Ecole Doctorale EDIIS - Informatique et Information pour la Société

MDE et CSCW
Groupware
Travail Coopératif capillaire

Projet présenté par : Pierre Parrend

effectué au laboratoire

ICTT

Iteraction Collaborative Téléformation Téléactivités

Sous la direction de :
Bertrand David (Professeur)
René Chalon (Maitre de Conférences)

Juin 2005
Master Recherche en Informatique de Lyon - Mastria
Université Claude Bernard, Lyon 1.

Remerciements

Je tiens à remercier Monsieur le Professeur Bertrand David, directeur du laboratoire Interaction Collaborative Téléformation Téléactivité pour m'avoir accueilli dans son laboratoire, et encadré lors de mon stage de Master Recherche.

J'adresse également mes remerciements à René Chalon, Maître de Conférence, pour avoir co-encadré le stage.

Je remercie de même tous les membres de l'équipe du laboratoire, qui m'ont conseillé et suivit pendant le stage, Olivier, Guillaume, Isabelle, ainsi que Fatah.

Table des matières

1 Introduction

1.1 Sujet

Le sujet du stage tel qu'il a été proposé est le suivant :

Le CSCW (Computer Supported Cooperative Work) ou en français Travail Coopératif Assisté par Ordinateur (TCAO) a pour but de supporter par ordinateur le travail à plusieurs, synchrone ou asynchrone, local ou distant. Le travail coopératif capillaire vise à y intégrer la mobilité des acteurs du travail coopératif. En informatique, pour pouvoir supporter une activité il faut en supporter le modèle. Un courant important de recherche appelé Model-Based Approach se préoccupe se préoccupe de cette explicitation de modèle. Un autre courant appelé Model-Driven Architecture (MDA) vise à aboutir à la construction de logiciels par une succession de Modèles.

Le but de ce travail est de faire émerger, notamment par l'étude bibliographique, des modèles de collaboration et de coopération, d'étudier les méthodes de mise en place et de proposer un formalisme pour modéliser cette coopération et aboutir à une architecture support. L'évolution des modèles de coopération est également à prendre en compte, non seulement de façon statique (d'un travail à l'autre), mais aussi de façon dynamique pendant le travail coopératif en cours pour assurer la co-évolution du modèle de coopération et du système qui supporte celui-ci. Le couplage modèle de coopération, modèle d'architecture est à prévoir pour assurer cette co-évolution.

1.2 Objectifs

Il s'agit de proposer un processus de conception de systèmes pour le CSCW, en s'appuyant sur la connaissance des besoins de ces systèmes, que nous avons identifié précédemment. Ces besoins sont constitués des scénarios d'utilisation - description des actions de l'utilisateur sur le système- et des caractéristiques de domaines - spécificités invariantes du système pour un domaine d'application donné-, qui doivent être convertis en besoins de conception.

Les besoins de conception doivent être traduits en terme de fonctionnalités de la future application. Ces fonctionnalités devront ensuite être projetés sur une architecture logicielle.

L'ensemble du processus devra, conformément aux travaux déjà réalisés, s'appuyer sur les outils de conceptions MDE (Model Driven Engineering), en particulier l'architecture MDA (Model Driven Architecture), proposée par l'OMG (Object Management Group).

Le processus de conception devra supporter les caractéristiques suivantes :

1. L'existant doit être réutilisé autant que possible,
2. Les modèles doivent pouvoir évoluer dynamiquement.

Le document présente l'étude des domaines de recherches concernées, à savoir le CSCW et MDE (Model Driven Engineering), et les liens existants entre ces deux domaines. Nous déterminerons ainsi les caractéristiques de domaines, ainsi que les besoins liés à la conception de systèmes pour le CSCW.

Nous présenterons ensuite nos contributions, qui doit permettre de faciliter l'automatisation du passage des scénarios d'utilisation à l'implémentation du système.

2 Etude du domaine

2.1 Définitions

2.1.1 MDE : Model-Driven Engineering

Le MDE (Model Driven Engineering - Conception Orientée Modèles, ou Approche Orientée Modèles) est une branche du génie logiciel qui cherche à centrer la réalisation d'une application non pas sur son implémentation, ce qui est le point de vue habituel du développement procédural et des technologies objets, mais sur sa conception.

Cette conception est représentée sous forme de modèles, issus du langage UML (Unified Modeling Language). Ces modèles offrent un point de vue abstrait sur l'application, et améliorent donc la réutilisation. Par le biais des technologies de génération de code, l'implémentation des applications est ensuite générée.

Les modèles sont conformes à la définition d'un méta-modèle, c'est à dire le langage représentant les modèles. On parle par conséquent de méta-modèlisation pour parler de l'approche proposée par le MDE.

MDA (Model Driven Architecture) est la spécification de référence mettant en oeuvre les principes MDE. Elle a été réalisée par l'OMG (Objet Magnagement Group), et formalise les concepts MDE. Nous la présenterons plus en détail dans la partie 2.3.

Le terme modèle, utilisé isolément, dans ce document, désigne les modèles au sens MDE du terme, qui devront être distingués des modèles comportementaux du framework CoCSys.

2.1.2 CSCW : Computer Supported Cooperative Work

Le CSCW (Computer Supported Cooperative Work) est un domaine de recherche dont l'objectif est de permettre, par le biais d'un support informatique, le travail conjoint de plusieurs personnes dans le cadre d'un même projet. Les applications de travail collaboratif doivent être à même de s'adapter aux différentes situations que peut rencontrer un groupe de travail. Cette flexibilité contraste avec la conception plus directive d'applications telles que les workflows, et introduit de nouveaux problèmes tant techniques que liés à l'acceptation de ces outils par les utilisateurs (voir la partie 2.4).

Le terme CSCW est traduit en français par TCAO (Travail Coopératif Assisté par Ordinateur).

2.1.3 Travail coopératif capillaire

Le travail coopératif capillaire cherche à étendre le domaine d'application du CSCW, habituellement conçu pour des ordinateurs de bureaux, aux applications mettant en œuvre la mobilité. Par exemple, une application collaborative peut être destinée à la télé-maintenance, et mettre en relation un technicien, sur site, et une personne chargée de la supervision des travaux. Une telle situation implique de nouvelles contraintes, liées à la mobilité, aux ressources limitées des supports informatiques mobiles.

L'un des objectifs du laboratoire ICTT est d'identifier et de résoudre ces contraintes, dans le cadre de systèmes basés sur les principes du CSCW.

Notre travail se situe dans ce cadre.

2.2 Le framework du laboratoire : CoCSys

Le framework théorique CoCSys - Cooperative Capillary System - a pour objectif de capitaliser les compétences du laboratoire ICTT en terme de mobilité (en particulier au niveau de la gestion des interfaces, et d'introduction de Réalité Augmentée), d'identification des scénarios d'utilisation (thèse en cours d'Olivier Delotte), et de structure d'applications cibles [David(2005)].

Le schéma 1 représente le framework CoCSys.

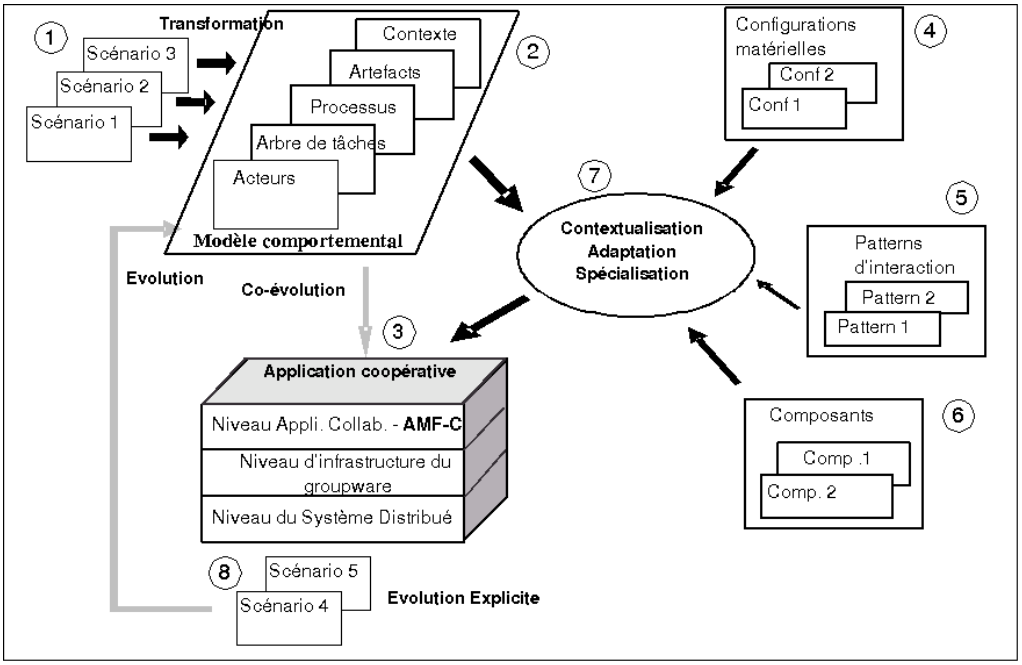

FIG. 1 – CoCSys : un framework théorique pour la conception de systèmes collaboratifs

Le Modèle comportemental permet de représenter de manière formelle les acteurs, processus, tâches, artifacts, ainsi que le contexte (2), à partir de scénarios (1) décrivant les situations collaboratives supportées par le système [Delotte(2005a)].

Ce modèle doit être transformé en un système coopératif (3), composé de trois couches logicielles : Niveau d'application collaborative, Niveau d'infrastructure de groupware, et Niveau du système distribué. Le Niveau d'application collaborative peut être réalisé à l'aide de AMF-C (Agents multi-facettes collaboratifs), un formalisme de représentation et d'implémentation d'applications complexes [Tarpin-Bernard(1997)]. Le Niveau d'infrastructure collaborative comporte les élèments invariants du système, par exemple des outils mis à disposition sous forme de composants. Le Niveau de système distribué comprend les services non spécifiques au travail collaboratif (distribution, communication, etc).

L'objectif de ce travail est d'expliciter le passage des modèles comportementaux au système coopératif (7), dans le cadre d'une approche orientée modèle, en intégrant l'existance de patterns

pré-définis, de composants, et le besoin d'évolution du système réalisé.

2.3 Model-Driven Engineering

L'approche MDE, que nous avons déjà introduite, constitue le support de notre processus de développement.

Le principe du MDE est de modéliser chaque caractéristique du futur système indépendamment, puis d'intégrer ces modèles. Les modèles principaux sont les modèles d'applications (Modèle Indépendant de Plate-forme, PIM), représentant le domaine et les interactions entre les utilisateurs et le système, d'une part, et les Modèles de Plate-forme (PM), d'autre part. Ceci permet la réutilisation des modèles, indépendamment du contexte : le modèle d'application est indépendant de la plate-forme, et peut être combiné à un autre modèle de plate-forme, pour obtenir un système fournissant les même fonctionnalités, mais adapté à une plate-forme différente, et donc une architecture, ou un langage d'implémentation, différents. La figure 2 représente ce processus.

MDA est spécifié par l'OMG, pour offrir un support à l'approche MDE
[Object Management Group(2001)]. Une présentation de MDA et de son utilisation est fournie par [Koch(2001)].

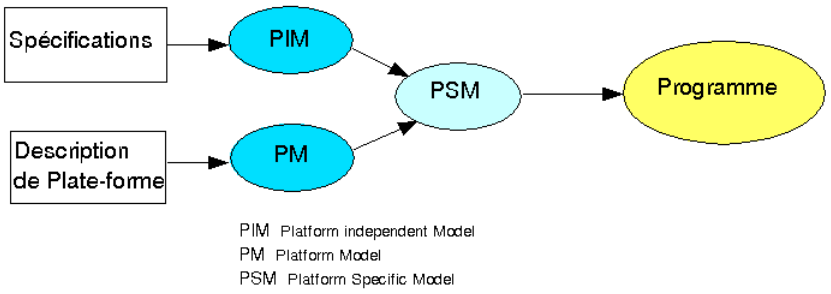

FIG. 2 – Principe des processus MDA

Le principal problème posé par la combinaison des différents modèles est la compatibilité des langages utilisés pour décrire ces modèles. Ceci a poussé l'OMG à définir une architecture en trois niveaux, notés M1, M2, M3. M1 représente les modèles, M2 les langages dans lesquels sont représentés ces modèles (méta-modèle), M3 le méta-métamodèle, c'est à dire la spécification de ce que peuvent être les langages de niveau M2. La spécification de niveau M3 est appelée MOF (Meta-Object Facility) dans le cadre de l'architecture de l'OMG. On ajoute un niveau M0, qui représente à la fois les objets du programme à l'exécution et les objets réels. La figure 3 montre les liens entre les différents niveaux.

L'approche orientée modèle permet de dépasser les limites des paradigmes objets [Bezivin(2003)]. Il ne s'agit pas de les remplacer, mais de les compléter. L'objectif est triple :

1. Séparer les descriptions fonctionnelles et l'implémentation,

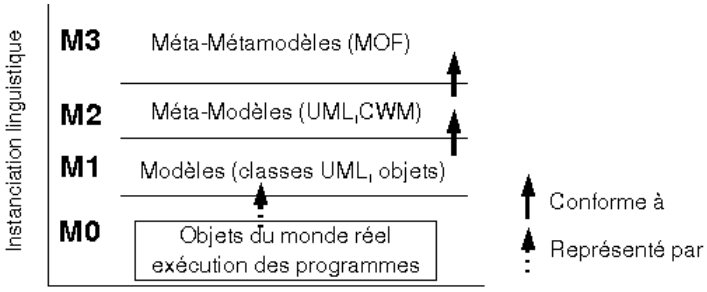

FIG. 3 – Architecture MDA en trois couches

2. Identifier et exprimer, à l'aide d'un langage spécifique de domaine, les différents aspects d'un système,
3. Permettre la transformation entre ces langages de domaine dans un framework global.

La mise en place d'une approche orientée modèle suppose l'existance d'un framework support, qui mette à disposition des outils pour réaliser la description et la transformation des modèles. De tels frameworks sont appelés Usines Logicielles.

La mise en place d'une usine logicielle nécessite un certain nombre d'outils et de langages [Greenfield(2003)]. Le coeur de l'usine logicielle est composée d'un schéma logiciel, c'est à dire d'un ensemble de langages et des transformations entre ces langages, à différents niveaux d'abstraction (conceptuel, logique, implémentation), pour les différentes parties du systèmes (Métier, Information, Application, Technologie) . Le schéma logiciel est complété par des processus et des outils de capture et d'exploitation de ces modèles, dans le but d'automatiser le traitement. C'est le patron logiel. Ce patron est enfin associé à un environnement de développement intégré, pour réaliser l'usine logicielle.

Une variante des usines logicielles sont les Méga-modèles, qui insistent plus sur la capitalisation et la réutilisation des modèles existants. Il s'agit de créer des bases de modèles, comportant des informations concernant les versions, l'existance éventuelle de définitions alternatives, l'historique, les modèles associés, les implémentations existantes [Bezivin(2004)].

Un tel environnement permet la réutilisation des modèles. Toutefois, le code produit, s'il est présenté sous une forme adaptée, peut lui aussi être réutilisé pour augmenter la productivité du développement d'applications. Ceci conduit à distinguer les modèles (appelés alors Walkers) des parties de code (appelé Producers) [Reich(2002)]. L'expression de contraintes sur les modèles permet de préciser leur portée, et de formaliser les spécifications des systèmes. C'est ce qui a donné naissance au langage d'expression de contraintes OCL.

A partir de cette étude bibliographique, nous pouvons préciser les élèments nécessaires à la réalisation d'un processus orienté modèle pour supporter la réalisation de systèmes collaboratifs :

- nécessité de proposer un cadre méthodologique permettant la réalisation d'applications nouvellles à partir de briques existantes,
- nécessité de capitaliser les modèles et les transformations,
- besoin de langages adaptés,

– besoins de mécanismes de transformations sur ces langages.
Vous pourrez trouver plus de détails sur le MDE, ainsi que des informations sur des approches moins utiles à notre problèmatique, dans [Parrend(2005d)].

2.4 Computer Supported Cooperative Work

2.4.1 Présentation

Le CSCW (Computer Supported Cooperative Work), ou en français TCAO (Traval Collaboratif Assisté par Ordinateur) désigne les 'Systèmes informatiques qui supportent des groupes de gens engagés dans une tâche (ou un but) commun, et qui fournissent une interface sur un environnement partagé' [Ellis(1991)].

Ce domaine de recherche a pour objectif d'analyser les caractéristiques des systèmes destinés à des groupes de taille moyenne, dans lesquels les rapports sociaux ont une importance capitale en vue du succès des projets menés. En particuliers, les systèmes CSCW se distinguent des workflows, même s'ils peuvent être amenés à en intégrer, par le caractère moins contraignant des processus supportés [Grudin(1994a)].
Le terme Groupware, ou Collecticiel, est utilisé pour désigner les outils basés sur ces recherches.

Trois grands types d'activités doivent être supportées, les '3 C' :
– la co-production, c'est à dire la réalisation d'un travail en commun,
– la communication, qui permet aux utilisateurs de synchroniser leur réalisation,
– la coordination, qui concerne l'organisation formelle du travail (dates butoirs, équipes, etc).
Ces trois types d'activités doivent impérativement être supportées.

Le travail collaboratif peut avoir lieu selon quatre modalités : de manière asynchrone, en session (communication et travail sur des données non partagées), en réunion (partage de certains objets et obligation de communication), de manière étroite (partage de l'ensemble des objets) [David(2001)]. Selon les applications, une ou plusieurs de ces modalités sont supportées.

La figure 4 montre l'exemple d'une application complète, qui dispose d'outils permettant le support des trois types activités, et des quatre modalités. Le même outil peut supporter diverses situations collaboratives.

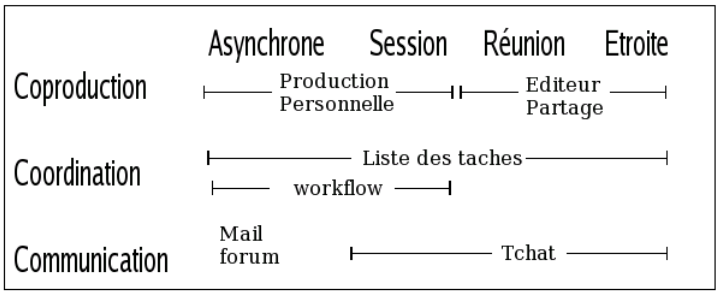

FIG. 4 – Exemple d'un système collaboratif

Les groupwares sont amenés à évoluer extrêmement rapidement. En effet, les besoins des utilisateurs en terme de collaboration évoluent, d'un projet à un autre ou au sein d'un même projet : il est indispensable de pouvoir supporter des situations non prévues. L'évolution concerne donc l'organisation du travail, d'une part - rôles, tâches, équipes - et les outils utilisés pour réaliser ce travail, d'autre part, souvent mis à disposition sous forme de composants. Il ne s'agit a priori pas de l'évolution de l'application elle-même.

De plus, l'implémentation doit pouvoir évoluer, pour garantir l'interopérabilité et l'adaptation à de nouvelles infrastructures.

On distingue deux niveaux de flexibilité : niveau macroscopique (évolution des activités, organisations) et microspique (implémentation) [Tarpin-Bernard(2000)].

Vous trouverez plus de détails dans [Parrend(2005c)].

2.4.2 Fonctionnalités

Nous avons présenté les principales caractéristiques des systèmes CSCW. Pour réaliser les modèles de domaines de notre processus MDE, il est nécessaire d'étudier les fonctionnalités des groupwares. En effet, les besoins des utilisateurs d'un groupware, pour pouvoir être traduits en modèles d'application, doivent correspondre à des services donnés. Plusieurs auteurs se sont concentrés sur cette traduction de besoins abstraits en caractéristiques fonctionnelles.

L'une des premières tentatives d'étude des fonctionnalités des systèmes collaboratifs, en vue de leur analyse, a été réalisée par le modèle Denver [Salvador(1995)]. Toutefois, il ne propose pas de lien entre les besoins des utilisateurs et les fonctionnalités requises. Il est donc insuffisant pour notre travail.

L'une des premières listes de référence, en ce qui concerne les fonctionnalités des groupwares, a été réalisée par Dewan en 1993, et mise à jour dans [Dewan(1993)]. Il s'agit d'interaction individuelle, couplage entre utilisateurs, faire-défaire, diffing (différence entre les versions d'une objet), merging (union de deux versions d'un objet), contrôle d'accès, contrôle de concurrence, support de workflow, awareness (conscience de l'autre), gestion de session.
Cette liste a été ré-organisée lors de la conférence EHCI'98, selon les '3C' dont nous avons parlé précédement[Graham(1998)].

Suite à ces travaux, des domaines plus spécifiques du CSCW ont été explorés. Nous ne pourrons pas les détailler, étant donné la place dont nous disposons. En particulier, les besoins de partage et réplication des données ont été étudiés [Tietze(2001)], ainsi que l'awareness, c'est à dire la conscience de la présence des autres utilisateurs. L'awareness doit pouvoir être contrôlée par l'utilisateur, sans quoi le risque de refus de l'application est grand [Laurillau(2002)]. L'awareness peut être supportée par l'extension de la notion de session, en distingant la collaboration réelle (session de travail partagée) et la collaboration potentielle (connaissance des utilisateurs disponibles pour partager une session) [y Solares(2005)].

Les trois thèses citées détaillent suffisament les besoins des groupwares, pour constituer une base de travail réaliste en terme de connaissance des fonctionnalités des groupwares.

Cette étude nous permet d'identifier une liste de fonctionnalités, non exhaustive, mais que nous considérons comme suffisante pour couvrir les principaux besoins des groupwares. Il est

possible d'organiser cette liste en identifiant des **domaines fonctionnels**. Un domaine fonctionnel est un ensemble de fonctions concernant un aspect particulier du travail collaboratif, sans recouvrement avec d'autres aspects (ex : coordination et gestion des données). Dans chacun de ces domaines, les caractéristiques peuvent être représentées de manière hiérachique, en fonction du lien de composition qu'elles ont : par exemple, la gestion des équipes fait partie de la coordination L'annexe A montre un exemple de cette liste de fonctionnalités, pour le domaine fonctionnel de la coproduction.

Vous trouverez plus de précisions dans [Parrend(2005a)].

2.4.3 Conception

Nous disposons d'informations relativement complètes sur les caractéristiques fonctionnelles des groupwares. Concevoir un système collaboratif à partir de ces fonctionnalités permettra de séparer les services collaboratifs réutilisables des interactions, qui dépendent fortement du besoin des utilisateurs et du contexte de l'application. Et donc d'améliorer la réutilisation de ces services, qui seront développés indépendamment d'un contexte particulier.

Cependant, les aspects techniques ne sont pas seuls primordiaux dans la réalisation d'un groupware, et les conditions de leur mise en œuvre est souvent plus importante dans le succès ou l'échec des groupwares. Ces conditions ne seront pas toutes importantes dans la mise en oeuvre d'une solution technique, mais devront être considérées si nous sommes amenés à réaliser des applications réelles. Elles permettent de mettre en évidence les besoins non techniques intervenant dans les groupwares, plus que dans d'autres types de systèmes.

[Grudin(1994b)] présente les problèmes spécifiques aux groupwares. Il s'agit en particulier du besoin d'une masse critique (le nombre d'utilisateurs doit être suffisant), de la possibilité de gérer les exceptions (tâches non connues du système), de la difficulté de l'évaluation avant déploiement d'un outil qui doit être adapté aux caractèristiques sociales et politiques de l'entreprise, et de la gestion de l'acceptabilité du système et de son appropriation.

L'appropriation est un facteur supplémentaire de succès. Elle désigne l'usage que l'utilisateur fait du système, parfois en le détournant de ses fonctions premières. L'appropriation permet donc à l'utilisateur d'exploiter au mieux les systèmes dont il dispose, en les adaptant à ses besoins propres. [Dourish(2003)] présente les conditions techniques d'une bonne appropriation par les utilisateurs :

1. Composabilité et extensibilité des informations,
2. Vues multiples sur les informations,
3. Séparation des informations et de leur structure,
4. Composabilité et extensibilité des fonctionnalités,
5. Visibilité des fonctionnalités,
6. Partage d'information liée à l'application, et non à l'infrastructure.

Les besoins des systèmes coopératifs sont donc de plusieurs types : besoins non fonctionnels, qui comprennent les besoins d'acceptabilité et les besoins système comme la sécurité, et besoins fonctionnels, qui correspondent aux services utilisés par les utilisateurs, et sont caractérisés par la

présence de telle ou telle fonctionnalités. Les besoins d'acceptabilité font partie des besoins ergonomiques, c'est à dire des besoins liés au confort d'utilisation du système.

[Laurillau(2002)] montre qu'il est souvent possible d'exprimer de manière qualitative ou quantitative ces besoins d'acceptabilité, afin de les introduire dans les spécifications du système de manière formelle. Par exemple, l'acceptabilité est conditionnée (entre autres) par la non possibilité pour un utilisateur de connaître les tâches qu'un autre réalise à un moment donné. Une telle condition peut être exprimée dans un langage de contrainte : par exemple, si les modèles (et donc les fonctionnalités) sont définis en UML, les besoins non fonctionnels pourront être exprimés dans le langage OCL (Object Contraint Language). La définition d'un profil UML spécifique peut faciliter leur spécification.

Nous avons présenté les conditions de succès des systèmes coopératifs. Toutefois, ceux-ci étant amenés à évoluer fortement, ils ne peuvent être conçu comme des applications finies, mais doivent également offrir un support à des développements ultérieurs. [Tietze(2001)] analyse les besoins d'un framework de développement pour le CSCW, qui doit se conformer aux contraintes suivantes :

1. Format générique (et donc stable dans le temps) de données,
2. Accès transparent aux données distantes dans le système distribué,
3. Distinction entre données personnelles et collectives,
4. L'awareness doit être définie au niveau applicatif, car elle est très dépendante des scénarios d'utilisation,
5. Lieu de stockage/diffusion des composants,
6. Accès aux services depuis l'extérieur du système.

Nous ajoutons, dans le cadre de notre processus de conception orientée modèle, la mise à disposition d'une usine logicielle.

Lors de la conception, ces besoins des applications coopératives peuvent être exprimés par le biais de modèles, s'il s'agit de besoins fonctionnels du système, ou par le biais de contraintes OCL, s'il s'agit de besoins non fonctionnels. La réalisation d'un Profil UML spécifique aux applications CSCW devra donc être envisagée pour rendre plus aisée l'expression tant des fonctionnalités que des contraintes. Un framework doit exister, afin de supporter l'évolution des applications.

2.4.4 Structure d'application

L'étude des contraintes de conception a permis de préciser les besoins fonctionnels et non fonctionnels des systèmes CSCW. Elle a permis également de mettre à jour quelques contraintes sur la structure de ce système : le support de données locales et distantes, l'awareness comme service de niveau applicatif, l'apport des composants.

La structure globale de notre système collaboratif dérive de celle de la plate-forme ECoop, présentée dans [Primet(2002)]. Il s'agit de la structure en trois couches présentée dans la partie 2.2. La figure 5 présente cette architecture, complétée, et les modèles qui déterminent le contenu de chacune des couches.

Nous avons vu que l'implémentation des modèles de comportement doit être séparé des fonctionnalités, afin d'améliorer la réutilisation. C'est le rôle de la distinction entre le niveau applicatif

et le niveau d'infrastructure de services. Cependant, la barrière entre niveau applicatif et niveau d'infrastructure n'est pas imperméable : en effet, les fonctions de niveau applicatif, quand elles deviennent génériques, peuvent devenir des services d'infrastructure, et donc passer dans le niveau d'infrastructure.

En plus du niveau applicatif, une couche d'IHM est présente. Il est probable que ces deux couches n'en représentent qu'une, dans la mesure où l'IHM désigne non seulement l'interface graphique, mais l'ensemble des Interactions Homme-Machine réalisées par cette interface. Toutefois, cette question dépasse le cadre de ce travail.

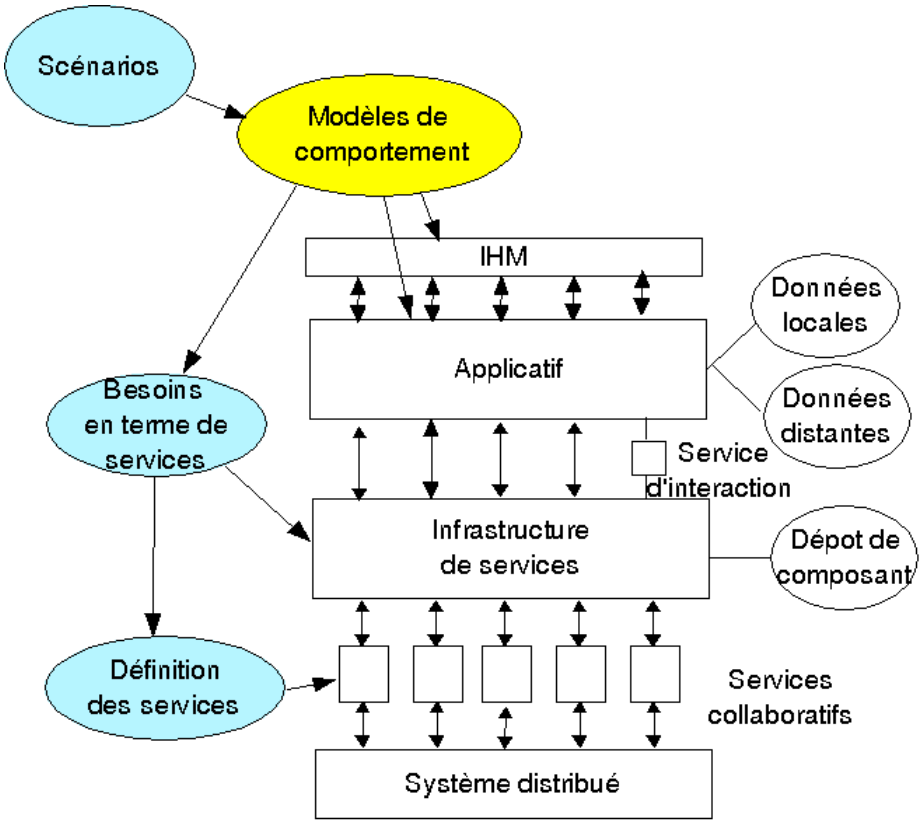

FIG. 5 – Structure d'un système collaboratif

Le niveau applicatif est l'implémentation des modèles de comportement. Il fait appel à différents services qui mettent à disposition les fonctionnalités réutilisables du système. Ces services peuvent exister sous la forme de composants, s'ils sont mûrs et non destinés à évoluer, ou sous forme de modèles de services, s'ils doivent être redéfinis lors de la conception. Cette distinction permet de réaliser des services progressivement, et de les faire évoluer jusqu'à ce qu'ils soient assez mûrs pour être distribués sous forme de composants. On a donc cohabitation dans le système entre des composants, et les modèles des services en cours d'élaboration.

Certains services, comme l'awareness, sont définis au niveau applicatif.

Les autres services sont présents au niveau infrastructure de services. En particulier, un de ces services doit mettre à disposition des composants, afin que les utilisateurs puisse charger de nouvelles fonctionnalités et ainsi étendre les capacités de leur application.

Les services nécessaires au niveau d'infrastructure de services sont déduits des scénarios d'utilisation.

D'autres éléments devront être intégrés ultérieurement, mais dépassent le cadre de ce travail.

En particulier, le support de la mobilité dans le cadre du TCAO capillaire suppose le support de distribution et réplication des données et fonctionnalités. Ceci peut être réalisé par AMF-C (Agent Multi-facettes collaboratifs) [VAISMAN(2002)].

Ensuite, l'évolution de l'application devra également être prise en compte. Nous étudierons cette question plus en détails dans la partie 2.5.

Vous trouverez les détails quant à la répartition des fonctionnalités en couches dans [Parrend(2005i)][Parrend(2005j)].

2.5 MDE et le CSCW

2.5.1 Première validation

La première validation que nous connaissons de l'approche MDE pour la réalisation de groupwares a été présentée par [Swaby(1999)]. Swaby montre que la conception orientée modèles est tout à fait pertinente, et réellement performante. En particulier, l'ajout de nouvelles fonctionnalités est grandement accélérée : dans l'exemple cité, cet ajout se fait en cinq minutes. Les avantages identifiés de la conception orientée modèles sont : le développement rapide d'applications, l'évolution d'applications existantes, l'intégration aisée de services existants, ainsi que le support d'interfaces utilisateurs variées.

2.5.2 Le niveau Méta

L'approche orientée modèle est fondée sur la conception, non plus directement du niveau d'implémentation, mais du niveau méta, c'est à dire du niveau de description de l'application. Centrer la conception sur le niveau de description permet d'isoler chaque partie du système, et d'accroître ainsi les possibilités d'évolution et de réutilisation.

L'évolution peut avoir lieu, comme nous l'avons vu dans la partie 2.4, au niveau macroscopique (système, fonctionnalités, activités supportées), et au niveau microscopique (évolution de l'implémentation). Le niveau macroscopique est décrit par des modèles, et son évolution doit être réalisée par l'évolution de ces modèles. Nous verrons que les solutions techniques pour l'évolution des modèles n'existent pas encore. Le niveau microscopique concerne le code de l'application, son évolution est réalisée par codage. L'évolution, donc, est permise par la manipulation soit de l'implémentation du système, soit de sa description.

En ce qui concerne le vocabulaire utilisé, et la distinction en niveau macroscopique et niveau microscopique, cependant, la dénomination utilisée ne nous parait pas appropriée. En effet, l'ampleur des modifications n'est pas forcément plus importante dans l'une ou l'autre approche. Il sera plus approprié de parler d'**évolution par méta-modélisation**, plutôt que de niveau macroscopique, et d'**évolution par méta-programmation**, plutôt que de niveau microscopique.

La méta-modélisation a été définie dans la partie 2.1. La méta-programmation, illustrée par la figure 6, consiste à définir des interfaces génériques de manipulation des classes, en permettant par exemple l'ajout de méthode de manière programmatique. Nous y reviendrons dans le paragraphe suivant.

On distingue donc évolution par méta-modélisation et évolution par méta-programmation. Nous verrons dans la suite de cette partie que l'évolution par méta-programmation (MP) existe, dans le cas des groupwares, et que la méta-modélisation (MM) commence à être introduite, mais de manière partielle. Ces deux approches ne sont pas antagonistes, mais complémentaires.

[Bezivin(2003)] montre qu'elles sont orthogonales, comme le montre la figure 6.

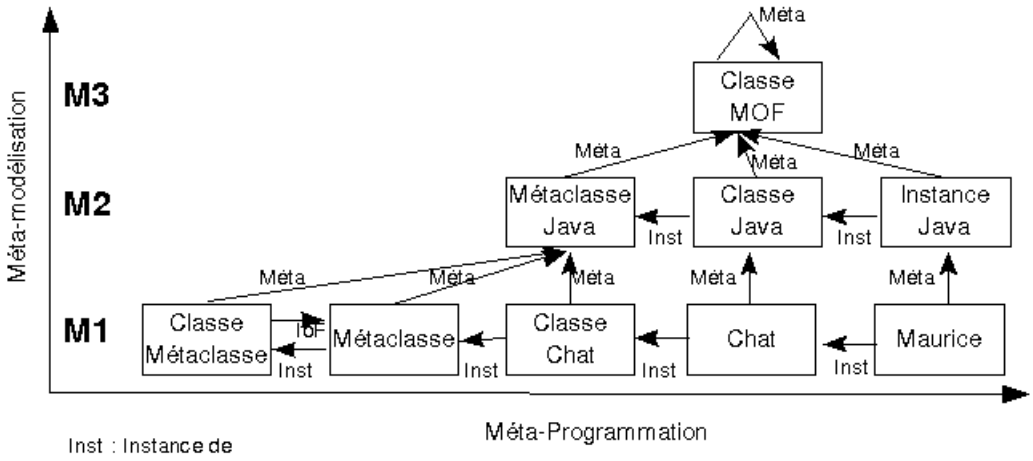

FIG. 6 – Méta-programmation et méta-modélisation

2.5.3 Méta-programmation et modélisation

Les implémentations existantes de groupwares évolutifs sont, à notre connaissance, essentiellement basées sur la méta-programmation.

Dans les travaux de [Bourguin(2000b)][Bourguin(2000a)][Bourguin(2001)], il s'agit d'un système dont les composants peuvent être manipulés, mais également modifiés. Cette modification est permise par la réflexivité des composants proposés, c'est à dire la combinaison d'introspection (possibilité de découvrir ce que sont les composants), et l'intercession (possibilité de modifier ce que sont les composants). La méta-programmation ne peut par conséquent être mise en œvre que par des langages supportant l'intercession, comme par exemple Smalltalk. La réalisation d'une telle approche avec d'autres langages, comme Java, n'est pas supportée par défaut.

Parallèlement à l'étude de l'évolutivité et de la méta-programmation, [Bourguin(2000b)] s'intéresse à la modélisation d'un groupware. Il se base sur les travaux réalisées en sociologie sur la théorie de l'activité, pour proposer un modèle spécifique pour le CSCW : il identifie plusieurs éléments invariants, sujets, communautés, objets, ainsi que des règles de travail qui définissent le lien entre ces éléments. Ces règles de travail peuvent être séparées en rôles et tâches.

Il nous semble qu'un élément fondamental du CSCW n'est pas représenté explicitement dans ce modèle : la Session. Nous proposons donc un modèle légèrement modifié, qui nous semble plus complet. Ce modèle, visible à la figure 7, peut servir de coeur d'application à une application CSCW.

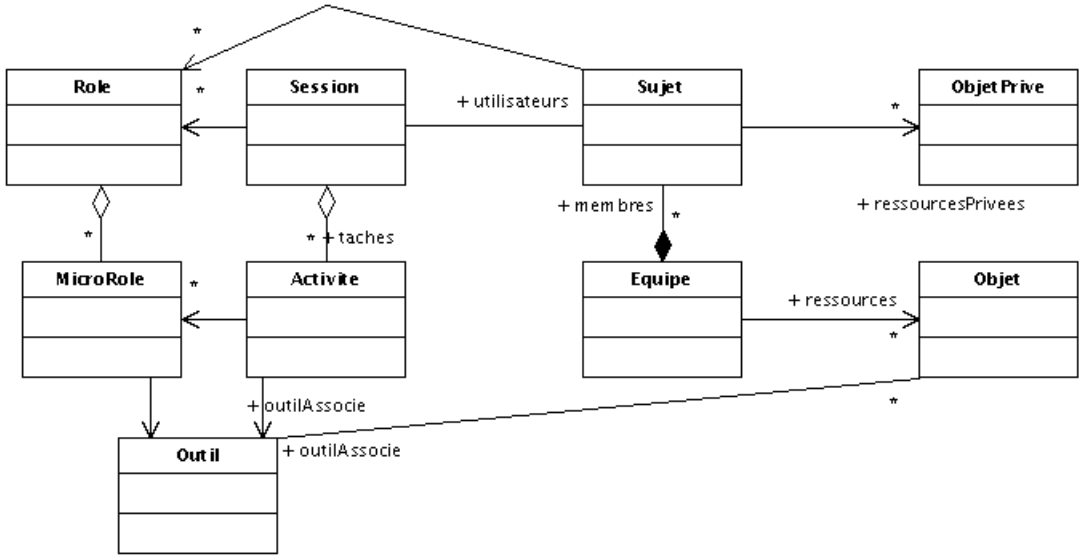

FIG. 7 – Modèle d'application collaborative

Nous avons déjà parlé du besoin de réaliser un langage facilitant la modélisation de systèmes CSCW, par exemple un profil UML. Un tel langage existe - UML-G -, mais il ne prend en compte que la distribution des données[Rubart(2002)][Rubart(2004)]. La représentation des fonctionnalités doit encore être réalisée.

2.5.4 Méta-modélisation

Les objectifs de la méta-modélisation sont de concilier la souplesse de la méta-programmation - évolution lors de l'exécution du programme, par exemple - et la puissance de la modélisation - visiblité, rapidité de conception.

Actuellement, la possibilité d'évolution dynamique, c'est à dire lors de l'exécution, existe pour les données, les workflows ([Rubart.(2001)] propose un exemple basé sur les technologies hypermédia), mais pas pour les modèles d'application eux-mêmes. De plus, les technologies de transformation de modèles, fondées sur le mapping d'un méta-modèle vers un autre, si elles permettent la traduction entre des types de modèles différents, par exemple entre Modèle Indépendant de Plate-forme (PIM) et Modèle Spécifique de Plate-forme (PSM), ne sont pas adaptées pour supporter leur évolution ponctuelle, par exemple ajouter une classe ou une opération.

Pour le moment, nous ne disposons pas de solution technique satisfaisante pour supporter l'évolution des modèles. Or cette possiblité d'évolution est la condition de l'évolution de l'application, et donc de son adaptation à l'environnement, fondamentale pour le travail coopératif

capillaire.

2.5.5 Evolution des systèmes

L'évolution des systèmes, au niveau applicatif, peut cependant être supporté par l'évolution non pas des modèles, mais de la manière dont ces modèles sont interprétés [LePallec(2002)]. Le langage d'interprétation peut alors être modifié si besoin, pour donner naissance à des implémentations particulières des modèles. Toutefois, une telle approche implique l'ensemble du modèle : aucune modification ponctuelle n'est possible. L'interprétation souple des modèles est un premier pas vers une gestion dynamique de la méta-modélisation, mais reste limitée dans la mesure où les modèles eux-mêmes sont statiques.

Les besoins mis en évidence ici sont donc d'une part la réalisation d'un langage complet de description de modèles, et d'autre part la possibilité d'évolution dynamique des modèles de l'application : un outil de manipulation et d'extension des modèles devra être réalisé.

Vous trouverez le détail de ces informations dans le document [Parrend(2005e)].

3 Contributions

3.1 Première Approche : les modèles

3.1.1 Proposition

L'étude de la bibliographie permet d'identifier les enjeux de ce travail, et en particulier de déterminer les besoins des systèmes collaboratifs en terme de conception, ainsi que le modèle de tels systèmes, qui a été présenté dans la partie 2.5. Ces besoins sont récapitulés dans le document [Parrend(2005h)]. Ils présentent le cadre dans lequel se situe notre processus de conception de systèmes collaboratifs.

3.1.2 Validation

Le prototype réalisé a pour objectif de valider le modèle d'application proposé. Les scénarios d'utilisation sont ceux mis en oeuvre par d'Olivier Delotte dans le cadre de sa thèse [Delotte(2005b)].

Le prototype permet de valider la puissance de la conception orientée modèle : l'ajout de nouveaux cas d'utilisation simples est réalisée quasi immédiatement. De plus, le modèle proposé semble approprié.

Toutefois, la taille limitée du prototype ne permet pas de tirer des conclusions sur le passage à l'échelle du modèle. Celui-ci devra en tout état de cause être complété.

3.1.3 Réutilisation de modèles

Nous disposons d'un modèle d'application collaborative. Toutefois, il ne représente que les fonctions fondamentales d'un groupware. Il pourra, dans la réalisation d'un système taille réelle, être utilement complété par les nombreux Patterns de conception, spécifiques au domaine du CSCW, existant dans la littérature.

Nous avons réalisé un catalogue de quelques uns de ces patterns [Parrend(2005b)], dans l'intention de l'exploiter dans notre processus de conception. Cependant, la capitalisation de Patterns s'avère insuffisante. En effet, il s'agit d'ensemble de modèles sans lien entre eux, qui sont des solutions à des problèmes ponctuels de conception. Ils pourront donc être intégrés à un framework, mais ne sont en aucun cas un moyen de construire ce framework. De plus, ces modèles, étant issus d'applications différentes, sont parfois redondants, voire incompatibles. Une étude fine de leur applicabilité et des conséquences de leur application doit donc être réalisée, pour préciser les conditions de leur mise en œuvre.

Cette tentative de réutilisation des modèles met en évidence le fait que les Patterns sont des outils de conception ponctuels, et nous permettent nullement la réalisation de spécifications de systèmes collaboratifs. Ils ne sont pas performants pour établir un lien entre les besoins des utilisateurs, représentés par les scénarios et le modèle comportemental, et l'implémentation de cette application.

Nous avons donc besoin d'un outil plus adapté à la traduction des besoins en un système implémenté, qui facilite la réutilisation de l'existant, en particulier en se basant sur les fonctionnalités que nous avons identifiées dans la partie 2.4.

3.2 Deuxième Approche : les ontologies

3.2.1 Principe

La liste présentée dans l'annexe A est un recensement qui vise à l'exhaustivité (même s'il ne prétend pas atteindre cet objectif) des fonctionnalités typiques des systèmes CSCW. Elle présente donc toutes les possibilités (du moins celles que nous avons identifiées) en terme de besoins fonctionnels des systèmes collaboratifs, et doit donc pouvoir représenter les services possibles du niveau d'infrastructure de notre architecture (voir partie 2.4). Elle constitue par conséquent le moyen de passer, dans le framework CoCSys, des modèles de comportement au niveau d'infrastructure de l'application collaborative.

Ces fonctionnalités sont organisées selon un arbre hiérarchique. Nous avons donc une ressemblance forte avec les ontologies, qui sont un recensement de données concernant un domaine de connaissance, précis ou générique.

Nous émettons donc l'hypothèse que l'usage des technologies d'ontologies permettra de répondre aux limitations des modèles dans la traduction des besoins vers un système implémenté, en identifiant les fonctionnalités et en les séparant de l'interaction, afin de maximiser la réutilisation possible.

Toutefois, les ontologies étant une technologie à part entière, la question de leur compatibilité avec les modèles que nous cherchons à obtenir doit être étudiée.

3.2.2 Ce qu'est une ontologie

La définition des ontologies est données par [Gruber(1993)].
"Une ontologie est une spécification explicite d'une conceptualisation. Il s'agit de représenter un domaine de connaissance dans un formalisme déclaratif. L'ensemble d'objets qui peuvent être représentés sont appelés 'univers du discours'."
Une ontologie est composée des termes et concepts d'un domaine, ainsi que des relations entre ces concepts [und Partner GmbH(2003)].

Les ontologies existent sous deux formes : les ontologies supérieures, qui visent à fournir une représentation aussi exhaustive que possible du monde réel (ex. : http ://suo.ieee.org), et les ontologies pragmatiques, qui visent à résoudre un problème précis d'un domaine donné. Notre proposition relève de ce dernier type.

De plus, on distingue les ontologies de domaines, auxquelles appartiennent les deux types précédents, représentant tous les élèments susceptibles d'exister, et les ontologies fondamentales, langage minimal permettant de décrire ces élèments [Smith(2004)]. Les méta-modèles sont proches des ontologies fondamentales.

3.2.3 Ontologie et MDE

L'utilisation conjointe des ontologies et de MDE a été proposée par plusieurs auteurs. En effet, ces deux technologies sont utilisées dans un objectif d'abstraction et de réutilisation
[und Partner GmbH(2003)].

De plus, il est établit que les ontologies représentent un sous-ensemble des modèles tels qu'ils sont définis par l'OMG dans la spécification MOF (Meta-Object facility) [Atkinson(2004)] : à tout

élément d'une ontologie correspond un élément équivalent d'un modèle, alors que certains élèments des modèles n'ont pas d'équivalent dans le domaine des ontologies (par exemple les relations entre classes, comme l'aggrégation ou la composition). La traduction des ontologies vers les modèles est donc possible, alors que l'inverse n'est pas évident.

Pour réaliser cette traduction, il est nécessaire que les concepts des deux technologies soient unifiés. Un groupe de travail a été initié sur ce thème dans le cadre de l'OMG [Djuric(2005)]. Le mapping de l'une à l'autre technologie est également étudié [Bezivin(2005)]. Les ontologies s'inscrivent dans l'architecture MDA en trois niveaux, en effectuant un simple glissement sématique : le niveau M1 contient les modèles, et les ontologies associées, et le niveau M2 contient les méta-ontologies (langages d'ontologies) et les méta-modèles (langages de modèles). Toutes les spécifications de niveau M2 sont conformes au méta-méta-modèle MOF, comme le montre la figure 8.

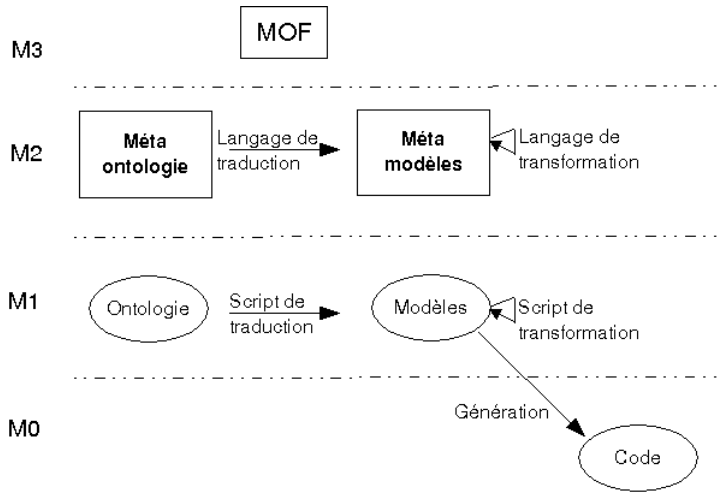

FIG. 8 – Ontologies et modèles

Les applications visées dans le cadre de ces travaux sont en particulier le Web sémantique : il s'agit d'utiliser les outils MDE pour réaliser des ontologies.

A notre connaissance l'usage d'ontologie dans un processus MDE n'a pas fait l'objet de publication. Il s'agit donc d'une proposition originale.

Vous trouverez des informations plus détaillées dans [Parrend(2005f)].

3.2.4 Intégration des ontologies dans MDE

Les ontologies sont utilisées pour modéliser des informations concernant un domaine particulier. Actuellement, leur usage est limité à la représentation des données, nous souhaitons l'étendre à la représentation de fonctionnalités d'applications. Par le passage au domaine sémantique, ceci doit faciliter la déduction des besoins fonctionnels d'un système à partir des scénarios d'utilisation. Pour chaque scénario d'utilisation, on définit un **Pattern d'utilisation**, qui représente les

fonctionnalités, et éventuellement les opérations, nécessaires à sa réalisation. Par l'intégration au processus de développement de ce système, l'automatisation du développement doit être rendue possible, ou du moins facilitée.

L'intégration des ontologies dans un processus MDE n'est pas directe, dans la mesure où ces deux technologies sont distinctes, et ont historiquement des domaines d'application différents. En effet, les ontologies traitent de connaissances, et cherchent à spécifier ce que sont les choses, c'est à dire de quel type elles sont - de quels éléments elles héritent- et quelle spécialisation de ce type elles effectuent. Par exemple, un chat (objet considéré) est un animal (type) à moustache (spécialisation). La hiérachie entre classes, héritée des applications d'intelligence artificielle, se fait donc par spécialisation, c'est à dire par héritage.

Les modèles, quant à eux, ont pour vocation de décrire les fonctions réalisées par les applications associées. Ils s'intéressent à la manière dont ces fonctions sont liées entre elles : par exemple, la gestion d'équipe fait partie de la coordination. L'extension se fait par ajout de nouvelles fonctionnalités, c'est à dire que la relation de dépendance (ou d'aggrégation) prime.

La question se pose de savoir si la représentation de fonctionnalités sous forme d'ontologie est conforme aux fondements théoriques des ontologies, malgré cette distorsion. La définition donnée plus haut, 'spécification explicite d'une conceptualisation [d'un] domaine de connaissance', nous permet d'affirmer que c'est le cas : en effet, la relation d'héritage sur laquelle sont baties les ontologies - ainsi que les outils - est contingente, et ne dépend que des domaines d'application existants.

Nous introduisons donc la distinction entre **Ontologie de données** (caractérisée par la relation d'héritage entre objets) et **Ontologie fonctionnelle** (caractérisée par la relation de composition), pour décrire ces deux familles d'ontologies. Elles appartiennent toutes deux aux ontologies de domaine. Les ontologies de données peuvent être des ontologies supérieures ou des ontologies pragmatiques. Les ontologies fonctionnelles sont des ontologies pragmatiques, car elles décrivent les fonctionnalités d'un domaine précis, mais n'ont pas de légitimité hors de l'usage pour lequel elles sont conçues, en l'occurence un processus de conception de systèmes informatiques.

Il est maintenant possible de représenter non seulement des données, mais également les fonctionnalités des applications, sous forme d'ontologie.

Les ontologies automatisent donc le lien entre les scénarios d'utilisation des systèmes, et leur réalisation. Les outils existants permettent de créer les ontologies utiles, mais manquent de souplesse dans la mesure où elles ne prennent pas en compte par défaut la possibilité de lien entre classes autre que l'héritage.

3.2.5 Proposition

Notre proposition a pour objectif de déterminer le format des ontologies fonctionnelles, et de mettre en oeuvre un processus MDE intégrant ces ontologies.

Les ontologies sont composées de différents domaines fonctionnels (voir partie 2.4). Chaque fonction est-elle même composée d'autres fonctions, comme le montre l'annexe A. Elle a alors un lien de dépendance avec ces autres fonctions. Une fonction peut également avoir besoin d'une seconde pour s'exécuter : on définit alors un lien de dépendance fonctionnelle, pour lequel la seconde fonction sera intégrée automatiquement au modèle avec la fonction qui dépend d'elle.

Une fonction peut comprendre des attributs. Une fonction peut hériter d'une autre, la fonction mère sera alors automatiquement intégrée au modèle.

Le processus proposé se compose de trois étapes, visibles à la figure 9 :

1. Etape 0 : création de l'ontologie, conformément aux connaissances sur le domaine concerné,
2. Etape 1 : sélection des fonctionnalités utiles pour l'application à partir de l'ontologie de domaine ; on obtient une ontologie simplifiée,
3. Etape 2 : traduction de l'ontologie simplifiée en modèle, manipulable par des outils MDE.

L'ontologie est réalisée préalablement, et a pour vocation à être réutilisée pour l'ensemble des applications du domaine.

L'ontologie simplifiée représente, au niveau sémantique, les fonctionnalités qui seront utilisées dans le système. La sélection a lieu manuellement (le développeur choisit les fonctionnalités dont il a besoin), ou selon des patterns d'utilisation pré-enregistrés. Il peut ainsi associer plusieurs patterns d'utilisation existants pour créer rapidement une application originale.

Les modèles obtenus peuvent être manipulés, transformés, complétés, ou servir de référence pour générer le code du programme correspondant, à l'aide des outils MDE existants.

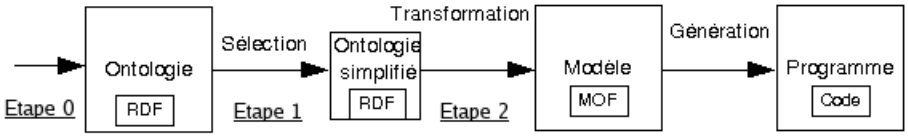

FIG. 9 – Passage des ontologies aux modèles

Les détails de cette propositions sont disponibles dans [Parrend(2005g)].

3.2.6 Processus global

Nous pouvons proposer un processus global intégrant les ontologies à la conception orientée modèle. Les informations sur le futur système sont représentées successivement sous forme d'ontologie, d'ontologie simplifiée (après la sélection), de modèle indépendant puis dépendant de l'architecture, de modèle spécifique de plate-forme, pour le système implémenté, c'est à dire de code, comme le montre la figure 10. Chaque modèle est conforme à un méta-modèle donné, c'est à dire à un langage particulier. Les modèles sont transformés ou intégrés, en précisant à chaque étape les besoins de conception.

Une proposition pour le CSCW, basée sur les outils développés au laboratoire ICTT, est présentée dans la partie 4.

3.2.7 Validation

La validation de notre proposition est réalisée à l'aide d'un prototype. Il s'agit de montrer qu'il est possible, à l'aide des outils existants, d'implémenter une application qui permette de sélectionner des fonctionnalités d'un système dans une ontologie, et de générer un modèle d'application

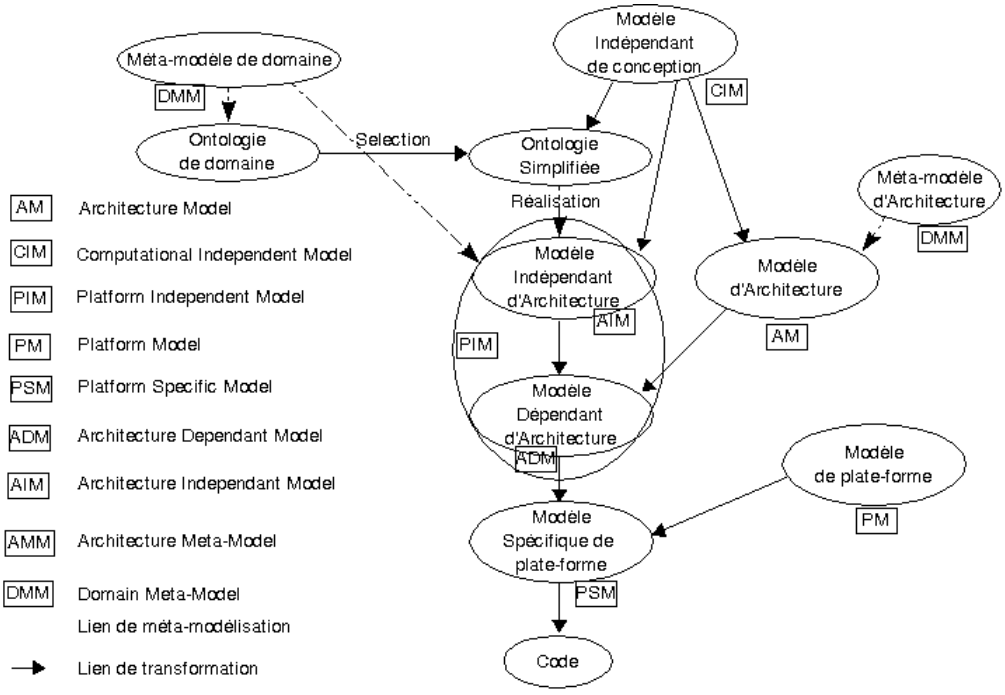

FIG. 10 – Processus de conception intégrant les ontologies

à partir de cette sélection. Nous appelerons notre outil K-MDE, pour Knewledge-based Model Driven Engineering (Conception Orientée Modèle basée sur la Connaissance).

Les systèmes réalisés ici sont des systèmes CSCW, dont l'objectif est de supporter des situations de travail collaboratif. Le processus proposé est indépendant du domaine des systèmes conçus, mais il est tout à fait adapté aux domaines d'applications pour lesquels les connaissances relatives à la conception sont nombreuses, et stables d'un système à l'autre. C'est le cas du CSCW.

Les ontologies sont réalisées à l'aide d'un outil répandu, Protégé (http ://protege.stanford.edu/). Les formats supportés sont OWL (Web Ontology Language) et RDF (Resource Description Framework).

L'implémentation de la sélection dans K-MDE utilise l'API Jena de manipulation d'ontologies, utilisée entre autres dans Protégé. Elle permet d'extraire une sous-ontologie représentant les fonctions sélectionnées, mais également les associations entre fonctions, les liens d'héritage, les attributs. Les classes dont les fonctionnalités héritent sont automatiquement sélectionnées, de même que celles auxquelles ces fonctionnalités sont liées par une 'dépendance fonctionnelle'.

La traduction d'ontologies vers des modèle est réalisée à l'aide de l'API de transformation MDE ATL (Atlas Trasformation Language). Le format de modèle supporté est Ecore, de la plateforme Eclipse, fortement inspiré du MOF (Meta-Object Facility). Le format XMI (XML Metadata Interchange), plus générique, devra bientôt être supporté, mais les outils correspondants ne sont pas encore distribués.

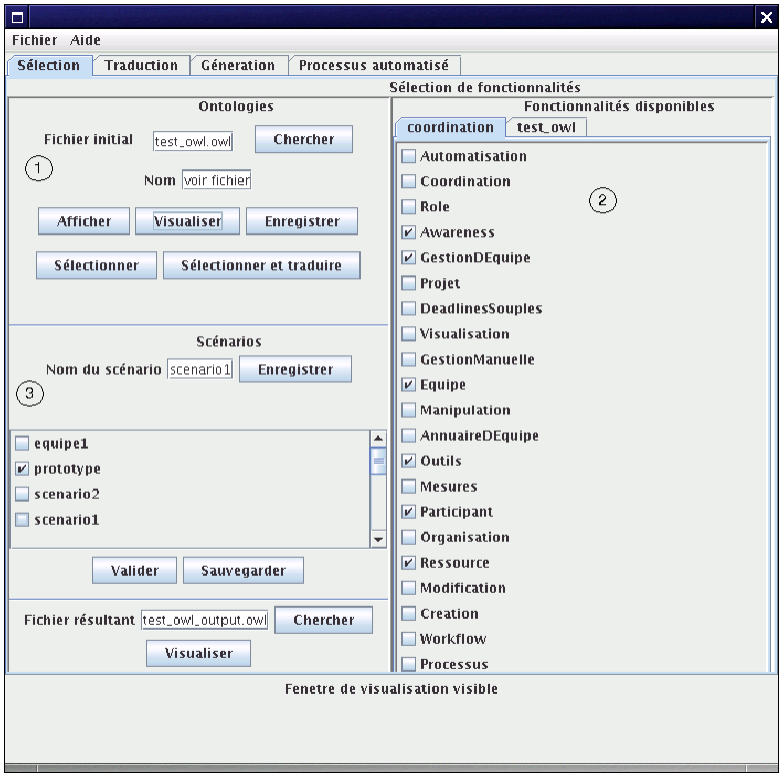

FIG. 11 – Interface graphique du prototype réalisé

Les modèles peuvent être étendus et transformés dans la plate-forme EMF (Eclipse Modeling Framework). Le code correspondant peut être généré.

La figure 11 montre l'interface graphique de notre outil K-MDE. Les onglets permettent d'accéder aux fonctions réalisant les étapes du processus proposé : sélection (étape 1, figure 9), transformation (étape 2, figure 9), génération de code.

La figure montre l'étape de sélection. Les ontologies sont chargées dans l'outil en (1), elles doivent donc être créées auparavant. Il est possible de les stocker, afin qu'elles soient disponibles lors de la prochaine utilisation de l'outil. Le fichier correspondant peut être visualisé (format texte). En (2), les fonctionnalités de chacune des ontologies disponibles sont représentées, chacune dans un onglet leur correspondant - l'exemple montre les fonctionnalités liées au domaine fonctionnel de la coordination. L'utilisateur peut les sélectionner selon ses besoins.

Il est possible de définir et d'enregistrer des patterns d'utilisation, en (3) : chaque pattern d'utilisation est constitué d'un certain nombre de fonctionnalités pré-sélectionnées. L'utilisateur qui souhaite créer un nouveau pattern sélectionne les fonctionnalités utiles dans les ontologies (2), et enregistre ce scénario sous un nom qu'il indique. Les scénarios peuvent être sauvegardés, pour être réutilisables ultérieurement. Pour sélectionner les fonctionnalités correspondant à un ou plu-

sieurs scénarios existants, il suffit de sélectionner les cases adéquates, et de valider. Les onglets représentant les ontologies sont alors mis à jour.

La sélection d'une ontologie simplifiée, contenant les fonctionnalités utiles au futur système, prend en compte les fonctionnalités indiquées en (2). L'héritage, les attributs, les associations entre les fonctionnalités sont intégrés de manière transparente. La traduction de l'ontologie simplifiée en modèle est faite à partir de cette ontologie simplifiée.

La possibilité existe, pour l'utilisateur, de spécifier les fichiers qu'il souhaite sélectionner ou traduire, si ces opérations doivent être réalisées de manière indépendante. Si, au contraire, les deux étapes doivent être exécutées immédiatement, le bouton 'Sélectionner et traduire' facilite la manipulation.

La manipulation des modèles, en particulier la génération de code, n'est pas supportée par K-MDE, les plate-formes de développement correspondantes étant suffisament nombreuses et puissantes pour le faire. Pour l'exemple présenté, EMF a été utilisé.

Afin de rendre cet outil performant, il sera nécessaire d'affiner la granularité de définition des patterns. Ceux-ci correspondant à des cas d'utilisation, il est possible de leur associer, en plus des fonctionnalités, des opérations. Il s'agit donc d'introduire du code fonctionnel, qui permet, à partir de la sélection des patterns d'utilisation souhaités, de générer une application exécutable. L'introduction de code rend nécessaire l'extension du langage de description de modèles, afin de l'intégrer dans les opérations.

3.2.8 Les ontologies dans le génie logiciel

L'utilisation des ontologies en amont des modèles dans un processus de génie logiciel permet de concentrer les efforts de réalisation d'applications non pas sur la phase de conception, mais sur la connaissance du domaine concerné. Il s'agit d'un gain d'abstraction qui nous semble comparable au gain obtenu lors du passage des objets aux modèles, c'est à dire du passage de l'implémentation d'une application, vers sa conception. Toutefois, comme nous l'avons vu, ce gain d'abstraction ne dépasse pas l'architecture en trois couches de la conception orientée modèle, mais introduit un niveau sémantique supplémentaire, comme l'illustre la figure 12.

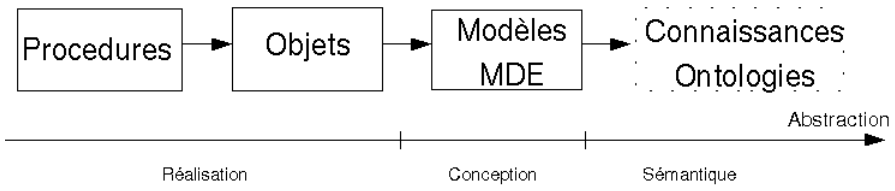

FIG. 12 – Intégration des ontologies dans le génie logiciel

Cette approche permet de dépasser les limites des modèles, qui représentent des solutions pour des fonctionnalités données. Ces modèles peuvent être intégrés dans le processus de génération,

par exemple au moment de la traduction : chaque fonctionnalité peut alors être réalisée non pas comme une classe, mais comme un ensemble de classes.

Cette approche est, à notre avis, performante dans le cas de domaine où la réutilisation des fonctionnalités est forte, comme le CSCW.

Nous pensons que la génération de modèles à partir de fonctionnalités, avec enrichissement par des modèles et du code pré-existants, peut être exploitée dans d'autres domaines, et permet de générer des applications semi-finies originales, à partir de patterns d'utilisation (et donc de scénarios) existants. L'introduction de contraintes lors de la génération de modèle doit permettre de prendre en compte des environnements particuliers : par exemple, pour une application coopérative mobile, les fonctionnalités liées à la mobilité peuvent être introduites, tout en veillant à ce que les ressources limitées des terminaux soient respectées.

4 Perspectives

4.1 Des scénarios au système

Nous avons déterminé un moyen d'établir un lien entre les scénarios et les modèles d'un système informatique, par le biais des patterns d'utilisation. Il s'agit maintenant d'intégrer cette transition dans un processus intégrant les informations dont nous disposons sur ces scénarios, et la connaissance concernant l'architecture et l'implémentation de ce système.

En amont, il est nécessaire de faire le lien entre le modèle comportemental, issus des scénarios d'utilisation, et nos patterns d'utilisation, afin de systématiser la traduction de l'un à l'autre, et permettre à terme son automatisation. Ce travail pourra être réalisé avec Olivier Delotte, dont l'un des objets d'étude est ce modèle comportemental.

En ce qui concerne notre proposition, il sera nécessaire de spécifier les différents langages utilisés.

L'architecture cible devra être prise en compte, indépendamment des fonctionnalités : le temps disponible pour ce stage n'a pas suffit pour traiter cette question de manière détaillée. Toutefois, nous proposons une première ébauche de projection sur un exemple d'architecture. Il est couramment admis que les systèmes coopératifs, s'ils veulent supporter la mobilité, doivent être hybrides : un serveur permettant de centraliser certaines données, et de contrôler en particulier la connexion des utilisateurs, et des clients lourds, supportant d'éventuelles déconnexions. Un accès web peut également être prévu. La figure 13 montre la projection de notre structure d'application vers cette architecture complexe.

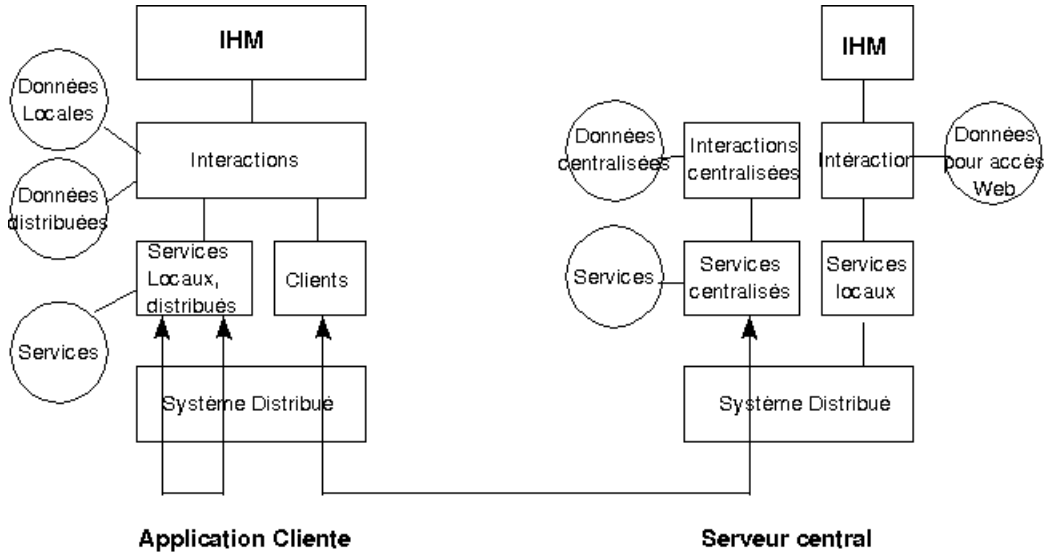

FIG. 13 – Système collaboratif : exemple de projection sur l'architecture

Le laboratoire dispose d'un outil permettant de réaliser cette projection. Il s'agit du formalisme

AMF-C (Agents multi-facettes coopératifs), que nous avons déjà évoqué, et pour lequel une application de support est développée (le moteur AMF-C). Il s'agit de scinder le système en plusieurs agents indépendants, et, à l'intérieur de chacun de ces agents, de distinguer des facettes dédiées à différentes fonctions : présentation, contrôle, abstraction, données, etc. AMF-C supporte la duplication et la distribution des facettes, ce qui est fondamental dans les systèmes mobiles.

Le moteur AMF-C permet d'intégrer les différentes facettes en une application exécutable, à partir d'un descripteur de cette application, et du code des facettes. Ce code peut être généré de manière semi-automatique par l'outil réalisé pendant ce stage, si l'on y intégre la réutilisation de code.

La question qui se pose maintenant est de savoir comment identifier les différents agents et les facettes de chacun de ces agents, à partir des modèles. Nous pensons que la formalisation du rôle des modèles comportementaux et de la projection sur une architecture permettra de résoudre ce problème.

Une autre question est celle de l'intégration des fonctionnalités (niveau d'infrastructure de services) et des interactions (niveau d'application). Cette question devra être résolue en prenant en compte le besoin d'ergonomie : en effet, nous pensons qu'une partie de l'interface graphique et des interactions peut être générée automatiquement à partir des modèles comportementaux et des patterns spécifiques au domaine IHM (Interface Homme-Machine), mais cette génération automatique est insuffisante pour créer une interface graphique ergonomique, et donc utilisable dans des conditions satisfaisantes.

La figure 14 montre les différentes étapes de ce processus de conception, telles que nous les avons identifiées.

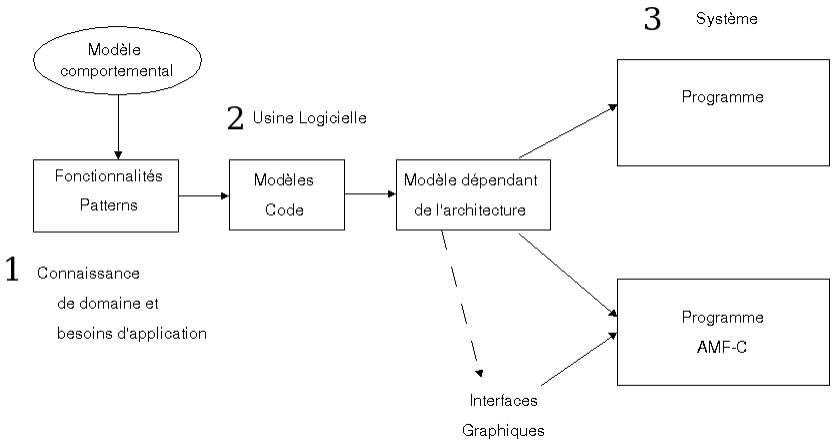

FIG. 14 – Vue du processus global de conception

4.2 Travail à réaliser

Chaque étape du processus présenté ci-dessus devra être détaillée.

En ce qui concerne notre proposition basée sur les ontologies, il sera nécessaire de spécifier les langages utilisés.

En ce qui concerne la connaissance de domaine, l'ontologie proposée devra être complétée, et validée par la réalisation d'applications réelles.

Les langages de description devront être spécifiés : le langage d'ontologie, en particulier, pour lequel nous avons proposé plusieurs éléments (voir partie 3.2).

Le langage de modèles, nous l'avons vu, doit être étendu pour supporter l'intégration de code existant (pour l'implémentation des opérations). La création d'un profil UML adapté pourra être envisagée.

Lors de la réalisation de l'ontologie, nous avons identifié un certain nombre de contraintes. Celles-ci peuvent être explicitées dans le langage OCL (Object Constraint Language) au niveau des modèles, mais nous ne disposons pas d'outils équivalents au niveau ontologique. L'équivalence entre OCL et les langages de prédicats utilisés pour les ontologies devra être étudiée, et un mécanisme de traduction proposé. L'ontologie pourra être complétée par cet ensemble de contraintes. La traçabilité des contraintes au travers des ontologies et des différents modèles pourra être étudiée.

Deux domaines d'applications pourront être distingués, et les langages séparés en deux parties : une partie générique, réutilisable pour l'ensemble des systèmes, et une partie spécifique au CSCW. Ceci doit permettre une réutilisation maximale de ce processus.

5 Carte de domaines

Le schéma 15 représente les domaines étudiés lors du stage. Les domaines vus dans le cadre de l'étude de domaine sont en traits pleins, les domaines concernant le travail personnel sont en pointillés. Les liens entre les domaines représentent l'existence de problèmatiques communes.

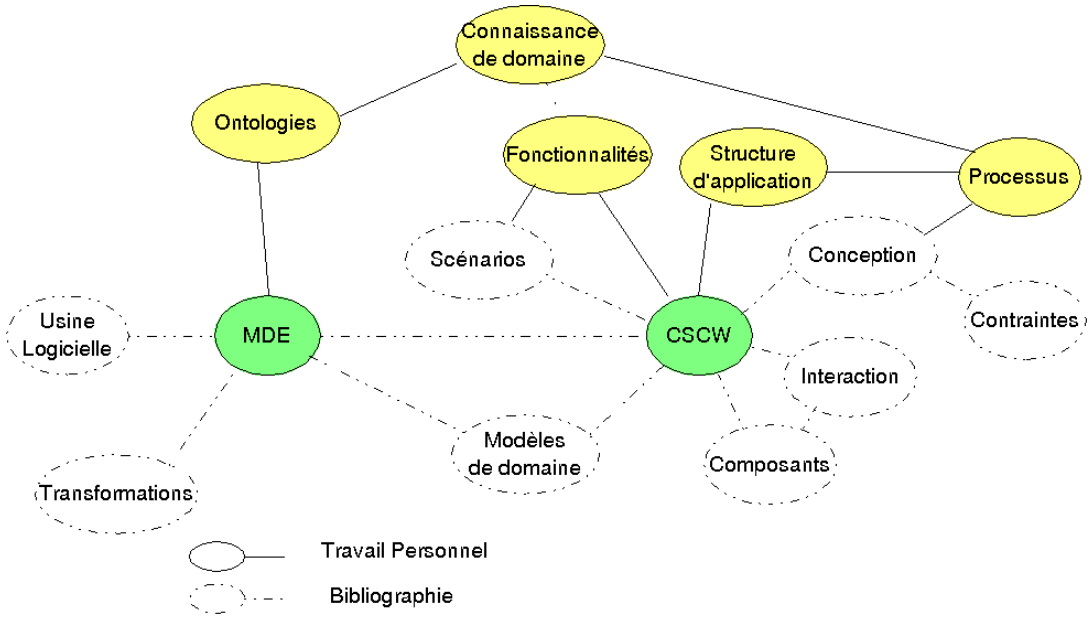

FIG. 15 – Domaines de recherche abordés lors du stage

6 Bilan

Mon travail s'inscrit dans le cadre des recherches du laboratoire ICTT, en particulier le framework théorique CoCSys. Il s'agit d'une méthodologie de conception d'applications collaboratives.

Le travail réalisé comprend, comme première étape, l'étude des domaines de recherche, à partir de la bibliographie, et l'intégration de ces informations dans le framework du laboratoire. Les domaines concernés sont le CSCW (Computer Supported Cooperative Work), et MDE (Model-driven-Engineering).

La deuxième étape est la proposition, que nous pensons originale, d'une extension du processus MDE existant, afin de palier aux limites intrinsèques des modèles, et permettre la capitalisation des connaissances du domaine du CSCW. Ma proposition est basée sur l'introduction d'ontologies dans le processus MDE, et permet d'effectuer la transition entre scénarios d'utilisation et modèles.

Les travaux existants sur le lien entre ces deux domaines offrent une validation théorique de notre proposition, mais se concentrent sur d'autres applications (web sémantique par exemple).

La troisième partie de ce travail consiste en l'étude des perspectives, ainsi que de l'intégration au framework et aux outils du laboratoire. L'utilisation conjointe avec le formalisme AMF-C (Agents multi-facettes Collaboratifs) permet la conception, puis la réalisation, d'applications collaboratives. Les langages permettant d'exprimer les connaissances de domaines, leur transformation vers les modèles, et la prise en compte formelle du modèle comportemental doivent par ailleurs être spécifiés.

Le travail sur l'utilisation des ontologies dans un processus orienté modèle pour les applications collaboratives pourra être poursuivit dans le cadre de groupes de travail actuellement en création : des groupes existent sur l'utilisation des ontologies pour la gestion du contexte des applications collaboratives, et des partenariats entre équipes scientifiques ont été initiés récemment en ce qui concerne les ontologies dans les processus MDE. Il s'agit donc d'un domaine de recherche en fort développement.

Références

[Atkinson(2004)] Colin Atkinson. Unifying mda and knowledge representation technologies. In *EDOC'2004*, 2004.

[Bezivin(2005)] Devedzic V Djuric D Favreau J Gasevic D Jouault F Bezivin, J. An m3-neutral infrastructure for bridging model engineering and ontology engineering. In *INTEROP-ESA*, 2005.

[Bezivin(2003)] Jean Bezivin. On the unification power of models. In *UML*, 2003.

[Bezivin(2004)] Jouault F. Valduriez P. Bezivin, J. On the need for megamodels preliminary draft. In *OOPSLA*, 2004.

[Bourguin(2000a)] Derycke A. Bourguin, G. Meta groupware design for cscl environments. In *ED-MEDIA'2000*. AACE (Association for Advance Computing in Education), 7 2000a.

[Bourguin(2000b)] Grégory Bourguin. *Un support informatique à l'activité coopérative fondé sur la théorie de l'activité : le projet DARE*. PhD thesis, Université des Sciences et Technologies de Lille, 2000b.

[Bourguin(2001)] Le Pallec X. Bourguin, G. Supporting human activities - the meta-level issue. In *ICEIS 2001*, 7 2001.

[David(2001)] Bertrand David. *Les télé-applications, IHM pour les Collecticiels*, page p 169 à 206. RSR-CP, 2001.

[David(2005)] Chalon R. Delotte O. David, B. Model-driven engineering of cooperative systems. In *HCI-International*, 07 2005. A paraitre.

[Delotte(2005a)] David D. Delotte, O. From scenarios to behaviour model for mobile collaborative systems. In *HCI International 05*, 2005a.

[Delotte(2005b)] Olivier Delotte. Des scénarios au modèle comportemental, 2005b.

[Dewan(1993)] Riedl J. Dewan, P. Toward computer-supported concurrent software engineering. *IEEE Computer*, 26(1) :17, 1993.

[Djuric(2005)] Gazevic D Devedzic V. Djuric, D. Ontology modeling and mda. *JOURNAL OF OBJECT TECHNOLOGY*, (1), 01 2005.

[Dourish(2003)] Paul Dourish. The appropriation of interactive technologies : Some lessons from placeless documents. *Computer Supported Cooperative Work : Special Issue on Evolving Use of Groupware*, pages 465–492, 2003.

[Ellis(1991)] Gibbs S.J. Rein G.L. Ellis, C.A. Groupware : some issues and experiences. *Communication of the ACM*, 34(1), 1 1991.

[Graham(1998)] Grundy J. Graham, N. External requirements of groupware development tools. In *EHCI*, 1998.

[Greenfield(2003)] Short K. Greenfield, J. Software factories assembling applications with patterns, models, frameworks and tools. In *OOPSLA*, 2003.

[Gruber(1993)] T. Gruber. A translation approach to portable ontology specifications. Technical Report KSL 92-71, Knowledge Systems Laboratory, Stanford University, 1993.

[Grudin(1994a)] Jonathan Grudin. Computer-supported cooperative work : History and focus. *IEEE Computer*, 5 1994a.

[Grudin(1994b)] Jonathan Grudin. Groupware and social dynamics : eight challenges for developpers. In *Conference ACM*, 1994b.

[Koch(2001)] Uhl A. Weise D. Koch, T. Model driven architecture, 11 2001. Interactive Objects Software GmbH (http ://www.io-software.com).

[Laurillau(2002)] Yann Laurillau. *Conception et réalisation logicielles pour les collecticiels centrés sur l'activité de groupe : le modèle et la plate-forme Clover*. Thèse de doctorat, Université Grenoble I, 09 2002.

[LePallec(2002)] Xavier LePallec. *Des Services d'adaptation de modèles pour la coopération de méta-sysèmes*. Thèse de doctorat, Université de Lille 1, 12 2002.

[Object Management Group(2001)] Architecture Board ORMSC Object Management Group. Model driven architecture, 07 2001. Document number ormsc/2001-07-01 (http ://www.omg.org/cgi-bin/doc ?ormsc/2001-07-01).

[Parrend(2005a)] P. Parrend. Caractéristiques fonctionnelles des groupwares, 2005a.

[Parrend(2005b)] P. Parrend. Catalogue de patterns pour le cscw, 2005b.

[Parrend(2005c)] P. Parrend. Cscw : une bibliographie, 2005c.

[Parrend(2005d)] P. Parrend. Mda : une bibliographie, 2005d.

[Parrend(2005e)] P. Parrend. Mde et cscw : une bibliographie, 2005e.

[Parrend(2005f)] P. Parrend. Mde et ontologies : une bibliographie, 2005f.

[Parrend(2005g)] P. Parrend. Précisions sur les ontologies, 2005g.

[Parrend(2005h)] P. Parrend. Réalisation d'un prototype, validation des méta-modèles, 2005h.

[Parrend(2005i)] P. Parrend. Structure en couches d'un groupware 1, 2005i.

[Parrend(2005j)] P. Parrend. Structure en couches d'un groupware 2, 2005j.

[Primet(2002)] Pascale Primet. Contribution au support réseau des applications réparties, qualité de service : pour un réseau sensible aux flux, 04 2002. Document d'Habilitation à Diriger des Recherches, ENS Lyon - UCB Lyon.

[Reich(2002)] Georges Reich. Towards a synthetic approach to the compilation of uml models (position paper), a practitioner viewpoint. In *OOPSLA*, 2002.

[Rubart(2002)] Dawabi P. Rubart, J. Towards uml-g : a uml-profile for modeling groupware. In *Groupware : Design, Implementation and Use, 8th International Workshop, CRIWG 2002*, pages 93–113, 2002.

[Rubart(2004)] Dawabi P. Rubart, J. Shared data modeling with uml-g. *International Journal of Computer Applications in Technology*, (3/4) :231–243, 2004.

[Rubart.(2001)] Haake J.M. Tietze D.A. Wang W. Rubart., J. Organizing shared enterprise workspaces using component-based cooperative hypermedia. In *HT'01*. ACM, 2001. Aarhus, Denmark.

[Salvador(1995)] Scholtz J Larson J. Salvador, T. The denver model for groupware design. In *Workshop on Designing and Evaluating Groupware, CHI'95*, 1995.

[Smith(2004)] Barry Smith. *Blackwell Guide to the Philosophy of Computing and Information, Ontology*. Blackwell, 2004. sous la direction de Luciano Floridi.

[Swaby(1999)] Dew P.M. Kearney P.J. Swaby, M.A. Model-based construction of collaborative systems. *BT Technol Journal*, (4), 10 1999.

[Tarpin-Bernard(1997)] Frank Tarpin-Bernard. *Travail coopératif synchrone assisté par ordinateur : Approche AMF-C,*. Thèse de doctorat, Ecole Centrale de Lyon, 07 1997.

[Tarpin-Bernard(2000)] Frank Tarpin-Bernard. *Ecole thematique Documents et evolution du GDR I3, T. 2, La flexibilité dans les collecticiels*. Cépaduès, 2000.

[Tietze(2001)] Daniel A. Tietze. *A Framework for Developing Component-based Co-operative Applications*. Thèse de doctorat, Technische Universität Darmstadt, 2 2001.

[und Partner GmbH(2003)] Jenz und Partner GmbH. Ontology faq, 2003. http ://www.bpiresearch.com/FAQ_Ontology.pdf.

[VAISMAN(2002)] Gérald VAISMAN. Tcao capillaire : apports du modèle amf-c. Rapport de master, Ecole Centrale de Lyon, 2002. Laboratoire ICTT.

[y Solares(2005)] Alberto Leopoldo Moràn y Solares. *Support pour la collaboration potentielle et réelle dans la conception de collecticiels répartis*. Thèse de doctorat, Institut National Polytechnique de Grenoble, 01 2005.

Les documents réalisés dans le cadre de ce stage sont disponibles à l'adresse suivante :

http ://www.rzo.free.fr/master.html.

Annexes

A Domaines fonctionnels

Les domaines fonctionnels identifiés sont les '3C' : coproduction, coordination, communication, ainsi que l'awareness, la configuration et les caractéristiques systèmes. Voici l'exemple (figure 16) des fonctionnalités identifiées pour la coproduction.

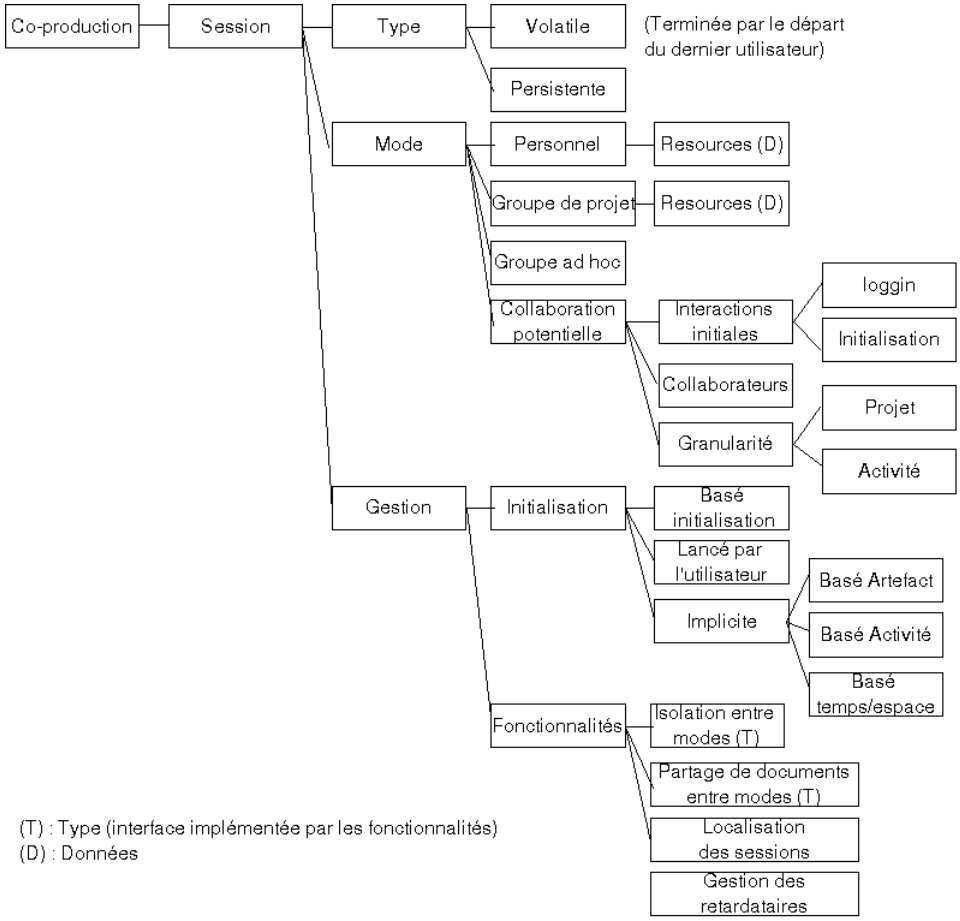

FIG. 16 – Ontologie fonctionnelles pour la coproduction

B Elements d'un langage pour le CSCW

Les éléments dont nous disposons pour la définition d'un langage de domaine pour le CSCW sont un méta-modèle et des contraintes sur les fonctionnalités.

B.1 Métamodèle

Les fonctionnalités identifiées sont conformes au méta-modèle suivant (figure 17) :

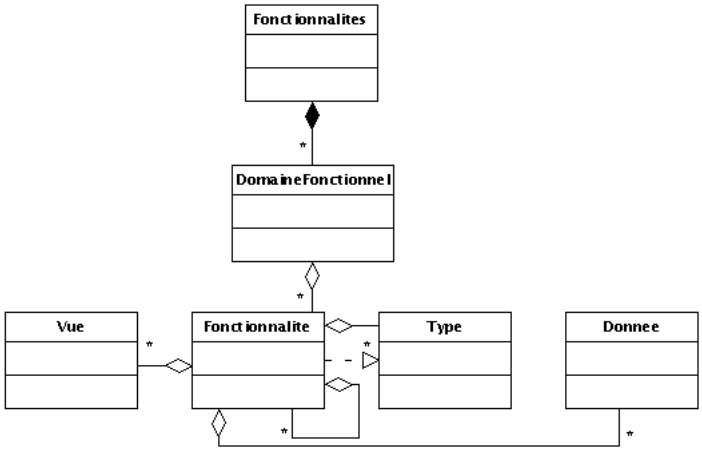

FIG. 17 – Métamodèle pour les fonctionnalités

B.2 Contraintes

Un certain nombre de contraintes ont été identifiées. Il ne s'agit pour le moment que d'indications ponctuelles sur les relations entre les fonctionnalités, qui devront être respectées. Cette liste doit naturellement être complétée.

Granularité : si elle est réalisée au niveau document, et si des annotations existent, celles-ci devront être externes au document ;

Session : Si chaque utilisateur dispose d'une session par défaut, celle-ci devra proposer deux modes : travail personnel (hors de toute collaboration), et collaboration potentielle ;

Awareness : elle est composée de l'historisation ET du support de la transparence des actions (dans le respect des codes sociaux) ;

Awareness : On distingue deux modules de gestion de l'awareness, le contrôle et l'observation. Le contrôle, géré par l'utilisateur, a pour rôle d'émettre les informations ; l'observabilité a pour rôle de collecter ces informations et de les afficher ;

Acceptabilité : le lien entre les utilisateurs et les actions en cours ne doit pas pouvoir être fait.

Ces contraintes sont des exemples de besoins d'acceptabilité, exprimé formellement. Leur mise en oeuvre permet leur respect, au niveau sémantique.

C Documents réalisés

Durant le stage, un certain nombre de documents ont été réalisés. Il s'agit d'études bibliographiques, et de notes concernant la problèmatique du stage et les réponses à y apporter.

1. Démarche globale
2. MDA : Une Bibliographie
3. Développement pour le CSCW : Proposition de Méthodologie
4. CSCW et MDA : une Problématique
5. Structure des différentes couches d'un groupware
6. CSCW : Une Bibliographie
7. Caractéristiques fonctionnelles des groupwares
8. Structure des différentes couches d'un groupware (2)
9. Validation de la structure proposée
10. Prototype de validation du modèle proposé
11. CSCW : le niveau infrastructure de services
12. Comparaison avec les autres approches MDA pour le CSCW
13. MDA et Ontologie : une bibliographie
14. Réalisation d'un prototype : validation des méta-modèles
15. Les ontologies dans un processus MDE

Annexes

1. MDA : Une Introduction
2. MDA : Un Tutoriel
3. Catalogue de Pattern pour le CSCW
4. UML vers Java annoté

Ces documents sont accessibles sur Internet à l'adresse suivante :

http ://www.rzo.free.fr/master.html.