IEEE Computer*, Vol. 25, N°3, p. 38-49, March (1992).

[WIEDERHOLD & GENESERETH, 1995] WIEDERHOLD G., GENESERETH M. — The Basis for Mediation, *Proceedings of the third International Conference on Cooperative Information Systems*, p. 140-157, May (1995).

[WIJEGUNARATNE & AL., 2000] WIJEGUNARATNE I., FERNANDEZ G., VALTOUDIS J. — A Federated Architecture for Enterprise Data Integration, *Proceedings of the Australian Software Engineering Conference*, Canberra, Australie, Avril (2000).

[WINOGRAD, 1972] WINOGRAD T. — *Understanding Natural Language*, Academic Press, New York, (1972).

[WOOLDRIDGE & JENNINGS, 1995] WOOLDRIDGE M.J., JENNINGS N.R. — Intelligent agents: Theory and Practice, *The Knowledge Engineering Review*, Cambridge University Press publisher, Vol.10, N°2, pp. 115-152, June (1995).

[WYSS & AL., 2003] WYSS C.M., JAMES A., HÖPFNER H. — Report on the Engineering Federated Information System 2003 Workshop, *Proceedings of the 5th Workshop EFIS 2003*, Coventry, UK, July 17-18 (2003).

[ZIMMER, 1995] ZIMMER W. — Relationships between design patterns, *Pattern languages of program design*, ACM Press/Addison-Wesley Publishing Co., p. 345-364, (1995).

[ZMUD & PRICE, 1999] ZMUD R.W., PRICE M.F. — MIS in North America: Past, Present and Future, *12ème conférence annuelle FNEGE, Management et Technologies de l'Information: Nouvelles Organisations, Nouveaux Métiers*, Toulouse, France, 3 décembre (1999).

Annexe A :
ARRP - Atelier de Représentation et de Réutilisation des Patterns de coopération

Cette annexe a pour objectif d'expliciter le prototype de l'environnement logiciel de représentation et de support de réutilisation des patterns de coopération dénommé ARRP (Atelier de Représentation et de Réutilisation des Patterns de coopération). Nous présentons tout d'abord un rapide état de l'art sur les outils permettant la mise en œuvre de patterns (§ 12), puis détaillons l'environnement logiciel que nous avons développé (§ 12).

1. Etat de l'art sur les outils de mise en œuvre de patterns

La mise en oeuvre d'un pattern, ou développement à base de patterns, comprend trois activités distinctes :

- le choix du pattern adéquat,
- son adaptation au contexte d'utilisation ou sa recopie,
- et, éventuellement, la production du code correspondant.

La première activité passe soit par une analyse du contexte puis une tentative d'assimilation à un problème déjà référencé pour enfin suggérer le pattern le plus adéquat, soit par la mise en œuvre d'un système de navigation entre les différents patterns afin de faciliter le choix de l'utilisateur.

La deuxième activité consiste à adapter la solution du pattern au contexte d'utilisation ou recopier celle-ci. La recopie consiste à utiliser la solution telle quelle ; elle est notamment usitée dans le cadre de solutions exprimées en langage naturel. Dans le cadre de solutions de type modèle, l'adaptation consiste principalement à renommer, redéfinir, ajouter et/ou supprimer les propriétés statiques (attributs), dynamiques (méthodes) et relationnelles (associations) des entités spécifiées.

La troisième activité consiste à produire le code correspondant à la solution du pattern. Elle concerne certains patterns de conception et patterns d'implantation.

La mise en œuvre de patterns nécessite cependant leurs représentation et stockage préalables.

Depuis l'émergence des patterns dans la communauté objet, plusieurs outils ont été proposés permettant de faciliter ou d'automatiser (souvent partiellement) cette mise en œuvre. Ces outils diffèrent cependant par leurs objectifs (relatifs aux activités de mise en œuvre des patterns), fonctionnalités et type [BORNE & REVAULT, 1999].

Les objectifs principaux sont ainsi multiples [BORNE & REVAULT, 1999] :

- aide à la conception : permet la mise en forme de classes et de relations sur le modèle de la structure préconisée par un ou plusieurs patterns,
- validation de la conception : vise à valider les modèles respectant les structures proposées par les patterns,
- génération de code : génération de programmes à partir des structures de modélisation proposées par les patterns,
- traçabilité : conservation de la trace de l'application des patterns,
- rétro-conception : reprise de modèles de conception existants pour y appliquer les solutions préconisées par les patterns.

Les fonctionnalités, permettant d'atteindre les objectifs précédents, sont elles aussi diverses :

- fonctions de navigation entre les patterns, voire les classes et méthodes inhérentes,
- fonctions de gestion d'une bibliothèque de patterns permettant l'ajout, la consultation, la modification et la suppression de patterns,
- fonctions d'adaptation et d'instanciation de la structure proposée par un pattern,
- opérations spécifiques sur les patterns permettant de modifier ou d'ajouter des éléments à la structure d'un pattern,
- possibilité de composer des patterns pour fournir des moyens de construire un pattern composé formant une entité à part entière,
- fonctionnalités de vérification de la cohérence permettant de valider une conception à base de patterns.

Borne & Revault [BORNE & REVAULT, 1999] distinguent trois types d'outils couvrant certaines de ces fonctionnalités. La première catégorie d'outils propose des extensions de langages de

programmation (au niveau du modèle objet) en intégrant les éléments de description de patterns dans les primitives des langages. (e.g. LayOM [BOSCH, 1997]). La deuxième catégorie d'outils fournit des systèmes de génération de code, pour implanter un pattern, indépendants à la fois de l'environnement de modélisation et du langage de programmation cible (par exemple, Pat [KRÄMER & PRECHELT,1996], [BUDINSKY ET AL., 1996]). Il s'agit d'outils de type boite noire auxquels l'utilisateur fournit des valeurs de paramètres en entrée. La troisième catégorie d'outils étend les environnements de développement et fournit à ces derniers des fonctionnalités pour la représentation et l'instanciation de patterns (e.g. Pattern Tool [MEIJERS, 1996], Framework Studio [BLUEPRINT TECHNOLOGIES, 2004], AGAP [CONTE & AL., 2001B],…). Ces derniers outils, destinés à améliorer les fonctionnalités d'un environnement de génie logiciel, sont, d'après Borne & Revault, les plus complets et prometteurs. ARRP est un environnement logiciel relevant de cette troisième catégorie. Ses objectifs et fonctionnalités sont présentées en section suivante (§ 2).

2. Présentation d'ARRP

2.1. Objectifs d'ARRP

ARRP a pour objectif de faciliter la tâche d'ingénierie des systèmes d'information coopératifs par réutilisation de patterns. Il assure ainsi la mise en œuvre – partielle – des patterns de coopération que nous avons conçus, mais aussi d'autres patterns pouvant être complémentaires à ceux que nous avons spécifiés (par exemple, d'autres patterns de support technique).

L'environnement logiciel a plus précisément deux objectifs principaux : la représentation et gestion des patterns et le développement d'applications à base de patterns. Nous détaillons chacun de ces deux objectifs.

1. la représentation et la gestion des patterns : l'atelier permet au concepteur de patterns de définir des formalismes, des collections de patterns, des patterns décrits suivant ces formalismes et également d'organiser ces patterns et collections. Les principales fonctionnalités sont :

- Visualiser un pattern
- Visualiser une collection de patterns
- Créer un formalisme
- Modifier un formalisme
- Créer une collection de patterns

- Créer un pattern
- Modifier un pattern
- Organiser des patterns

2. Développement d'applications à base de patterns : l'atelier permet au concepteur ou développeur d'applications d'appliquer des patterns pour modéliser et concevoir son système d'information coopératif. Cette application repose sur la présence de plusieurs collections de patterns dans ARRP. Les principales fonctionnalités sont les suivantes [91]:

- Visualiser une collection de patterns
- Rechercher un pattern
- Visualiser un pattern

La version actuelle d'ARRP[92] est ainsi principalement destinée à la représentation des patterns et à la sélection du (ou des) pattern(s) de coopération adapté(s) à la résolution d'un problème spécifique.

2.2. Méta-modèle de l'atelier ARRP

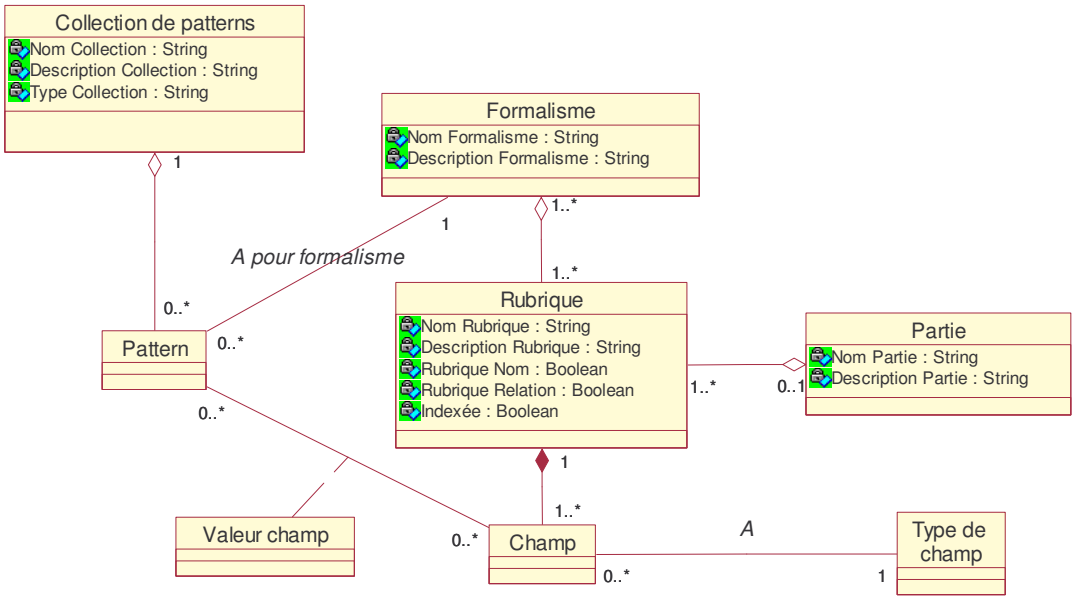


Figure 40 : Méta-modèle de l'atelier ARRP

[91] Une version future du logiciel actuel assurera également l'adaptation (création, modification et suppression d'une adaptation de pattern) et la traçabilité des patterns de coopération.

[92] ARPP Version 0.2 a été développée en Visual Basic.

Le méta-modèle d'ARRP (Cf. Figure 40) est constitué de deux composants métier, les composants "Formalisme" et "Collection de patterns".

- *Composant Formalisme :*

ARRP permet de créer et de gérer des patterns. Pour ce faire, il a besoin de définir des formalismes de patterns. Dans ARRP, un formalisme est défini comme étant un ensemble de rubriques regroupées ou non en parties. Une rubrique peut éventuellement être de type nom[93] ou relation[94] et peut être indexée ou non[95]. Elle est constituée d'un ou plusieurs champs de type texte (langage naturel), expression logique ou modèle (diagramme de classes UML, diagramme d'activité, modèle d'architecture,…).

- *Composant Collection de patterns :*

Une collection de patterns est composée de zéro ou plusieurs patterns. Chaque collection a un type spécifique (catalogue, système, langage). Un pattern est décrit conformément à un formalisme spécifique. Ce formalisme peut-être identique pour tous les patterns appartenant à une unique collection ou différent (cas des patterns appartenant au système de patterns de support technique). Les valeurs associées aux champs de chaque pattern doivent respecter la définition donnée dans le formalisme du champ.

2.3. Description des fonctionnalités et formulaires graphiques d'ARRP

ARRP assure principalement les fonctionnalités de création d'une collection de patterns (en l'occurrence les systèmes de patterns de coopération que nous avons conçus), la création d'un formalisme de pattern, la création de patterns et la recherche de patterns (préalable à la phase d'adaptation à un contexte d'utilisation spécifique). Les trois premières fonctionnalités sont utilisées une seule fois pour définir et stocker les patterns dans l'atelier. La dernière fonctionnalité est utilisée à chaque nouveau projet de spécification d'un SIC. Nous décrivons dans les sous-sections suivantes (§ 2.3.1 à 2.3.4) le scénario associé à chacune de ces fonctionnalités et présentons quelques interfaces graphiques correspondantes.

[93] Une rubrique nom est une rubrique stockant le nom d'un pattern. Cette rubrique pourra être dénommée différemment selon le formalisme usité : nom, identifiant, nom du problème,…

[94] Une rubrique relation indique que cette rubrique permet de stocker une relation inter-patterns ou entre les artefacts constituant les patterns.

[95] Une rubrique indexée est une rubrique sur laquelle une recherche textuelle pourra être effectuée lors de la sélection d'un pattern.

2.3.1. Création d'une collection de patterns

Scénario :

- Saisir le nom de la collection, sa description et son type (catalogue, système ou langage) (Cf. Figure 41)
- Enregistrer la description de la collection de patterns

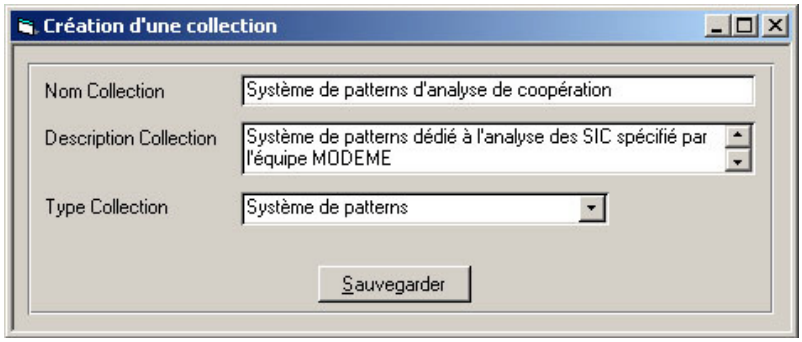


Figure 41 : Création d'une collection de patterns

2.3.2. Création d'un formalisme de patterns

Scénario :

- Saisir le nom et la description du formalisme (Cf. Figure 42)
- Saisir les rubriques (et leur description) associées au formalisme (Cf. Figure 43, Figure 44 et Figure 45)
- Regrouper éventuellement les rubriques en parties (Cf. Figure 46)

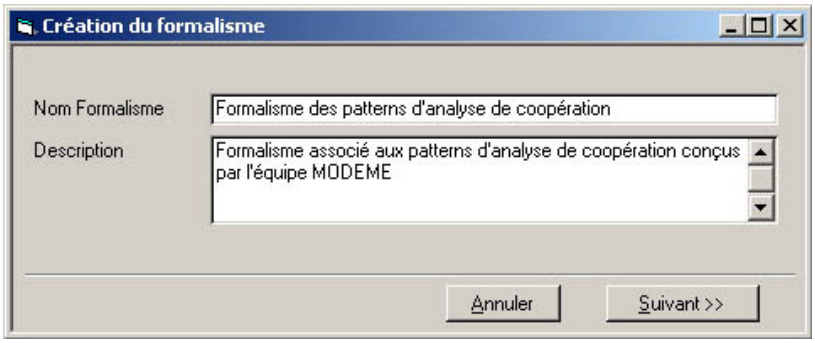


Figure 42 : Saisie du nom et de la description d'un formalisme

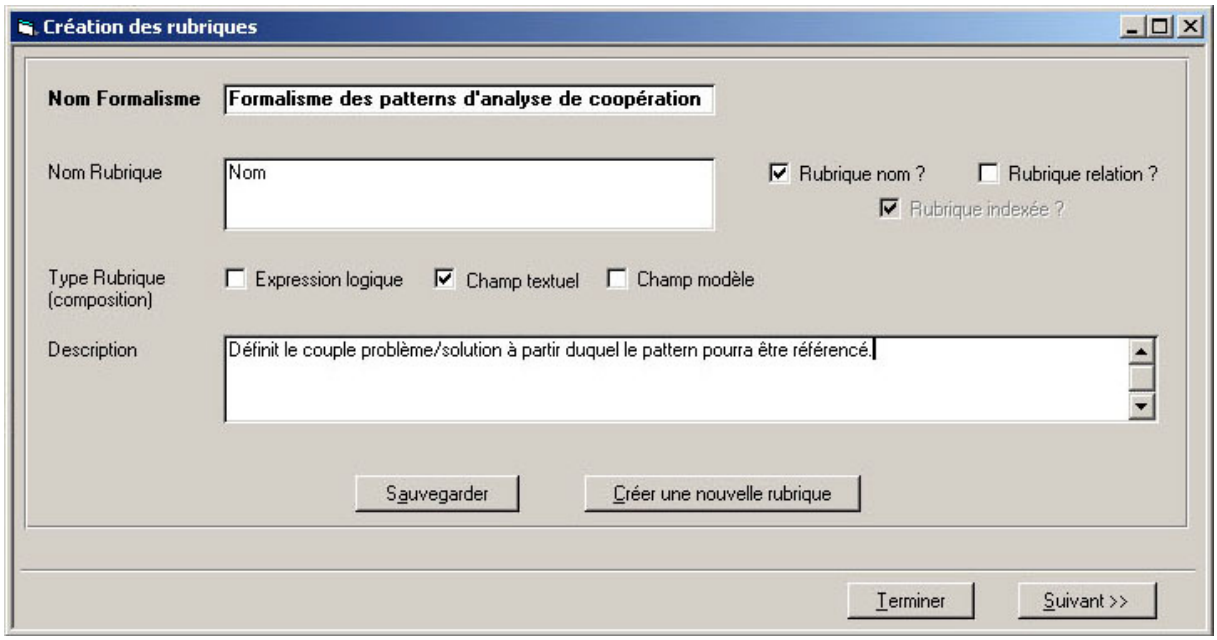


Figure 43 : Création des rubriques du formalisme (cas d'une rubrique Nom)

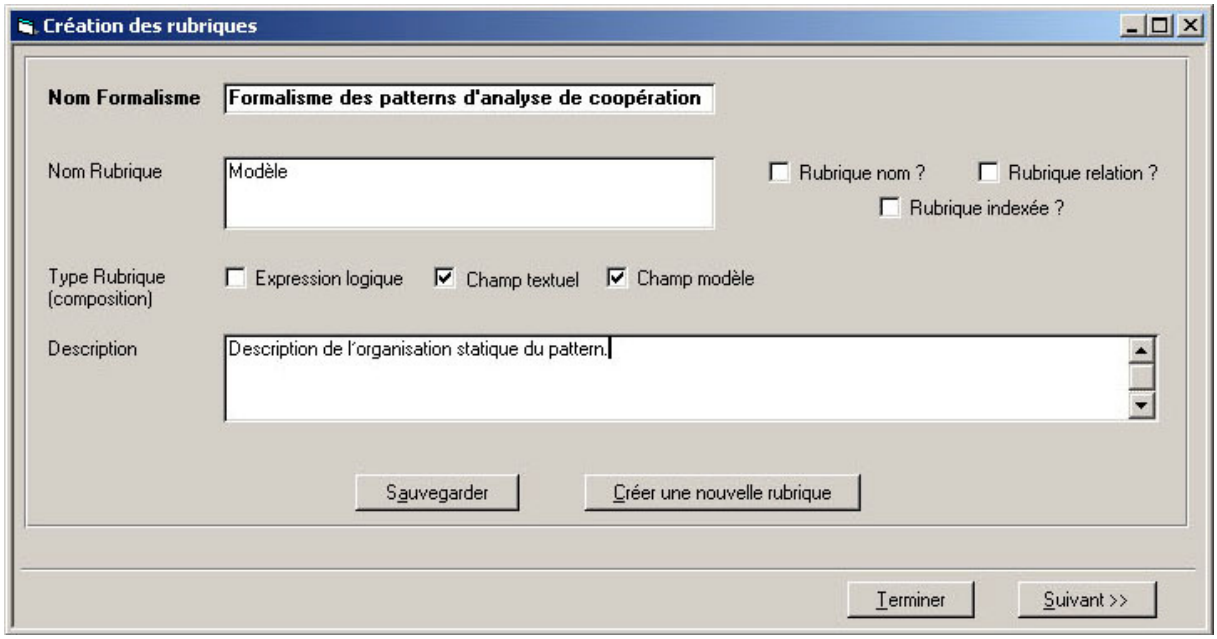


Figure 44 : Création des rubriques du formalisme (cas d'une rubrique contenant un champ de type modèle)

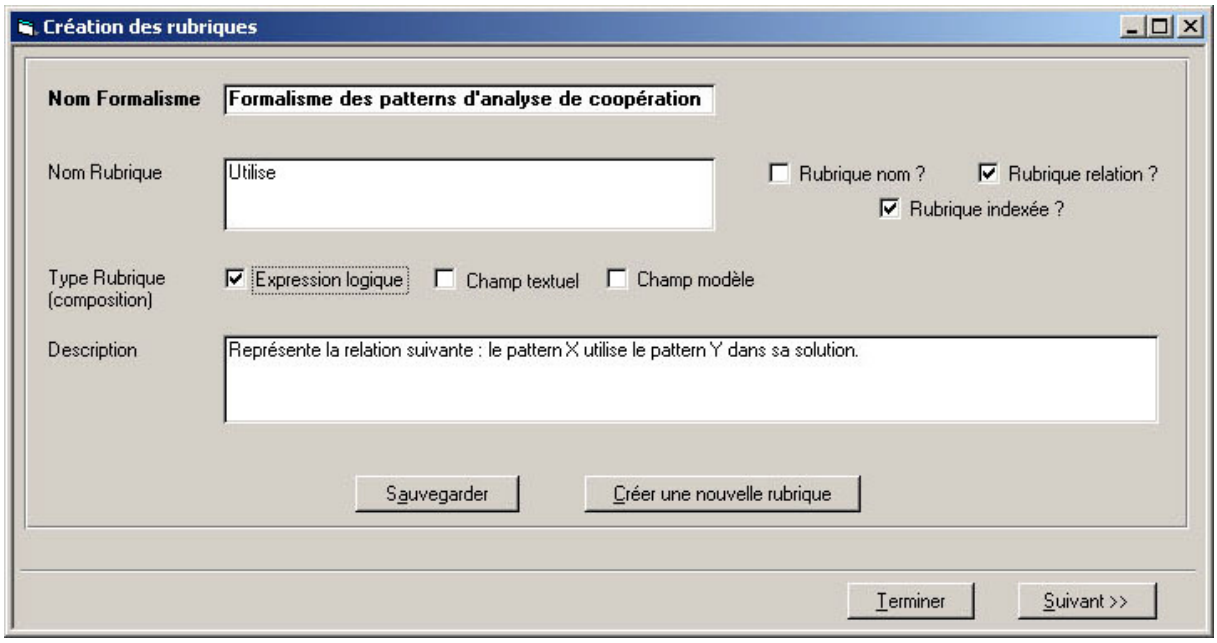


Figure 45 : Création des rubriques du formalisme (cas d'une rubrique contenant des relations inter-patterns)

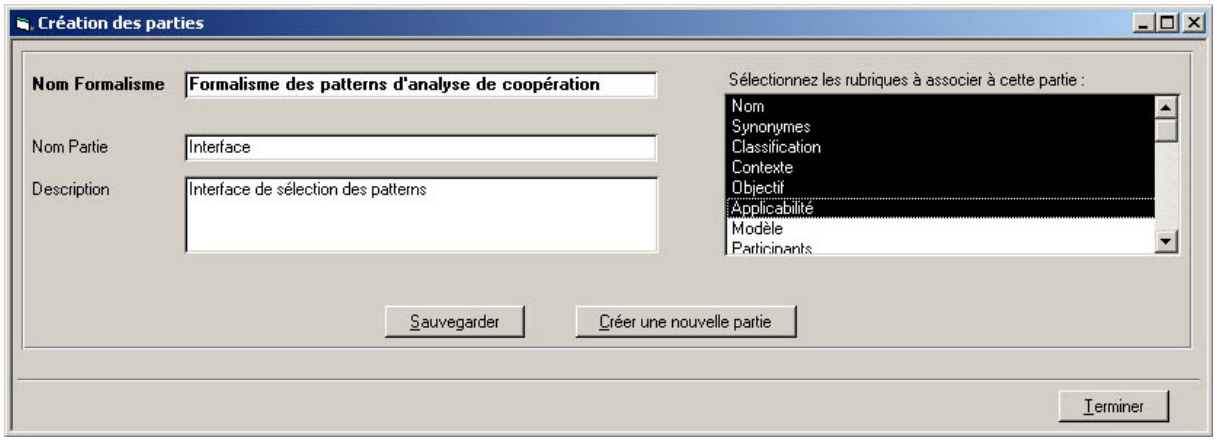


Figure 46 : Création des parties du formalisme et regroupement des rubriques en fonction de ces parties

2.3.3. Création d'un pattern

Scénario :

- Rechercher les formalismes prédéfinis
- Choisir un formalisme parmi les formalismes prédéfinis (Cf. Figure 47)

- Créer un nouveau formalisme dans le cas où aucun formalisme prédéfini n'est adapté au pattern (Cf. section 2.3.2)
- Rechercher et visualiser les collections de patterns existantes dans l'atelier
- Choisir une collection de patterns ou en créer une nouvelle si nécessaire (Cf. Figure 48)
- Saisir le contenu de chaque rubrique[96] (champs) (Cf. Figure 49 et Figure 50)
- Enregistrer le contenu du pattern créé

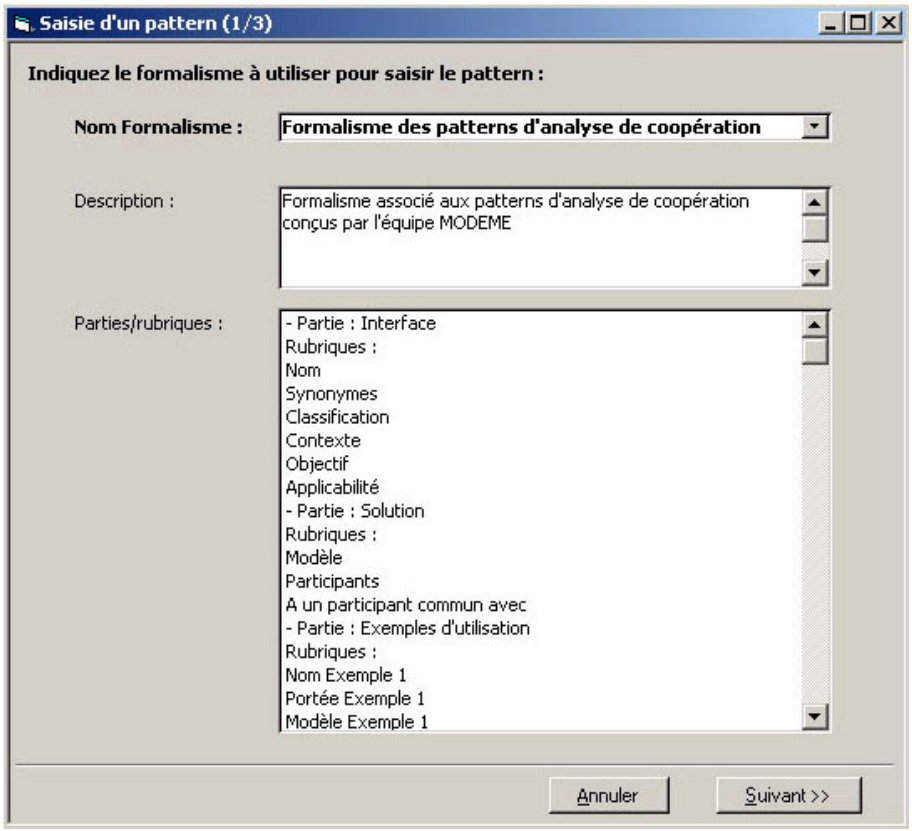


Figure 47 : Saisie d'un pattern (choix du formalisme)

[96] Un champ de type "modèle" (diagramme UML, modèle d'architecture,…) ne contient que le lien vers le fichier contenant ce modèle.

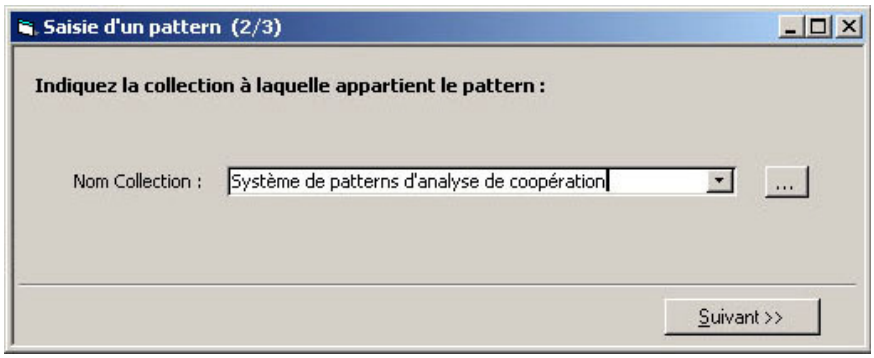


Figure 48 : Saisie d'un pattern (sélection de la collection de patterns)

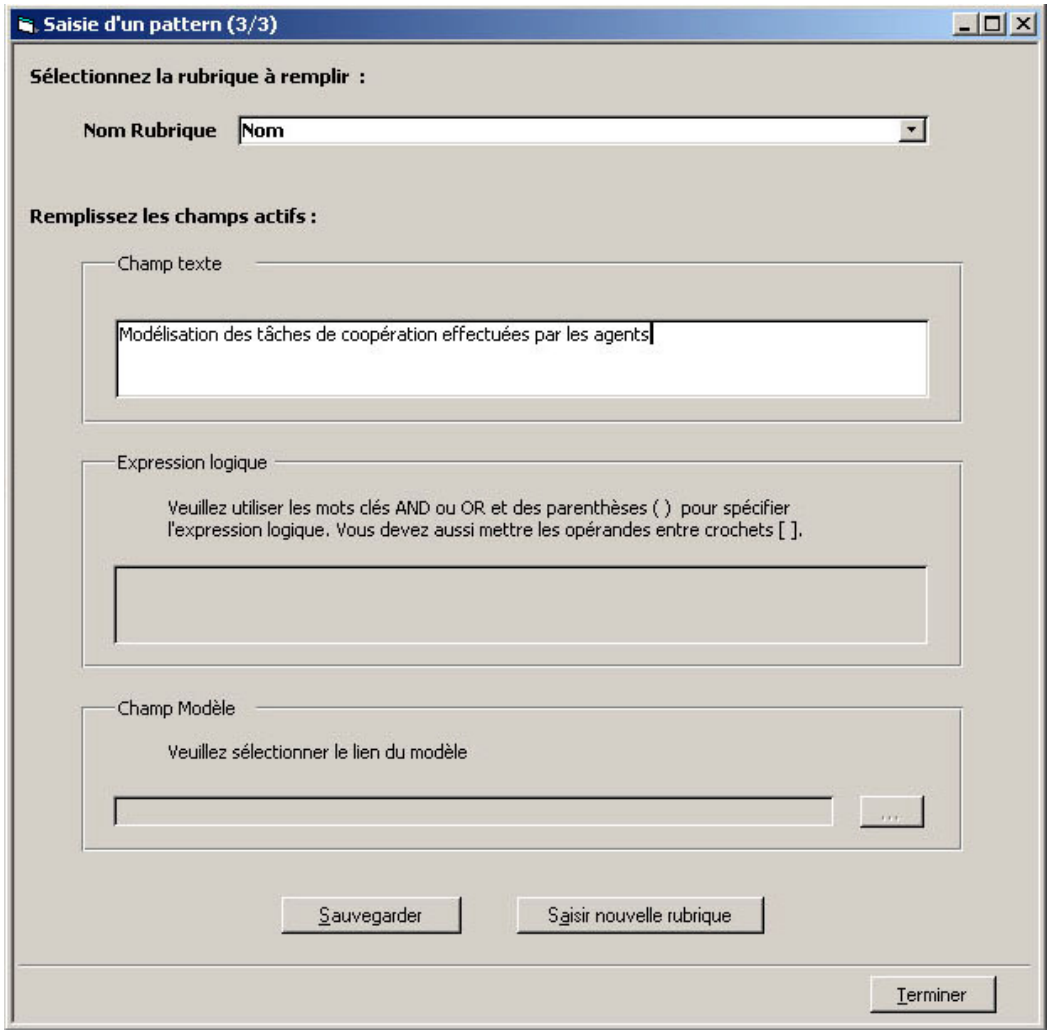


Figure 49 : Saisie des valeurs de champs du pattern (cas d'un champ de type texte)

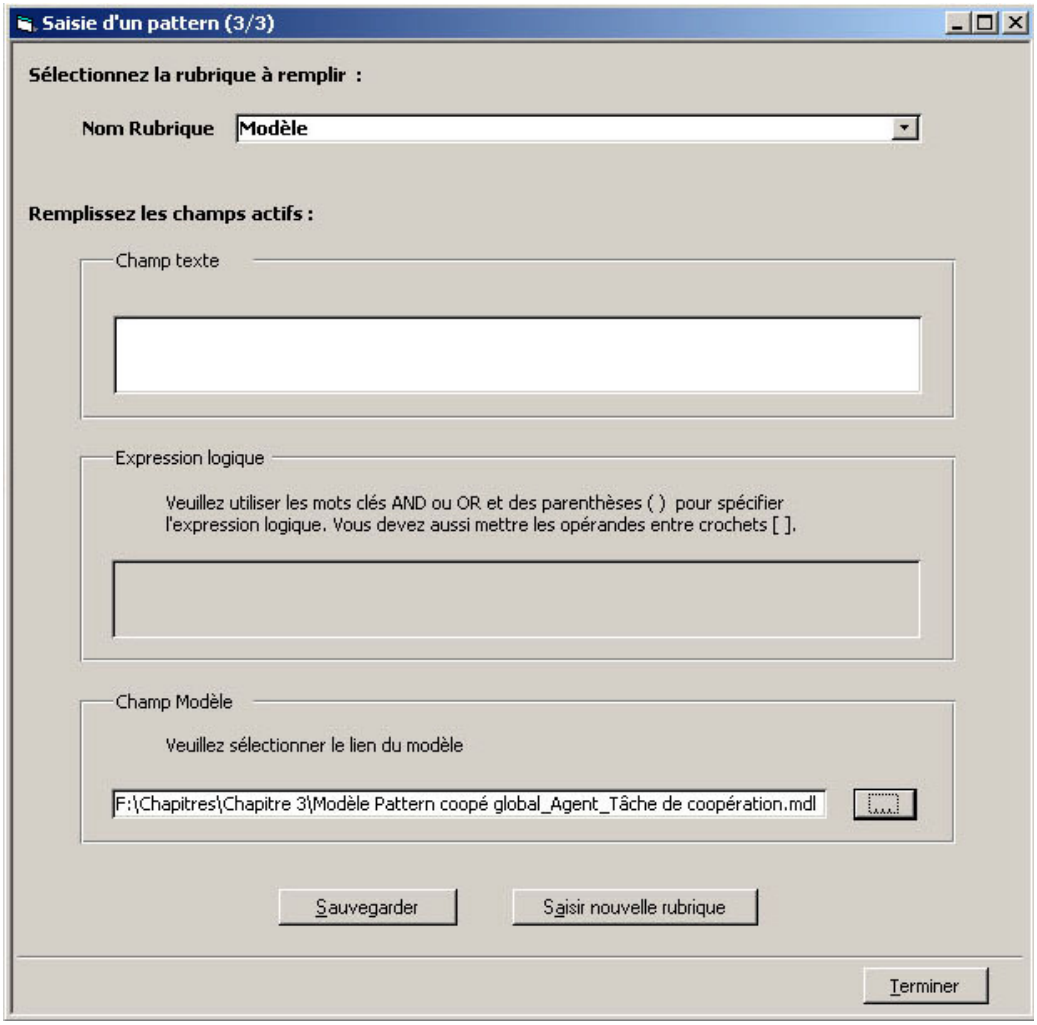


Figure 50 : Saisie des valeurs de champs du pattern (cas d'un champ de type modèle)

2.3.4. Sélection d'un pattern

Scénario :

- Effectuer une recherche textuelle[97] (mots-clés) sur l'ensemble des patterns de coopération disponibles ou sélectionner un pattern dans la liste déroulante (Cf. Figure 51).
- Visualiser les patterns correspondants à la recherche, ainsi que les patterns liés (Cf. Figure 52).
- Sélectionner le pattern adéquat dont la solution sera, par la suite, adaptée.
- Afficher l'ensemble des rubriques du pattern avec leurs différents champs (Cf. Figure 53).

[97] La recherche ne s'effectue que sur les rubriques indexées définies lors de la création du formalisme.

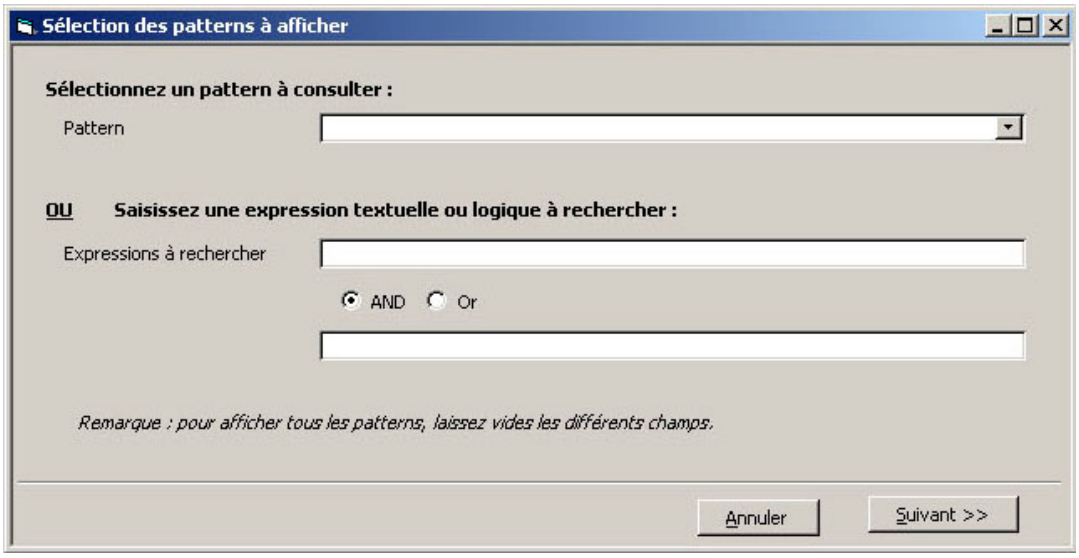


Figure 51: Recherche d'un ou plusieurs patterns

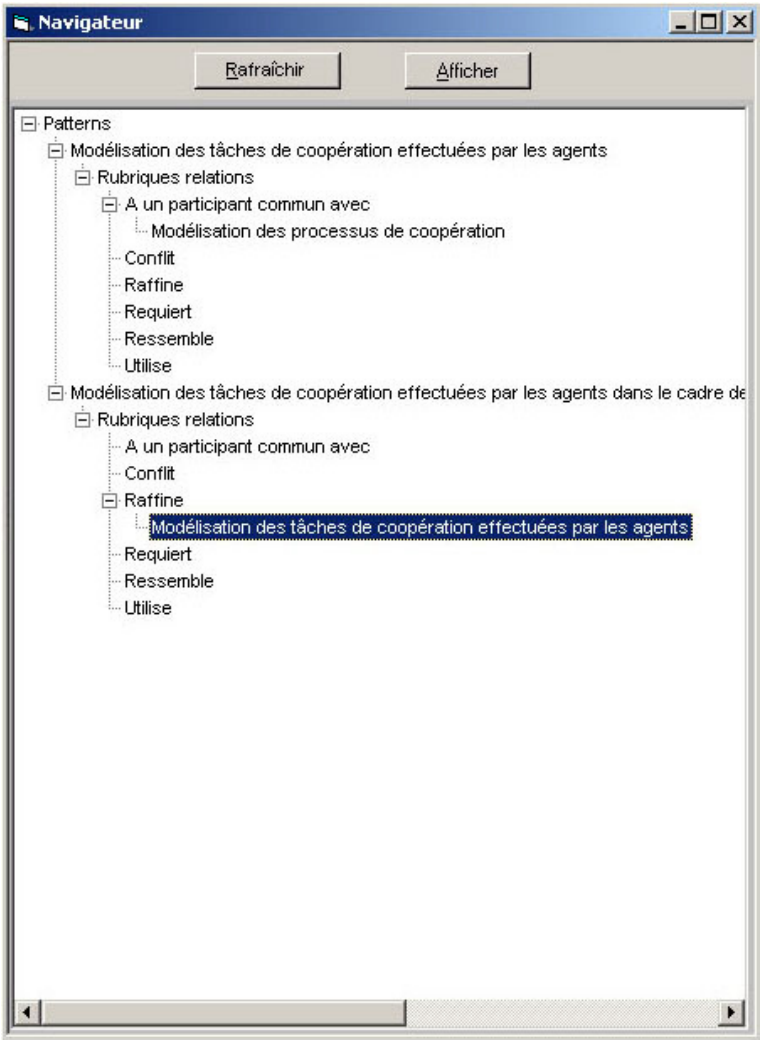


Figure 52 : Visualisation du résultat de la recherche

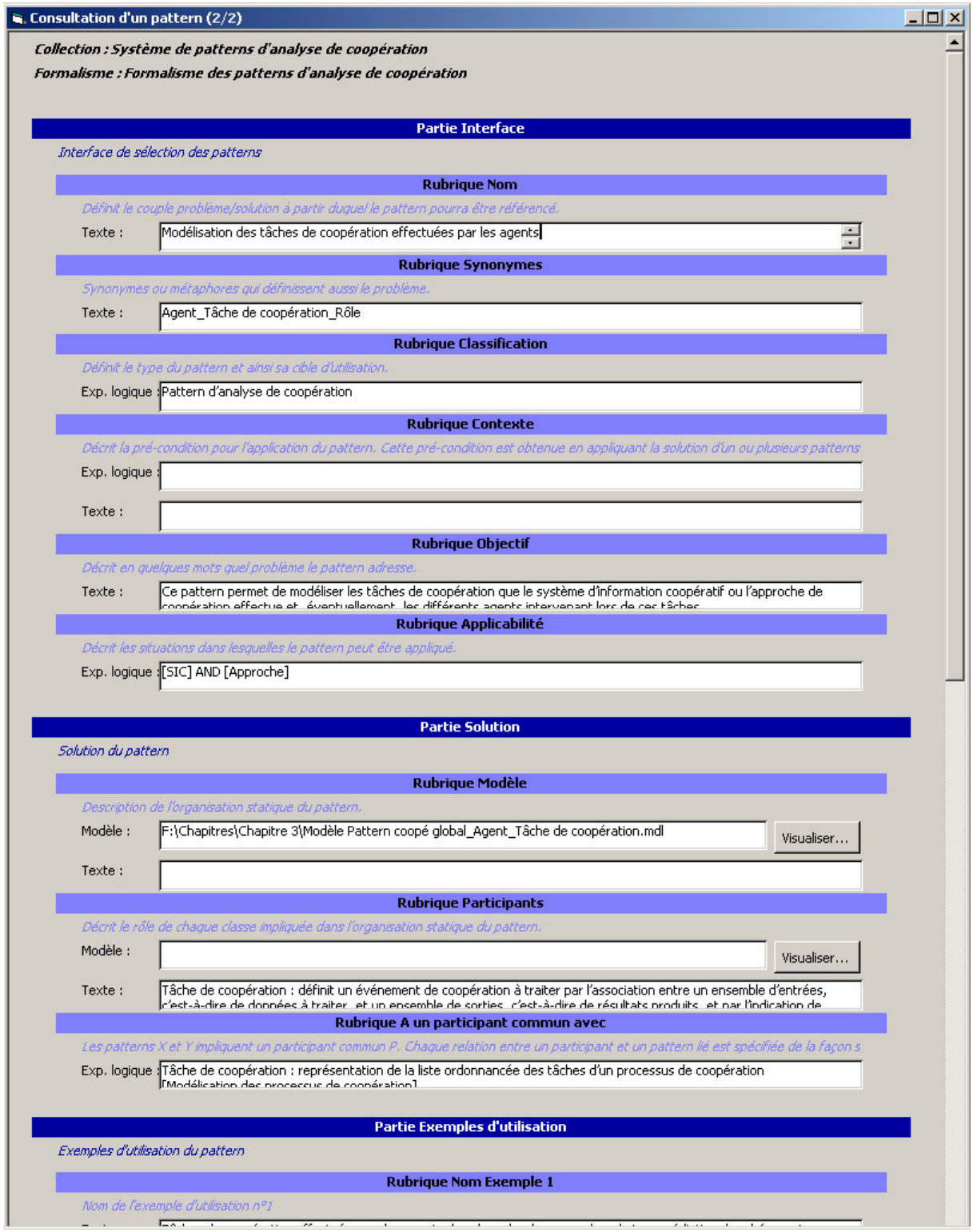


Figure 53 : Consultation d'un pattern

3. Conclusion

L'atelier ARRP facilite la réutilisation des différents systèmes de patterns présentés. Il propose une assistance à l'ingénierie des systèmes d'information coopératifs basée sur la réutilisation de patterns. Il permet notamment de représenter, stocker et rechercher les patterns dans une base spécifique et présente les relations entre ceux-ci. Il préfigure ainsi ce que pourrait être un environnement d'aide à la conception de SIC après l'ajout de fonctionnalités essentielles telles que la gestion de l'adaptation et de l'instanciation des patterns.

Index des figures

Figure 1 : Développement pour la réutilisation vs Développement par réutilisation *22*
Figure 2 : Modèle de domaine de gestion d'un hôtel (d'après [SNOECK & POELS, 2000]) *24*
Figure 3 : Les cinq niveaux de schémas de l'approche fédérée *38*
Figure 4 : Architecture basée sur un système de médiation *42*
Figure 5 : Utilisation des avoirs logiciels dans le cycle de développement du logiciel *61*
Figure 6 : Proposition d'une classification des patterns logiciels *64*
Figure 7 : Un exemple de pattern d'architecture, le pattern « Pipes and Filters » *67*
Figure 8 : Le pattern de conception "Composite" [GAMMA & AL., 1995] *69*
Figure 9 : Positionnement des patterns conçus en fonction des étapes du processus d'ingénierie de SIC *86*
Figure 10 : Rubriques du formalisme associé aux patterns d'analyse de coopération *105*
Figure 11 : Rubriques du formalisme associé aux patterns d'architecture de coopération *107*
Figure 12 : Rubriques du formalisme associé aux patterns de conception de coopération *109*
Figure 13 : Rubriques du formalisme associé aux patterns de support technique *110*
Figure 14 : Rubriques du formalisme associé aux patterns de support d'utilisation *111*
Figure 15 : Positionnement des recherches en ingénierie des systèmes d'information *113*
Figure 16 : Structure de l'entité « Source d'information » *139*
Figure 17: Les différents types de sources d'information locales gérées par les projets de coopération *139*
Figure 18 : Modélisation des entités « Conflit » et « Type de conflit » *140*
Figure 19 : Types de conflit spécifiés dans le projet de coopération ACSIS *141*
Figure 20 : Types de conflit spécifiés dans le projet de coopération DILEMMA *141*
Figure 21 : Types de conflit spécifiés dans le projet de coopération COIN *143*
Figure 22 : Structure de l'entité « Tâche de coopération » *146*
Figure 23 : Tâches de coopération spécifiées dans le projet MRDSM *147*
Figure 24 : Tâches de coopération spécifiées dans le projet DILEMMA *148*
Figure 25 : Structure de l'entité « Processus de coopération » *149*
Figure 26 : Structure de l'entité « Référentiel de coopération » *150*
Figure 27 : Les différents types de référentiel de coopération *151*
Figure 28 : Structure de l'entité « Modèle » *152*
Figure 29 : Les différents modèles utilisés dans le projet de coopération InfoSleuth [BAYARDO & AL., 1997] [NODINE & AL., 2000] *153*
Figure 30 : Structure de l'entité « Acteur » *154*
Figure 31 : Les entités « Acteur », « Expert » et « Utilisateur » *154*

Figure 32 : Les entités « Module fonctionnel », « Module fonctionnel conceptuel » et « Module fonctionnel logiciel » *155*
Figure 33 : Modules fonctionnels spécifiés dans l'approche à base de médiation de schémas *156*
Figure 34 : Modules fonctionnels spécifiés dans le projet DILEMMA *157*
Figure 35 : Modules fonctionnels spécifiés dans le projet InfoSleuth *158*
Figure 36 : Structure de l'entité « Rôle » *159*
Figure 37 : Modèle du domaine de la coopération de systèmes d'information *161*
Figure 38 : Patterns d'analyse dérivés du modèle du domaine de la coopération de systèmes d'information .. *162*
Figure 39 : Les niveaux ACSIS et la description des données locales. *212*
Figure 40 : Méta-modèle de l'atelier ARRP *268*
Figure 41 : Création d'une collection de patterns *270*
Figure 42 : Saisie du nom et de la description d'un formalisme *270*
Figure 43 : Création des rubriques du formalisme (cas d'une rubrique Nom) *271*
Figure 44 : Création des rubriques du formalisme (cas d'une rubrique contenant un champ de type modèle) . *271*
Figure 45 : Création des rubriques du formalisme (cas d'une rubrique contenant des relations inter-patterns) *272*
Figure 46 : Création des parties du formalisme et regroupement des rubriques en fonction de ces parties *272*
Figure 47 : Saisie d'un pattern (choix du formalisme) *273*
Figure 48 : Saisie d'un pattern (sélection de la collection de patterns) *274*
Figure 49 : Saisie des valeurs de champs du pattern (cas d'un champ de type texte) *274*
Figure 50 : Saisie des valeurs de champs du pattern (cas d'un champ de type modèle) *275*
Figure 51: Recherche d'un ou plusieurs patterns *276*
Figure 52 : Visualisation du résultat de la recherche *276*
Figure 53 : Consultation d'un pattern *277*

Index des tableaux

Tableau 1 Comparaison des différentes approches de coopération *57*
Tableau 2 : Exemple de pattern d'analyse de domaine de Fowler [FOWLER, 1997] *66*
Tableau 3 : Rubriques de la partie « Interface » *101*
Tableau 4 : Rubriques de la partie « Solution » *103*
Tableau 5 : Rubriques de la partie « Relation » *104*
Tableau 6 : Comparaison des patterns de coopération proposés par divers travaux de recherche *133*
Tableau 7 : Pattern d'analyse de coopération « Modélisation des types de conflit » *167*
Tableau 8 : Pattern d'analyse de coopération « Modélisation des sources d'information gérées » *170*
Tableau 9 : Pattern d'analyse de coopération « Modélisation du référentiel de coopération » *174*
Tableau 10 : Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents » *178*
Tableau 11 : Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » *185*
Tableau 12 : Pattern d'analyse de coopération « Modélisation des processus de coopération » *189*
Tableau 13 : Pattern de support d'utilisation « Utilisation des patterns d'analyse de coopération pour la spécification d'un SIC » *192*
Tableau 14 : Pattern de support d'utilisation « Utilisation des patterns d'analyse de coopération pour la spécification d'une approche de coopération » *193*
Tableau 15 : Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche fédérée » *201*
Tableau 16 : Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche à base de médiation de schémas » *204*
Tableau 17 : Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche à base de médiation de contextes » *208*
Tableau 18 : Pattern de support d'utilisation « Utilisation des patterns d'architecture de coopération » *210*
Tableau 19 : Pattern de conception de coopération « Objet Descriptif des Données » *215*
Tableau 20: Pattern de conception de coopération « Lien sémantique » *218*
Tableau 21 : Pattern de conception de coopération « Traduction ACSIS » *220*
Tableau 22 : Pattern de conception de coopération « Création des liens sémantiques » *223*
Tableau 23 : Pattern de conception de coopération « Contract-Net Protocol » *226*
Tableau 24 : Pattern de conception de coopération « Interaction d'agents » *229*
Tableau 25 : Pattern de support technique « Démarche de transformation de modèle » *231*
Tableau 26: Pattern de support technique «Traduction des interfaces spécifiées en OMG IDL vers Microsoft IDL » *232*

Tableau 27: Pattern de support d'utilisation « Création de la couche de description des données ACSIS »..... 234
Tableau 28 : Pattern de support d'utilisation « Représentation des interactions entre les composants de l'architecture ACSIS » 235

Table des matières

SOMMAIRE **9**

INTRODUCTION GENERALE **11**

CHAPITRE 1 : L'INGENIERIE DES SYSTEMES D'INFORMATION COOPERATIFS **15**

1. INTRODUCTION 15
2. DE L'INGENIERIE DES SYSTEMES D'INFORMATION… 15
   2.1. Les systèmes d'information *15*
   2.2. L'ingénierie des systèmes d'information *16*
   2.3. La réutilisation, une aide à l'ingénierie des systèmes d'information *20*
      2.3.1. Définition de la réutilisation 20
      2.3.2. Développement d'un système de réutilisation vs développement d'un système d'information par réutilisation 21
      2.3.3. Les niveaux de réutilisation 22
      2.3.4. Les avoirs logiciels réutilisables 23
         2.3.4.1. Les modèles de domaine 23
         2.3.4.2. Les patterns 25
         2.3.4.3. Les composants logiciels 27
         2.3.4.4. Les frameworks 29
3. … A L'INGENIERIE DES SYSTEMES D'INFORMATION COOPERATIFS 30
   3.1. Les systèmes d'information coopératifs : définition et caractéristiques *30*
      3.1.1. Définition 31
      3.1.2. Caractéristiques et propriétés des SIC 33
         3.1.2.1. L'autonomie 33
         3.1.2.2. L'hétérogénéité 34
         3.1.2.3. La distribution 35
         3.1.2.4. La transparence 36
         3.1.2.5. L'évolutivité 36
         3.1.2.6. La scalabilité 36
      3.1.3. Taxonomie des systèmes d'information coopératifs 36

3.1.3.1. L'approche fédérée 37
3.1.3.2. Les approches à base de médiation 40
3.1.3.3. Comparaison des approches 44
3.2. L'ingénierie des systèmes d'information coopératifs 58
4. VERS LE CHOIX D'UN TYPE D'AVOIR LOGICIEL ADAPTE A L'INGENIERIE DES SYSTEMES D'INFORMATION COOPERATIFS : LES PATTERNS 60
4.1. Le choix du type d'avoir logiciel adapté à l'ingénierie des SIC 60
4.2. Les patterns logiciels 62
4.2.1. Classification des patterns 62
4.2.1.1. Classification des patterns en fonction de leur utilisation dans le cycle de développement du logiciel 63
4.2.1.2. Patterns génératifs vs patterns non génératifs 71
4.2.2. Collection de patterns 72
4.2.2.1. Catalogue de patterns 72
4.2.2.2. Système de patterns 72
4.2.2.3. Langage de patterns 72
4.2.3. Mise en oeuvre des patterns 73
4.2.4. Avantages des patterns pour l'ingénierie des SI 74
4.2.4.1. Aide à la conception de logiciels 74
4.2.4.2. Transfert de la connaissance 75
4.2.4.3. Aide à la compréhension des systèmes 75
4.2.4.4. Définition et enrichissement d'un vocabulaire commun 75
4.2.4.5. Complémentarité avec les outils et méthodes existants 76
5. CONCLUSION 76

CHAPITRE 2 : PRESENTATION DES PATTERNS DE COOPERATION 77

1. INTRODUCTION 77
2. TYPOLOGIE DES PATTERNS 77
2.1. Patterns de coopération 78
2.1.1. Patterns de coopération dédiés à l'analyse des SIC 79
2.1.2. Patterns de coopération dédiés à la conception des SIC 80
2.1.2.1. Patterns d'architecture de coopération 80
2.1.2.2. Patterns de conception de coopération 81
2.2. Patterns de support de coopération 83
2.2.1. Patterns de support technique 83
2.2.2. Patterns de support d'utilisation 84
2.3. Positionnement des patterns conçus 86

3. DEMARCHE D'INGENIERIE DES PATTERNS DE COOPERATION 87
3.1. Introduction 87
3.2. Identification des éléments réutilisables (problèmes) 88
3.2.1. Préambule : techniques d'identification des éléments réutilisables 88
3.2.2. Les patterns d'analyse de coopération et l'analyse de domaine 91
3.2.3. Les patterns d'architecture de coopération et la conception de domaine 93
3.2.4. Les patterns de conception de coopération et la rétro-conception 94
3.2.5. Réorganisation des éléments réutilisables 94
3.3. Spécification des solutions des patterns 95
4. FORMALISME DES PATTERNS DE COOPERATION ET SUPPORT DE COOPERATION 96
4.1. Quel degré de formalisation pour les patterns ? 97
4.1.1. Les notations mathématiques ou langages formels 97
4.1.2. Le langage naturel (expressions textuelles) 98
4.1.3. Les langages semi-formels 98
4.1.4. Approches de modélisation retenues 98
4.2. Structure documentaire des patterns de coopération et de support de coopération 99
4.2.1. Rubriques du formalisme 100
4.2.1.1. Partie Interface 100
4.2.1.2. Partie Solution 101
4.2.1.3. Partie Relation 103
4.2.2. Formalisme des différents types de patterns conçus 105
4.2.2.1. Formalisme des patterns d'analyse de coopération 105
4.2.2.2. Formalisme des patterns d'architecture de coopération 107
4.2.2.3. Formalisme des patterns de conception de coopération 109
4.2.2.4. Formalisme des patterns de support technique 110
4.2.2.5. Formalisme des patterns de support d'utilisation 111
5. VALIDATION ET REUTILISATION DES PATTERNS DE COOPERATION ET DE SUPPORT DE COOPERATION 112
5.1. Introduction 112
5.2. Validation des patterns 114
5.3. Réutilisation des patterns 114
6. TRAVAUX DU DOMAINE 116
6.1. Patterns issus des domaines de l'Intelligence Artificielle Distribuée et des Bases de données 117
6.1.1. Patterns issus de l'Intelligence Artificielle Distribuée 117
6.1.1.1. Patterns relatifs aux agents et systèmes multi-agents 117
6.1.1.2. Patterns dédiés aux ontologies 125

6.1.2. Patterns dédiés au domaine des bases de données 127
6.2. Patterns de coopération de systèmes d'information 127
6.2.1. Patterns dédiés à l'interopérabilité des données 128
6.2.1.1. Patterns de conception d'une architecture fédérée de Fernandez & al. 128
6.2.1.2. Pattern d'interopérabilité sémantique de Ganguly & al. 128
6.2.1.3. Les patterns de coopération de l'équipe SIGMA 130
6.2.2. Patterns dédiés à l'interopérabilité de procédés et de processus : le projet ECOO131
6.2.3. Positionnement de nos patterns par rapport aux patterns de coopération spécifiés par les travaux du domaine 131
7. CONCLUSION 134

CHAPITRE 3 : PATTERNS DEDIES A L'ANALYSE DES SYSTEMES D'INFORMATION COOPERATIFS 137

1. INTRODUCTION 137
2. REFERENTIEL DU DOMAINE DE LA COOPERATION 137
2.1. Entités et concepts caractéristiques du domaine de la coopération 138
2.1.1. Source d'information 138
2.1.2. Conflit 140
2.1.3. Tâche de coopération 144
2.1.4. Processus de coopération 149
2.1.5. Référentiel de coopération 150
2.1.6. Modèle 151
2.1.7. Acteur 153
2.1.8. Module fonctionnel 154
2.1.9. Rôle 159
2.2. Modèle du domaine de la coopération de systèmes d'information 160
3. PATTERNS D'ANALYSE DE COOPERATION 161
3.1. Introduction 161
3.2. Présentation des patterns d'analyse de coopération 163
3.2.1. Pattern d'analyse de coopération « Modélisation des types de conflit » 164
3.2.2. Pattern d'analyse de coopération « Modélisation des sources d'information gérées » 168
3.2.3. Pattern d'analyse de coopération « Modélisation du référentiel de coopération » 171
3.2.4. Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents » 174

3.2.5. Pattern d'analyse de coopération « Modélisation des tâches de coopération effectuées par les agents dans le cadre des approches à base de médiation de contextes et de médiation de schémas » ... 178
3.2.6. Pattern d'analyse de coopération « Modélisation des processus de coopération » 186
3.3. Réutilisation des patterns d'analyse de coopération ... *189*
4. TRAVAUX FUTURS ... 193
5. CONCLUSION ... 194

CHAPITRE 4 : PATTERNS DEDIES A LA CONCEPTION ET A L'IMPLANTATION DES SYSTEMES D'INFORMATION COOPERATIFS ... 195

1. INTRODUCTION ... 195
2. PATTERNS D'ARCHITECTURE DE COOPERATION ... 195
2.1. La conception de domaine et ses résultats ... *196*
2.2. Présentation des patterns d'architecture de coopération ... *197*
2.2.1. Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche fédérée » ... 197
2.2.2. Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche à base de médiation de schémas » ... 201
2.2.3. Pattern d'architecture de coopération « Architecture d'un SIC issu de l'approche à base de médiation de contextes » ... 204
2.3. Réutilisation des patterns d'architecture de coopération ... *208*
3. PATTERNS DE CONCEPTION DE COOPERATION ET PATTERNS DE SUPPORT TECHNIQUE ... 210
3.1. Patterns de conception de coopération ... *210*
3.1.1. Introduction ... 210
3.1.2. Le système d'information coopératif ACSIS ... 212
3.1.3. Présentation des patterns de conception de coopération ... 212
3.1.3.1. Pattern de conception de coopération « Objet Descriptif des Données » ... 214
3.1.3.2. Pattern de conception de coopération « Lien sémantique » ... 215
3.1.3.3. Pattern de conception de coopération « Traduction ACSIS » ... 218
3.1.3.4. Pattern de conception de coopération « Création des liens sémantiques » ... 221
3.1.3.5. Pattern de conception de coopération « Contract-Net Protocol » ... 223
3.1.3.6. Pattern de conception de coopération « Interaction d'agents » ... 226
3.2. Patterns de support technique ... *229*
3.2.1. Pattern de support technique « Démarche de transformation de modèle » ... 230
3.2.2. Pattern de support technique « Traduction des interfaces spécifiées en OMG IDL vers Microsoft IDL » ... 231

3.3. Réutilisation des patterns de conception de coopération et des patterns de support technique ... *232*
4. CONCLUSION ... 235

BILAN & PERSPECTIVES ... **237**

1. BILAN ... 237
2. PERSPECTIVES ... 240

BIBLIOGRAPHIE ... **243**

ANNEXE A : ARRP - ATELIER DE REPRESENTATION ET DE REUTILISATION DES PATTERNS DE COOPERATION ... **265**

1. ETAT DE L'ART SUR LES OUTILS DE MISE EN ŒUVRE DE PATTERNS ... 265
2. PRESENTATION D'ARRP ... 267
2.1. Objectifs d'ARRP ... *267*
2.2. Méta-modèle de l'atelier ARRP ... *268*
2.3. Description des fonctionnalités et formulaires graphiques d'ARRP ... *269*
2.3.1. Création d'une collection de patterns ... 270
2.3.2. Création d'un formalisme de patterns ... 270
2.3.3. Création d'un pattern ... 272
2.3.4. Sélection d'un pattern ... 275
3. CONCLUSION ... 278

INDEX DES FIGURES ... **279**

INDEX DES TABLEAUX ... **281**

TABLE DES MATIÈRES ... **283**