Adaptation Contextuelle et Personnalisée de l'Information de Conscience de Groupe au sein des Systèmes d'Information Coopératifs

Manuele Kirsch Pinheiro

UNIVERSITÉ JOSEPH FOURIER – GRENOBLE 1

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER – GRENOBLE 1

Spécialité : Informatique

préparée au laboratoire Logiciels, Systèmes, Réseaux

dans le cadre de l'École Doctorale
Mathématiques, Informatique, Sciences et Technologie de l'Information

présentée et soutenue publiquement

par

Manuele KIRSCH PINHEIRO

le 29 septembre 2006

Adaptation Contextuelle et Personnalisée de l'Information de Conscience de Groupe au sein des Systèmes d'Information Coopératifs

Directeur de Thèse : Hervé Martin

JURY

M. Khalid Benali,	Rapporteur
M. Bruno Defude,	Rapporteur
M. Hervé Martin,	Directeur de Thèse
M. Jérôme Gensel,	Co-encadrant
M. Dominique Decouchant,	Examinateur
M. José Valdeni de Lima,	Examinateur
M. Jacques Mossière,	Examinateur

La différence entre un rêve et la réalité peut être,
tout simplement,
notre envie de le rendre réel...

Je dédie cette thèse ...
à ma mère, qui m'a appris à croire...
à mon mari, qui m'a appris à aimer...
... au pouvoir des rêves...

Remerciements

C'est intéressant de voir qu'une thèse est, malgré tout, une œuvre collective. La mienne est l'aboutissement d'un long chemin, la réalisation d'un rêve, et le début d'un autre chemin, d'un autre rêve... Dans ce chemin, j'ai été aidée et suivie par plusieurs personnes, auxquelles je dois mes plus sincères remerciements. Voici quelques-unes de ces personnes auxquelles je dois la réalisation de mon rêve…

Tout d'abord, je voudrais remercier les rapporteurs de ma thèse, M. Khalid Benali et M. Bruno Defude, qui ont accepté d'être les rapporteurs de ce travail, et ceci pendant même les vacances. Merci beaucoup...

Je voulais également remercier les examinateurs de ce travail, M. Jacques Mossière, M. José Valdeni de Lima et M. Dominique Decouchant. J'ai une pensée toute particulière pour Valdeni, qui a accepté de venir d'aussi loin pour m'accompagner dans cette nouvelle étape de mon parcours. Merci, professeur... J'ai aussi un merci particulier à Dominique, qui m'a beaucoup appris non seulement sur TCAO, mais aussi sur la culture de ce beau pays qui est la France.

Par ailleurs, ce travail ne serait jamais arrivé à son terme s'il n'y avait pas eu Hervé et Jérôme, mes encadrants, à qui je dois énormément. J'ose imaginer que, grâce à vous deux, j'ai pu grandir, professionnellement et personnellement. Merci... J'ai aussi un merci spécial à Marlène, qui a beaucoup participé à ce travail. Sans mes trois encadrants, ce travail n'aurait jamais vu le jour. Merci...

Et quoi dire de mes collègues (Aurélie, Céline, Marius, Bogdan, Tati, Angela et tous les autres)? On était tous dans ce voyage. Aujourd'hui, c'est mon tour, le vôtre viendra bientôt… Merci de m'avoir accompagné, ça a été un énorme plaisir de vous avoir à mes côtes...

Je voudrais également remercier mes amis brésiliens (Leonardo, Marina, Andreia, Luciano, Windson et Karol, sans oublier Ninah), qui m'ont soutenu énormément dans cette dernière année... Vous êtes les meilleurs amis qu'on puisse rêver... Merci... Et un énorme merci aussi au geek le plus brésilien de Grenoble, Yan : merci de m'avoir soutenu, d'avoir pris le temps de réviser mon texte, et merci surtout de ton amitié…

Enfin, je ne serais jamais arrivée si loin si je n'avais pas ma famille derrière moi : ma mère, Sandra, ma sœur, Damaris, mon frère et sa famille qui s'agrandit, Ramiro, Mari, Luiza, Bianca et Carolina (que j'ai hâte de connaître personnellement), sans oublier Andrea et tous les amis qui m'ont soutenu de loin. Merci, merci d'être là, merci d'avoir cru, même quand je ne croyais plus...

Et, comment parler de ma famille, sans parler de mon grand amour ? Un merci tout particulier à mon mari, Angelo, mon Angelo... Merci de m'avoir aidé, merci de ton amour, de ton soutien... Ces dernières années n'ont pas été simples, mais on est là, on est toujours là, et on sera toujours là, mon amour...

Enfin, merci à toutes les personnes qui ont rendu possible cette thèse, ce rêve qui devient aujourd'hui réel...

Table des Matières

Table des Illustrations

1 Introduction

1.1 Problématique

Le présent travail de thèse se situe à l'intersection des deux thématiques de recherche : le Travail Coopératif Assisté par Ordinateur (TCAO)[1] et l'Informatique Sensible au Contexte. Notre point de départ est l'utilisation des nouvelles technologies mobiles pour le travail coopératif. Aujourd'hui, nous ne sommes plus assignés à nos ordinateurs de bureau du fait de la disponibilité des réseaux sans fil et d'une grande variété de dispositifs mobiles. Étant donné que nous voulons ou devons coopérer dans plusieurs scénarios différents, il est important de considérer le potentiel de ces nouvelles technologies mobiles [Guerrero 2004]. Selon Renevier [Renevier 2004], les activités de groupe en situation mobile font partie de notre vie quotidienne. Pour cet auteur, l'émergence de nouveaux systèmes coopératifs sur supports mobiles implique de comprendre (et d'intégrer) les situations d'activités coopératives à distance, puisque ces situations constituent un débouché potentiellement important pour les technologies mobiles. Les nouvelles technologies deviennent des outils de travail, offrant une opportunité pour le travail coopératif [Alarcón 2004] : l'opportunité de pouvoir continuer le travail malgré le nomadisme des participants. Ceci renforce l'intérêt porté à la conception de nouveaux collecticiels sur support mobile (voir, par exemple, [Guerrero 2004], [Hibino 2002], [Muñoz 2003]). Nous pouvons désormais parler de conception de collecticiel[2] sensible au contexte, puisque ces systèmes sont confrontés aux mêmes problèmes liés notamment aux nouvelles technologies.

De manière générale, l'utilisation des nouvelles technologies et la mobilité qu'elles supportent ne semblent pas introduire de nouveaux problèmes, elles ne font qu'exacerber des difficultés existantes. Par exemple, la question de la déconnexion réseau existait avant les réseaux sans fil, surtout lorsque le matériel était moins fiable qu'aujourd'hui. Également, les ordinateurs de poche ont des processeurs plus rapides et plus de mémoire vive que les premiers ordinateurs familiaux... Le changement essentiel est que, de nos jours, les utilisateurs possèdent des habitudes fortes avec les situations fixes, qui ont des capacités nettement supérieures [Renevier 2004]. Les utilisateurs comparent les performances des dispositifs mobiles et réseaux sans fil avec celles des dispositifs fixes (ordinateur de bureau) et réseaux filaires auxquels ils sont habitués. De cette comparaison est né le sentiment d'avoir en sa possession un dispositif ou un réseau aux capacités limités.

Or, une étude de Sarker et Weels [Sarker 2003] rapporte que les utilisateurs sont plutôt compréhensifs vis-à-vis des limitations des dispositifs, mais qu'ils sont gênés par les imperfections et les problèmes de l'interface logique des dispositifs. Ce ne sont donc pas exactement les contraintes physiques, telles que la taille de l'écran, qui posent des problèmes dans l'utilisation des dispositifs mobiles, mais plutôt la non adaptation de l'interface (et du contenu) à ces contraintes. Si l'interface et le contenu ne sont pas adaptés au dispositif, les utilisateurs ont tendance à abandonner le dispositif pour un autre plus adapté. Par exemple, Hibino et Mockus [Hibino 2002] ont observé que la majorité des ordinateurs de poche révèlent des difficultés ergonomiques lors de l'écriture de longs messages. En conséquence, les utilisateurs ont tendance à écourter les messages ou encore à changer de dispositif. On peut citer, par exemple, le SMS, dans lequel on observe souvent une dégradation du langage,

[1] En anglais, « *Computer Supported Cooperative Work* » (CSCW).
[2] En anglais, « *Groupware Systems* ». Synonyme : *système d'information coopératif*

à travers l'utilisation d'abréviations. Si on généralise ce comportement, on peut imaginer que si l'effort pour accomplir une tâche sur un dispositif devient trop important, l'utilisateur préférera changer de dispositif. Ceci rejoint l'étude de Tang *et al.* [Tang 2001] qui mettent en évidence une utilisation concurrente des dispositifs.

Nous avons donc un besoin clair d'adaptation dans tout système accessible à travers les nouvelles technologies. L'adaptation dont on parle ici ne concerne pas que l'adaptation de l'interface aux contraintes des nouvelles technologies mobiles. Cette adaptation concerne aussi l'adaptation du contenu au contexte d'utilisation. Ce besoin d'adaptation est particulièrement ressenti par les collecticiels sur le Web. La majorité des dispositifs mobiles (ordinateurs de poche, téléphones cellulaires, etc.) disposent déjà d'un accès au Web. Par conséquent, nous pouvons considérer que tous les systèmes sur le Web sont désormais accessibles aux utilisateurs nomades à travers ces technologies. Les collecticiels sur le Web ne sont pas une exception. Il existe donc une urgence, pour les collecticiels sur le Web, à traiter dès maintenant les problèmes auxquels sont confrontés les utilisateurs nomades.

La situation de mobilité de l'utilisateur implique donc des contextes d'interaction par définition variables (dans le train, dans la rue, etc.) [Canals 2002], qui accentue certains aspects qui caractérisent les collecticiels, dont la question de la conscience de groupe. Le nomadisme favorise la perte de contact entre les participants et réduit chez eux le sentiment d'appartenir à un groupe. Il faut donc que les collecticiels compensent ce phénomène par des mécanismes de conscience de groupe adaptés. Par ailleurs, les contraintes liées aux nouvelles technologies et les variations dans le contexte d'utilisation qui caractérisent les utilisateurs nomades (ces utilisateurs peuvent se servir de ces technologies à partir de différents contextes) ont un impact important sur l'utilisation d'un système. Tout ceci accentue les risques de surcharge cognitive et empêche l'utilisation des techniques traditionnellement appliquées par les mécanismes de conscience de groupe.

Les collecticiels ne peuvent plus utiliser des mécanismes de visualisation élaborés, puisqu'on ne peut plus garantir que les utilisateurs disposent d'un dispositif capable de supporter ces mécanismes. Les collecticiels ne peuvent pas non plus envoyer d'importantes quantités de données par le réseau, car les utilisateurs peuvent être reliés par un réseau sans fil, à faible débit. Par ailleurs, ces utilisateurs ne disposent pas forcément de temps suffisant pour assimiler les informations de conscience de groupe, et leurs intérêts changent en fonction de leur contexte (par exemple : on ne s'intéresse pas forcément aux mêmes informations lorsqu'on est au bureau, en déplacement ou à la maison). Par conséquent, nous pouvons affirmer que les collecticiels, et tout particulièrement les collecticiels sur le Web, nécessitent désormais des mécanismes de conscience de groupe qui soient sensibles au contexte.

La majorité des systèmes sensibles au contexte se concentrent majoritairement sur l'adaptation du contenu ou de la présentation de ce contenu à la localisation de l'utilisateur et aux caractéristiques de son dispositif d'accès. Il s'agit d'une notion de contexte trop limitée pour les collecticiels. Les utilisateurs de ces systèmes sont engagés auprès d'un ou plusieurs groupes, au sein desquels ils ont des responsabilités et partagent des objectifs communs avec leurs collègues. Tout ceci participe également à la situation dans laquelle se trouve l'utilisateur (par exemple, on se comporte différemment lorsqu'une échéance importante pour le groupe de travail est imminente). Par conséquent, outre les éléments qui décrivent la situation physique de l'utilisateur en tant qu'individu, plusieurs éléments relatifs au groupe et au processus coopératif doivent également faire partie du contexte de cet utilisateur. Dès lors, *il apparaît nécessaire d'élargir la notion de contexte d'utilisation pour prendre en compte, dans un processus d'adaptation, tout ce qui relève de l'information relative au processus coopératif*

dans lequel l'utilisateur est impliqué. Par ailleurs, la majorité des systèmes sensibles au contexte manque de représentation formelle de la notion de contexte. Les quelques modèles qui sont proposés sont souvent difficiles à modifier et à faire évoluer, et la représentation d'informations incomplètes ou ambiguës n'est pas prise en compte, malgré l'occurrence de ce type d'information en raison de possibles défaillances dans le processus d'acquisition du contexte.

Cette notion élargie de contexte, si modélisée, peut être utilisée pour l'adaptation de l'information de conscience de groupe. Cette adaptation doit concerner d'avantage le filtrage de l'information de conscience de groupe, puisque, d'une part, les utilisateurs nomades sont souvent confrontés à diverses contraintes physiques (dispositif aux capacités limitées, par exemple) et environnementales (utilisation du système dans la rue, ou dans l'aéroport, etc.). D'autre part, ces utilisateurs s'attendent à recevoir du système seulement les informations les plus pertinentes pour eux étant donné le contexte courant, puisqu'ils ne disposent pas, en règle générale, de beaucoup de temps pour les analyser.

Ainsi, afin de réduire les risques de surcharge cognitive pour les utilisateurs nomades au sein des collecticiels sur le Web, *nous devons désormais réfléchir à un filtrage plus "intelligent" de l'information de conscience de groupe, capable non seulement de limiter l'ensemble d'informations délivrées à l'utilisateur aux informations pertinentes par rapport à son contexte courant, mais également capable d'organiser ces informations* de manière à ce qu'ils disposent d'abord des informations les plus pertinentes et ensuite puissent avoir, progressivement, l'accès aux informations les moins pertinentes.

Or, déterminer la pertinence d'une information pour un utilisateur dépend de cet utilisateur et de ses préférences. Cependant, les préférences d'un utilisateur nomade peuvent varier conformément au contexte dans lequel il se trouve. Nous avons observé que les utilisateurs d'un collecticiel, même les utilisateurs nomades, acquièrent certaines habitudes d'utilisation au fur et à mesure qu'ils incorporent le collecticiel dans leur quotidien. Par exemple, dans le cas du courrier électronique, il a été déjà observé que les utilisateurs consultent majoritairement leurs messages à partir des certains lieux bien définis (voir [Renevier 2004]) : dans l'ensemble, leur bureau et leur maison. De même, un utilisateur qui voyage va s'habituer à lire ses messages à partir des endroits qu'il fréquente normalement lors de ses déplacements : les aéroports, les gares, les hôtels, etc.

Nous pouvons alors considérer que, dans le cas des collecticiels, les utilisateurs nomades utilisent ces systèmes, dans la majorité des cas, à partir de situations plus ou moins connues et qui peuvent être caractérisées par certains éléments. Par exemple, l'utilisation du système à partir d'une certaine localisation (le bureau, la maison, une salle de réunion, ...), utilisant un dispositif donné (l'ordinateur de bureau, l'ordinateur portable, le téléphone cellulaire, etc.) ou encore en réalisant une activité précise (la rédaction d'un document, la lecture des messages, etc.). Pour chacune de ces situations, l'utilisateur peut développer certaines préférences en matière de conscience de groupe particulières à ces situations (par exemple, ne recevoir que les informations relatives à un document précis lorsqu'on réalise une activité de rédaction, ou ne recevoir que les informations relatives à la localisation courante, lorsqu'on se déplace avec l'ordinateur portable).

Il existe donc un lien clair entre les préférences d'un utilisateur nomade et son contexte courant. Ce lien, quoique évident, n'est pas pris en considération à ce jour ni par les collecticiels sur le Web, ni par les systèmes sensibles au contexte.

Face à toutes ces considérations, nous pouvons soulever quatre besoins qui découlent de l'utilisation des nouvelles technologies au sein des collecticiels sur le Web. Ces besoins

concernent, particulièrement pour nous, les mécanismes de conscience de groupe qui doivent pallier au problème de surcharge cognitive pour les utilisateurs nomades en les fournissant une information plus adaptée à leur contexte. Ainsi, nous pensons que :

- il faut élargir la notion de contexte afin de tenir compte des aspects collaboratifs qui entourent les utilisateurs nomades dans un collecticiel ;
- il faut une représentation de contexte capable de formaliser cette notion de contexte élargie, tout en respectant les caractéristiques propres à la notion de contexte, notamment, le fait d'être une notion qui peut évoluer dans le temps, avec les nouvelles technologies, le fait qu'il s'agit d'une information à caractère dynamique, qui change continuellement, et le fait que souvent il n'est pas possible de capturer toutes les informations liées à cette notion. Tout ceci doit être pris en considération par un modèle représentant la notion de contexte ;
- les informations de conscience de groupe doivent être filtrées selon le contexte et les préférences de l'utilisateur, de manière à lui présenter celles les plus pertinentes ;
- les informations délivrées à l'utilisateur doivent être organisées de manière à permettre un accès facile et potentiellement progressif à ces informations.

1.2 Aperçu de la Proposition

Dans cette thèse, nous nous focalisons sur les quatre besoins identifiés ci-dessus, et nous organisons notre proposition selon quatre axes :

(1) modélisation de la notion de contexte ;

(2) manipulation de la notion de contexte ;

(3) filtrage guidé par le contexte de l'information de conscience de groupe ;

(4) réalisation des propositions.

Dans le premier axe de travail, nous proposons une *formalisation de la notion de contexte* qui prend en compte les aspects collaboratifs qui caractérisent les collecticiels. Nous analysons la notion de contexte étant données les interactions entre l'utilisateur et le système dans un environnement coopératif. Nous considérons l'utilisateur comme étant membre d'un collectif, le groupe, dans lequel il a des responsabilités et des objectifs à atteindre. Par conséquent, nous proposons la prise en compte, dans la notion de contexte, de plusieurs concepts liés au processus de coopération mis en place par les collecticiel à travers un modèle à objets qui formalise cette notion.

Notre objectif avec le *modèle de contexte* est de séparer les informations contextuelles de celles propres au domaine d'application, afin de permettre l'évolution de ce modèle. Nous proposons donc un modèle de contexte évolutif qui utilise un formalisme par objets du type classe/association. Nous utilisons ce formalisme d'abord parce qu'il est largement connu et maîtrisé par les concepteurs, notamment à travers le langage UML[3]. Ensuite, ce formalisme nous permet d'exprimer non seulement les concepts qui prennent part à la notion de contexte, mais également les relations qui unissent ces concepts et enrichissent cette notion.

[3]« *Unified Modeling Language* » - http://www.uml.org/.

Notre second axe de travail considère la *manipulation de la notion de contexte*. Nous proposons, dans cet axe, une série d'*opérations* capables de comparer le contenu des instances du modèle de contexte proposé dans l'axe précédent. Chacune de ces opérations vérifie l'existence d'un type de relation entre les instances : (*i*) une *relation d'égalité*, qui analyse si deux instances ont un contenu équivalent ; (*ii*) une *relation d'inclusion*, qui vérifie si la description d'une situation précise se produit également dans la description d'une autre situation. On compare ici les concepts qui composent la description d'un contexte précis, et les associations qui unissent ces concepts, avec les concepts et les associations qui décrivent un autre contexte, afin de vérifier si la deuxième description se trouvent incluse dans la première description ; (*iii*) une *relation de similarité* qui cherche à quantifier le degré de similarité entre deux instances du modèle de contexte.

Dans le troisième axe composant ce travail, nous proposons un processus de *filtrage guidé par le contexte*. Ce processus vise à adapter l'information de conscience de groupe délivrée par le collecticiel sur le Web à un utilisateur nomade selon son contexte courant et ses préférences dans ce contexte d'utilisation précis. L'objectif est de mieux adapter l'information de conscience de groupe à l'individu, de lui fournir une information qui soit la plus pertinente pour lui dans son contexte courant, afin qu'il puisse ainsi optimiser ses propres actions dans le groupe. Pour cela, ce processus de filtrage utilise le modèle de contexte et les opérations que nous proposons dans ce travail. Ce modèle est utilisé à la fois pour représenter le contexte courant d'un utilisateur et pour représenter les situations dans lesquelles il peut potentiellement se trouver lors de l'utilisation du collecticiel. Ces contextes potentiels sont associés à l'ensemble des préférences que l'utilisateur a pour ces situations couramment rencontrées, à travers un *modèle de profil* que nous proposons dans cet axe du travail. Dans les profils, on exprime quels types d'information de conscience de groupe sont considérés comme pertinents pour un contexte donné, et l'organisation de ces informations. L'organisation de ces informations se fait à l'aide d'un *modèle d'accès progressif* [Villanova 2002] qui est utilisé au sein des profils afin d'exprimer les conditions de filtrage et d'organiser l'information de conscience de groupe en plusieurs niveaux de détails. Les informations de conscience de groupe sont représentées à travers un *modèle de contenu* que nous proposons également dans ce travail. Ce modèle de contenu est formé autour du concept d'*événement* qui représente un élément d'information de conscience de groupe.

À travers la combinaison de ces modèles, nous voulons permettre l'expression de préférences telles que "si l'utilisateur se retrouve chez lui, alors telles classes d'information sont prioritaires et doivent s'organiser de telle manière". Le *processus de filtrage* que nous proposons utilise donc ces modèles pour choisir, d'abord, quel ensemble de préférences est applicable au contexte courant de l'utilisateur, et ensuite pour appliquer ces préférences, en filtrant et en organisant les informations de conscience de groupe.

Enfin, le quatrième axe de notre travail concerne la réalisation des propositions faites par les axes précédents. En d'autres termes, nous mettons en œuvre, à travers un canevas nommé BW-$\mathcal{M}$, le processus de filtrage de l'information de conscience, ainsi qu'une implémentation du modèle de contexte et des opérations sur ce modèle. Le canevas BW-$\mathcal{M}$ est écrit en langage Java et son but est de fournir un mécanisme de conscience de groupe sensible au contexte. Nous le proposons en tant que composant pour la gestion de l'information de conscience de groupe, exploitable lors de la conception d'un nouveau collecticiel sur le Web sensible au contexte. De plus, nous proposons également deux démarches pour son application : d'abord une démarche traditionnelle, qui importe les paquetages Java composant le canevas BW-$\mathcal{M}$; ensuite une démarche reposant sur l'invocation d'un service Web, qui encapsule ces mêmes paquetages.

1.3 Organisation du Document

Ce document se divise en deux parties : *état de l'art* et *proposition*. Dans l'*état de l'art*, nous abordons les principaux thèmes de recherche qui sont directement liés à notre travail. Tout d'abord, nous présentons les concepts liés au domaine de recherche du Travail Coopératif Assisté par Ordinateur, notamment la notion de conscience de groupe. Puis, nous introduisons la seconde thématique centrale de ce travail : l'Informatique Sensible au Contexte. Nous définissons cette thématique à travers l'exposé de plusieurs travaux s'y rapportant. Par ailleurs, nous portons une attention particulière à la notion d'utilisateur nomade, que nous utilisons tout au long de ce document, et à l'étude du concept clé de cette thématique : la notion de contexte d'utilisation. La partie *proposition* présente ensuite les quatre axes de travail qui composent notre proposition : la formalisation de la notion de contexte, la proposition d'opérations pour la manipulation de la notion de contexte, la définition d'un processus de filtrage guidé par le contexte, et un canevas qui met en œuvre les propositions précédentes.

Ainsi, ce document s'organise en neuf chapitres D'abord, l'*état de l'art* s'organise comme suit : dans le *chapitre 2*, nous abordons le domaine du Travail Coopératif Assisté par Ordinateur et ses concepts clés, ainsi que notion de conscience de groupe et les caractéristiques des collecticiels sur le Web. Le *chapitre 3* concerne l'Informatique Sensible au Contexte, et notamment, les notions d'utilisateur nomade et de contexte, à la base de cette thématique. Pour conclure l'état de l'art, le *chapitre 4* fait une synthèse des problèmes mis en évidence et présente notre approche. Ensuite, la partie *proposition* de ce document s'organise en quatre chapitres : le *chapitre 5* présente la formalisation du contexte, et donc le modèle par objets que nous proposons ; le *chapitre 6* définit les opérations que nous définissons afin de manipuler le contenu de ce modèle de contexte ; le *chapitre 7* aborde le filtrage guidé par le contexte, et présente les autres modèles utilisés par ce processus de filtrage ; et le *chapitre 8* discute l'implémentation de ce processus dans le canevas BW-$\mathcal{M}$ et expose la méthodologie d'application de ce canevas. Enfin, le *chapitre 9* conclut cette thèse, en présentant notre bilan et les perspectives pour ce travail.

État de l'Art

2 Travail Coopératif et Conscience de Groupe

2.1 Introduction au domaine

Le rôle central d'un collecticiel consiste à aider le groupe à mener à bien son travail en commun. Pour cela, David [David 2001] a identifié cinq objectifs principaux : *(i)* obtenir des gains de performances; *(ii)* capitaliser des connaissances; *(iii)* améliorer des temps de réponse; *(iv)* partager des compétences; et *(v)* faciliter le travail à distance. Cependant, atteindre ces objectifs nécessite de relever de nombreux défis, non seulement techniques, mais également humains. Même des collecticiels bien conçus pourront encore échouer sans vraiment satisfaire les attentes des utilisateurs, si ces utilisateurs ne se sentent pas plus performants lors de l'utilisation [Fernández 2002]. En effet, la coopération au sein d'un groupe d'individus dépend en grande partie de la synergie existante entre les individus. Un groupe de personnes qui entament un travail ensemble ne caractérise pas forcément une coopération car à travers la coopération, ce groupe cherche à atteindre des résultats qui sont supérieurs à la simple addition des parties, à la simple somme des contributions individuelles. Pour Jeantet [Jeantet 1998], le résultat de la coopération n'est pas obtenu par ajustement ou assemblage de résultats partiels, mais par la confrontation de compétences et par la négociation entre différentes logiques. Pour cet auteur, ce ne sont donc pas seulement les résultats des actions qui sont mis en commun, mais les savoirs et les ressources qui y concourent. Prenons l'exemple de l'édition coopérative de documents. Selon Martínez-Enríquez *et al.* [Martínez 2002a], la production coopérative de documents est le résultat de la coordination entre les différents coauteurs pour mener à terme les activités nécessaires à la production conjointe de documents.

Parmi les défis techniques qui doivent être surmontés pour la mise en œuvre d'un collecticiel, nous pouvons souligner la question de la distribution des données et celle de l'interface. Les données dans un collecticiel peuvent n'être pas toutes localisées sur un même site. Certains collecticiels utilisent une architecture répartie, concernant plusieurs sites. Ces collecticiels sont donc concernés par les problèmes liés à l'utilisation du réseau, notamment la manière dont les données sont transmises à travers le réseau et dont les instances du collecticiel partagent ces données [Boyle 2002]. Ceci sous-entend le choix des protocoles de communication et des formats des données, ainsi que la gestion de problèmes tels que les déconnexions ou encore les retards de transmission.

Par ailleurs, l'interface d'un collecticiel est le moyen à travers lequel les utilisateurs peuvent percevoir le groupe, produire ensemble et communiquer. Elle est souvent le seul lien entre les membres du groupe qui peuvent être géographiquement dispersés. En d'autres termes, c'est à travers l'interface du collecticiel que la coopération aura lieu. Il faut donc que cette interface fournisse les moyens nécessaires pour la production en commun et pour la communication entre les membres. Pour David [David 2001], du fait de la vocation des collecticiels, cette interface est nécessairement de type multi-utilisateurs et devra rendre compte de l'activité du groupe. Pour cet auteur, il ne s'agit plus seulement de gérer une interface homme-machine classique liée à l'application concernée, mais de la faire évoluer vers une interface *homme*-machine-*homme*. Il faut également montrer l'état actuel du travail

en distinguant les actions menées par les autres membres du groupe, ainsi que la propre présence de ses acteurs dans le système. Ces informations font partie de la notion de conscience de groupe.

En plus des questions liées aux données partagées et à l'interface, se pose également la question de la communication entre les participants. Un collecticiel doit fournir à ses utilisateurs les moyens de communiquer, pour qu'ils puissent ainsi échanger des informations et aussi se coordonner. Il est important d'observer que la communication est un moyen de résoudre les éventuels conflits qui peuvent apparaître au cours de la coopération, ainsi qu'un moyen de coordination entre les membres de l'équipe. D'un point de vue technique, cette communication peut être implantée soit de façon synchrone, à travers les outils tels qu'un *chat*, soit de façon asynchrone, à travers le courrier électronique, par exemple. La première option exige que tous les acteurs soient présents au même moment dans le système, tandis que pour la deuxième, il existe un certain décalage entre le moment où un message est déposé et celui où ce message est reçu.

Enfin, les utilisateurs ont besoin de mécanismes qui les maintiennent informés des actions menées sur les objets au centre de leur intérêt, et sur *qui* a mené ces actions. Cette conscience des contributions des collaborateurs peut influencer les efforts immédiats d'un participant ou lui suggérer certaines modifications qui pourront améliorer la productivité du groupe dans son ensemble [Carroll 2003]. En d'autres termes, les utilisateurs participant à une tâche coopérative ont besoin de savoir ce qui se passe dans leur groupe. Ceci correspond à la notion de conscience de groupe, dont nous parlerons dans la section 2.2.

À travers ces questions, nous avons un aperçu des problèmes techniques qui doivent être surmontés par un collecticiel. Nous continuons notre présentation du domaine à travers la définition de quelques concepts clés.

2.1.1 Concepts clés

Avant de continuer, il est important de définir un vocabulaire de base qui sera employé dans les chapitres à venir. L'objectif ici est d'expliciter certains concepts que nous considérons comme clés pour une meilleure compréhension de ce domaine de recherche et des systèmes qui s'y attachent.

Groupe ou équipe

Le premier concept clé est celui de ***groupe***. Un groupe (synonyme, une ***équipe***) correspond à un ensemble de personnes qui travaillent ensemble afin d'atteindre un objectif commun, ou encore, *un ensemble d'individus travaillant sur un même domaine* [TarpinBe 1997]. Il s'agit par exemple d'un groupe de coauteurs qui désirent rédiger ensemble un document (un livre, un chapitre, un rapport, un article...), ou d'une équipe de développeurs qui travaillent sur la conception d'un logiciel. Par ailleurs, nous pouvons également considérer comme groupe, un ensemble d'acteurs qui exécutent une tâche commune, même si, individuellement, tous n'ont pas forcément les mêmes objectifs. C'est le cas, par exemple, des communautés virtuelles telles que les *MUD*[4] ou les jeux multi-utilisateurs en ligne comme *Warcraft* [Warcraft 2005]. Ces derniers représentent un important marché commercial qui se trouve en pleine expansion avec la démocratisation de l'ADSL[5] et d'autres technologies d'accès à Internet en haut débit [BBC 2005], [Elliott 2005], [PWC 2005].

[4]« *Multi-User Dungeon* » (donjons multi-utilisateurs).
[5]« *Asymmetrical Digital Subscriber Line* ».

Le concept de groupe n'est pas associé à une taille précise. Celle-ci peut varier de quelques unités (ou membres) à peine, à plusieurs membres sur toute une organisation, ou même sur plusieurs organisations. Dans ce cas, nous parlons de *groupes virtuels*[6]. Ces groupes sont très prisés par les organisations car ils sont mis en place rapidement, avec un objectif commercial précis et dans un cadre de fortes contraintes de temps et de ressources. Ils fournissent aux différentes organisations qui prennent part au groupe l'opportunité de coopérer en misant sur leurs compétences respectives [Skaf-Molli 2003]. Les collecticiels permettent donc à ce genre de groupe la création d'un environnement virtuel partagé par tous, qui n'existerait pas sans le collecticiel [Alarcón 2002].

La taille du groupe a pourtant une influence directe sur sa coordination : plus la taille du groupe augmente, plus importants sont les efforts de coordination nécessaires au sein du groupe. D'ailleurs, selon David [David 2001], dans les groupes de taille restreinte, on n'a pas besoin d'une organisation préétablie pour faciliter la concertation au sein du groupe. C'est la faible taille de ces groupes et la forte densité du réseau de communication interne ainsi formé qui constituent leur atout majeur.

Membre ou participant

Si nous appelons l'ensemble des utilisateurs un *groupe*, chaque individu dans ce collectif est appelé ***membre*** du groupe ou, tout simplement, ***participant*** du groupe. Il s'agit, du point de vue de la conception des collecticiels, des ***utilisateurs*** de ces systèmes. Un membre du groupe est donc quelqu'un qui participe aux activités de ce groupe, partageant ainsi avec les autres membres (ses ***collègues***) un objectif commun. Cependant, il est important d'observer qu'un individu peut appartenir, simultanément ou non, à plusieurs groupes. Ainsi, de façon plus générale, « *chaque participant est un individu personnalisé qui appartient à un ou plusieurs groupes* » [TarpinBe 1997].

Rôle

Les membres s'organisent au sein du groupe à travers le ou les rôles qu'ils y jouent. Un ***rôle*** représente les responsabilités d'un utilisateur (ou d'une classe d'utilisateurs) vis-à-vis du groupe. Le rôle est souvent lié aux droits dont dispose cet utilisateur dans l'environnement coopératif. Il s'agit d'un ensemble de privilèges et de responsabilités attribué à une personne [Ellis 1991]. Pour Tarpin-Bernard [TarpinBe 1997], un rôle est caractérisé par l'ensemble des droits et des devoirs du participant vis-à-vis de ses partenaires et des données partagées, pouvant évoluer au cours du temps. Nous pouvons ainsi considérer que chaque membre du groupe peut disposer d'un certain nombre de rôles (un nombre supérieur ou égal à un) dans le groupe, mais, à un instant *t*, chaque membre jouera effectivement un rôle en particulier (parmi ceux dont il dispose). Salcedo [Salcedo 1998] parle, dans ce cas, de plusieurs rôles *potentiels* et d'un rôle *effectif* pour se référer, respectivement, aux divers rôles que l'utilisateur peut jouer dans l'équipe et celui qu'il effectivement joue au moment présent.

Les rôles représentent donc l'organisation interne du groupe, sa structure hiérarchique. Cette organisation détermine les responsabilités de chacun dans le groupe. Elle a aussi une grande influence sur la façon dont se déroulent les interactions entre les membres du groupe. Le concept de rôle, selon David [David 2001], est capital pour comprendre ces interactions. Pour cet auteur, un échange entre les membres ne répond pas du hasard. Pour lui, il n'y a pas d'interaction sans un minimum de réciprocité, chacun doit s'efforcer de réagir de façon adéquate à la conduite de l'autre, et le rôle fonde les attentes réciproques. Il sert à orienter les interactions. Ainsi, on trouve des groupes très hiérarchisés qui comptent plusieurs rôles

[6]En anglais, « *virtual teams* » [Skaf-Molli 2003] ou « *collaborative virtual workgroup* »[Alarcón 2002].

organisés en divers niveaux, et d'autres groupes avec des structures très aplaties, avec un rôle unique (le *participant*) ou deux rôles : les participants et le responsable de l'équipe, celui qui joue un rôle de *leader*, appelé le plus souvent *coordinateur*. Le choix entre une structure plus ou moins hiérarchique ne dépend pas seulement du groupe et des tâches que le groupe doit exécuter, mais aussi du réseau social (l'organisation, la société, etc.) dans lequel il se trouve. Et selon ce choix, les interactions entre les membres du groupe seront plus ou moins libres.

Par ailleurs, à partir du constat qu'un rôle est un indicateur des responsabilités des individus qui le jouent dans le groupe, nous pouvons donc imaginer ce rôle comme un indicateur des informations nécessaires à ces individus pour mener à bien leurs responsabilités. En fait, les membres jouant des rôles distincts ont des besoins aussi distincts, certains rôles nécessitant des informations plus détaillées et plus précises que d'autres [KirschPi 2001]. Nous pouvons faire notamment la différence entre les informations nécessaires au coordinateur du groupe et celles nécessaires aux autres participants du groupe. Un coordinateur a besoin d'une vue de l'ensemble de la coopération afin de mieux suivre l'avancée des travaux et de prendre les décisions en conséquence. Un participant n'a pas forcément besoin des mêmes informations, puisqu'il n'a pas les mêmes responsabilités vis-à-vis du groupe. Ceci démontre bien toute l'importance que le concept de rôle a pour les collecticiels. Il s'agit d'un concept qui doit être bien maîtrisé par les concepteurs.

Processus

Un autre concept clé du domaine du TCAO est celui de processus. Le travail au sein d'un groupe s'organise, de façon implicite ou explicite, autour d'un ***processus*** de coopération qui peut être plus ou moins structuré. Ce processus décrit la manière dont la coopération entre les membres du groupe doit se dérouler, décrivant comment les données doivent circuler à l'intérieur du groupe ou encore quelles activités doivent être exécutées par le groupe afin d'atteindre ses objectifs. David [David 2001] a constaté que les activités professionnelles sont plus ou moins structurées. Les membres du groupe ne sont pas généralement banalisés, leur participation au traitement d'un dossier, par exemple, est définie par des règles précises spécifiant le cheminement du dossier dans l'organisation, pour que celui-ci soit traité, approuvé, etc. Ces règles définissent donc ce que nous appelons ici le *processus de coopération*.

Certains de ces processus sont très structurés, tandis que d'autres ne le sont pas. Dans le cas des processus structurés, tout le processus peut être exprimé dans un logiciel [Manheim 1998]. On parle donc de flots de travail ou *workflow*, et les collecticiels qui contrôlent ces flots sont appelés *systèmes de gestion des flots de travaux* ou *Workflow Management Systems*[7] [WfMC 2005]. Ces systèmes sont utilisés pour structurer le travail entre les individus dans une organisation, surtout lorsque le processus qu'ils formalisent est souvent répété et implique de multiples individus [Manheim 1998]. Lorsque le processus coopératif est complexe, il devient important d'en produire une représentation explicite pouvant servir de guide et de support au contrôle de son exécution [ECOO 2001]. Cependant, expliciter un processus de coopération n'est pas toujours possible ou souhaitable. Une alternative à cela consiste à définir ce processus de façon implicite, à travers notamment les activités et les conventions sociales du groupe. Pour Ellis *et al.* [Ellis 1991], le contrôle du processus de coopération peut être pris en compte par les conventions sociales du groupe, qui sont consenties et connues des membres du groupe, mais non nécessairement contrôlées par le collecticiel.

[7]Dorénavant, nous l'appelons simplement *workflow* ou systèmes de gestion de *workflow*.

Activité ou tâche

Un concept directement lié, par un lien de composition, à celui de processus est le concept d'***activité*** (ou ***tâche***). Une activité correspond à une action menée par un ou plusieurs membres du groupe afin d'atteindre l'objectif commun. Il s'agit, avec le concept d'objet partagé que nous verrons ci-dessous, d'une unité de base du travail en groupe. Souvent, pour atteindre son objectif, le groupe organise le travail en termes d'activités (et sous-activités) qui sont réalisées par les membres du groupe. Elles composent, en réalité, le contenu du processus de coopération : si le groupe est capable d'expliciter et de structurer ses activités, nous avons donc un processus bien défini et structuré. Dans le cas contraire, nous avons un processus implicite, semi- ou non-structuré. Par exemple, la rédaction d'un rapport technique peut s'organiser à l'intérieur soit d'un processus structuré, capable d'utiliser un *workflow* pour le guider, soit d'un processus semi-structuré. En revanche, un *brainstorming* va, en général, relever d'un processus non-structuré.

Par ailleurs, il est important d'observer la relation directe qui existe entre le concept d'activité et celui de rôle. Par définition, un rôle définit les responsabilités et les droits des membres du groupe. Il est donc normal qu'un rôle puisse indiquer les droits relatifs à une activité. Souvent les collecticiels attribuent directement aux rôles le droit de réaliser ou non une activité. Par exemple, dans le cas de l'édition coopérative, l'activité d'édition d'un document est habituellement associée au rôle d'*auteur*, tandis que le rôle de *lecteur* ne peut que consulter le document. À travers cet exemple, apparaît également le lien qui existe entre les rôles et le contrôle d'accès aux données partagées. Ce contrôle s'effectue à travers l'attribution des droits d'accès, lesquels peuvent être attribués soit individuellement à chaque membre du groupe, soit à des rôles en particulier.

Objet partagé ou artefact

L'autre unité de base du travail en groupe est le concept d'***objet partagé*** (aussi appelé ***artefact*** par certains travaux dans le domaine du TCAO). Les objets partagés correspondent à l'ensemble des données qui sont partagées par le groupe et sur lesquelles le groupe travaille. En d'autres termes, c'est sur les objets partagés que la coopération se concrétise : le groupe produit et manipule ces objets. Il le fait pendant les activités, un même objet pouvant être utilisé au cours de plusieurs activités du processus de coopération. D'ailleurs, certains objets peuvent correspondre à l'objectif même de la coopération. Par exemple, produire un document est souvent l'objectif dans le cas de l'édition coopérative, ou encore produire un plan 2D ou 3D dans un projet de conception collaborative. Ces objets peuvent également n'être que des produits intermédiaires de la coopération, agissant ainsi comme un espace d'échange entre les membres. Ces derniers sont aussi connus sous la dénomination d'*objets intermédiaires* [Jeantet 1998] qui sont des objets produits ou utilisés au cours du processus de conception, traces et support de l'action de concevoir, en relation avec les outils, les procédures et les acteurs.

Espace de travail partagé

Le concept d'***espace de travail partagé***[8] fait référence, comme son nom l'indique, à un espace dans lequel les membres du groupe peuvent partager leurs informations et mener des actions ensemble. Ceci renvoie, à la fois, à l'idée d'un espace où les utilisateurs partagent des artefacts, et dans ce cas on parle souvent de *répertoire partagé*, et à l'idée d'un espace au

[8]En anglais, « *shared workspace* ».

niveau de l'interface, où les membres du groupe peuvent interagir entre eux directement. Dans ce cas, lorsque cette interaction est synchrone (c'est-à-dire, elle se produit au même moment), le concept d'espace partagé vient souvent accompagner la notion d'interface WYSIWIS[9].

Session

Le concept de ***session*** correspond à une période d'interaction assistée par le collecticiel [Ellis 1991]. Il s'agit de la période dans laquelle un ou plusieurs utilisateurs sont connectés au système. Pour Laurillau [Laurillau 2002], cette période est celle pendant laquelle les utilisateurs participent à l'activité du groupe. Dans le contexte d'une organisation, la notion de session correspond à l'acte de présence au sein de l'organisation [TarpinBe 1997]. Techniquement, une session peut être vue comme un ensemble d'objets partagés et une série de moyens de communication communs connectant un certain nombre d'instances du collecticiel [Lopez 2003]. Nous considérons ici qu'une session est principalement la période pendant laquelle les utilisateurs sont connectés au système et participent directement aux activités du groupe.

2.1.2 Classifications

Étant donné leur complexité, la littérature classifie les collecticiels selon plusieurs critères différents : par type d'application (voir [Ellis 1991] [Dias 1998] [Laurillau 2002]), par l'occupation espace et temps (voir [Ellis 1991] [Grudin 1994] [Renevier 2004]), ou encore par la spécificité de la tâche (voir [Benali 2002]), entre autres. Parmi ces classifications, les deux premières sont les plus classiques.

2.1.2.1 Classification par type d'application

Au regard de la classification par type d'application, il existe plusieurs types de collecticiel : forums de discussions, systèmes de conférences, systèmes d'aide à la décision, juste pour citer quelques uns. Parmi les types les plus connus, nous pouvons souligner les *éditeurs coopératifs*, les *systèmes de gestion de workflow* et les *espaces de travail partagés*.

Les ***éditeurs coopératifs*** permettent aux membres du groupe de composer conjointement un objet partagé, le plus souvent un document. Cette catégorie comporte les éditeurs de documents, tels que *Grove* [Ellis 1991] et *AllianceWeb* [Salcedo 1998], ainsi que les systèmes pour le développement coopératif de logiciels, tel que *COPSE* [Dias 1999], ou encore les outils de tableaux blancs[10]. Ces derniers permettent aux membres du groupe de dessiner ou d'écrire sur l'espace partagé (le tableau) en même temps, comme sur un tableau réel [Prakash 1999].

Les ***systèmes de gestion de workflows*** sont dédiés à la gestion de processus (industriels, commerciaux, administratifs, etc.) et à la coordination des différents intervenants au cours d'un processus [Laurillau 2002]. Un *workflow* est défini comme l'automatisation d'un processus pendant lequel documents, informations ou tâches sont passés d'un participant à un

[9]L'acronyme *WYSIWIS* (« *What You See Is What I See* »), dont la traduction est « ce que tu vois est ce que je vois », dénote les interfaces dans lesquelles le contexte partagé est garanti [Ellis 1991]. Il s'agit du couplage de l'interaction entre les différents utilisateurs [Renevier 2004]. En d'autres termes, dès lors qu'un membre fait une modification dans cet espace, celle-ci est répercutée dans les interfaces des autres membres du groupe.
[10]En anglais, « *shared whiteboard* ».

autre afin qu'une action soit menée selon une de procédure préétablie [WfMC 2005]. Les systèmes de *workflow* conduisent à une explicitation complète du modèle de tâche [Benali 2002]. Ces outils fournissent un formalisme rigoureux pour décrire ce type de modèle et des mécanismes pour l'automatiser [WfMC 2005]. Cependant, de l'explicitation du modèle découle généralement une certaine rigidité du processus et un manque du support à la négociation. Cette rigidité peut donner lieu à une certaine insatisfaction des utilisateurs vis-à-vis de ces outils, à laquelle tentent de répondre plusieurs travaux récents portant sur les *workflows* adaptatifs (voir, par exemple, [Grinter 2000], [Agostini 2000] et [ECOO 2001]).

Les ***espaces de travail partagés*** permettent essentiellement le partage d'un ensemble d'objets entre les membres d'un groupe. Il s'agit également de fournir un espace de travail partagé au groupe. Un des plus célèbres collecticiels de cette catégorie est le *BSCW*[11] [BSCW 2005]. D'autres exemples sont *ToxicFarm* [Skaf-Molli 2003], *LibreSource*[12] [Forest 2005a] ou encore *Collabora* [ECOO 2004]. Ce dernier est particulièrement dédié à l'enseignement à distance.

2.1.2.2 Classification Espace et Temps

Pour tous les types de collecticiels indiqués par la classification par type d'application, nous pouvons considérer où et quand se déroulent les interactions entre les participants. Ceci est l'objectif de la classification Espace et Temps. La classification « Espace et Temps » a été originellement proposée par Ellis *et al.* [Ellis 1991], qui organisent les collecticiels sur deux axes (voir Tableau 1). Un premier axe, l'*Espace*, considère la distance spatiale entre les utilisateurs lors de l'interaction. Un deuxième axe, le *Temps*, considère la distance temporelle [Laurillau 2002]. L'axe *Espace* se divise en « *même lieu* » et « *lieux différents* ». Le premier indique que les membres du groupe sont co-localisés au moment de l'utilisation du collecticiel, tandis que le second indique qu'ils sont éparpillés dans l'espace. L'axe *Temps* est divisé en « *même moment* » et « *moments différents* », le premier indiquant que les utilisateurs interagissent simultanément (c'est-à-dire, de façon *synchrone*), et l'autre indiquant qu'il peut exister un décalage entre les interactions des utilisateurs, c'est-à-dire qu'ils interagissent de façon *asynchrone*. Le Tableau 1 présente cette division en deux axes, en présentant également quelques exemples de collecticiel pour chaque région.

Cette classification « Espace et Temps » a été élargie par Grudin [Grudin 1994], qui inclut dans celle-ci l'imprévisibilité du lieu ou du moment de l'interaction. Ainsi, l'axe *Temps* se divise en « *même moment* », « *moments différents et prévisibles* », et « *moments différents et imprévisibles* », tandis que l'axe *Espace* se divise en « *même lieu* », « *lieux différents et prévisibles* » et « *lieux différents et imprévisibles* ». Selon Renevier [Renevier 2004], l'aspect prévisible/imprévisible est lié aux contraintes imposées aux utilisateurs. Il se résume à une mise en correspondance entre l'interaction, les attentes et les suppositions des utilisateurs : un aspect est imprévisible si les utilisateurs ne peuvent prévoir *a priori* la valeur de cet aspect. On dit donc « *lieux différents et imprévisibles* » et « *moment différents et imprévisibles* » lorsqu'on ne peut pas prévoir à l'avance ni l'endroit (localisation physique), ni le moment où et quand se déroulera une action.

Dans les deux cas, il est important d'observer que tous les collecticiels ne sont pas forcément enfermés dans une seule région de cette matrice. Pour Grudin [Grudin 1994], le "travail réel" ne s'inscrit pas dans une seule région. En situations de travail, se produisent aussi bien des rencontres face-à-face, que des communications distribuées et asynchrones. Ellis *et al.* [Ellis 1991] suggèrent qu'un même collecticiel recouvre toutes les régions.

[11] *Basic Support for Cooperative Work.*
[12] « *LibreSource Community* » - http://dev.libresource.org/.

D'ailleurs, selon Salcedo [Salcedo 1998], les collecticiels pour l'édition coopérative couvrent toutes les catégories, puisqu'ils interviennent dans plusieurs situations, selon les besoins des auteurs et le domaine d'application des documents en production. Par exemple, il existe des éditeurs coopératifs asynchrones et distribués, tel que *AllianceWeb* [Salcedo 1998], des collecticiels synchrones et distribués, comme *Grove* [Ellis 1991], ou encore en face-à-face, tels que les outils d'édition proposés dans l'environnement *RoomWare* [Prante 2004].

Bien que simple, les classifications « Espace et Temps » comportent certains points d'ombre. Un de ces points est la distinction entre travail synchrone et asynchrone qui est très mince en certaines occasions. Prenons l'exemple de l'éditeur *AllianceWeb* [Salcedo 1998]. Cet éditeur permet à ses utilisateurs de travailler sur un même document de manière simultanée, donc synchrone, aussi que de manière asynchrone, sur une période de temps étendue. Pourtant, pour Salcedo [Salcedo 1998], cet éditeur est un exemple de collecticiel asynchrone, car il peut exister un décalage entre les actions de chaque coauteur, et ceci n'empêche pas la réalisation du travail par le groupe. Ceci rejoint la définition utilisée par Tarpin-Bernard [TarpinBe 1997]. Pour cet auteur *asynchrone* a simplement le sens : *qui ne se passe pas forcément en même temps*.

Par ailleurs, depuis peu la mobilité des utilisateurs a remis en cause la classification *Espace et Temps* [Canals 2002]. Grâce à l'évolution des technologies mobiles et aussi à l'accessibilité des systèmes sur le Web, les utilisateurs des collecticiels disposent désormais d'une liberté d'accès dont ils ne disposaient pas lorsque les classifications *Espace et Temps* ont été proposées. Il devient donc de plus en plus compliquée l'utilisation de cette classification.

	Même moment	***Moments différents***
Même lieu	Interaction face-à-face exemples : salle de réunion instrumentalisée [Prante 2004]	Interaction asynchrone exemples : *geonotes* ou *post-it* électroniques [Rubinsztejn 2004] [Espinoza 2001]
Lieux différents	Interaction synchrone et distribuée exemples : vidéoconférences et audioconférences [Skype 2005]	Interaction asynchrone distribuée exemples : workflow [ECOO 2004]

Tableau 1. Classification Espace et Temps (à partir de Ellis *et al.* [Ellis 1991]).

2.1.3 Technologies

Comme permettent d'observer les classifications présentées précédemment, les collecticiels connectent souvent un groupe de participants géographiquement distribués. Ceci implique l'utilisation d'une infrastructure réseau. L'architecture « *client-serveur* » est très souvent utilisée : un serveur central contient les données et fournit les services aux clients, à travers lesquels les participants se connectent au système. Bien que simple, l'architecture « *client-serveur* » présente un désavantage certain. Il possède un point de défaillance, le serveur, qui, une fois surchargé ou hors service, compromet tout le système. De plus, Internet n'est pas réellement fiable. Les concepteurs des collecticiels doivent fournir des solutions pour permettre aux utilisateurs de travailler en mode déconnecté (défaillance du réseau ou du serveur, travail nomade) ou en mode dégradé (délai ou lenteur de la transmission)

[Decouchant 2001]. Afin de pallier ces problèmes, une possible solution consiste à répartir soit les données, soit les services fournis par le collecticiel (soit les deux) sur différents sites.

Pour cela, on peut utiliser un *intergiciel*[13]. Un intergiciel est défini, dans les systèmes repartis, comme une couche logicielle entre le système d'exploitation et l'application sur chaque site du système. Son rôle consiste à faciliter la mise en oeuvre d'un système reparti, en cachant la distribution du système et l'hétérogénéité des systèmes d'exploitation [ObjectWeb 2005]. A l'intérieur d'un collecticiel, un intergiciel peut être utilisé aussi bien pour l'acheminement des messages du système, comme c'est le cas du canevas *ANTS* [Lopez 2003], que pour la distribution des données, comme c'est le cas de l'intergiciel *Piñas* [Decouchant 2001], particulièrement conçu pour la distribution des documents partagés dans le cadre des collecticiels pour l'édition coopérative. Une autre possibilité est l'utilisation des systèmes « *pair-à-pair* »[14]. Ces systèmes permettent le partage des données entre un ensemble dynamique et hétérogène de participants [Villamil 2004]. L'utilisation des systèmes pair-à-pair pose encore certains problèmes, surtout pour l'indexation et la propagation des requêtes dans le réseau de pairs. Cependant, leur flexibilité est un atout non négligeable grâce auquel ces systèmes deviennent une option intéressante pour la distribution des données à l'intérieur des collecticiels, ainsi que pour la structure même des collecticiels qui pourront adopter ce modèle pour leur architecture.

En tous les cas, il n'existe pas un modèle d'architecture pour les collecticiels qui soit largement accepté dans la communauté du TCAO. Du point de vue de l'implémentation, il est habituel de concevoir un collecticiel comme un ensemble de composants qui remplissent un certain nombre de fonctionnalités dans le collecticiel : support aux sessions, contrôle de concurrence, contrôle d'accès, etc. (voir, par exemple, le canevas *ANTS* [Lopez 2003]). On parle souvent des *services* fournis par le collecticiel, et dernièrement, des collecticiels composés par un ensemble de services, plus particulièrement des services Web (voir, par exemple, [Skaf-Molli 2003]).

En ce qui concerne l'accès aux données dans les collecticiels, plusieurs alternatives existent pour garantir un contrôle de concurrence cohérent. Selon Prakash *et al.* [Prakash 1999], ce contrôle doit garantir un temps de réponse compatible avec l'interactivité attendue du système, tout en assurant la consistance des données. Une alternative simple est l'utilisation des verrous. Dans ce cadre, un utilisateur qui désire apporter des modifications à un objet partagé pose un verrou sur l'objet. À partir de ce moment, d'autres utilisateurs ne peuvent plus modifier cet objet, seul le propriétaire du verrou est capable de le modifier. Cette situation perdurera jusqu'à l'utilisateur enlève son verrou. Ce modèle est très similaire au paradigme « *copier-modifier-fusionner* »[15] [Molli 2001] [ECOO 2001]. Dans ce paradigme, un utilisateur fait une copie locale d'un objet partagé dans son espace personnel. Il modifie ensuite cet objet et, une fois les modifications terminées, l'utilisateur dépose la nouvelle version de l'objet sur le répertoire partagé. C'est à ce moment que la fusion des objets a lieu. Le résultat de ce paradigme est que l'objet partagé devient une collection de versions successives (ce qui n'est pas forcément le cas avec l'utilisation des verrous). Par ailleurs, plusieurs recherches sur les bases de données proposent de nouveaux modèles de transaction plus flexibles, particulièrement adaptés aux collecticiels [Jiang 2001].

Par ailleurs, ces données deviennent de plus en plus souvent multimédia. L'utilisation croissante d'objets multimédia a un coût, surtout en termes de matériel et d'infrastructure. L'utilisation d'objets multimédias exige généralement l'emploi d'un matériel de pointe, possédant une large capacité d'affichage et de mémoire. Leur transmission exige aussi une

[13]En anglais, « *middleware* ».
[14]En anglais, « *peer-to-peer* » (ou simplement, P2P).
[15]En anglais, « *copy – modify - merge* » [Molli 2001].

infrastructure réseau puissante, à haut débit et ayant la possibilité de garantir un niveau de qualité compatible avec ces objets. Actuellement, ces exigences sont respectées en ce qui concerne les dispositifs fixes (ordinateurs de bureau fixes), qui sont de plus en plus puissants, et pour lesquels l'utilisation des réseaux à haut débit s'est démocratisée. Cependant, ces exigences représentent un réel problème lors de l'utilisation des dispositifs mobiles, tels qu'un téléphone cellulaire ou un ordinateur de poche, connectés à un réseau sans fil. Autant ces dispositifs que ce type de réseau présentent encore de sérieuses limitations. La question de l'adaptation du contenu multimédia aux environnements à fortes contraintes est donc posée pour les collecticiels qui utilisent cette technologie.

2.2 Conscience de Groupe

Le terme *conscience de groupe* est un concept très large, souvent ambigu, qui n'a de sens que lorsqu'il fait référence à la conscience que quelqu'un a de quelque chose [Liechti 2000], [Schmidt 2002]. Le terme « conscience de groupe » désigne certaines informations à travers lesquelles les membres d'un groupe, alors qu'ils sont engagés dans leurs activités individuelles, capturent ce que les autres participants font (ou ne font pas) et ajustent leurs propres activités en conséquence [Gutwin 2002] [Schmidt 2002]. Pour Dourish et Bellotti [Dourish 1992], l'information de conscience de groupe désigne *la connaissance qu'un utilisateur a à propos de son groupe, de ses collègues et de leurs activités, et constitue un contexte pour les activités individuelles*. Ce contexte est utilisé pour garantir que les contributions individuelles soient pertinentes pour le groupe dans son ensemble, et pour évaluer les actions individuelles par rapport aux objectifs et à la progression du groupe[16]. Celle-ci est la définition la plus répandue du terme « conscience de groupe » et celle adoptée par ce travail.

Le point central de toutes les définitions est la connaissance des activités des autres. Il s'agit de permettre la construction d'un espace cognitif commun, par le partage d'information et d'intention. Un collecticiel doit ainsi fournir des mécanismes capables de garantir que les utilisateurs soient conscients des activités des autres participants (c'est-à-dire, qu'ils soient capables de répondre à la question « *que font les autres* ? »). Ces mécanismes permettent une meilleure coordination car ils permettent aux utilisateurs de prévenir la duplication du travail et de réduire les conflits [Prakash 1999].

L'importance de la conscience de groupe au sein des collecticiels a été démontrée au cours de la dernière décennie. Par exemple, Espinosa *et al*. [Espinosa 2000] ont trouvé des résultats qui encouragent l'idée selon laquelle la conscience de groupe contribue à la coordination du groupe et à son efficacité. Le fait d'être conscient de la présence de ses collègues et de leurs activités est très important pour que le travail soit plus fluide et naturel [Gutwin 2002]. La conscience de groupe permet, par exemple, à un membre du groupe de savoir quelle est sa part de travail à effectuer dans le groupe pour atteindre le but commun. Cela permet, en retour, une meilleure coordination entre les membres du groupe [Bouthier 2004]. De plus, maintenir les différents acteurs informés de l'utilisation et des modifications apportées au produit de leur travail commun contribue à réduire le risque de duplication du travail et les problèmes d'intégration [Farschian 2001].

En somme, sans la conscience de groupe, il est difficile un groupe de travailler de manière coopérative et coordonnée [Sohlenkamp 1998]. Sans cette connaissance sur les

[16]De l'original en anglais, « *Awareness is an understanding of the activities of others, which provides a context for your own activity. This context is used to ensure that individual contributions are relevant to the group's activity as a whole, and to evaluate individual actions with respect to group goals and progress* » [Dourish 1992].

activités des autres membres du groupe, la coordination du travail devient difficile. Le résultat de ce travail perd alors en qualité. Il devient inconsistant, perd en cohésion et ne traduit plus les idées du groupe dans son ensemble, mais seulement une réunion d'idées qui ne sont pas forcément concordantes et qui sont souvent contradictoires.

2.2.1 Conception de mécanismes de conscience de groupe

La conception d'un mécanisme de conscience de groupe est un problème complexe. En plus des questions techniques liées à la diffusion de l'information de conscience de groupe, les concepteurs de ces mécanismes doivent également définir quelles informations seront diffusées aux utilisateurs par le mécanisme.

Plusieurs travaux cherchent à aider les concepteurs dans cette tâche. Certains, dont [KirschPi 2001] et [Tam 2004], proposent des canevas conceptuels qui guident les concepteurs dans leur choix à travers une série de questions. De ces questions ressortent un certain nombre d'observations intéressantes (voir [KirschPi 2001] et [Tam 2004] pour plus de détails). Par exemple, un collecticiel peut supporter les deux types d'interaction (synchrone et asynchrone). Cependant, les besoins en matière de conscience de groupe sont différents pour chaque type d'interaction. Lorsqu'un groupe interagit de façon synchrone sur un espace de travail partagé, ses membres ont besoin de se maintenir informés des actions de chaque participant dans les brefs délais et de manière très détaillée. Dans ce cas, des mécanismes tels que le télépointage[17] ou les ascenseurs multiples[18] indiquent, avec plus ou moins de précision, la position, dans l'espace de travail, de chaque membre du groupe actif en ce moment. Par contre, dans le cas d'une interaction asynchrone, cette information relative à la position des collègues dans l'espace de travail partagé est presque inutile puisque les membres du groupe ne travaillent pas forcément ensemble sur un même intervalle de temps.

Par ailleurs, ces travaux nous rappellent que la conscience de groupe ne se rapporte pas qu'aux actions effectuées au moment présent. La connaissance sur les activités planifiées pour un moment futur, tel qu'une date limite ou une réunion face-à-face prévue, peut être aussi importante pour la coordination des efforts de l'équipe que la connaissance sur les activés présentes menées au sein du groupe. La même chose se produit avec l'information de conscience de groupe relative aux actions passées. Selon Carroll *et al.* [Carroll 2003], la connaissance sur ce qui s'est passé récemment sur un objet et *qui* en est le responsable peut influencer les décisions d'un participant. Dans ce cas, nous parlons de « *conscience des événements passés* »[19] [KirschPi 2001], ou de « *conscience des modifications* »[20] [Tam 2004].

Dans tous les cas, avant de déterminer quelles sont les informations concernées par les mécanismes de conscience de groupe, il est important de comprendre comment cette information est construite. Il s'agit d'un processus de nature cognitive qui implique la capture et le traitement des informations dans un contexte donné. Tout d'abord, il s'agit de la capture, de la perception des divers stimuli qui parviennent à l'utilisateur à travers différents médias. Ces stimuli sont ensuite traités et interprétés par l'utilisateur, qui finira par les transformer en connaissances. Bouthier [Bouthier 2004] synthétise ainsi la construction de la conscience de groupe en quatre étapes :

1) **Agrégation** : les informations élémentaires sont agrégées en informations complexes ;

[17]En anglais, « *telepointer* ».
[18]En anglais, « *multi-scrollbar* » ou « *multi-user scrollbar* »
[19]En anglais, « *past event awareness* » ou « *past awareness* ».
[20]En anglais, « *asynchronous change awareness of artifacts* » ou « *change awareness* » [Tam 2004]

2) **Filtrage** : les informations complexes sont filtrées par rapport à l'information elle-même, au contexte et aux ressources cognitives de la personne ;

3) **Interprétation** : les informations ayant passé le filtrage sont interprétées par rapport au contexte auquel elles sont liées ;

4) **Internalisation** : les informations interprétées sont appréhendées par l'utilisateur.

Une fois appréhendées, les informations de conscience de groupe deviennent des connaissances pour l'utilisateur, lequel pourra les utiliser dans ses prochaines actions. L'utilisateur décide des actions à entreprendre en fonction de ses connaissances propres et de sa vision du reste du groupe. La conscience de groupe influence donc ses actions directement ou indirectement [Bouthier 2004]. Ce cycle « *perception – construction* (agrégation/filtrage/ interprétation/ internalisation) – *action* » se produit naturellement au cours des interactions en mode face-à-face. Selon Schmidt [Schmidt 2002], dans leur "attitude naturelle", les acteurs d'une coopération ne sont pas des spectateurs ignorants et désintéressés. Ils sont activement engagés dans la coopération. Ces acteurs, toujours selon Schmidt [Schmidt 2002], surveillent, d'une façon ou d'une autre, les activités de leurs collègues de manière à établir l'état, la progression, les directions, etc., de ces activités. Le problème est que, à partir du moment où les acteurs de la coopération ne sont plus en situation de face-à-face et doivent coopérer à travers un système informatique, ces informations, qui avant étaient naturellement captées, ne le sont plus et disparaissent. C'est au système (donc, au collecticiel) de rétablir ces informations et de permettre au cycle de « *perception – construction – action* » de se remettre en place.

Cependant, les ressources dont dispose un système informatique sont très limitées par rapport aux interactions du monde "réel". De plus, il ne s'agit pas simplement de reconstruire les moyens de communication en mode face-à-face à travers des liens vidéo ou audio, comme dans les vidéoconférences et audioconférences. Naturellement, la conscience de groupe émerge non seulement de la communication entre les participants et de la perception visuelle des actions entreprises par les autres, mais également des gestes, des regards, de tout l'environnement social qui accompagne le travail en mode face-à-face, ainsi que de la manipulation des objets partagés par le groupe pendant la coopération. Selon Hill et Gutwin [Hill 2004], lorsque les personnes sont face-à-face, elles acquièrent l'information de conscience de groupe de trois manières : *(i)* à travers la communication explicite; *(ii)* à travers l'observation de l'orientation et du mouvement des corps des collègues dans l'espace de travail; *(iii)* et à travers l'observation des effets des actions des collègues sur les objets dans l'espace. Il est donc clair que l'émergence de la conscience de groupe ne peut être réduite à un simple problème de richesse de canal de communication [Bouthier 2004]. Par ailleurs, alors que la compréhension de toutes les infinies facettes de la situation d'interaction réelle permettait des gains considérables, il faut être plus pragmatique si l'objectif est le développement d'une représentation formelle et finie de la conscience de groupe pour les collecticiels [ChalmersM 2002].

Un problème qui accable les mécanismes de conscience de groupe au sein des collecticiels est la question de la surcharge cognitive. Selon Bouthier [Bouthier 2004], d'une part les informations échangées et présentées peuvent être très nombreuses, particulièrement lorsque le groupe est formé d'un grand nombre de membres ou bien de nombreux artefacts sont manipulés. D'autre part les ressources mentales de l'utilisateur, comme sa mémoire ou son attention, sont limitées. Il doit pourtant interpréter et intégrer ces informations pour construire *sa* conscience de groupe. La surcharge survient lorsque l'utilisateur se retrouve en face d'un trop grand nombre d'informations à traiter. Il subit alors une situation de stress qui

peut l'amener à rejeter en bloc l'ensemble des informations. Ce stress peut poser des difficultés à un membre. Ces difficultés, bien que subies par un membre de groupe, peuvent être à l'origine de ruptures dans la réalisation d'activités et dans la circulation d'informations qui pénalisent le travail des autres membres.

Ainsi, afin de palier ce problème de surcharge cognitive, il convient de limiter les informations de conscience de groupe présentées à l'utilisateur. Cela peut se faire à deux niveaux : en utilisant une représentation qui facilite l'interprétation et l'intégration des informations (visualisation), ou bien en limitant les informations reçues aux seules informations pertinentes (filtrage) [Bouthier 2004]. La visualisation réside essentiellement dans la conception d'interfaces moins intrusives et qui permettent l'agrégation des informations. L'idée est de présenter l'information d'une manière périphérique, à travers l'utilisation de couleurs, d'icônes, et d'autres ressources d'interface capables de transmettre, rapidement et sans requérir beaucoup d'attention de la part de l'utilisateur, une certaine information (la présence des collègues, leurs activités, les modifications sur un objet, etc.). Dans la section suivante, nous présentons certains mécanismes qui utilisent des techniques de visualisation afin de réduire le risque de surcharge cognitive.

En ce qui concerne le filtrage, il a deux rôles : diffuser les informations les plus pertinentes à chaque membre du groupe et permettre à l'utilisateur de choisir les informations à diffuser aux autres membres du groupe [Bouthier 2004]. En d'autres termes, le filtrage des informations de conscience de groupe se situe à deux niveaux distincts : au moment de délivrer ces informations aux utilisateurs et au moment de sa production, c'est-à-dire lorsque les changements à l'intérieur du groupe se produisent. Dans ce premier cas, il s'agit d'un *filtrage en sortie*, dans le second d'un *filtrage en entrée*.

Le *filtrage en sortie* consiste à filtrer les informations de sorte que chaque utilisateur puisse recevoir les informations qui lui paraissent pertinentes. Néanmoins, déterminer la pertinence de l'information de conscience de groupe est un défi en soi, surtout lorsque l'on considère que cette pertinence dépend directement des intérêts de chaque membre du groupe. Or, ces intérêts peuvent changer au fur et à mesure que le travail évolue et que la situation dans laquelle se trouve l'utilisateur change. Afin de résoudre ce problème, David et Borges [DavidB 2001] proposent deux pistes à suivre : l'utilisation de filtres définis par l'utilisateur lui-même, et l'utilisation de filtres directement liés au rôle joué par le participant à un moment donné. Les filtres définis par les utilisateurs eux-mêmes leur permettent d'exprimer leurs attentes vis-à-vis du mécanisme de conscience de groupe, ainsi que leurs intérêts personnels vis-à-vis du groupe. La définition des filtres liés aux rôles permet la détermination *a priori* des informations (ou des types d'informations) capables d'aider un membre du groupe à jouer un rôle donné. Prenons l'exemple du coordinateur d'un groupe. Quelqu'un qui joue un tel rôle détient une position de responsable dans ce groupe. Il doit, par conséquent, prendre de décisions importantes concernant le travail du groupe dans son ensemble (où concentrer les efforts, comment organiser le travail, etc.), et ceci pendant toute la durée du travail, en tenant compte de l'évolution et des résultats obtenus. Le coordinateur doit donc avoir une vue d'ensemble sur le groupe, ses activités et sa progression : savoir qui a contribué et comment, quand sont les dates limites, comment avance le travail, etc. Toutes ces informations ne sont pas forcément intéressantes pour les autres participants qui n'ont pas les mêmes responsabilités vis-à-vis du groupe. Ainsi, quelqu'un qui joue, dans un groupe, un rôle de coordinateur (ou l'équivalent) a besoin d'informations de conscience de groupe qui correspondent particulièrement à ses fonctions dans le groupe. S'il dispose de ces informations, il peut remplir plus facilement ce rôle de coordinateur.

Une autre piste pour le filtrage en sortie est celle suivie par Bouthier [Bouthier 2004], qui consiste à utiliser *le contexte de travail de l'utilisateur* pour filtrer l'information de

conscience de groupe. Pour cet auteur, le contexte de travail d'une personne (dans le cadre du travail en groupe) est l'ensemble des informations qui peuvent être utilisées pour caractériser son activité. Il peut être vu comme la source des informations de conscience de groupe, c'est-à-dire les informations que le mécanisme de conscience de groupe doit capturer et propager à tous les membres du groupe pour que ces derniers puissent construire leur conscience de groupe. Bouthier [Bouthier 2004] utilise cette notion dans un mécanisme de visualisation de l'information de conscience de groupe. L'idée est de présenter en premier plan les informations liées aux activités courantes de l'utilisateur. Cet exemple illustre le fait qu'il n'existe pas forcément une séparation entre la visualisation et le filtrage : ils peuvent être combinés au sein d'un mécanisme de conscience de groupe.

Le *filtrage en entrée* permet quant à lui le contrôle des informations produites par le mécanisme de conscience de groupe. Il s'agit d'un contrôle nécessaire au respect de la vie privée de l'utilisateur. Le respect de la vie privée est un autre défi que les concepteurs de mécanismes de conscience de groupe doivent surmonter. Il est particulièrement sensible dans les mécanismes qui fournissent la *conscience de la communauté*, lesquels focalisent sur la présence de l'utilisateur qu'ils détectent souvent à l'aide de connections vidéos (voir, par exemple, [Greenberg 1996] et [Boyle 2005]). Au fur et à mesure qu'un collecticiel capte des informations concernant les activités et la présence des utilisateurs, ces derniers sont plus susceptibles de se sentir envahis par le collecticiel. Celui-ci peut endosser, pour les utilisateurs, un caractère d'espion, conçu pour les observer, pour contrôler leurs moindres faits et gestes. Selon Bouthier [Bouthier 2004], si l'utilisateur trouve que le mécanisme de conscience de groupe ne constitue qu'un intrus et non pas une aide, alors il ne l'utilisera pas. Même s'il ne s'agit pas de l'intention délibérée des collecticiels, il arrive que les mécanismes de conscience de groupe donnent cette impression à leurs utilisateurs, l'observation des activités des utilisateurs étant une condition *sine qua non* pour la mise en place d'un mécanisme de conscience de groupe. Cependant, il est compréhensible qu'*a priori* l'utilisateur ne souhaite pas que tous les membres du groupe aient accès à toutes ses informations [Bouthier 2004]. La conception d'un collecticiel impose donc un équilibre entre le respect de la vie privée et la conscience de groupe [Laurillau 2002].

Enfin, il convient de noter que la production de l'information de conscience de groupe doit se faire de manière automatique par le collecticiel. Les membres du groupe ne doivent pas être conscients de cette production, et n'avoir qu'à se concentrer sur leur travail au sein du groupe. Il revient au collecticiel d'observer le groupe et de capturer les informations concernant le groupe et ses activités, et de les transformer ensuite en informations de conscience de groupe. Dans la section suivante nous présentons quelques mécanismes de conscience de groupe et étudions comment ils traitent les défis présentés ci-dessus.

2.2.2 Les mécanismes de conscience de groupe

Les mécanismes de conscience de groupe sont généralement intégrés à l'interface du collecticiel. Il est préconisé que cette intégration soit aussi souple que possible, les mécanismes de conscience de groupe ne devant pas interrompre l'utilisateur dans sa tâche principale qui est la coopération. Afin de permettre à l'utilisateur de mettre à jour sa conscience de groupe, le mécanisme doit notifier l'utilisateur d'un changement qui a eu lieu dans le groupe en lui présentant l'information reçue en tenant compte des problèmes liés à la surcharge cognitive et au respect de la vie privée.

Bouthier [Bouthier 2004] identifie quatre approches pour les mécanismes de conscience de groupe : l'approche *périphérique*, l'approche *graphique*, l'approche de *diffusion des messages*, et l'approche de *modélisation de la conscience de groupe*. Pour cet auteur, ces

approches ne sont pas antagonistes et ne représentent en rien une segmentation de l'espace des solutions. Il faut, selon lui, les voir comme complémentaires, car elles abordent chacune un aspect différent de la conscience de groupe.

L'approche dite « ***périphérique*** » se focalise sur les échanges « périphériques », presque inconscients, de la conscience de groupe. Selon cette approche, les mécanismes doivent s'appliquer sans que l'utilisateur n'ait à y accorder une attention particulière [Bouthier 2004]. Des exemples typiques de cette approche sont les systèmes destinés à faciliter les communications informelles dans le groupe, tels que *Peepholes* [Greenberg 1996], *handiMessenger* [Hibino 2002], *AwareNex* [Tang 2001], ou encore *Community Bar* [McEwan 2005]. Ces systèmes utilisent de petites images, icônes ou mots pour passer l'information relative aux autres membres. Le système *handiMessenger* [Hibino 2002], par exemple, introduit à côté de chaque message certaines informations sur la présence de l'émetteur du message, tandis que *AwareNex* [Tang 2001] utilise une technique similaire sur la liste des contacts de l'utilisateur (voir Figure 1). Pour sa part, *Community Bar* [McEwan 2005] cherche à donner une vue complète de la communauté. *Community Bar* (voir Figure 2) fournit non seulement les informations concernant la présence des autres membres (par des images statiques ou vidéo), mais aussi des moyens de communication (*chat* ou *post-it*) et l'accès à certains objets partagés (notamment des pages Web).

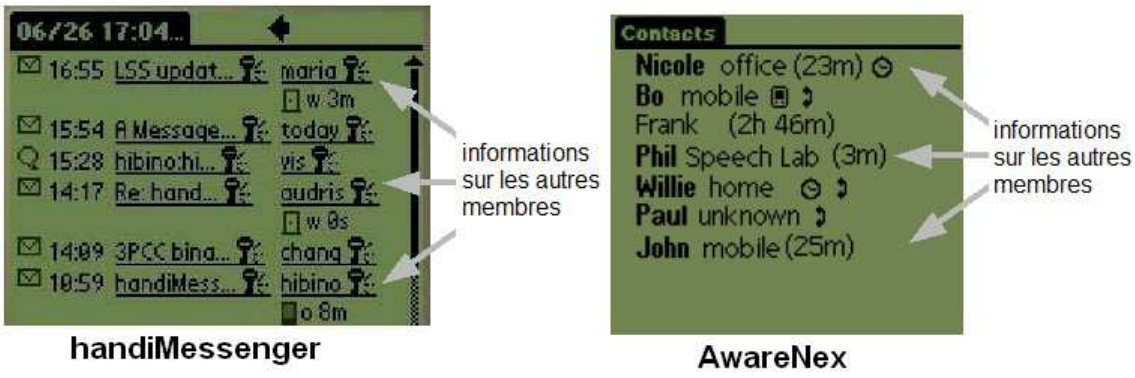


Figure 1. Les informations sur la présence ou la disponibilité des collègues dans handiMessenger [Hibino 2002] (à gauche) et dans AwareNex [Tang 2001] (à droite).

L'approche *périphérique* limite donc la surcharge d'informations au niveau de la présentation, car elle met en avant des mécanismes ne demandant pas d'attention active de la part de l'utilisateur [Bouthier 2004]. Par exemple, un rapide coup d'oeil dans *Community Bar* suffit pour s'apercevoir de la présence d'un collègue. En revanche, même si le problème du respect de la vie privée est particulièrement important dans ces systèmes, souvent il n'est pas traité directement (c'est le cas de *handiMessenger* [Hibino 2002]) ou il n'est assuré qu'à travers l'application de filtres (voir [Boyle 2005]).

L'approche « ***graphique*** » axe ses efforts sur la création de nouvelles interfaces graphiques pour représenter les informations de conscience de groupe. Dans le cadre des espaces de travail partagés, le but de cette approche est de fournir de nouveaux composants d'interface, des *widgets* (contraction de *window* et *gadget*) permettant de rendre compte de la collaboration. L'approche *graphique* se concentre donc sur la représentation de la conscience de groupe, particulièrement la conscience de l'espace de travail, laquelle, comme son nom l'indique, représente la conscience de ce qui se passe dans l'espace de travail partagé [Bouthier 2004].

L'exemple le plus classique de cette approche est *Groupkit*, une boîte à outils[21] pour le développement des collecticiels synchrones et répartis [Roseman 1996]. *Groupkit* propose aux développeurs une série de *widgets* pour la conscience de l'espace de travail : télépointage, ascenseurs multiples, etc. Un autre exemple est la boîte à outils MAUI (*Multi-user Awareness UI*) [Hill 2004], qui, en plus des *widgets* spécifiques aux collecticiels, fournit aussi des versions "partagées" de composants d'interface traditionnels, tels que des boutons ou des menus (voir Figure 3).

Figure 2. Les différents indicateurs de Community Bar [McEwan 2005].

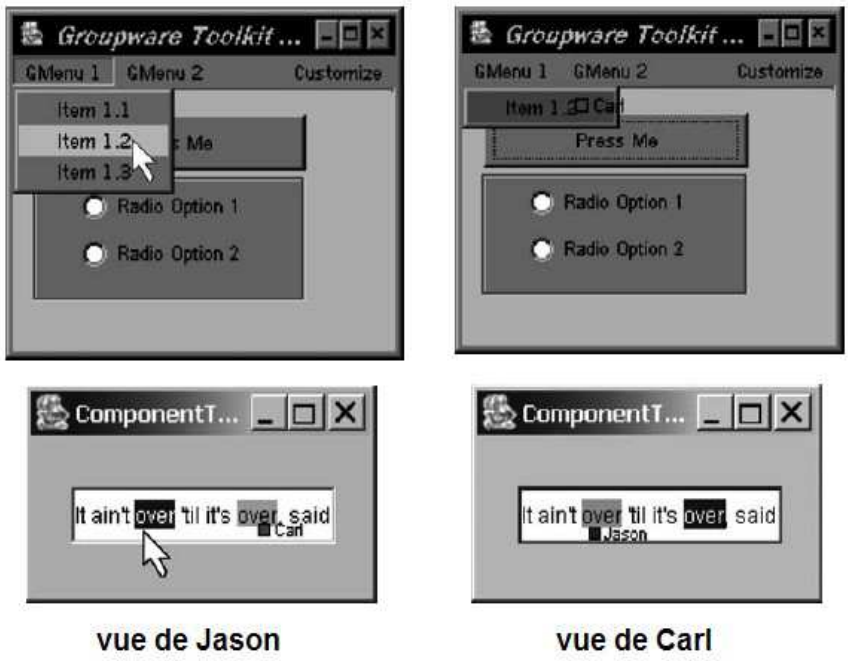


Figure 3. Deux exemples de *widgets* partagés disponibles dans la boîte à outils MAUI (un menu et une zone de texte) [Hill 2004]. À gauche, leur vue chez l'utilisateur Jason, et à droite leur vue distante, telle qu'un autre utilisateur, Carl, les percevra.

[21]En anglais, « *toolkit* ».

L'approche graphique limite le problème de la surcharge par la présentation de l'information de conscience de groupe dans son contexte de production au sein de l'espace de travail partagé, ce qui la rend plus facile à interpréter. Par ailleurs, la présentation des informations de manière graphique permet généralement à l'utilisateur de saisir facilement et rapidement les informations importantes, économisant ainsi ses ressources cognitives [Bouthier 2004]. Cependant, il est important de souligner que si le nombre de participants augmente de manière significative, ces informations deviendront difficiles à gérer, et la question de surcharge sera à nouveau posée.

L'approche « ***diffusion de messages*** » s'intéresse plus particulièrement à l'infrastructure technique nécessaire à un mécanisme de conscience de groupe. L'idée est de fournir des moyens pour la propagation des informations permettant l'émergence de la conscience de groupe. Il est à noter que ni la collecte de ces informations en amont ni leur visualisation en aval ne sont abordées par cette approche. Les résultats sont le plus souvent des infrastructures génériques et réutilisables de propagation de messages, possédant différents mécanismes de souscription, d'envoi-réception, etc. [Bouthier 2004].

Dans cette approche, s'inscrivent des infrastructures telles que le canevas *BW* [KirschPi 2003a], ou encore *ANTS* [Lopez 2003]. Le premier est un canevas totalement dédié à la diffusion de messages de conscience de groupe, et plus particulièrement ceux relatifs à la conscience des événements passés. Le second est une infrastructure pour la conception de collecticiels à part entière, et non pas seulement de mécanismes de conscience de groupe. *ANTS* contient un composant dédié à la conscience de groupe qui est déjà intégré dans la structure globale, ainsi qu'un intergiciel pour la diffusion des messages entre les différents composants.

En ce qui concerne la question de la surcharge, cette approche concentre ses efforts sur le filtrage des informations qui permet de limiter les informations reçues. Néanmoins, c'est généralement l'utilisateur lui-même qui spécifie les messages qu'il veut ou non recevoir. Cela pose, selon Bouthier [Bouthier 2004], les problèmes suivants : *(i)* la tâche consistant à choisir les messages peut représenter une surcharge de travail importante ; *(ii)* si ce choix utilise des règles ou des expressions régulières, il demande un savoir faire supplémentaire à l'utilisateur ; *(iii)* l'utilisateur ne sait pas forcément quels sont les messages qui seront pertinents pour son activité ; *(iv)* la spécification de messages à recevoir ne change pas automatiquement lorsque l'activité de l'utilisateur change. Toutefois, il est important de remarquer que l'approche *diffusion de messages* est facilement combinée à l'approche *graphique* afin d'obtenir un résultat plus complet dans la conception des mécanismes de conscience de groupe.

Les travaux de l'approche « ***modélisation de la conscience de groupe*** » cherchent à représenter les relations entre les informations constituant la conscience de groupe et les membres du groupe. Ces travaux ne fournissent qu'un moyen de décrire et d'utiliser les relations entre les informations, mais ces relations doivent tout de même être mises en place soit par le concepteur, soit par l'utilisateur [Bouthier 2004]. Un exemple de cette approche est le travail de Leiva-Lobos et Covarrubias [Leiva-Lobos 2002] qui cherchent à conceptualiser la notion de conscience de groupe selon trois axes principaux : la communauté, l'espace, et les événements.

Finalement, il est important d'observer que, malgré les efforts entrepris par les différentes approches, la conception d'un collecticiel et, plus particulièrement, de mécanismes de conscience de groupe qu'il intègrent, reste un problème complexe. La conception de mécanismes de conscience de groupe nécessite toujours beaucoup de temps lors de la conception, et une partie importante du code pour la capture, la distribution et la visualisation de l'information de conscience de groupe n'est pas toujours réutilisable [Hill 2004]. Mis à part quelques exceptions, le support à la conscience de groupe apporte en général des solutions

spécifiques à un domaine d'application et des principes difficiles à généraliser à d'autres situations, obligeant les concepteurs à réinventer ce support à chaque fois, à partir de leur propre expérience [Gutwin 2002].

Parmi les solutions *ad-hoc* spécifiques à un domaine d'application, le projet *POLITeam* [Sohlenkamp 1998] [Sohlenkamp 2000] propose un système pour le partage des documents dans le cadre de l'administration publique allemande. Ce projet traite, avec une attention particulière, la conscience des événements passés, en proposant plusieurs solutions telles que l'utilisation des icônes qui changent de forme et de couleur suivant les modifications, des messages textuels courts, entre autres (voir Figure 4). Ces solutions sont trop spécifiques à ce projet (par exemple, l'utilisation des couleurs selon les codes de l'administration en question) et sont difficilement réutilisables.

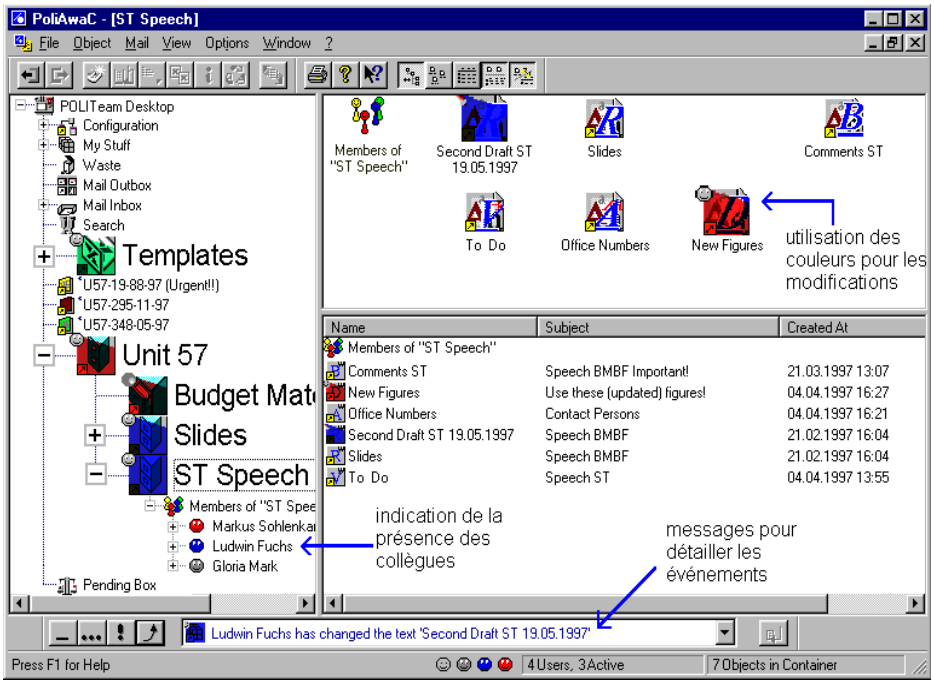


Figure 4. L'environnement POLITeam [Sohlenkamp 1998] et ses mécanismes de conscience de groupe.

2.3 Les Collecticiels sur le Web

2.3.1 Caractéristiques générales

Ce travail de thèse porte un intérêt particulier aux collecticiels sur le Web. La conception des systèmes reposant sur cette technologie s'est accrue, et les collecticiels ne sont pas une exception. Selon Liechti [Liechti 2000], le Web peut être vu comme une plate-forme d'implémentation pour les collecticiels, puisque, au cours de son évolution, le Web a été étendu (avec les CGI, les contenus dynamiques, etc.) vers une collection de services. Les navigateurs ne sont plus utilisés seulement pour la visualisation des documents, mais également pour accomplir des actions. En ce qui concerne la conscience de groupe, Liechti [Liechti 2000] voit le Web d'abord comme une plate-forme pour la construction de mécanismes de conscience de groupe, puis comme un espace d'activités (un espace de travail) dont les utilisateurs doivent être informés. Ainsi, on peut voir le Web dans deux optiques

complémentaires, tantôt comme une infrastructure sur laquelle les collecticiels seront construits, tantôt comme l'espace de travail dans lequel les activités du groupe ont lieu.

Le Web présente certaines caractéristiques qui influencent les collecticiels qui l'utilisent. La première de ces caractéristiques est son architecture. Le Web suit traditionnellement une architecture *client-serveur*, dans laquelle un serveur détient les services et les données, alors que les clients demandent ces données (ou services) au serveur (voir Figure 5). Au-dessus de cette architecture, un modèle *requête-réponse* est mis en place dans lequel une action du côté serveur n'est exécutée qu'après qu'une demande de la part d'un client a été formulée. Le serveur est donc une entité plutôt passive qui attend les demandes des clients, tandis que ces derniers sont les entités actives, qui demandent un service ou une donnée.

Figure 5. Structure de fonctionnement du Web.

Ces caractéristiques (l'architecture client-serveur et le modèle requête-réponse) favorisent la construction de collecticiels asynchrones et centralisés, qui les exploitent naturellement. En revanche, dans le cas contraire (collecticiels synchrones ou répartis), les concepteurs sont contraints de construire des mécanismes supplémentaires pour surmonter les limitations posées par l'architecture client-serveur (la centralisation des données et des services, par exemple) et le modèle requête-réponse (la passivité du serveur, par exemple).

Par ailleurs, le protocole HTTP[22], qui sert de base au Web, est un protocole sans état[23]. En d'autres termes, un serveur Web ne garde pas d'information concernant la session de chaque utilisateur. Une fois la réponse du serveur envoyée (le contenu, un message d'erreur, etc.), la communication entre le serveur et le client est fermée [Lemlouma 2004a]. Or, dans un collecticiel, les actions des utilisateurs se trouvent généralement dans une session qui dépasse largement le cadre d'une simple interaction requête-réponse. D'ailleurs, on ne trouve pas dans les standards Web de support adéquat à la gestion des utilisateurs, ni à l'identification des auteurs des contributions et des actions dans le cadre d'une coopération [Decouchant 2001]. Ainsi, les concepteurs d'un collecticiel sur le Web doivent, encore une fois, créer des mécanismes internes au collecticiel et extérieurs aux standards Web pour contrôler les sessions des utilisateurs.

En revanche, le Web présente un sérieux avantage pour la construction des collecticiels : l'accessibilité. En construisant son système sur le Web, le concepteur d'un collecticiel est sûr de rendre son système disponible partout dans le monde. Il s'agit d'un sérieux atout pour les systèmes qui cherchent à supporter des groupes géographiquement répartis en des lieux différents et non connus à l'avance.

Enfin, les collecticiels sur le Web présentent les mêmes problèmes de mise en oeuvre que les collecticiels et les mécanismes de conscience de groupe traditionnels, surtout les

[22] *HyperText Transfer Protocol* - http://www.w3.org/Protocols/.
[23] En anglais, « *stateless protocol* ».

collecticiels synchrones et repartis pour lesquels l'architecture Web est trop limitée. Cependant, une préoccupation est accrue pour les concepteurs de collecticiels sur le Web : celle concernant l'adaptation. En raison de l'immense diversité des plates-formes d'exécution (dispositifs et systèmes d'exploitation) capables d'accéder à un système sur le Web, celui-ci, qu'il soit coopératif ou non, est tenu d'adapter son interface aux capacités de la plate-forme d'exécution effectivement employée par un utilisateur donné. Actuellement, les dispositifs les plus variés, de l'ordinateur de bureau au téléphone cellulaire, sont capables d'accéder au Web, et, par conséquent, aux systèmes sur le Web. Or, ces dispositifs, ainsi que les systèmes d'exploitation embarqués, présentent des capacités (d'affichage, de mémoire, etc.) aussi variées. Concevoir un système sur le Web demande donc de prendre en compte ces variations. Lorsqu'il s'agit d'un collecticiel, il faut aussi considérer que les membres du groupe n'utilisent pas forcément *tous* les mêmes plates-formes d'exécution. Il faut donc que ce collecticiel soit capable d'accomplir sa mission (assister le travail en groupe) sur plusieurs plates-formes. Même s'il n'aura pas exactement les mêmes fonctionnalités dans toutes les plates-formes (un utilisateur muni d'un téléphone cellulaire ne sera pas forcément intéressé par les mêmes fonctionnalités qu'un autre muni d'un ordinateur fixe), il faut que le collecticiel ait une interface appropriée à chacune des plates-formes diverses depuis lesquelles il est accessible.

Ce besoin d'adaptation de l'interface rappelle l'intérêt porté par le domaine du TCAO aux recherches autour de la *malléabilité* des collecticiels[24]. Ces recherches concernent l'adaptation des collecticiels aux nouveaux (c'est-à-dire non prévus pendant la conception du collecticiel) besoins du groupe (voir, par exemple, [Benali 2002], [Teege 2000], [Wang 2000]). La malléabilité est vue comme l'extension dynamique d'une application afin de couvrir les nouvelles demandes qui n'étaient pas envisagées au départ [Fernández 2002]. L'objectif est donc de permettre aux utilisateurs d'adapter les fonctionnalités du système à leurs besoins spécifiques [Wang 2000]. Selon Teege [Teege 2000], la flexibilité et la malléabilité sont devenues deux aspects cruciaux pour le succès des collecticiels. Ainsi, on peut voir la malléabilité comme l'adaptation des fonctionnalités du collecticiel à la demande du groupe. Il s'agit d'une activité coopérative en soi, puisque c'est une adaptation valable pour tout le groupe, et non seulement pour un individu dans le groupe (voir [Fernández 2002]).

Outre la malléabilité, il convient d'observer que d'autres types d'adaptation sont également possibles : *(i)* l'*adaptation des fonctionnalités* du système de manière automatique (sans l'intervention directe du groupe, comme dans la malléabilité) ; *(ii)* l'*adaptation de l'interface* (aussi nommée *adaptation de la présentation*) ; et *(iii)* l'*adaptation des données.* Tous les trois types peuvent suivre ou non des instructions préétablies par les utilisateurs, comme, par exemple, le filtrage des données ou la personnalisation de l'interface selon un profil personnel préétabli.

Les collecticiels sur le Web sont donc, avec l'évolution de cette technologie et de ses supports physiques, de plus en plus concernés par l'adaptation, qu'elle concerne les données, l'interface, ou les fonctionnalités.

Dans la prochaine section, nous illustrons les collecticiels sur le Web, en présentant certains représentants et la manière dont ceux-ci traitent la question de la conscience de groupe.

2.3.2 Représentants

À titre d'illustration, nous présentons ici quelques exemples de collecticiels sur le Web connus dans la littérature. Nous commençons par un des plus connus : le *BSCW* [BSCW

[24]En anglais, « *taillorability* » (voir [Benali 2002]).

2005] [Appelt 2001]. *BSCW* (*Basic Support for Cooperative Work)* est un système de répertoire(s) partagé(s) sur le Web. Il permet le partage de documents entre les membres d'un groupe à travers un serveur Web enrichi d'une extension *BSCW*. Il s'agit d'un collecticiel essentiellement asynchrone et centralisé, qui repose entièrement sur le standard Web. Selon Appelt [Appelt 2001], la métaphore centrale de ce système est celle d'un espace de travail partagé (« *shared workspace* »). Un tel espace est capable de contenir différentes sortes d'information, telles que des documents, des images, des liens vers d'autres pages Web, des discussions, etc. Le contenu d'un espace de travail est représenté comme un ensemble d'objets organisés dans une hiérarchie de répertoires. Un serveur BSCW gère donc un certain nombre d'espaces de travail partagés, c'est-à-dire un certain nombre de répertoires pour les informations partagées, qui sont alors accessibles aux membres du groupe à travers un schéma d'identification simple (nom d'utilisateur et mot de passe). La Figure 6 présente un exemple d'espace de travail partagé dans le système *BSCW*.

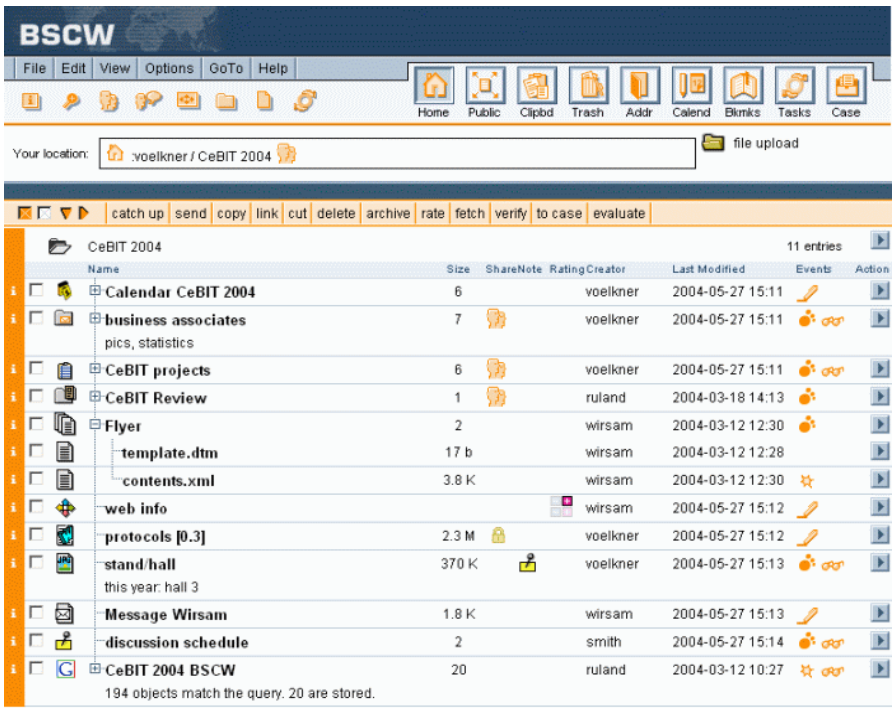


Figure 6. Un espace de travail dans le système BSCW [BSCW 2005].

Le système *BSCW* fournit aux utilisateurs une série de fonctionnalités qui inclut un service d'authentification, un service de forum de discussions, un service de recherche, un service de contrôle de versions, et un service de conscience de groupe (appelé services d'événements – « *event services* »). Ce dernier fournit aux utilisateurs des informations relatives aux activités des autres membres du groupe concernant les objets dans l'espace de travail partagé. Les événements sont déclenchés à chaque fois qu'un utilisateur réalise une action dans l'espace de travail, comme, par exemple, transférer un nouveau document, télécharger (lire) un document existant, etc. [Appelt 2001]. La visualisation de ces événements se fait soit à travers l'interface, dans laquelle une colonne contient des icônes qui informent sur l'existence d'événements selon leur type (voir colonne « *Events* » dans la Figure 6), soit par courrier électronique, selon les préférences préétablies par l'utilisateur.

Appelt [Appelt 2001] a réalisé une analyse de l'utilisation d'un serveur *BSCW* public. Cet auteur a analysé les traces d'utilisation du système pendant 220 jours. Pendant cette

période, les actions d'un ensemble de plus de 15.000 utilisateurs enregistrés ont été séparées en douze groupes d'actions distincts. Parmi ces groupes d'actions, ceux concernant les actions de production (essentiellement lecture, création et modification des documents) sont les plus utilisés par les utilisateurs observés. Juste après, viennent les groupes d'actions concernant la conscience de groupe, lesquels sont, par ailleurs, très prisés des utilisateurs réguliers.

Dans la même lignée que *BSCW*, *ToxicFarm* [Skaf-Molli 2003] est un système de répertoire(s) partagé(s) conçu pour les groupes virtuels. Dans *ToxicFarm*, il existe deux types d'espace de travail : les espaces communs au groupe, et ceux privés des utilisateurs, puisqu'un utilisateur peut créer un espace de travail local à sa propre machine. L'utilisateur récupère d'abord les objets de l'espace commun dans un espace privé, dans lequel il effectuera les modifications souhaitables, avant de remettre dans l'espace commun les nouvelles versions. Cependant, contrairement à d'autres systèmes, *ToxicFarm* procède au contrôle de versions au niveau de l'espace de travail : l'utilisateur copie, met à jour et valide (*commit*) tout un espace de travail, et non pas un fichier isolé [Skaf-Molli 2003].

Le système *ToxicFarm* [Skaf-Molli 2003] fournit un ensemble de services, parmi lesquels des services de communication (messages instantanés, forums, listes de discussion), des services de coordination, basés à la fois sur les « *to-do lists* » et sur un système de gestion de *workflow*, et des services de conscience de groupe, particulièrement la *conscience de l'état,* qui consiste à informer les membres du groupe sur l'état des objets partagés par le groupe [Skaf-Molli 2003]. Une particularité de *ToxicFarm* est l'utilisation des *services Web* pour l'implémentation des services offerts aux membres du groupe. L'utilisation de services Web pour l'implémentation de nouveaux systèmes sur le Web est une tendance prononcée de nos jours. Un service Web est défini comme un logiciel identifié par une URI, dont les interfaces publiques sont définies et décrites en XML. Cette définition des interfaces peut être découverte par d'autres logiciels. Ces logiciels peuvent alors interagir avec le service Web de la manière prescrite en utilisant des messages basés sur XML transmis au moyen de protocoles Internet[25] [Austin 2004]. Selon Ferris et Farrel [Ferris 2003], les bénéfices des services Web incluent la séparation entre l'interface des services (qui est décrite à l'aide du langage WSDL[26], un langage basé sur XML) et les considérations concernant leurs implémentations et leurs plates-formes, ce qui permet l'affectation dynamique des services et l'accroissement de l'interopérabilité entre les langages et entre les différentes plates-formes.

Une autre caractéristique intéressante de *ToxicFarm* est le fait que les modifications sur l'espace de travail sont réalisées "en local", et non pas sur le serveur. En effet, les systèmes de répertoire partagé souvent ne manipulent pas directement le contenu des objets partagés. Par exemple, un groupe peut partager un document, mais l'édition proprement dite du document sera réalisée localement (dans la machine client) par chaque membre du groupe, à travers l'éditeur de texte de son choix.

Ce principe, selon lequel les informations (c'est-à-dire les objets partagés) sont stockés de manière centralisé (sur le serveur) mais leur traitement est réalisé du côté client, est appelé par Decouchant *et al.* [Decouchant 2001] « *architecture hybride stockage/traitement* »[27]. Selon ces auteurs, ce principe est également adopté par le protocole *WebDAV*. Le protocole WebDAV est une extension du protocole HTTP pour l'édition coopérative de ressources Web [Goland 1999]. Adoptant ce principe, le protocole WebDAV prévoit l'utilisation de verrous pour l'accès concurrent aux ressources dans le cadre d'un processus d'édition coopérative. Il

[25]De l'original en anglais, « *A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols* » [Austin 2004].

[26]« *Web Services Description Language* » - http://www.w3.org/TR/wsdl.

[27]De l'original en anglais, « *'hybrid storage/processing architecture' principle* » [Decouchant 2001].

introduit de nouvelles méthodes dans le protocole HTTP, parmi lesquelles deux méthodes pour la gestion des verrous : la méthode LOCK pour poser un verrou sur une ressource, et la méthode UNLOCK pour enlever un verrou. Ainsi, le protocole WebDAV défend l'idée selon laquelle pour modifier une ressource il faut d'abord la verrouiller (méthode LOCK), puis la charger localement (méthode GET traditionnelle au HTTP), et seulement ensuite procéder aux modifications localement. Une fois modifiée, la ressource est copiée vers le serveur (méthode PUT) et déverrouillée.

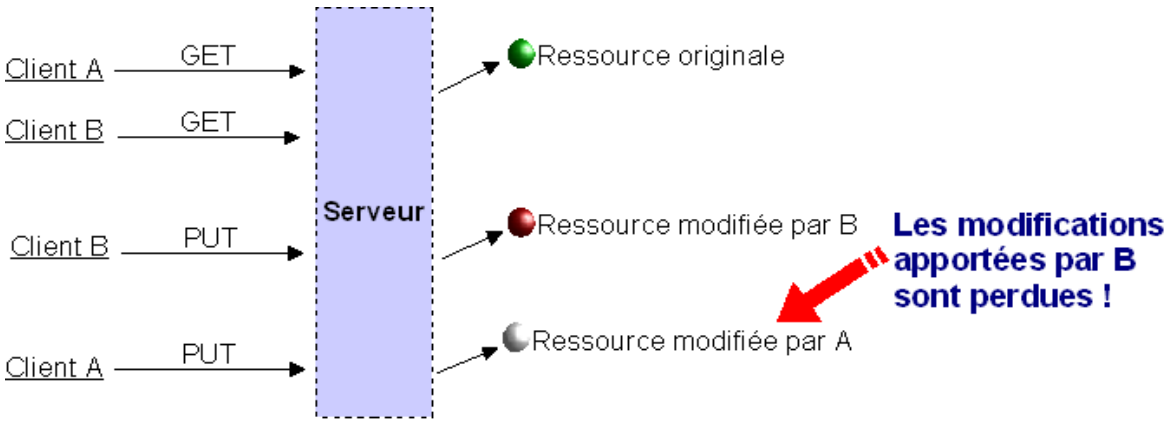


Figure 7. Problème de la mise à jour perdue sur le Web.

L'utilisation de la séquence LOCK-GET-PUT-UNLOCK permet d'éviter un problème très connu en bases de données et dans l'édition coopérative sur le Web : le *problème de la mise à jour perdue*. Le problème de la mise à jour perdue survient lorsque les modifications proposées par un utilisateur sont écrasées par celles d'un deuxième utilisateur sans que celui-ci puisse se rendre compte des modifications apportées par le premier. Prenons l'exemple présenté dans la Figure 7. Dans cet exemple, un utilisateur, utilisant le *client A*, et un second utilisateur, utilisant le *client B,* désirent modifier une même ressource. Le *client A* récupère la ressource (il fait un GET) et la change localement. Pendant cette période, le *client B* récupère la même ressource (deuxième GET) et la change également. Cependant, le *client B* termine ses modifications avant le *client A*, et *B* dépose ses modifications sur le serveur (il fait un PUT). Ce n'est qu'après cela que le *client A* dépose ses propres modifications (second PUT). Au moment où le *client A* dépose sa nouvelle version de la ressource, les modifications réalisées par le *client B* sont perdues, écrasées par celles du *client A*. Or, l'utilisation de la séquence LOCK-GET-PUT-UNLOCK, qui pouvait prévenir ce problème, n'est pas obligatoire pour tous les clients et les serveurs utilisant le protocole WebDAV, pour des raisons de compatibilité avec HTTP (voir [Goland 1999]). Le protocole WebDAV se veut une infrastructure standard pour la rédaction en collaboration asynchrone sur Internet. C'est donc l'application qui l'utilise qui doit faire respecter cette séquence. La définition d'une véritable politique de contrôle d'accès concurrent revient donc toujours aux collecticiels, même s'ils utilisent le protocole WebDAV.

Néanmoins, l'utilisation du protocole WebDAV ne représente pas la seule alternative pour l'édition coopérative sur le Web. Plusieurs collecticiels ne l'utilisent pas. C'est le cas de l'éditeur *AllianceWeb* [Salcedo 1998]. *AllianceWeb* permet l'édition de documents structurés sur le Web. À travers cet éditeur, plusieurs coauteurs peuvent travailler sur un même document, chacun modifiant différentes "régions" (fragments) du document, et ceci de manière asynchrone, et même déconnectée (voir [Salcedo 1998]). Au contraire des exemples précédemment présentés, *AllianceWeb* utilise une architecture répartie, à travers l'utilisation de l'intergiciel *Piñas,* lequel gère la distribution des messages et des documents qui sont

répartis et souvent répliqués sur les différents sites des participants [Decouchant 2001] [Martínez 2002b]. Ceci fait de l'éditeur *AllianceWeb* un exemple intéressant d'utilisation du Web sans pour autant que l'architecture centralisée du Web soit respectée. Dans l'éditeur *AllianceWeb*, il n'existe pas de serveur unique, chaque client se comporte comme un serveur Web. Par contre, ceci exige l'installation de l'application sur chaque machine client, ce qui n'est pas le cas, généralement, des collecticiels centralisés qui utilisent pour la plupart les navigateurs Web standard, sans exiger d'autre installation côté client.

En ce qui concerne la conscience de groupe, l'éditeur *AllianceWeb* présente un ensemble limité d'informations, comprenant essentiellement celles concernant les rôles joués par chaque participant vis-à-vis du document. Ce collecticiel permet à chaque membre du groupe de jouer un rôle effectif (manager, auteur, lecteur, ou nul) sur un fragment du document (voir [Salcedo 1998]). Les rôles effectivement joués sont indiqués dans l'interface par des icônes spécifiques à chaque rôle (Figure 8) qui indiquent aussi les préférences de l'utilisateur par rapport à la fréquence des mises à jours (si l'utilisateur souhaite que les modifications apportées par les autres participants soient immédiatement reportées sur son texte). Ces préférences peuvent être modifiées de manière automatique par un moteur d'inférence adaptatif (« *adaptative inference engine* ») dont l'objectif est de faciliter la coopération entre les coauteurs (voir [Martínez 2002b]).

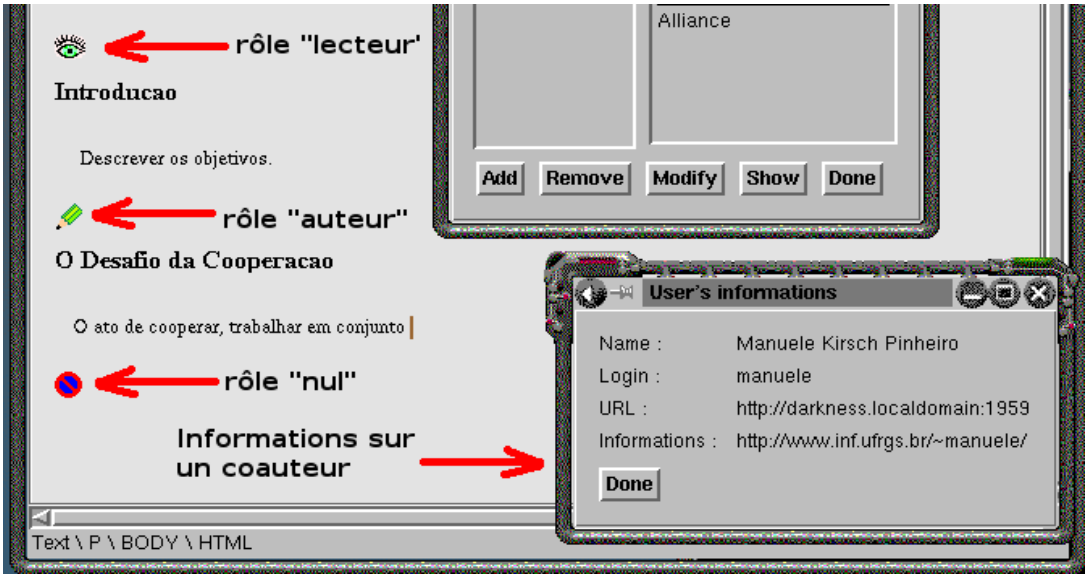


Figure 8. Indicateurs des rôles effectifs dans l'éditeur AllianceWeb.

Tous les exemples ci-dessus mettent en relief les interactions asynchrones entre les membres du groupe. Ces interactions sont facilitées par la structure même du Web, mais celle-ci n'est pas la seule possibilité. Il existe aussi des collecticiels sur le Web qui supportent les interactions synchrones. Parmi ces systèmes, le système *CoVitesse* [Laurillau 2002] permet la navigation collaborative synchrone sur l'espace de réponses à une requête formulée auprès d'un moteur de recherche. *CoVitesse* organise les résultats du moteur de recherche sous la forme d'un agencement de polygones (chaque polygone représente une réponse), et les utilisateurs sont représentés chacun par un avatar qui se déplace dès que l'utilisateur sélectionne une réponse. Les utilisateurs disposent également d'un *chat* afin de communiquer avec les autres participants pendant le travail de recherche. Dans ce système, le support à la conscience de groupe se limite notamment à la conscience de l'espace de travail, à travers

principalement l'utilisation des avatars qui fonctionnent comme un mécanisme de télépointage.

2.4 Conclusions

Dans ce chapitre, nous avons présenté une vue globale du domaine du TCAO, en portant une attention toute particulière à la question de la conscience de groupe et aux collecticiels sur le Web. Nous avons vu que les mécanismes de conscience de groupe peuvent conduire à un problème de surcharge cognitive qui se produit lorsque les utilisateurs sont confrontés à une masse trop importante d'informations. Les concepteurs de ces mécanismes doivent donc prévoir des moyens de prévenir ce problème, soit à travers des techniques de visualisation de l'information (une visualisation périphérique, etc.), soit à travers le filtrage de l'information en entrée (lors de la production de l'information) ou en sortie (juste avant sa visualisation). Nous avons vu également que l'utilisation du Web, en tant qu'infrastructure de base ou en tant qu'espace de travail, croît constamment, notamment grâce à son accessibilité.

L'utilisation du Web concerne particulièrement les collecticiels asynchrones qui supportent les interactions en des lieux différents. En raison de la structure et de l'étendue du Web, celui-ci représente une opportunité non négligeable pour ces systèmes. Ces collecticiels sont également demandeurs de mécanismes de conscience de groupe performants, car ce n'est qu'à travers ces mécanismes que les participants seront capables de percevoir le groupe et ses activités, puisque les membres du groupe ne travailleront ni aux mêmes lieux, ni aux mêmes moments.

Cependant, le cadre du travail coopératif sur le Web doit intégrer un nouvel aspect important : la mobilité. L'utilisation des nouvelles technologies, telles que les ordinateurs de poche et les téléphones cellulaires, a donné plus de liberté aux utilisateurs qui peuvent désormais se déplacer librement tout en accédant au Web dès lors qu'une connexion de réseau est disponible. Elles constituent donc autant d'opportunités pour le travail coopératif, qui peut désormais s'affranchir de l'environnement fixe traditionnel du bureau. Dans le prochain chapitre, nous discutons des enjeux liés à l'utilisation des technologies mobiles et les nouvelles thématiques de recherches qui s'en intéressent.

3 Mobilité et Informatique Sensible au Contexte

Dans ce chapitre, nous présentons les principes de l'informatique sensible au contexte. Nous introduisons cette approche de l'informatique (section 3.1), ainsi que la notion d'utilisateur nomade (section 3.2) et la notion de contexte (section 3.3). Nous abordons plusieurs définitions pour cette notion (section 3.3.1), ainsi que quelques représentations proposées dans la littérature (section 3.3.2) et décrivons quelques approches pour l'acquisition des éléments (section 3.3.3). Nous présentons également quelques applications sensibles au contexte proposées à ce jour (section 3.4), et discutons nos conclusions sur cette thématique (section 3.5).

3.1 L'approche

L'informatique sensible au contexte est apparue au début des années quatre-vingt-dix à travers notamment les travaux de Schilit et Theimer [Schilit 1994a] et Schilit *et al.* [Schilit 1994b]. Ces travaux définissent la *sensibilité au contexte*[28] comme la capacité d'une application à découvrir et à réagir aux modifications dans l'environnement où se trouve l'utilisateur [Schilit 1994a]. Les *systèmes sensibles au contexte*[29] sont définis comme des systèmes qui s'adaptent à la localisation de l'utilisateur et à l'ensemble des personnes, des machines, et des dispositifs proches ou accessibles, ainsi qu'aux changements dans le temps de ces éléments [Schilit 1994b]. La prémisse de l'informatique sensible au contexte est donc d'être conscient des circonstances dans lesquelles se trouve l'utilisateur et d'être capable d'interpréter les interactions de manière appropriée à ces circonstances [O'Hare 2002].

Depuis les premiers travaux, la définition de la sensibilité au contexte ou des systèmes sensibles au contexte n'a pas beaucoup évolué, restant très attachée à une prise en compte de l'environnement de l'utilisateur. On retrouve néanmoins, dans la littérature, plusieurs définitions, plus ou moins proches. Cheverest *et al.* [Cheverest 2002], par exemple, considèrent que les systèmes sensibles au contexte sont des *systèmes qui peuvent adapter la présentation des informations afin de les rendre plus pertinentes au contexte d'un utilisateur* en tenant compte, par exemple, des intérêts signalés par l'utilisateur ou de son contexte environnant. D'autre part, Chaari *et al.* [Chaari 2004] étendent la possibilité d'adaptation au delà de la présentation, en définissant la sensibilité au contexte comme *la capacité d'un système à percevoir la situation dans laquelle se trouve l'utilisateur et d'adapter en conséquence le comportement du système en termes de services, de données et d'interface*. Finalement, Dey [Dey 2000] présente une définition encore plus générale, en considérant qu'*un système est sensible au contexte s'il utilise le contexte pour fournir à l'utilisateur des informations ou des services pertinents, où la pertinence dépend de la tâche exécutée par l'utilisateur*[30].

On constate, dans les définitions ci-dessus, l'importance de la notion de contexte d'utilisation pour les systèmes sensibles au contexte. Pour Dourish [Dourish 2004], la notion de contexte est centrale à l'informatique sensible au contexte, plus que pour d'autres thématiques de recherche en informatique qui utilisent cette notion (comme la recherche d'information), car les interactions dans un système sensible au contexte se trouvent dans des

[28]En anglais, « *context-awareness* ».
[29]En anglais, « *context-aware systems* ».
[30]De l'original en anglais, « *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task* » [Dey 2000].

contextes d'utilisation très variés et souvent distincts de l'environnement du bureau mieux maîtrisé. C'est le contexte qui guide tout le comportement d'un tel système, ainsi que les informations qu'il fournit.

Dans la pratique, plusieurs travaux adoptent une approche limitée de la notion de contexte, tels que celui de Lemlouma et Layaïda [Lemlouma 2004b] qui voient le contexte comme l'ensemble des informations qui sont en relation directe avec la requête d'un document. D'autres, comme Burrell *et al.* [Burrell 2002] et Rubinsztejn *et al.* [Rubinsztejn 2004], ne considèrent essentiellement que la localisation de l'utilisateur. Cependant, même si plusieurs systèmes se limitent à ceci, il est important d'observer que la sensibilité au contexte est plus que la conscience de l'identité de l'utilisateur et de sa localisation [Grudin 2001]. Schilit *et al.* [Schilit 1994b], par exemple, soulignent trois aspects du contexte : où se trouve l'utilisateur, qui est avec lui et quelles sont les ressources proches. Ces auteurs remarquent toutefois que la notion de contexte englobe d'autres éléments, tels que les bruits environnants, la connexion réseau, la situation sociale, etc. La notion de contexte correspond à un concept très large qui est étudié par plusieurs domaines de recherche, et que nous abordons dans la section 3.3. Pour l'instant, il est important de garder à l'esprit que le contexte peut être vu comme une collection des caractéristiques de l'environnement physique ou virtuel qui affecte le comportement d'une application [Benerecetti 2001].

Le qualificatif de « sensible au contexte » est donc associé aux systèmes guidés principalement par le contexte d'utilisation. Ces systèmes ont tendance à être également mobiles, puisque c'est dans ce cas que les changements dans le contexte d'utilisation arrivent le plus souvent. L'intérêt pour l'informatique sensible au contexte est très lié à l'évolution et au développement des nouvelles technologies mobiles. Selon Dourish [Dourish 2004], lorsque l'informatique est sortie de l'environnement du bureau (fixe) traditionnel, il est devenu important de suivre à la trace la situation dans laquelle la technologie est utilisée. Ainsi, c'est dans les systèmes mobiles que la capacité de détecter le contexte semble particulièrement pertinente, puisqu'ils peuvent être utilisés dans une large variété de localisations, par différents utilisateurs avec des objectifs également distincts [Burrell 2002]. Pour Chalmers [ChalmersD 2002], même la présentation des données adaptées au contexte d'utilisation est déjà une nécessité dans les systèmes mobiles. Pour cet auteur, cette adaptation doit traduire les limitations des ressources et les préférences de l'utilisateur liées au contexte.

Les utilisateurs *nomades* sont donc la cible privilégiée (non la seule, mais la plus importante) des technologies et des systèmes dits *sensibles au contexte.* À partir du moment où l'utilisateur se trouve en situation de mobilité, et donc passible d'être soumis à un environnement contraignant, l'adaptation devient une nécessité. Certains auteurs, tel que Chaari *et al.* [Chaari 2004], considèrent la *sensibilité au contexte* comme la capacité de percevoir la situation de l'utilisateur en plusieurs aspects, et d'adapter en conséquence le comportement du système, et l'*adaptation* comme la capacité de fournir différentes versions d'un service ou différentes présentations d'un document afin de satisfaire les besoins de l'utilisateur, les contraintes de l'environnement, du dispositif, etc. Ainsi, pour ces auteurs, il existe une relation claire entre ces deux concepts, le second étant l'objectif du premier. En généralisant, on peut affirmer que l'objectif ultime de l'informatique sensible au contexte est de concevoir des *systèmes capables de s'adapter au contexte d'utilisation de chaque utilisateur et de lui fournir la bonne information au bon moment.*

L'informatique sensible au contexte est donc liée à la question de l'adaptation. On voit apparaître, dès les premiers travaux, une adaptation, surtout liée aux utilisateurs nomades et aux limitations des dispositifs mobiles et des réseaux sans fil (voir [Schilit 1994b]). On la voit également liée à la localisation de l'utilisateur et au filtrage des informations selon le contexte (voir, par exemple, [Schilit 1994a]). Ces aspects de l'adaptation (adaptation à la localisation et

aux technologies mobiles) sont toujours observables dans les travaux ultérieurs, tel que Espinoza *et al.* [Espinoza 2001], Banerjee *et al.* [Banerjee 2002], Lemlouma [Lemlouma 2004a] ou encore Rubinsztej *et al.* [Rubinsztejn 2004]. Dans la plupart de ces travaux, on cherche à adapter soit le contenu, soit la présentation de ce contenu, à la localisation et au dispositif client employé par l'utilisateur. Ce qui différencie ces travaux est l'approche adoptée par chacun, qui s'étend de la proposition des protocoles comme dans [Lemlouma 2004a], à la proposition d'une architecture pour les systèmes sensibles au contexte ([Banerjee 2002] ou [Rubinsztejn 2004]).

Ainsi, on peut considérer que le vrai challenge de l'informatique sensible au contexte est d'explorer l'environnement dynamique de l'utilisateur au sein des systèmes conscients du contexte dans lequel ils sont utilisés [Schilit 1994b]. Ce défi cache plusieurs problèmes, parmi lesquelles on peut souligner : (*i*) l'*adaptation* du système (données et services) au contexte ; (*ii*) l'*acquisition* des données contextuelles ; et (*iii*) la *représentation* de l'information contextuelle dans son ensemble. De manière générale, on peut considérer que, pour qu'un système soit sensible au contexte, il lui faut trois étapes de base [Chaari 2004] : *1*) capturer les données contextuelles (données de bas niveau) à partir des capteurs; *2*) interpréter les données capturées afin de construire l'information contextuelle (haut niveau) pertinente pour le système; et *3*) délivrer cette information au système. Pour réussir ces trois étapes, il faut donc répondre à ces questions : « *qu'est-ce que le système doit adapter et comment* ? », « *quels éléments du contexte de l'utilisateur doivent être détectés et comment* ? », et « *comment représenter, à l'intérieur du système, ce contexte ?* ».

Par ailleurs, Grudin [Grudin 2001] a soulevé trois problèmes qui concernent plus particulièrement l'acquisition du contexte : (*i*) le respect de la vie privée ; (*ii*) la précision des données capturées ; et (*iii*) l'interprétation de ces données. Le respect de la vie privée est un problème qui survient lorsque les informations contextuelles concernant un individu deviennent publiques. Par exemple, certains travaux (voir [Grudin 2001], [Schilit 1994a], et [Weiser 1993]) ont observé que, souvent, les utilisateurs trouvent gênant que les informations concernant leur localisation ou leurs déplacements soient accessibles à d'autres utilisateurs, même si ceux-ci sont des collègues de travail. La précision des données concerne essentiellement les méthodes de détection et les problèmes qui peuvent survenir lorsque le contexte n'est pas détecté de façon précise ou l'est de façon erronée. Puisque les systèmes sensibles au contexte utilisent cette information contextuelle pour s'adapter, si cette information n'est pas suffisamment précise ou est erronée, l'adaptation réalisée par le système pourra fournir des résultats qui ne correspondent pas à la réalité de l'utilisateur, ni à ses besoins. Finalement, les systèmes sensibles au contexte doivent savoir comment interpréter les données brutes qui sont extraites la plupart de temps des capteurs (par exemple, un GPS, un capteur de température...), afin que ces données deviennent effectivement une information contextuelle capable d'être utilisée par le système.

Pour O'Hare et O'Grady [O'Hare 2002], identifier le contexte de l'utilisateur à un moment donné avec un degré de certitude raisonnable peut s'avérer une tâche difficile, voire même impossible. Et même si on suppose que le contexte d'utilisation a pu être déterminé, identifier quelle action doit être menée peut être également difficile. Ces auteurs citent l'exemple de la localisation. Les dispositifs de type GPS suivent normalement des standards qui facilitent la capture de ces données. Toutefois, ces auteurs soulignent qu'interpréter ces données est un processus plus subtil. Transformer une série de coordonnées dans un système de coordonnées précis en quelque chose de compréhensible pour un être humain (comme le nom d'une rue ou d'un quartier) est un processus plus délicat et qui peut s'avérer difficile selon le type de données dont on dispose. La sensibilité au contexte est plus qu'une question d'acquisition des données contextuelles sur une situation complexe. Plus d'information n'est

pas nécessairement plus avantageux : l'information contextuelle n'est utile que lorsqu'elle peut être interprétée d'une manière également utile [Moran 2001].

Ensuite, une fois que l'information sur le contexte d'utilisation est détectée et interprétée, elle est délivrée au système pour être exploitée. À partir de ce moment, Dey [Dey 2001] a identifié trois manières possibles d'exploiter cette information : (*i*) la présentation des informations et des services à l'utilisateur ; (*ii*) l'exécution automatique d'un service pour un utilisateur selon le contexte ; et (*iii*) l'étiquetage d'une information (contenu informationnel) avec des données contextuelles afin de permettre la recherche et la récupération de cette information ultérieurement. Dans le premier cas, on a, par exemple, la présentation, dans un musée, des informations sur une toile de maître devant laquelle s'est arrêté l'utilisateur (voir, par exemple, [Banerjee 2002]). Dans le deuxième cas, on retrouve les systèmes qui envoient, sans que l'utilisateur le demande, des informations considérées comme pertinentes pour le contexte courant de l'utilisateur, tels que le plan de la région ou encore les promotions sur un magasin qui est proche de l'utilisateur. Dans le troisième cas, on a les travaux du type « *geonotes* », qui permettent l'association de l'information sur une localisation donnée et d'une annotation faite par un utilisateur (voir, par exemple, [Burrell 2002] et [Espinoza 2001]).

De manière similaire à Dey, Dourish [Dourish 2004] défend également que l'information sur le contexte est exploitée principalement : (*i*) pour combiner l'information contextuelle avec le contenu informationnel du système afin de l'utiliser ultérieurement pour la recherche et la récupération du contenu (ce qui rejoint le troisième point énuméré par Dey) ; et (*ii*) pour utiliser l'information du contexte afin d'adapter dynamiquement le comportement ou les réponses du système à certains patrons d'utilisation (ce qui englobe les deux premiers points cités dans le paragraphe ci-dessus). Pour cet auteur, les systèmes sensibles au contexte utilisent le plus souvent la notion de contexte pour la réalisation de requêtes en association avec d'autres informations (par exemple, l'utilisation de la localisation comme paramètre de la requête), afin d'adapter le comportement et la réponse du système à des patrons d'utilisation (par exemple, un utilisateur sur un dispositif donné).

Pour sa part, Chalmers [ChalmersD 2002] aborde la question de l'utilisation de l'information contextuelle de manière plus large, en proposant cinq usages distincts pour cette information : (*i*) la *détection du contexte*, dans laquelle les informations sur le contexte (la localisation, la température, etc.) sont simplement détectées et délivrées à l'utilisateur ; (*ii*) l'*augmentation contextuelle*, dans laquelle le contexte est associé à certaines données comme, par exemple, l'association entre un registre sur un objet physique et la localisation de l'objet, ou encore l'association entre d'un côté les notes concernant une réunion et les participants et de l'autre le lieu de la réunion ; (*iii*) la *découverte des ressources* selon le contexte (par exemple, imprimer dans l'imprimante la plus proche) ; (*iv*) le *déclenchement des actions* par le contexte, comme, par exemple, charger automatiquement le plan de la région dans laquelle on vient d'entrer ; (*v*) la *médiation contextuelle* qui correspond à l'utilisation du contexte pour modifier un service (par exemple, décrire les contraintes et les préférences sur un ensemble des données afin de présenter les parties les plus appropriées).

À partir des analyses ci-dessus, nous identifions six usages possibles pour la notion de contexte :

a) présenter à l'utilisateur les informations sur son contexte courant (sa localisation, la température de l'environnement, etc.) ;

b) délivrer le contenu qui soit le plus approprié au contexte d'utilisation (comme montrer automatiquement le plan du quartier où se trouve l'utilisateur) ;

c) associer à une ou à un ensemble de données une information contextuelle précise pour la réalisation *a posteriori* des requêtes (comme associer à une annotation la localisation mentionné par son contenu) ;

d) exécuter automatiquement une action selon le contexte dans lequel se trouve l'utilisateur (par exemple, transférer les appels d'un bureau à un autre selon les déplacements de l'utilisateur) ;

e) découvrir les ressources disponibles selon le contexte (trouver l'imprimante ou le téléphone le plus proche) ;

f) proposer des services spécialisés selon le contexte (par exemple, des services particulièrement proposés à l'utilisateur selon sa localisation ou son dispositif).

Même si les usages possibles pour la notion de contexte ouvrent de nombreuses opportunités, la majorité des systèmes sensibles au contexte ont une utilisation limitée du contexte. Selon Chaari *et al.* [Chaari 2004], plusieurs travaux se limitent à l'analyse de petites quantités d'information contextuelle et présentent des solutions *ad-hoc* à des besoins très spécifiques. Les premiers travaux, tels que [Schilit 1994a], n'ont considéré que la localisation de l'utilisateur et, encore aujourd'hui, plusieurs systèmes ne s'intéressent qu'à cet aspect du contexte (voir, par exemple, [Rubinsztejn 2004]), ou aux aspects liés à l'utilisation des dispositifs mobiles (comme dans [Lemlouma 2004a]).

Néanmoins, selon Chaari *et al.* [Chaari 2004], certains travaux traitent désormais de la manière dont le contexte est détecté et délivré à un système (voir, par exemple, [Rey 2004]). En revanche, ces auteurs observent qu'il n'existe pas, à ce jour, de réponse satisfaisante à la question de l'adaptation, de comment un système peut s'adapter au contexte d'utilisation, ou encore à l'impact de l'utilisation du contexte dans les systèmes vis-à-vis des utilisateurs et des coûts de production. De surcroît, la majorité des travaux portent sur des applications très spécifiques (voir, par exemple, [Burrell 2002]), et les solutions génériques ne sont pas encore disponibles. Par conséquent, les concepteurs d'un système qui souhaitent lui ajouter des fonctionnalités sensibles au contexte doivent faire face à une large gamme d'alternatives et de techniques, ce qui fait du choix de la plus utile un véritable problème [Mostéfaoui 2004]. Ceci a un impact négatif sur le développement des systèmes sensibles au contexte car la complexité de la conception de ces systèmes est largement amplifiée par le nombre réduit d'outils génériques capables d'aider les concepteurs.

En somme, la conception des systèmes sensibles au contexte passe obligatoirement par une utilisation des informations contextuelles pour l'adaptation du système, soit en termes de données fournies, soit en termes de services proposés, en fonction du contexte d'utilisation. Cette capacité d'utiliser les informations contextuelles pour adapter, de manière automatique, le comportement d'un système à un environnement dynamique serait, selon Benerecetti *et al.* [Benerecetti 2001], l'avantage majeur des systèmes sensibles au contexte. Cependant, pour cela, il faut comprendre la notion de contexte et celle d'utilisateur nomade, pour lequel on va généralement observer le contexte. Les prochaines sections discutent ces deux notions.

3.2 L'utilisateur nomade

Il n'existe pas de véritable consensus autour de la définition de l'utilisateur nomade, et ce malgré l'importance de ce concept pour l'informatique mobile. Cependant, une certaine perception commune de ce que représente cette notion d'utilisateur nomade peut être notée. Il

est clair à ce jour que, grâce aux technologies mobiles, les utilisateurs ne sont plus contraints d'accéder à un système quelconque à travers un dispositif unique (l'ordinateur de bureau) ou à partir d'une localisation prédéterminée (le bureau, la maison...). Les utilisateurs peuvent actuellement alterner entre différents dispositifs (ordinateur de poche, téléphone cellulaire, etc.) et diverses localisations (à partir de chez eux, d'une gare, d'un restaurant, etc.), même de manière continue. La notion d'utilisateur nomade se réfère à ces utilisateurs qui peuvent se déplacer et changer de dispositif à tout moment.

Un « *utilisateur nomade* » est donc un utilisateur qui est *capable* de se *déplacer* librement, d'accéder à un système à partir de différentes *localisations*, en utilisant des *dispositifs* distincts dont il peut changer à tout moment. Enfin, un utilisateur qui est *capable* de *changer* son *contexte d'utilisation* tout en accédant au système. Un utilisateur nomade n'est pas simplement quelqu'un qui est muni d'un ordinateur ou dispositif mobile, ou quelqu'un qui est constamment en déplacement (comme un visiteur dans un musée). C'est aussi et surtout un utilisateur qui dispose de plusieurs options en termes de dispositif et d'accès au réseau, qui peut se déplacer tout en utilisant ces options (pas toujours les mêmes). C'est donc quelqu'un qui peut être dans une situation de mobilité à tout moment, mais qui ne l'est pas forcément à un moment donné. Ainsi, un utilisateur qui travaille à un instant *t* sur son poste fixe, mais qui peut prendre son ordinateur portable et continuer son travail dans une autre salle (par exemple, dans le bureau d'un collègue ou chez lui), peut être considéré comme un utilisateur nomade. À travers cette notion, on observe notamment la *liberté* dont un utilisateur nomade dispose. C'est cette liberté de mouvement et de moyens (et d'action) qui différencie pour nous un utilisateur "traditionnel" d'un utilisateur nomade.

Les études menées par Tang *et al.* [Tang 2001] ont dégagé des points communs entre les utilisateurs nomades : *(i)* ils ont besoin d'accéder à leurs informations depuis des endroits différents ; *(ii)* ils utilisent des dispositifs mobiles généralement de façon concurrente ou en relation les uns avec les autres; *(iii)* ils exigent des temps de réponse appropriés et des applications efficacement conçues.

À partir de ces études, on observe que l'utilisation concurrente et coordonnée de plusieurs dispositifs est fréquente. Celle-ci est influencée par le contexte d'utilisation. Par exemple, selon la situation où se trouve un utilisateur (debout dans un métro, ou assis dans un aéroport...), il va probablement choisir différents dispositifs (téléphone cellulaire, ou ordinateur portable...), en cherchant toujours celui qui est le plus approprié à la situation (par rapport au temps disponible, au confort, à la tâche à réaliser, etc.). De plus, les études démontrent que les utilisateurs nomades désirent des informations et des applications qui soient appropriées à leur mobilité. Ceci implique la conception de systèmes qui puissent s'adapter au contexte d'utilisation. Cette adaptation ne doit pas se limiter à la présentation des informations selon les capacités des dispositifs, mais doit également inclure le choix du contenu à présenter, puisque l'utilisateur nomade ne s'intéresse qu'à certaines informations prioritaires ou directement liées à son travail et son contexte courant. Hibino et Mockus [Hibino 2002] constatent, par exemple, que les utilisateurs nomades munis d'un ordinateur de poche ne sont habituellement pas intéressés par la lecture en détail de tous leurs messages, mais plutôt par la lecture des messages prioritaires.

Ce désir d'adaptation est également présent chez les utilisateurs traditionnels, cependant, ce problème est ici accentué en raison des conditions dans lesquelles un utilisateur nomade peut se trouver lorsqu'il utilise le système, des contraintes physiques souvent liées aux capacités des dispositifs et des réseaux, ainsi qu'en raison de son environnement (s'il est dans un théâtre, dans la rue, chez lui, etc.). En d'autres termes, la question de l'adaptation devient la pierre angulaire de l'acceptation des systèmes par les utilisateurs nomades.

Par ailleurs, à l'aide des nouvelles technologies, le travail coopératif peut désormais prendre place dans différents contextes et environnements, même si les postes de travail fixes ou les ordinateurs portables constituent toujours le contexte le plus répandu. Grâce à la démocratisation des dispositifs mobiles, les collecticiels doivent permettre aux utilisateurs de réaliser une tâche coopérative à partir de leurs postes de travail respectifs, qui peuvent être désormais des postes informatiques fixes ou portables, des ordinateurs de poche, des téléphones cellulaires, ou des installations spécialisées (par exemple, des salles de vidéoconférences, centres concentrateurs pour télétravailleurs, etc.) [David 2001]. Ainsi, pour accompagner les utilisateurs dans leurs activités mobiles, ces nouveaux collecticiels "mobiles" devront allier les caractéristiques des systèmes mobiles aux fonctionnalités des collecticiels [Renevier 2004].

Cependant, selon Renevier [Renevier 2004] et Canals *et al.* [Canals 2002], il faut désormais réfléchir en termes de lieux d'interactions et ainsi organiser les collecticiels en deux classes : *confiné* et *vagabond*. L'utilisation *confinée* d'un collecticiel est une utilisation qui se déroule dans un seul ou dans quelques lieux particuliers. Une zone est donc dite confinée si elle est délimitée et bien cernée : cela peut être un bureau, un bâtiment, un ensemble de bâtiments, un site[31]. En revanche, une utilisation sera dite *vagabonde* si les interactions entre les utilisateurs à travers le système peuvent se produire n'importe où.

Cette notion d'utilisation confinée ou vagabonde a des incidences sur la conception des collecticiels. Un utilisateur dans une condition d'utilisation vagabonde est susceptible d'utiliser le système à partir d'une gamme plus vaste et potentiellement imprévisible de contextes d'utilisation qu'un utilisateur dans une condition d'utilisation confinée. Par conséquent, les systèmes doivent tenir compte de cette variété pour relever le défi d'adaptation posé par les utilisateurs nomades.

3.3 Notion de contexte

Afin d'utiliser efficacement la notion de contexte, il faut comprendre ce qu'est le contexte et comment l'utiliser. Définir ce qu'on entend par contexte est une question complexe, puisque la notion de contexte est utilisée par plusieurs domaines de recherche (et pas seulement l'informatique) qui l'appréhendent sous différentes perspectives. Cette importante question est abordée par certains auteurs (voir, par exemple, [Coutaz 2003] et [Dourish 2004]), mais elle est paradoxalement souvent négligée par les travaux spécifiques à l'informatique sensible au contexte. Par ailleurs, afin d'utiliser le contexte dans un système, il faut chercher les moyens d'acquérir, d'interpréter et de représenter les informations sur le contexte. On trouve dans la littérature quelques propositions qui cherchent à répondre à ces problèmes. Ces propositions ne sont pas très nombreuses à ce jour, surtout lorsqu'on les compare avec le nombre d'applications qui utilisent, d'une manière ou d'une autre, la notion de contexte. Or, afin d'avancer vers une conception plus aisée des systèmes sensibles au contexte, il est nécessaire d'offrir des modèles ou des outils tels que de canevas, qui s'étendent de l'acquisition du contexte à sa représentation à l'intérieur des applications.

Dans les prochaines sections, nous nous intéressons à ces questions. Ainsi, dans la section 3.3.1, nous abordons les définitions pour la notion de contexte. Dans la section 3.3.2, nous discutons les représentations existantes à ce jour, et dans la section 3.3.3, nous considérons l'acquisition du contexte.

[31]Les lieux de la zone ne sont pas forcément proches géographiquement.

3.3.1 Les définitions

Au regard de la littérature, il apparaît clairement qu'il n'existe pas une seule et unique définition pour la notion de contexte. Divers domaines de recherche se sont attelés à la définir, tant à l'intérieur de l'informatique comme l'intelligence artificielle, qu'à l'extérieur avec les recherches dans les sciences humaines. Chacun de ces domaines de recherche propose une ou plusieurs définitions qui conviennent aux recherches menées. En ce qui concerne l'informatique, par exemple, Brézillon pour l'intelligence artificielle [Brézillon 2002a] et Mostéfaoui *et al.* pour l'informatique pervasive [Mostéfaoui 2004] ont analysé certaines de ces définitions. Dans cette section, nous nous concentrons sur les possibles définitions que la notion de contexte peut avoir dans l'informatique sensible au contexte.

On peut constater que les premiers travaux dans l'informatique sensible au contexte ne s'intéressent qu'à moitié à la définition de contexte, à travers notamment une vision particulièrement limitée. Schilit et Theimer [Schilit 1994a], par exemple, ne considèrent que la localisation de l'utilisateur, les personnes qui l'accompagnent, les objets autour, et les changements dans ces éléments. Ces auteurs ne cherchent pas à comprendre la nature du contexte, même si Schilit *et al.* [Schilit 1994b] concèdent que d'autres éléments, comme le niveau de bruit, le coût de communication, ou encore la situation sociale (si on se trouve avec un collègue, par exemple), peuvent faire partie de la notion de contexte. Un autre exemple est donné par Brown *et al*. [Brown 1997], qui considèrent le contexte de l'utilisateur comme formé de sa localisation, les personnes avec lui, du moment de la journée, et ainsi de suite.

On observe alors que les premiers travaux restent plutôt vagues sur la définition de contexte, en accordant plus d'attention aux éléments qui peuvent décrire le contexte d'utilisation d'un système qu'à comprendre sa signification. La notion de contexte est pourtant le concept clé de la thématique de recherche dans laquelle ces travaux se situent. Ce n'est que depuis quelques années que cette question de la définition du contexte a gagné plus d'attention, sans, pour autant, devenir un véritable centre d'intérêt pour les travaux de l'informatique sensible au contexte. Les travaux s'intéressant à cette définition (ou à sa représentation) restent encore très minoritaires par rapport au nombre de travaux liés à l'informatique sensible au contexte.

Cependant, le terme « contexte » est un mot largement utilisé dans le quotidien, ayant donc une signification dans le sens commun des gens. Dans la langue française[32], le mot « contexte » est défini comme « *ce qui accompagne, précède ou suit un texte, l'éclaire* », ou encore comme « *ensemble des circonstances qui accompagnent un événement* » [Larousse 1992]. Cette dernière définition est particulièrement intéressante pour l'informatique sensible au contexte, qui utilise la notion de contexte pour décrire les circonstances dans lesquelles un utilisateur accède à un système. Dans ce cas, on peut parler du « contexte dans lequel les interactions entre un individu et un système ont lieu » [Dourish 2001].

D'une manière plus générale, on peut parler du contexte comme *ce qui entoure le centre d'intérêt de l'individu et qui apporte des informations additionnelles capables d'aider à la compréhension de ce centre d'intérêt* [Mostéfaoui 2004]. On dit donc que, pour un *focus* d'attention donné (par exemple, l'étape courante de la résolution d'un problème), une définition du contexte serait *un ensemble d'éléments qui donnent un sens au focus d'attention sans intervenir explicitement dans celui-ci* [Brézillon 2002b]. Dans cette approche, on peut considérer le contexte comme *un amalgame très structuré des ressources informationnelles, physiques et conceptuelles* qui va au-delà des simples questions de 'qui' ou 'quoi' est 'où' et

[32]On retrouve, en langue anglaise, les mêmes définitions : « *the text or speech that comes immediately before and after a particular phrase or piece of text and helps to explain its meaning* », ou encore « *the situation within which something exists or happens, and that can help explain it* », selon le *Cambridge Dictionary* (http://dictionary.cambridge.org/).

'quand' pour inclure également l'état des ressources numériques, l'état des tâches, les relations sociales et organisationnelles, entre autres [Kirsh 2001]. Ce contexte, autant organisationnel et culturel que physique, joue un rôle critique en fournissant les moyens d'interpréter et de comprendre l'action [Dourish 2001].

En d'autres termes, le contexte dans lequel une action se produit donne du sens à l'action. Il inclurait tout élément capable d'aider la bonne identification et la compréhension de cette action. À partir de ce point de vue, on observe que la notion de contexte ne se limite donc pas aux acteurs qui sont engagés dans une action ou à leur localisation. Le contexte peut inclure des informations concernant aussi bien des aspects physiques, comme la localisation de l'utilisateur, le niveau de bruit de la salle, etc., que sociales ou même organisationnels. Par exemple, une action entreprise par un employé peut avoir une toute autre signification lorsque cet employé joue un rôle important dans son organisation, ou lorsqu'il est accompagné de quelqu'un qui joue un tel rôle.

Ces définitions présentent la notion de contexte comme un ensemble d'informations qui accompagne une action, laquelle, dans le cas de l'informatique sensible au contexte, est une interaction entre l'utilisateur et le système. Cependant, même si on comprend à travers ces définitions ce qui représente la notion de contexte, celles-ci sont encore trop abstraites pour être directement exploitables par ces systèmes. Ainsi, certains auteurs ont préféré une approche plus pragmatique, proposant des définitions plus convenables pour la conception de systèmes sensibles au contexte. Lemlouma [Lemlouma 2004a] définit le contexte comme « *l'ensemble de toutes les informations de l'environnement qui peuvent influencer le processus de l'adaptation et de la transmission du contenu vers l'utilisateur final* ». Chaari *et al.* [Chaari 2005] définissent le contexte comme « *l'ensemble des paramètres externes à l'application pouvant influer sur le comportement d'une application en définissant de nouvelles vues sur ses données et ses services* ». Pour Dey [Dey 2000] *le contexte est construit à partir de tous les éléments d'information qui peuvent être utilisés pour caractériser la situation d'une entité. Une entité correspond ici à toute personne, tout endroit, ou tout objet (en incluant les utilisateurs et les applications eux-mêmes) considéré(e) comme pertinent(e) pour l'interaction entre l'utilisateur et l'application*[33]. Cette définition concerne particulièrement la conception des systèmes sensibles au contexte, puisqu'elle tient compte de la pertinence des éléments pour l'interaction entre l'utilisateur et l'application.

La définition donnée par Dey [Dey 2000] reste générale : elle permet aussi bien l'inclusion des informations explicitement fournies par l'utilisateur à travers l'interface du système, que des informations implicitement fournies, lesquelles nécessitent la réalisation de certains calculs avant d'être effectivement utilisées (par exemple, l'information sur l'état actuel du réseau estimé à partir du nombre d'utilisateurs connectés) [Mostéfaoui 2004]. Néanmoins, cette généralité induit un autre problème : l'identification des éléments qui composent la notion contexte. Pour Chaari *et al.* [Chaari 2005], la définition donnée par Dey ne permet pas d'identifier les données appartenant au contexte, ni de les distinguer des données propres à l'application. Ceci n'empêche pas que cette définition soit désormais la plus utilisée par les travaux liés à l'informatique sensible au contexte et à l'informatique pervasive. La définition donnée par Dey est également celle adoptée par le présent travail, en raison de sa généralité et son implication dans la conception des systèmes.

À partir de cette définition, on se positionne dans une perspective de représentation du contexte. Selon Dourish [Dourish 2004], la notion de contexte dans l'informatique pervasive a deux origines majeures : elle est, d'une part, un aspect technique pour la conception des

[33]De l'original en anglais : « *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves* » [Dey 2000].

systèmes, et, d'autre part, un thème d'étude en collaboration avec les sciences humaines. Dans la perspective des sciences humaines, deux approches distinctes pour son analyse sont considérées : une approche *positiviste* et une approche *phénoménologique*. La première aborde la notion de contexte comme un *problème de représentation*. La seconde considère le contexte comme un *problème d'interaction*. Dans la première approche, on cherche à répondre à la question « *qu'est-ce que le contexte et comment le représenter ?* », tandis que la seconde approche cherche à comprendre « *comment et pourquoi, au cours de leurs interactions, les gens atteignent et entretiennent une compréhension mutuelle du contexte de leurs actions ?* ». Dans ce travail, nous nous concentrons sur la conception de systèmes coopératifs et sensibles au contexte. Il nous paraît essentiel d'adopter la première approche, en traitant la question du contexte comme un *problème de représentation*. Dourish [Dourish 2004] même le concède, en affirmant que *toute proposition visant à prendre en compte la notion de contexte dans le fonctionnement d'un système implique naturellement de considérer la façon dont ce contexte peut être représenté et codé*.

Ainsi, dans cette perspective de représentation, et indépendamment de la définition adoptée, les travaux abordant la notion de contexte partagent, selon Rey et Coutaz [Rey 2004], quatre principes :

(i) Il n'y a *pas de contexte sans contexte*. Autrement dit, la notion de contexte se définit en fonction d'une finalité ;

(ii) Le *contexte est un espace d'informations qui sert l'interprétation*. La capture du contexte n'est pas une fin en soi, mais les données capturées doivent servir un objectif ;

(iii) Le contexte est *un espace d'informations infini et évolutif*. Par conséquent, il n'existe pas a priori de « *contexte absolu* », mais un espace qui se construit au cours du temps ; et

(iv) Le contexte est *un espace d'informations partagé par plusieurs acteurs*. On parle du *contexte d'interaction* qui est l'ensemble des informations que l'utilisateur et le système ont en commun.

À travers les principes énumérés ci-dessus, on observe l'importance d'avoir une compréhension plus générale de la notion de contexte, puisqu'il est impossible de déterminer un seul ensemble d'éléments composant le contexte qui soit valable pour toutes les applications qui l'utilisent. L'impossibilité d'énumérer tous les éléments d'information pertinents pour toutes les situations est évidente, car ces éléments changent inévitablement d'une situation à l'autre [Dey 2001]. Pour Greenberg [Greenberg 2001], plusieurs éléments contribuent au contexte, et leur pertinence est fortement dépendante de la situation en particulier. Même les actions qui sont menées par les personnes, ou encore les réponses qu'elles attendent d'un système sensible au contexte, dépendent de la situation dans laquelle elles se trouvent. En conséquence, chaque système faisant usage de la notion de contexte doit déterminer les éléments qui composent cette notion à partir de ses propres objectifs.

Ce constat est également une conséquence d'une autre caractéristique de la notion de contexte : la récursivité. Selon Brézillon [Brézillon 2002b], le contexte est toujours relatif à un autre contexte plus général. Il ne peut donc être décrit totalement. Pour cet auteur, il faut toujours parler de contexte en référence à quelque chose (*pas de définition de contexte hors contexte*) : le contexte d'un objet, le contexte des interactions... Toutefois, un seul contexte est reconnu d'intérêt commun : le contexte des interactions entre agents (un être humain et un logiciel, un ordinateur ou un objet) car c'est par rapport à celui-ci que les autres sont référencés ou évoluent.

Ainsi, lorsqu'on représente en machine le contexte, on ne représente forcément qu'une partie du contexte, ce qui entraîne un effort supplémentaire de représentation [Grudin 2001]. Cet effort correspond au choix des éléments qui composent le contexte. Greenberg [Greenberg 2001] souligne ce problème en affirmant que, dans tous les cas, à l'exception des plus faciles, les concepteurs auront des difficultés à identifier les informations capables de déterminer, avec plus ou moins de précision, l'état courant d'une entité (une personne, un objet...). Par ailleurs, les capteurs utilisés pour détecter le contexte ne peuvent pas capter tous les types d'information contextuelles : certaines informations sont perdues, d'autres sont acquises. Certains éléments, comme les données de l'environnement physique, sont relativement simples à capturer, tandis que d'autres, comme les buts de l'utilisateur ou l'état de ses activités, sont extrêmement difficiles à capturer [Grudin 2001] [Greenberg 2001].

Par ailleurs, selon Brézillon [Brézillon 2002b], le contexte peut être implicite ou explicite, mais il doit absolument être explicite pour être exploitable et donc communiqué. Pour ce même auteur, à partir du moment où il devient explicite, le contexte doit permettre une organisation dynamique des données, des informations et des connaissances en mémoire aussi bien pour l'extraction des éléments d'une réponse que pour l'acquisition de nouveaux items. D'ailleurs, selon Coutaz *et al.* [Coutaz 2003], il est communément accepté que la notion de contexte concerne un espace d'information structuré et partagé, qui évolue dans le temps. Il est également accepté que cet espace soit conçu pour servir un but particulier. Dans l'informatique pervasive, ce but est d'augmenter les activités humaines avec de nouveaux services qui peuvent s'adapter aux circonstances dans lesquelles ils sont utilisés. Dans l'informatique sensible au contexte, on considère que ce but est l'adaptation d'un système, en termes de données, de présentation, ou encore de services, à des circonstances d'utilisation qui sont décrites par le contexte.

Enfin, plusieurs auteurs dans la littérature, dont Greenberg [Greenberg 2001] et Brézillon [Brézillon 2002b], soulignent l'aspect dynamique et évolutif du contexte, observé également par Rey et Coutaz [Rey 2004]. Pour Greenberg [Greenberg 2001], on doit considérer le contexte comme un espace qui évolue continuellement et qui dépend fortement de la situation. Pour Chaari *et al.* [Chaari 2004], les paramètres qui composent le contexte évoluent durant l'exécution du système. Une nouvelle instance de ces paramètres caractérise un nouveau contexte qui ne modifie pas directement les données du système, mais qui est capable d'influencer son comportement d'une manière ou d'une autre. Ainsi, toute forme de représentation du contexte en machine doit absolument prendre en compte le dynamisme de l'information contextuelle. Par exemple, un système sensible au contexte de type guide touristique pour les utilisateurs nomades doit garder continuellement à jour l'information relative à la localisation d'un utilisateur afin de pouvoir lui fournir un contenu informationnel qui correspond à son contexte courant. Ce type de système doit s'assurer que l'information délivrée à l'utilisateur est appropriée à la configuration spatio-temporelle dans laquelle il se trouve [O'Hare 2002]. Ainsi, un tel système doit suivre chaque déplacement de l'utilisateur, afin de garder à jour l'information relative à son contexte courant et donc être capable de s'adapter correctement à ce contexte.

Pour conclure, une fois qu'on a compris la signification de la notion de contexte (tout ce qui entoure l'utilisateur et qui contribue à l'interaction de cet utilisateur avec le système) et ses caractéristiques (notamment le fait d'être un espace infini et évolutif, dirigé vers un but précis), on peut s'intéresser à sa représentation en machine.

3.3.2 Représentations existantes

Selon Dey [Dey 2001], afin d'utiliser efficacement la notion de contexte, il faut non seulement la comprendre, mais également savoir comment l'utiliser. Pour ceci, il faut un certain support « architectural » (des services et des abstractions) pour mieux concevoir les systèmes sensibles au contexte. Ce besoin d'un support « architectural » voulu par Dey accompagne la recherche d'une représentation du contexte en machine qui soit exploitable pour la conception des systèmes sensibles au contexte. Dans cette section, nous abordons cette question à travers quelques représentations exposées par la littérature.

Pour Brézillon [Brézillon 2002b], une représentation efficace du contexte en machine, tant en termes de modélisation des connaissances que de raisonnement à partir de celles-ci, est un problème à résoudre, et ce, aussi bien du point de vue de la programmation que de son utilisation. Bien que, depuis quelques années, certaines propositions visant la résolution de ce problème soient apparues dans la littérature, cette question reste toujours ouverte. On sait, à travers la compréhension de la notion de contexte (voir section précédente), que toute modélisation du contexte doit être centrée sur un domaine ou une thématique précise, puisqu'une modélisation de tous les éléments possibles est irréalisable. Par ailleurs, l'information contextuelle, qui est numériquement représentée, n'est jamais totalement exacte, car il y a toujours un contexte lié à cette information qui n'est pas inclus dans cette représentation [Grudin 2001]. Ce processus de sélection des éléments qui doivent ou pas être pris en compte par une représentation de la notion de contexte est réalisé, selon Mostéfaoui *et al.* [Mostéfaoui 2004], tout au long de l'évolution du système qui l'utilise et doit également tenir compte des réactions des utilisateurs et des observations de ceux qui en sont responsables. Ainsi, on ne devrait donc pas apercevoir les représentations du contexte comme des modèles figés, puisque celles-ci devront évoluer dans le temps.

Cependant, la majorité des infrastructures pour les systèmes sensibles au contexte sont construites de manière *ad-hoc*, avec le seul but de construire un système précis. Les systèmes qui en résultent sont alors difficiles à modifier et à réutiliser [Mostéfaoui 2004]. Par ailleurs, ces systèmes mélangent souvent les codes pour le traitement du contexte avec le code propre à l'application, ce qui augmente considérablement leur complexité [Chaari 2004]. Or, la séparation entre la logique de l'application et celle propre à la gestion du contexte est une clé de voûte pour la conception des systèmes et des infrastructures qui soient réutilisables ou encore qui puissent évoluer dans le temps. Cette séparation dépend fortement de la proposition de modèles pour représenter le contexte qui soient également réutilisables et évolutifs.

Parmi les propositions pour la représentation du contexte, Chaari *et al.* [Chaari 2004] ont identifié trois approches : (*i*) l'utilisation de paires « attribut/valeur » (par exemple, « {Name="context1", User="x", Location="y", Time="t"} ») ; (*ii*) l'utilisation de RDF[34], notamment à travers l'extension du standard CC/PP[35] ; (*iii*) l'utilisation des ontologies. Mostéfaoui *et al.* [Mostéfaoui 2004] rapportent également l'utilisation de XML (non seulement RDF ou CC/PP), ainsi que l'utilisation des modèles à objets ou à graphes pour la représentation du contexte comme un ensemble d'états.

L'utilisation de paires attribut/valeur correspond au modèle le plus simple. Il s'agit de représenter le contexte d'utilisation comme un ensemble de paires contenant chacune un attribut et la valeur lui correspondant. Ce type de représentation est facile à utiliser, mais il est difficile à gérer puisque, avec un nombre de paires trop important, la gestion de ces paires, de leur signification et de leur contenu, devient complexe. Chaque paire détient une forte

[34] « *Resource Description Framework* » - http://www.w3.org/RDF/.
[35] « *Composite Capability/Preference Profiles* » - http://www.w3.org/Mobile/CCPP/.

signification sémantique : elle représente un élément du contexte d'utilisation. Plus le nombre de paires possibles augmente, plus difficile sera la gestion des aspects sémantiques liés à ces paires, et plus important sera le risque d'avoir des informations équivalentes en double. Cette duplicité d'informations comporte des risques bien connus (naturels à toute duplicité d'information), notamment l'incohérence entre les paires contenant des informations sur un même élément du contexte.

Outre la gestion qui devient critique avec l'augmentation du nombre des paires, il est également important de souligner que la réutilisation d'un tel modèle n'est pas assurée. Ce type de représentation est souvent couplé aux besoins du système dans lequel elle a été proposée. Par ailleurs, les aspects sémantiques liés à une paire attribut/valeur ne sont pas forcément explicites au sein même de la représentation. Tout ceci fait que les représentations de type paire attribut/valeur sont difficilement réutilisables dans d'autres systèmes que celui d'origine. Enfin, ce type de représentation est utilisé, par exemple, dans Schilit et Theimer [Schilit 1994a] et internement par le *Context Toolkit* [Dey 2000].

En ce qui concerne l'utilisation de RDF et de CC/PP, ce dernier est devenu une référence dans les travaux cherchant à adapter la présentation des informations aux capacités des dispositifs. Ce standard (voir [Klyne 2004]) permet la description des capacités d'un dispositif et de certaines préférences de l'utilisateur à travers la définition des « profils ». Un profil CC/PP sert à guider l'adaptation de l'information présentée sur un dispositif précis. Chaque profil contient un certain nombre d'attributs et de valeurs associées qui sont utilisées par le serveur pour déterminer quelle version de la ressource doit être délivrée au client. Ces profils sont structurés de manière à permettre à un client de décrire ses capacités par rapport à un vocabulaire prédéterminé (qui définit les noms d'attributs possibles). Ce vocabulaire, ainsi que la structure du profil, sont définis en respectant la recommandation donnée par le W3C [Klyne 2004]. Un exemple de profil CC/PP est présenté dans la Figure 9. Dans cet exemple, on voit la description des capacités d'affichage d'un dispositif en particulier, à travers les valeurs de l'élément *HardwarePlatform* (largeur 320 et hauteur 200), ainsi que la description du système d'exploitation et du navigateur Web installé dans le dispositif, à travers les valeurs des éléments *SoftwarePlatform* (EPOC Symbian 2.0) et *BrowserUA* (Mozilla 5.0).

Un exemple de travail utilisant CC/PP pour représenter le contexte d'utilisation est celui de Lemlouma [Lemlouma 2004a], qui propose un protocole de négociation pour l'adaptation de la présentation aux contraintes des dispositifs mobiles. Cet auteur utilise le CC/PP pour décrire les capacités des dispositifs, aussi bien en termes strictement physiques (taille d'écran, affichage en couleur, capacité de mémoire, etc.), que en termes de logiciels disponibles (le système d'exploitation, le navigateur...).

Le standard CC/PP est lui-même basé sur un autre standard, RDF, lequel a été conçu comme un langage pour la description de métadonnées [Klyne 2004]. L'objectif du RDF est de permettre la description des ressources Web en termes de leurs propriétés et de leurs valeurs, en faisant des affirmations (« *statements* ») sur ces ressources [Manola 2004]. Chaque affirmation correspond à un triplet « `ressource, propriété, valeur` ». Ces triplets forment un graphe dirigé qui peut être représenté également sous un dialecte XML, nommé XML/RDF, ce qui facilite le traitement automatique et la transmission de ces informations entre un client et un serveur (voir [Manola 2004]).

Le standard RDF peut être utilisé pour décrire des éléments du contexte autres que ceux liés à un dispositif et à l'environnement d'exécution d'une application (comme CC/PP le fait). L'avantage de l'utilisation de RDF par rapport à l'utilisation d'un ensemble de paires attribut/valeur réside sur la possibilité de décrire des métadonnées sur ces paires et de définir un vocabulaire commun englobant les propriétés susceptibles d'être décrites. Le standard RDF permet, quoique d'une manière simple, la description des concepts et des relations binaires

entre les concepts. Par contre, il ne permet pas la description des éventuelles contraintes sur ces concepts et les relations, telle que la multiplicité d'une relation [Bruijn 2003].

```
<?xml version="1.0"?>
<rdf:RDF
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"
         xmlns:ex="http://www.example.com/schema#">

  <rdf:Description rdf:about="http://www.example.com/profile#MyProfile">

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.example.com/profile#TerminalHardware">
        <rdf:type
            rdf:resource="http://www.example.com/schema#HardwarePlatform" />
        <ex:displayWidth>320</ex:displayWidth>
        <ex:displayHeight>200</ex:displayHeight>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.example.com/profile#TerminalSoftware">
        <rdf:type
            rdf:resource="http://www.example.com/schema#SoftwarePlatform" />
        <ex:name>EPOC</ex:name>
        <ex:version>2.0</ex:version>
        <ex:vendor>Symbian</ex:vendor>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.example.com/profile#TerminalBrowser">
        <rdf:type
            rdf:resource="http://www.example.com/schema#BrowserUA" />
        <ex:name>Mozilla</ex:name>
        <ex:version>5.0</ex:version>
        <ex:vendor>Symbian</ex:vendor>
        <ex:htmlVersionsSupported>
          <rdf:Bag>
            <rdf:li>3.2</rdf:li>
            <rdf:li>4.0</rdf:li>
          </rdf:Bag>
        </ex:htmlVersionsSupported>
      </rdf:Description>
    </ccpp:component>
  </rdf:Description>
</rdf:RDF>
```

Figure 9. Exemple de profil CC/PP (à partir de [Lemlouma 2004a]).

Afin de trouver une représentation du contexte avec un fort contenu sémantique, certains travaux ont choisi l'utilisation d'*ontologies*. Les ontologies sont utilisées notamment pour la définition d'un vocabulaire autour d'un ensemble de concepts et des relations unissant ces concepts. De manière générale, en informatique, une ontologie est un ensemble structuré de concepts qui permet de définir des termes les uns par rapport aux autres, chaque terme étant la représentation textuelle d'un concept[36]. Le W3C a défini un standard pour la description des ontologies, le OWL[37]. Celui-ci définit une ontologie comme *une collection d'informations, notamment des informations sur des classes et des propriétés* [Smith 2004]. En d'autres termes, une ontologie définit les termes utilisés pour décrire et pour représenter un domaine de connaissances. Les ontologies sont utilisées pour les informations appartenant à

[36]« *Wikipedia: l'encyclopédie libre* » - http://fr.wikipedia.org/wiki/Ontologie_%28informatique%29.
[37]« *Web Ontology Language* » - http://www.w3.org/2004/OWL/.

ce domaine. Elles incluent les définitions des concepts de base du domaine et les relations entre les concepts. Les ontologies encodent les connaissances du domaine, et ainsi, celles-ci deviennent réutilisables [Heflin 2004].

Parmi les travaux qui utilisent les ontologies afin de représenter le contexte d'utilisation, Bucur *et al.* [Bucur 2005] se sont servis du standard OWL pour définir une ontologie à travers laquelle ils recherchent une représentation homogène des éléments du contexte (appelés, par ces auteurs, « attributs contextuels »), tout en permettant l'association de plusieurs propriétés à un même élément. Ces éléments peuvent être soit d'ordre statique (dont la valeur ne change pas), telles que les informations relatives à l'utilisateur (nom, date de naissance...), soit d'ordre dynamique (la localisation de l'utilisateur, par exemple). En revanche, Bucur *et al.* [Bucur 2005] n'abordent pas la question des éléments qui doivent ou non faire partie de cette ontologie, et ne donnent que peu d'exemples, comme les informations concernant l'utilisateur (intérêts, rôles, etc.) à travers l'élément « *Person-related* », ses activités, à travers l'élément « *Activity-related* », ou encore sa localisation, avec l'élément « *Location-related* ».

Cette question des éléments qui composent la notion de contexte pour un domaine précis est abordée indirectement par Alarcón et Fuller [Alarcón 2002] et par Alarcón *et al.* [Alarcón 2004]. Leurs travaux utilisent également les ontologies pour représenter le contexte, mais dans le cadre particulier du travail coopératif et du support à la conscience de groupe. L'objectif de ces travaux est de fournir une information de conscience de groupe adaptée au contexte de l'utilisateur. Ces auteurs suggèrent la décomposition de la notion de contexte en "sous-contextes" à travers la définition d'ontologies spécifiques à chaque sous-contexte, en énumérant alors, pour chacune de ces ontologies, les éléments qui la composent. Ainsi, un premier sous-contexte est associé au travail mené de façon coopérative, un deuxième est relatif exclusivement à l'utilisateur, et enfin un troisième est lié aux documents partagés par le groupe (voir [Alarcón 2002]). Dans le premier sous-contexte (appelé *contexte de travail*), les auteurs exploitent un contenu constitué par les applications et par le processus coopératif, avec les activités qui le composent et le calendrier qu'il doit respecter. Dans le deuxième sous-contexte (*contexte de l'utilisateur*), ils considèrent les utilisateurs eux-mêmes, en portant une attention particulière à leur localisation "électronique" (c'est-à-dire les moyens grâce auxquels on peut les contacter : SMS, messagerie électronique, etc.). Dans le dernier sous-contexte, les objets partagés, avec leurs structures, sont pris en compte. À travers ces ontologies et des règles prédéfinies, un ensemble d'agents intelligents choisissent par quel moyen une information de conscience de groupe sera délivrée à l'utilisateur.

Dans une approche semblable à celle d'Alarcón *et al.*, les recherches de Leiva-Lobos et Covarrubias [Leiva-Lobos 2002] sont également centrées sur le support à la conscience de groupe. Pour ces auteurs, la conscience de groupe est directement liée au contexte. Ces auteurs proposent donc trois ontologies pour modéliser le contexte d'un utilisateur dans le cadre d'une coopération. Ces ontologies décrivent trois axes : *spatial*, *temporel* et *culturel*. A travers ces axes, les auteurs mettent en avant les artefacts qui peuplent l'espace physique et numérique (à travers l'axe spatial), les événements qui composent l'historique de la coopération ainsi que les attentes de l'utilisateur pour le futur (axe temporel), et les acteurs qui composent la communauté et leurs pratiques communes (axe culturel).

Les ontologies constituent une forme de représentation de connaissances très puissante, qui permet la représentation de concepts et de relations. Cependant, cette puissance dépend du langage utilisé pour les exprimer (voir [Corcho 2000]). Les ontologies ne permettent pas toujours la définition de contraintes fortes dans les relations (*i.e.* les relations transitives, les quantificateurs, etc.), ce qui peut être préjudiciable lors de la construction dynamique du contexte [Alarcón 2004]. De plus, les ontologies sont, en général, difficiles à

comprendre et à manipuler, surtout pour les professionnels non habitués à ce mode de représentation de connaissances. Alarcón et Fuller [Alarcón 2004] le concèdent, en soulignant que cette méthode de représentation exige un large effort de la part des concepteurs (et des utilisateurs qui sont invités à personnaliser certaines ontologies dans [Alarcón 2002]), lesquels ne sont pas forcément familiers de ce type de représentation.

Les *graphes contextuels*[38] constituent un autre mode de représentation du contexte présenté par Brézillon [Brézillon 2002a] et par Mostéfaoui *et al.* [Mostéfaoui 2004]. Un graphe contextuel est un graphe dirigé acyclique qui représente l'ensemble d'actions à réaliser pour résoudre un problème selon un contexte donné. Ces graphes décrivent l'enchaînement des actions pour la résolution d'un problème donné, avec les éléments de contexte explicitement représentés [Brézillon 2002a] [Mostéfaoui 2004]. L'objectif des graphes contextuels est donc de représenter une fraction du contexte nécessaire à une prise de décision. Chaque graphe est composé par des nœuds dits d'action (« *action nodes* »), qui décrivent les actions à effectuer pour atteindre un but précis, et par des nœuds contextuels (« *contextual nodes* »), qui décrivent les éléments de contexte nécessaires pour la poursuite d'un chemin d'actions. Il s'agit de nœuds de décision qui guident le processus de prise de décision à travers un chemin précis parmi les possibles. Chaque contexte est ainsi représenté par une séquence de nœuds contextuels dans une branche du graphe, laquelle déclenche les actions. Ces actions sont dues à un raisonnement non représenté directement dans le graphe, mais reconnu et traité en amont par une personne [Brézillon 2002a] [Mostéfaoui 2004]. La Figure 10 illustre un graphe contextuel à travers la description d'une prise de décision relative à une politique de contrôle d'accès. Dans cette figure, les grands cercles correspondent aux nœuds contextuels, tandis que les rectangles correspondent aux nœuds d'action. Les petits cercles noirs représentent des nœuds de "recombinaison", à travers desquels la convergence entre plusieurs chemins d'actions est représentée.

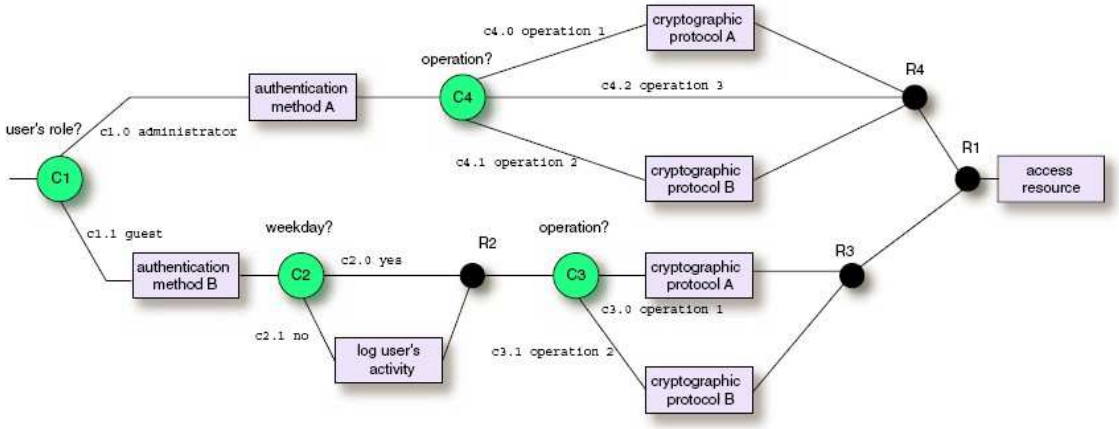


Figure 10. Exemple de graphe contextuel pour un problème de politique de contrôle d'accès à une ressource (d'après [Mostéfaoui 2004]).

Les graphes contextuels concentrent donc la représentation du contexte nécessaire à la résolution d'un problème précis. Ils décrivent comment le contexte peut guider le cheminement des actions pour atteindre le but déterminé par le processus de résolution du problème. Néanmoins, comme on peut l'observer à travers la Figure 10, les graphes contextuels n'explicitent pas une représentation interne précise pour les nœuds contextuels. Ceci ouvre la possibilité à l'utilisation, en parallèle, d'autres méthodes de représentation, telles que les ontologies ou encore XML et RDF.

[38]En anglais, « *contextual graphs* ».

Par ailleurs, on retrouve également dans la littérature des représentations de contexte qui reposent sur un modèle à objets. Dans cette approche, le contexte est représenté à travers un ensemble de classes et d'objets, et des associations mettant en relation ces éléments. Un exemple de travail adoptant cette approche est celui de Henricksen *et al.* [Henricksen 2002]. Ces auteurs se basent sur cette approche sans pour autant utiliser les formalismes qui lui sont familiers, tels que le langage UML[39]. Au contraire, ces auteurs proposent un formalisme qui leur est propre (voir un exemple dans la Figure 11), dans lequel ils représentent le contexte à travers un ensemble d'entités, d'attributs et d'associations. Chaque entité décrit un objet physique ou conceptuel, comme une personne ou un moyen de communication. Les attributs représentent les propriétés des entités auxquelles ils sont attachés par le biais des associations. Les associations connectent également les entités entre elles. Les associations ont toujours, selon les auteurs, une entité origine et une ou plusieurs entités destinations. Le modèle proposé par Henricksen *et al.* [Henricksen 2002] classifie les associations selon plusieurs critères distincts : le dynamisme (statiques ou dynamiques), l'origine des informations (acquises par des capteurs, obtenues à travers l'information d'autres associations ou fournies par l'utilisateur), ou encore la structure des associations (simple ou complexe).

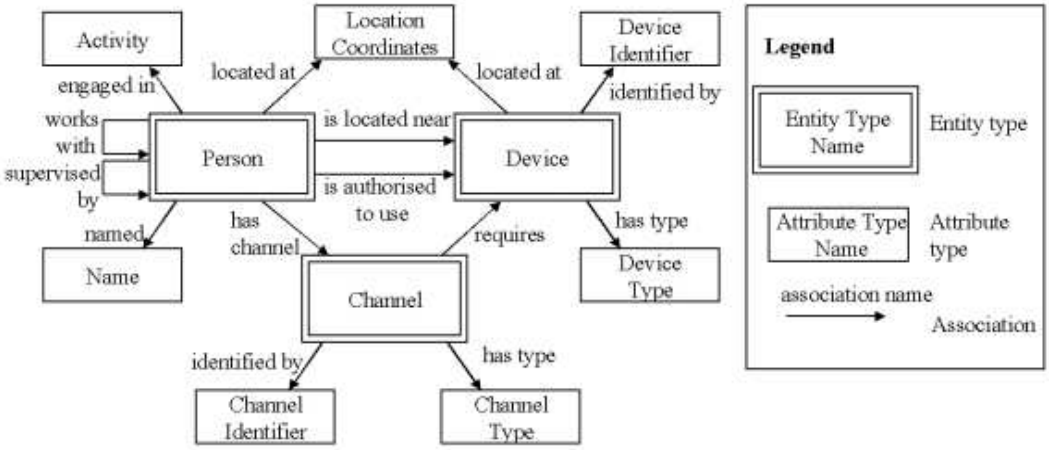


Figure 11. Exemple de représentation de contexte de Henricksen *et al.* [Henricksen 2002].

De plus, les auteurs proposent de rendre explicite une contrainte de dépendance entre deux associations par laquelle un changement sur l'une peut entraîner des modifications sur le contenu de l'autre. Néanmoins, ce mécanisme de contraintes se limite à ce type de dépendance. Il n'a donc pas le même pouvoir d'expression qu'offre la spécification UML et le langage OCL[40]. Par exemple, il paraît difficile de représenter dans le modèle de Henricksen *et al.* [Henricksen 2002] des contraintes telles que la multiplicité d'une association, ou encore une contrainte sur la valeur d'un attribut. On déplore également l'apparente impossibilité de représentation d'associations *n-aires* autres que les compositions. Enfin, à l'instar de Bucur *et al.* [Bucur 2005], Henricksen *et al.* [Henricksen 2002] cherchent une représentation générique qui soit applicable à n'importe quelle application. Par conséquent, étant donnée la nature même de la notion de contexte, qui se définit en fonction d'une finalité précise (cf. section 3.3.1), ces auteurs sont dans l'incapacité d'aborder la question de quels éléments composent cette représentation, laissant cette difficile tâche aux concepteurs des systèmes désirant utiliser ces représentations.

[39]« *Unified Modeling Language* » - http://www.uml.org/.
[40]« *Object Constraint Language* » - http://www.omg.org/docs/ptc/05-06-06.pdf.

Toujours dans l'approche d'une modélisation par objets, on retrouve la proposition de Bardram [Bardram 2005]. Cet auteur propose *JCAF*[41], une infrastructure et une API[42] Java pour le développement des systèmes sensibles au contexte, qui couvre de l'acquisition du contexte à son exploitation par le système. L'infrastructure *JCAF* utilise un modèle de communication pair-à-pair dans lequel un ensemble de services (« *context services* ») collaborent entre eux. Chaque service est conçu pour manipuler et fournir des informations contextuelles bien spécifiques (par exemple, un service pour observer le contexte dans une salle de chirurgie d'un hôpital : qui est là, qui est le patient, quel est l'état actuel de l'opération...). L'infrastructure prévoit qu'un client désirant une information fournie par un service particulier puisse l'obtenir, soit directement à travers une requête au service, soit en s'inscrivant auprès du service, afin d'être automatiquement averti en cas de changement dans le contexte observé par le service.

Bardram [Bardram 2005] propose, de manière implicite à travers l'API *JCAF*, un modèle orienté à objet de la notion de contexte. Celui-ci se base sur la définition d'une série d'interfaces Java : entité (*Entity*), contexte (*Context*), relation (*Relation*) et item de contexte (*ContextItem*). Un contexte décrit le contexte autour d'une entité qui peut être une personne, une salle, etc. Ce contexte est composé par un ensemble d'items de contexte qui sont liés aux entités par les relations. Par exemple, *JCAF* permet de définir un objet contexte « contexte de l'hôpital » sur une entité « hôpital », et une relation « se localise » (*located*) liant une entité « personneX » à un item de contexte « salle333 ». Ces interfaces sont les éléments de base de la représentation de contexte utilisé par *JCAF*. Toutefois, puisque ces éléments sont des interfaces Java, ils doivent être implémenté par les développeurs en tenant compte de l'application en construction, si bien que *JCAF* propose des implémentations qui peuvent être étendues. Par conséquent, même si on imagine que, dans la majorité des cas, les services utilisés par un système seront conçus par un seul groupe de développeurs, rien n'empêche deux services d'avoir des représentations distinctes pour un même concept. Par exemple, une information relative à une localisation peut être fournie par un service sous la forme de coordonnées (latitude et longitude) et par un autre sous la forme d'une information de haut niveau (« salle333 »). Ceci peut entraîner des problèmes d'intégration entre les services coopérants, ou même entre un client et les services qu'il utilise. Dans ce dernier cas, le client devrait savoir manipuler les différentes représentations d'un même concept utilisées par différents services.

La proposition de Bardram [Bardram 2005] se caractérise donc par sa spécificité Java : *JCAF* a été entièrement conçu avec le but de faciliter le développement rapide des prototypes de systèmes sensibles au contexte en Java. Il ne peut être employé que dans le cadre de ce langage. On regrette aussi l'absence d'une discussion plus approfondie sur la notion de contexte et sur les composants de la représentation utilisée au sein de l'API *JCAF*.

D'autres travaux proposent également des infrastructures pour le développement de systèmes sensibles au contexte. L'exemple le plus connu parmi ces travaux est le *Context Toolkit* proposé par Dey [Dey 2000]. Cependant, même si celui-ci s'intéresse également à la construction de ce type de système, le *Context Toolkit* est moins dépendant d'un langage de programmation que *JCAF*. De plus, il est centré principalement sur le processus d'acquisition du contexte, que nous abordons dans la section suivante.

[41]En anglais, « *Java Context Awareness Framework* » [Bardram 2005].
[42]En anglais, « *Application Programming Interface* » - interface de programmation.

3.3.3 Acquisition de contexte

L'acquisition de contexte correspond au processus à travers lequel un système sensible au contexte capture les informations contextuelles nécessaires à son fonctionnement. Ces informations varient d'un système à un autre. Selon Grudin [Grudin 2001], différents éléments du contexte d'utilisation supposent différents niveaux d'importance, qui varient dans le temps. Les concepteurs doivent déterminer quels éléments seront capturés et où placer le seuil qui détermine leur utilisation ou non. Toutes ces décisions se basent, selon cet auteur, sur le type et l'importance d'information, ainsi que sur le fait que cette information puisse être capturée ou inférée facilement, ou d'une manière économiquement viable. Le coût du processus d'acquisition de certains éléments est un facteur clé lors de la conception d'un système sensible au contexte. Certains aspects du contexte d'utilisation, comme le moment de l'utilisation (temps) et la localisation de l'utilisateur, sont plus faciles à détecter pour un système que d'autres comme l'activité de l'utilisateur [Burrell 2002]. Ceci explique, en partie, le grand nombre de systèmes qui adoptent une vision limitée du contexte d'utilisation (cf. section 3.1).

Le contexte peut être obtenu par différentes méthodes qui varient selon l'élément détecté. Selon la méthode d'acquisition utilisée, on distingue dans la littérature trois types d'information contextuelle [Mostéfaoui 2004] : (*i*) Contexte détecté : ce type d'information est acquis à travers des capteurs physiques ou logiciels tels que des capteurs de température, de pression atmosphérique, de lumière ou de niveau de bruit ; (*ii*) Contexte dérivé : ce type d'information contextuelle est calculé lors de l'exécution, tels que l'heure et la date ; (*iii*) Contexte explicitement fourni : c'est le cas lorsque l'utilisateur communique explicitement au système les informations nécessaires.

Les exemples d'acquisition de contexte par le biais de capteurs sont multiples : la détection de la température ou du niveau de bruit dans une pièce à travers l'utilisation de capteurs physiques postés dans cette pièce ; l'utilisation de GPS (géopositionnement par satellite)[43] pour la découverte de la localisation d'un utilisateur, ou encore l'utilisation des capteurs de mouvement pour détecter la présence de quelqu'un dans un espace (une salle, devant un objet, etc.). Anne *et al.* [Anne 2005] proposent, par exemple, de combiner l'utilisation des réseaux *WiFi* (à travers la puissance des signaux) et des étiquettes électroniques (identification par radiofréquence – RFID[44]) avec l'analyse des images vidéo afin d'identifier, à la fois, l'utilisateur et sa localisation. D'autre part, il existe certaines informations contextuelles, plus difficiles à capturer de manière automatique, qui peuvent être fournies directement au système par l'utilisateur lui-même, à travers une interface appropriée. Ceci est typiquement le cas des buts et de l'activité courante de l'utilisateur, qu'il n'est pas toujours possible d'inférer automatiquement.

Par ailleurs, on observe que la majorité des informations contextuelles utilisées à l'intérieur des systèmes sensibles au contexte proviennent de capteurs. Néanmoins, il ne faut pas oublier qu'il est souvent nécessaire d'interpréter les données fournies par ces capteurs [Henricksen 2002]. Par exemple, les données obtenues à travers un GPS (la latitude, la longitude, l'altitude) sont souvent interprétées par l'application afin d'être réellement exploitables. Ainsi, un ensemble de coordonnées devient une adresse dans une ville pour un système de navigation. De plus, on trouve souvent, pour un élément de contexte donné, plusieurs méthodes de détection possibles. Ces méthodes ne sont pas toujours équivalentes. Prenons l'exemple de la localisation (il s'agit de l'élément le plus utilisé par les systèmes sensibles au contexte). Lorsqu'on se trouve à l'intérieur d'un bâtiment, dans un environnement confiné, la détection de la localisation par GPS devient inefficace, la précision des données

[43]En anglais « *Global Positioning System* ».
[44]En anglais « *Radio Frequency IDentification* ».

ainsi obtenues est fortement dégradée [Samama 2005]. Dans ce cas, d'autres méthodes deviennent plus intéressantes (voir, par exemple, [Anne 2005] et [Rubinsztejn 2004]). Selon Banerjee *et al.* [Banerjee 2002], le choix de la méthode de détection de la localisation affecte la granularité et la précision de l'information. On peut généraliser cette affirmation, puisque ceci est également vrai pour des éléments de contexte autres que la localisation.

Un autre problème lors de l'acquisition du contexte est lié aux erreurs qui peuvent survenir. Ces erreurs sont dues, par exemple, aux résultats des méthodes de détection, ou encore à des changements trop rapides dans l'environnement. Or, comme l'information contextuelle est utilisée par les systèmes notamment pour la prise de décision à l'insu de l'utilisateur, il est important que ces systèmes puissent juger de la fiabilité de cette information [Henricksen 2002]. Selon Greenberg [Greenberg 2001], on peut espérer que les systèmes détectent correctement le contexte dans la majorité de cas, mais l'erreur est inévitable. Ainsi, pour cet auteur, les utilisateurs doivent être capables d'ajuster les informations collectées et le processus de collecte afin que l'acquisition du contexte soit réellement appropriée. Cet auteur souligne également que les utilisateurs devraient pouvoir supplanter les informations du système, puisque certains éléments du contexte ne sont pas toujours inférés correctement (et l'utilisateur est toujours le mieux placé pour corriger ces informations).

Ainsi, même l'acquisition du contexte à travers les capteurs ne constitue pas toujours un processus facile, car l'information peut être acquise à partir de différents capteurs, comporter des erreurs, ou encore exiger une interprétation supplémentaire pour être exploitée par le système. De plus, l'information contextuelle est également de nature dynamique, et les changements qui en découlent doivent être rapidement saisis par l'infrastructure d'acquisition [Mostéfaoui 2004].

On observe donc trois difficultés majeures qui concernent l'acquisition de contexte [Dey 2000] : (*i*) le contexte est dynamique : les changements dans l'environnement doivent être détectés en temps réel et les applications doivent s'adapter à ces changements continus ; (*ii*) le contexte est capturé à partir de multiples sources, souvent hétérogènes et reparties ; (*iii*) le contexte est obtenu à travers la manipulation des périphériques non conventionnels (autres que la souris ou le clavier).

La manipulation des capteurs ne comporte pas, à ce jour, de techniques aussi bien développées et maîtrisées que la manipulation des périphériques traditionnels tels qu'un clavier ou une souris, pour lesquels il existe un large support au niveau des langages de programmation. Cependant, il faut remarquer que ce support ne tardera pas à apparaître, surtout avec l'évolution des standards et techniques comme *Jini*[45], *UPnP*[46] et *OSGi*[47]. Ceux-ci s'intéressent à la communication et l'interconnexion des dispositifs (pas forcément des ordinateurs) dans un réseau. Dans un futur proche, avec l'évolution de ce type de technique, nous pouvons imaginer que la programmation pour l'acquisition de données à partir des capteurs ou d'autres dispositifs sera aussi facile que la manipulation des données d'une souris. Néanmoins, pour l'instant, comme l'a observé Dey [Dey 2000], l'information contextuelle est manipulée d'une façon quasiment improvisée. Les concepteurs des systèmes sensibles au contexte choisissent les méthodes d'acquisition selon leur facilité et les technologies disponibles, en dépit de quelconque considération sur la réutilisation ou l'évolution des systèmes (par exemple, l'utilisation des nouveaux capteurs non prévus auparavant). Par conséquent, ces systèmes deviennent trop dépendants des technologies de détection. Ces systèmes, qui sont conçus pour adapter leur comportement au contexte, sont, paradoxalement, incapables de s'adapter aux changements dans le processus d'acquisition du contexte.

[45] http://www.jini.org/.
[46] « *Universal Plug and Play* » - http://www.upnp.org/.
[47] « *Open Services Gateway Initiative* » - http://www.osgi.org/.

Afin d'aider la conception des systèmes sensibles au contexte dans le processus d'acquisition, certaines infrastructures ont été proposées. Pour la plupart, ces infrastructures cherchent à organiser ce processus d'acquisition du contexte. Ceci est notamment le cas du *Context Toolkit* [Dey 2000] [Dey 2001]. Celui-ci propose un ensemble d'éléments de base pour la capture et l'interprétation des informations contextuelles sans pour autant imposer un modèle de représentation pour ces informations, une fois capturées. L'objectif du *Context Toolkit* est de faciliter l'évolution des systèmes sensibles au contexte en les rendant plus résistants aux changements dans la couche d'acquisition de contexte [Dey 2001]. Le *Context Toolkit* isole les applications des méthodes d'acquisition réellement utilisées à travers la définition de cette couche d'acquisition, qui manipule directement les technologies.

Le *Context Toolkit* organise le processus d'acquisition à travers un nombre limité de composants de base [Dey 2000] [Dey 2001] : les « *context widgets* », les « *context interpreter* », les « *context aggregators* » et les « *context services* ». Les *widgets* interagissent directement avec les capteurs (qu'il s'agisse de capteurs physiques ou de logiciels) qui détectent un certain élément du contexte. Ce sont donc les *widgets* qui collectent les informations à partir de l'environnement. À travers ces composants, les applications peuvent accéder aux données sur un certain élément de contexte sans, pour autant, manipuler directement les technologies utilisées pour sa détection. Les *interpreters* sont chargés de l'interprétation des données capturées par un ou plusieurs *widgets*. Les *interpreters* réalisent une abstraction des données brutes collectées par les *widgets* en une information de plus haut niveau. Les *aggregators* rassemblent les informations de contexte sur une entité (par exemple, une personne, un objet, un endroit...), agissant ainsi comme une passerelle entre le contexte et le système. Ils agrègent toutes les informations collectées et interprétées concernant une entité donnée. Enfin, les services sont un type particulier de *widget* qui, au lieu de collecter les informations, contrôlent ou modifient ces informations dans l'environnement. Ces composants peuvent être découverts dynamiquement par les applications à travers un autre composant, le « *discoverer* », auprès duquel les composants s'enregistrent à leur démarrage. La Figure 12 présente un exemple typique d'interaction entre ces composants et les applications.

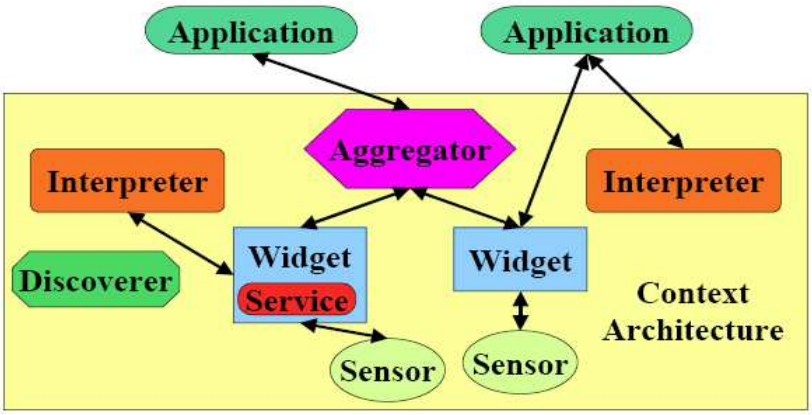


Figure 12. L'architecture du *Context Toolkit* d'après [Dey 2000].

Le *Context Toolkit* représente donc une des premières infrastructures pour l'acquisition des informations contextuelles. Cependant, selon Rey et Coutaz [Rey 2004], cette infrastructure présente quelques limitations importantes : (i) Elle ne fournit pas de métadonnées pour renseigner l'application sur la qualité et/ou la précision des informations fournies ; (ii) Bien que les composants du *Context Toolkit* puissent être répartis, le *Discoverer* doit être connu de tous. Ce point central constitue un point faible par rapport tant au passage à

l'échelle (lorsque le nombre des composants et d'applications les utilisant s'amplifie) qu'à la mobilité (le *Discoverer* ne sera pas toujours accessible pour tous les clients mobiles).

Pour Rey et Coutaz [Rey 2004], la mise en oeuvre des systèmes sensibles au contexte nécessite la mise à disposition des concepteurs, d'infrastructures capables de fournir les données contextuelles nécessaires à chaque système. Ces infrastructures doivent satisfaire, selon ces auteurs, certaines exigences : 1) chaque infrastructure doit permettre la mobilité de l'utilisateur ; 2) elle doit également fonctionner sur de petits dispositifs, aux capacités réduites ; et 3) elle doit autoriser les déconnexions et mesurer la qualité des données captées ou calculées.

Afin d'atteindre ces objectifs, Rey et Coutaz [Rey 2004] proposent une autre infrastructure basée sur la combinaison des composants nommées « *contexteurs* ». Pour ces auteurs, un *contexteur* est une abstraction logicielle qui fournit la valeur d'un élément de contexte. Un *contexteur* est souvent combiné à d'autres, et ceci de plusieurs manières distinctes (dans un réseau de *contexteurs*, de manière hiérarchique, par encapsulation, etc.), selon les besoins et les caractéristiques du système.

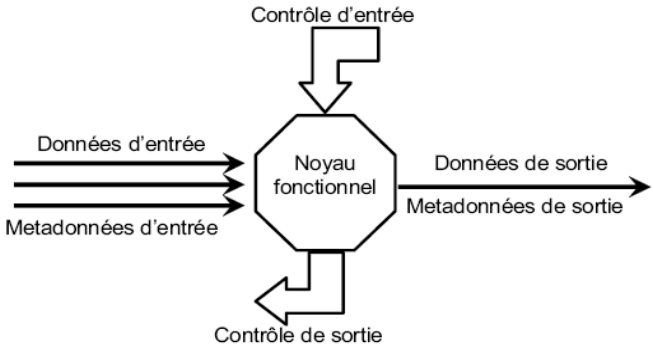


Figure 13. La structure d'un *contexteur* (à partir de [Rey 2004]).

Chaque *contexteur* comprend trois classes d'éléments [Rey 2004] : les *entrées*, les *sorties* et un *corps fonctionnel* (voir Figure 13). Les *entrées* sont de deux types : les données d'entrée et le contrôle d'entrée. Les données d'entrée correspondent aux données que le *contexteur* a à sa charge. Le contrôle d'entrée permet à un autre *contexteur* ou à une application de modifier les paramètres internes du *contexteur*, modifiant le comportement de son corps fonctionnel. Par ailleurs, toutes les données d'entrée sont associées à des métadonnées qui expriment leur qualité à travers des propriétés comme, par exemple, la précision, la stabilité, la résolution, la latence, ou encore un facteur de confiance calculé à partir des caractéristiques des capteurs. Les *sorties* sont aussi de deux types : les données de sortie, qui représentent les informations transmises soit à d'autres *contexteurs*, soit à une application, et le contrôle de sortie, qui permet de modifier les paramètres internes d'un *contexteur* cible. De même que pour les entrées, chaque donnée de sortie est assortie de métadonnées qui en expriment la qualité. Enfin, le *corps fonctionnel* désigne la fonction que le *contexteur* remplit. Il peut s'agir, par exemple, d'un *contexteur* élémentaire, qui encapsule un capteur physique, d'un *contexteur* à mémoire, qui mémorise un historique des données et métadonnées d'entrée, d'un *contexteur* à seuil, qui teste le franchissement d'un seuil, d'un *contexteur* d'abstraction, qui produit des informations de plus haut niveau d'abstraction, etc.

À travers l'utilisation des *contexteurs*, il est possible d'organiser le processus d'acquisition du contexte de manière indépendante de l'application. Cette séparation entre le code propre au domaine d'application et celui propre à la manipulation de contexte est aussi le

but du *Context Toolkit*. À travers l'utilisation de ces infrastructures, il est possible de changer les technologies utilisées pour l'acquisition du contexte (à travers l'utilisation des nouveaux capteurs, par exemple), sans pour autant avoir d'importantes répercutions sur le reste du système, puisque le code propre au domaine d'application n'est pas concerné par les changements. Ce type d'infrastructure peut opérer en combinaison avec les représentations de contexte (voir section 3.3.2). L'utilisation d'une représentation propre aux informations contextuelles, ainsi que celle d'une infrastructure pour l'acquisition de ces informations représentent un premier pas vers la conception des systèmes sensibles au contexte réutilisables et évolutifs.

Pour conclure, il ne faut pas oublier que la numérisation des informations contextuelles change, d'une certaine façon, la nature même de ces informations. L'acquisition de contexte est un processus « décontextualisé » : lorsqu'on enregistre une information contextuelle dans un disque dur, cette information sera naturellement moins riche que la situation qu'elle est censée décrire, ce qui peut parfois conduire à des problèmes d'interprétation [Moran 2001]. De plus, cette numérisation des informations qui avant n'étaient disponibles que physiquement soulèvent également des problèmes liés au respect de la vie privée [Grudin 2001]. Ce problème du respect de la vie privée est souvent négligé par les recherches liées à l'acquisition du contexte. Il en va de même pour la précision des informations acquises. Les possibles ambiguïtés ou l'absence d'informations contextuelles (lorsque le processus d'acquisition est défaillant) ne sont que très rarement étudiées [Mostéfaoui 2004]. On voit donc que l'étude du contexte (sa signification, sa représentation et son acquisition) pose encore nombreux verrous scientifiques et technologiques qui doivent être traités par l'informatique sensible au contexte.

3.4 Les applications

Les applications sensibles au contexte se caractérisent par la prise en compte du contexte de l'utilisateur. Les premières applications de ce type sont apparues dès les premiers travaux dans cette thématique (voir, par exemple, [Schilit 1994a]). Certains de ces travaux ont également essayé d'organiser ces applications selon différents critères. Parmi les classifications proposées, on peut citer celles de Schilit *et al.* [Schilit 1994b] et celle de Brown *et al.* [Brown 1997]. Schilit *et al.* [Schilit 1994b] ont identifié quatre catégories d'applications sensibles au contexte qui varient selon deux axes orthogonaux : 1) si la tâche réalisée par l'application consiste à collecter des informations (*information*) ou à exécuter une action (*commande*) ; 2) si l'application réalise cette tâche sous la demande de l'utilisateur (*manuelle*) ou de manière automatique (*automatique*).

D'autre part, Brown *et al.* [Brown 1997] divisent ces applications en deux groupes, aux frontières floues : les *continues* et les *discrètes*. Dans une application continue, l'information délivrée à l'utilisateur change continuellement avec le contexte. Dans une application discrète, au contraire, l'information délivrée à l'utilisateur est organisée en plusieurs segments auxquels on associe un contexte. Lorsque l'utilisateur se trouve dans ce contexte, l'information est délivrée. On ne considère plus ici si l'information délivrée à l'utilisateur a été demandée de manière explicite ou non (comme dans les catégories présentées ci-dessus), mais seulement si cette information est continuellement disponible ou non.

La classification proposé par Brown *et al.* [Brown 1997] s'applique facilement aux applications de type « guide touristique ». Ces applications, dans leur grande majorité, cherchent à délivrer à l'utilisateur (typiquement, un touriste qui visite une ville, un musée, un campus...) un contenu adapté à sa localisation et à l'activité touristique (donner les

informations sur une œuvre dans un musée, sur les restaurants dans une ville, etc.). Ces applications de guide touristique sont bien développées dans la littérature et ont servi, en grande partie, à démontrer la validité des recherches sur l'informatique sensible au contexte.

Parmi les applications de type guide touristique, on peut retenir l'application *GUIDE* proposée par Cheverest *et al.* [Cheverest 2002]. *GUIDE* est un système conçu pour fournir aux visiteurs d'une ville, munis des dispositifs mobiles (les expériences ont été menées à Lancaster, au Royaume-Uni, avec des *Tablet PCs*), une information touristique sensible au contexte. Ce système utilise le réseau *WiFi* pour disséminer l'information de localisation et les unités de contenu. Ces informations sont rendues disponibles par la diffusion (*broadcast*) de pièces d'information envoyées par des serveurs spécifiques (chaque serveur est responsable d'une zone géographique donnée, comme un musée). Le contenu mis à disposition des utilisateurs est un hypertexte en langage HTML, lequel est donc adapté à la localisation de l'utilisateur (les points touristiques dont il est proche), aux attractions déjà visitées (historique des visites), et aux préférences de l'utilisateur, représentées par un profil contenant son âge, ses centres d'intérêts, ses préférences alimentaires, etc.

Les applications de type guide touristique ne sont pas toujours tournées vers la découverte d'une ville. Ce type d'application peut également se focaliser sur une région plus réduite qu'une ville, comme un musée ou un campus universitaire. Ce dernier est la cible de l'application *Campus Aware* [Burrell 2002]. Le *Campus Aware* fonctionne à la fois comme un guide touristique dans le but d'aider les jeunes étudiants à connaître le campus, et comme une application de type « *geonotes* ». Il permet aux membres d'une communauté, typiquement des étudiants, d'annoter l'espace physique (un restaurant ou un bâtiment, par exemple) avec des commentaires, des opinions, des questions, etc., ou encore de donner leur avis (par vote) sur une annotation préexistante. Les futurs utilisateurs, particulièrement les nouveaux étudiants, peuvent consulter ces informations déposées par les anciens étudiants et, ainsi, être guidés dans leurs déplacements sur le campus (par exemple, aller dans un restaurant plutôt qu'un autre, trouver le bâtiment où a lieu une certaine manifestation, etc.).

Burrell *et al.* [Burrell 2002] ont procédé à plusieurs tests. Dans ces tests, les auteurs ont utilisé plusieurs mécanismes visant à réduire l'attention que l'utilisateur doit prêter au système lors de son utilisation, afin que cet utilisateur puisse se déplacer sur le campus en toute tranquillité. Les mécanismes ainsi mis en place incluent l'utilisation d'alertes sonores (lorsqu'une nouvelle annotation est disponible), la sauvegarde en mémoire des notes déjà consultées ou leur ordonnancement selon le nombre des votes. Les résultats pratiques de ces tests ont démontré l'intérêt des utilisateurs nomades pour ce type de mesure contre la surcharge cognitive.

Les guides touristiques tels que *Campus Aware* [Burrell 2002] se classifient, selon Schilit *et al.* [Schilit 1994b], comme des applications d'*information* (leur but principal est de délivrer des informations à l'utilisateur) et *automatiques* (elles le font automatiquement). Les problèmes traités par ces applications sont notamment la découverte de la localisation et l'adaptation du contenu. Par exemple, la proposition de Burrell *et al.* [Burrell 2002] porte sur une adaptation du contenu par la sélection d'informations basée sur la localisation de l'utilisateur. D'autres travaux, comme celui de Schilit *et al.* [Schilit 2002b] et celui de Lemlouma [Lemlouma 2004a], portent plutôt sur l'adaptation de la présentation de ce contenu délivré à l'utilisateur selon les capacités physiques du dispositif client. Schilit *et al.* [Schilit 2002b] soulignent l'utilisation de différentes techniques pour l'adaptation de la présentation des pages Web (traduction, *scaling*, transformation, etc.), ainsi que la possibilité de présenter le contenu d'une page Web séparément : d'abord les liens qu'elle contient, ensuite son texte. Leur objectif est de faciliter la navigation lorsqu'on utilise un dispositif mobile tel qu'un téléphone cellulaire.

Pour sa part, Lemlouma [Lemlouma 2004a] propose un protocole de négociation basé sur un ensemble de profils décrits en RDF et en CC/PP. Le but de ce protocole est de permettre une négociation entre le client et le serveur afin que le contenu le mieux adapté aux contraintes du dispositif client, en termes de présentation, soit délivré. L'adaptation de la présentation se fait notamment à travers la transformation du contenu suivant des règles de transformation exprimées en XSLT[48]. Le contexte de l'utilisateur dans ce travail, comme dans celui de Schilit *et al.* [Schilit 2002b], se résume aux capacités du dispositif employé par l'utilisateur au moment de l'accès au système.

D'autres travaux proposent des architectures pour la conception de systèmes sensibles au contexte. L'objectif de l'architecture *Rover* [Banerjee 2002], par exemple, est de suivre à la trace la localisation des utilisateurs et de configurer dynamiquement les informations au niveau de l'application, et ceci pour différents types de réseaux et de dispositifs. Cette architecture considère des utilisateurs nomades munis de dispositifs mobiles aux capacités variées. Pour cela, *Rover* garde un profil pour chaque type de dispositif qui identifie ses capacités (comme dans les profils proposés par Lemlouma [Lemlouma 2004a]). À travers ces profils, *Rover* filtre les médias disponibles selon les capacités d'un dispositif donné. De plus, *Rover* filtre les informations disponibles selon la localisation courante de l'utilisateur, afin qu'il dispose d'un contenu adapté à sa situation courante. Pour cela, l'architecture *Rover* compte sur un *contrôleur*, qui fait la gestion des services demandés par les clients et filtre le contenu selon la localisation et les profils du dispositif client et de l'utilisateur, et sur un *serveur de localisation*, qui calcule la localisation de chaque utilisateur dans le système.

L'architecture *Rover* [Banerjee 2002] adapte donc le contenu disponible en utilisant une notion de contexte limitée à la localisation de l'utilisateur et aux capacités du dispositif client. L'information de localisation est également utilisée par l'architecture *MoCA* [Rubinsztejn 2004]. Cependant, *MoCA* vise particulièrement la construction d'applications sensibles au contexte qui favorisent la collaboration informelle entre les utilisateurs munis des dispositifs mobiles. Selon Rubinsztejn *et al.* [Rubinsztejn 2004], la collaboration entre les utilisateurs nomades se caractérise par un dynamisme plus important que la collaboration sur réseau fixe exclusivement, à cause notamment de la mobilité des utilisateurs et de l'intermittence de la connexion sur les réseaux sans fil. Grâce à ce dynamisme, la collaboration "mobile" dépend fortement du contexte de l'utilisateur, de son activité et de ses intérêts personnels. Les applications construites à l'aide de l'architecture *MoCA* visent donc à favoriser les collaborations spontanées, qui s'organisent de manière dynamique entre les individus, selon les occasions et les opportunités.

Ainsi, comme pour l'architecture *Rover* [Banerjee 2002], l'architecture *MoCA* [Rubinsztejn 2004] est composée de plusieurs éléments. Dans le cas de *MoCA*, il s'agit d'une architecture client-serveur pour laquelle les auteurs proposent une série d'APIs (une pour le client et une pour le serveur), un canevas pour la construction des *proxies* dédiés à l'application, et une série de services prédéfinis qui supportent plusieurs aspects de l'application collaborative. Ces services remplissent plusieurs fonctionnalités clés pour le système, telles que l'acquisition de l'information relative à la localisation de chaque utilisateur, ou encore le suivi des changements dans le contexte des utilisateurs. Néanmoins, les informations contextuelles utilisées par *MoCA* se résument aux caractéristiques du réseau sans fil de type *WiFi* (puissance des signaux, localisation du point d'accès, etc.) qui sont utilisés notamment par le service responsable d'inférer la localisation de l'utilisateur.

À travers l'architecture *MoCA*, Rubinsztejn *et al.* [Rubinsztejn 2004] ont conçu *NITA,* une application sensible au contexte de type « *geonotes* », c'est-à-dire une application qui permet aux utilisateurs de déposer des messages (et de fichiers) concernant une région

[48] http://www.w3.org/TR/xslt.

physique donnée. Dans *NITA*, chacune de ces régions est représentée par une région symbolique. Ainsi, tout utilisateur qui entre dans une telle région, et qui dispose des autorisations nécessaires, recevra les messages qui ont été déposés dans cette région symbolique (une salle de conférence, par exemple). Par ailleurs, *NITA* permet à tout utilisateur désirant envoyer un message de fixer la destination du message (c'est-à-dire, la région symbolique à laquelle il est destiné), les utilisateurs qui pourront le consulter, et la période de temps pendant laquelle le message sera disponible. Les lecteurs des messages peuvent, pour leur part, modifier leur propre statut (visible aux autres utilisateurs ou non), choisir les types de messages à recevoir et le mode de réception (si les messages sont immédiatement délivrés ou s'ils sont gardés pour une consultation *a posteriori*). De plus, *NITA* dispose d'un outil de *chat* qui associe automatiquement une salle de discussion à toute région symbolique, visant essentiellement à permettre aux personnes présentes dans cette réunion de discuter librement à travers leurs dispositifs mobiles (par exemple, permettre des discussions entre les participants lors d'une conférence).

Conformément à ce que nous avons discuté dans les sections précédentes, la notion de contexte ne se limite pas toujours à la simple localisation de l'utilisateur et aux dispositifs qu'il utilise pour accéder à un système. Pour plusieurs applications, telles que les guides touristiques, ces informations peuvent être suffisantes pour parvenir à un comportement sensible au contexte satisfaisant au regard de l'utilisateur. Pour d'autres, ceci peut ne pas être le cas. Selon Grudin [Grudin 2001], une importante distinction doit être faite entre les applications visant un individu seul, et les systèmes où l'interaction entre personnes est l'élément central. Pour cet auteur, lorsqu'on considère la communication entre un groupe de personnes, la complexité des questions liées au contexte augmente sensiblement. À partir du moment où on considère les applications coopératives (les collecticiels), il est important de considérer la possibilité d'utiliser des éléments de contexte autres que la localisation et le dispositif client. Il faut également considérer les conséquences que les interactions entre les utilisateurs dans le cadre d'une coopération peuvent avoir dans l'analyse du contexte.

Muñoz *et al.* [Muñoz 2003] ont proposé un exemple d'application qui considère les utilisateurs dans un environnement coopératif à travers un système de messagerie instantanée sensible au contexte pour un hôpital. Un hôpital est ici vu comme un environnement coopératif (au sein duquel les acteurs coopèrent dans le traitement des patients) qui comporte des exigences particulières. Ces exigences se réfèrent notamment à une communication intense et temporellement critique (il faut absolument que chaque message arrive au bon moment) entre les participants, laquelle est basée essentiellement sur les rôles joués par les uns et les autres, et non sur l'identité de chacun. Pour ces auteurs, la notion de contexte dans un tel environnement est plus que la simple localisation d'un utilisateur. Elle doit inclure d'autres éléments comme le moment pendant lequel se déroule l'action (l'aspect temporel), la localisation d'une personne, mais aussi un équipement ou un objet (une imprimante, les résultats de l'examen d'un patient, etc.), et les rôles joués par chacun. Ainsi, une messagerie instantanée (voir Figure 14) est proposée qui permet à un utilisateur d'indiquer le contexte dans lequel un message est délivré. Par exemple, un utilisateur (un médecin, par exemple) peut envoyer un message aux utilisateurs qui jouent un rôle donné (le prochain médecin de garde, l'infirmière, etc.) et qui sont localisés à un endroit donné (une chambre de l'hôpital...), à partir d'un certain moment (le soir même, à partir de 20h, le lendemain matin...).

Même si Muñoz *et al.* [Muñoz 2003] proposent d'autres éléments en plus que la localisation de l'utilisateur, ces auteurs ne sont pas allés au-delà de la notion de rôle, n'explorant pas d'autres concepts qui peuvent se montrer intéressants pour la coopération entre les acteurs (les activités prévues, par exemple). De plus, ces auteurs n'explorent pas une représentation de contexte précise, laissant croire que les informations relatives au contexte d'un utilisateur sont transmises, et peut-être même représentées, avec les messages, à travers

le protocole IETF XMPP[49]. Par ailleurs, le système de messagerie instantanée proposé ne permet pas aux utilisateurs d'exprimer leurs propres préférences, en indiquant, par exemple, quel type de message ils souhaitent ou non recevoir. Ceci est probablement en raison des exigences de l'environnement hospitalier. Cependant, il faut souligner que, même dans ce cas, puisque les utilisateurs sont exposés à tous les messages, ils sont également exposés à un risque de surcharge cognitive si le nombre de messages devient trop important.

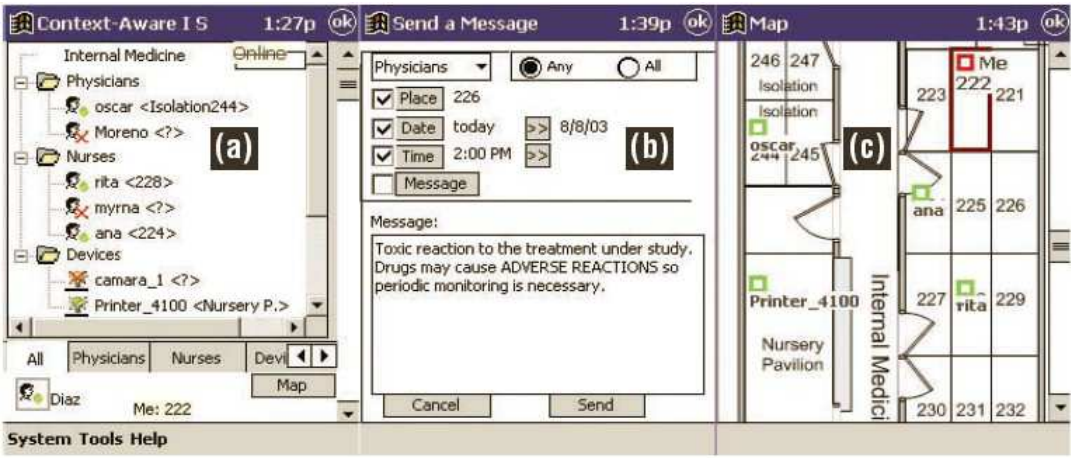


Figure 14. Exemple d'utilisation de la messagerie instantanée proposé par Muñoz et al. [Muñoz 2003]. En (a) une vue globale sur la situation et sur la localisation des collègues ; en (b) l'envoi d'un message adressé à un contexte déterminé ; et en (c) une visualisation du plan de l'hôpital avec la localisation des collègues et des équipements (d'après [Muñoz 2003]).

Un autre exemple de système sensible au contexte appliqué à un environnement coopératif est l'architecture *AWARE* [Bardram 2004]. Celle-ci est basée sur l'hypothèse selon laquelle les informations relatives au contexte courant des utilisateurs peuvent être utilisées comme un paramètre pour initier une conversation entre deux collègues. Le partage des informations sur le contexte des personnes avec lesquelles un utilisateur a des contacts (ses collègues, ses amis, sa famille, etc.) peut l'aider à déterminer si son interlocuteur est disponible pour une conversation, et ainsi créer plusieurs opportunités de communication [Schilit 2002a]. Pour Bardram et Hansen [Bardram 2004], le fait d'initier une conversation peut être la source d'interruptions pour son interlocuteur qui peuvent être réduites par la connaissance du contexte de cet interlocuteur. Selon ces auteurs, certaines informations observées par les systèmes sensibles au contexte sont capables de fournir aux membres d'un groupe des indications sur leurs collègues, des indications concernant l'information de *conscience de la communauté*.

L'architecture *AWARE* combine ainsi des composants de collecticiels pour le support à la conscience de groupe (voir section 2.2) avec certains composants de systèmes sensibles au contexte. Cette architecture est munie d'une couche pour la gestion des informations de conscience de groupe qui se positionne au-dessus d'une couche de gestion du contexte. Cette dernière est, pour sa part, basée sur l'infrastructure *JCAF*, dont nous avons parlé précédemment dans ce chapitre. Le résultat est donc une architecture pour la conception d'applications capables de permettre aux participants d'une équipe d'initier une conversation avec leurs collègues au moment le plus approprié pour eux, à travers l'utilisation d'une information de conscience de groupe enrichie avec des informations sur le contexte de chaque

[49]En anglais, « *Extensible Messaging and Presence Protocol* » - http://www.jabber.org/protocol/, RFC 3920 – http://www.ietf.org/rfc/rfc3920.txt, RFC 3921 – http://www.ietf.org/rfc/rfc3921.txt.

participant. Cependant, l'architecture *AWARE* ne précise pas quels éléments de ce contexte sont les plus intéressants pour que cette communication soit réellement appropriée.

En ce qui concerne l'utilisation de l'information de conscience de groupe par des utilisateurs nomades, il est intéressant également de souligner les applications qui explorent la présentation des informations sur le contexte des collègues, notamment leur localisation et parfois le dispositif qu'ils utilisent. C'est le cas d'*AwareNex* [Tang 2001] et de *handiMessenger* [Hibino 2002]. Ces applications, dont l'objectif est de favoriser les opportunités de communication entre les membres d'un groupe, présentent à leurs utilisateurs des informations sur la dernière localisation connue de leurs collègues (sous la forme de régions symboliques, telles que "le bureau", "la maison", etc.), sur leur activité courante (dans le cas de *AwareNex*) ou sur le dispositif utilisé (dans le cas de *handiMessenger*). Ces informations, même si elles sont basiques par rapport à des systèmes sensibles au contexte, peuvent être considérées comme étant des éléments du contexte de ces utilisateurs. Néanmoins, ces travaux traitent ces informations comme une partie du mécanisme de conscience de groupe. On voit clairement que la notion de contexte est très liée, dans les collecticiels, à la notion de conscience de groupe, ce qui a été également mis en évidence par Leiva-Lobos et Corrubias [Leiva-Lobos 2002].

3.5 Conclusions

Dans ce chapitre, nous avons abordé notamment la thématique de l'*informatique sensible au contexte*. On a observé que l'utilisation des technologies mobiles, telles que les dispositifs mobiles (ordinateurs de poche ou portables, téléphones cellulaires, etc.) et les réseaux sans fil (notamment, *WiFi* et *Bluetooth*), pose plusieurs défis liés notamment aux contraintes présentées par ces technologies et au nomadisme des utilisateurs. Grâce à ce nomadisme, les systèmes, notamment les systèmes sur le Web, sont désormais accessibles à partir des situations diverses qui le plus souvent n'ont pas été prévues au départ par ces systèmes.

Les limitations des nouvelles technologies modèrent leurs atouts en termes de flexibilité d'accès et d'utilisation des systèmes. Les inconvénients majeurs des dispositifs mobiles incombent en premier lieu à leurs limites techniques par rapport aux dispositifs fixes : capacités d'affichage et de mémoire réduites, processeurs moins performants, durée de vie restreinte des batteries, etc. Les réseaux sans fil, pour leur part, pâtissent d'un débit plus faible que celui des réseaux filaires et de problèmes de déconnexions fréquentes [Jing 1999] [Canals 2002].

Un utilisateur nomade (cf. section 3.2) se différencie d'un utilisateur traditionnel par la liberté dont il dispose pour se déplacer, passer d'un dispositif à un autre, etc. Or, les utilisateurs nomades sont souvent limités (ou gênés) par l'utilisation de systèmes non adaptés à leurs contraintes matérielles. Par conséquent, il devient crucial de doter ces systèmes, notamment les systèmes sur le Web, de capacités d'adapter leur offre, en termes de diffusion d'information (par le choix du contenu et de la présentation) et de mise à disposition de services, à ces utilisateurs nomades en tenant compte des contraintes auxquelles ceux-ci sont exposés. Par ailleurs, les conditions matérielles et logicielles dans lesquelles s'effectue l'utilisation du système via les technologies mobiles sont d'une part réduites, hétérogènes d'un utilisateur à l'autre mais également pour un même utilisateur, avec l'utilisation de différents dispositifs, avec les déplacements de l'utilisateur, en somme, avec les différentes situations dans lesquelles l'utilisateur peut se trouver au moment de son accès au système. Ainsi, un support approprié aux utilisateurs nomades demande une adaptation au contexte

d'utilisation courant. Cette adaptation au contexte est le but des *systèmes sensibles au contexte*.

L'informatique sensible au contexte s'intéresse à l'utilisation de la notion de contexte afin de concevoir des applications capables d'avoir un comportement adapté à une utilisation nomade. La conception de telles applications pose des défis liés notamment à l'acquisition, à la représentation et à l'utilisation de la notion de contexte pour l'adaptation des services, du contenu, ou de la présentation de ce contenu.

Ainsi, une question qui se pose est la représentation de la notion de contexte. Certains travaux (cf. section 3.3) ont proposé des modèles, plus ou moins complexes, pour la représentation de cette notion clé de l'informatique sensible au contexte. Néanmoins, ces modèles sont souvent difficiles à faire évoluer dans le temps (comme les représentations par XML et CC/PP, ou encore les ontologies). Il n'est pas rare non plus que certains modèles ignorent le dynamisme naturel à l'information de contexte, ou encore l'expression des critères de précision de cette information. Plus important encore, la majorité de ces modèles ne traite pas la question de la représentation partielle du contexte. Les modèles que nous avons étudiés ne gèrent pas les situations dans lesquelles le système est dans l'incapacité de faire l'acquisition de tous les éléments de contexte décrits par la représentation. Or, on sait que le processus d'acquisition du contexte n'est pas infaillible. Il peut toujours retourner des informations erronées, incomplètes, ou même ambiguës, particulièrement lorsqu'on utilise plusieurs capteurs pour la détection d'un élément de contexte donné.

L'analyse des représentations proposées dans la littérature et des caractéristiques propres à la notion de contexte et au processus d'acquisition (cf. section 3.3) permet de dégager certains points souhaitables pour des modèles visant la représentation de l'information de contexte : (*i*) *être évolutif*, c'est-à-dire pouvoir évoluer, soit par l'ajout, soit par la modification des nouveaux éléments de contexte ; (*ii*) *permettre la représentation d'une information au caractère dynamique*, dont la valeur peut changer rapidement ; (*iii*) *permettre l'expression de critères de précision* (ou de qualité) des informations ; (*iv*) *permettre l'expression d'informations ambiguës et incomplètes*. Les modèles décrits ici ne respectent pas toutes ces exigences. Et même si on considère une représentation de contexte qui respecte ces prérequis, il reste toujours la question de l'utilisation de ces informations dans le système.

La construction d'un système sensible au contexte est actuellement une tâche complexe. En l'absence d'outil, le développement de systèmes sensibles au contexte est une tâche difficile nécessairement réalisée au cas par cas [Rey 2004]. La plupart des systèmes sensibles au contexte éludent la question de la définition, et certains éludent même la question de la représentation du contexte. La majorité des systèmes exploite une notion de contexte qui se limite à la localisation de l'utilisateur et aux caractéristiques de son dispositif (voir, par exemple, [Burrell 2002]). Ces systèmes ne séparent pas non plus les informations qui font référence au contexte de celles qui sont relatives au domaine d'application traité par le système [Chaari 2004]. Par conséquent, ces systèmes deviennent difficiles à concevoir, et les composants liés à l'acquisition et à la manipulation du contexte ne sont pas (ou sont très peu) réutilisables par d'autres systèmes. Par exemple, l'étude de *Campus Aware* [Burrell 2002] ou de *MoCA* [Rubinsztejn 2004] relève l'inexistence d'une représentation propre et dédiée au contexte. Dès lors, en l'absence d'une formalisation de la notion de contexte, l'introduction et l'exploitation dans ces systèmes d'un nouvel élément de contexte, autre que la localisation comme, par exemple, les activités de l'utilisateur, ne peuvent se faire facilement, ni reposer sur la réutilisation de composants préexistants. On perçoit, à travers cet exemple, l'importance de l'utilisation d'un modèle de contexte qui formalise cette notion. Cette formalisation est, à notre avis, un prérequis nécessaire à la conception de systèmes sensibles au contexte.

Par ailleurs, les représentations du contexte ainsi que les systèmes sensibles au contexte ne se sont intéressés, pour leur majorité, qu'aux utilisateurs en tant qu'individus. Toute l'analyse des éléments de contexte se fait en tenant compte de l'individu isolé. Or, les questions autour de l'interaction entre un groupe de personnes devraient, selon Grudin [Grudin 2001], être prises en compte lorsqu'on considère la notion de contexte dans sa signification la plus large. Il existe donc encore aujourd'hui un besoin par rapport à l'analyse du contexte pour des utilisateurs qui sont membres d'un groupe. Ceci a une importante répercussion pour les collecticiels, puisque les utilisateurs de ces systèmes sont engagés dans un processus de coopération. À ce jour, on ne trouve, dans la littérature que quelques propositions, comme celle de Muñoz *et al*. [Muñoz 2003] (voir section 3.4), qui se limitent à l'analyse de quelques éléments de contexte tels que la localisation de l'utilisateur et son rôle dans le groupe.

Pour conclure, il est important également de considérer le rôle que l'utilisateur lui-même tient dans le processus d'adaptation réalisé par les systèmes sensibles au contexte. Selon Dey [Dey 2001], l'utilisateur est en meilleure position pour spécialiser une application sensible au contexte à ses besoins personnels. Cependant, on observe que la majorité des systèmes sensibles au contexte ne tiennent que très peu compte des préférences de l'utilisateur pour faire de l'adaptation. Les systèmes qui prennent en compte les préférences de l'utilisateur emploient des critères tels que l'âge de l'utilisateur (comme dans *GUIDE* [Cheverest 2002]) ou ses centres d'intérêts (comme dans *Guilliver's Genie* [O'Hare 2002]), sans les mettre en relation avec l'information de contexte. Il s'agit, dans ces cas, des préférences de l'utilisateur valables quel que soit le contexte dans lequel celui-ci se trouve. Or, étant donné le fait que cet utilisateur est nomade, il peut donc se trouver dans des contextes très variés, aux contraintes également distinctes, tant du point de vue technique (dispositif ou réseau aux capacités limitées, par exemple), que du point de vue social (on n'agit pas de la même façon au bureau, à la maison, ou dans un train, etc.). Nous pensons donc que les préférences d'un utilisateur peuvent changer en fonction de l'environnement dans lequel il se trouve. Ceci est d'ailleurs l'avis de Greenberg [Greenberg 2001], pour qui les actions que les gens accomplissent, ainsi que leurs attentes par rapport à un système sensible au contexte, dépendent directement du contexte. Ainsi, un système qui ne prend en compte qu'un ensemble de préférences fixe et indépendant du contexte risque autant qu'un système qui n'observe pas les préférences personnelles de l'utilisateur, d'avoir un comportement ou de fournir des informations qui ne correspondent pas aux attentes de l'utilisateur.

4 Problématique et Approche

Dans la partie « état de l'art » du présent document, nous avons abordé les thématiques fondamentales de cette thèse : le travail coopératif et la conscience de groupe, l'informatique mobile et l'informatique sensible au contexte. Nous avons présenté, dans les chapitres 2 et 3, une vue globale de ces thématiques et de leurs problèmes. L'analyse de ces chapitres a permis ainsi de dégager certaines questions qui restent encore aujourd'hui ouvertes.

En ce qui concerne le travail coopératif et la conscience de groupe, nous avons observé une tendance forte à l'utilisation des technologies mobiles pour l'accès aux collecticiels, particulièrement les collecticiels sur le Web. Les solutions mobiles constituent une opportunité intéressante pour le travail en groupe [Alarcón 2004]. Par exemple, grâce aux nouvelles technologies mobiles, un utilisateur nomade peut garder à tout moment le contact avec ses collègues de bureau (et vice-versa), ce qui facilite grandement le travail en groupe. Cette tendance confirmée dans les études menées par Perry *et al.* [Perry 2001]. Ces auteurs ont étudié le comportement de travailleurs nomades et ont observé que ces employés ont besoin de garder le contact avec leurs bureaux d'origine lorsqu'ils sont engagés dans un quelconque processus collaboratif (par exemple, développer un produit ou coordonner un projet). Ces professionnels ont besoin d'informations pertinentes liées à ce processus (des informations relatives aux collègues, aux échéances, etc.) afin d'améliorer son exécution. En d'autres termes, les travailleurs nomades ont besoin d'informations de *conscience de groupe*, même (et peut-être principalement) lorsqu'ils ne sont pas dans leurs bureaux respectifs. De plus, Guerrero *et al.* [Guerrero 2004] observent que dans le cadre du travail coopératif certaines activités sont réalisées de manière individuelle. Ainsi, pour ces auteurs, les ordinateurs de poche (et autres dispositifs mobiles) sont potentiellement utiles pour ces activités individuelles dans le cadre d'une coopération, et cette hypothèse se renforce lorsque ces activités exigent une certaine mobilité des participants.

À partir du moment où nous considérons que l'utilisateur emploie les nouvelles technologies mobiles pour accéder aux informations dans un collecticiel, nous pouvons supposer qu'il dispose (cf. chapitre 3), pour recevoir les informations, d'un dispositif avec une capacité d'affichage et une autonomie de batteries limitées, ainsi que d'une connexion réseau avec un faible débit, parmi d'autres contraintes physiques. Nous pouvons également supposer que cet utilisateur dispose d'une courte période de temps pour interpréter ces informations, puisque l'utilisation des dispositifs mobiles se fait souvent sur de courtes périodes (par exemple, accéder à un système pendant son temps libre à l'aéroport ou en tirant profit d'une disponibilité temporaire du réseau sans fil). Toutes ces contraintes ont un impact sur les mécanismes de conscience de groupe qui ne peuvent plus exploiter des techniques de visualisation élaborées (comme celle proposée par [Bouthier 2004]), ni envoyer des quantités importantes d'information au client.

La conception de collecticiels pour les dispositifs mobiles (tels que les ordinateurs de poche, les téléphones cellulaires, etc.) pose ainsi certains défis principaux par rapport à la conception de collecticiels pour les ordinateurs de bureau [Tang 2001], [Guerrero 2004]. Tang *et al.* [Tang 2001] énumèrent comme difficultés principales : *(i)* la taille des écrans des dispositifs mobiles qui est extrêmement limitée ; *(ii)* le fait que ces dispositifs sont consultés par intermittence, au lieu d'une observation plus ou moins continue telle qu'elle s'effectue sur un ordinateur de bureau (ce qui affecte la perception de la conscience de groupe, notamment la conscience périphérique) ; et *(iii)* les contraintes inhérentes à la bande passante, le débit, la couverture, et la fiabilité des connexions réseaux de ces dispositifs. À ces difficultés,

s'ajoutent des ressources pour la saisie et le stockage des informations très limitées, la courte durée de vie des batteries, et la lenteur des processeurs [Guerrero 2004].

Néanmoins, les études menées par Guerrero *et al.* [Guerrero 2004] montrent que les ordinateurs de poche proposent également une série d'avantages pour leurs utilisateurs. Ces opportunités incluent une grande portabilité, des temps de réponse et de démarrage courts, et une capacité pour la présentation en petite quantité d'informations. De tels atouts rendent l'utilisation de ces dispositifs intéressante dès lors que : *(i)* la mobilité de l'utilisateur est une préoccupation centrale pendant l'utilisation du système ; *(ii)* les utilisateurs doivent travailler dans des endroits inhabituels, encombrés ou inconfortables (dans la rue ou dans les transports publics, par exemple) ; *(iii)* les utilisateurs peuvent, sur une courte période de temps, prendre des notes, qui pourront donner lieu ensuite à des contributions plus importantes ; *(iv)* l'utilisation de ces dispositifs a lieu lors de périodes de travail individuel divergentes ou différentes de celles des autres membres (suivies ensuite de périodes de travail convergentes, probablement avec des interactions synchrones).

L'ensemble des opportunités à exploiter et des défis à surmonter découle donc de l'émergence des nouvelles technologies mobiles dans les collecticiels. Leur exploitation ajoute de nouvelles contraintes aux défis traditionnels de la conception des collecticiels. Les problèmes tels que la surcharge cognitive sont toujours présents, mais ils sont, en partie, accentués par l'environnement mobile. Pour Benali *et al.* [Benali 2002], l'évolution actuelle du TCAO passe, entre autres, par la prise en compte de leur mobilité et de l'utilisation de dispositifs légers. En plus de l'accès aux informations, de l'interaction et de la collaboration par la virtualisation de l'espace et du temps, il ajoute la mobilité des personnes, leur plus grande disponibilité et la contextualisation de leurs actions et des informations grâce à la prise en compte du contexte précis, de la localisation des utilisateurs et des objets sur lesquels portent le travail.

Lorsque nous considérons la question de la conscience de groupe dans les collecticiels sur le Web, nous avons vu, dans le chapitre 2, que les membres d'un groupe ne sont pas intéressés ni même concernés par toutes les informations de conscience de groupe, mais seulement par des sous-ensembles de ces informations. En réalité, les utilisateurs n'ont besoin que des informations qui peuvent contribuer directement ou indirectement à leurs activités courantes. Dans le cadre d'une utilisation nomade, encore plus qu'ailleurs, l'utilisateur n'est pas intéressé par toutes les informations dont le système dispose. Il privilégie les informations relatives aux activités auxquelles il participe (par exemple, la rédaction du rapport annuel), ou encore celles qui ont un lien avec sa localisation actuelle (par exemple, une réunion qui se tiendra dans une salle à côté), le tout dans la mesure des capacités dont il dispose.

À partir de ces considérations, nous constatons que l'utilisation des nouvelles technologies mobiles fait en sorte que les collecticiels et les mécanismes de conscience de groupe confrontent les utilisateurs de ces technologies à un risque de surcharge accrû. L'utilisateur est dans un environnement plus contraignant, il ne dispose pas du même temps pour interpréter les informations, et il ne s'intéresse pas forcément aux mêmes informations que lorsqu'il était dans un environnement fixe (dans son bureau, par exemple). Ses intérêts peuvent changer en fonction de son activité, de sa localisation, et même des ressources matérielles dont il dispose. Dans un tel cas, le filtrage de l'information de conscience de groupe tel qu'il est fait à ce jour (cf. section 2.2) semble inapproprié, puisque, comme Bouthier [Bouthier 2004] l'a souligné, ce filtrage ne prend pas en compte les activités couramment réalisées par l'utilisateur, alors que la pertinence (ou non) d'une information de conscience de groupe dépend de l'activité courante de l'utilisateur qui la reçoit, puisque les intérêts de cet utilisateur peuvent changer en fonction de son activité.

Nous pouvons donc conclure qu'*il existe un besoin clair, de la part des collecticiels sur le Web, de mécanismes de conscience de groupe qui soient capables d'adapter le contenu des informations délivrées aux utilisateurs aux intérêts, au(x) rôle(s) dans le groupe, et à l'activité courante de chaque utilisateur, ainsi qu'à sa situation courante : sa localisation, son dispositif client, etc.* L'adaptation vise ici l'optimisation du travail de l'utilisateur, et par suite, celui du groupe auquel il appartient. De plus, ces mécanismes doivent concentrer leurs efforts sur le filtrage des informations du côté du serveur, puisque nous ne pouvons plus supposer l'utilisation de clients puissants (comme le sont les ordinateurs de bureau et les ordinateurs portables), avec une capacité d'affichage suffisamment importante pour les techniques de visualisation.

Nous avons donc un besoin clair de mécanismes de conscience de groupe qui soient sensibles au contexte. En d'autres termes, il existe une tendance affirmée à faire évoluer les collecticiels vers de collecticiels dits « sensibles au contexte », c'est-à-dire, des collecticiels qui se comportent également comme des systèmes sensibles au contexte. Dans ce travail, nous considérons un système sensible au contexte (cf. section 3.1) comme *un système qui est capable de détecter, à un instant t, un ou plusieurs éléments du contexte dans lequel se trouve l'utilisateur et de fournir, en retour, soit des informations, soit des services adaptés à ce contexte et aux changements de un ou plusieurs éléments de ce contexte*.

Cependant, les systèmes sensibles au contexte se concentrent majoritairement sur l'adaptation du contenu ou de la présentation de ce contenu à la localisation de l'utilisateur et aux caractéristiques de son dispositif d'accès (cf. section 3.4). On remarque que la majorité des travaux dans le domaine de l'informatique sensible au contexte, adoptent une notion de contexte qui est limitée au contexte purement physique de l'utilisateur. Par ailleurs, les systèmes et les architectures présentés dans la section 3.4 n'abordent pas de manière explicite le modèle de représentation de contexte utilisé (s'il existe). Par conséquent, on peut s'interroger sur l'impact qu'un possible changement dans l'ensemble d'éléments de contexte (par exemple, la prise en compte d'un nouvel élément) aura dans ces systèmes et dans les systèmes construits avec les architectures citées précédemment.

Par ailleurs, la majorité des systèmes sensibles au contexte considère l'utilisateur en tant qu'individu isolé. Or, un collecticiel qui se veut sensible au contexte doit, au contraire d'autres systèmes sensibles au contexte, considérer l'utilisateur nomade en tant que participant d'un ou plusieurs processus coopératifs, pour lesquels il collabore avec d'autres membres d'un groupe. En tant que membre d'un groupe, cet utilisateur a besoin de garder le contact avec ses collègues et d'être conscient des activités réalisées dans le groupe (et qui concernent son propre travail). Par conséquent, outre les éléments qui décrivent la situation physique de l'utilisateur en tant qu'individu (tels que sa localisation et son dispositif actuel), plusieurs éléments relatifs au groupe et au processus coopératif doivent également faire partie du contexte de cet utilisateur nomade. Dès lors, il apparaît nécessaire d'élargir la notion de *contexte d'utilisation* pour prendre en compte, dans un processus d'adaptation, tout ce qui relève de l'information relative au processus collaboratif dans lequel il est impliqué.

Ainsi, nous croyons qu'est nécessaire la proposition d'une représentation de la notion de contexte propre aux collecticiels, capable à la fois de respecter les prérequis énumérés dans la section 3.5, et de prendre en compte le processus de collaboration propre à chaque collecticiel. Une telle représentation est nécessaire pour l'adaptation du contenu et, particulièrement, pour le filtrage des informations au sein des collecticiels. Selon Coppola *et al.* [Coppola 2003], les utilisateurs nomades attendent des technologies mobiles un filtrage des informations disponibles de manière que seules les informations pertinentes soient présentées. Par ailleurs, puisque les risques de surcharge cognitive sont accrûs par l'emploi des technologies mobiles, le filtrage des informations de conscience de groupe devient encore

plus critique pour tout collecticiel accessible à travers ces technologies (donc, pour tous les collecticiels sur le Web, qui sont à présent déjà confrontés à ces technologies). Ceci a été démontré par les expériences menées par Hibino et Mockus [Hibino 2002], dans lesquelles les utilisateurs ont exprimé, lors de l'analyse du système *handiMessenger*, un intérêt particulier par la personnalisation des informations délivrées par le système, en demandant notamment la possibilité de filtrer les messages qui leur sont délivrées.

Cependant, Billsus *et al.* [Billsus 2002] argumentent que, pour que les informations soient réellement accessibles aux technologies mobiles, les systèmes ne doivent pas se limiter au filtrage des informations, mais ils doivent également les ordonner de manière à proposer à l'utilisateur d'abord les plus pertinentes. D'autre part, le fait d'avoir une liste d'informations pertinentes peut s'avérer inutile si cette liste est trop longue et exige un temps pour l'examiner supérieur au temps disponible pour cela ou encore supérieur à la période de temps pendant laquelle les informations sont pertinentes [Coppola 2003]. *Il faut donc un filtrage plus "intelligent" de l'information de conscience de groupe, capable non seulement de limiter l'ensemble d'informations délivrées à l'utilisateur aux informations pertinentes par rapport à son contexte courant, mais également capable d'organiser ces informations de manière à permettre à l'utilisateur d'accéder à ces informations progressivement, des plus pertinentes aux moins pertinentes, selon sa disponibilité de temps et de ressources.*

Par ailleurs, nous avons pu observer que les systèmes sensibles au contexte (cf. section 3.4) comme les collecticiels sur le Web (cf. section 2.3.2), ne considèrent que modérément les préférences de l'utilisateur afin d'adapter leur comportement, soit en termes de fonctionnalités, soit en termes de données fournies. L'utilisateur n'est que très peu impliqué dans le processus d'adaptation. Dans les systèmes sensibles au contexte, même lorsque les préférences de l'utilisateur sont prises en compte, ceci est fait indépendamment du contexte d'utilisation, Or, comme nous avons remarqué dans le chapitre 3, les préférences d'un utilisateur nomade peuvent varier conformément au contexte dans lequel il se trouve.

De plus, nous avons observé également que les utilisateurs d'un collecticiel, même les utilisateurs nomades, acquièrent certaines habitudes d'utilisation au fur et à mesure qu'ils incorporent le collecticiel dans leur quotidien. Par exemple, un utilisateur qui voyage souvent tend à s'habituer rapidement à utiliser le temps mort dans les aéroports ou dans les gares pour se connecter (dès lors une connexion réseau est disponible) et lire ses messages (et probablement continuer son travail). De même pour les utilisateurs qui disposent d'un long temps de trajet quotidien dans les transports en commun. Celui-ci est, d'ailleurs, un phénomène observé dans les grandes villes, où les gens utilisent habituellement leur téléphone cellulaire ou leur ordinateur de poche pour continuer (ou anticiper) le travail lors de leurs déplacements. Ceci est d'autant plus observable lorsque l'utilisateur se trouve, avec son équipe, dans un moment sensible de la coopération : par exemple, pendant la réalisation d'une activité critique ou à l'approche d'une échéance importante. Dans ces situations de pression, les utilisateurs profitent souvent de la moindre opportunité pour continuer leur travail. D'autre part, nous avons observé que les collecticiels, en tant qu'outils de travail, se prêtent facilement à une utilisation confinée, c'est-à-dire, dans des zones bien définies.

Nous pouvons alors considérer que, même si les utilisateurs nomades se caractérisent par la variation de contexte dans lequel ils peuvent se trouver au moment de leur accès au système, cette variation n'est pas aussi significative pour les collecticiels que pour les systèmes pervasifs. Pour les collecticiels, nous avons observé que, dans la majorité des cas, les utilisateurs nomades utilisent ces systèmes dans des situations plus ou moins connues à l'avance, et qui ont souvent des éléments de contexte en mesure de les caractériser (par exemple, l'accès à partir de la maison, au bureau, en déplacement, à travers l'ordinateur portable, le PDA, le téléphone cellulaire, lorsqu'une échéance s'approche...). L'imprévisibilité

complète du contexte d'utilisation se résume normalement à une minorité des cas d'utilisation. Il est donc possible à un utilisateur d'associer à ces situations connues des préférences qui se forment au fur et à mesure des utilisations. Une telle association permettrait une adaptation plus appropriée du système tant au contexte de l'utilisateur qu'à ses préférences.

Par ailleurs, il faut d'ores et déjà considérer que l'utilisation de mécanismes de visualisation de l'information très élaborés est déconseillé étant donné les contraintes qui caractérisent les dispositifs mobiles. L'envoi d'une importante quantité des données par le réseau est, lui aussi, déconseillé sur les réseaux sans fil, étant donné les caractéristiques de ces réseaux. Cependant, il est intéressant d'observer que, même si la littérature s'accorde sur l'impact des contraintes imposées par l'utilisation de ce type de réseau, il n'existe que très peu de systèmes sensibles au contexte qui adaptent leur comportement aux capacités du réseau. Ce phénomène se doit notamment au dynamisme des réseaux sans fil, dont le débit varie en fonction de nombreux facteurs, et à la complexité de l'acquisition des informations concernant l'état courant du réseau (voir [Marchand 2004] et [Heusse 2003]). La plupart des systèmes sensibles au contexte adoptent une approche qu'on peut appeler "indirecte", qui vise à optimiser ou à minimiser l'utilisation du réseau quel que soit le type, à travers, par exemple, l'utilisation de *cache*, comme dans [O'Hare 2002], ou d'un mécanisme de filtrage, comme dans [Hibino 2002]. Tout ceci renforce donc l'intérêt pour le filtrage de l'information de conscience de groupe en amont, afin que l'utilisation du réseau soit minimale et que le risque majeur de surcharge soit réduit.

En somme, à partir des observations faites dans les chapitres 2 et 3, en ce qui concerne les collecticiels sur le Web (qui sont particulièrement accessibles à travers les technologies mobiles) et notamment le support à la conscience de groupe (dont les risques de surcharge sont accrus), nous pouvons conclure que :

(a) il faut élargir la notion de contexte afin de tenir compte des aspects collaboratifs qui entourent les utilisateurs nomades dans un collecticiel ;

(b) il faut une représentation de contexte capable de formaliser cette notion élargie, respectant les prérequis définis dans la section 3.5 et facilement exploitable pour la conception des nouveaux systèmes ;

(c) les informations de conscience de groupe doivent être filtrées selon le contexte et les préférences de l'utilisateur, de manière à lui présenter les plus pertinentes ;

(d) les informations délivrées à l'utilisateur doivent être organisées de manière à permettre un accès facile et potentiellement progressif à ces informations.

Ainsi, dans ce travail, nous adoptons une approche qui vise l'adaptation de l'information de *conscience de groupe* fournie par les collecticiels sur le Web. L'objectif ici est de réduire les risques de surcharge cognitive auxquels les utilisateurs nomades sont exposés à travers la sélection des informations qui leur sont présentées. L'idée est de ne présenter que les informations pertinentes par rapport au contexte courant de l'utilisateur afin de ne pas le surcharger avec des informations inutiles en ce moment précis. Dans cette approche, nous traitons les quatre points ci-dessus à travers notamment : (*i*) la proposition d'un *modèle de représentation par objets du contexte* ; (*ii*) la proposition d'un ensemble d'*opérations capables de manipuler les instances de ce modèle* ; (*iii*) la définition d'un *processus de filtrage des informations de conscience de groupe* capable de sélectionner et d'organiser ces informations selon le contexte d'utilisation et les préférences de l'utilisateur. Cette approche est concrétisée par la spécification d'un *canevas nommé BW-*$\mathcal{M}$ qui met en place tous les éléments de l'approche.

La première partie de notre proposition concerne donc la *formalisation de la notion de contexte* à travers la proposition d'un *modèle à objets* qui représente cette notion. Celle-ci est basée sur la définition de contexte donnée par Dey [Dey 2000] (cf. section 3.3) et considère particulièrement les aspects coopératifs inhérents aux collecticiels. Par ailleurs, ce modèle prend en compte les quatre prérequis pour la représentation du contexte que nous avons déterminé dans la section 3.5, à savoir : (*i*) être évolutif ; (*ii*) permettre la représentation d'une information dynamique ; (*iii*) permettre l'expression de critères de précision ; et (*iv*) permettre l'expression d'informations ambiguës et incomplètes. Ces quatre prérequis sont, pour nous, *quatre critères indispensables* pour la représentation de contexte. Nous pensons qu'une telle formalisation est nécessaire à la conception de nouveaux systèmes sensibles au contexte coopératifs. L'enjeu ici est de concevoir des systèmes dont la notion de contexte puisse évoluer (et être modifiée) dans le temps, et dont les composants soient réutilisables. Ce modèle permet la représentation du contexte courant d'un utilisateur nomade, ainsi que la représentation des contextes potentiels dans lesquels cet utilisateur peut se trouver lors de ses interactions avec le collecticiel.

Ensuite, la deuxième partie de notre proposition se concentre sur la définition d'une série d'*opérations capables d'être appliquées au modèle par objets* que nous proposons. À travers ces opérations, nous voulons comparer, de différentes manières, les objets et les associations qui composent le modèle.

Puis, la troisième partie propose un *processus de filtrage guidé par le contexte* qui adapte l'information de conscience de groupe au contexte de l'utilisateur, ainsi qu'à ses préférences par rapport à ce contexte. À travers ce processus, l'information de conscience de groupe est filtrée et organisée en différents niveaux de détails, de manière à proposer à l'utilisateur un accès progressif à cette information. Ce processus de filtrage utilise, outre le modèle de contexte et les opérations que nous proposons, un ensemble de modèles sous-jacents qui représentent (respectivement) le contenu (l'information de conscience de groupe), l'accès à ce contenu à travers différents niveaux et les préférences de l'utilisateur.

Finalement, le *canevas BW-$\mathcal{M}$* qui implémente ce processus, et la méthodologie, d'application de ce canevas clôturent cette proposition.

L'approche adoptée par ce travail consiste donc à adapter l'information de conscience de groupe par le filtrage. Ce filtrage est guidé à la fois par le contexte d'utilisation et par les préférences de l'utilisateur. Cette approche vise le groupe à travers les individus qui le composent. À l'opposé des travaux sur la malléabilité, qui font de l'adaptation pour tous les membres du groupe sans distinctions (cf. section 2.3), nous avons choisi d'adapter les informations pour chaque individu selon son contexte courant. Nous basons ce choix sur l'hypothèse selon laquelle, pour améliorer la performance du groupe dans son ensemble, il faut d'abord améliorer les conditions de travail de chaque membre. Ainsi, nous adoptons une approche selon laquelle l'adaptation se fait pour l'individu pour qu'il puisse mieux agir dans le cadre de la coopération. Cependant, le fait de considérer chaque membre du groupe individuellement ne signifie pas, pour autant, que nous traitons les individus de manière isolée. Nous les considérons comme parties d'un collectif, le groupe, à travers notamment une notion de contexte qui est élargie par rapport aux travaux présentés dans le chapitre 3 et qui tient compte à la fois des aspects physiques du contexte (la localisation de l'utilisateur, par exemple), et des aspects directement liés à la coopération et au groupe (tels que les rôles joués par l'utilisateur).

Dans la prochaine partie de ce document (« proposition »), nous détaillons cette approche et tous les éléments qui la composent : le chapitre 5 présente le modèle de contexte, le chapitre 6 discute les opérations définies sur ce modèle, le chapitre 7 aborde le processus de filtrage, et le chapitre 8 présente le canevas BW-$\mathcal{M}$.

Proposition

5 Formalisation du Contexte

Dans ce travail, nous nous intéressons à l'adaptation de l'information de conscience de groupe au contexte d'utilisation dans un cadre coopératif et nomade. Conformément à la section 3.3, il n'existe pas de définition de contexte qui soit acceptée par tous. Dans ce travail, nous nous sommes donc inspirés de la définition donnée par Dey [Dey 2000]. Cette définition aborde, tout particulièrement, le développement des systèmes sensibles au contexte.

La majorité des systèmes sensibles au contexte manque de représentation formelle de la notion de contexte, et ceci malgré les propositions faites dans ce sens (voir section 3.3.2). La grande majorité de ces systèmes utilise encore les représentations *ad-hoc*, non formalisées (voir section 3.4). Parmi les raisons qui pousseraient les concepteurs à ignorer les propositions de représentations existantes, nous pouvons considérer le fait que ces représentations ne respectent pas tous les quatre critères que nous avons énumérés précédemment (cf. section 3.5). Par exemple, les modèles proposés sont souvent difficiles à modifier et à faire évoluer, la représentation d'informations incomplètes ou ambiguës n'est pas prise en compte, malgré le risque réel d'occurrence de ce type d'information en raison de possibles défaillances dans le processus d'acquisition du contexte.

Par ailleurs, la majorité des travaux que nous avons étudié dans le chapitre 3 envisage la notion de contexte dans le cadre des interactions entre un utilisateur isolé et le système. Nous adoptons ici un point de vue différent qui considère les interactions utilisateur/système *dans un environnement coopératif*. Ceci nécessite la prise en compte d'éléments relatifs aux processus de coopération impliquant cet utilisateur (que nous considérons, de plus, comme nomade, et donc susceptible d'employer divers dispositifs mobiles et de se déplacer pendant ou entre les connexions au système).

Pour répondre à ces préoccupations, nous proposons une représentation par objets de la notion de contexte. Cette représentation formalise la notion de contexte tout en séparant les informations contextuelles de celles propres au domaine d'application. Nous avons choisi l'utilisation d'un formalisme par objets de type classe/association, car ce formalisme est largement connu et maîtrisé par les concepteurs, à travers notamment l'utilisation d'UML [OMG 2004]. Nous proposons un modèle par objets, composé par un ensemble d'éléments de contexte qui sont représentés par des classes et d'associations. Nous discutons également les propriétés de ce modèle par rapport aux critères que nous avons définis, ainsi que les défis posés par sa mise en œuvre. Nous proposons une implémentation du modèle en tant que base de connaissances, à travers un système de représentation de connaissances par objets nommé AROM [Page 2000] [Page 2001]. Le système AROM propose des concepts et une notation qui sont propres à la représentation par objets. Ces atouts nous permettent de compléter notre formalisation de la notion de contexte.

Les prochaines sections décrivent le modèle et ses composants : la section 5.1 présente la représentation du contexte que nous proposons, avec les éléments et le modèle à objets qui la composent, tandis que la section 5.2 introduit les concepts du système AROM et la mise en œuvre du modèle à l'intérieur de ce système.

5.1 Une Représentation par Objets du Contexte

Nous proposons dans ce travail une représentation par objets de la notion de contexte. Cependant, la nature même de la notion de contexte exige qu'elle soit dirigée vers une finalité précise (cf. section 3.3). Dans le cas de ce travail, cette finalité est la représentation du contexte d'utilisation au sein des collecticiels sur le Web (semblables à *LibreSource* [Forest 2005a], à *ToxicFarm* [Skaf-Molli 2003] ou encore à *AllianceWeb* [Martínez 2002b]), lesquels doivent dorénavant être perçus également comme des systèmes sensibles au contexte. Par ailleurs, cette représentation vise particulièrement l'adaptation de l'information de conscience de groupe.

Ces objectifs nous ont conduits tout d'abord à considérer l'utilisateur, au contraire d'autres représentations, en tant que membre d'un ou plusieurs groupes et au titre de sa participation à un ou plusieurs processus coopératifs. Ceci nous a conduit ensuite à l'identification de certains éléments de contexte dont plusieurs sont liés à des processus de coopération qui nous semblent essentiels pour la représentation du contexte d'utilisation dans un cadre nomade et coopératif. Ces éléments sont organisés en un ensemble de classes et d'associations, qui représentent respectivement les éléments et les relations qu'ils entretiennent. Ces classes et ces associations sont illustrées à travers un ensemble de diagrammes UML, présentés ci-dessous (section 5.1.2), qui constitue le modèle de contexte que nous proposons.

La représentation proposée agit comme un modèle conceptuel, un schéma décrivant les éléments d'information à caractère contextuel et les relations que ces éléments entretiennent entre eux. Ce modèle peut être donc appliqué à différents collecticiels, à l'intérieur desquels celui-ci est mis en œuvre, et possiblement étendu selon les besoins du système.

Ainsi, nous débutons la description de cette représentation[50] par l'analyse des concepts que nous avons choisis de prendre en considération (cf. section 5.1.1). Puis, nous organisons ces concepts dans l'ensemble de diagrammes UML qui représentent notre modèle (section 5.1.2), avant d'analyser les propriétés du modèle (section 5.1.3) et les défis de sa mise en œuvre (section 5.1.4).

5.1.1 Éléments de contexte

La définition de contexte que nous adoptons [Dey 2000] considère que le contexte est construit à partir d'un ensemble d'éléments qui caractérisent une situation. L'identification de ces éléments est, par ailleurs, une étape décisive dans la conception d'un système sensible au contexte. Or, comme nous avons vu dans le chapitre 3, la majorité des travaux laisse cette tâche difficile totalement à la charge des concepteurs. Certaines représentations, comme celle de Henriciksen *et al.* [Henricksen 2002], ne donnent même pas de pistes pour les concepteurs qui souhaitent l'utiliser. Contrairement à ces travaux, nous avons voulu identifier un ensemble minimal d'éléments de contexte afin d'attirer l'attention des concepteurs sur certains éléments que nous considérons comme pertinents pour la représentation du contexte d'utilisation dans un cadre à la fois coopératif et nomade.

Cependant, étant donné l'impossibilité d'énumérer tous les éléments d'information pertinents (cf. section 3.3), il nous paraît peu réaliste de décrire de façon exhaustive les éléments qui composent le contexte d'utilisation, et ceci même dans un cadre précis, tel que

[50] Par souci de simplification du langage, nous ferons dorénavant référence à cette représentation simplement comme « *modèle de contexte* ».

l'utilisation d'un collecticiel sur le Web par un utilisateur nomade. Nous nous sommes donc centrés sur certains éléments qui nous semblent indispensables, soit de par leur importance à l'utilisateur, soit de par leur pouvoir de caractériser une situation. Notre but à travers l'identification de ces éléments est de fournir des suggestions, des pistes aux concepteurs.

En nous basant sur les travaux étudiés dans l'état de l'art (voir chapitres 2 et 3), et plus particulièrement sur les travaux de représentation de contexte (section 3.3.2), nous avons procédé à une analyse des informations jugées pertinentes pour l'identification du contexte d'un utilisateur nomade dans un environnement coopératif. Cette analyse a été réalisée selon cinq points de vue distincts : ***espace, outil, temps, communauté,*** et ***processus*** (voir Figure 15). Ces cinq points de vue cherchent à répondre aux questions élémentaires « *où ?* », « *comment ?* », « *quand ?* », « *qui ?* » et « *quoi ?* ». L'objectif est de déterminer les éléments d'information capables de répondre à chacune de ces questions : « où se déroule l'utilisation ? », « comment se déroule-t-elle ? », « quand se produit l'utilisation ? », « qui participe, directement ou indirectement, à l'utilisation ? », « à quoi se réfère-t-elle ? ». Les éléments qui permettent de répondre à ces questions sont des éléments capables de caractériser la situation dans laquelle un utilisateur se trouve : ils participent donc au contexte d'utilisation.

Ainsi, à travers le point de vue ***espace***, nous voulons savoir où se déroule l'interaction entre l'utilisateur et le système (question « *où ?* »). Ceci nous renvoie directement au concept de *localisation* physique de l'utilisateur. La localisation influe sur le comportement d'un utilisateur et peut, à elle seule, donner plusieurs indications sur le contenu voulu par l'utilisateur. Il s'agit d'un critère important pour le choix de l'information à délivrer à cet utilisateur. Par ailleurs, il s'agit d'un concept capturé ou calculé par différentes méthodes, telles que l'utilisation des capteurs de type GPS ou basées sur la puissance du signal des réseaux sans fil (cf. section 3.3.3). Ce n'est donc pas par hasard que la localisation est le principal élément observé par les systèmes sensibles au contexte (cf. section 3.4). Le concept de localisation physique de l'utilisateur est un élément incontournable pour la représentation de contexte au sein de tout système sensible au contexte.

De son côté, le point de vue ***outil*** nous renvoie à la question « *comment* se déroule l'utilisation du système ? ». En d'autres termes, il faut analyser quels sont les outils employés par l'utilisateur au moment de son interaction avec le système. Nous pouvons analyser cette question autant d'un point de vue matériel (le matériel nécessaire à l'interaction), que logique. D'un point de vue matériel émerge le concept de *dispositif* client, c'est-à-dire le dispositif à travers lequel l'utilisateur accède au système. Dans le cas des systèmes sur le Web, il peut s'agir tant de dispositifs plus traditionnels, tels qu'ordinateurs de bureau ou ordinateurs portables, que de dispositifs mobiles (ordinateurs de poche, téléphones cellulaires...). Les caractéristiques de ce type de dispositif imposent certaines contraintes à l'utilisateur (faible taille de l'écran, capacité de mémoire limité, etc.). Ces contraintes ont une influence sur l'utilisateur, qui attend du système une réponse adaptée aux capacités matérielles dont il dispose. Il nous paraît donc nécessaire de représenter ce concept de dispositif dans le modèle de contexte que nous proposons. D'ailleurs, d'autres représentations, comme celle basée sur le standard CC/PP proposée par Lemlouma [Lemlouma 2004a], utilisent également ce concept.

D'un point de vue d'outil logique, nous nous sommes intéressés au concept d'*application* en tant que logiciel avec lequel l'utilisateur interagit. Alarcón *et al.* [Alarcón 2002] ont déjà observé ce concept à travers notamment leur notion de « localisation électronique », qui représente les moyens (les logiciels) grâce auxquels l'utilisateur peut être contacté à un moment donné de la journée : SMS, messagerie électronique, etc. La notion d'application utilisée pour accéder au système est également mise en valeur par Lemlouma [Lemlouma 2004a], qui décrit les capacités du navigateur Web utilisé (identification du

navigateur, protocole d'accès, ressources supportées, etc.). Par ailleurs, dans le cadre de notre travail, nous avons pu observer un intérêt grandissant pour la conception de ces systèmes comme un ensemble de services Web (cf. section 2.3). Dans ce cas, il est possible d'imaginer l'adaptation de fonctionnalités du système par l'adaptation de la composition des services ou des services eux-mêmes (voir, par exemple, [Chaari 2005], [LopezVel 2005] et [Keidl 2004]). Pour que cette adaptation puisse se faire selon le contexte de l'utilisateur, il devient intéressant de représenter, dans la notion même de contexte, cette notion de service. Ainsi, nous avons identifié, dans le point de vue *outil*, l'élément *application* comme une manière de représenter, dans le modèle de contexte, les pièces logicielles, telles qu'un service Web, avec lesquelles l'utilisateur interagit directement.

Toujours dans le point de vue outil, il est possible également d'identifier le concept de connexion réseau comme étant le moyen par lequel l'utilisateur interagit avec le système. Cependant, l'acquisition dynamique des informations concernant l'état et la qualité du réseau s'avère souvent complexe, notamment pour les réseaux sans fil, dont le débit varie avec les nombreuses interférences (voir [Marchand 2004]). Le caractère dynamique des réseaux sans fil transforme l'acquisition de ces données en un processus coûteux, qui souvent n'offre pas au système le retour sur investissement attendu. Les données acquises varient avec une telle rapidité que le système risque d'exploiter des données qui ne correspondent plus à la réalité lors de leur utilisation. De ce fait, la plupart des systèmes sensibles au contexte n'intègrent pas le concept de réseau dans la notion de contexte. Lemlouma [Lemlouma 2004a] considèrent ce concept de manière statique, prenant en compte les valeurs attendues selon le type de réseau. Or, le débit réel d'un réseau sans fil n'atteint que très rarement son débit théorique (voir [Marchand 2004] et [Pujolle 2003]). Aussi, malgré l'évidente importance de ce concept, nous pensons judicieux que, dans un premier moment, ce concept ne soit pas considéré comme un élément de contexte dans notre modèle. Ceci n'empêche pas que cet élément soit introduit lors de la mise en œuvre ou d'une éventuelle extension du modèle (cf. section 5.1.3).

Le troisième point de vue est le ***temps***. À travers ce point de vue, nous capturons le moment où l'interaction entre l'utilisateur et le système a lieu (question « *quand* ? »). Dans le cas des collecticiels, cet aspect a une importance particulière, puisque cette interaction s'insère dans un cadre plus large : celui de la coopération entre l'utilisateur et ses collègues. Autant Leiva-Lobos et Covarrubias [Leiva-Lobos 2002] qu'Alarcón *et al.* [Alarcón 2002] mettent en relief cet aspect dans leurs modèles (cf. section 3.3), à travers la notion de calendrier ou d'agenda partagé par le groupe. Ces auteurs suggèrent que l'information relative au *calendrier* de l'équipe doit faire partie de la notion de contexte de l'utilisateur. C'est à travers ce concept de calendrier qu'on représente les échéances et le planning du groupe, d'où son importance pour le contexte. L'utilisateur d'un collecticiel n'est pas un individu isolé, il appartient à un ou plusieurs groupes, et le calendrier de ces groupes influence directement son travail, surtout en ce qui concerne les échéances. Par ailleurs, certains auteurs comme Borges *et al.* [Borges 2004], suggèrent non seulement la représentation du calendrier prévisionnel de l'équipe, mais aussi la comparaison entre ce qui était prévu par ce calendrier et ce qui a été réellement exécuté. L'objectif serait de mieux comprendre les raisons qui ont mené à la réalisation des activités d'une manière différente de celle qui était prévue. Il s'agirait d'un retour d'expérience enrichissant pour les membres du groupe dans le futur. Enfin, ce concept de calendrier, et la représentation des échéances qui en résulte, contribue à déterminer la pertinence ou non d'une information délivrée à l'utilisateur. Par exemple, la proximité d'une échéance peut faire en sorte que toute information liée à cette échéance devienne plus importante que d'habitude. Nous percevons ainsi que le concept de *calendrier* participe activement de la notion de contexte lorsqu'on considère l'utilisateur dans un environnent coopératif.

Le concept de calendrier vise notamment à représenter l'importance de la gestion du temps lors de l'utilisation d'un collecticiel. Néanmoins, il est intéressant d'observer que tout

élément du contexte, y compris le calendrier, comporte un aspect temporel, car les données associées à chacun de ces éléments évoluent dans le temps.

Par ailleurs, le concept de calendrier met en relief les aspects coopératifs du contexte d'utilisation dans un collecticiel. Ces aspects sont développés davantage par les deux derniers points de vue (*communauté* et *processus*), lesquels cherchent dans les concepts clés des collecticiels (voir section 2.1.1), les concepts qui peuvent participer à la notion de contexte.

Le point de vue ***communauté*** se penche sur les éléments liés à l'individu et aux groupes auxquels il participe. L'objectif ici est d'identifier qui participe, directement ou indirectement, au contexte d'utilisation (question « *qui* ? »). Le tout premier concept identifié dans ce point de vue est celui de l'*utilisateur* lui-même. Cependant, ce concept d'utilisateur comporte, pour nous, une particularité : il est également membre d'au moins un groupe de travail. Nous identifions ainsi le concept d'utilisateur en tant que *membre* d'un (ou plusieurs) groupe(s).

Si nous considérons l'utilisateur en tant membre d'un groupe, il faut considérer également le concept de *groupe*, comme nous l'avons défini dans la section 2.1.1. Un groupe pour nous correspond à un ensemble d'individus qui partagent un objectif commun et qui coopèrent pour l'atteindre. Puisque l'utilisateur participe à un de ces groupes, il est donc impliqué dans cette coopération, qui devient une partie de son activité. Le groupe devient alors un élément du contexte de cet utilisateur, puisque lui-même est engagé auprès du groupe et de ces objectifs.

Un troisième concept vient enrichir cette relation entre l'utilisateur et le(s) groupe(s) : celui des *rôles* tenus par l'utilisateur dans chaque groupe. Un rôle indique les responsabilités des membres vis-à-vis du groupe, ainsi que les droits de ces membres auprès du groupe (cf. section 2.1.1). Par ailleurs, un rôle est également un fort indicateur de la pertinence ou non d'une information [KirschPi 2001]. Il est donc normal que ce concept participe également de la notion de contexte de l'utilisateur, puisqu'il est inhérent à sa situation de membre d'un ou plusieurs groupes.

Le dernier point de vue, ***processus***, s'intéresse à la question « *quoi* ? » : « à quoi réfère l'interaction entre l'utilisateur et le système ? ». À travers cette question, nous nous intéressons aux aspects du travail en groupe liés au processus de coopération, lequel peut être implicite ou explicite. Dans un collecticiel, l'interaction de l'utilisateur s'insère dans le cadre d'un processus de coopération. Il est donc important d'identifier les éléments liés à ce processus qui peuvent caractériser cette interaction entre l'utilisateur membre d'un groupe et le système qui supporte le travail de ce groupe. Le premier concept que nous avons identifié dans ce point de vue est la notion de *processus* de collaboration, comme nous l'avons défini dans la section 2.1.1. Un processus décrit, d'une façon plus ou moins structurée, la manière dont la coopération entre les membres du groupe doit se dérouler. Il s'agit du fil conducteur de la coopération. D'autres travaux, tels qu'Alarcón *et al.* [Alarcón 2002] et Borges *et al.* [Borges 2004], mettent également en relief ce concept pour décrire le contexte de l'utilisateur. Il s'agit aussi également d'un concept clé pour les systèmes qui utilisent les *workflows*. Dans ce cas, le processus de coopération est explicite et peut alors être utilisé pour caractériser la situation de l'utilisateur par rapport à ce processus.

Un autre concept que nous avons identifié à travers ce point de vue est celui d'*activité* coopérative. Un processus de coopération est souvent composé par un ensemble d'activités qui doivent être réalisées par les membres du groupe afin que l'objectif de l'équipe soit atteint. Ce concept a été mis en évidence par plusieurs auteurs, dont Alarcón *et al.* [Alarcón 2002] et Borges *et al.* [Borges 2004]. Les premiers exploitent, à travers le *contexte de travail*, le concept d'activité comme l'élément de base qui compose le processus de coopération. Les

derniers considèrent que les activités réalisées par les membres du groupe dans le processus de coopération constituent, au même titre qu'un processus préalablement planifié, une information contextuelle importante pour les utilisateurs membres du même groupe. Il s'agit donc de représenter les tâches exécutées au sein de l'équipe, car, pour les participants de cette équipe, l'exécution d'une activité peut caractériser leur situation vis-à-vis du groupe. Le concept d'activité devient naturellement un élément de leur contexte d'utilisation.

Enfin, toujours dans le point de vue ***processus***, nous mettons en évidence un troisième concept, celui d'*objet partagé*. Conformément à la section 2.1.1, le concept d'objet partagé est un concept de base pour le travail en groupe. Il correspond à l'ensemble des données qui sont partagées par le groupe et sur lesquelles le groupe travaille. Dans plusieurs situations, toute la coopération se tient autour de ces objets. Par exemple, dans un collecticiel d'édition coopérative ou de répertoire partagé, la coopération s'organise autour du partage d'un document qui est édité en commun par le groupe, ou des données qui sont partagées entre les membres du groupe. Ce concept d'objet partagé devient ainsi un élément important pour décrire le contexte d'utilisation dans le cadre d'une telle coopération. Leiva-Lobos et Covarrubias [Leiva-Lobos 2002] soulignent également l'importance des artefacts autant physiques que numériques qui peuplent l'espace de travail.

En somme, chaque point de vue que nous avons décrits ci-dessus fait référence à certains éléments que nous considérons comme pertinents pour décrire le contexte d'utilisation d'un utilisateur nomade dans un collecticiel (voir Figure 15) : le point de vue ***espace*** nous fournit les concepts de *localisation* de l'utilisateur, tandis que le point de vue ***outil*** fournit les concepts de *dispositif* et d'*application* employés par cet utilisateur ; le troisième point de vue (***temps***) inclut le concept de *calendrier* partagé ; la notion de ***communauté*** nous donne les concepts de *groupe*, de *rôle* et d'utilisateur en tant que *membre* d'un (ou plusieurs) groupe(s) ; finalement, le point de vue ***processus*** concerne les concepts de *processus* mis en œuvre par un groupe, d'*activité* coopérative et d'*objet partagé* par ce même groupe. Chacun des éléments énumérés ci-dessus est représenté par une classe dans le modèle que nous proposons (voir Figure 15). Ces classes et les relations qu'elles entretiennent sont décrites dans la section 5.1.2.

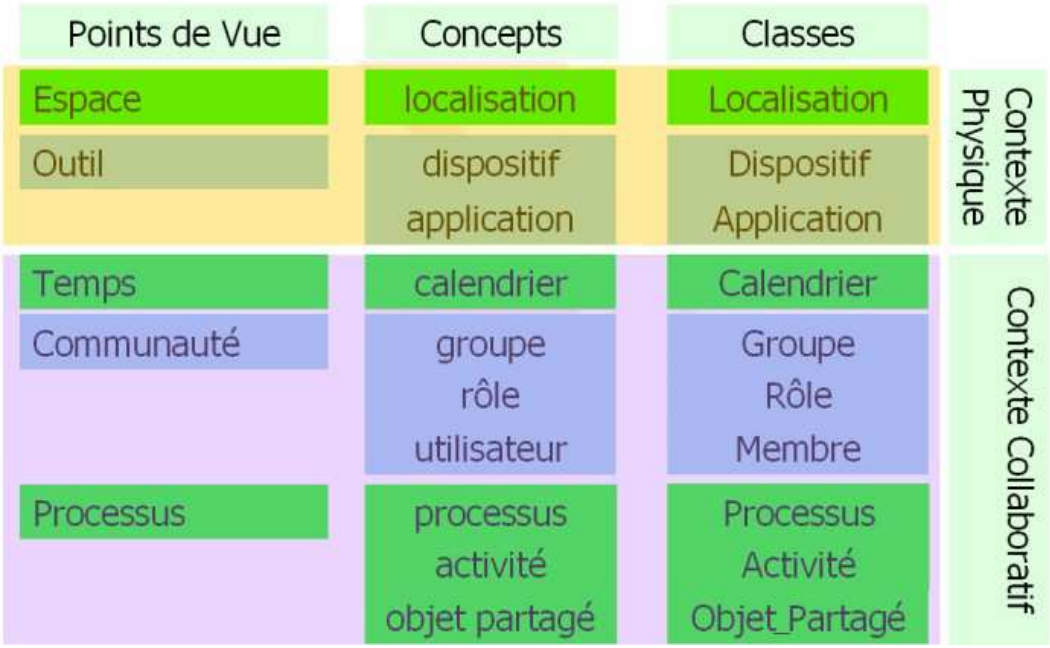


Figure 15. Points de vue et éléments identifiés, avec les classes correspondantes dans le modèle proposé.

Les éléments de contexte que nous avons identifiés peuvent être rassemblés, de manière générale, dans deux grands “sous-contextes” distincts (voir Figure 15). Le premier, appelé « *contexte physique* », est lié aux aspects physiques de l’utilisateur (sa localisation, son dispositif, sa localisation virtuelle, c’est-à-dire l’application qu’il utilise) ; le second, dit « *contexte collaboratif* », est lié au processus coopératif, ce qui inclut les notions de groupe, de rôle, d’activité, d’objet partagé, entre autres. Cette division, même si elle n'est pas toujours nette, souligne la présence des aspects coopératifs dans le modèle de contexte. La majorité des systèmes sensibles au contexte et des représentations de contexte (cf. sections 3.4 et 3.3.2) n’adoptent que la notion de contexte physique, en considérant l’utilisateur comme un individu isolé. Nous ne nous limitons pas à cette notion puisque nous traitons l’utilisateur comme membre d’un collectif. Ce collectif (le groupe) et tout ce qui touche le processus de coopération sont, selon nous, des éléments clés pour bien cerner les circonstances dans lesquelles l’utilisateur accède au système. La notion de contexte collaboratif que nous proposons va dans ce sens.

Par ailleurs, la notion de contexte collaboratif rappelle les informations utilisées par les mécanismes de conscience de groupe (voir, par exemple, [KirschPi 2003a]). La notion de conscience de groupe est naturellement liée à celle de contexte. La définition même de conscience de groupe mentionne déjà son rôle pour la formation d'un « contexte » dans lequel se positionnent les activités de chaque participant (cf. section 2.2). Il s'agit donc d'un ensemble de connaissances qui interviennent directement dans le contexte d'utilisation. Il est donc naturel, pour nous, que certains concepts présents dans les mécanismes de conscience de groupe soient aussi considérés dans la définition du contexte d'utilisation. Ceci ne signifie pas pour autant que dans la mise en œuvre de ce modèle de contexte au sein d'un collecticiel, les informations relatives à ces concepts sont dupliquées à la fois dans le mécanisme de conscience de groupe et dans le modèle de contexte. Dans ce dernier, nous ne trouvons pas forcément toutes les informations que nous trouvons dans le premier, mais seulement celles qui peuvent contribuer à l'identification de ces éléments de contexte. Le modèle de contexte n'est pas supposé gérer le fonctionnement du collecticiel dans sa totalité. Il capture, certes, certains aspects qui font également partie de ce fonctionnement, mais le modèle de contexte doit pouvoir évoluer dans le temps sans affecter de manière importante le fonctionnement du système qui l'utilise. Il s'agit ici de respecter le premier des quatre critères que nous avons souhaités pour la représentation de contexte, à savoir « pouvoir évoluer, soit par l'ajout, soit par la modification de nouveaux éléments de contexte » (cf. section 3.5).

Dans la prochaine section, nous représentons les éléments de contexte que nous avons identifiés ici dans un modèle à objets, à travers un ensemble de classes et d'associations.

5.1.2 Modèle à objets

Une fois que la notion de contexte est maîtrisée (cf. section 3.3.1) et les éléments de contexte nécessaires sont identifiés (cf. section 5.1.1), il faut les organiser dans un modèle. L'objectif est donc de formaliser la notion de contexte, avec les aspects physiques et collaboratifs. Nous avons choisi pour cela une représentation par objets de type classe/association, illustrée à l'aide du standard UML. L'utilisation d'un formalisme par objets contribue, selon nous, à une meilleure compréhension du modèle, puisque ce formalisme est largement connu et maîtrisé par les concepteurs de systèmes. De plus, une représentation par objets de type classe/association permet l'évolution du modèle à travers divers mécanismes, tels que l'héritage. Ainsi, par le choix de ce type de formalisme, nous souhaitons non seulement faciliter la compréhension du modèle, mais également rendre plus facile la mise en place des critères que nous avons fixés pour la représentation de contexte. Notre modèle de

contexte organise les éléments de contexte, ainsi que les relations que ces éléments peuvent entretenir, à travers un ensemble de diagrammes de classes UML

Le concept de contexte est décrit, à un moment donné, par un certain nombre de ces éléments, lesquels caractérisent différents aspects de la situation dans laquelle se trouve une entité. Cette entité, toujours selon la définition que nous adoptons, peut être une personne (l'utilisateur), un objet, un endroit, etc. (en réalité, toute entité considérée comme pertinente pour l'interaction entre l'utilisateur et l'application). Ainsi, dans notre modèle, le concept de contexte d'utilisation est représenté par la classe `Description_de_Contexte`, illustré par le diagramme de classes UML de la Figure 16. Cette classe est la composition des classes représentant les éléments de contexte identifiés précédemment (voir Figure 15). Ces classes sont les sous-classes d'une classe abstraite `Élément_de_Contexte`. On notera, dans la Figure 16, la présence de la relation `décrit_contexte_de` entre la classe `Description_de_Contexte` et la classe `Élément_de_Contexte`. Cette relation représente le fait qu'une description de contexte est toujours relative à une entité donnée laquelle, pour nous, est également un élément de contexte : contexte d'un utilisateur, contexte d'un dispositif, contexte dans lequel une application peut être exécutée…

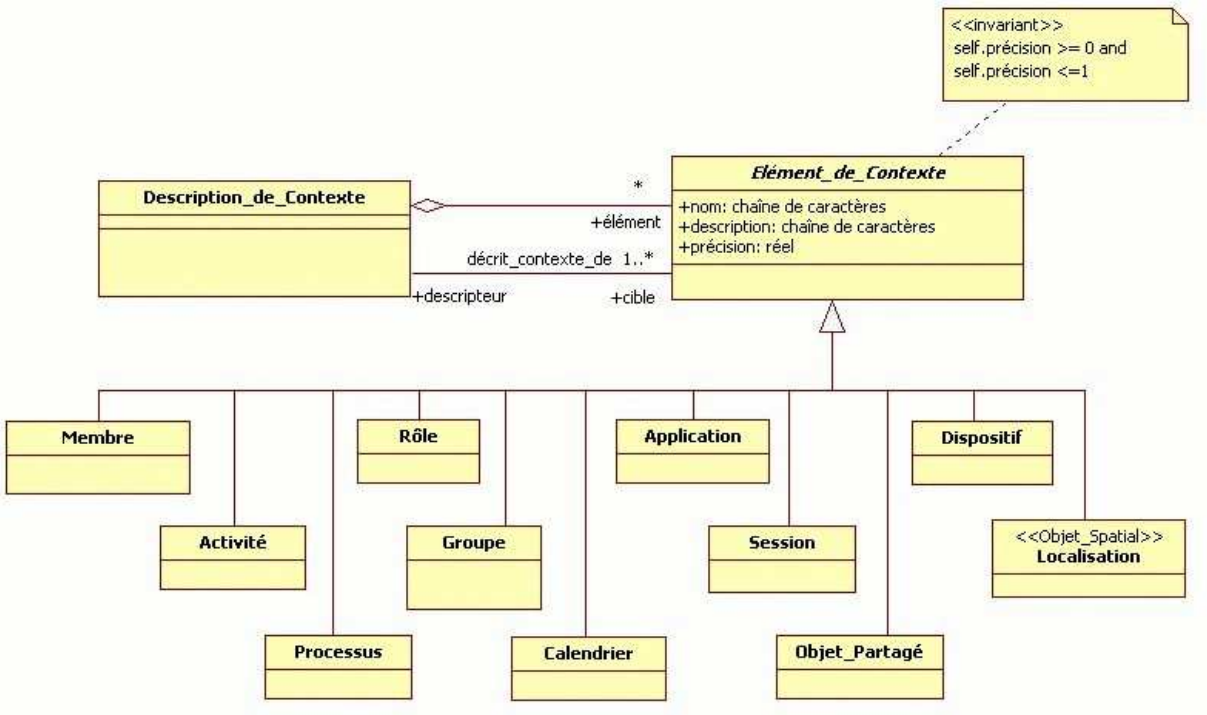


Figure 16. La description du contexte vue comme une composition d'éléments de contexte.[51]

Par ailleurs, on observe dans la Figure 16 que nous proposons, dans la super-classe `Élément_de_Contexte`, certains attributs de base, qui sont hérités par toutes les sous-classes. Parmi ces attributs, on trouve un indicateur de précision, sous la forme d'un nombre réel entre `0` et `1` (attribut `précision`). L'objectif de cet indicateur est de permettre à la couche d'acquisition de contexte de donner une indication de la précision avec laquelle les données relatives à l'élément ont été capturées. Il s'agit, en quelque sorte, d'un « méta-donnée » sur les informations maintenues par l'élément. Cet indicateur pourra être employé par les composants du système pour la prise de décision. Le contexte est souvent utilisé par le système pour prendre des décisions à la place de l'utilisateur (cf. chapitre 3). Ainsi, à travers l'indicateur de

[51]Tous les diagrammes UML présentés dans ce document suivent la spécification UML 2.0 et ont été réalisés à l'aide des outils *StarUML* (http://www.staruml.com/) et *Objecteering UML* (http://www.objecteering.com/).

précision, des seuils pour l'utilisation effective de l'information contenue dans un objet `Élément_de_Contexte` pourront être définis. Par exemple, lorsque cet indicateur signale une précision médiocre, c'est-à-dire la valeur de l'attribut `précision` est proche de `0` (`précision ≈ 0`), cet élément pourrait être ignoré, puisque les informations qu'il contient n'ont pas un niveau de précision minimal.

Hormis cet indicateur de précision, les autres attributs de la classe abstraite `Élément_de_Contexte` (attributs `nom` et `description`) décrivent le type d'information qu'elle contient. Ils constituent, en quelque sorte, des méta-données qui décrivent les informations contenues dans un objet de cette classe. Cet ensemble de trois attributs (deux descriptifs et un indicateur de précision) est, selon nous, le minimum nécessaire pour décrire un élément de contexte. Ensuite, pour chaque élément que nous avons identifié dans la section 5.1.1, une classe spécifique est proposée, et, pour chaque classe, un ensemble d'attributs particuliers est suggérés à titre indicatif : d'autres sont envisageables pour chaque classe et association, en fonction du système qui exploite le modèle.

La Figure 16 comporte un élément en plus par rapport aux éléments présents dans la Figure 15. La classe `Session` représente la présence de l'utilisateur dans le système. L'analyse des éléments qui composent les points de vue *espace* et *outil* nous a montré que les relations entre ces éléments passent par la notion de session. On n'observe l'interaction entre un utilisateur et le système que lorsque cet utilisateur est actif dans le système, c'est-à-dire, lorsqu'il est connecté. Il devient donc nécessaire de représenter le fait que l'utilisateur soit connecté, puisque c'est à partir de ce moment que la notion de contexte devient observable par un système sensible au contexte. Par ailleurs, nous observons l'utilisateur en tant que membre d'un groupe. Il est donc représenté dans le modèle par une classe `Membre`. Ainsi, nous considérons qu'un utilisateur (un membre d'au moins un groupe) se trouve, pendant un intervalle de temps donné, dans un espace présentant un caractère à la fois réel et applicatif (relation `est_localisé_dans` dans la Figure 17). Cette idée correspond à la notion de session active. Une fois connecté au système, ce membre se trouve donc dans un `espace_physique`, composé par sa localisation dans le monde réel (classe `Localisation`), dans un `espace_d'exécution`, qui indique le dispositif (classe `Dispositif`) qu'il emploie, et dans un `espace_applicatif`, comportant l'application (classe `Application`) qu'il utilise. La Figure 17 présente les classes `Membre`, `Session`, `Localisation`, `Dispositif` et `Application`, avec les relations qu'elles entretiennent.

Pour un grand nombre de systèmes sensibles au contexte, le concept de localisation est vu comme une région précise dans l'espace dans laquelle se trouve une entité (utilisateur ou objet) observable par le système. Les données brutes concernant la localisation sont souvent interprétées, donnant naissance à ces régions symboliques. Celles-ci sont associées à une connaissance de plus haut niveau, qui garantit une signification plus compréhensible pour l'utilisateur (par exemple, le numéro d'une chambre d'hôpital dans [Muñoz 2003]). Ce sont ces régions symboliques qui sont effectivement comprises et utilisées par le système. D'ailleurs, cette perception de la localisation comme une région s'adapte particulièrement aux besoins d'une utilisation confinée (cf. section 3.2), laquelle doit se dérouler dans un ensemble de régions prédéterminées. Ceci est le cas, par exemple, de l'application *NITA* [Rubinsztejn 2004], dont la représentation de la localisation des utilisateurs distingue des régions symboliques prédéfinies (voir section 3.4).

Les systèmes sensibles au contexte ne sont pas les seuls à s'appuyer sur le concept de localisation. La représentation de la localisation physique d'une entité (une personne, un objet, un événement...) est le sujet de différents travaux (par exemple, [Moisuc 2004] [Moisuc 2005]), et des normes *OpenGIS* définies par le consortium OGC[52]. Ces normes s'intéressent

[52] *Open Geospatial Consortium (OGC)* - http://www.opengeospatial.org/.

particulièrement à la représentation d'objets spatiaux et aux opérateurs capables de les comparer. Elles s'appliquent notamment aux *systèmes d'information géographiques* (SIG)[53] et à des « services sensibles à la localisation »[54], qui sont des services capables d'adapter leur comportement à la localisation. Ce type de système utilise de manière intensive les informations à caractère spatial comme la localisation d'un objet.

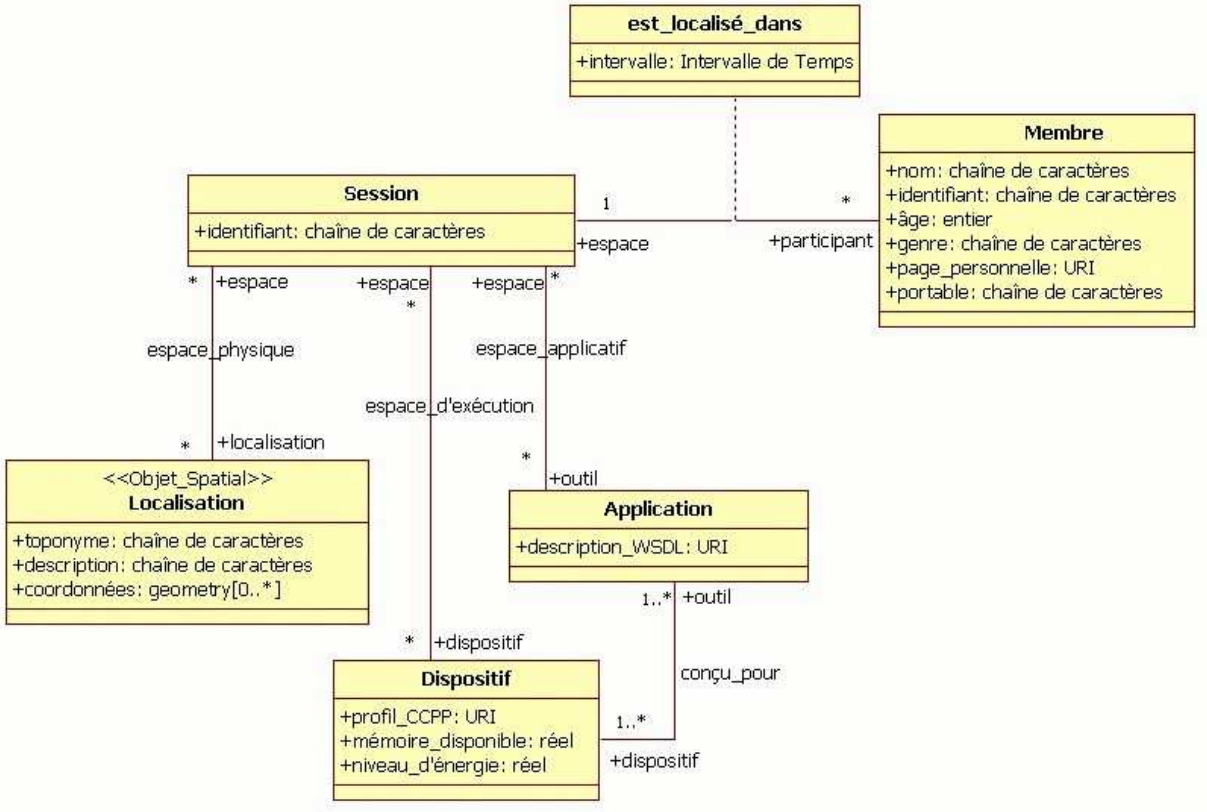


Figure 17. Classes et relations représentant un utilisateur qui se localise dans un espace réel et virtuel.

Nous basons notre représentation du concept de localisation sur ces travaux de représentation d'objets spatiaux, tout particulièrement, sur les normes *OpenGIS* et le modèle de représentation spatio-temporelle proposé par Moisuc *et al.* [Moisuc 2004] [Moisuc 2005]. Ainsi, chaque objet de la classe `Localisation` (voir Figure 17) est décrit par un ensemble de coordonnées (attribut `coordonnées`) qui sont représentées par un ensemble de points géométriques (type « *`Geometry`* »), selon la norme *OpenGIS*. Cet ensemble permet la description tant d'un point précis dans l'espace que d'une région, à travers les notions de point (« *Point* ») et de polygone (« *Polygon* ») qui dérivent du type « *Geometry* », selon les normes *OpenGIS* [Ryden 2005]. Par ailleurs, la classe `Localisation` est dotée d'un stéréotype `Objet_Spatial` (« *`SpatialObject`* »), qui la positionne par rapport au modèle spatio-temporel de Moisuc *et al.* [Moisuc 2004] [Moisuc 2005]. Un objet spatial est, pour ces auteurs, un objet doté d'au moins un attribut spatial qui donne sa représentation ou sa position géographique. Dans notre cas, il s'agit d'une position géographique qui est référencée à travers l'attribut `coordonnées`. De plus, nous associons à chaque localisation un nom (attribut `toponyme`), qui agit comme un identifiant représentatif de sa signification (par exemple, le numéro d'une salle), et une description (par exemple, la description d'où se trouve la salle). Ces attributs (`toponyme` et `description`) permettent l'association entre un ensemble de coordonnées

[53]En anglais, « *Geographical Information System* » (GIS).
[54]En anglais, « *location based services* ».

(obtenues, par exemple, à travers un ensemble de points données par un GPS) et un nom qui a une signification pour les utilisateurs (par exemple, « IMAG – bât. D », « gare routière »...).

Dans la Figure 17, on observe également que les classes `Dispositif` et `Application` contiennent, elles aussi, des attributs particuliers. Ces attributs concernent l'utilisation des standards respectifs : *CC/PP*[55] et *WSDL*[56]. Le standard CC/PP est devenu incontournable pour la description des capacités d'un dispositif (cf. section 3.3.2). L'adoption de ce standard est effective chez plusieurs fabricants, pour la publication des caractéristiques de leurs produits (c'est, par exemple, le cas du téléphone cellulaire *P900* du fabricant *Sony Ericson*, dont la spécification est disponible sur Internet[57]). Par ailleurs, l'apparition de répertoires contenant ces spécifications est désormais une tendance forte (voir, par exemple, le répertoire proposé par le site *W3Developement*[58]). Il nous paraît donc opportun d'utiliser ces profils CC/PP pour décrire les capacités d'un dispositif. Ainsi, ces descriptions ont été introduites dans la classe `Dispositif` à travers l'attribut `profil_CCPP` (voir Figure 17), lequel se présente comme une référence (une URI) à une description CC/PP (fournie soit par le fabricant du dispositif, soit par un répertoire de spécifications).

Pour le concept d'application (cf. section 5.1.1), le constat est le même. Les collecticiels sur le Web utilisent de plus en plus les services Web, lesquels sont décrits par le biais du standard *WSDL* (cf. section 2.3). À travers ce standard, il est possible de décrire les capacités, ainsi que les entrées et les sorties d'un service. Par ailleurs, certains travaux proposent des extensions pour ce langage, ce qui lui assure un pouvoir d'expression plus important encore. Par exemple, Lopez-Velasco [LopezVel 2005] propose une extension de WSDL capable de décrire les capacités d'adaptation d'un service. Ainsi, il semble judicieux d'utiliser le standard WSDL pour décrire l'interface et les capacités de chaque application observée dans le cadre d'une représentation du contexte d'utilisation. Nous utilisons pour cela l'attribut `description_WSDL` de la classe `Application`, lequel se présente également comme un lien vers la description WSDL (voir Figure 17).

Par ailleurs, les classes `Application` et `Dispositif` sont également liées par l'association `conçu_pour` (cf. Figure 17), qui représente le fait qu'une application est conçue afin d'être compatible avec un certain nombre de dispositifs. Cette compatibilité est particulièrement intéressante lorsqu'elle concerne l'interface. On sait que les dispositifs mobiles se caractérisent par leurs contraintes physiques qui impactent notamment l'interface avec l'utilisateur. Celui-ci attend que les applications soient adaptées à ces contraintes, d'où l'intérêt de rendre explicite cette relation entre les applications et les dispositifs.

Hormis la description CC/PP, la classe `Dispositif` contient également d'autres attributs qui donnent l'état courant du dispositif en termes de mémoire disponible (attribut `mémoire_disponible`) et d'énergie (attribut `niveau_d'énergie`). Le standard CC/PP fournit majoritairement les informations statiques sur le dispositif : taille de l'écran, nombre de bits par pixel, etc. Ces informations ne changent pas au cours de l'utilisation. En revanche, d'autres caractéristiques du dispositif telles que la mémoire disponible pour l'exécution d'une application, ou encore le niveau d'énergie disponible (particulièrement important pour les dispositifs mobiles qui disposent d'une batterie avec une capacité limitée) varient en fonction de l'utilisation. Puisqu'il est possible de capturer ces deux informations lors de l'exécution, nous avons opté pour les représenter dans la classe `Dispositif` à travers les attributs `mémoire_disponible` et `niveau_d'énergie`. Conjointement à la description CC/PP du dispositif, ces informations sont les plus pertinentes à représenter dans le cadre d'une utilisation nomade.

---

[55] « *Composite Capability/Preference Profiles* » - http://www.w3.org/Mobile/CCPP/.
[56] « *Web Services Description Language* » - http://www.w3.org/TR/wsdl.
[57] http://wap.sonyericsson.com/UAprof/P900R101.xml
[58] http://w3development.de/rdf/uaprof_repository/

Outre les classes et les relations décrivant l'espace réel et virtuel dans lequel se trouve l'utilisateur, on observe également dans la Figure 17 que l'utilisateur, représenté par la classe `Membre`, est caractérisé dans le modèle par plusieurs attributs. Ces attributs décrivent un minimum d'informations à son sujet, telles que son `nom`, sa `page_personnelle` et le numéro de son téléphone cellulaire (attribut `portable`). Ces informations ne correspondent probablement pas à la totalité des informations conservées par le système à propos de l'utilisateur, mais seulement à un sous-ensemble capable de caractériser cet utilisateur. D'ailleurs, l'aspect le plus important concernant la représentation de l'utilisateur ne réside pas dans les informations représentées par les attributs, mais sur les relations que la classe `Membre` entretient avec les autres classes du modèle. Les figures 17 et 18 présentent ces relations.

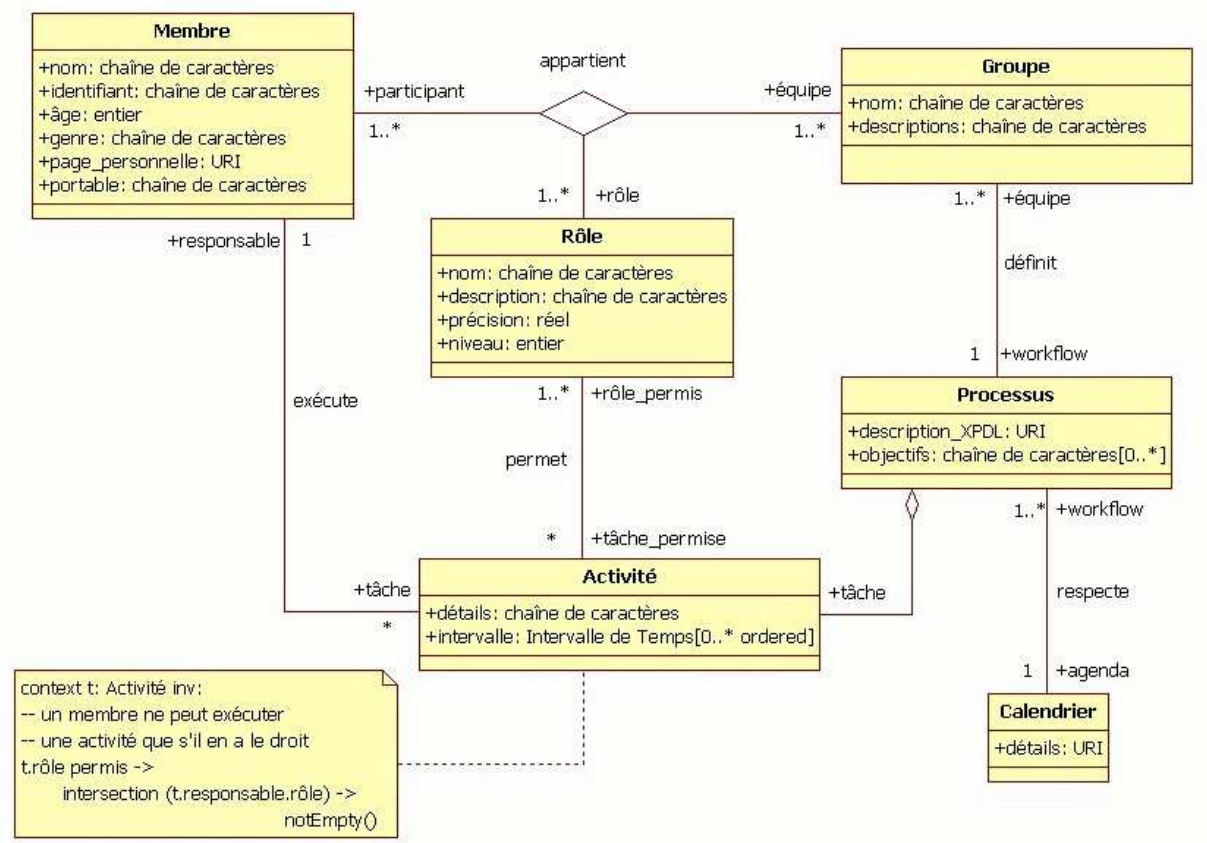


Figure 18. Les classes et les relations concernant l'utilisateur et le groupe.

Nous pouvons observer, dans la Figure 18, qu'un utilisateur (classe `Membre`) appartient à un ou plusieurs groupes (qui sont représentés à leur tour par la classe `Groupe`), à travers les rôles qu'il tient (représentés eux aussi par une classe nommée `Rôle`). À travers les classes `Membre`, `Groupe` et `Rôle`, nous présentons ces trois concepts de base d'un collecticiel (cf. section 2.1.1). Un groupe est un ensemble d'individus qui coopèrent pour atteindre un objectif commun. Un rôle est la description des droits et des responsabilités attribués à un membre du groupe. Un groupe admet plusieurs rôles. Par exemple, un comité de programme est un groupe dont les membres peuvent tenir le rôle de réviseur et/ou de président du comité. La personne qui tient le rôle de président du comité tient également souvent le rôle de réviseur. Une même personne peut encore jouer le rôle de réviseur dans un comité de programme donné, et le rôle de président dans un deuxième comité. Ainsi, cette relation est vue dans le modèle (voir Figure 18) à travers l'association ternaire `appartient` qui met en relation ces trois classes (`Membre`, `Groupe` et `Rôle`). L'utilisation d'une association ternaire dénote l'importance de chacune de ces classes pour l'utilisateur et pour le contexte d'utilisation dans un collecticiel.

Chacune de ces classes représente un concept indépendant dans un tel système. Aucune d'entre elles ne peut se réduire à une classe-association.

La Figure 18 présente également la relation entre le concept de groupe et celui de processus, représenté par la classe `Processus`. Ce dernier concept décrit la manière dont la coopération doit se dérouler pour atteindre ses objectifs (cf. section 5.1.1), en respectant un calendrier donné (association `respect` avec la classe `Calendrier`). Ces objectifs sont donc directement liés au processus mis en place par le groupe, et à ce titre, ils sont représentés dans la classe `Processus` par l'attribut `objectifs`. Ainsi, nous considérons qu'un groupe peut définir un processus, ou encore réutiliser un processus qui a été défini auparavant (voir association `définit` dans la Figure 18). La réutilisation d'un processus est une pratique courante dans les collecticiels de gestion de *workflow*, puisque, dans ces systèmes, chaque processus en exécution est vu comme une nouvelle instance d'un *workflow* donné. Cette réutilisation est donc possible grâce à la formalisation du processus à travers une définition explicite. Cette description du processus peut se faire à l'aide du standard XPDL[59], lequel a été proposé par le WfMC[60]. Le XPDL est un langage basé sur le standard XML permettant la description d'un processus structuré. Il devient donc intéressant de permettre que tout objet de la classe `Processus` puisse disposer d'une telle description. Ceci est fait, au même titre que pour les classes `Dispositif` et `Application`, à travers un attribut qui se présente sous la forme d'une URI vers cette description (attribut `description_XPDL`).

D'autre part, d'après la WfMC [WfMC 2005], un processus est composé par un ensemble d'activités. Nous partageons cette perception représentée dans le modèle par l'agrégation entre les classes `Processus` et `Activité`, visible dans la Figure 18. Chaque activité fait ainsi partie d'un processus donné, et contient une description plus détaillée de son contenu (attribut `détails`), et un ensemble d'intervalles de temps (attribut `intervalle`) qui dénotent les périodes dans lesquelles cette activité a été exécutée. De plus, nous percevons, à travers les associations `permet` et `exécute` dans la Figure 18, qu'un rôle permet la réalisation d'une ou plusieurs activités, qui sont effectivement exécutées par un membre du groupe, responsable de son exécution. Néanmoins, il est important d'observer que, comme l'exprime la contrainte associée à la classe `Activité`[61], un membre ne peut exécuter une activité que s'il joue un rôle qui lui donne le droit de l'exécuter.

Une activité, comme un processus, peut être, à son tour, composée par d'autres activités (cf. Figure 19). Cette relation de composition est également mise en évidence par Carroll *et al.* [Carroll 2003]. Ces auteurs considèrent que le contexte qui entoure une activité collaborative correspond à la manière dont cette activité est décomposée en d'autres, ainsi qu'à son exécution par un collègue (ce que nous avons déjà mis en évidence ci-dessus à travers les associations `permet` et `exécute`). Par ailleurs, cette composition d'activités admet la création d'un tissu d'activités et de sous-activités qui permet, indirectement, l'expression de processus très complexes.

De plus, comme le montre la Figure 19, chaque activité peut manipuler, à travers une ou plusieurs applications (classe `Application`), un ensemble d'objets partagés, qui sont représentés par la classe `Objet_Partagé` (cf. section 2.1.1). Cette relation entre les activités, les objets partagés et les applications se présente comme une association ternaire entre les classes respectives (association `manipule` dans la Figure 19). Cette association possède des attributs décrivant notamment l'historique des manipulations (attribut `historique`) et le degré d'importance de cette opération (attribut `criticité`). Ce dernier attribut sert à indiquer si la manipulation d'un objet partagé dans une activité est plus ou moins critique par rapport au

[59]En anglais, « *XML Process Definition Language* » - http://www.wfmc.org/standards/XPDL.htm.
[60]« *Workflow Management Coalition* » - *http://www.wfmc.org/*.
[61]Les contraintes sont décrites en OCL, suivant la spécification UML 2.0.

travail en cours. Dans certains processus de coopération (par exemple, la conception de pièces d'un moteur), le fait de manipuler un objet partagé dans une activité donnée est plus critique que la manipulation du même objet dans une autre activité, en raison de plusieurs facteurs tels que l'état de l'objet (si son contenu est stable ou doit encore évoluer), ou encore l'importance de l'activité pour le processus (une prise de décision au début du projet, par exemple). Ce n'est pas l'objet partagé en soi ou l'activité de manière isolée qui sont critiques, mais surtout la manipulation de cet objet dans cette activité qui sera critique, puisqu'elle aura des conséquences importantes pour la suite du travail. C'est donc l'association de ces facteurs qui a une signification pour le déroulement du travail du groupe.

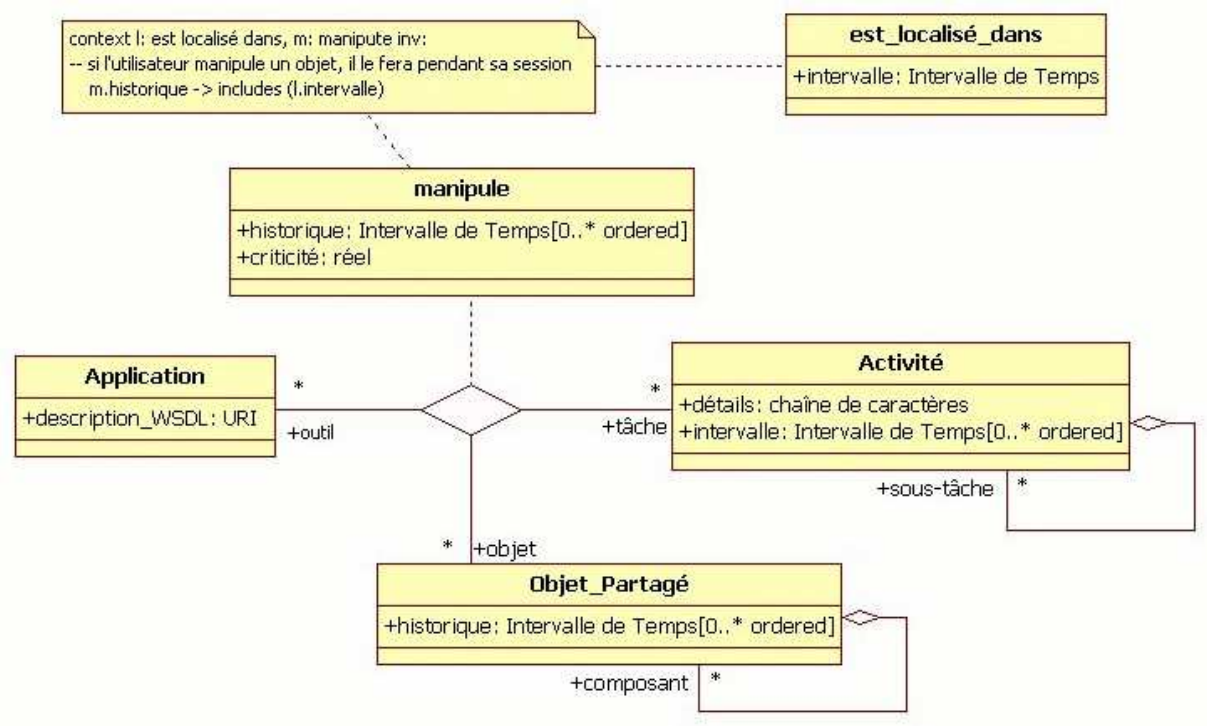


Figure 19. Relations entre la classe Objet_Partagé et les classes Activité et Application.

Par ailleurs, on observe également dans la Figure 19 que l'historique de la manipulation est sujet à une contrainte par rapport à la session de l'utilisateur. Pour qu'un utilisateur puisse manipuler un objet partagé dans le cadre d'une activité, il faut qu'il soit connecté au système et qu'il utilise une application qui lui permette cette manipulation. Par conséquent, si l'utilisateur effectue réellement cette manipulation (ce qui correspond à un objet de la classe-association `manipule`), l'intervalle de temps dans lequel il est connecté au système fait partie de l'`historique` de manipulation, d'où la contrainte liant les classes-associations `est_localisé_dans` et `manipule`. Cependant, il ne faut pas oublier que probablement plusieurs membres du groupe seront capables d'effectuer la même manipulation. L'historique ne concerne pas donc qu'un seul utilisateur, mais potentiellement tous les membres du groupe qui ont le droit d'exécuter l'activité.

Enfin, il est important d'observer que cette structure « `Processus - Activité - Objet_Partagé` » présentée dans le modèle de contexte permet la représentation de la structure adoptée à la fois par des équipes très hiérarchisées, qui utilisent souvent des collecticiels de gestion de *workflow* (cf. section 2.1.2.1), et par celles qui adoptent un modèle de travail plus souple et qui coopèrent surtout à travers la manipulation des objets partagés. Le modèle peut donc s'appliquer aux collecticiels adoptant ces deux philosophies de travail.

Nous avons discuté dans la section 5.1.1 que les éléments de contexte que nous avons identifiés peuvent être divisés en deux « sous-contexte ». Il en va de même pour les classes qui représentent ces éléments. Ces classes se divisent dans les mêmes « sous-contextes », « *contexte collaboratif* » (voir Figure 20) et « *contexte physique* » (voir Figure 21). Le premier considère les éléments de contexte liés aux aspects coopératifs : notions de processus, d'activité, de groupe, etc. Le deuxième considère surtout les aspects physiques, comme la localisation. Celui-ci correspond à la notion de contexte traditionnellement adoptée par les systèmes sensibles au contexte. Comme le montrent les figures 20 et 21, il n'existe pas de séparation nette entre le contexte collaboratif et le contexte physique, les deux partageant certains éléments et associations (par exemple, la classe `Application`). Ceci souligne que cette séparation entre ces deux « sous-contextes » est surtout conceptuelle, et sert essentiellement à indiquer la nature des informations capturées (coopératives ou physiques).

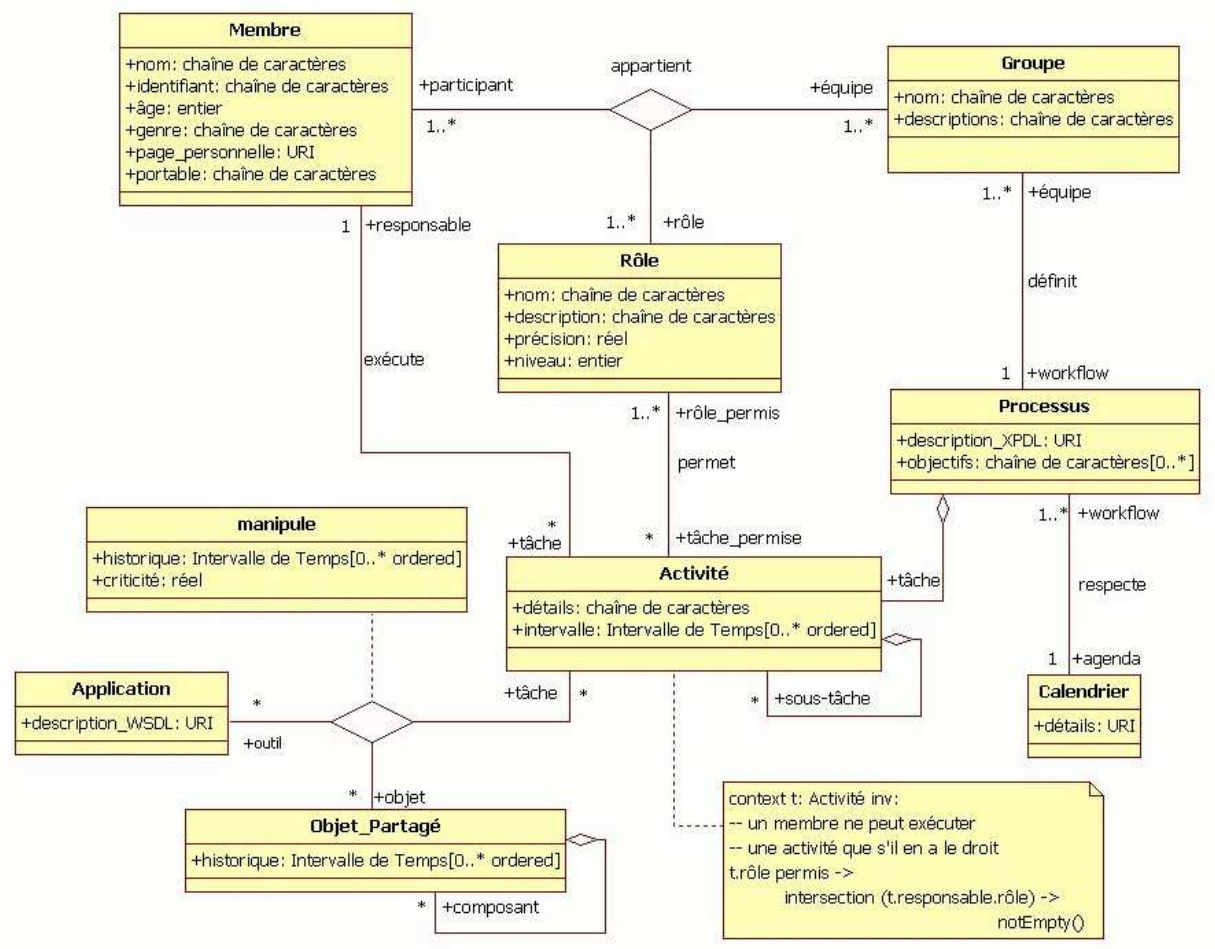


Figure 20. Les éléments qui composent le contexte collaboratif.

La Figure 23 donne une vue d'ensemble de toutes les classes qui composent le modèle et les associations qui les lient, en indiquant également quelques attributs possibles. Chacun des éléments énumérés dans la Figure 15 est représenté dans le modèle que nous proposons par une classe. Les relations entre ces classes témoignent des liens existant entre les concepts. En d'autres termes, chaque élément de contexte dans ce modèle n'est pas une entité isolée mais appartient, au contraire, à une représentation complexe de la situation réelle dans laquelle évolue l'utilisateur. Pour nous, les associations entre les classes apportent au modèle un complément vis-à-vis des éléments, permettant l'expression des situations plus complexes.

Nous avons donc dans ce modèle l'expression de relations « horizontales » à côté de la relation « verticale » de spécialisation/généralisation.

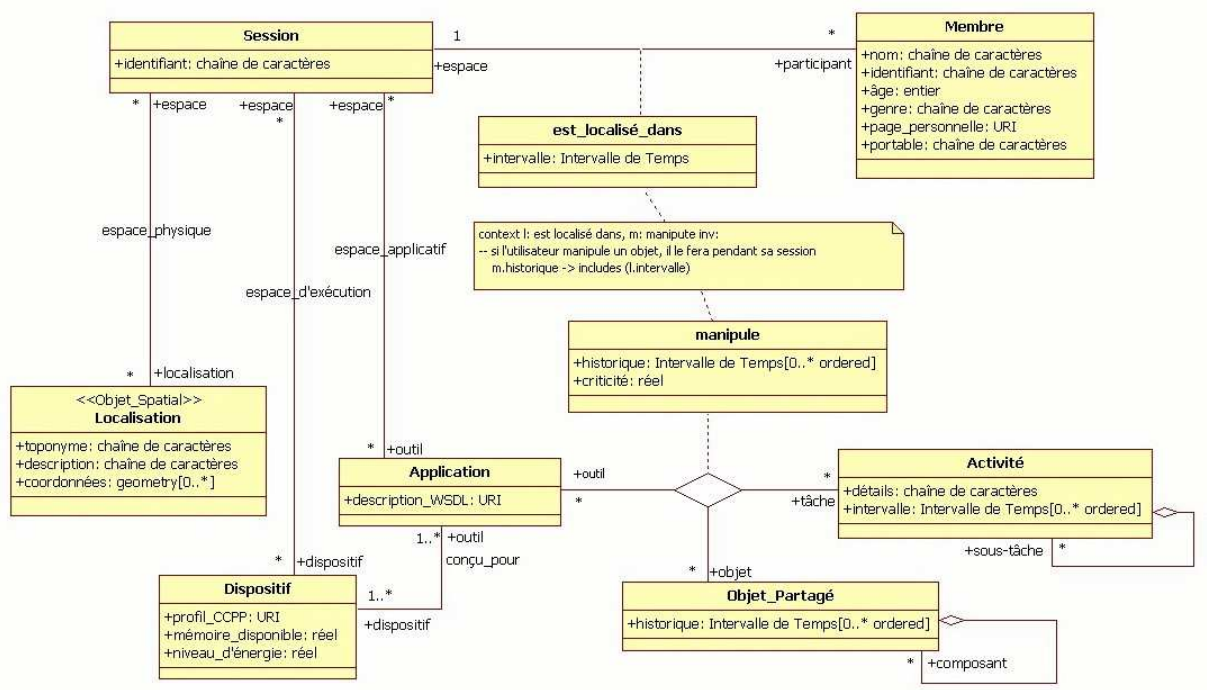


Figure 21. Le contexte physique, ses éléments et les relations qu'ils entretiennent.

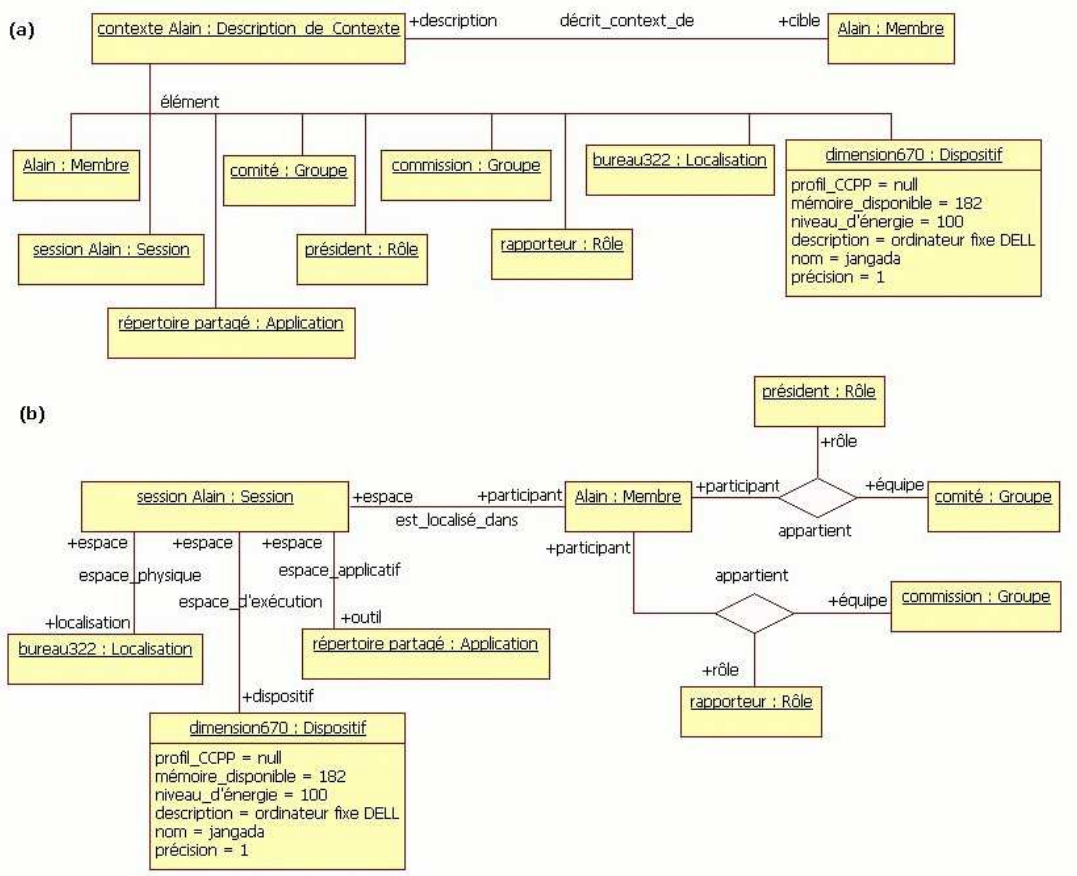


Figure 22. Exemple de description de contexte. En (a) les éléments qui composent cette description et, en (b) les relations entre ces éléments.

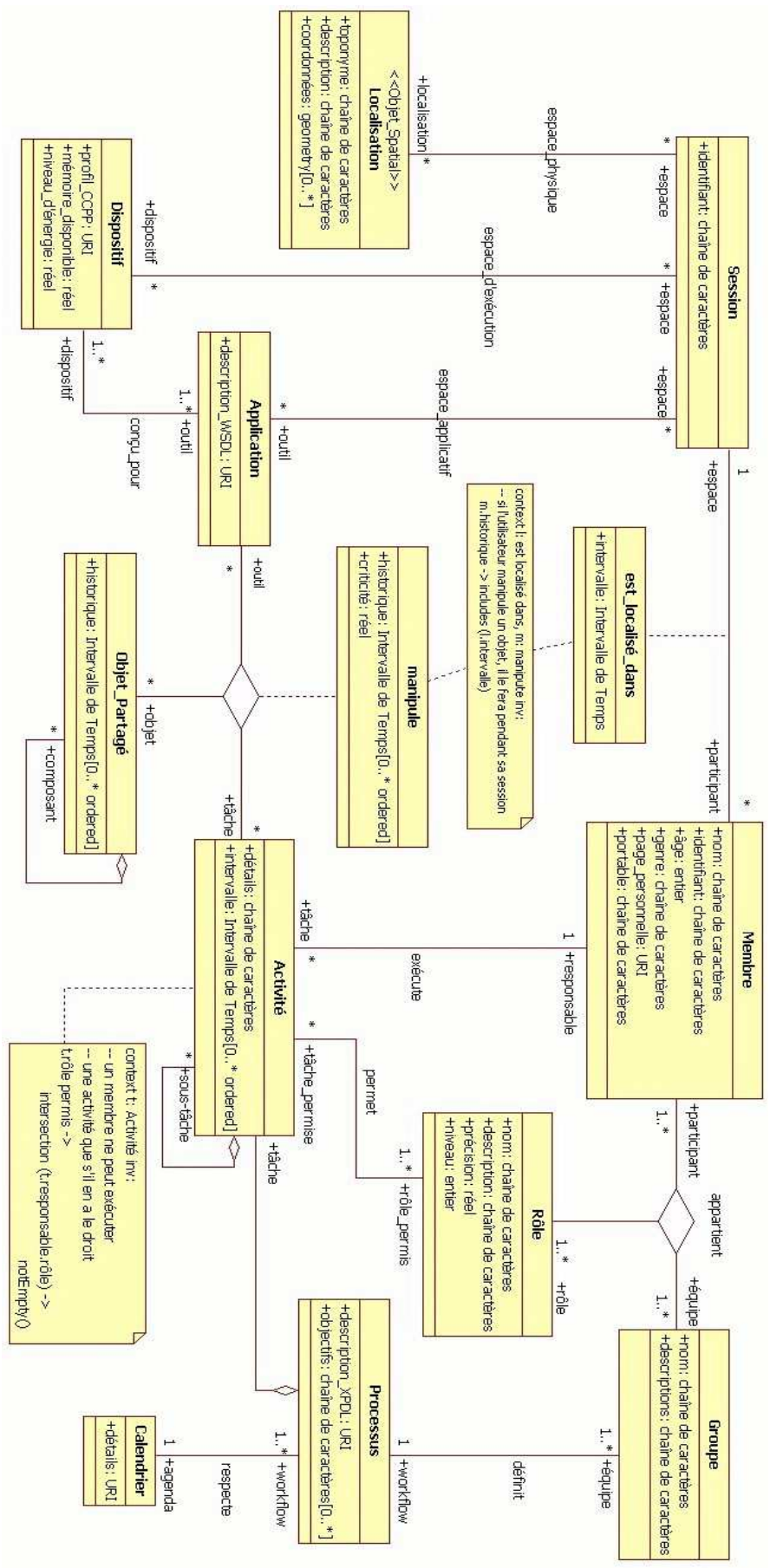


Figure 23. Les classes représentant des concepts de contexte et leurs relations.

La Figure 16 et la Figure 23 résument le modèle de contexte que nous proposons. Les instances des classes et des associations représentées dans ces deux figures forment un tissu d'éléments capables de décrire le contexte d'utilisation dans un collecticiel. Ce contexte d'utilisation est accessible à travers les objets de la classe `Description_de_Contexte`. C'est à travers les objets de cette classe qu'un collecticiel manipulera la notion de contexte représentée par les objets `Élément_de_Contexte` liés à une description de contexte.

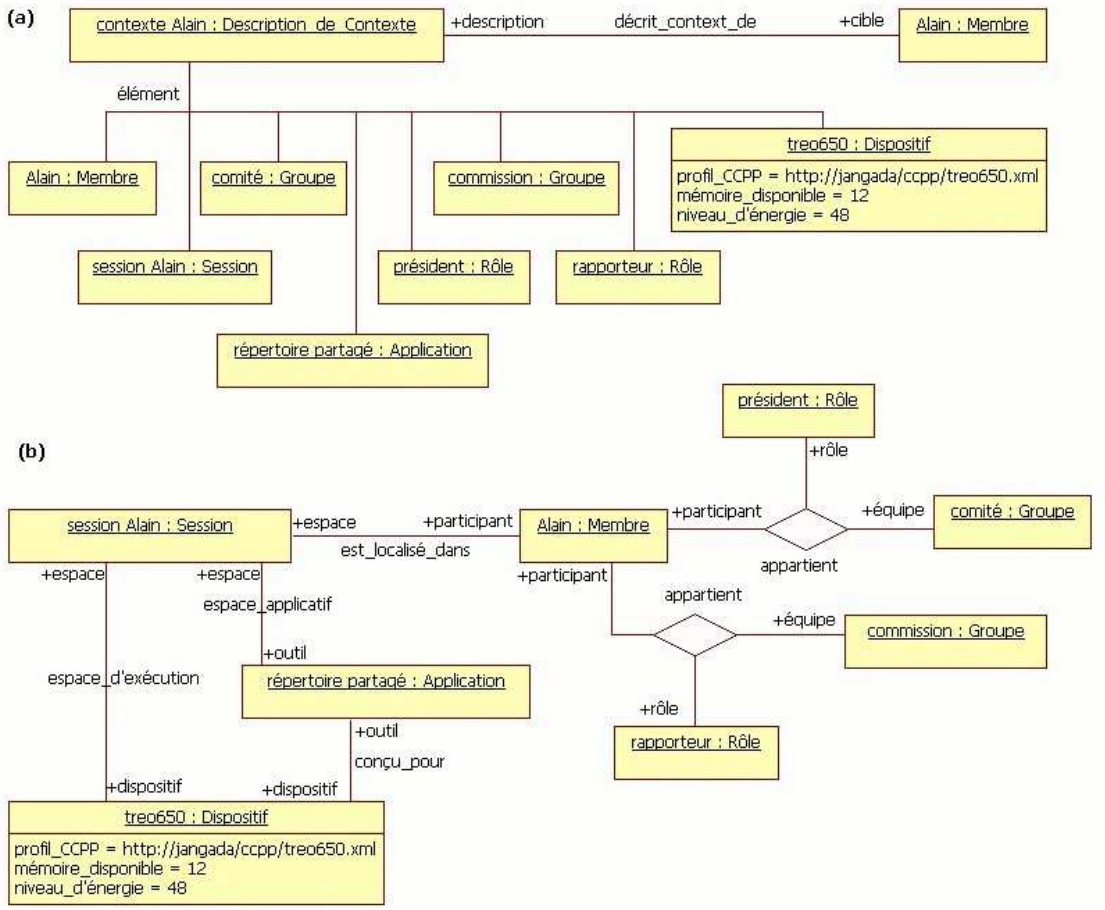


Figure 24. Contexte de l'utilisateur Alain suite à un déplacement. En (a) les éléments qui composent la description de contexte, et en (b) leurs relations.

La Figure 22 présente un exemple d'objet `Description_de_Contexte`[62]. Dans cet exemple, un collecticiel pour le partage de documents, semblable à *LibreSource* [Forest 2005a] ou à *BSCW* [BSCW 2005] (voir section 2.3), fournit à ses utilisateurs un service de répertoire partagé, parmi d'autres services. Un utilisateur spécifique (appelé 'Alain') participe, à travers ce collecticiel, à deux groupes distincts : le comité de programme d'une conférence, en tant que président du comité, et une commission d'évaluation, en tant que rapporteur. À un moment donné, nous supposons que l'utilisateur 'Alain' accède à l'application de répertoire partagé à partir du poste fixe de son bureau. Dans ce cas, nous pouvons considérer le contexte d'utilisation d'Alain comme étant l'objet `Description_de_Contexte` présenté dans la Figure 22a, lequel est composé par les objets de la classe `Élément_de_Contexte` également représentés dans la même figure, et par les relations entre ces éléments (cf. Figure 22b). Dans une deuxième situation, nous pouvons supposer que, suite à un déplacement, le contexte d'Alain a changé. Nous pouvons supposer, par exemple, qu'Alain n'utilise plus son ordinateur fixe, mais

[62]Les valeurs des attributs de chaque objet représenté dans les figures 22 et 24, à l'exception des objets de la classe `Dispositif`, ont été omises dans un souci de simplification des images.

un ordinateur de poche, et que sa localisation n'est plus connue. Dans ce cas, l'objet `Description_de_Contexte` et les objets de la classe `Élément_de_Contexte` relatifs au contexte d'Alain changeront afin de refléter sa nouvelle situation : les objets « `bureau322` » et « `dimension670` », qui représentaient, respectivement, le bureau et l'ordinateur fixe d'Alain, disparaissent pour faire place à l'objet «`treo650` », qui représente son ordinateur de poche. Cette nouvelle situation est présentée par la Figure 24.

On observe, dans les figures 22 et 24, que la description de contexte de l'utilisateur Alain contient tous les groupes auxquels il appartient et tous les rôles qu'il peut tenir dans ces groupes. Ce comportement est un choix et n'est pas obligatoire. Certains collecticiels pourront s'intéresser plus particulièrement à la représentation du rôle effectivement joué par un membre du groupe à un moment donné, et non à tous les rôles potentiels. Ce choix dépend directement de la couche d'acquisition du contexte qui va alimenter le modèle. Par ailleurs, on observe également, dans la Figure 24, la présence de l'association `conçu_pour` qui n'apparaît pas dans la Figure 22. Ceci est encore un exemple de l'effet que le processus d'acquisition de contexte a sur les instances du modèle. Selon son implémentation, plusieurs possibilités de représentation peuvent être mises en place. En d'autres termes, la décision finale concernant cette représentation appartient aux concepteurs qui utilisent le modèle lors de la création d'un système sensible au contexte. Nous reviendrons sur la question de l'acquisition de contexte et de la mise en œuvre du modèle dans la section 5.1.4.

Outre la représentation du contexte courant d'un utilisateur, ce modèle peut également être utilisé pour exprimer le contexte lié à une entité du système. Par exemple, un objet donné (un tableau, une imprimante, un rapport…) qui se trouve dans une localisation donnée et qui est utilisé dans une certaine activité peut être associé à une description de contexte. Celle-ci contient les éléments de contexte faisant référence à cette localisation et à cette activité (voir Figure 25). Un tel cas permettrait, par exemple, l'utilisation de ce modèle pour la construction de requêtes en tant que paramètres pour ces requêtes. Par exemple, une requête demandant à trouver tout objet lié à une certaine localisation peut se traduire en une requête qui demande tout objet dont la `Description_de_Contexte` associée inclut cette localisation. Ainsi, les instances du modèle proposé peuvent intervenir dans l'expression de règles et de conditions visant à sélectionner, parmi les informations disponibles, celles qui sont pertinentes pour le contexte actuel d'un utilisateur. Néanmoins, pour que ceci soit possible, il faut que ce modèle soit mis en œuvre de manière à faciliter non seulement la réalisation de requêtes, mais également l'association entre les instances du modèle et les informations mises à disposition de l'utilisateur par le système. Dans les sections suivantes, nous traitons des propriétés et de la mis en œuvre du modèle.

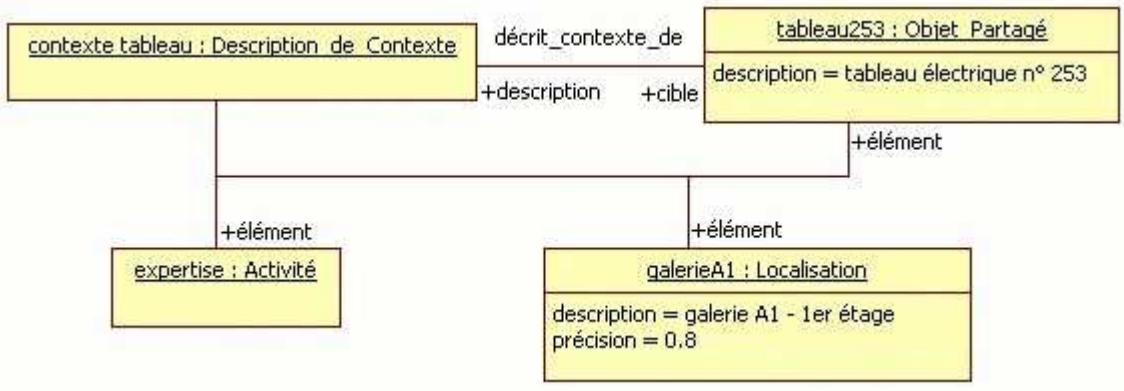


Figure 25. Exemple d'utilisation du modèle pour l'expression des informations contextuelles liées à un objet partagé.

5.1.3 Propriétés du modèle

Le modèle présenté dans la section précédente formalise, à travers une représentation par objets de type classe/association, une notion de contexte d'utilisation qui appréhende à la fois les aspects physiques et collaboratifs. Dans cette section, nous présentons les propriétés de ce modèle, et notamment comment celui-ci satisfait les critères que nous avons retenus pour les représentations de contexte (cf. section 3.5).

Le critère concernant l'évolution de la notion de contexte dans le temps est directement lié à l'objectif même de la représentation de contexte. La notion de contexte peut évoluer dans le temps, notamment en raison des modifications dans les technologies d'acquisition (cf. chapitre 3). Ainsi, la disponibilité de nouveaux capteurs ou de nouvelles techniques peut permettre l'acquisition d'autres éléments de contexte qui n'étaient pas prévus à l'origine. Dans ce type de situation, afin que le système soit capable de prendre en compte les nouveaux éléments, il faut non seulement un modèle de contexte qui isole les informations propres au contexte de celles propres au domaine d'application (de manière à réduire l'impact de ces modifications sur le système), mais il faut également que ce modèle puisse lui-même évoluer. Si le modèle ne permet pas facilement l'ajout d'un nouvel élément ou la modification d'un élément existant, l'évolution de la notion de contexte aura difficilement lieu.

Le modèle de contexte que nous proposons utilise un formalisme par objets de type classe/association, et de ce fait, il présente naturellement une capacité d'évolution à travers la relation de spécialisation et le mécanisme d'héritage propres au formalisme objet. Ainsi, il est possible d'étendre le modèle par l'ajout de nouvelles classes d'élément de contexte. Il s'agit de spécialiser la classe `Élément_de_contexte` dans de nouvelles sous-classes qui pourront représenter de nouveaux concepts introduits. À travers le mécanisme d'héritage, la nouvelle sous-classe hérite des attributs de la super-classe et peut, en parallèle, définir de nouveaux attributs. Les objets de la sous-classe peuvent également prendre place dans les associations définies pour la super-classe. Par conséquent, la nouvelle classe peut s'inscrire facilement dans le modèle.

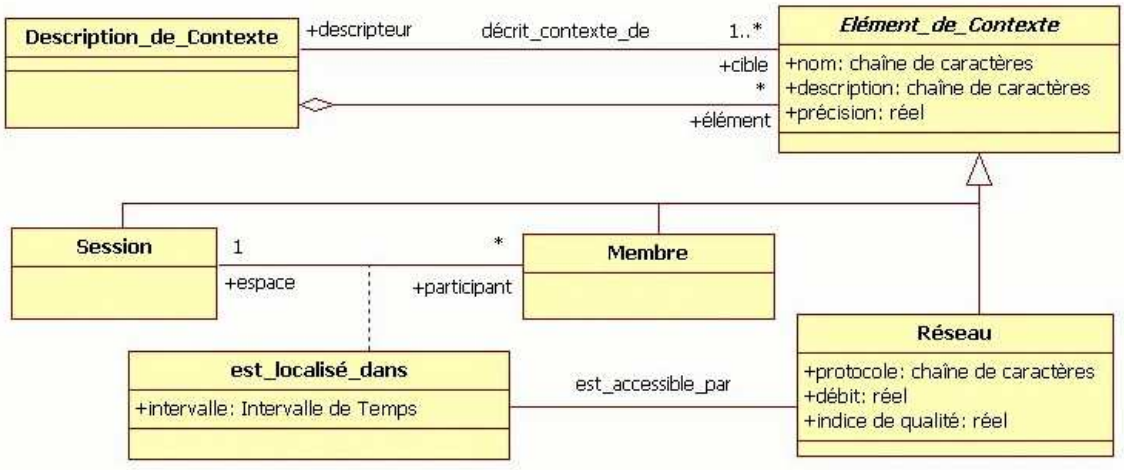


Figure 26. Exemple d'introduction d'un nouvel élément.

Avec les nouvelles classes, il est également possible d'étendre le modèle par l'ajout de nouvelles associations. En réalité, les nouvelles classes doivent également se positionner par rapport aux autres classes du modèle. Ces nouvelles classes, au même titre que celles déjà définies, ne doivent pas être isolées. Au contraire, elles doivent également participer au tissu de relations qui lient les éléments de contexte entre eux, permettant ainsi une description plus

complète du contexte. La Figure 26 donne un exemple d'une extension possible du modèle. Dans cet exemple, nous imaginons l'ajout d'une nouvelle classe pour représenter le concept de réseau à travers une classe `Réseau`. À travers celle-ci, on pourra représenter les caractéristiques et l'état du réseau utilisé par l'utilisateur pour se connecter au système. La nouvelle classe `Réseau` se positionne donc dans le modèle comme une sous-classe d'`Élément_de_Contexte`, au même titre que les autres classes d'élément, tels que `Membre` ou `Session`. Elle présente également une nouvelle association qui la connecte à d'autres classes du modèles : l'association `est_accessible_par` qui représente le fait qu'un membre se connecte au système par le biais d'une connexion réseau. Enfin, la nouvelle classe présente des attributs qui lui sont propres, tels que l'identification du protocole utilisé (attribut `protocole`), le débit maximal (attribut `débit`), ou encore un possible indicateur de qualité (attribut `indice_de_qualité`). Une fois insérée dans le modèle, la nouvelle classe `Réseau` et l'association `est_accessible_par` peuvent participer normalement à la description d'un contexte d'utilisation.

Une autre possibilité d'extension du modèle réside dans l'évolution des classes. De la même façon que pour les nouveaux éléments, la disponibilité de nouvelles techniques (ou capteurs) pour l'acquisition d'un élément de contexte donné peut inciter les concepteurs d'un système sensible au contexte à changer la définition de cet élément. De même, l'analyse du comportement des utilisateurs auprès du système peut également révéler de nouveaux besoins qui peuvent occasionner des modifications sur la notion de contexte. L'idée est donc de permettre l'évolution de ces classes, par modification de leur description. Cette évolution est possible également par la spécialisation des classes représentant les éléments de contexte. Ces modifications peuvent alors se traduire par l'introduction de nouveaux attributs ou par la redéfinition (« *overriding* ») d'attributs définis dans les super-classes. La redéfinition est possible dans les représentations par objets. Ces mécanismes permettent de répondre au critère d'évolution du modèle.

Le critère concernant le caractère dynamique de l'information contextuelle : la représentation doit être capable de refléter ce dynamisme. Nous avons vu dans le chapitre 3 qu'une couche d'acquisition de contexte est censée alimenter le modèle avec les informations contextuelles. L'acquisition de contexte est un processus continu qui doit détecter tout changement effectué sur les éléments de contexte observables par le système. Puisque ce processus alimente continuellement le modèle de contexte, il faut que celui-ci soit capable de répondre à ces changements de manière à respecter le caractère dynamique de l'information contextuelle.

Dans le cas du modèle de contexte proposé ici, le caractère dynamique des informations est assuré par les modifications possibles des instances du modèle. Comme nous pouvons l'observer dans la Figure 22, un contexte d'utilisation (dans le cas de cette figure, le contexte de l'utilisateur 'Alain') est représenté par un objet de la classe `Description_de_Contexte` qui est composé par un ensemble d'objets et par les associations qui les lient. En d'autres termes, le contexte d'utilisation est représenté par un ensemble d'instances des classes et des associations présentées dans les figures 16 et 23. Les modifications dans le contexte d'utilisation se traduisent alors par des modifications soit sur le contenu d'au moins une de ces instances, soit sur la composition de l'objet `Description_de_Contexte` relatif à ce contexte d'utilisation. La première possibilité correspond notamment à la mise à jour de la valeur d'un attribut. Par exemple, nous pouvons imaginer, dans la Figure 22, que la valeur de l'attribut `mémoire_disponible` de l'objet « `dimension670` » pourra changer de `182` (182Mo de mémoire disponible pour l'exécution) à `120` (120Mo). De même, la valeur de l'attribut `niveau_d'énergie` de l'objet « `treo650` », représenté dans la Figure 24, pourra évoluer de `48` (niveau de batterie à 48%) à `24` (niveau de batterie à 24%). La deuxième possibilité est également illustrée par la Figure 24, dans laquelle on voit que le même objet `Description_de_Contexte` de la Figure 22 a évolué, puisque sa composition a

changé : il contient un objet de la classe `Dispositif` différent de celui de la Figure 22, et il ne contient plus d'objet de la classe `Localisation`. Par ailleurs, ces modifications peuvent se faire également au niveau des associations, par l'ajout ou la suppression d'une association, ou encore par la modification d'attributs liés à cette association (notamment pour les classes-associations). La Figure 24 illustre également cette possibilité à travers l'association `conçu_pour`, qui n'existe pas dans la Figure 22.

Le critère ayant trait à la représentation d'informations incomplètes ou ambiguës, n'est apparemment pas respecté, pour l'instant, par aucune des représentations de contexte étudiées dans la section 3.3.2. Or, nous avons vu dans le chapitre 3 que le processus d'acquisition du contexte est souvent incapable de détecter certains éléments de contexte dans leur totalité. Dans ce cas, le processus d'acquisition fournit un ensemble d'informations contextuelles qui est incomplet par rapport au modèle. Celui-ci prévoit un certain nombre d'éléments, pour lesquels il ne disposera pas toujours des informations. Il faut donc que le modèle soit capable de représenter partiellement l'ensemble d'éléments de contexte. Si on observe le modèle ici proposé (cf. Figure 16), on remarque que la composition qui unit les classes `Description_de_Contexte` et `Élément_de_Contexte` permet la description de contextes incomplets par l'omission d'objets de certaines classes d'éléments du modèle. Autrement dit, cette composition n'oblige pas que toutes les sous-classes d'`Élément_de_Contexte` prévues dans le modèle soient représentées dans une description de contexte. Celle-ci ne compte pas obligatoirement sur la présence d'un objet de chaque classe du modèle. Ainsi, si le processus d'acquisition est incapable de détecter un ou plusieurs éléments de contexte (par exemple, la localisation ou les activités d'un utilisateur), il peut tout simplement ne pas créer les objets correspondants aux éléments inconnus, ni associer à un objet `Description_de_Contexte` des objets existants déjà s'il n'y a pas suffisamment d'informations pour affirmer leur appartenance au contexte d'utilisation. La Figure 24 illustre cette situation. Dans cette figure, on observe que plusieurs classes d'`Élément_de_Contexte` prévues dans le modèle ne sont pas représentées dans l'exemple : la localisation de l'utilisateur 'Alain', ainsi que ses activités, ou encore le calendrier des groupes auxquels il participe ne sont pas représentés dans l'exemple.

En revanche, l'omission de la totalité d'un élément de contexte peut s'avérer inadéquate. Puisque le modèle que nous proposons se base sur une représentation par objets, chaque élément de contexte est décrit par une classe qui réunit plusieurs informations concernant l'élément. Il est donc possible que le processus d'acquisition de contexte ne soit pas en mesure de fournir la totalité de ces informations. Dans ce cas, le modèle n'est alimenté que par des informations partielles sur un élément donné. Dans cette situation, l'omission totale de cet élément de contexte est inadéquate, car il existe, malgré tout, un certain nombre d'informations disponibles le concernant. Dans ce cas, l'idéal est de laisser non évalués, dans ces objets, les attributs dont la valeur est inconnue. Les représentations par objets permettent, en l'absence d'une contrainte exigeant la spécification d'une valeur, l'expression d'attributs non évalués dans une instance. Le standard UML a même défini pour cet usage le littéral « `null` » qui indique l'absence de valeur (voir [OMG 2004]). Ainsi, tout objet d'une quelconque sous-classe d'`Élément_de_Contexte` peut contenir des attributs dont la valeur est « `null` », donc inconnue (voir, par exemple, dans la Figure 22, l'attribut `profil_CCPP` de l'objet « `dimension670` »).

Parallèlement à la question de la représentation partielle du contexte, se pose la question de la gestion des éventuelles incohérences entre les éléments de contexte, ou encore de la duplication de ces éléments. Le processus d'acquisition de contexte (cf. section 3.3.3) peut être à l'origine de ces phénomènes. L'utilisation de plusieurs capteurs (ou de différentes technologies) pour la détection d'un élément de contexte peut entraîner l'acquisition d'informations incohérentes ou redondantes. La distribution même de ces capteurs facilite l'occurrence de ces problèmes. Lors de l'utilisation distribuée de plusieurs *widgets*, dans

l'infrastructure *Context Toolkit* [Dey 2000], ou de plusieurs *contexteurs*, dans le modèle de Rey et Coutaz [Rey 2004] (cf. section 3.3.3), les données captées peuvent varier d'un capteur à l'autre, et nous ne pouvons pas supposer que l'interprétation de ces données, réalisée par les *interpreters* dans le *Context Toolkit* [Dey 2000], ou par d'autres *contexteurs* dans Rey et Coutaz [Rey 2004], pourra toujours éliminer ces incohérences. Par exemple, supposons l'utilisation de balises électroniques (d'étiquettes RFID) pour la détection de la localisation. Deux capteurs de ce type étant suffisamment rapprochés (par exemple, en deux salles contiguës) pourront capter la présence d'un utilisateur qui se déplace entre les deux. Dans ce cas, les données captées pourront laisser supposer que l'utilisateur se trouve en deux salles distinctes (donc, deux localisations) au même moment. Si l'interprétation n'a pas pu départager ces données, nous aurons, au niveau du modèle de contexte, deux objets distincts de la classe `Localisation` pour un même utilisateur.

Le modèle de contexte que nous proposons ici permet la représentation de ce type de situation grâce à la multiplicité des relations, surtout de la composition qui unit une *description du contexte* aux *éléments de contexte*. Nous avons choisi une approche dans laquelle la présence d'incohérence entre le contenu de différents objets est permise. En d'autres termes, au lieu d'essayer de choisir entre les objets dupliqués ou incohérents (au risque de prendre la mauvaise décision), nous adoptons une représentation plus riche qui n'exclut aucun élément. Ceci signifie que nous déléguons la prise de décision au système qui exploite le modèle de contexte. Cependant, il faut que ce système soit en mesure de tirer profit de cette situation. Dans le cas précis de ce travail, nous utilisons ce modèle de contexte au sein d'un processus d'adaptation de l'information de conscience de groupe (cf. chapitre 7). Pour que ce processus puisse profiter de cette propriété du modèle, nous utilisons au sein du processus une série d'opérations définies sur le modèle (voir chapitre 6), qui prennent en compte la présence de plusieurs éléments répétés ou incohérents dans une description de contexte. À travers ces opérations, le processus d'adaptation peut alors utiliser le modèle de contexte et prendre les décisions qui conviennent même en présence de ces phénomènes.

Enfin, concernant le critère de la précision des informations contextuelles, nous avons vu dans le chapitre 3 que le processus d'acquisition de contexte n'est pas infaillible, et, par conséquent, que toute représentation du contexte doit permettre l'expression de la précision (ou de la qualité) des informations. Dans notre modèle, la précision des informations contenues dans un objet est possible à travers l'attribut `précision` défini dans la super-classe `Élément_de_Contexte` (voir Figure 16). L'objectif de cet attribut est de permettre à la couche d'acquisition d'indiquer le degré de précision de l'information représentée par un objet `Élément_de_Contexte`. L'idée ici est de calculer cet indice, pendant le processus d'acquisition de chaque élément, selon la précision des informations dont dispose la couche d'acquisition. L'indicateur qui en résulte (un nombre réel entre `0` et `1`) pourrait alors être utilisé comme un critère de décision à l'intérieur d'un processus d'adaptation. Par exemple, si l'information est suffisamment précise (c'est-à-dire, si la valeur de cet attribut est suffisamment élevée), l'adaptation devrait la prendre en compte, et dans le cas contraire, elle pourrait être ignorée.

Pour conclure, il est important d'observer que, bien que ce modèle de contexte vise la représentation du contexte d'utilisation dans un collecticiel sur le Web, nous croyons que la structure du modèle (en particulier la `Description_de_Contexte` vue comme une composition de plusieurs objets `Élément_de_Contexte`) assure la généralité nécessaire à son application à d'autres situations. À travers notamment les relations de généralisation et de spécialisation intrinsèques aux représentations par objets, nous pensons que ce modèle de contexte peut être étendu à d'autres systèmes, non seulement aux collecticiels, et à une échelle plus importante que celle d'une application informatique isolée. Nous pouvons imaginer, par exemple, à l'échelle d'une organisation, l'introduction d'éléments de contexte décrivant la culture et la structure de l'organisation, et donc, l'utilisation d'un même modèle de contexte mis en

commun à tous les systèmes de cette organisation. Nous croyons également que le modèle que nous proposons est capable de contribuer, à l'intérieur d'un système sensible au contexte (coopératif ou non), à l'adaptation du comportement du système, surtout en ce qui concerne le contenu délivré aux utilisateurs. Le chapitre 7 propose un mécanisme de filtrage qui vise l'adaptation du contenu informationnel fourni à un utilisateur, notamment les informations de conscience de groupe, à l'aide du modèle proposé ici.

5.1.4 Mise en œuvre du modèle

Le modèle proposé dans la section 5.1.2 se présente comme un modèle conceptuel, qui doit être concrétisé au moment de sa mise en œuvre à l'intérieur d'un collecticiel sensible au contexte. Ce modèle est, par définition, indépendant de toute technique spécifique à la mise en œuvre, et notamment indépendant de tout modèle ou technologie d'acquisition de contexte. C'est seulement au moment de la mise en œuvre que les détails d'implémentation sont abordés. Celle-ci comporte, par conséquent, plusieurs défis, dont nous soulevons les principaux dans cette section.

La mise en œuvre du modèle passe, avant tout, par la traduction de ce modèle conceptuel vers un modèle concret qui s'intègre à l'intérieur d'un système précis. Cette traduction est à la charge des concepteurs du système qui désirent employer le modèle. À travers cette traduction, les concepteurs doivent s'approprier le modèle, comprendre ses éléments, et les adapter à leur situation concrète, c'est-à-dire les adapter au système en conception.

Concrètement, les concepteurs d'un collecticiel sensible au contexte doivent choisir et spécifier les classes de ce modèle conformément aux besoins du système à concevoir. Les concepteurs ne sont pas obligés d'employer dans leur système toutes les classes prévues dans le modèle. Ils peuvent, en fait, choisir parmi les classes d'`Élément_de_Contexte` que nous proposons celles qui s'appliquent à leur cas. Par ailleurs, les concepteurs doivent spécifier le contenu de chacune de ces classes d'élément. Ils ne sont pas tenus d'utiliser exactement les attributs que nous suggérons. Ces attributs, ainsi que les éléments de contexte que nous avons identifiés, sont des suggestions de ce que nous croyons être le plus important (cf. sections 5.1.1 et 5.1.2). Ce sont essentiellement des pistes pour les concepteurs, puisqu'il est impossible d'énumérer de manière générale tous les éléments possibles pour la description du contexte d'utilisation, et ceci même dans le cadre spécifique des collecticiels sur le Web. De plus, les concepteurs, selon les besoins du système en conception, peuvent, par la même occasion, étendre le modèle à travers la proposition de nouvelles classes, ou encore la spécialisation des classes existantes.

Ainsi, les concepteurs d'un collecticiel sur le Web peuvent décrire dans le détail les activités et les rôles propres au système en conception à travers la spécialisation des classes `Activité` et `Rôle`. Ils peuvent également prédéterminer les types d'objets partagés que le système est capable de manipuler par la spécification de sous-classes de la classe `Objet_Partagé`, ou encore, définir de nouvelles contraintes entre ces objets et les activités. Ils peuvent aussi ajouter une nouvelle classe `Réseau`, afin de représenter les caractéristiques et l'état du réseau, comme dans l'exemple proposé dans la Figure 26. Par exemple, nous pouvons imaginer que les concepteurs d'un collecticiel sur le Web pour le partage de documents pourront déterminer plusieurs sous-classes d'objet partagé selon le type de document : `Document_Texte`, `Document_Vidéo`, `Audio`, `Répertoire`, etc. Ils pourront aussi créer des sous-classes d'activité pour chaque action autorisée pour les membres du groupe : `Dépôt_Document`, `Mise_à_Jour_Document`, `Verrouillage_Document`, etc. Pour d'autres systèmes, la traduction du modèle de contexte sera différente. Par exemple, dans le cas d'un éditeur

coopératif tel qu'*AllianceWeb* [Salcedo 1998] [Martínez 2002b], les concepteurs auront probablement des classes d'activité différentes du précédent, puisque les actions menées dans le système changent d'un système à un autre : `Nouveau_Document`, `Modification_Document`, `Révision_Document`, etc. Par ailleurs, les concepteurs d'un collecticiel qui n'adopte pas un processus structuré de manière explicite, comme *AllianceWeb*, n'éprouveront peut-être pas le besoin d'inclure dans le modèle la classe `Processus`. Ceci ne sera probablement pas le cas d'un collecticiel qui permet la définition explicite du processus de travail, tel que *ToxicFarm* [Skaf-Molli 2003]. En ce qui concerne les rôles, le modèle peut être étendu afin de représenter les différents niveaux de responsabilité qui existent entre les membres du groupe. On retrouve typiquement un rôle de coordinateur et un rôle de participant, ce qui peut être traduit par deux sous-classes de `Rôle` : `Coordinateur` et `Participant`. Nous pouvons alors remarquer, à travers ces exemples, l'importance de cette étape de traduction du modèle de contexte vers le système où il sera appliqué. C'est dans cette étape que le modèle est adapté aux besoins spécifiques du système en question.

D'autre part, il est important d'observer qu'un système conçu avec ce modèle doit manipuler le contexte d'utilisation à travers les objets de ces classes, et plus particulièrement à travers les objets de la classe `Description_de_Contexte`. Chaque objet de la classe `Description_de_Contexte` représente, pour le système, le contexte d'un élément donné (un utilisateur, un dispositif, une application, etc.). Au fur et à mesure que les circonstances liées à cet élément évoluent, les éléments qui composent la `Description_de_Contexte` pourront aussi évoluer par leur mise à jour ou par l'introduction (ou la disparition) d'autres objets de la classe `Élément_de_Contexte`. Cette évolution des instances garantit le caractère dynamique nécessaire à l'information de contexte (cf. section 5.1.3). Néanmoins, la manipulation concrète de ces objets `Description_de_Contexte` dépend de l'implémentation du modèle. Ainsi, à la fin de cette première étape d'appropriation et d'adaptation du modèle, les concepteurs disposent d'une nouvelle version du modèle de contexte qui est adaptée à leurs besoins. C'est cette nouvelle version qui sera implémentée au sein du système en cours de conception. Pour cela, il faut une deuxième étape, qui détermine la technologie qui sera utilisée pour la mise en œuvre effective du modèle. Plusieurs possibilités s'offrent aux concepteurs : l'utilisation des langages orientés à objets, du langage XML, ou de la représentation de connaissances par objets. Chacune de ces possibilités traduit une approche différente pour la mise en œuvre du modèle.

La première approche d'implémentation, l'utilisation d'un langage orienté à objets tel que C++ ou Java, correspond à l'intégration directe du modèle de contexte au sein même de la logique d'application du système. Il s'agit de la traduction du modèle vers des classes et des objets qui sont codés directement à l'intérieur du système. Il s'agit d'une approche possible et d'application facile, car le modèle, décrit dans le formalisme objet de UML, se traduit naturellement dans ces langages. Cependant, nous déconseillons cette approche, car elle risque d'être bien trop rigide pour un système sensible au contexte. Le codage du modèle en dur dans le système rend difficile son évolution, puisque tout changement dans la structure du modèle implique un changement dans la partie du code du système lui correspondant. On peut se retrouver alors dans la même situation que la majorité des systèmes sensibles contexte, qui ne font pas la distinction entre les informations contextuelles et celle du domaine d'application. En somme, cette approche met en péril l'application effective des propriétés du modèle.

Une autre approche possible est l'utilisation d'un dialecte XML. Dans cette approche, le modèle est traduit vers un schéma XML, lequel peut être utilisé pour la construction d'une base de documents XML, où sont gardées les instances du modèle. L'utilisation de XML présente certains avantages par rapport à l'approche précédente, notamment une séparation claire entre le modèle de contexte et ses instances (les descriptions de contexte et les éléments

de contexte), et le système. Cette séparation peut permettre le partage du modèle entre différentes applications, et la définition d'un vocabulaire pour la représentation du contexte commun à toute une organisation. Par ailleurs, l'utilisation de XML correspond à l'utilisation d'un standard défini par la W3C, qui est déjà largement adopté par les systèmes sur le Web. Ceci est un atout non négligeable pour tout système sur le Web, puisque l'emploi des standards est un facteur clé reconnu pour l'interopérabilité de ces systèmes.

```
<contextDescription xmlns="http://jangada/DTD/context.dtd"
                    xmlns:gml="http://www.opengis.net/gml"
                    id="cd001" name="context alain">
   <contextualize refid="#obj1001" />
   <contextComposition>
       <contextElement  name="Alain" id="#obj1001">
           <class name="Membre" />
           <attribut name="nom" value="Alain" />
           ...
       </contextElement>

       <contextElement  name="comité" id="#obj1002">
            <class name="Groupe" />
            <attribut name="description" value="comité de programme Conf06"/>
            ...
       </contextElement>

       ...

       <contextElement  name="bureau322" id="#obj1006">
            <class name="Localisation" />
            <attribut name="description" value="bureau 322 - bâtiment D - 3et"/>
            <attributMultiVal name="coordonnées">
                <set>
                    <setElement pos="0">
                        <gml:location>
                            <gml:Point gml:id="point96" srsName="urn:EPSG:geographicCRS:62836405">
                            <gml:pos>-31.936 115.834</gml:pos>
                            </gml:Point>
                        </gml:location>
                    </setElement>
                    ...
                </set>
            </attributMultiVal>
            ...
       </contextElement>

       <contextElement  name="dimension670" id="#obj1007">
            <class name="Dispositif" />
            <attribut name="nom" value="jangada" />
            <attribut name="mémoire disponible" value="182"/>
            <attribut name="niveau d'énergie" value="100" />
            <attribut name="précision" value="1" />
            ...
       </contextElement>
       ...

   </contextComposition>
</contextDescription>
```

Figure 27. Illustration d'une possible utilisation de XML pour la mise en œuvre du modèle.

De plus, le langage XML permet également l'utilisation de l'ensemble de technologies et de standards associés, tels que XQuery[63]. Le standard XQuery facilite la formulation de requêtes et l'extraction de données sur des documents XML. À travers ce standard et les outils qui l'implémentent, il est donc possible d'exécuter des requêtes sur une base XML. Dans le cas précis de l'implémentation du modèle de contexte, on dispose alors d'outils capables de rechercher les instances du modèle qui satisfont certains critères, comme la présence d'un objet (une localisation, un objet partagé, etc.). Nous pouvons ainsi formuler des requêtes comme, par exemple, retrouver tous les utilisateurs qui se trouvent dans une telle région symbolique. Ce type de requête est traduit en trouver tous les objets `Description_de_Contexte`

[63]« W3C XML Query » - http://www.w3.org/XML/Query/.

liés (présence de l'association `décrit_contexte_de`) à un membre du groupe qui est connecté au système (présence de l'association `est_localisé_dans`) et dont la localisation physique (objet de la classe `Localisation` lié à l'objet `Description_de_Contexte`) correspond à la région symbolique voulue. Les standards XML et XQuery permettent ce type de construction, mais il reste néanmoins nécessaire que les concepteurs du système puissent prévoir et mettre en place ce type de requête au sein du système en conception.

La Figure 27 illustre une possible traduction vers XML de l'exemple présenté dans la Figure 22. On observe, dans la Figure 27, l'utilisation de la spécification GML du *OpenGIS*[64] pour la description de la localisation. Nous l'utilisons, plus précisément, pour la description des coordonnées de l'objet `Localisation`, à travers le type « `Point` », un dérivé du type « `Geometry` » selon les spécifications *OpenGIS*. En réalité, le langage XML, par le biais de son mécanisme de *namespacing*, permet l'utilisation de balises définies dans différents dialectes XML dans un même document. Ceci renforce la possible utilisation, lors de la mise en œuvre du modèle, d'autres standards définis sur XML, tels que GML l'a été dans cet exemple. Par ailleurs, on observe à travers la Figure 27 que la représentation d'informations incomplètes, soit par l'omission d'un élément de contexte (matérialisé, dans la figure, par les balises « `contextElement` »), soit par l'omission d'attributs (balises « `attribut` » ou « `attributMultiVal` », selon l'attribut) au sein d'un élément, se fait naturellement en XML.

Néanmoins, l'utilisation de XML présente certains désavantages vis-à-vis du modèle de contexte, notamment en ce qui concerne le dynamisme propre à l'information contextuelle. Traditionnellement, on ne modifie pas le contenu d'un document XML avec la même fréquence que l'information contextuelle change. Or, toute modification sur une instance du modèle doit être transmise à son implémentation, ce qui implique, dans ce cas, un changement sur le document XML contenant cette instance. Par ailleurs, la structure même du modèle de contexte peut évoluer dans le temps (cf. section 5.1.3), ce qui implique la modification du schéma XML qui implémente le modèle. Ce type de modification n'est pas toujours simple à réaliser et peut invalider la base de documents XML disponibles : les documents qui étaient valides par rapport au schéma originel peuvent n'être plus valides vis-à-vis du nouveau schéma proposé.

Une autre possibilité pour la mise en œuvre du modèle de contexte consiste à utiliser la représentation de connaissances (voir [Capponi 1995]). Deux approches nous semblent plus appropriées pour l'implémentation du modèle de contexte : l'utilisation d'ontologies à travers le langage OWL, et la représentation de connaissances par objets.

Avec l'évolution de la notion de *Web sémantique*[65], de plus en plus de systèmes sur le Web se tournent vers la représentation de connaissances à travers le standard OWL[66]. Le standard OWL est le langage de représentation du Web sémantique [Napoli 2004]. Il permet la description d'ontologies qui représentent les informations sur des catégories d'objets et les interrelations entre ceux derniers [Horrocks 2003]. L'OWL repose ainsi sur trois notions de base (voir [Horridge 2004] et [Smith 2004]) : celle d'*individus*, qui correspond à la notion d'objet dans le formalisme par objets ; celle de *classes*, qui sont interprétées comme un ensemble d'individus (ce qui correspond, dans le formalisme objet, à la notion de classe en tant qu'entité qui regroupe un ensemble d'objets aux caractéristiques et comportement similaires [Capponi 1995]) ; et celle de *propriétés*, qui sont des relations entre deux individus. La notion de propriété permet à la fois l'expression des relations binaires entre deux objets (« *object properties* »), et l'expression des attributs d'un objet. Cette dernière ce fait à travers

[64] « OpenGIS® Geographically Markup Language (GML) Encoding Specification » - http://portal.opengeospatial.org/files/?artifact_id=4700

[65] En anglais, « Semantic Web » - http://www.w3.org/2001/sw/.

[66] En anglais, « Web Ontology Language » - http://www.w3.org/2004/OWL/.

les propriétés dites « *datatype properties* », qui lient un objet à une donné précise. On peut donc s'apercevoir que, malgré certaines limitations, comme l'expression d'associations n-aires pour lesquelles il n'y a pas une traduction directe en OWL (une même association n-aire est traduite en plusieurs propriétés), il est tout à fait possible d'exprimer le modèle de contexte proposé dans ce travail à travers ce langage. Cette approche de mise en œuvre est similaire à l'utilisation du langage XML (OWL est lui-même basé sur les standards XML et RDF), mais elle offre l'avantage de permettre l'utilisation de mécanismes d'inférences propres aux représentation de connaissances qui sont disponible via les raisonneurs OWL[67].

En ce qui concerne la représentation de connaissances par objets, on retrouve les *Systèmes de Représentation de Connaissances par Objets* (SRCO). Ceux-ci se basent sur les mêmes concepts que le formalisme de représentation par objets que nous avons utilisé pour modéliser le contexte : la notion de classe, d'objet, de relation, d'attribut, de spécialisation, etc. Selon Napoli *et al.* [Napoli 2004], « *tous les systèmes à objets font appel à des entités individuelles – les objets – possédant des propriétés – les attributs – regroupés dans des entités génériques – les classes – ces classes étant elles-mêmes hiérarchisées par une relation de spécialisation permettant un héritage des propriétés* ». Par conséquent, la traduction du modèle de contexte ici proposé vers une base de connaissances à objets nous paraît comme l'approche la plus naturelle et directe pour la mise en œuvre du modèle. Une *base de connaissances*, selon Napoli *et al.* [Napoli 2004], représente un état d'un domaine modélisé, sur lequel un certain nombre d'actions (des inférences, des interrogations, des mises à jour, etc.) peuvent être exécutées afin de parvenir à la résolution d'un problème. Cette résolution d'un problème est menée à l'extérieur de la base, par le système informatique qui l'exploite et qui agit sur la base, en modifiant son état. On garde donc une séparation claire entre la base de connaissances, qui est une représentation d'un état du monde réel, et les actions qu'on réalise sur cette base.

De plus, ces systèmes possèdent d'autres mécanismes d'exploitation, comme l'inférence de la valeur d'un attribut ou la classification d'une instance (voir [Gensel 1995]), qui sont complémentaires par rapport aux mécanismes d'exploitation proposés par les ontologies et qui peuvent s'avérer intéressants pour l'exploitation du modèle. D'ailleurs, certains travaux, comme Napoli *et al.* [Napoli 2004] et Horrocks *et al.* [Horrocks 2003], font déjà le parallèle entre les représentations par objets et les représentations de connaissances de manière générale, et le langage OWL. Miron [Miron 2006] compare le langage OWL à un SRCO en particulier, nommé AROM. Celui-ci offre, entre autres, la possibilité de représenter directement la notion d'association entre les classes et un mécanisme de classification d'instances, se présentant donc comme une alternative complémentaire à OWL. Miron [Miron 2006] propose même une traduction possible entre les bases de connaissances AROM et les ontologies OWL. Dans la section 5.2, nous proposons une mise en œuvre particulière à travers le SRCO AROM.

La mise en œuvre du modèle doit être achevée par la connexion entre le modèle de contexte et celui d'acquisition. Le modèle que nous proposons ici est indépendant d'un modèle d'acquisition de contexte spécifique. Il peut donc être utilisé avec différents infrastructures telles que le *Context Toolkit* [Dey 2000] ou les *contexteurs* [Rey 2004]. Dans tous les cas, il est nécessaire de prévoir, pour chaque élément de contexte prévu dans la traduction du modèle de contexte, les moyens pour acquérir chacune des informations décrites. Ainsi, dans le cas du *Context Toolkit* [Dey 2000], par exemple, nous aurons potentiellement plusieurs *context widgets* pour chaque élément de contexte, et un *aggregator* condensant les données captées par ces *widgets* à propos d'un certain élément pour une description de contexte donnée. Ces *widgets* seront très probablement distribués, certains seront disposés à l'intérieur du serveur,

[67]Voir la disponibilité d'outils OWL sur le site officiel du langage – http://www.w3.org/2004/OWL/#projects.

d'autres au sein même du dispositif client. Au fur et à mesure que les données d'un élément sont acheminées vers l'*aggregator* chargé de cet élément, celui-ci pourra mettre à jour l'état courant du modèle. L'état courant du modèle est idéalement conservé par un gestionnaire spécifique, qui manipule l'implémentation du modèle de contexte. On observe ici que nous supposons une architecture client-serveur typique du Web.

La Figure 28 montre deux diagrammes de séquence qui illustrent le processus d'acquisition de la localisation tant avec l'infrastructure *Context Toolkit* qu'avec un ensemble de *contexteurs*. Dans les deux cas, les données de localisation sont détectées par un capteur GPS, interprétées et centralisées sur un même élément, qui informe le gestionnaire de contexte du changement de contexte d'un utilisateur donné. Nous imaginons donc que pour chaque membre du groupe connecté au système, ce même ensemble de composants sera disponible. Le gestionnaire de contexte, une fois informé du changement, pourra mettre à jour le modèle de contexte en informant les instances concernées. Ces instances sont le résultat de la traduction et de l'implémentation du modèle.

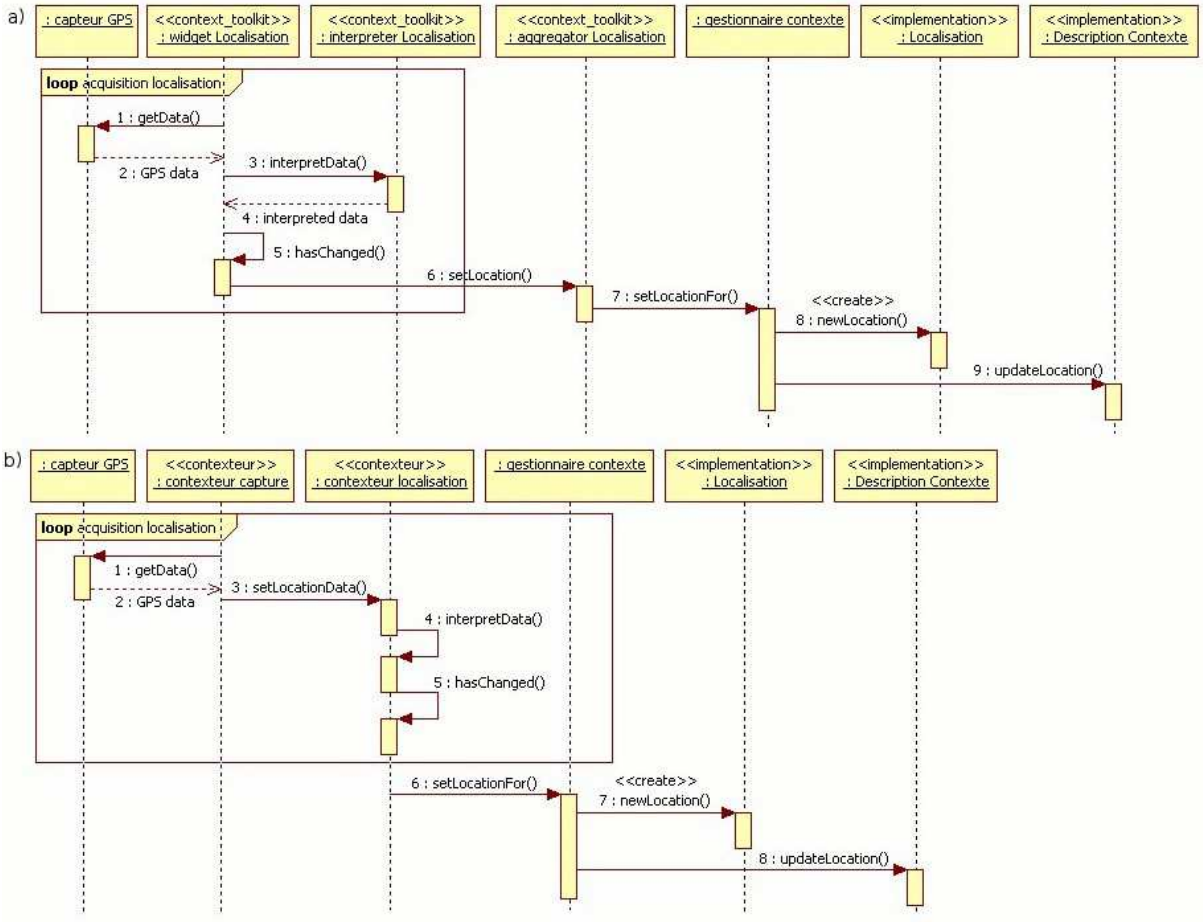


Figure 28. Diagrammes de séquence qui illustrent deux possibilités pour le processus d'acquisition de la localisation.

Par ailleurs, il est important d'observer que, pour chaque élément de contexte prévu dans le modèle, il existe plusieurs techniques pour l'acquisition des informations contextuelles. Par exemple, la localisation peut utiliser la technique utilisée par Rubinsztejn *et al.* [Rubinsztejn 2004], qui utilise la puissance des signaux du réseau sans fil, ou celle proposée par Anne *et al.* [Anne 2005], qui combine le réseau sans fil aux balises électroniques, ou encore celle de Samama *et al.* [Samama 2005], qui utilise le système GPS.

D'une part, pour la majorité des éléments du contexte physique (voir Figure 21), l'acquisition de donnés passe par l'utilisation de capteurs distribués. Par exemple, la détection du dispositif ou de l'application en exécution se place notamment sur le dispositif client lui-même. D'autre part, pour les éléments qui composent la notion de contexte collaboratif (voir Figure 20), l'acquisition passe surtout par l'analyse des informations disponibles au sein même du système (et donc, typiquement, sur le serveur, dans une architecture Web). Par exemple, la détection des groupes auxquels un membre participe et des rôles qu'il y joue passe par l'analyse de la définition de ces groupes, laquelle est une information habituellement conservée au sein du collecticiel (nous pouvons imaginer ici la présence d'un gestionnaire d'utilisateurs). L'acquisition de ces éléments consiste donc à rechercher ces informations auprès des bases de données ou des composants propres au collecticiel.

En somme, en ce qui concerne l'acquisition du contexte, les concepteurs d'un système sensible au contexte qui utilise le modèle ici proposé peuvent, non seulement étendre ce modèle selon les besoins du système, mais également choisir les techniques d'acquisition qui conviennent le mieux à chaque élément de contexte et au système.

5.2 Le Modèle AROM

Comme nous l'avons vu dans la section précédente, la représentation de connaissances par objets est une possibilité intéressante pour la mise en œuvre du modèle de contexte. Nous avons traduit le modèle tel qu'il a été présenté dans la section 5.1.2 vers une base de connaissances dans un *Systèmes de Représentation de Connaissances par Objets* (SRCO) nommé AROM. Le système AROM[68] est développé conjointement par l'équipe Helix à l'INRIA Rhônes-Alpes, par les laboratoires LIF[69] à Marseille et LSR[70] à Grenoble. Nous présentons les concepts fondamentaux de ce système dans la section suivante et, dans la section 5.2.2, nous représentons notre modèle avec les mêmes concepts utilisés par ce système.

5.2.1 Concepts de base

Le système AROM est un système de représentation de connaissances par objets écrit en Java, qui se présente sous la forme d'une interface de programmation (API) Java et d'une interface graphique, toutes les deux permettant la manipulation d'une base de connaissances. Le système AROM se démarque des autres SRCO notamment par la représentation explicite des *associations* entre les objets. La majorité des SRCO ne disposent typiquement que du concept de classe. Or, la modélisation des connaissances utilise normalement deux concepts de base : d'une part, les *classes* d'objets pour décrire les regroupements d'individus similaires, et, d'autre part, les relations ou *associations* pour regrouper les liens similaires entre ces différents individus. Cependant, les associations ne bénéficient pas, en général, d'une représentation qui leur soit propre. Elles sont fréquemment implantées à l'aide d'attributs-liens, qui sont des attributs typés par une classe (un attribut dont la valeur référence une ou plusieurs instances de cette classe) [Page 2000]. L'absence d'une représentation propre aux associations est préjudiciable à l'expressivité des SRCO, car elle oblige les concepteurs à s'occuper de l'implantation de ces associations au lieu de simplement

[68]Acronyme de « Allier Relations et Objets pour Modéliser » [Page 2000], disponible sur http://www.inrialpes.fr/romans/pub/arom/.
[69]« Laboratoire d'informatique fondamentale » – http://www.lif.univ-mrs.fr/.
[70]« Logiciels, Systèmes et Réseaux » - http://www-lsr.imag.fr/.

les représenter, ce qui nuit à l'intelligibilité de la base de connaissance [Genoud 2000]. Le système AROM, en revanche, représente explicitement les liens entre les objets par le concept d'association. Les associations occupent, au sein de ce système, une place d'importance égale à celle des classes. De plus, AROM reprend un certain nombre de principes souvent rencontrés en représentation de connaissances par objets : distinction entre classes et instances, spécialisation des classes, présence de facettes de typage, d'inférence, etc. [Page 2000].

Ainsi, le système AROM utilise un ensemble de concepts fondamentaux, dont l'origine se trouve au sein même de la thématique de recherche de représentation de connaissances. Nous présentons ici certains de ces concepts nécessaires à la compréhension et à la construction de bases de connaissances en AROM.

Classes et objets

Le premier de ces concepts est celui de ***classe***, et de son instance, l'***objet***. Dans la littérature de représentation de connaissances, une classe regroupe un ensemble d'objets similaires [Capponi 1995]. Une classe agit également comme un moule générateur d'instances en donnant la structure (la liste d'attributs) qui doit posséder chacune des instances qui lui sont rattachées [Gensel 1995]. Dans AROM, ces deux sémantiques sont conservées. Selon Page *et al.* [Page 2000], une classe décrit un ensemble d'objets ayant des propriétés et des contraintes communes. Ces propriétés qui caractérisent la classe sont appelées les ***variables*** (elles correspondent à la notion d'*attribut* dans le standard UML). Il est important d'observer que les classes AROM ne disposent pas de méthodes au sens des langages orienté objets. Par contre, chaque classe fournit un ensemble de conditions nécessaires mais non suffisantes d'appartenance des objets à la classe. Les classes sont, par ailleurs, organisées de manière arborescente par la relation de ***spécialisation*** qui, en AROM, supporte seulement un héritage simple [Bruley 2003].

Pour sa part, un ***objet*** est une instance de la classe. Il représente, dans AROM, une entité distinguable du domaine modélisé [Page 2000]. L'identité est assurée par un identificateur unique dans la base de connaissances, fourni soit par le modélisateur, soit par AROM lors de la création de l'objet [Bruley 2003]. Un objet possède une identité et se décrit par un ensemble de couples attributs-valeurs [Napoli 2004]. La structure d'une instance est formée à partir des attributs de la classe d'appartenance. L'instance contient les valeurs des attributs de la classe qui dénotent un individu en particulier [Gensel 1995]. Chaque objet AROM est attaché à une classe à un moment donné. Cependant, contrairement aux langages de programmation orienté objets, AROM autorise le déplacement d'un objet d'une classe à une autre lorsque des informations additionnelles ont été obtenues sur cet objet [Page 2000]. Ce déplacement est permis seulement si l'objet peut satisfaire les conditions d'appartenance de la classe cible. Les objets possèdent ainsi les propriétés décrites dans la classe directement ou par héritage, avec les contraintes imposées dans la classe [Napoli 2004]. Les valeurs des attributs doivent satisfaire les contraintes définies par la classe pour chaque attribut, par exemple le type ou le domaine de valeurs [Chabalier 2004]. Par ailleurs, au regard de la description de la classe, un objet peut être *complet*, et dans ce cas, tous les attributs ont une valeur, ou *incomplet*, lorsqu'il y a au moins un attribut qui n'a pas de valeur [Gensel 1995].

Lorsqu'on parle de la description d'une classe, on parle de son ***intension***. L'intension d'une classe est donc formée par l'ensemble des attributs (des variables, dans le vocabulaire AROM) de la classe [Bruley 2003] [Gensel 1995]. En revanche, lorsqu'on mentionne les instances d'une classe, on parle de son ***extension***. L'extension d'une classe est donc l'ensemble d'instances de cette classe [Gensel 1995]. L'intension de la classe représente les conditions d'appartenance à la classe. C'est l'intension de la classe qui décrit les contraintes auxquelles

les objets doivent satisfaire pour pouvoir participer à l'extension de la même classe. De manière générale, l'intension d'une structure (une classe, mais aussi une association, comme nous allons voir ci-dessous) permet d'identifier les propriétés communes aux individus d'une même structure, tandis que l'extension correspond à l'ensemble d'individus (les instances, qui sont les objets pour les classes) qui appartiennent effectivement à la structure [Bruley 2003].

À travers l'intension d'une classe, on aperçoit les ***variables*** qui composent la description de la classe. Dans AROM, une variable correspond à une propriété (un attribut) dont le type n'est pas issu d'une classe de la base de connaissances [Page 2000]. Puisque le système AROM permet la représentation explicite des associations, tout lien entre deux objets passe obligatoirement par la définition d'une association. Chaque variable sur AROM se caractérise par un ensemble de facettes qui se divisent en trois catégories principales [Genoud 2000] [Page 2000] :

(i) les *facettes de restriction de domaine*, qui correspondent aux *facettes de type*, aux *facettes de domaine* et aux *facettes min-card* et *max-card*. Une facette de *type* précise le type de la variable. AROM supporte les types de base traditionnels (entier, réel, booléan et chaîne de caractères), en plus des types multi-valuée (ensemble et liste de valeurs). Une facette de *domaine* définit l'ensemble des valeurs admissibles pour la variable. Les facettes *min-card* et *max-card* restreignent, respectivement, le nombre minimum et maximum de valeurs pour une variable multi-valuée (en d'autres termes, elles donnent la cardinalité de la valeur d'une telle variable) ;

(ii) les *facettes d'inférence*, qui permettent d'inférer la valeur de la variable. Les facettes d'inférence les plus connues sont la *facette default*, la *facette de définition* et la *facette d'attachement*. Une facette *default* indique la valeur par défaut de la variable. La facette de *définition* permet le calcul de la valeur à travers une équation exprimée dans le langage de modélisation algébrique (LMA) offert par AROM. Une facette *d'attachement*, quant à elle, associe une portion de code en langage Java, externe à la base de connaissances, lequel permet de calculer la valeur de la variable ;

(iii) les *facettes de documentation* qui incluent les facettes *documentation* et les facettes *unit*. Les premières permettent l'association d'une documentation générale à propos de la variable, et les secondes indiquent l'unité dans laquelle s'exprime la valeur de la variable.

Ainsi, pour qu'une instance (un objet) puisse être rattachée à une classe (et puisse donc faire partie de l'extension de la classe), les valeurs de ses variables doivent satisfaire notamment les conditions fixées par les facettes de restriction de domaine.

Par ailleurs, les classes, dans une base de connaissances AROM, peuvent s'organiser de manière structurée, par la relation de spécialisation. La ***spécialisation*** d'une classe AROM est réalisée soit par ajout ou modification d'une facette, soit par ajout d'une nouvelle variable. Ainsi, la relation de spécialisation se comporte comme une relation d'ordre partiel qui organise les classes en une hiérarchie. De plus, les sous-classes d'une classe ne sont pas supposées mutuellement exclusives, ni exhaustives, ce qui signifie que leurs descriptions peuvent englober des instances communes et l'union de l'extension des sous-classes n'est pas nécessairement égal à l'extension de la super-classe [Page 2000] [Genoud 2000]. D'autre part, la sémantique de la relation de spécialisation est celle de l'inclusion ensembliste : une classe s'interprète comme un ensemble d'individus et les individus appartenant à l'interprétation d'une classe doivent appartenir à celle des super-classes [Napoli 2004]. En d'autres termes, l'extension d'une sous-classe est un sous-ensemble de l'extension de sa super-classe. Ceci est une conséquence directe du mécanisme d'héritage : lorsqu'une classe `c'` spécialise une classe `c`, elle hérite toutes les variables de `c` qu'elle n'a pas redéfini. La redéfinition se fait par la

modification des facettes, laquelle ne permet que la restriction des valeurs qui étaient acceptables pour cette variable dans C. Selon Chabalier [Chabalier 2004], la sous-classe possède tous les attributs de la super-classe avec des domaines de valeurs égaux ou inclus, et peut décrire de nouveaux attributs qui lui sont propres. Par conséquent, toute valeur qui est acceptable pour une variable sur C' le sera aussi pour la même variable sur C. Ainsi, si une instance peut être rattachée à C', elle appartiendra automatiquement à l'extension de C', et à celle de C. Par exemple, l'objet treo650 rattaché à la classe Dispositif (voir Figure 24) doit accepter des valeurs non seulement pour les attributs profil_CCPP, mémoire_disponible et niveau_d'énergie, définis par l'intension de cette classe, mais également des valeurs pour les attributs nom, description et précision, définis par l'intension de la super-classe Élément_de_Contexte.

Un objet AROM est rattaché à une classe spécifique dont l'intension est définie à travers les variables. Les objets acceptent donc des valeurs pour ces variables. Ainsi, un objet qui appartient à une classe définit un ensemble de couples « variable/valeur ». Toutes les variables de la classe ne sont pas obligatoirement valuées dans un objet, mais elles doivent être reconnues : l'objet doit pouvoir accepter à tout moment une valeur pour cette variable [Bruley 2003]. Ceci est valable non seulement pour les variables directement définies dans la classe à laquelle l'objet est rattaché, mais également pour les variables qui sont hérités à partir des super-classes.

Ainsi, d'une manière plus formelle, nous pouvons noter ces concepts comme suit[71] :

- l'intension (*I*) d'une classe C dans une base de connaissances k est notée :

  $C_{k,I}=\{a_i : d_i\}_{i=1..n}$, où a_i est l'identificateur de la variable (attribut), et

  d_i est le domaine de valeurs acceptables pour la variable a_i.

- l'extension (*E*) d'une classe C dans une base de connaissances k est notée :

  $C_{k,E}=\{o_i : C'\}_{i\geq 1}$, où o_i indique une instance de la classe C' et

  C' est, soit une sous-classe de C, soit C elle-même.

- un objet o est interprété comme étant :

  $o=\{a_i=v_i\}_{i\geq 1}$, où a_i est l'identificateur de la variable (attribut), et

  v_i est la valeur attribuée à la variable a_i.

- l'appartenance d'un objet o à l'extension d'une classe C dans la base de connaissances k est notée par :

  $$o \in C_{k,E}$$

- la valeur v_j d'une variable a_i dans un objet o est notée par :

  $$o.a_j \text{ ou } v_j, \text{ pour un objet } o=\{a_i=v_i\}_{i\geq 1} \text{ avec } j\in[1..i]$$

- la spécialisation directe d'une classe $C_{k,I}=\{a_i : d_i\}_{i=1..m}$ dans une classe $C'_{k,I}=\{a'_i : d'_i\}_{i=1..n}$ dans la base de connaissances k est notée par C' < C. La spécialisation directe n'est valable que si :

[71]Cette formalisation est inspirée notamment de [Capponi 2005].

$$(C'<C)_k \;\; est\ valable\ ssi \;\; \begin{cases} \forall i \in [1..m], \exists j \in [1..n] \;\; a_i = a_j' \;\; et \;\; d_j' \subseteq d_i, \\ \neg \exists C'' \;\; (C'<C'')_k \;\; et \;\; C'' \neq C \end{cases}$$

En d'autres termes, chaque variable définie dans la super-classe `C` se trouve également dans la sous-classe `C'`, le domaine de valeurs pour la sous-classe est un sous-ensemble du domaine de valeurs de la super-classe `C` ; et il n'existe pas de classe `C''`, différente de `C`, dont la classe `C'` serait aussi une spécialisation directe (*il n'y a pas d'héritage multiple dans AROM*).

Associations et tuples

L'autre concept clé du système AROM est celui d'***association*** et les instances des associations, qui sont appelées ***tuples***. Une association représente un ensemble de liens similaires entre `n` (`n ≥ 2`) classes, distinctes ou non. Un lien est un *n-uplet* d'objets appartenant aux extensions des classes reliées par l'association [Page 2000]. Une association AROM est nommée, et peut posséder des ***variables*** qui décrivent ses propriétés intrinsèques, et qui sont spécifiés par les mêmes facettes que les variables dans une classe. Une association *n-aire* en AROM dénote donc les liens décrits par ces variables et impliquant chacun `n` objets (le *n-uplet* du tuple), chaque objet ayant au moins un rôle[72] spécifié au sein de cette association. Le lien entre une association et une classe partenaire se fait par un rôle nommé, typé par cette classe, et pour lequel la multiplicité traduit, à l'exemple du langage UML, le nombre de fois qu'un même objet peut être partenaire dans cette association, ses autres partenaires étant fixés. [Chabalier 2003] [Chabalier 2004].

Une association est donc décrite notamment par ses rôles et ses variables. Un rôle `r` d'une association correspond à une connexion entre l'association et une des classes partenaires, également appelée *classe correspondante* et souvent notée `C(r)`. Une association *n-aire* possède donc `n` rôles et la valeur de chaque rôle r_i ($1 \leq i \leq n$) est une instance de la classe correspondante $C(r_i)$. La multiplicité de chaque rôle est décrite par une facette *multiplicity* pour laquelle est donnée la valeur minimum et la valeur maximum (le symbole `*` dénote une valeur infinie). On peut également associer à un rôle une facette *documentation*, au même titre que pour les variables dans les classes [Page 2000].

L'instance d'une association est appelée un ***tuple*** : un tuple d'une association *n-aire* possédant `m` variables v_i ($1 \leq i \leq m$) est le *(n+m)-uplet* formé des `n` objets du lien et des `m` variables de l'association [Genoud 2000] [Page 2000]. Les tuples doivent non seulement définir des valeurs pour les variables (comme le font les objets), mais ils doivent également référencer les objets pour les différents rôles. Dans un tuple AROM, tous les rôles doivent nécessairement être définis, contrairement aux variables qui peuvent être non valuées [Bruley 2003].

Une association dans AROM est une entité au même titre qu'une classe. Elle a donc une ***intension***, qui est définie par les rôles et les variables, avec leurs facettes respectives, et une ***extension***, qui correspond à l'ensemble des tuples.

De la même manière que les classes, les associations peuvent aussi être organisées à travers la relation de ***spécialisation***. La spécialisation entre associations implique la spécialisation des classes références par les rôles et la restriction des multiplicités, mais l'arité reste invariable [Chabalier 2003]. La relation de spécialisation pour les associations garde ainsi le même caractère d'inclusion ensembliste qu'on retrouve dans la spécialisation des classes. La spécialisation d'une association correspond à l'inclusion ensembliste des liens.

[72] À ne pas confondre avec le concept `Rôle` que nous représentons dans le modèle de contexte (section 5.1.1). Un rôle ici fait référence au rôle d'un objet dans une association (dans le même sens que dans le langage UML).

Autrement dit, les liens d'une association (l'extension de l'association) appartiennent à l'ensemble des liens de sa "super-association", si elle existe. C'est pour cette raison que la spécialisation d'une association par l'ajout d'un rôle n'est pas autorisée en AROM. Ainsi, la spécialisation d'une association `A` ("super-association") dans une association `A'` ("sous-association") se fait notamment : (i) par la spécialisation de la classe correspondante d'un rôle r_i de `A` ($C'(r_i') < C(r_i)$) ; (ii) par la modification ou l'ajout d'une facette à un rôle r_i ou à une variable v_j de `A'` ; et (iii) par l'ajout d'une variable à `A` [Page 2000].

Ainsi, de la même manière que pour les classes, nous pouvons noter la notion d'association et de tuple comme suit[73] :

- l'intension (*I*) d'une association `A` dans une base de connaissances `k` est notée :

  $A_{k,I}=\{a_j : d_j\}_{j=1..m}+\{r_i : C(r_i)(m_i, M_i)\}$, où

  a_i est l'identificateur de la variable, et d_j est le domaine de valeurs acceptables pour la variable a_i.

  r_i est l'identificateur du rôle, $C(r_i)$ la classe correspondante du rôle r_i, et m_i et M_i la multiplicité minimale (par défaut $m=0$) et maximale (par défaut $M=\infty$) de r_i.

- l'extension (*E*) d'une association `A` dans une base de connaissances `k`[74] est notée :

  $A_{k,E}=\{t_i : A'\}_{i\geq 0}$, où t_i indique une instance (un tuple) de l'association `A'`, et `A'` est, soit une sous-association de `A`, soit `A` elle-même.

- Un tuple `t` est interprété comme étant :

  $t=\{a_j=v_j\}_{j=1..m}+\{r_i=o_i\}_{i\geq 2}$, où

  a_i est l'identificateur de la variable, et v_j est la valeur attribuée à la variable a_i.

  r_i est l'identificateur du rôle, o_i est l'objet de la classe correspondante $C(r_i)$ attribué au rôle r_i.

- l'appartenance d'un tuple `t` à l'extension d'une association `A` dans la base de connaissances `k` est notée par :

  $$t \in A_{k,E}$$

- la valeur v_j d'une variable a_i dans un tuple `t` est notée par :

  $$t.a_p \;\; ou \;\; v_p, \;\; pour\ un\ tuple\ t=\{a_j=v_j\}_{j\geq 0}+\{r_i=o_i\}_{i\geq 2} \;\; avec \;\; p\in[1..j]$$

- l'objet o_i qui joue le rôle r_i dans un tuple `t` est noté par :

  $$t.r_q \;\; ou \;\; o_q, \;\; pour\ un\ tuple\ t=\{a_j=v_j\}_{j\geq 0}+\{r_i=o_i\}_{i\geq 2} \;\; avec \;\; q\in[1..i]$$

- la spécialisation directe d'une association $A_I=\{a_j : d_j\}_{j=1..m}+\{r_i : C(r_i)\}$ dans une association $A'_I=\{a'_j : d'_j\}_{j=1..l}+\{r'_i : C(r'_i)\}$ de la base de connaissances `k` est notée par `A' < A`. La spécialisation directe n'est valable si :

$$(A'<A)_k \;\; est\ valable\ ssi \begin{cases} \forall i\in[1..m], \exists j\in[1..l] \;\; a_i=a_j' \;\; et \;\; d_j'\subseteq d_i, \\ \forall i\in[1..n] \;\; C'(r_i')<C(r_i), \\ \neg\exists A'' \;\; (A'<A'')_k \;\; et \;\; (A''\neq A) \end{cases}$$

[73]Cette formalisation est également inspirée de [Capponi 2005].

[74]Pour une question de simplicité, l'indication de la base de connaissances pourra être omise, puisque nous travaillons avec une seule base de connaissances.

> En d'autres termes, chaque variable définie dans la super-association `A` se trouve également dans la sous-association `A'` avec un domaine de valeurs qui est un sous-ensemble du domaine de valeurs de la même variable dans `A`. Par ailleurs, pour tous les rôles redéfinis dans la sous-association, la classe correspondante est une sous-classe de celle correspondante au même rôle dans la super-association. Et il n'existe pas une deuxième association `A''`, différente de `A`, pour laquelle l'association `A'` serait aussi une spécialisation directe (*il n'y a pas d'héritage multiples pour les associations* non plus).

Enfin, il est important d'observer que, même si AROM supporte naturellement les associations, il ne présente pas une représentation explicite pour les agrégations. Selon Page *et al.* [Page 2001], les formalismes par objets disposent de trois types de relations : les relations de spécialisation, les associations et l'agrégation. AROM support la représentation explicite du premier type (la spécialisation) et celle du deuxième, les associations, qui est d'ailleurs, une des forces de ce SRCO. En revanche, ce SRCO ne possède pas une structure qui représente la sémantique des relations d'agrégation. Celles-ci sont toujours représentées comme étant des associations de multiplicité `n` entre les classes, même si AROM tend à évoluer dans ce sens (voir, par exemple, [Gensel 2006]). Ainsi, les relations d'agrégation du modèle proposé, comme celle qui unit les classes `Description de contexte` et `Élément de contexte` (voir Figure 16), sont traduites dans une base de connaissances AROM vers des associations de multiplicité `n:n` entre les classes.

Mécanismes d'exploitation

Une base de connaissances dans AROM est constituée d'un ensemble de structures (classes et associations) et d'un ensemble d'instances de ces structures (objets et tuples). Afin de permettre l'exploitation de cette base de connaissances, plusieurs mécanismes sont possibles. Parmi les mécanismes supportés par AROM, on retrouve tout d'abord l'instanciation. L'***instanciation*** est l'action qui consiste à créer une instance [Gensel 1995]. Dans le cas du système AROM, celle-ci peut être autant l'instance d'une classe (c'est-à-dire un objet), que l'instance d'une association (un tuple).

De manière générale, pour l'instanciation d'une classe, il est nécessaire de fournir, d'un côté, la classe d'appartenance, et de l'autre, les informations destinées à déterminer cet individu et le distinguer des autres [Gensel 1995]. Ces informations correspondent, hormis un éventuel identificateur pour l'objet, aux valeurs pour les variables qui ont été définies par l'intension de la classe. Ces valeurs doivent satisfaire les conditions imposées par les facettes de chaque variable pour que l'instanciation soit réussie. Néanmoins, lors de l'instanciation, les valeurs de certains attributs[75] peuvent n'être pas renseignées. Dans ce cas, l'instance est dite incomplète.

Pour l'instanciation d'une association, le mécanisme est le même, sauf que, pour les tuples, il faut également fournir les objets qui forment le lien de l'association. Il faut informer, au moment de la création du tuple, autant d'objets que le nombre de rôles de l'association, et chaque objet doit en plus satisfaire la condition d'appartenance du rôle auquel il est affecté (l'objet o_i ne pourra jouer le rôle r_i que s'il appartient à l'extension de la classe correspondante $C_{k,E}(r_i)$), tout en respectant la multiplicité du rôle auquel il est attribué.

Une fois les instances créées, elles deviennent partie intégrante de la base de connaissances. Le système AROM permet que ces instances soient modifiées par l'altération

[75]Désormais, nous utiliserons de manière indistincte les termes *variable* et *attribut*.

des valeurs de leurs variables. Cependant, le lien d'un tuple n'est pas modifiable, car il représente l'identité même du tuple.

De plus, il est important d'observer qu'une base de connaissances n'est pas une structure figée. Outre l'instanciation, qui permet de peupler une base de connaissances, AROM permet également que les structures (classes et associations) appartenant à une telle base soient modifiées. Il est possible d'altérer l'intension d'une classe ou d'une association une fois celle-ci définie. Ceci apporte une capacité d'évolution aux bases de connaissances exprimées dans AROM. Ces bases ne sont plus une structure statique, qui n'est plus modifiable une fois créées. Elles peuvent évoluer, non seulement par l'instanciation de nouvelles instances, mais également par des changements dans les descriptions des classes et des associations (c'est-à-dire, dans la structure même de la base de connaissances).

Un autre mécanisme que le système AROM met à disposition est le ***mécanisme d'inférence d'attribut***. L'objectif de ce mécanisme est de déterminer la valeur d'un attribut de manière automatique. Ce mécanisme se concrétise dans AROM à travers les facettes d'inférence. La plus simple de ces facettes est la facette `default`, qui permet l'attribution d'une valeur par défaut si aucune valeur n'a été informée. La facette d'attachement procédural, quant à elle, permet d'associer à une variable une portion de code Java (une méthode spécifique dans une classe Java), qui calcule automatiquement la valeur de la variable [Bruley 2003].

Par ailleurs, AROM possède également un mécanisme de ***classification*** d'instances, notamment la classification d'objets [Chabalier 2004]. De manière générale, le mécanisme de classification vise à déterminer pour un concept donné sa position dans la hiérarchie. Pour une instance (objet ou tuple), ceci signifie trouver la structure (classe ou association) la plus spécifique à laquelle on peut rattacher l'instance. L'objectif ultime est l'acquisition de nouvelles connaissances. Pour une instance, la classification permet d'obtenir une caractérisation plus précise de cet individu via les variables. Les algorithmes mis en œuvre dans AROM sont récursifs. Étant donné une instance `i` initialement rattachée à une structure `S`, les différents algorithmes de classification disponibles réalisent un parcours en profondeur dans la hiérarchie de classes, chaque étape consistant à décider d'un possible rattachement de `i` à une structure `S'` plus spécialisée que `S` [Chabalier 2004]. Les algorithmes analysent notamment les valeurs attribuées aux variables dans l'instance, si elles sont conformes aux conditions imposées par la structure. Pour chaque structure disponible, ces algorithmes indiquent par une étiquette (*sûr*, *possible* ou *impossible*) si l'instance est sûre d'appartenir, si elle peut appartenir ou si elle n'est peut appartenir à l'extension de la structure. Dans le cas d'une instance incomplète, ces algorithmes ne peuvent qu'affirmer la *possibilité* de rattachement de l'instance à la structure (étiquette *possible*). Ils ne peuvent pas affirmer avec certitude (étiquette *sûre*) l'appartenance de l'instance à l'extension de la structure. Finalement, la classification d'un objet peut entraîner la classification des tuples dans lesquels l'objet participe. Il y a ici un effet de propagation de la connaissance : les connaissances acquises pour un objet (son rattachement à une classe plus spécialisée) sont transmises aux tuples liés à cet objet, qui pourront utiliser ces connaissances pour leur propre classification.

Le système AROM présente également un ***langage de modélisation algébrique*** (LMA) qui permet d'exprimer des équations impliquant objets et tuples de la base de connaissances. Le LMA peut être utilisé pour l'écriture d'équations permettant le calcul de la valeur d'une variable dans les instances, ou encore pour la réalisation des requêtes dans la base de connaissances. Les expressions LMA sont construites à partir de constantes, d'opérateurs et de fonctions (`+`, `-`, `sqrt`, `sin`, etc.), d'opérateurs itérés (`sum`, `product`, `inter`...), d'expressions quantifiées (`all`, `exist`...), de variables de classes et d'associations, et d'expressions d'accès aux tuples d'une association à partir d'un objet et vice-versa [Chabalier 2004] [Page 2000]. Ainsi, à travers ce langage, il est possible d'explorer une base de

connaissances par des expressions qui permettent tant la formulation de requêtes que la définition des facettes d'inférence.

Le LMA permet également l'utilisation d'opérateurs spatiaux et temporels. Ceux-ci ont été introduits dans le LMA par une extension du système AROM nommé AROM-ST [Moisuc 2004] [Moisuc 2005]. À travers AROM-ST, Moisuc *et al*. [Moisuc 2004] [Moisuc 2005] ont introduit dans le système AROM le support à plusieurs types spatio-temporels, et dans le LMA, une série d'opérateurs capables de manipuler ces types de données. Parmi les types géomatiques spatiaux, on retrouve les types proposés par les normes *OpenGIS*[76], tels que « *point* », « *polyline* » et « *polygone* ». Parmi les types temporels, on retrouve notamment les notions d'un instant dans le temps (« *instant* ») et d'intervalle de temps (« *interval* »). Avec ces nouveaux types, les auteurs ont proposé un ensemble d'opérateurs LMA particuliers à ces types et leurs caractéristiques. On retrouve ainsi dans l'extension AROM-ST des opérateurs topologiques qui testent la position relative de deux entités, des opérateurs ensemblistes qui traitent l'espace et le temps comme un ensemble de points et d'instants respectivement, ou encore des opérateurs de mesure qui donnent une description quantitative de l'espace et du temps. Avec ces opérateurs, il est possible, par exemple, de vérifier si deux régions dans l'espace se croisent (opérateur « *crosses* »), possèdent une intersection (« *intersects* »), ou sont disjointes (« *disjoint* »). Enfin, il convient d'observer, parmi les opérateurs topologiques, la présence des opérateurs spatiaux définis par les normes *OpenGIS*, et les opérateurs temporels traditionnels proposés par Allen [Allen 1983].

```
knowledge-base: ContextModel

class: ContextDescription
variables:
 variable: reference
  type: string

class: Activity
super-class: Element
variables:
 variable: done
  type: boolean
  default: false
 variable: details
  type: string

class: Device
super-class: Element
variables:
 variable: memory
  type: float
 variable: energy
  type: float
  domain: [0.0..100.1]
 variable: ccppProfileURL
  type: string
 variable: model
  type: string

class: Element
variables:
 variable: typeId
  type: integer
 variable: objId
  type: integer
 variable: description
  type: string
 variable: precision
  type: float
  domain: [0.0..1.01]
 variable: name
  type: string

. . .
```

```
association: description_composition
roles:
 role: element
  type: Element
 role: descriptor
  type: ContextDescription

association: object_composition
roles:
 role: component
  type: SharedObject
 role: conteiner
  type: SharedObject

association: belong
roles:
 role: team
  type: Group
 role: role
  type: Role
 role: player
  type: Member

association: contextualize
roles:
 role: descriptor
  type: ContextDescription
 role: target
  type: Element

association: allows
roles:
 role: allowed_role
  type: Role
 role: allowed_task
  type: Activity

association: perform
roles:
 role: responsible
  type: Member
 role: task
  type: Activity

. . .
```

```
instance: contexte_Alain
is-a: ContextDescription

instance: treo650
is-a: Device
 memory = 18.0
 energy = 82.0
 ccppProfileURL = "http://jangada/ccpp/treo650.xml"
 model = "Palm Treo 650"
 precision = 0.6
 name = "treo650"

instance: comité
is-a: Group
 description = "comité de programme Conf'06"
 name = "comité"

instance: Alain
is-a: Member
 age = 32
 login = "duponta"
 cellphone = "06 55 55 55 55 "
 homepage = "http://www.host.fr/~duponta/"
 email = "Alain.Dupont@host.fr"
 name = "Alain Dupont"
. . .

tuple:
is-a: description_composition
 element = Alain
 descriptor = contexte_Alain

tuple:
is-a: description_composition
 element = treo650
 descriptor = contexte_Alain

tuple:
is-a: contextualize
 descriptor = contexte_Alain
 target = Alain

. . .
```

Figure 29. Extrait d'une base de connaissances AROM.

[76] *Open Geospatial Consortium (OGC)* - http://www.opengeospatial.org/.

Par ailleurs, les bases de connaissances AROM sont exprimées dans un langage propre au système AROM. Ce langage permet la description d'une base de connaissances avec tout ses composants : classes, associations, avec leurs variables et rôles respectifs, les objets et les tuples, avec leurs valeurs définies pour les variables. La Figure 29 présente une base de connaissances dans le langage AROM. On observe qu'il s'agit d'un langage textuel, facile à comprendre pour une personne. Par ailleurs, le système AROM propose également un *parser* XML. Celui-ci permet la traduction d'une base de connaissances vers un dialecte XML défini par AROM, nommé *XAROM*. La particularité de cette traduction vers XML est que, contrairement au langage AROM, la base de connaissances est séparée en deux parties, une pour les structures (classes et associations), et une autre pour les instances (objets et tuples). L'utilisation de XML dans le système AROM implique ainsi la représentation de la base de connaissances sur deux fichiers XML distincts, un comportant les descriptions (l'intension de chaque classe et association), et un second comportant les instances (l'extension de chaque classe et association). Les figures 30 et 31 montrent la traduction de la même base présentée dans la Figure 29 vers XML : la Figure 30 montre les classes et les associations de la base de connaissances, tandis que la Figure 31 montre les instances (objets et tuples).

Figure 30. Extrait d'une base de connaissances AROM en XML : les structures.

5.2.2 Le modèle de contexte vu à travers AROM

Pour une première mise en œuvre du modèle de contexte, nous avons choisi le système AROM. Nous avons procédé à une mise en œuvre partielle, dans laquelle nous avons traduit le modèle tel qu'il est présenté dans la section 5.1.2 vers une base de connaissances AROM.

Pour cette mise en œuvre, nous avons retenu AROM pour plusieurs raisons. Tout d'abord, le fait qu'il soit un système de représentation de connaissances par objets a facilité la transformation du modèle vers une base de connaissances. Cette transformation n'a rien perdu en pouvoir de représentation car le système AROM permet la représentation des associations au même titre que celle des classes. En effet, dans le modèle proposé, les associations entre les classes ont une forte signification puisqu'elles représentent les relations qu'entretiennent

les concepts dans le monde réel. Elles font partie intégrante de la description du contexte d'utilisation, au même titre que les objets qu'elles relient.

Figure 31. Extrait d'une base de connaissances AROM en XML : les instances.

Ainsi, chaque classe et association illustrées dans les figures 16 et 23 ont été traduites vers une classe ou une association dans une base de connaissances AROM. La Figure 29 montre certains extraits de cette base de connaissances dans le format propre au système AROM. La Figure 30 et la Figure 31 illustrent la même base de connaissances, mais en format XML. La première montre les classes et les associations décrites dans modèle de contexte, tandis que la seconde présente les instances illustrées dans l'exemple de la Figure 24. On s'aperçoit que les relations de composition proposées dans le modèle ont été traduites vers des associations dont la multiplicité est `n`. Ceci est le cas notamment de la composition liant la classe `Description_de_Contexte` à la classe `Élément_de_Contexte`, qui a été traduit vers l'association `description_composition` (voir Figure 30). La raison de cela est l'inexistence d'une sémantique propre à ce type de relation dans le système AROM et, de manière plus générale, dans les SRCO. En revanche, toutes les autres associations proposées dans le modèle de contexte ont pu être directement traduites vers des associations dans le système AROM.

D'autre part, le système AROM présente d'autres avantages qui nous ont conduit à le choisir. Un de ces avantages consiste à permettre, à travers notamment son API Java, la modification de la base de connaissances lors de l'exécution, aussi bien à travers l'introduction et la modification des instances (les objets et les tuples), qu'à travers la création et la modification du schéma de la base de connaissance (les classes et les associations). Le système AROM permet également la gestion des attributs non évalués. Ceci nous permet de respecter les critères que nous avons énoncés (cf. section 5.1.3), notamment ceux concernant l'évolution du modèle et le caractère dynamique du contexte.

Par exemple, si on considère l'utilisateur 'Alain', dont le contexte d'utilisation, dans un premier moment, est présenté dans la Figure 22, les changements détectés par le processus d'acquisition son retransmis à la base de connaissances. Ainsi, le déplacement d'Alain illustré par la Figure 24 se traduit par l'élimination des instances (objets et tuples) relatifs à son ancienne localisation et le dispositif qui était utilisé avant (voir les figures 22 et 24), et par la création d'un nouveau objet de la classe `Dispositif` (objet `treo650`) et des nouveaux tuples

reliant cet objet aux objets `session_Alain, répertoire_partagé` et `contexte_Alain` (tuples, respectivement, des associations `espace_d'exécution, conçu_pour,` et de la composition du contexte). Un autre cas serait la détection d'un changement au niveau de la batterie de son dispositif. Dans ce cas, les changements se situent au niveau des attributs de l'objet `treo650` dans la base de connaissances, dont les valeurs seront modifiées afin de refléter la nouvelle situation d'Alain.

Un autre avantage de l'utilisation du système AROM est la possibilité de traduire automatiquement la base de connaissances en XML. L'utilisation de XML constitue un atout non négligeable pour les systèmes sur le Web, puisque ce format facilite la transmission et le traitement des informations. Par ailleurs, les collecticiels sur le Web sont souvent des systèmes partagés entre plusieurs sites, et dans ce cas, la communication entre les différents sites est primordiale pour la performance générale du système. Il est également important de considérer une éventuelle communication avec d'autres systèmes qui utilisent le modèle de contexte. Dans ce cas, l'utilisation de XML devient primordiale, puisque ce format présente de sérieux avantages dans le traitement des informations. En somme, cette possibilité de générer une version XML d'une base de connaissances AROM permet d'allier les avantages de l'utilisation d'un SRCO à ceux de l'utilisation du standard XML.

De manière plus formelle, nous pouvons analyser le modèle de contexte à travers les mêmes notations que nous avons présenté dans la section précédente. Ainsi, conformément à la section 5.1.2, nous considérons le contexte à partir de la description de contexte. Une description de contexte est, pour nous, un objet de la classe homonyme :

$d \in DC_E$, où `d` est un objet de l'extension de la classe `Description_de_Contexte (DC)`

Une `Description_de_Contexte` est définie comme une composition d'éléments de contexte. Ceux-ci sont des objets de la classe `Élément_de_Contexte` ou des sous-classes de celle-ci. Ils appartiennent donc à son extension :

$O = \{o_i \mid o_i \in EC_E\}_{i \geq 1}$, où `oi` est un objet qui appartient à l'extension d'`Élément de contexte (EC).`

Les éléments de contexte (objets `oi`) sont liés à l'objet `d` à travers les tuples de l'association `description_composition (ADC)` ayant d et oi jouant de rôles.

$\mathcal{E} = \{o \mid (o \in O) \wedge \exists t \in ADC_E \ (t.r_i = o \wedge t.r_j = d)_{i \neq j}\}$, où `o` est un objet appartenant à l'extension d'`Élément_de_Contexte (EC)`, `d` est un objet de la classe `Description_de_Contexte`, et `t` un tuple de l'association `description_composition (ADC)` qui contient `o` et `d` jouant des rôles distincts.

Nous avons également affirmé que ces éléments qui composent le contexte (les objets `oi` en $\mathcal{E}$) ne sont pas isolés, ils entretiennent des relations avec d'autres objets, et ces relations participent aussi à la notion de contexte :

$\mathcal{T} = \{t \in A_E \mid (\forall r_j \in A_I)(t.r_j = o \wedge o \in \mathcal{E})\}_{i \geq 1}$, où t est un tuple d'une association `A` qui est formé par des objets `o` de la classe `Éléments_de_Contexte` lié à l'objet `Description_de_Contexte d.`

Nous définissons le contexte d'utilisation donc comme étant l'ensemble d'éléments de contexte liés à une description et les tuples qui les reliant :

$C_o = (d \cup \mathcal{E}) + \mathcal{T}$ soit `Co` le contexte d'utilisation relatif à un élément `o`, `Co` est constitué à la fois de l'objet `Description_de_Contexte`

d et des objets qui la composent, et des tuples reliant ces objets entre eux.

5.3 Conclusions

Dans ce chapitre[77], nous avons proposé un modèle de contexte qui formalise la notion de contexte d'utilisation à travers une représentation par objets. Cette représentation se démarque des autres représentations par la prise en compte autant des aspects physiques liés au contexte (qui sont normalement considérés par les systèmes sensibles au contexte), que des aspects collaboratifs, tel que les rôles joués par un utilisateur dans une équipe. Ces aspects collaboratifs sont nécessaires pour la représentation du contexte d'utilisation dans un collecticiel. Par ailleurs, la notion de « contexte collaboratif » peut être également pertinente pour d'autres systèmes sensibles au contexte, car l'utilisateur dans un tel système peut être aussi engagé dans plusieurs activités de coopération ou participer à un environnement coopératif, même si le système, en soi, ne supporte pas directement cette coopération.

Enfin, le modèle que nous proposons satisfait les critères d'être évolutif, de permettre la représentation des informations au caractère dynamique, de l'expression d'informations incomplètes et de critères de précision. Le respect à ces critères constitue une avancé vis-à-vis des autres représentations, surtout en ce qui concerne l'expression d'informations incomplètes.

Dans les prochains chapitres nous exploitons ce modèle de contexte, en définissant, tout d'abord, une série d'opérations pour sa manipulation, et ensuite, en l'utilisant pour l'adaptation de l'information de conscience de groupe au sein d'un collecticiel sur le Web.

[77]Les sujets abordés dans ce chapitre ont été traités par les publications [4], [5], [6], [7] et [8] indiquées dans l'Annexe IV.

6 Opérations sur le Modèle de Contexte

Dans le chapitre précédent, nous avons proposé un modèle de contexte décrit dans une représentation par objets. Dans le présent chapitre, nous définissons un ensemble d'opérations de base capables de comparer les instances de ce modèle. À travers ces opérations nous voulons être capables de déterminer si un certain élément appartient à une *description de contexte* (par exemple, pouvoir affirmer qu'un utilisateur emploie un certain dispositif par l'analyse de son contexte courant), ou encore si un contexte décrit par une *description de contexte* est un sous-contexte d'un autre (afin de vérifier, par exemple, si un utilisateur se trouve dans une situation particulière). Ainsi, nous proposons un ensemble d'opérations qui gardent la même généralité du modèle de contexte. Ceci est important car le modèle proposé se présente comme un modèle conceptuel qui doit être appliqué à un système en particulier (voir section 5.1). Par conséquent, les opérations destinées à manipuler et à exploiter les instances de ce modèle se doivent aussi génériques que le modèle lui-même.

Nous proposons trois types d'opération de base : (*i*) un premier type d'opération destiné à comparer le contenu de deux objets ou de deux tuples de manière isolée ; (*ii*) un deuxième type qui effectue une comparaison en tenant compte du contenu des instances et de leurs relations avec d'autres instances de la base de connaissances ; et (*iii*) des mesures qui évaluent la similarité entre le contenu de deux instances de manière isolée ou en tenant compte de leurs relations. Le premier type d'opération vérifie l'existence d'une *relation d'égalité* entre deux instances, si elles ont un contenu équivalent. Le deuxième type vérifie une possible *relation d'inclusion* entre deux objets et l'ensemble formé par les tuples et les objets qui lient à chacun de ces objets dans une base de connaissances. Autrement dit, la relation d'inclusion vérifie si un ensemble contient tous les éléments de l'autre. Pour sa part, le troisième type d'opération décrit une *relation de similarité* entre deux instances (isolées ou avec les objets et tuples en relation avec lui), en mesurant le degré de correspondance entre ces instances. Nous nous positionnons dans le cadre d'une représentation de connaissances par objets, à travers les concepts et les notations définies dans AROM (cf. section 5.2). Les opérations sont donc proposées sous l'angle de la représentation de connaissances par objets, et plus particulièrement, à travers celle du modèle AROM. Par conséquent, les opérations que nous proposons peuvent prendre en considération tous les aspects du modèle de contexte, non seulement les classes qui le composent, mais également les associations.

Ce chapitre s'organise en trois parties : la section 6.1 présente la *relation d'égalité* et l'opérateur *Equals* qui l'implémente ; la section 6.2 aborde la *relation d'inclusion*, avec l'opérateur *Contains* qui lui correspond ; la section 6.3 discute la *relation de similarité*, et les opérateurs que nous proposons pour l'implémenter (*SimO*, *SimT* et *Sim*). Enfin, la section 6.4 présente nos conclusions sur ce chapitre.

6.1 Relation d'Egalité – Opérateur *Equals*

6.1.1 Définition générale

L'objectif général de la relation d'égalité est de vérifier si le contenu deux instances est le même. Pour déterminer l'existence de cette relation entre deux instances, nous proposons un opérateur de comparaison, l'opérateur *equals,* dont l'objectif est de comparer le contenu de

deux instances données (soit deux objets, soit deux tuples) d'une base des connaissances. L'opérateur *equal* compare localement les instances de manière isolée, sans considérer leurs relations avec d'autres instances de la même base de connaissances.

Sémantiquement, nous considérons deux objets comme égaux (noté comme $o \; equals_c \; o'$) s'ils contiennent des valeurs égales pour leurs attributs. Pour les tuples, la définition est similaire, mais nous lui ajoutons l'égalité entre les liens qui déterminent les tuples, c'est-à-dire, l'égalité entre les objets impliqués dans les rôles de chaque tuple. Ainsi, deux tuples sont dits égaux (noté aussi comme $t \; equals_A \; t'$) : (*i*) s'ils contiennent des valeurs égales pour leurs variables ; et (*ii*) si les objets valeurs rôle dans chaque tuple sont égaux.

Puisque dans système AROM un objet ou un tuple ne comporte pas d'objets valeur d'un attribut, l'égalité entre les attributs correspond à l'identité entre leurs valeurs. Nous représentons l'identité entre deux valeurs par une ***fonction d'identité*** *fi (v, v')*, dont la signification concrète varie en fonction du type des valeurs comparées. Ce type de fonction est également suggéré par d'autres auteurs dans la littérature, dont Valtchev [Valtchev 1999]. Cet auteur défend l'utilisation des fonctions δ_i capables de calculer les différences élémentaires pour chaque attribut a_i. Les résultats de ces fonctions peuvent ensuite être combinés dans une seule valeur par une fonction globale, que Valtchev [Valtchev 1999] définit comme $d=Aggr(\delta_i)$. Ainsi, à l'instar de cet auteur, nous définissons une fonction d'identité *fi (v, v')* qui compare la valeur de chaque attribut, en respectant son type. En d'autres termes, étant donné deux valeurs `v` et `v'` attribuées à une variable `a : d` définie par une structure `S`, *fi* `(v, v')` dépend de `d`. Ainsi, nous définissons *fi (v, v')* comme suit :

$$fi(v,v') = \begin{cases} 1 & si\ les\ valeurs\ v\ et\ v'\ sont\ identiques \\ 0 & dans\ le\ cas\ contraire \end{cases}$$

- Pour les types simples (entier, réel, booléen et chaîne de caractères), nous pouvons affirmer que :

$$fi(v,v')=1 \;\rightarrow\; v=v' \;,$$

- Pour les types multivalués dont l'ordre n'est pas significatif :

si `v` et `v'` sont des ensembles, et `v.i` indique l'élément occupant la position `i` dans `v`, alors

$$fi(v,v')=1 \;\rightarrow\; \forall i \exists j (v.i=v'.j) \;\wedge\; \forall j \exists i (v'.j=v.i)$$

- Pour les types multivalués dont l'ordre est significatif :

si `v` et `v'` sont des listes, et `v.i` indique l'élément occupant la position `i` dans `v`, alors

$$fi(v,v')=1 \;\rightarrow\; |v|=|v'| \;\wedge\; \forall i \in [1..|v|] \; (v.i=v'.i)$$

Par ailleurs, la fonction d'identité ***fi (v,v')*** doit assurer la validité d'au moins deux propriétés de base : d'abord que la comparaison d'une valeur avec elle même soit toujours `1` (*fi (v, v)* = `1`), et ensuite que la fonction d'identité soit communative (c'est-à-dire, *fi (v, v')* = *fi (v',v)*).

Une fois définie la fonction d'identité *fi (v,v')* pour les attributs, nous pouvons définir l'opérateur *equals*. De manière générale, nous considérons que :

- Un objet o est considéré comme égal à un autre objet o' par rapport à une classe C ($o\ equals_C\ o' = vrai$) si et seulement s'ils appartiennent à l'extension de la classe C, et s'ils contiennent les mêmes valeurs pour les mêmes variables ;

- Un tuple t est égal à un tuple t' par rapport à une association A ($t\ equals_A\ t' = vrai$) si et seulement si tous les deux appartiennent à l'extension de l'association A, s'ils contiennent des valeurs identiques pour les mêmes variables, et si l'objet valeur de chaque rôle r_i en t est égal à l'objet valeur de rôle r_i en t'. Par exemple, si A est une association binaire, le tuple t connecte les objets o_1 et o_2 et le tuple t' connecte les objets o'_1 et o'_2, alors pour que $t\ equals_A\ t'$ soit vrai, il faut que $o_1\ equals_{C(r_1)}\ o'_1$ et $o_2\ equals_{C(r_2)}\ o'_2$ le soient également.

L'opérateur *equals* compare les instances par rapport à une structure de référence donnée (une classe C, ou une association A, selon le cas). Les instances doivent appartenir à l'extension de cette structure. Par conséquent, elles sont comparées en tenant compte des attributs de l'intension de cette structure, que les deux instances ont en commun. Néanmoins, ces instances peuvent être incomplètes, et donc ne pas avoir de valeurs attribuées pour tous les attributs définis par la structure qui sert de référence pour la comparaison. Par conséquent, ce ne sont que les attributs définis dans les deux instances comparées qui sont pris en compte. Les attributs dont la valeur est inconnue dans l'une ou l'autre instance sont ignorés.

Par ailleurs, il faut toujours considérer que, même si on compare les deux instances (les deux objets ou les deux tuples) par rapport à une structure de référence (une classe C ou une association A), dont les instances appartiennent à l'extension, ceci ne signifie pas qu'elles sont tous les deux rattachées à celle-ci. Par exemple, dans la comparaison de deux objets, il se peut que chaque objet soit une instance d'une sous-classe différente de C. Pour illustrer cette question, la Figure 32 présente deux objets, o et o', qui appartiennent à l'extension d'une classe C, qui définit trois attributs ($C_I = \{a_1:d_1,\ a_2:d_2,\ a_3:d_3\}$). Le premier objet contient les valeurs pour tous les attributs définis par C tandis que l'objet o' ne contient pas de valeur pour l'attribut a_3. De plus, l'objet o est attaché à la classe C', sous-classe de C qui définit un attribut a_4 (donc $o = \{a_1=0,\ a_2=1,\ a_3=2,\ a_4=3\}$), tandis que l'objet o' est attaché à la classe C'', également sous-classe C et qui définit aussi un nouveau attribut a_5 (donc $o' = \{a_1=0,\ a_2=1,\ a_5=6\}$). Dans ce cas, même si o contient une valeur pour a_4 et o' une valeur pour a_5 , celles-ci ne sont pas prises en compte par l'opérateur *equals*, car ces attributs n'appartiennent pas à l'intension de la classe C, qui est utilisée comme base pour la comparaison. Dans ce cas de figure, seulement les valeurs des attributs a_1 et a_2 sont utilisées par l'opérateur *equals*, puisque ces attributs appartiennent à l'intension de la classe C (ils sont donc communs aux deux instances) et ont une valeur définie dans chacune.

En ce qui concerne les tuples, l'opérateur *equals* garde le même comportement. Les tuples sont comparés par rapport à une association A donnée. Par conséquent, l'opérateur ne considère que les variables appartenant à l'intension de l'association A et dont la valeur est connue dans les deux tuples. Ainsi, si l'intension de l'association A inclut m variables, mais les tuples n'ont de valeur connue que pour p variables ($p \leq m$), alors l'opérateur *equals* ne tient compte que des p variables dont la valeur est connue dans les tuples. En ce qui concerne les rôles qui définissent chaque tuple, la définition de l'opérateur *equals* considère un nombre précis de rôles, celui de l'association A. Nous assumons que la spécialisation d'une association ne change pas l'arité de l'association. Il est interdit d'ajouter un nouveau rôle à l'association lors d'une spécialisation, car une telle modification casserait la relation d'ensemble qui existe entre les associations (un tuple de la sous-association ne pourrait plus être considéré comme un tuple de l'association mère). Celui-ci est le comportement affiché du modèle AROM,

lequel ne permet pas que le nombre de rôles qui caractérisent une association soit changé lors que l'association est spécialisée (cf. section 5.2.1). Par ailleurs, le modèle AROM interdit la création d'un tuple sans que son lien soit complet, c'est-à-dire, sans qu'un objet soit attribué à chaque rôle (voir section 5.2.1). Par conséquent, l'opérateur *equals* compare, pour les tuples, tous les rôles déterminés par l'association de référence `A`.

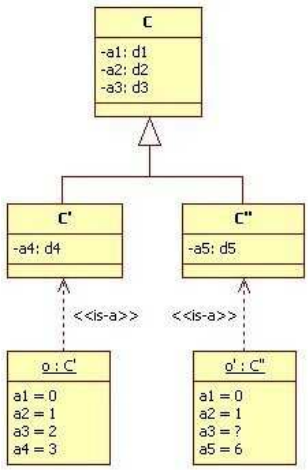


Figure 32. Exemple de deux objets qui peuvent être comparés par l'opérateur *equals*.

Ainsi, nous pouvons formaliser cette définition générale, à travers les notations propres au modèle AROM de la manière suivante (l'Annexe I présente une vision algorithmique de cette définition) :

- soit *fd(o)* la cardinalité de l'ensemble formé par les variables dont la valeur est connue dans un objet `o`

$$fd(o)=\{a_i \mid o.a_i \neq null\}$$

- étant donné deux objets `o` et `o'`, appartenant à l'extension d'une classe `C` donnée

$$o, o' \in C_E,\ o=\{a_i=v_i\}_{1\leqslant i\leqslant p}\ et\ o'=\{a_i=v_i'\}_{1\leqslant i\leqslant q},\ C_I=\{a_i:d_i\}_{i=1..n},\ (p,q)\leqslant n$$

- `o` *equals*$_C$ `o'` si et seulement si tous les variables communes à `o` et `o'` contiennent des valeurs identiques

$$\forall i \in (fd(o)\cap fd(o'))\ \ (fi(o.a_i, o'.a_i)=1)$$

En ce qui concerne les tuples, la définition est la même, mais elle inclut également la comparaison des objets qui forment le lien de chaque tuple. Ainsi :

- soit *fd(t)* la cardinalité de l'ensemble formé par les variables dont la valeur est connue dans un tuple `t`

$$fd(t)=\{a_j \mid o.a_j \neq null\}$$

- étant donné deux tuples `t` et `t'`, appartenant à l'extension d'une association `A` donnée

$$t, t' \in A_E,\ \ A_I=\{a_j:d_j\}_{i=1..m}+\{r_i:C(r_i)\}_{i=1..n}$$
$$t=\{a_j=v_j\}_{0\leqslant j\leqslant p}+\{r_i=o_i\}_{i\geqslant 2}\ \ et\ \ t'=\{a_j=v_j\}_{0\leqslant j\leqslant q}+\{r_i=o_i'\}_{i\geqslant 2},\ \ (p,q)\leqslant m$$

- `t` *equals*$_A$ `t'` si et seulement :

- si tous les variables communes à t et à t' contiennent des valeurs identiques

$$\forall j \in (fd(t) \cap fd(t'))\ (fi(t.a_j, t'.a_j) = 1)$$

- si pour chaque objet o_i valeur du rôle r_i dans t, l'objet o_i' valeur du même rôle r_i en t' soit égal à o_i

$$(\forall i \in [1..n])(t.r_i\ equals_{C(r_i)}\ t'.r_i)$$

Le comportement de l'opérateur *equals* par rapport aux variables peut paraître conceptuellement différent de celui qu'on attend habituellement d'une relation d'égalité. Cependant, ce comportement est moins restrictif, car il ne considère que les connaissances dont on dispose réellement (on n'utilise que les attributs qui ont une valeur dans chaque instance) et il compare les instances par rapport à une référence commune donnée (la classe `C` ou l'association `A` dont les deux instances appartiennent à l'extension). Cette caractéristique de l'opérateur *equals* devient intéressante lorsqu'on considère les instances du modèle de contexte. Par exemple, supposons que, pour un collecticiel d'édition collaborative, les concepteurs aient défini une sous-classe de `Rôle` (voir section 5.1.2) nommée `Participant`, dont l'attribut `niveau` a une valeur précise (`niveau : entier = 10000`). Dans ce cas, nous pouvons imaginer qu'il soit nécessaire de comparer un objet de la classe `Participant` qui se trouve stocké dans la base de connaissances (objet `réviseur` dans la Figure 33), et un deuxième objet, qui vient d'être créé suite au processus d'acquisition de contexte (objet `o` dans la Figure 33). L'opérateur *equals*, de la manière dont il a été défini ici, nous permet d'affirmer que ces deux objets sont égaux par rapport à la classe `Rôle` (`o` $equals_{\texttt{Rôle}}$ `réviseur`). Cependant, ceci est possible parce que cet opérateur compare ces objets à partir d'une classe donnée (la classe `Rôle`), commune à tous les deux, et ne considère que les attributs dont la valeur est connue dans les deux instances (variables `niveau` et `nom`).

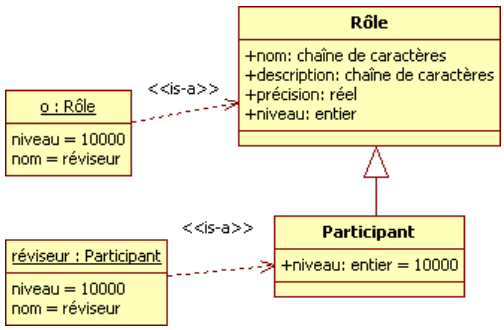


Figure 33. Exemple de deux objets du modèle de contexte comparés à l'aide de l'opérateur *equals*.

L'opérateur *equals*, tel qu'il a été défini ci-dessus, présente encore d'importantes restrictions qui modèrent ses atouts lors de son application au modèle de contexte. Ces restrictions concernent le fait que cet opérateur prend en compte *tous* les attributs qui ont une valeur attribuée dans les instances, sans distinction. Ceci peut s'avérer très contraignant lorsque la signification de certains attributs rend difficile l'égalité entre deux valeurs. Un exemple est l'attribut `précision`, défini par la super-classe `Élément_de_Contexte` dans notre modèle (cf. section 5.1.2, Figure 16). Cet attribut sert à indiquer le degré de précision des informations contenues dans un objet. Compte tenu des variations possibles dans le processus d'acquisition de contexte, tout particulièrement en ce qui concerne le contexte physique (voir Figure 21), nous pouvons supposer que la valeur de l'attribut `précision` pourra être distincte

même pour deux objets représentant un même objet réel. Un autre exemple est les attributs `mémoire_disponible` et `niveau_d'énergie` définis par la classe `Dispositif` (cf. section 5.1.2, Figure 17). Les valeurs de ces attributs correspondent à la disponibilité de mémoire et d'énergie dans un dispositif à un moment donné. Ces valeurs varient donc en fonction du temps et de l'utilisation du dispositif, et, en conséquence, deux objets représentant le même dispositif pourront avoir différentes valeurs pour ces attributs. La Figure 34 illustre cette situation. Dans cette figure, on observe deux objets de la classe `Dispositif` (`dispositif1` et `dispositif2`) qui représentent le même dispositif réel (un téléphone cellulaire modèle « *Treo 650* ») en deux situations distinctes. Dans ces deux objets, nous trouvons des valeurs différentes pour les attributs `précision` et `mémoire_disponible` (respectivement, `1` et `32` pour le premier objet, et `0,5` et `16` pour le second), ce qui rend l'opérateur `dispositif1` $equals_{dispositif}$ `dispositif2` faux.

Figure 34. Exemple de deux objets représentant un même dispositif.

Sur cet exemple, on peut s'apercevoir que, dans certaines situations, il serait plus intéressant de comparer seulement certaines variables, et non plus toutes celles définies par l'intension de la structure de référence. Autrement dit, il serait souhaitable que certaines variables soient considérées comme plus significatives que d'autres, voir même d'ignorer certaines variables. Par exemple, nous pouvons imaginer que, dans le modèle de contexte, la variable `précision`, définie par la classe `Élément_de_Contexte`, soit moins significative que la variable `nom`, définie dans la même classe, puisque la première a un contenu moins stable pour permettre une comparaison fiable avec l'opérateur *equals*. Cet opérateur cherche à identifier une relation d'égalité entre deux instances, et cette instabilité sur le contenu de la variable `précision` rend difficile cette comparaison. Ainsi, nous déclinons la définition générale de l'opérateur *equals* dans une deuxième version, laquelle associe un poids à chaque attribut de la structure de référence. Nous présentons cette seconde version de l'opérateur *equals* dans la prochaine section.

En observant l'exemple de la Figure 34, on peut imaginer la définition d'une version de l'opérateur dédiée uniquement aux instances de la classe `Dispositif`. Une telle version serait capable de prendre en compte les particularités de chaque attribut défini pour cette classe (par exemple, la variabilité de l'attribut `mémoire_disponible`). Ceci correspondrait à une approche dans laquelle on adapte l'opérateur *equals* pour chaque classe du modèle de contexte. Cependant, celle-ci dépend du système dans lequel le modèle est appliqué. Comme nous avons affirmé dans la section 5.1.4, pendant sa mise en œuvre, les concepteurs adaptent et spécialisent le modèle selon les besoins du système en construction. Dans ce cas, certaines questions se posent : comment ferait-on pour les sous-classes ? Garderont-elles la même sémantique pour les attributs ? Les nouveaux attributs, seront-ils pris en compte ? Face à ces questions, nous pouvons observer que la définition d'une version de l'opérateur *equals* propre à chaque classe et association du modèle de contexte risque de ne pas satisfaire les exigences de tous les systèmes sensibles au contexte qui appliquent le modèle, et ainsi de nuire à la généralité de notre proposition.

D'un point vue UML, nous pouvons exprimer cette question de la manière suivante (Figure 35) : l'opérateur *equals*, ainsi comme les autres opérations que nous proposons dans ce travail, est vu comme une opération définie dans les classes `Description_de_Contexte` et `Élément_de_Contexte`, et qui peut être redéfinie dans les sous-classes. La Figure 35 présente cette vision, en prenant comme exemple les classes `Dispositif`, `Localisation` et `Rôle`, lesquelles pourraient bénéficier d'une telle redéfinition. Ainsi, nous avons choisi une approche aussi générale que celle adoptée pour le modèle de contexte en proposant des opérations générales qui sont applicables à toutes les classes et les associations du modèle de contexte, et même à d'autres modèles qui utilisent une représentation par objets. Dans la section suivante, nous discutons la seconde version de l'opérateur *equals*, en présentant la formalisation (en utilisant toujours les concepts et la notation AROM) qui lui correspond.

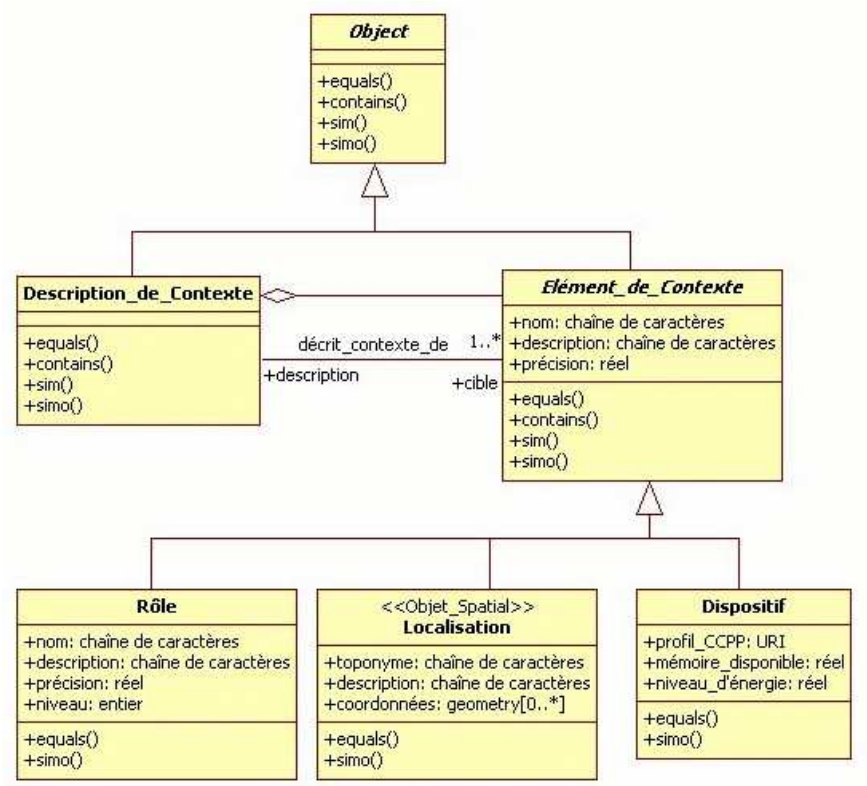


Figure 35. Les opérations proposées vues dans le modèle UML.

6.1.2 Définition de l'opérateur Equals par poids

En étudiant la littérature sur la représentation de connaissances par objets, nous pouvons constater une certaine similitude entre l'opérateur *equals* et les mesures de similarité entre objets, comme celles proposées par Bisson [Bisson 1995], Valtchev [Valtchev 1999], par Valtchev et Euzenat [Valtchev 1997]. L'opérateur *equals* cherche à identifier l'existence d'une relation d'égalité entre deux instances (il s'agit d'un opérateur de comparaison, dont la réponse est seulement vrai ou faux), tandis que les mesures de similarité proposées par ces auteurs (de même pour les mesures que nous proposons dans la section 6.3) cherchent à quantifier une distance sémantique entre les instances (ces mesures fournissent une valeur réel le entre `0` et `1` qui représente la distance entre les instances). Dans les mesures de similarité proposées par les auteurs cités précédemment, nous trouvons l'attribution d'un poids à chaque

attribut défini dans une classe[78]. Bisson [Bisson 1995], par exemple, associe un poids numérique (un numéro positif) à chaque attribut de manière à exprimer la pertinence dont celui-ci dispose dans la classe. L'objectif affiché est donc de hiérarchiser les attributs en donnant plus de poids à ceux qui sont sémantiquement plus significatifs.

Dans cette version de l'opérateur *equals*, nous adoptons cette notion de poids par attribut afin de pouvoir différencier les attributs les plus significatifs de ceux dont l'importance est moindre lors de la recherche d'une relation d'égalité. Ainsi, nous considérons qu'un poids est donné à chaque attribut de l'intension d'une classe ou d'une association. La seule condition imposée est que, à l'instar des autres auteurs ([Bisson 1995], [Valtchev 1999] et [Valtchev 1997]), la somme de tous les poids soit égale à 1. De cette manière :

- étant donné l'intension d'une classe $C_I = \{a_i : d_i\}_{i=1..n}$, ou l'intension d'une association $A_I = \{a_i : d_i\}_{i=1..n} + \{r_j : C(r_j)\}_{j=2..m}$, à chaque variable a_i est associé un poids w_i tel que $\sum_{i=1}^{n} w_i = 1$.

Toutefois, il est important d'observer que ces poids sont attribués au niveau de la structure d'appartenance (classe ou association), et non au niveau de l'instance (objet et tuple). Par conséquent, deux instances appartenant à l'extension d'une même structure pourront être comparées en fonction d'un même système de poids. Néanmoins, puisque ce système d'attribution de poids est lié aux structures (aux classes et aux associations), il est donc étranger à la situation des instances incomplètes. Dans ce cas, les instances n'ont pas de valeur attribuée à tous leurs attributs, et, par conséquent, la somme des poids maximale peut être inférieure à 1. Il est donc nécessaire de tenir compte de cette situation au moment de la définition de l'opérateur *equals*.

Une fois les poids attribués, nous pouvons redéfinir l'opérateur *equals* afin que les attributs soient évalués en fonction de leur poids, et non plus indistinctement. L'idée est de combiner le résultat de la fonction d'identité $fi(v, v')$ de chaque attribut en commun avec le poids qui lui est attribué, et de comparer ceci avec une limite $\mathcal{L}$, définie comme un numéro réel entre 0 et 1. Cette limite fixe un seuil au-delà duquel deux instances sont considérées comme suffisamment proches pour qu'elles soient considérées comme égales. Ainsi, nous définissons une nouvelle version de l'opérateur *equals* par poids comme suit :

- en ce qui concerne les objets :
  - soit *fd(o)* la cardinalité de l'ensemble formé par les variables dont la valeur est connue dans un objet o

    $$fd(o) = \{a_i \mid o.a_i \neq null\}$$

  - étant donné deux objets o et o', et une classe de référence C

    $$o, o' \in C_E,\ C_I = \{a_i : d_i\}_{i=1..n}$$
    $$o = \{a_i = v_i\}_{1 \leq i \leq p},\ o' = \{a_i = v_i'\}_{1 \leq i \leq q},\ (p, q) \leq n$$

  - étant donné un poids w_i donné à chaque variable a_i de la classe C

    $$\sum_{i=1}^{n} w_i = 1$$

[78]Les travaux précédemment cités ([Bisson 1995], [Valtchev 1997] et [Valtchev 1999]), ainsi que la majorité de travaux sur la représentation de connaissances par objets, ne considèrent pas les associations comme une structure de représentation co-existant avec les classes.

- soit une limite $\mathcal{L}$ définie comme un nombre réel entre 0 et 1

  $\mathcal{L} \in \mathbb{R}^+, \mathcal{L} \in [0,1]$

- o *equals*$_C$ o' si et seulement si :

  $$\sum_{i}^{fd(o) \cap fd(o')} w_i \times fi(o.a_i, o'.a_i) \geq \mathcal{L}' \text{ , où } \quad \mathcal{L}' = \mathcal{L} - \mathcal{L} \times \left(1 - \sum_{i}^{fd(o) \cap fd(o')} w_i\right)$$

- en ce qui concerne les tuples :

  - soit *fd(t)* la cardinalité de l'ensemble formé par les variables dont la valeur est connue dans un tuple t

    $$fd(t) = \{a_j \mid t.a_j \neq null\}$$

  - étant donné deux tuples t et t', et une association de référence A

    $$t, t' \in A_E,\ A_I = \{a_j : d_j\}_{j=1..m} + \{r_i : C(r_i)\}_{i=1..n}$$
    $$t = \{a_j = v_j\}_{0 \leq j \leq p} + \{r_i = o_i\}_{i=1..n},\quad t' = \{a_j = v_j'\}_{0 \leq j \leq q} + \{r_i : o_i'\}_{i=1..n},\ (p,q) \leq m$$

  - étant donné un poids w_j donné à chaque variable a_j de l'association A

    $$\sum_{j=1}^{n} w_j = 1$$

  - soit une limite $\mathcal{L}$ définie comme un numéro réel entre 0 et 1

    $\mathcal{L} \in \mathbb{R}^+, \mathcal{L} \in [0,1]$

  - t *equals*$_A$ t' si et seulement si

    $$\left(\sum_{j}^{fd(o) \cap fd(o')} w_j \times fi(t.a_j, t'.a_j) \geq \mathcal{L}'\right) \wedge \left(\forall i \in [1..n] (t.r_i \; equals_{C(r_i)} \; t'.r_i)\right) \text{ , où}$$
    $$\mathcal{L}' = \mathcal{L} - \mathcal{L} \times \left(1 - \sum_{j}^{fd(o) \cap fd(o')} w_j\right)$$

Nous pouvons observer que, dans la définition ci-dessus, la limite $\mathcal{L}$ est corrigée en fonction du poids des attributs absents. Ceci est nécessaire pour les instances incomplètes, qui sont pénalisées. Elles ne peuvent pas de fait atteindre la limite $\mathcal{L}$. Pour illustrer ce problème, prenons les deux objets présentés dans la Figure 32 (page 114). Dans cette figure, nous avons deux objets, dont un incomplet ($o = \{a_1=0, a_2=1, a_3=2, a_4=3\}$ et $o' = \{a_1=0, a_2=1, a_5=6\}$). Ces objets sont comparés par rapport à la classe C, dont l'intension définit trois variables ($C_I = \{a_1:d_1, a_2:d_2, a_3:d_3\}$). Supposons maintenant qu'on a défini pour les variables de cette classe les poids suivants $\{w_1=0,4, w_2=0,2, w_3=0,4\}$, et qu'on a fixé une limite $\mathcal{L}=0,8$. Étant donné que l'objet o' ne contient pas de valeur pour la variable a_3, le poids concernant cette variable ne lui sera pas accordé. Ainsi, pour les objets o et o', $\sum_{i}^{fd(o) \cap fd(o')} w_i \times fi(o.a_i, o'.a_i)$ sera, au maximum, égal à $0,6$, c'est-à-dire inférieur à la limite $\mathcal{L}$, même si, pour ces instances, tous les variables en commun dont la valeur est connue sont identiques. Dans ce cas, si on ne procède pas à une correction de la limite $\mathcal{L}$ en fonction des variables qui seront réellement utilisées par l'opérateur *equals*, ces deux objets seront considérés comme distincts (o *equals*$_C$ o' vaut `faux`). En revanche, si on considère la limite corrigée $\mathcal{L}'$($\mathcal{L}' = \mathcal{L} - \mathcal{L}$ x (1 - 0,6) = 0,8 - 0,8 x 0,4 = 0,48), ces objets sont considérés comme égaux, puisqu'on ne tient compte que des variables dont la valeur est connue.

Par ailleurs, il est important d'observer que la formule de correction que nous proposons pour $\mathcal{L}'$ n'est pas la seule possible. D'autres corrections sont envisageables. Par exemple, nous pouvons considérer une version plus simples $\mathcal{L}' = \mathcal{L} - \left(1 - \sum_{j}^{fd(o) \cap fd(o')} w_j\right)$, qui donne des résultats similaires à celle proposée. Ce qu'il faut retenir ici est la nécessité de pondérer les attributs dynamiquement, en fonction de leurs valeurs connues.

En ce qui concerne les tuples, il est important de noter le respect de l'identité du tuple. Ainsi comme la version générale, cette version de l'opérateur *equals* tient compte de tous les rôles qui forment le tuple en comparant tous les objets qui jouent ces rôles. Pour que deux tuples soient considérés comme égaux, tous les objets jouant des rôles équivalents doivent être égaux (selon la définition du même opérateur). Nous avons choisi cette approche car ce qui caractérise un tuple est justement le lien formé par les objets jouant ces rôles. Par exemple, un tuple $t=\{a_1=3,\ a_2=4,\ a_5=6\}+\{r_1=o_1,\ r_2=o_2,\ r_3=o_3\}$ est identifié par le lien formé par les objets o_1, o_2, et o_3. En supposant que tous les rôles sont toujours définis (ce qui est une exigence pour le système AROM) et en obligeant que les objets jouant les mêmes rôles soient égaux, l'opérateur *equals* respecte l'identité des tuples lors de son application. Ainsi, d'un côté, si deux tuples sont égaux, ceci signifie qu'ils sont caractérisés par deux liens formés par des objets égaux. D'autre part, tous les rôles sont également importants, et par conséquent, le système de poids ne s'applique pas aux rôles d'un tuple pour que la relation d'égalité soit vérifiée. L'Annexe I présente l'algorithme qui implémente cette version de l'opérateur *equals*, tant pour les objets que pour les tuples.

À travers l'attribution des poids pour les variables, il est possible de mieux adapter le comportement de l'opérateur *equals* aux classes et aux associations d'un modèle de représentation par objets, dans notre cas, le modèle de contexte. Il est désormais possible de spécifier, pour chaque classe et association du modèle, un ensemble de poids selon l'application et la sémantique de chaque variable pour la classe ou l'association. L'idée est de donner, d'une part, un poids plus important aux variables capables de mieux caractériser une instance, et d'autre part, d'attribuer un poids inférieur aux variables moins significatives.

Par exemple, dans le cas du modèle de contexte (chapitre 5), nous pouvons imaginer l'attribution, pour la classe `Dispositif`, de l'ensemble des poids qui soulignent les attributs `nom` et `profil_CCPP` ($\{w_{nom}=0,35, w_{description}=0,15,\ w_{précision}=0,05,\ w_{profil\_ccpp}=0,35,\ w_{mémoire\_disponible}=0,05,\ w_{niveau\_d'énergie}=0,05\}$). Dans ce cas, nous pouvons considérer les objets illustrés par la Figure 34 (`dispositif1={nom=treo650, profil_CCPP=http://jangada/ccpp/treo650.xml, mémoire_disponible=32, précision=1}`, et `dispositif2={nom=treo650, profil_CCPP=http://jangada/ccpp/treo650.xml, mémoire_disponible=16, précision=0,5}`), qui étaient considérés comme distincts par la définition générale de l'opérateur *equals*. Ces objets sont à présent considérés comme égaux par l'opérateur *equals* avec poids (`dispositif1` $equals_{dispositif}$ `dispositif2 vrai`), même si on utilise une limite élevée (pour ce cas précis, $\mathcal{L} = 0,85$), puisque ces deux objets contiennent des valeurs identiques pour les attributs considérés, par les poids attribués, comme les plus significatifs de la classe `Dispositif`. En revanche, si on suppose un troisième objet de la même classe (`dispositif3={nom=tungsteinE2, profil_CCPP=http://jangada/ccpp/tungsteine2.xml, mémoire_disponible=32, précision=1}`), qui contient des valeurs identiques à l'objet `dispositif1` pour les attributs les moins significatifs (`mémoire_disponible, précision`), ces deux objets ne sont pas considérés comme égaux par cette version de l'opérateur (`dispositif3` $equals_{dispositif}$ `dispositif1`[79] `faux`), et ceci même avec une limite de `0,3`, inférieure au cas précédent. Ceci est une conséquence des poids attribués et

[79]Par souci de lisibilité, la structure de référence (classe ou association) utilisée par l'opérateur *equals* pourrait être omise de la notation.

de la définition de cette version de l'opérateur *equals*, qui modère la prise en compte des attributs les moins significatifs.

Outre la classe `Dispositif`, nous pouvons imaginer des ensembles de poids attribués à toutes les classes et les associations du modèle de contexte. Par exemple, pour la classe `Rôle`, nous pouvons accentuer l'importance des attributs `nom` et `niveau`, qui caractérisent mieux le rôle joué par un membre du groupe dans un collecticiel ($\{w_{nom}=0,35,\ w_{description}=0,2,\ w_{précision}=0,1,\ w_{niveau}=0,35\}$). D'autre part, pour la classe `Membre` nous pouvons considérer comme plus significatif l'attribut `identifiant`, puisque, habituellement, chaque membre du groupe doit s'identifier auprès du collecticiel pour pouvoir l'utiliser. Un autre attribut significatif est le `nom` de l'utilisateur, suivi de son `age` et de son `genre`, qui peuvent le caractériser, et ainsi de suite (par exemple, $\{w_{nom}=0,2,\ w_{description}=0,1,\ w_{précision}=0,05,\ w_{identifiant}=0,25,\ w_{age}=0,15,\ w_{genre}=0,15,\ w_{page\ personnelle}=0,05,\ w_{portable}=0,05\}$).

Cependant, même avec l'attribution de poids à chaque variable et l'utilisation d'un seuil $\mathcal{L}$, l'opérateur *equals*, tel qu'il est présenté ici, reste général. Il élude la question de la spécialisation de cette définition pour chaque classe et association, dans un système donné, en tenant compte de l'utilisation et l'importance de chaque classe et association dans le système. Par exemple, supposons une application de type « *geonotes* », tel que *NITA* [Rubinsztejn 2004] (cf. section 3.4). Dans ce type d'application sensible au contexte, les utilisateurs peuvent déposer, dans un lieu précis, des messages pour leurs collègues. Dans ce cas, au lieu de vérifier si un utilisateur se trouve exactement dans un lieu donné, il est préférable de vérifier si l'utilisateur est à proximité de ce lieu qui comporte un message. Par conséquent, on pourrait redéfinir l'opérateur *equals* pour la classe `Localisation`, afin qu'il utilise, dans la relation d'égalité, les opérateurs géographiques définis par *OpenGIS*[80], lesquels permettent de vérifier si deux régions géographiques se touchent ou possèdent une intersection.

6.2 Relation d'Inclusion – Opérateur Contains

6.2.1 Définition générale

Une des caractéristiques du modèle de contexte que nous avons proposé dans le chapitre 5 est l'utilisation tant des classes et objets que des associations et tuples pour représenter le contexte. Selon ce modèle, chaque élément de contexte n'est pas une entité isolée mais appartient, au contraire, à une représentation complexe de la situation dans laquelle évolue l'utilisateur. L'opérateur *contains* vise à explorer cette caractéristique du modèle de contexte, en comparant non plus deux objets de manière isolée, mais en tenant compte de leurs relations avec d'autres objets et tuples de la base de connaissances.

L'objectif de l'opérateur *contains* est de vérifier l'existence d'une *relation d'inclusion* entre deux objets et leurs relations. En d'autres termes, cet opérateur observe si l'ensemble formé par un objet donné et toutes les instances en liaison avec lui *contient* l'ensemble formé par un autre objet donné et toutes ses liaisons. Cette relation d'inclusion est similaire à la notion de graphe et sous-graphe. En réalité, une base de connaissances peut être décrite comme un graphe, dont les nœuds expriment un objet ou une classe [Bisson 1995]. En extrapolant cette vision, nous pouvons considérer qu'un objet ○ définit un ***graphe connexe***, où les sommets représentent les objets liés, de manière directe ou indirecte, à ○ par l'existence d'un ou plusieurs tuples, et les arêtes représentent ces tuples qui mettent ces objets en relation. Par exemple, la Figure 36 montre l'objet `contexte_Alain` et les instances qui sont liées à cet

[80]Voir http://www.opengeospatial.org/, [Ryden 2005], [Moisuc 2004] et [Moisuc 2005].

objet, tandis que la Figure 37 présente le graphe défini par ce même objet. Dans ce graphe, les nœuds représentent les objets connectés à l'objet `contexte_Alain`, et les arêtes représentent l'existence d'un tuple liant les objets en question.

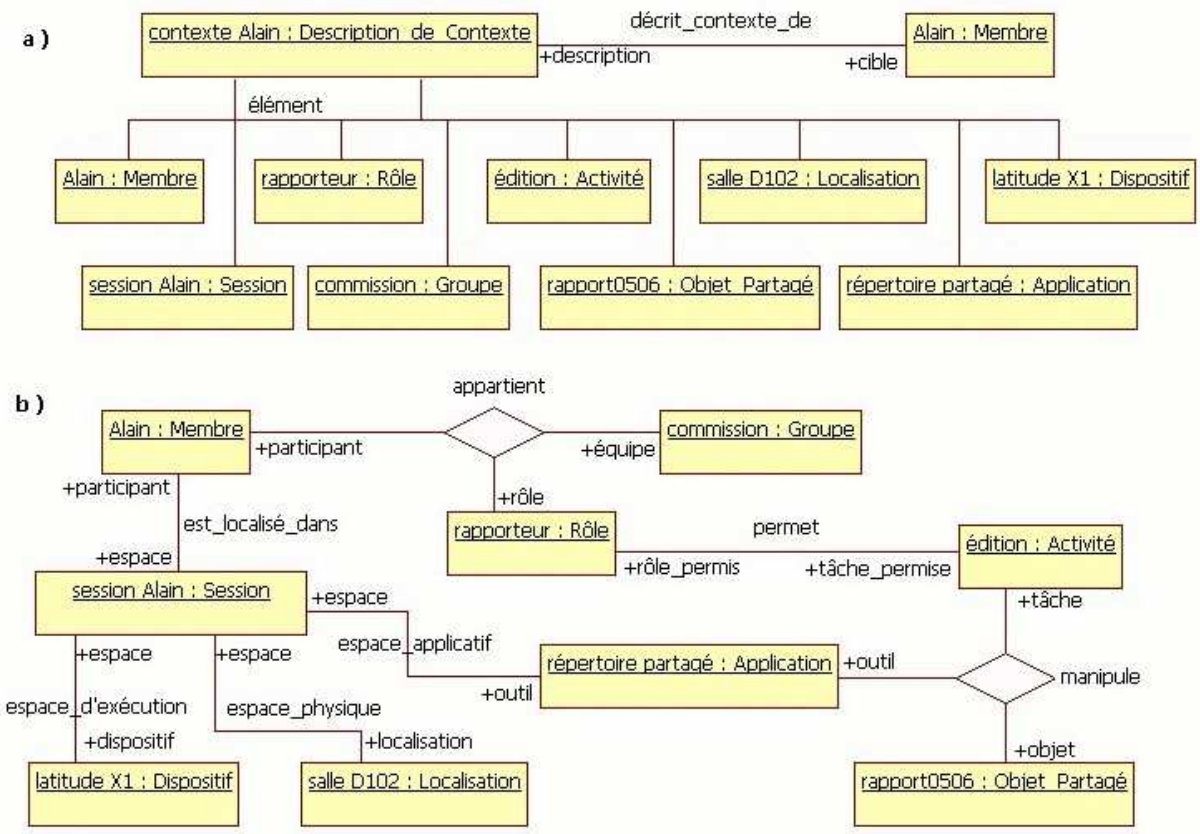


Figure 36. La représentation (en UML) d'un objet de la classe Description de Contexte, avec toutes les relations, directes (a) et indirectes (b), qu'il entretient avec d'autres instances de la base de connaissance.

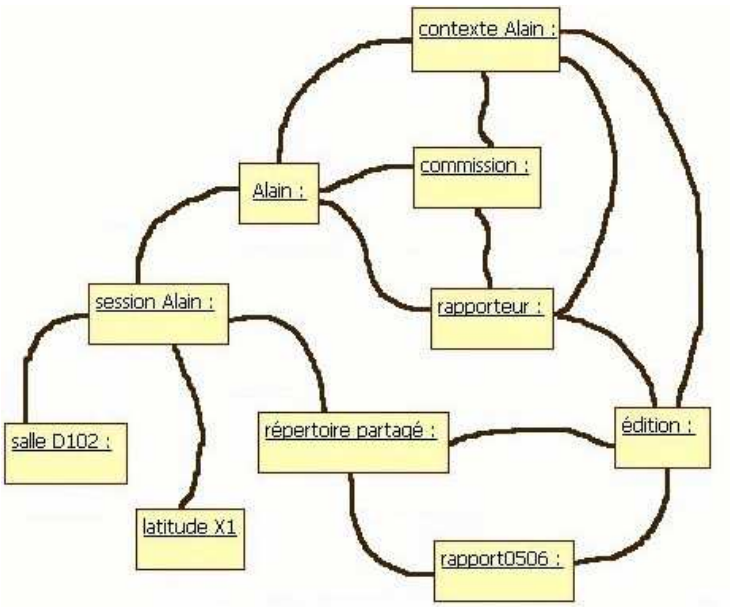


Figure 37. La représentation du graphe défini par l'objet de la classe Description de Contexte et les relations représentés dans la Figure 36 (quelques arêtes de ce graphe ont été omises afin de simplifier la présentation).

Formellement, un graphe $G = (X, U)$ est défini comme étant le couple constitué par [Berge 1970] :

- Par l'ensemble $X = \{x_1, \ldots, x_n\}$
- Par la famille $U = (u_1, \ldots, u_m)$ d'élément du produit cartésien $X \times X = \{ (x,y) \mid x \in X, y \in X \}$

Un sous-graphe est donc défini comme un sous-ensemble des sommets et des arêtes d'un graphe donné [Berge 1970] [Beineke 1988]. Ainsi, un graphe $H = (A, V)$ est un sous-graphe de G si $A \subseteq X$ et $V \subseteq U$.

Dans cette perspective, nous pouvons considérer que, si $\mathcal{G}(o) = (X, U)$ est le graphe connexe décrit par un objet o, l'ensemble X serait l'ensemble des objets lié à l'objet o, et l'ensemble U, l'ensemble des tuples qui relient les objets dans X.

Cependant, on peut se poser la question de la représentation des tuples des associations *n-aires* (cet-à-dire, les tuples liant plus de 2 objets), puisque les éléments de U sont traditionnellement définis comme un élément (x,y) du produit cartésien $X \times X$. Ceci est le cas, par exemple, des tuples de l'association `appartient` du modèle de contexte, laquelle relie les classes `Membre`, `Groupe` et `Rôle` (un exemple de tuple de cette association se trouve dans la Figure 36). Un tuple d'une telle association ne se traduit pas en une seule et unique arête dans le graphe, puisque plusieurs arêtes sont nécessaires pour représenter tous les rôles de ce type de tuple. De cette manière, chaque arête représenterait l'existence d'un tuple contenant les deux objets représentant les extrémités. Par exemple, le tuple `{participant=Alain, équipe=commission, rôle=rapporteur }` présenté dans la Figure 36 est représenté, dans la Figure 37, par 3 arêtes : une liant l'objet `Alain` à l'objet `commission`, une deuxième liant l'objet `commission` à l'objet `rapporteur`, et une troisième liant l'objet `Alain` à l'objet `rapporteur`.

Une autre représentation possible est l'utilisation *d'hypergraphes*. Un hypergraphe est une extension naturelle des graphes, où les arcs peuvent connecter plus de deux sommets, étant l'hypergraphe donc spécifié par son ensemble d'arêtes [Berge 1988]. Formellement, un hypergraphe $H = (X, \mathcal{E})$ est défini comme suit [Berge 1970] :

- soit $X = \{x_1, \ldots, x_n\}$ un ensemble fini, et $\mathcal{E} = (E_i \mid i \in I)$ une famille de parties de X. On dira que $\mathcal{E}$ constitue un *hypergraphe sur X* si :
  - $E_i \neq \emptyset \quad (i \in I)$
  - $\bigcup_{i \in I} E_i = X$
- Le couple $H = (X, \mathcal{E})$ s'appelle donc hypergraphe, et les éléments $\{x_1, \ldots, x_n\}$ de X sont les sommets, tandis que les ensembles de H, qu'on dénote $\{E_1, \ldots, E_m\}$, sont les arêtes de l'hypergraphe.

Dans le cas d'un hypergraphe, nous pouvons traduire les tuples *n-aires* sous la forme des hyperarêtes E_i. Par ailleurs, la définition d'un sous-graphe ne s'altère pas pour un hypergraphe, par rapport à la définition donnée pour un simple graphe : un hypergraphe $H' = (X', \mathcal{E}')$ est un "sous-graphe" d'un hypergraphe $H = (X, \mathcal{E})$ si $X' \subseteq X$ et $\mathcal{E}' \subseteq \mathcal{E}$.

Ainsi, afin de simplifier l'analyse, nous parlerons dans ce texte en termes de graphes, même s'il est également possible de représenter un objet de la base de connaissances et ses relations sous la forme d'un hypergraphe, puisque la définition de sous-graphe est la même tant pour un hypergraphe que pour un graphe.

Une fois que nous considérons qu'un objet et ses relations forme un graphe, la définition de la relation d'inclusion entre deux objets devient la recherche d'une relation de sous-graphe entre les deux graphes. Ainsi, nous considérons qu'un objet o contient un objet o''

si le graphe défini par o' est un sous-graphe de celui défini par o (on note $\mathcal{G}$(o) *contains* $\mathcal{G}$(o')). Par exemple, le graphe décrit par l'objet description_profil1 de la classe Description_de_Contexte présenté dans la Figure 38 est un sous-graphe de de celui présenté dans la Figure 36. Ceci signifie que le graphe décrit par l'objet contexte_Alain, représenté dans la Figure 37, contient celui décrit par l'objet description_profil1 montré dans la Figure 38 ($\mathcal{G}$(contexte_Alain) *contains* $\mathcal{G}$(description_profil1)). À travers cet exemple, nous pouvons observer que toutes les instances (objets et tuples) liées directement ou indirectement, à l'objet description_profil1 (voir Figure 36) trouvent une instance égale dans l'ensemble d'instances liées à l'objet contexte_Alain, représentées dans la Figure 38.

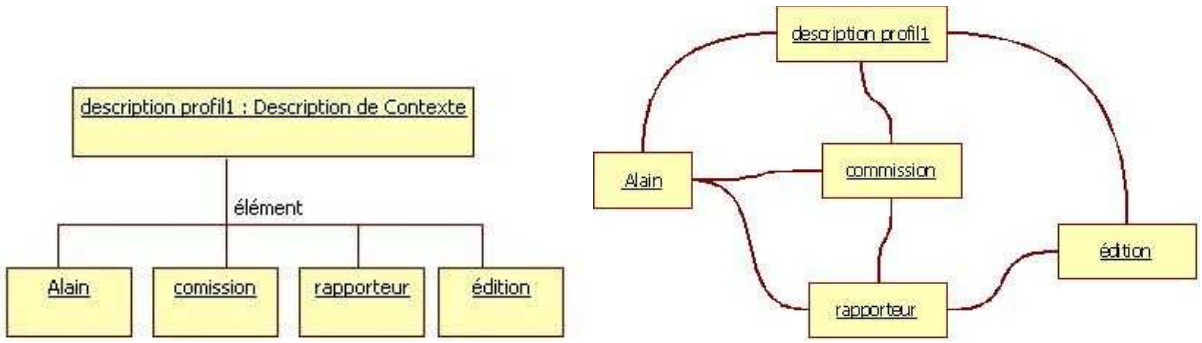


Figure 38. A droite, la représentation (en UML) de l'objet description profil1, instance de la classe Description de Contexte, et ses relations directes. A gauche, le graphe défini par cet objet.

Nous considérons qu'un graphe C=(X,U) contient un graphe C'=(X',U') si et seulement si les conditions suivantes sont vérifiées :

- pour chaque sommet N' appartenant à C' (noté $N'_{C'}$), il existe un sommet N appartenant à C (noté N_C) tel que N_C *equals* $N'_{C'}$; et
- pour toute arête E' de C' (notée $E'_{C'}$), il existe au moins une arête E de C, (notée E_C) telle que E_C *equals* $E'_{C'}$.

De manière plus formelle, afin de définir l'opérateur *contains*, il faut d'abord déterminer quels éléments forment le graphe décrit par un objet. Ainsi, étant donné un objet $o=\{a_i=v_i\}_{i\geq 1}$, le graphe $\mathcal{G}$(o) décrit par cet objet est défini comme étant $\mathcal{G}(o) = (O(o), \mathcal{T}(o))$, où $O(o)$ est l'ensemble d'objets o' pour lesquels il existe un chemin $\mathcal{P}(o, o')$ liant o' à o, formé par un ensemble de tuples, et $\mathcal{T}(o)$ est l'ensemble de ces tuples :

- soit $\mathcal{P}(o, o')$ l'ensemble de tuples qui forme un chemin entre les objets o et o',

$$\mathcal{P}(o,o') = \left\{ t_1 \ldots t_n \mid \begin{array}{l} (t_1.r_i = o \wedge t_1.r_j = o_1)_{i\neq j} \ \ldots \\ (t_m.r_k = o_{m-1} \wedge t_m.r_l = o_m)_{k\neq l} \ \ldots \\ (t_n.r_p = o_{n-1} \wedge t_n.r_q = o')_{p\neq q} \end{array} \right\}_{n\geq 1},$$

  alors, si $\mathcal{P}(o, o')$ n'est pas un ensemble vide, l'objet o' appartient à $O(o)$

$$\mathcal{P}(o,o') \neq \emptyset \rightarrow o' \in O(o)$$

- $\mathcal{T}$(o) est l'ensemble des tuples t dont chacune des extrémités $t.r_i$ correspond à un objet o' appartenant à $O(o)$,

$$\mathcal{T}(o) = \{t \mid (\forall r_i)\,((\exists o' \in O(o))\,(t.r_i = o'))\}$$

La Figure 39 illustre les définitions ci-dessus. Dans cette figure, le chemin entre les objets o et o' est donné par les tuples $\mathcal{P}$(o, o')={t_1, t_2, t_3} : le tuple t_1 connecte o à o_1, t_2 connecte o_1 à o_2 et o_3, et t_3 connecte o_3 à o'. Par conséquent, le graphe décrit par o est défini par les objets O(o) = {o, o_1, o_2, o_3, o'}, puisque pour chacun de ces objets, il existe un chemin qui le relie à o, et par les tuples $\mathcal{T}$(o)={t_1, t_2, t_3}, puisque ces tuples relient des objets dans O(o).

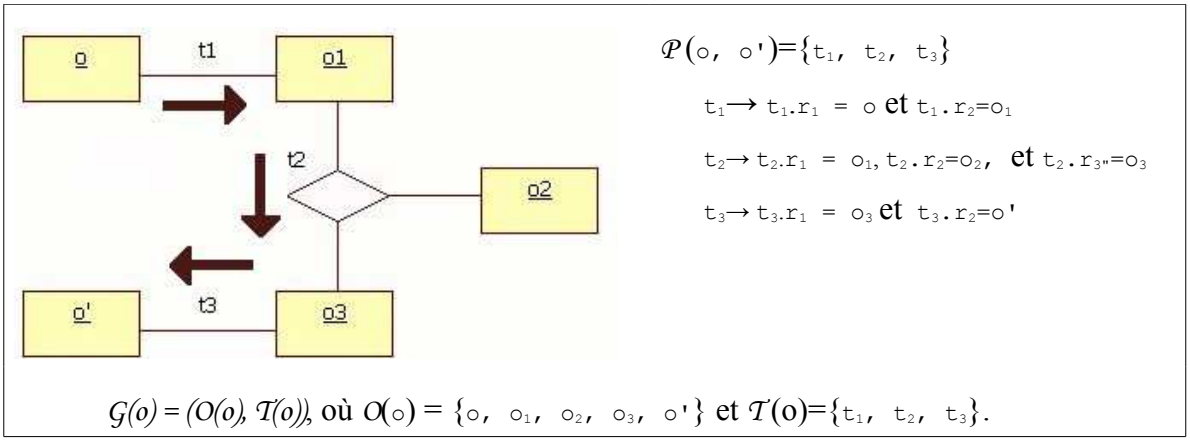


Figure 39. Exemple de chemin liant deux objets o et o'.

Une fois connu l'ensemble de tuples et d'objets qui sont liés à un objet, nous pouvons définir l'opérateur *contains* de la manière suivante :

- étant donné deux objets o={ $a_i=v_i$ }$_{i\geq 1}$ et o'={$a_i=v_i$'}$_{i\geq 1}$, lesquels définissent, respectivement, les graphes $\mathcal{G}$(o)= (O(o), $\mathcal{T}$(o)) et $\mathcal{G}$(o')=(O(o'), $\mathcal{T}$(o')), $\mathcal{G}$(o) *contains* $\mathcal{G}$(o') si et seulement si :

$$\mathcal{G}(o) \text{ contains } \mathcal{G}(o') \text{ ssi } \begin{cases} \forall o_i \in O(o') \ \exists o_j \in O(o) \ (o_i \text{ equals } o_j) \\ \forall t_i \in \mathcal{T}(o') \ \exists t_j \in \mathcal{T}(o) \ (t_i \text{ equals } t_j) \end{cases}$$

Il est intéressant d'observer l'utilisation de l'opérateur *equals* à l'intérieur même de la définition de l'opérateur *contains*. D'une part, ceci permet d'établir la relation d'inclusion basée sur le contenu des instances. D'autre part, ceci renforce l'importance d'une définition plus souple de l'opérateur *equals*, puisque celui-ci a une forte influence sur l'opérateur *contains*. L'Annexe I présente un algorithme qui correspond à cette définition générale de l'opérateur *contains*. Cet algorithme utilise un parcours en largeur pour trouver les correspondances entre les tuples, et ainsi identifier l'existence (ou non) de la relation d'inclusion.

Par ailleurs, de la même façon que pour l'opérateur *equals*, cette définition générale de l'opérateur *contains* reste assez restrictive. Lorsqu'on compare deux objets o et o' ($\mathcal{G}$(o) *contains* $\mathcal{G}$(o')), cette définition tient compte de tous les instances qui forment les graphes. Or, dans certains cas, il serait intéressant de ne considérer que certaines instances du graphe, et non sa totalité, et d'éviter ainsi un calcul supplémentaire. Par exemple, dans le modèle de contexte que nous proposons, l'association `décrit_contexte_de` met en relation un objet de la classe `Description_de_Contexte` et un objet de la classe `Élément_de_Contexte` dont cette description se réfère (la Figure 36 illustre cette association par un tuple unissant l'objet `contexte_Alain` et l'objet `Alain`). De manière conceptuelle, nous pouvons imaginer qu'un tuple de cette

association ne fait pas réellement partie du contexte d'un élément (de l'utilisateur `Alain`, dans le cas de la Figure 36), mais il représente le lien entre le contexte décrit et son "propriétaire". Avec la définition générale, cette association est toujours prise en compte par l'opérateur *contains*, même si on cherche à comparer seulement les éléments qui composent deux descriptions de contexte. Dans le cas des objets représentés dans les figures 36 et 38, tous les tuples représentés dans la Figure 38 sont comparés au tuple de cette association montré dans la Figure 36. Ceci a un coût qui pourrait être évité si les tuples de cette association pouvaient être ignorés. Ainsi, nous proposons, dans la prochaine section, un assouplissement de cette définition générale, en proposant une nouvelle version de l'opérateur *contains* qui admet une liste d'éléments de la base de connaissances qui peuvent être ignorés lors d'une comparaison.

Enfin, il convient d'observer que, de manière similaire à l'opérateur *equals*, l'opérateur *contains* est un opérateur de comparaison. Il s'agit d'un opérateur discret, son résultat peut être représenté par `0` *ou* `1` (vrai ou faux), et non d'un opérateur continu, dont le résultat oscillerait *entre* `0` et `1`. Les mesures de similarité que nous définissons dans la section 6.3 sont, par contre, des opérateurs continus, dont la valeur est un nombre réel entre `0` et `1`.

6.2.2 Définition de l'opérateur Contains avec une « ignore list »

À partir de la définition générale de l'opérateur *contains*, nous proposons dans cette section une nouvelle version de cet opérateur, laquelle admet une liste d'éléments de la base de connaissances qui seront ignorés lors de l'exécution de l'opérateur. Ceci permet de concentrer l'effort seulement sur une partie des connaissances disponibles.

La liste d'éléments ignorés est appelée « *ignore list* » et dénotée par IL. Cette liste est définie comme une liste d'éléments de la base de connaissances qui doivent être ignorés pendant l'exécution d'un opérateur. Suivant cette définition, l'IL peut contenir : (*i*) une classe, indiquant que tous ses objets (l'extension de la classe) et les tuples qui les contiennent doivent être ignorés ; (*ii*) une association, qui indique le besoin d'ignorer tous les tuples de cette association (son extension) ; (*iii*) un objet, qui dénote que cet objet et tous les tuples qui le contiennent doivent être ignorés ; (*iv*) un tuple, désignant celui-ci comme devant être ignoré. Dans tous les cas, l'intérêt d'une telle liste est d'agir au niveau de l'extension, c'est-à-dire sur les instances. De cette manière, nous pouvons traduire la liste IL par un ensemble d'instances (objets et tuples) qui doivent être ignorées lors de l'exécution de l'opérateur *contains* :

- Soit O_k l'ensemble des objets d'une base de connaissances `k`, et $\mathcal{T}_k$ l'ensemble de tuples de cette même base, IL est définie comme étant :

$$IL = \{o_i \mid o_i \in O_k\} \cup \{t_j \mid t_j \in \mathcal{T}_k\}$$

Par la suite, nous pouvons définir l'opérateur *contains* comme suit :

- étant donné deux objets $o=\{a_i=v_i\}_{i\geq 1}$ et $o'=\{a_i=v_i'\}_{i\geq 1}$, lesquels définissent, respectivement, les graphes $\mathcal{G}(o)=(O(o),\mathcal{T}(o))$ et $\mathcal{G}(o')=(O(o'),\mathcal{T}(o'))$, $\mathcal{G}(o)$ *contains* $\mathcal{G}(o')$ si et seulement si :

$$\mathcal{G}(o) \text{ contains } \mathcal{G}(o') \text{ ssi } \begin{cases} \forall o_i \in (O(o') - IL) \ \exists o_j \in (O(o) - IL) \ (o_i \text{ equals } o_j) \\ \forall t_i \in (\mathcal{T}(o') - IL) \ \exists t_j \in (\mathcal{T}(o) - IL) \ (t_i \text{ equals } t_j) \end{cases}$$

À travers cette définition, les éléments de la liste IL sont éliminés des graphes et ne sont pas pris en compte lors de l'application de l'opérateur *contains*.

L'Annexe I présente un algorithme qui met en œuvre cette version de l'opérateur *contains*. Cet algorithme organise, à partir de deux objets originaux (nommés `set` et `subset`), l'ensemble des tuples liés à chacun de ces objets et compare ces ensembles, en tenant compte des éléments représentés dans l'*ignore list*. Pour chaque tuple de l'ensemble déterminé par l'objet `subset`, l'algorithme doit trouver un tuple qui lui équivaut (un tuple égal) dans l'ensemble de tuples déterminé par l'objet `set`. On observe alors qu'on utilise notamment les arêtes (les tuples), et non les nœuds (les objets) appartenant au graphe. Ceci est possible puisque la définition même de l'opérateur *equals* pour deux tuples impose qu'ils mettent en relation des objets égaux. Par conséquent, le fait de tester les tuples revient à tester, de façon implicite, les objets appartenant aux liens de ces tuples.

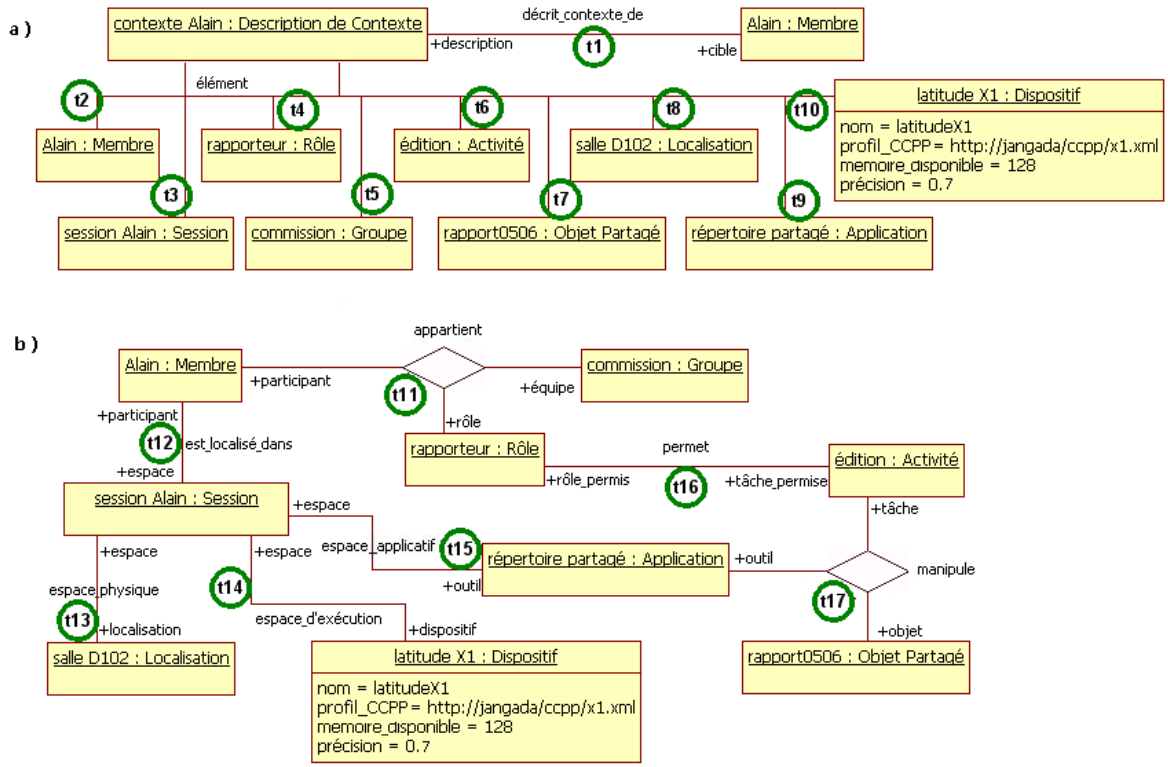


Figure 40. Instances (objets et tuples) présentes dans le graphe décrit par un objet de la classe Description de Contexte lié à un utilisateur en particulier[81].

Cette version de l'opérateur est particulièrement intéressante lorsque certaines instances ne contribuent pas directement au but recherché par la relation d'inclusion et constituent donc autant d'éléments qui pénalisent l'exécution de l'opération. Par exemple, dans la Figure 40, nous représentons, en utilisant notre modèle (cf. chapitre 5), le contexte d'un utilisateur ('`Alain`') qui tient le rôle de rapporteur dans une commission d'évaluation (rôle `rapporteur`, groupe `commission` dans la Figure 40). On s'aperçoit que, l'objet `Description_de_Contexte` qui décrit cette situation est lié, par l'association `décrit_contexte_de`, à l'objet qui représente l'utilisateur en question. D'autre part, dans la Figure 41, nous représentons un autre objet `Description_de_Contexte` relatif, non plus à un utilisateur, mais plutôt à un rôle en particulier, le rôle de `rapporteur` (les tuples dans les figures 40 et 41 sont indiqués par des numéros pour une meilleure identification). Dans ce cas, l'association `décrit_contexte_de` ne connecte plus un objet de la classe `Description_de_Contexte` à un objet de la classe `Membre`,

[81]Les tuples sont énumérés pour une meilleure identification et, hormis les objets de la classe `Dispositif`, les attributs des instances ont été omis par souci de clarté.

mais une description à un objet de la classe `Rôle` (deux spécialisations de la classe `Élément_de_Contexte`, cf. section 5.1.2).

Par ailleurs, la Figure 41 illustre la description d'une situation générale, celle d'un utilisateur quelconque (représenté par un objet par défaut de la classe `Membre`) qui joue le rôle de `rapporteur` dans le groupe `commission`. Dans ce cas précis, si on compare les objets de la classe `Description_de_Contexte` représentés dans les figures 40 et 41 en utilisant la définition générale de l'opérateur *contains*, nous aurons une réponse fausse ($\mathcal{G}$(`contexte_Alain`) *contains* $\mathcal{G}$(`description_profil2`) = `faux`), même si la situation décrite par le deuxième objet est également décrite par le premier (un utilisateur qui joue le rôle de `rapporteur` dans le groupe `commission`). Or, dans ce cas, nous pouvons appliquer l'opérateur *contains* avec une liste $\mathcal{IL}$ qui inclut l'extension de l'association `décrit_contexte_de` et l'objet « `default` » de la classe `Membre` ($\mathcal{IL}$ = {default} ∪ {t_1, t_1'}). Avec cette configuration, il est possible d'ignorer non seulement les tuples de cette association, lesquels n'apportent pas beaucoup à la description de deux situations, mais également l'objet `default` qui n'est pas significatif dans ce cas précis.

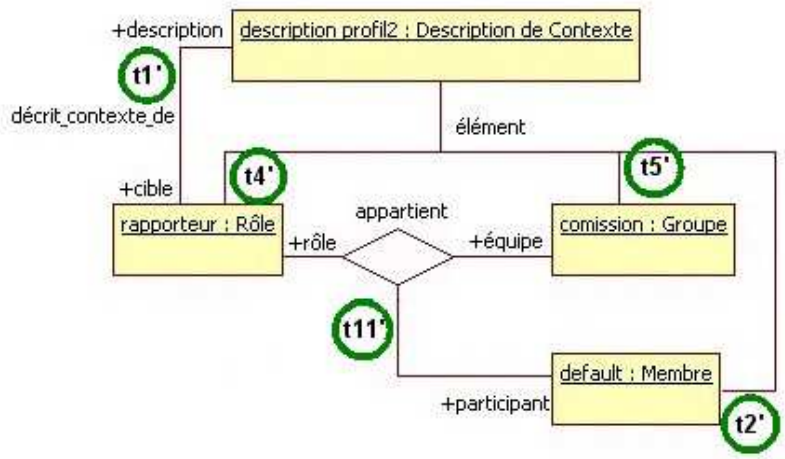


Figure 41. Exemple d'une description de contexte pour une situation générale d'un rôle dans un groupe donné[82].

6.2.3 Définition de l'opérateur Contains sous une fonction de condition

Les versions précédentes de l'opérateur *contains* ne prennent en compte que les graphes décrits par les instances comparées. Le contenu des instances qui composent les graphes n'est pas particulièrement pris en compte. Or, nous pouvons facilement imaginer l'utilisation du contenu de certains attributs en tant que condition pour l'emploi d'une instance. Ceci est notamment le cas de l'attribut `précision`, conformément à ce que nous avons discuté dans le chapitre 5. Cet attribut peut être utilisé pour indiquer si les informations gardées dans un objet `Élément_de_Contexte` sont suffisamment précises pour être exploitées. Nous pouvons donc imaginer l'utilisation d'un seuil sur cet attribut qui détermine si l'utilisation d'une instance est possible ou non lors de l'application de l'opérateur *contains*. Néanmoins, une approche limitée au seul attribut `précision` comme critère d'utilisation serait trop spécifique au modèle de contexte que nous proposons. Afin de garder la généralité de l'opérateur *contains*, nous pouvons étendre cette idée à l'utilisation d'une condition prédéfinie sur le contenu des instances. Cette condition est exprimée sous la forme d'une ***fonction de condition*** qui vérifie la validité du contenu d'une instance face à une condition donnée. Ainsi, nous pouvons définir

[82]Les tuples sont énumérés pour une meilleure identification.

une fonction qui vérifie si la valeur de l'attribut `précision` est supérieure à un seuil donné, ou d'autres conditions similaires sur le contenu des instances.

De manière générale, la *fonction condition fc (o)*, pour une instance (tuple ou objet) `o`, est définie comme suit :

$$fc(o)=\begin{cases}1 & si\ l'instance\ o\ satisfait\ les\ conditions\\ 0 & dans\ le\ cas\ contraire\end{cases}$$

De manière plus spécifique au modèle de contexte, nous proposons une fonction condition qui prend en compte la valeur de l'attribut `précision` :

- soit $\mathcal{L}$ un seuil définit comme un numéro réel entre `0 et 1`

  $\mathcal{L}\in\mathbb{R}^+,\ \mathcal{L}\in[0,1]$

- soit `o` un objet appartenant à l'extension de la classe `élément de contexte`

  $o\in EC_E$

- la fonction condition *fc(o)* est définie comme étant :

  $$fc(o)=\begin{cases}1 & si\ \ o.précision\neq null\ \ et\ \ o.précision\geqslant\mathcal{L}\\ 0 & dans\ le\ cas\ contraire\end{cases}$$

Cependant, il est important de remarquer que celle-ci n'est pas la seule fonction de condition qui peut être définie sur le modèle de contexte. D'autres possibilités existent, même si, pour nous, l'utilisation de la `précision` pour une telle fonction demeure une priorité (cf. chapitre 5). Une fois déterminée cette fonction de condition, nous pouvons définir une nouvelle version de l'opérateur *contains* qui tient compte de cette fonction. Cette nouvelle version est définie de la manière suivante :

- soit O l'ensemble d'objets d'une base de connaissances, et $\mathcal{T}$ l'ensemble de tuples de cette même base, la liste IL d'instances qui doivent être ignorées est définie comme étant un sous-ensemble de ces ensembles :

  $$IL=\{o_i \mid o_i\in O\}\ \cup\ \{t_j \mid t_j\in\mathcal{T}\}$$

- étant donné deux objets $o=\{a_i=v_i\}_{i\geq 1}$ et $o'=\{a_i=v_i'\}_{i\geq 1}$, lesquels définissent, respectivement, les graphes $\mathcal{G}(o) = (O(o), \mathcal{T}(o))$ et $\mathcal{G}(o')= (O(o), \mathcal{T}(o))$, nous appelons $\mathcal{G}_{fc}(o)$ et $\mathcal{G}_{fc}(o')$ l'ensemble d'éléments de ces graphes qui ont été validés par la fonction de condition $fc_{(o_i)}$:

$$\mathcal{G}_{fc}(o) = (O_{fc}(o), \mathcal{T}_{fc}(o))$$
$$O_{fc}(o) = \{o_i\in O(o) \mid o_i\notin IL \ \wedge\ fc(o_i)\neq 0\}$$
$$\mathcal{T}_{fc}(o) = \{t_i\in\mathcal{T}(t) \mid t_i\notin IL \ \wedge\ fc(t_i)\neq 0\}$$

- Ainsi, nous pouvons affirmé que $\mathcal{G}(o)$ *contains* $\mathcal{G}(o')$ si et seulement si :

$$\mathcal{G}(o)\ \textit{contains}\ \mathcal{G}(o')\ \text{ssi}\ \begin{cases}\forall o_i\in(O_{fc}(o'))\ \ \exists o_j\in(O_{fc}(o))\ \ (o_i\ \textit{equals}\ o_j)\\ \forall t_i\in(\mathcal{T}_{fc}(o'))\ \ \exists t_j\in(\mathcal{T}_{fc}(o))\ \ (t_i\ \textit{equals}\ t_j)\end{cases}$$

Afin d'illustrer l'application de cette version de l'opérateur *contains*, nous pouvons considérer l'exemple ci-dessous (voir Figure 42), dans lequel nous comparons deux objets `o` et `o'` appartenant à l'extension d'une classe `C` ($C_I=\{a_1:d_1,\ a_2:d_2,\ a_3:d_3\}$) avec une fonction *fc(o)*= $o.a_1 \geq 7$. Dans ce cas, la fonction de condition *fc(o)* ne permet pas l'utilisation des objets

o_1', o_3 et o_3', puisqu'ils ne satisfont pas les conditions mises en place par cette fonction. Ces instances ne participent pas au calcul. Par conséquent, le nombre d'instances comparées est réduit, et le coût de l'exécution de l'opérateur *contains* également.

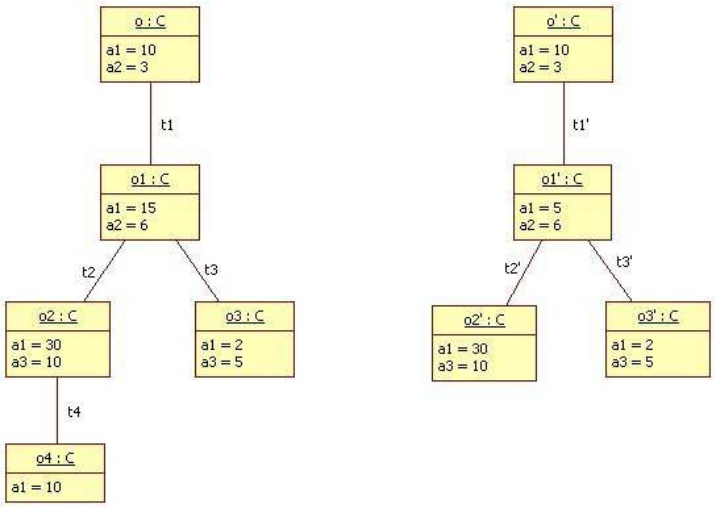


Figure 42. Exemple d'instances qui peuvent être comparées à travers l'opérateur *contains*

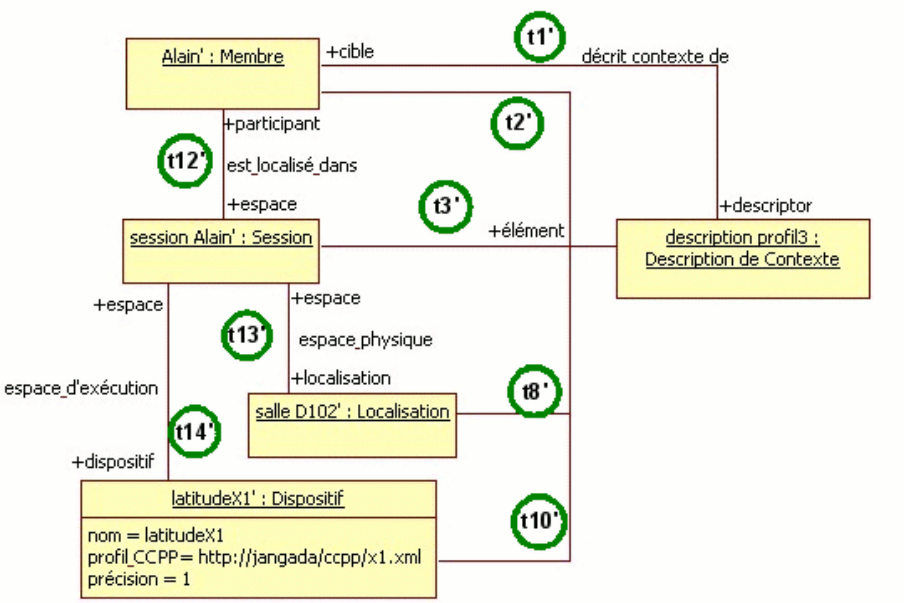


Figure 43. Description de Contexte décrivant une situation où l'utilisateur Alain se trouve dans la salle de réunion et utilise son ordinateur portable (les tuples sont énumérés pour une meilleure identification).

En ce qui concerne le modèle de contexte, nous pouvons imaginer, par exemple, la comparaison de l'objet de la classe `Description_de_Contexte` représenté dans la Figure 40 et celui décrit par la Figure 43. En utilisant les versions précédentes de l'opérateur *contains*, tous les objets de la classe `Dispositif` sont pris en compte et le résultat dépend de la version de l'opérateur *equals* utilisée (si la version générale est utilisée, les deux objets sont considérés comme distincts, puisqu'ils ont une valeur différente pour l'attribut `précision`). Par contre, en

utilisant l'opérateur *contains* avec la fonction de condition que nous avons défini ci-dessus (*fc(o)* (`o.précision` $\geq \mathcal{L}$), ces deux objets sont pris en considération seulement si le seuil $\mathcal{L}$, utilisé par la fonction *fc*, est égal ou inférieur à `0.8`. Dans le cas contraire ($\mathcal{L}$ `>0,8`) l'opérateur *contains* ne tient pas compte de l'objet `latitude_X1` montré dans la Figure 40, puisqu'il ne satisfera pas la condition exprimée par la fonction de condition *fc(o)* (`o.précision` $\geq \mathcal{L}$, où $\mathcal{L}$ `>0.8`), et, par conséquent, l'objet `contexte_Alain` (voir Figure 40) ne contiendra pas l'objet `description_profil3` présenté dans la Figure 43 ($\mathcal{G}$`(contexte_Alain)` *contains* $\mathcal{G}$`(description_profil3)` = `faux`).

L'exemple ci-dessus illustre bien la flexibilité que cette nouvelle version de l'opérateur, ainsi que les effets que celle-ci peut avoir lors de son application. Il est important d'observer que les différentes versions de l'opérateur *contains*, ainsi que celles de l'opérateur *equals*, ne sont pas équivalentes. Nous pouvons obtenir des résultats différents pour la comparaison entre deux objets `o` et `o'` selon les versions utilisées de chaque opérateur. En revanche, contrairement à l'opérateur *equals*, l'opérateur *contains* n'est pas commutatif. Nous pouvons affirmer que `o` $equals_c$ `o'=o'` $equals_c$ `o`, quel que soit la version de l'opérateur utilisée, mais nous ne pouvons pas affirmer que l'opération $\mathcal{G}$`(o)` *contains* $\mathcal{G}$`(o')` aura forcément le même résultat que l'opération $\mathcal{G}$`(o')` *contains* $\mathcal{G}$`(o)`.

Enfin, l'Annexe I présente un algorithme qui exécute cette version de l'opérateur *contains* sous une fonction de condition. Cet algorithme est similaire à ceux des versions précédentes de l'opérateur : il organise d'abord les deux graphes à travers un parcours en largeur. Par contre, pendant ce parcours, l'algorithme analyse si les instances (tuples et objets) satisfont la fonction de condition. Si une instance ne satisfait pas les conditions imposées par cette fonction, elle ne doit pas être prise en compte par l'opérateur. Ceci équivaut donc à l'ajouter à la liste $\mathcal{IL}$. Par conséquent, dans le cas d'un objet, les tuples qui connectent celui-ci à d'autres objets sont également ignorés. D'autre part, d'un point de vue pratique, cette fonction de condition peut être réalisée à travers un mécanisme de *callback*, ou encore à travers le passage d'une référence vers une méthode qui implémente la fonction de condition. Tous ces mécanismes sont disponibles dans la majorité des langages de programmation orientée à objets.

6.3 Relation de Similarité

Dans cette section, nous nous intéressons aux relations de similarité entre les instances d'une base de connaissances. Contrairement aux sections précédentes, nous proposons ici des mesures quantitatives pour évaluer la similarité entre deux instances. L'objectif est de quantifier le degré de correspondance entre ces instances, soit de manière isolée, à l'instar de l'opérateur *equals*, soit en tenant compte de leur connexions avec d'autres instances, à l'instar de l'opérateur *contains*. Le résultat ici est donc un nombre réel entre `0` et `1` qui quantifie la relation de similarité entre les instances.

Nous proposons trois mesures de similarité distinctes ($\mathcal{SimO}$, $\mathcal{SimT}$ et $\mathcal{Sim}$). La première, $\mathcal{SimO}$, s'intéresse à la relation de similarité entre deux objets de manière isolée, sans prendre en compte les relations que les objets peuvent entretenir avec d'autres. La seconde, $\mathcal{SimT}$, est semblable à la mesure $\mathcal{SimO}$, mais elle s'intéresse à la relation de similarité entre les tuples d'une base de connaissances. La troisième mesure, $\mathcal{Sim}$, comme l'opérateur *contains*, analyse deux instances en tentant compte de leurs relations dans la base de connaissances. Néanmoins, contrairement à l'opérateur *contains*, qui cherche à établir une relation d'inclusion, la mesure

Sim quantifie une relation de similarité entre les instances et leurs relations. Les prochaines sections décrivent en détail ces trois nouveaux opérateurs.

6.3.1 Mesure de similarité pour les objets : SimO

L'objectif de la mesure de similarité SimO est d'évaluer le degré de similarité entre deux objets isolés. Cette mesure ne tient compte que du contenu de ces objets, donc des valeurs de leurs attributs. Cependant, ces attributs ne sont pas considérés de la même manière. Les attributs définis par une classe ne sont pas toujours d'égale importance, certains sont plus significatifs que d'autres (cf. section 6.1.2). Afin de prendre en compte ces différences, nous utilisons, à l'instar des travaux de Bisson [Bisson 1995] et de Valtchev [Valtchev 1999], la notion de poids en vue de calculer une distance sémantique entre deux objets. Ces poids correspondent à ceux que nous avons utilisé lors de la définition de l'opérateur *equals* (cf. section 6.1.2). On associe donc à chaque attribut d'une classe un poids numérique, sous la forme d'un nombre réel entre `0` et `1`, afin d'exprimer la pertinence dont celui-ci dispose dans la classe.

Étant donné l'attribution d'un poids à chaque attribut, la mesure de similarité pourrait être définie :

- soit deux objets `o` et `o'` appartenant à l'extension d'une classe `C` de la base de connaissances `k`

$$o, o' \in C_E,\ C_{I,k} = \{a_i : d_i\}_{i=1..n},\ o = \{a_i = v_i\}_{i \geq 1}\ et\ o' = \{a_i = v_i'\}_{i \geq 1}$$

- soit w_i le poids attribué à un attribut a_i de `C`

$$\sum_{i=1}^{n} w_i = 1$$

- soit la mesure de similarité entre `o` et `o'` :

$$SimO(o, o') = \sum_{i=1}^{n} fi(o.a_i, o'.a_i) \times w_i$$

Néanmoins, conformément à ce qui a été déclaré dans la section 6.2.3, l'utilisation de l'attribut `précision` afin d'indiquer si les informations tenues par un objet `Élément_de_Contexte` peuvent être exploitées constitue un atout pour le modèle de contexte. Le contenu de cet attribut peut donc être utilisé pour modérer la relation de similarité entre les objets, puisque nous ne pouvons pas affirmer qu'il existe une forte similarité entre deux objets si leur contenu a un faible degré de précision (les informations sont trop imprécises pour que la mesure soit elle-même précise). Si deux objets se caractérisent par une faible `précision`, les informations qu'ils contiennent peuvent être erronées et la mesure de similarité doit prendre ceci en compte. Toutefois, l'utilisation directe de l'attribut `précision` dans la mesure de similarité *SimO* constitue, comme pour l'opérateur *contains* (cf. section 6.2.3), un élément qui restreint cette opération au modèle de contexte, ce qui peut nuire à la généralité de la proposition.

Ainsi, de la même manière que pour l'opérateur *contains* (cf. section 6.2.3), nous adoptons ici une approche plus générale, à travers la définition d'une ***fonction de précision*** *fp(o, o')* ($fp(o, o') = x,\ x \in \mathbb{R}^+\ et\ x \in [0,1]$). Le but de cette fonction est d'évaluer si le contenu de deux objets permet la réalisation d'une mesure de similarité fiable. Cette fonction

peut être définie, de la même manière que la fonction de condition *fc(o)* utilisé par l'opérateur *contains*, selon les besoins du modèle où elle est appliquée. En ce qui concerne notre modèle de contexte (cf. chapitre 5), nous définissons cette fonction comme suit, cependant, il est important d'observer que celle-ci n'est pas la seule option possible :

- soit deux objets o et o' appartenant à l'extension de la classe `Élément_de_Contexte`

$$o, o' \in EC_E$$

- la fonction de précision *fp(o, o')* est définie comme étant :

$$fp(o, o') = \frac{p(o) + p(o')}{2}, \text{ où } p(o) = \begin{cases} o.precision & si\ o.précision \neq null \\ 0 & si\ o.précision = null \end{cases}$$

$$p(o') = \begin{cases} o'.precision & si\ o'.précision \neq null \\ 0 & si\ o'.précision = null \end{cases}$$

Ainsi, nous pouvons définir l'opérateur *SimO* comme suit :

- soit *fd(o)* la cardinalité de l'ensemble formé par les variables dont la valeur est connue dans un objet o
  - $fd(o) = \{a_i \mid o.a_i \neq null\}$
- étant donné les objets o et o'

$$o, o' \in C_E,\ o = \{a_i = v_i\}_{1 \leq i \leq p},\ o' = \{a_i = v_i\}_{1 \leq i \leq q}$$

- la mesure de similarité *SimO (o, o')* est définie comme

$$SimO(o, o') = \left(\sum_{i}^{fd(o) \cap fd(o')} fi(o.a_i, o'.a_i) \times w_i \right) \times fp(o, o')$$

L'utilisation de la fonction de précision sert à modérer la simple comparaison entre les valeurs des attributs : si au moins un des objets comparés détient un faible indice de précision, cette fonction tend à réduire la valeur obtenue par la comparaison entre les attributs, puisque les informations contenues dans les objets ne sont pas suffisamment précises. Ainsi, il en résulte un opérateur quantitatif, dont le résultat est une valeur réel entre `0` et `1` (*SimO (o, o')* ∈ `[0,1]`), qui dépend directement du poids donné à chaque attribut et du contenu des objets, lequel est évalué à travers la fonction d'identité (*fi(v,v')*) et la fonction de précision (*fp(o,o')*).

Poids w_i : $\{w_{nom}=0,35,\ w_{description}=0,15,\ w_{précision}=0,05,\ w_{profil\_ccpp}=0,35,\ w_{mémoire\_disponible}=0,05,\ w_{niveau\ d'énergie}=0,05\}$

Objet 1 : dispositif1={nom=treo650, profil_CCPP=http://jangada/ccpp/treo650.xml, mémoire_disponible=32, précision=1}

Objet 2 : dispositif2={nom=treo650, profil_CCPP=http://jangada/ccpp/treo650.xml, mémoire_disponible=16, précision=0,5}

SimO (dispositif1, dispositif2) = 0,525

dispositif1 $equals_{Dispositif}$ dispositif2 = vrai seulement si $\mathcal{L} \leq 0,85$

Figure 44. Exemple d'utilisation de l'opérateur *SimO*.

Par ailleurs, la principale différence entre la mesure *SimO* et l'opérateur *equals* par poids réside dans la nature du résultat : un nombre réel pour le premier et un booléen pour le second. Ainsi, au lieu d'avoir une réponse "binaire" (`vrai` ou `faux`), comme c'est le cas pour l'opérateur *equals*, nous avons ici une mesure plus fine qui évalue la proximité entre le contenu des objets. Par exemple, dans le Figure 44, nous considérons les objets de la classe `Dispositif` présentés dans la Figure 34 (page 116) avec les poids suggérés pour cette classe dans la section 6.1.2. Nous pouvons observer que, dans ces conditions, ces objets ont, avec la mesure `SimO`, un indice de similarité proche de `0,5`, alors que, en utilisant l'opérateur *equals*, nous obtenons seulement l'indication qu'ils sont suffisamment proches pour être considérés comme égaux.

L'Annexe I contient l'algorithme de mesure de similarité *SimO*. Cet algorithme, à l'instar de la structure de référence donnée lors de l'application de l'opérateur *equals*, suit le principe selon lequel les objets que l'on compare ont une structure en commun (ils appartiennent à l'extension d'une même classe). Cette structure en commun guide le choix des poids attribués aux variables. Par exemple, dans la Figure 32 (page 114), nous avons deux objets `o` et `o'` qui appartiennent à l'extension d'une même super-classe `C`, tout en étant attachés à deux sous-classes distinctes de `C` (`o` est un objet de la sous-classe `C'`, tandis que `o'` est un objet de la classe `C''`). Dans ce cas, l'opérateur *SimO* ne considère que les variables définies par la super-classe `C`, commune aux deux objets en question, car celle-ci garantit une base sur laquelle la comparaison entre les objets est possible. Il s'agit de fixer un espace de valeurs sur lequel on procède à la comparaison, sur lequel la mesure de similarité est réalisée.

6.3.2 Mesure de similarité pour les tuples : SimT

La mesure de similarité *SimT* est le complément naturel de la mesure *SimO*, puisqu'elle évalue le degré de similarité entre deux tuples. L'objectif de cette mesure est d'évaluer le degré de correspondance entre deux tuples, en analysant, pour cela, les valeurs de leurs attributs, ainsi que le contenu des objets qu'ils connectent. Par ailleurs, la mesure de similarité *SimT* utilise la même notion de poids celle utilisé dans la mesure *SimO* et dans l'opérateur *equals* par poids. En revanche, contrairement à l'opérateur *equals*, la mesure de similarité *SimT* considère différemment les rôles d'une association, certains étant plus significatifs que d'autres. Par conséquent, nous pouvons avoir la même notion de poids attribuée, cette fois-ci, aux rôles de l'association. Une autre conséquence, celle-ci théorique, serait la possibilité d'avoir des tuples qui n'ont pas d'objets attribués à tous les rôles. Cependant, il convient d'observer que le modèle AROM qui nous utilisons dans ce travail ne permet pas ce cas de figure, qui reste, de ce fait, théorique.

De la même manière que pour la mesure *SimO*, le résultat de la mesure *SimT* est un nombre réel entre `0` et `1` ($SimT(t, t') \in [0,1]$), qui représente le degré de similarité entre les tuples. Pour obtenir ce résultat, cette mesure évalue les valeurs attribuées aux variables et les objets associés aux rôles. Pour les variables, elle utilise une mesure similaire à la mesure *SimO*, en analysant chaque variable en fonction du poids qui lui est attribué. Pour les objets qui forment les liens des tuples, l'opérateur *SimT* évalue la similarité entre les objets jouant un même rôle dans chaque tuple (à travers l'opérateur *SimO*). Cette similarité entre les objets est ensuite modérée par le poids qui est attribué au rôle dans l'association. Par ailleurs, la mesure *SimT* utilise également une association commune aux tuples comparés comme point de départ pour la comparaison, puisque les poids sont attribués aux variables et aux rôles d'une association donnée. Ceci permet l'utilisation d'un ensemble de poids unique pour les deux

tuples comparés. Enfin, la mesure $\mathcal{SimT}$ considère que les variables et les rôles qui forment un tuple sont d'égale importance lors de l'évaluation de la similarité entre deux tuples.

Ainsi, nous définissons la mesure de similarité $\mathcal{SimT}$ comme suit :

- soit $fd(t)$ la cardinalité de l'ensemble formé par les *variables* dont la valeur est connue dans un tuple `t`

$$fd(t) = \{a_j \mid t.a_j \neq null\}$$

- soit $fdr(t)$ la cardinalité de l'ensemble formé par les *rôles* dont la valeur est connue dans un tuple `t`

$$fdr(t) = \{r_i \mid t.r_i \neq null\}$$

- étant donné deux tuples `t` et `t'` appartenant à l'extension d'une association `A`

$$t, t' \in A_E,\ A_I = \{a_j : d_j\}_{j=0..m} + \{o_i : C(r_i)\}_{i=2..n},$$
$$t = \{a_j = v_j\}_{0 \leqslant j \leqslant k} + \{r_i = o_i\}_{2 \leqslant i \leqslant p}$$
$$t' = \{a_j = v_j'\}_{0 \leqslant j \leqslant l} + \{r_i = o_i'\}_{2 \leqslant i \leqslant q}$$

- étant donné qu'à chaque variable a_j présente dans l'intension de `A` est attribué un poids w_j, et qu'à chaque rôle r_i de l'association `A` est associé un poids w_i, tel que :

$$\sum_{j=0}^{m} w_j = 1,\ et\ \sum_{i=0}^{n} w_i = 1$$

- la mesure $\mathcal{SimT}(t, t')$ est définie comme

$$\mathcal{SimT}(t, t') = \frac{\sum_{j}^{fd(t) \cap fd(t')} (fi(t.a_j, t'.a_j) \times w_j) + \sum_{i}^{fdr(t) \cap fdr(t')} (SimO(t.r_i, r'.r_i) \times w_i)}{2}$$

L'utilisation de poids pour les rôles est particulièrement intéressante, car à travers l'attribution des poids, nous pouvons donner plus d'importance à certains aspects d'une association. Par exemple, nous pouvons considérer, dans le cas du modèle de contexte, l'association « `manipule` » (cf. Figure 19, page 82). Cette association compte deux variables (l'`historique` et la `criticité`) et trois rôles : un rôle nommée `tâche`, typé par la classe `Activité`, un seconde rôle nommé `objet`, typé par la classe `Objet_Partagé`, et un rôle nommé `outil`, typé par la classe `Application`. Dans le cas de cette association, nous pouvons imaginer l'attribution d'un poids plus important aux rôles `objet` et `tâche` ($\{w_{objet}=0,4,\ w_{tâche}=0,4,\ w_{outil}=0,2\}$), puisque ces rôles peuvent être considérés comme plus pertinents dans le cadre d'un travail en équipe. De même, nous pouvons considérer la variable `criticité` comme plus pertinente pour cette association et lui attribuer un poids supérieur à celui attribué à la variable `historique` ($\{w_{criticité}=0,6,\ w_{historique}=0,4\}$).

L'algorithme de cet opérateur est présenté dans l'Annexe I.

6.3.3 Mesure de similarité pour les graphes : Sim

La dernière opération que nous proposons vise à évaluer la similarité entre les graphes décrits par deux objets. Cette mesure de similarité $Sim(o,o')$ tient compte non seulement du contenu des objets, mais surtout des relations que ces objets entretiennent dans la base de connaissances. À ce sujet, la mesure *Sim* est similaire à l'opérateur *contains*, puisque tous les

deux comparent les graphes décrits par deux objets. Cependant, contrairement à l'opérateur *contains*, qui analyse la simple l'existence d'une relation d'inclusion, la mesure *Sim* cherche à évaluer le degré de correspondance entre les deux graphes. Elle nous donne une indication quantitative de la similarité entre les graphes, sous la forme d'un nombre réel entre 0 et 1 (*Sim (o, o')* ∈ [0,1]).

La mesure de similarité *Sim (o,o')* analyse dans quelle proportion le graphe décrit par l'objet o est composé par des éléments (objets et tuples) qui trouvent une correspondance dans le graphe décrit par o'. Cette correspondance se fait à l'aide de l'opérateur *equals*. En d'autres termes, la mesure de similarité comptabilise la proportion d'éléments du graphe décrit par o qui ont un élément égal dans celui décrit par o'. Par conséquent, plus les graphes disposent d'éléments en commun, plus la mesure *Sim (o, o')* sera élevée, jusqu'au point où tous les éléments de $\mathcal{G}(o)$, tant les objets que les tuples, disposent d'un élément égal dans $\mathcal{G}(o')$. Ainsi, nous définissons la mesure *Sim(o, o')* comme suit :

- soit deux objets o et o', lesquels décrivent, respectivement, les graphes $\mathcal{G}(o)$ et G(o') :

$$o=\{a_i=v_i\}_{i\geq 1},\ o'=\{a_i=v_i'\}_{i\geq 1},$$
$$\mathcal{G}(o)=(O(o),\mathcal{T}(o)),\quad \mathcal{G}(o')=(O(o'),\mathcal{T}(o'))$$

- nous définissons $Sim(o,o')=x,\ x\in\mathbb{R}^+\ et\ x\in[0,1]$ tel que :

$$Sim(o,o')=\frac{|XO|+|XT|}{|O(o)|+|\mathcal{T}(o)|},\ où\ \begin{cases} XO=\{x|x\ equals\ y,\ x\in O(o),\ y\in O(o')\} \\ XT=\{x|x\ equals\ y,\ x\in\mathcal{T}(o),\ y\in\mathcal{T}(o')\} \end{cases}$$

  - ainsi, *Sim (o, o')* = 1 si et seulement si tous les éléments du graphe $\mathcal{G}(o)$ ont un élément égal en $\mathcal{G}(o')$

Objet 1 : contexte_Alain
O (contexte_Alain) = { contexte_Alain, Alain, session_Alain, rapporteur, commission, édition, rapport0506, salle_D102, répertoire_partagé, latitude_X1 }
$\mathcal{T}$(contexte_Alain) = { t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17 }
| O (contexte_Alain) | = 18
| $\mathcal{T}$ (contexte_Alain) | = 17

Objet 2 : description_profil3
O (description_profil3) = { description_profil3, Alain', session_Alain', salle_D102', latitude_X1' }
$\mathcal{T}$(description_profil3) = { t1', t2', t3', t8', t10', t12', t13', t14' }
| O (description_profil3) | = 5
| $\mathcal{T}$ (description_profil3) | = 8

***Sim* (contexte_Alain, description_profil3) = 0,371**
XO = { contexte_Alain, Alain, session_Alain, salle_D102, latitude_X1 } | XO | = 5
XT = { t1, t2, t3, t8, t10, t12, t13, t14 } | XT | = 8
(en utilisant l'opérateur *equals* par poids, et une limite $\mathcal{L}$ = 0,8)

***Sim* (description_profil3, contexte_Alain) = 1**
XO = { description_profil3, Alain', session_Alain', salle_D102', latitude_X1' } | XO | = 5
XT = { t1', t2', t3', t8', t10', t12', t13', t14' } | XT | = 8
(en utilisant l'opérateur *equals* par poids, et une limite $\mathcal{L}$ = 0,8)

Figure 45. Exemple d'utilisation de l'opérateur *Sim*.

Selon la définition ci-dessus, l'opération *Sim (o,o')* est naturellement asymétrique, puisque l'application de *Sim (o,o')* ne donne pas forcément le même résultat que l'application de *Sim (o',o)*. Pour illustrer cette situation, prenons, par exemple, le cas des objets de classe `Description_de_Contexte` représentés dans les figures 40 (objet `contexte_Alain`) et 42 (objet `description_profil3`). La comparaison entre ces deux objets fournit des valeurs variées selon l'ordre utilisé, comme nous pouvons observer dans le Figure 45.

Par ailleurs, nous pouvons également observer l'importance du choix de la version de l'opérateur *equals* utilisée lors du calcul de la mesure de similarité *Sim*. Selon la version de l'opérateur, le résultat de la mesure de similarité peut varier sensiblement. Dans l'exemple de la Figure 45, l'utilisation de la définition générale fait chuter la valeur de *Sim* (`contexte_Alain, description_profil3`) de `0,37` à `0,28`, et *Sim* (`description_profil3, contexte_Alain`) de `1` à `0,76`, puisque, selon la définition générale de l'opérateur *equals*, les objets `latitude_X1` et `latitude_X1'` seront considérés comme distincts (voir les attributs de ces objets dans les figures 40 et 42). Ainsi, l'utilisation d'une version plus ou moins souple de l'opérateur *equals* rend la mesure de similarité *Sim* plus ou moins contraignante, selon l'intérêt du système qui l'applique. L'Annexe I présente l'algorithme qui met en place la mesure de similarité *Sim*.

6.4 Conclusions

Dans ce chapitre[83], nous avons proposé une série d'opérations capables de comparer, de différentes manières, deux instances d'une base de connaissances. Ces opérations sont complémentaires : elles analysent différents types de relations (relation d'égalité, relation d'inclusion, relation de similarité). Par ailleurs, d'une part, les opérateurs *equals* et *contains* sont des opérateurs de comparaison, leur résultat oscille entre deux valeurs possibles : `0` *ou* `1` (vrai ou faux). D'autre part, le résultat des mesures *SimO*, *SimT* et *Sim* est un nombre qui varie dans un ensemble continu de valeurs possibles : *entre* `0` *et* `1`.

La proposition de ces opérations vise particulièrement le modèle de contexte et son exploitation. Elles permettent la comparaison entre les instances de ce modèle, tant de manière isolée, qu'en tenant compte de leurs relations avec d'autres instances. Néanmoins, il convient d'observer que tous ces opérations ont été proposées de manière aussi générale que le modèle de contexte. Par conséquent, chacune de ces opérations (*equals*, *contains*, *SimO*, *SimT* et *Sim*) peut être appliquée à des instances d'une quelconque base de connaissances par objets.

[83]Les sujets abordés dans ce chapitre ont été traités par les publications [1] et [6] indiquées dans l'Annexe IV.

7 Filtrage Guidé par le Contexte

7.1 L'Approche

Nous avons vu, dans le chapitre 4, que face à une utilisation de plus en plus nomade, les collecticiels sur le Web doivent désormais être conçus en tant que systèmes sensibles au contexte. Les contraintes liées aux nouvelles technologies et le nomadisme croissant des utilisateurs accentuent certains défis de la conception des collecticiels, dont la question de la surcharge dans les mécanismes de conscience de groupe. Ces mécanismes doivent être repensés, car les techniques traditionnellement utilisées ne s'appliquent plus face aux contraintes actuelles. Nous avons constaté (cf. chapitre 4) qu'il faut désormais un filtrage des informations de conscience de groupe qui ne soit plus limité à la simple sélection des informations. Il faut également organiser les informations délivrées à l'utilisateur en fonction de leur pertinence, car les ressources mises à disposition de l'utilisateur sont souvent réduites, ainsi que le temps dont il dispose pour appréhender ces informations.

Pour un utilisateur nomade dans un collecticiel, la pertinence d'une information ne dépend plus uniquement de ses propres préférences. Le contexte courant de l'utilisateur joue également un rôle important. Nous croyons qu'une information de conscience de groupe devient plus ou moins pertinente selon la situation dans laquelle se trouve l'utilisateur. Par ailleurs, la notion de contexte doit également prendre en compte les aspects liés au processus de coopération, puisque l'utilisateur d'un collecticiel n'est pas isolé, mais appartient à un ou plusieurs groupes de travail dans lesquels il est engagé. Le modèle de contexte que nous proposons (cf. chapitre 5) représente ces aspects, tels que les concepts de rôle, de groupe ou encore d'objet partagé. Ce modèle permet ainsi la prise en charge de ces aspects dans un processus de filtrage.

Dans ce chapitre, nous proposons donc un processus de *filtrage guidé par le contexte* capable de sélectionner un sous-ensemble des informations de conscience de groupe disponibles. L'objectif est de mieux adapter l'information de conscience de groupe à l'individu et à ses préférences, de lui fournir une information qui soit la plus pertinente pour lui dans son contexte courant, afin qu'il puisse ainsi optimiser ses propres actions dans le groupe. Il s'agit de cibler l'individu pour mieux aider le groupe. Nous croyons qu'un utilisateur nomade pourra mieux organiser ses propres interventions dans le groupe, à la mesure du temps disponible et des opportunités, s'il dispose d'informations pertinentes lorsqu'il en demande. Il faut donc présenter à l'utilisateur des informations qui lui sont pertinentes vis-à-vis à la fois de son contexte actuel et de ses préférences pour ce contexte.

Le processus que nous proposons repose sur l'hypothèse selon laquelle un utilisateur nomade accède à un collecticiel donné majoritairement à partir d'un ensemble limité et plus ou moins défini de situations : de son bureau, de chez lui, de l'aéroport, avec son ordinateur portable, son ordinateur de poche, etc. À partir du moment où ces situations types sont identifiées, il est possible pour l'utilisateur de définir ses préférences pour chaque situation, et de couvrir ainsi la majorité des cas d'utilisation du système. Ceci est particulièrement intéressant pour les collecticiels sur le Web. La majorité des dispositifs mobiles disponibles à ce jour offre un accès Web. Cependant, cet accès est souvent coûteux et contraignant (par exemple, l'utilisation du Web à travers un téléphone cellulaire). Par conséquent, les utilisateurs emploient ces dispositifs habituellement pour des actions précises et attendent du système, en retour, des informations qui soient pertinentes pour eux.

Ainsi, le processus de filtrage que nous proposons est guidé par notre *modèle de contexte* (voir chapitre 5), lequel est utilisé afin d'indiquer ces situations types rencontrées par l'utilisateur, ainsi que pour représenter le *contexte courant* de cet utilisateur. Les préférences de l'utilisateur par rapport à ces situations sont représentées à travers un ensemble de *profils* qui servent notamment à indiquer la pertinence des informations par rapport à un contexte donné. Nous considérons que chaque utilisateur (ou le coordinateur de groupe, ou encore l'administrateur du système) pourra définir plusieurs profils pour les situations dans lesquelles il utilise le plus souvent le système. Dans ces profils, on exprime quels types d'information de conscience de groupe sont considérés comme pertinents pour un contexte donné, et l'organisation de ces types d'information. Les informations de conscience de groupe sont représentées à travers un *modèle de contenu* que nous proposons également dans ce chapitre. Ce modèle de contenu est formé autour du concept d'*événement* qui représente un élément d'information de conscience de groupe. De plus, l'organisation de ces informations se fait à l'aide d'un *modèle d'accès progressif* qui est utilisé au sein des profils afin d'exprimer les conditions de filtrage et d'organiser l'information de conscience de groupe en plusieurs niveaux de détails. À travers la combinaison de ces modèles, il est possible d'exprimer des contraintes telles que “si l’utilisateur se retrouve chez lui, alors telles classes d’information sont prioritaires et doivent s'organiser de telle manière”. Ainsi, chaque objet profil contient des conditions pour le filtrage des informations et est associé à au moins un objet `Description_de_Contexte`, qui décrit les conditions pour lesquelles le profil s’applique.

Par ailleurs, le processus de filtrage que nous proposons comporte deux étapes : la première étape identifie les profils que le système doit appliquer afin de sélectionner les informations de conscience de groupe. La deuxième étape consiste à ordonnancer les profils sélectionnés par ordre de priorité et à les appliquer, à travers la mise en œuvre des règles de filtrage définies par ces profils. Ces règles sont chargées de filtrer et d’organiser le contenu informationnel mis à la disposition de l’utilisateur par le système. De plus, ces étapes appliquent directement les opérations définies dans le chapitre 6, tant pour comparer ces instances et que pour déterminer l'ordre de priorité de chaque profil.

Ce chapitre s'organise comme suit : tout d'abord, dans la section 7.2, nous présentons les modèles qui, en plus du modèle de contexte, interagissent au sein de ce processus. Ensuite, nous montrons, dans la section 7.3, les étapes qui forment ce processus. Enfin, dans la section 7.4, nous présentons nos conclusions à propos de ce processus de filtrage.

7.2 Modèles Sous-jacents

Nous proposons dans ce chapitre un processus de filtrage guidé par le contexte et basé sur un ensemble de modèles interconnectés. Le premier de ces modèles est le modèle de contexte que nous avons présenté dans le chapitre 5. Les autres modèles sont présentés dans les sections suivantes : le modèle d'accès progressif, utilisé pour notamment organiser l'information de conscience en plusieurs niveaux de détails est présenté dans la section 7.2.1 ; le modèle de contenu, qui représente l'information de conscience de groupe, est introduit dans la section 7.2.2 ; et dans la section 7.2.3, nous discutons le modèle de profil, qui représente les préférences de l'utilisateur.

7.2.1 Modèle d'Accès Progressif

Le filtrage de l'information que nous proposons repose sur la notion d'*accès progressif*. L'accès progressif est fondé sur le postulat selon lequel un utilisateur n'a pas *tout le temps* besoin d'accéder à *toute* l'information disponible. L'objectif visé est que le système soit capable de fournir *progressivement* une information personnalisée aux utilisateurs : dans un premier temps, l'utilisateur accède aux informations qui sont essentielles pour lui. Puis, de façon graduelle, le système lui donne la possibilité, s'il le souhaite, d'accéder à d'autres informations. Le ***Modèle d'Accès Progressif (MAP)*** que nous utilisons ici a été proposé par Villanova-Oliver [Villanova 2002]. Il s'agit d'un modèle générique, décrit en UML, qui permet l'organisation d'un modèle de données en multiples niveaux de détail, selon les préférences de l'utilisateur. Le MAP utilise des concepts de base, indépendants de tout type particulier de système d'information (coopératifs ou non) et de tout domaine d'application spécifique. Nous rappelons brièvement ces concepts ci-dessous (une description complète peut être trouvée dans [Villanova 2002]).

7.2.1.1 Définitions générales

La notion d'accès progressif est directement liée à celle d'***Entité Masquable*** (*EM*). Une EM est un ensemble d'au moins deux éléments (la cardinalité de l'ensemble EM est donnée par $|EM| \geq 2$) sur lequel on souhaite mettre en place un accès progressif. L'accès progressif sur une EM repose sur la définition de ***Représentations d'Entité Masquable*** (*REM*) associées à cette EM. En règle générale, les REM sont des sous-ensembles de l'EM ordonnés par la relation d'inclusion ensembliste. Ces REM sont dites *extensionnelles* lorsqu'elles agissent sur l'*extension* de l'EM, c'est-à-dire sur ses éléments. Dans le cas où l'*EM* correspond à un ensemble de données structurées, on peut également lui associer des REM *intensionnelles*, qui agissent sur la structure de l'EM. Les REM intensionnelles définissent des sous-ensembles de *l'intension* de l'EM, également ordonnés par la relation d'inclusion ensembliste. Dans le cas où l'EM est une classe dans un modèle à objets, les REM extensionnelles agissent sur les instances de la classe (l'extension de la classe), tandis que les REM intensionnelles agissent sur la structure de la classe, c'est-à-dire sur les attributs qui composent l'intension de la classe. Quelle que soit sa nature (extensionnelle ou intensionnelle), la REM est associée à un *niveau de détail*. Ainsi, la REM_i est définie comme la représentation de niveau de détail *i* avec $1 \leq i \leq max$, où max est le niveau maximum de détail défini pour l'EM : $REM_1 = \{e_{11}, e_{12}, \ldots, e_{1l}\}$, … , $REM_i = REM_{i-1} \cup \{e_{i1}, e_{i2}, \ldots, e_{il}\}$, $REM_{i+1} = REM_i \cup \{e_{i+11}, e_{i+12}, \ldots, e_{i+1l}\}$, et ainsi de suite.

La Figure 46 présente un exemple générique, dans lequel on retrouve une EM qui comporte trois REM associées. Les règles (voir [Villanova 2002]) imposent que toute REM_{i+1} associée au niveau de détail $i+1$, $1 \leq i \leq max-1$, quel que soit son type, contient, au moins, un élément de plus que REM_i. Une ***stratification*** correspond à une organisation d'une EM en REM, comme l'illustre la Figure 46. De plus, il est important d'observer que nous pouvons définir plusieurs stratifications différentes pour une EM donnée. Par ailleurs, les stratifications définies sur une EM n'impliquent pas obligatoirement tous les éléments de l'EM. Certains éléments peuvent être exclus de la stratification (dans ce cas, aucune REM ne les contient), et par conséquent, ils ne seront pas mis à disposition de l'utilisateur. Par exemple, dans la Figure 46, un des éléments de l'EM (celui symbolisé par un losange) n'appartient à aucune REM. Il sera donc caché de l'utilisateur.

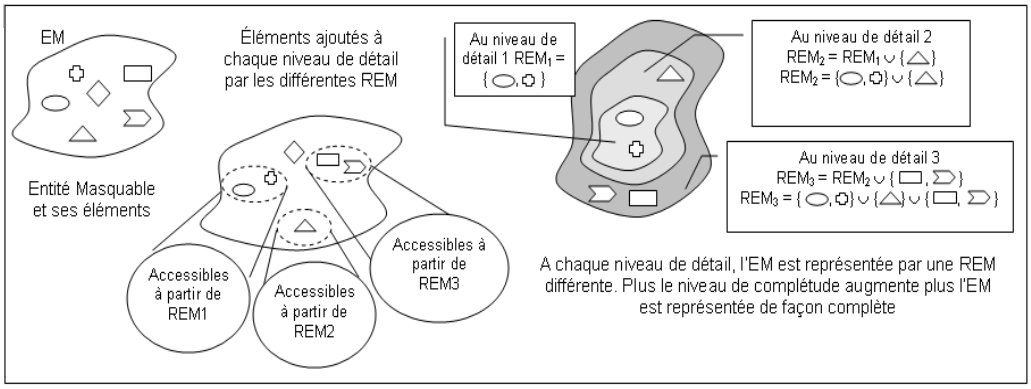


Figure 46. Une Entité Masquable et trois Représentations d'Entité Masquable (d'après [Villanova 2002]).

Le Modèle d'Accès Progressif (MAP) est défini par Villanova-Oliver [Villanova 2002] comme un diagramme de classes UML générique (voir Figure 47) qui exprime les règles de bonne formation des stratifications pour une entité masquable donnée. La classe *Stratification* est présentée comme une agrégation impliquant au moins deux instances de la classe *Représentation d'Entité Masquable*, celles-ci étant ordonnées. Chaque instance de cette classe est liée, par l'association *ajoute*, à une ou plusieurs instances de la classe *Elément d'Entité Masquable*, lesquelles représentent les éléments de l'EM qui sont ajoutés à la REM_i de niveau de détail *i*. La restriction *{sous-ensemble}* garantit que les éléments ajoutés à la REM appartiennent à l'ensemble d'éléments qui correspondent à l'EM.

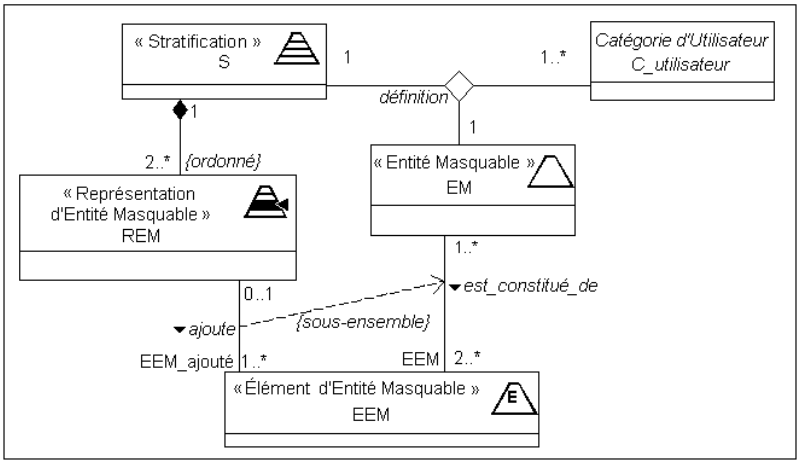


Figure 47. Modèle d'Accès Progressif, décrit à l'aide des stéréotypes en UML (d'après [Villanova 2002]).

Une caractéristique importante du MAP est l'association ternaire *définition* qui met en relation les classes *Stratification* (S), *Entité Masquable* (EM) et *Catégorie d'Utilisateur* (U). Ainsi, chaque utilisateur peut bénéficier d'un accès progressif personnalisé à une entité masquable donnée.

Par ailleurs, deux aspects du MAP sont à noter. D'une part, les EM peuvent être de différents types, de granularités diverses et être représentées dans des formalismes variés. Ainsi, un modèle de données exprimé sous la forme d'un diagramme de classes, mais aussi chaque constituant de ce modèle (une classe, une association…) peut être considéré comme une EM, et donc être stratifié (voir [Villanova 2002]). D'autre part, la classe *Catégorie d'Utilisateur* est une classe abstraite qui vise à connecter le MAP avec un modèle utilisateur

quel qu'il soit, permettant ainsi d'adopter les principes de l'accès progressif à des fins de personnalisation du système. Dans la section suivante, nous montrons comment nous exploitons ces aspects du MAP dans notre travail.

7.2.1.2 Opérations du MAP

L'accès progressif exploite deux opérations, proposées par Villanova-Oliver [Villanova 2002], qui permettent de passer d'un niveau de détail à l'autre : l'opération *Masquer* donne accès à la REM de niveau de détail immédiatement inférieur lorsqu'elle existe, tandis que l'opération *Dévoiler* permet d'atteindre la représentation de niveau de détail immédiatement supérieur, lorsqu'elle existe. Ces opérations sont définies comme suit :

- à partir d'une REM_i, au niveau de détail i, l'opération *Masquer* donne accès à la REM_{i-1} et à ses possibles prédécesseurs, au niveau $i-1$:

  Masquer $(REM_i)=REM_{i-1}$, avec $2 \leq i \leq max$

- à partir d'une REM_i, au niveau de détail i, l'opération *Dévoiler* donne accès à la REM_{i+1} et à ses possibles prédécesseurs, au niveau $i+1$:

  Dévoiler $(REM_i)=REM_{i+1}$, avec $1 \leq i \leq max-1$

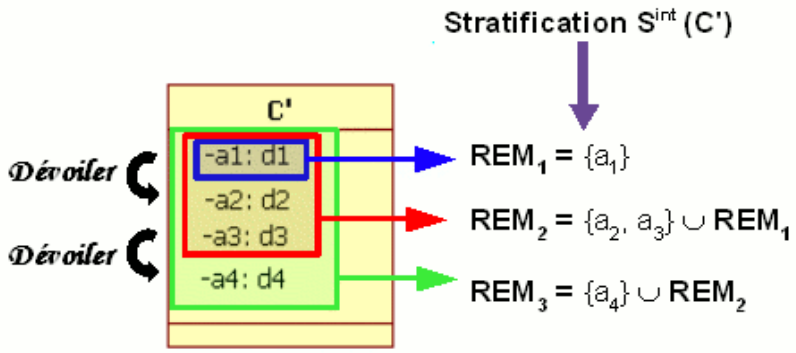


Figure 48. Exemple de stratification intensionnelle sur une classe.

La Figure 48 montre un exemple de l'application de ces opérations. Dans cet exemple, une stratification intensionnelle[84] S^{int} est composée par trois REM intensionnelles sur une classe C' (la classe C' est l'EM). Le premier niveau de cette stratification contient un seul élément ($REM_1 = \{a_1\}$), le deuxième niveau ajoute deux autres éléments ($REM_2 = \{a_2, a_3\} \cup REM_1$) et le troisième introduit encore un élément ($REM_3 = \{a_4\} \cup REM_2$). Ainsi, dans un premier temps, seul l'élément qui compose le premier niveau de détail est disponible. L'application de l'opération *Dévoiler* sur ce premier niveau nous offre alors l'accès aux éléments du deuxième niveau, ainsi qu'à l'élément du premier ($REM_2 = \{a_1, a_2, a_3\}$), tandis que les éléments du troisième niveau restent cachés. Ceux-ci ne seront disponibles qu'après l'application de l'opération *Dévoiler* sur le deuxième niveau de détail ($REM_3 = \{a_1, a_2, a_3, a_4\}$). D'autre part, l'application de l'opération *Masquer* sur le deuxième niveau va, au contraire, occulter les éléments des niveaux deux et trois, retournant au contenu du premier niveau ($REM_1 = \{a_1\}$).

[84]Une stratification intensionnelle sur une EM nommée C' est notée $S^{int}(C')$, ou simplement S^{int}, tandis qu'une stratification extensionnelle sur la même EM est notée $S^{ext}(C')$, ou simplement S^{ext}.

À partir de cet exemple, nous pouvons observer que lorsqu'on applique les opérations *Dévoiler* et *Masquer*, le contenu des niveaux précédents au niveau courant reste disponible. Par exemple, lorsque l'opération *Dévoiler* est appliquée, le contenu du niveau suivant est ajouté au contenu des niveaux précédents. En d'autres termes, l'utilisateur aura à sa disposition toutes les informations fournies par les niveaux précédents ainsi que les informations associées au niveau courant. Ceci est une conséquence de la définition même des REM (des sous-ensembles ordonnés par une relation d'inclusion ensembliste). Cependant, ce comportement peut s'avérer inopportun lorsqu'il s'agit d'un utilisateur nomade qui accède au système depuis un dispositif mobile. Par exemple, supposons qu'un utilisateur consulte son carnet d'adresses depuis son téléphone cellulaire. Dans le cas présent, l'EM correspond au carnet d'adresses, pour lequel peut être définie une stratification intensionnelle S^{int} composée de deux niveaux de détails : REM_1 = {nom, téléphone} et $REM_2 = REM_1 \cup$ {adresse}. Ainsi, au premier niveau de détail, cet utilisateur pourra consulter les noms et les numéros de téléphone de ses contacts (contenu de la REM_1). L'application de l'opération *Dévoiler* conduit l'utilisateur au deuxième niveau et lui donne alors accès aux adresses de ses contacts (contenu de la REM_2), en plus de leurs noms et numéros de téléphone. En considérant la taille d'écran limitée qui caractérise bien souvent les téléphones cellulaires, les informations appartenant à la REM_1 qui restent présentes lorsque l'utilisateur se trouve au niveau 2 peuvent surcharger inutilement l'espace d'affichage. Dans ce cas, un comportement plus adéquat consisterait à présenter à l'utilisateur *seulement* le contenu du niveau courant, c'est-à-dire à cacher les éléments issus des niveaux précédents.

Nous proposons dans ce but deux nouvelles opérations, qui s'appliquent particulièrement aux environnements mobiles. Ces opérations, *Avancer* et *Retourner*, contrairement aux opérations *Dévoiler* et *Masquer*, présentent séparément les informations de chaque niveau. En d'autres termes, l'opération *Avancer* ne préserve pas les informations des niveaux précédents lorsqu'elle présente le niveau d'après. L'opération *Retourner* cache les informations du niveau i lorsqu'elle retourne au niveau i-1, mais surtout, ne présente que les informations associées à ce niveau i-1 (et pas celles des niveaux inférieurs). Ces opérations sont définies comme suit :

- considérons, tout d'abord, une opération de base *Proper*(REM_i), qui isole les éléments ajoutés au niveau i :

$$Proper(REM_i) = \begin{cases} REM_i & si\ i = 1 \\ REM_i - REM_{i-1} & si\ 2 \leqslant i \leqslant max \end{cases}$$

- à partir d'une REM_i, au niveau de détail i, l'opération *Retourner* donne accès *seulement* aux éléments de la REM_{i-1}, au niveau i-1 :

$$Retourner(REM_i) = Proper(REM_{i-1}),\ \text{avec}\ 2 \leq i \leq max$$

- à partir d'une REM_i, au niveau de détail i, l'opération *Avancer* donne accès *seulement* aux éléments au REM_{i+1}, au niveau i+1 :

$$Avancer(REM_i) = Proper(REM_{i+1}),\ \text{avec}\ 1 \leq i \leq max-1$$

Ainsi, dans l'exemple précédent (Figure 48), l'application de l'opération *Avancer* à partir de la REM_1 nous donne accès aux éléments qui sont propres au deuxième niveau ({a_2, a_3}), tout en cachant les éléments du premier niveau, qui sont désormais inaccessibles ({a_1}). De même, lorsqu'on applique l'opération *Retourner* à partir du troisième niveau, seulement les éléments qui ont été ajoutés au deuxième niveau sont disponibles, les éléments du premier niveau restent indisponibles.

Par ailleurs, le modèle d'accès progressif, tel qu'il a été proposé (cf. Figure 47 et [Villanova 2002]), permet la définition de plusieurs stratifications sur une même EM pour un utilisateur donné. Il est donc possible que plus d'une stratification soit susceptible d'être appliquée à un moment donné. Typiquement, nous pouvons avoir des stratifications extensionnelles et des stratifications intensionnelles qui doivent être appliquées. Dans ce cas précis, Villanova-Oliver [Villanova 2002] défend que ces deux types de stratifications peuvent être combinés, puisqu'ils ont des objectifs distincts : la stratification en extension agit sur l'extension en masquant à certains niveaux (dans certaines REM) des éléments de l'EM ; la stratification en intension masque à certains niveaux des attributs de la structure de l'EM. La combinaison des stratifications intensionnelles et extensionnelles peut donc être ramenée à deux cas de base [Villanova 2002] : (*i*) on applique une stratification en intension S^{int} à partir d'un élément d'une stratification en extension d'une EM ; ou (*ii*) on applique une stratification en extension S^{ext} à partir d'un élément d'une stratification en intension d'une entité masquable. Dans le premier cas, l'idée est de commencer par masquer certains individus de la population de l'EM (en la stratifiant en extension), puis à partir d'un certain niveau de masquage de la population, masquer l'intension de l'EM (on applique donc la stratification en intension sur la stratification en extension). Dans le deuxième cas, on fait l'inverse, on applique la stratification en extension sur une stratification en intension.

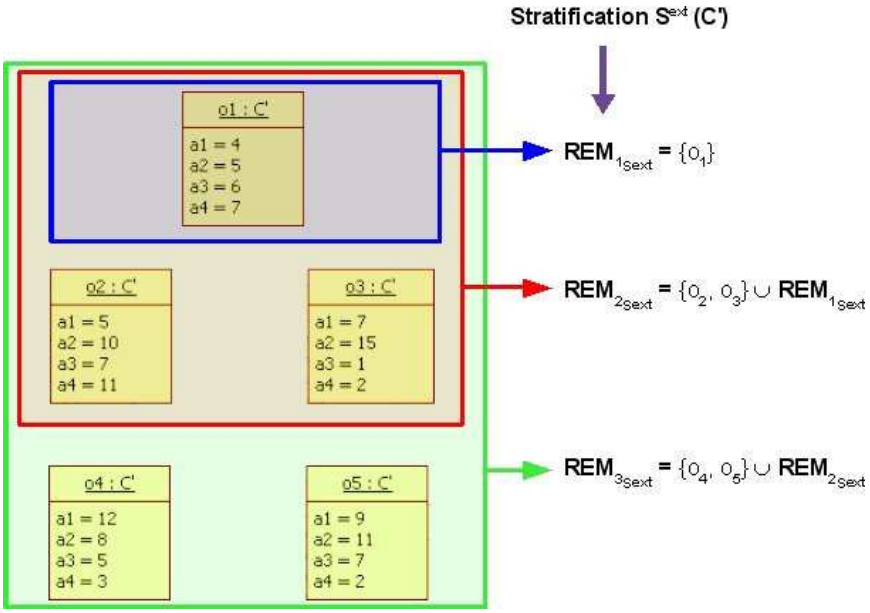


Figure 49. Exemple de stratification extensionnelle sur une classe.

À partir de ces deux cas de base, nous pouvons imaginer que les stratifications intensionnelles et extensionnelles peuvent évoluer de manière orthogonale et indépendante : on peut *avancer* dans une stratification en intension tout en gardant sa position (ou même *retourner*) dans une stratification en extension, et vice-versa (et ceci pour toute autre opération, non seulement *avancer*/ *retourner*). Par exemple, nous pouvons combiner la stratification en intension présentée sur la Figure 48 (S^{int} (C')) avec la stratification en extension (S^{ext} (C')) montrée dans la Figure 49, laquelle est définie sur la même entité masquable (la classe C'). Nous avons, dans ce cas, deux stratifications (S^{int}(C') = {REM_{1sint}, REM_{2sint}, REM_{3sint}} et S^{ext}(C') ={REM_{1sext}, REM_{2sext}, REM_{3sext}}) qui sont appliquées en parallèle. Ainsi, à partir du premier niveau, dans lequel nous avons disponible les informations relatives à l'attribut a_1 de l'instance o_1 (REM_{1sint} et REM_{1sext}), nous pouvons appliquer de manière indépendante les opérations citées ci-dessus. Par exemple, il est possible d'appliquer

l'opération *avancer* dans la stratification S^{ext}, ce qui donne accès au contenu de l'attribut a_1 pour les instances o_2 et o_3 (REM_{2Sext} et REM_{1Sint}), ou appliquer l'opération *dévoiler* sur la stratification S^{int}, ce qui donne accès au contenu des attributs a_1, a_2 et a_3 pour l'instance o_1 (REM_{2Sint} et REM_{1Sext}), ou encore d'appliquer les deux simultanément, ce qui donne l'accès au contenu des attributs a_1, a_2 et a_3 pour instances o_2 et o_3 (REM_{2Sint} et REM_{2Sext}).

Néanmoins, la question de l'application de plusieurs stratifications d'un même type reste entière. De manière similaire à Villanova-Oliver [Villanova 2002], nous avons proposé, dans [KirschPi 2003b], un processus d'application des plusieurs stratifications, mais ce processus se limite, en grande partie, à l'application séquentielle des stratifications en extension et en intension, et à l'union des stratifications d'un même type. Cependant, l'union des stratifications pose certains problèmes. Elle peut notamment générer des stratifications contentant trop d'éléments par niveau, ce qui est contraire à l'objectif même du modèle d'accès progressif. Par ailleurs, l'union peut également produire des stratifications qui ne satisfassent pas les besoins exprimés par l'utilisateur à travers les stratifications originelles. Une alternative à l'union des stratifications d'un même type est de demander directement à l'utilisateur quelle stratification, parmi les possibles, il préfère appliquer. Toutefois, cette option est loin d'être pratique pour les utilisateurs, ce qui décourage son utilisation avec des utilisateurs nomades, déjà soumis à plusieurs contraintes.

Une autre alternative consiste à ordonnancer et à appliquer dans l'ordre défini les stratifications d'un même type. Ceci permet de garantir que ce qui a été défini par chacune des stratifications sera respecté. L'ordre dans laquelle les stratifications sont ordonnancées dépend, bien évidemment, du système dans lequel le modèle d'accès progressif s'applique. Néanmoins, une fois que les stratifications sont organisées dans un ensemble ordonné, il est important de pouvoir naviguer dans cet ensemble. Pour cela, nous proposons deux nouvelles opérations nommées *Suivant* et *Précédent*, lesquelles permettent de passer d'une stratification à une autre (la suivante ou la précédente, respectivement) dans un ensemble de plusieurs stratifications. Ces opérations sont définies comme suit :

- étant donné un ensemble ordonné de stratification $\mathcal{S} = \{S_1, \ldots, S_n\}_{n \geq 2}$, dans lequel chaque stratification est composée par un ensemble également ordonné de REM :

  $S_i = \{REM_{1_{S_i}}, \ldots, REM_{max_{S_i}}\}$

- à partir d'une stratification S_i, à la position i de l'ensemble de stratifications $\mathcal{S}$, l'opération *Suivant* donne accès au premier niveau de la stratification S_{i+1}, quel que soit le niveau de détail de S_i :

  $Suivant(S_i) = REM_{1_{S\ i+1}}$, avec $1 \leq i \leq n-1$

- à partir d'une stratification S_i, à la position i de l'ensemble de stratifications $\mathcal{S}$, l'opération *Précédent* donne accès au dernier niveau de la stratification S_{i-1}, quel que soit le niveau de détail de S_i :

  $Précédent(S_i) = REM_{max_{S\ i-1}}$, avec $2 \leq i \leq n$

À l'aide de ces opérations, il est possible de passer directement d'une stratification à une autre, en sautant d'un niveau quelconque dans une stratification à un niveau (le premier ou le dernier, selon l'opération utilisée) de l'autre stratification. De plus, la notion d'un ensemble ordonné de stratifications n'implique pas que ces stratifications soient définies sur une seule EM. Chaque stratification peut impliquer une EM distincte des autres stratifications. Ceci garantit une souplesse supplémentaire pour le modèle d'accès progressif. Par exemple, nous

pouvons imaginer que la stratification $S^{ext}(C')$ présentée dans la Figure 49 peut être appliquée dans la séquence d'autres stratifications sur la même classe C', ou de stratifications définies sur d'autres classes du modèle. Une telle application en séquence n'interfère pas dans la définition de la stratification $S^{ext}(C')$.

7.2.1.3 Le MAP et la conscience de groupe

Dans ce travail, nous utilisons le *modèle d'accès progressif* (MAP) afin d'adapter l'information de conscience de groupe dans le cadre d'un processus de filtrage guidé par le contexte. La simple utilisation du MAP pour la conscience de groupe a été déjà explorée dans le passé [KirschPi 2003b]. Cependant, lors des travaux précédents, la connexion entre les préférences de l'utilisateur et le contexte de utilisation était absente. Les stratifications étaient associées directement à l'utilisateur ou aux rôles que l'utilisateur pouvait jouer dans le groupe, représentant ainsi tant les préférences de l'utilisateur que les besoins d'un rôle donné (voir [KirschPi 2003b]). Ces stratifications restaient inaltérées dans le cas d'un utilisateur nomade, malgré les possibles changement de contexte d'utilisation. Dans ce cas, les stratifications sont appliquées quel que soit le contexte dans lequel l'utilisateur se trouve.

Ce comportement est inadapté aux besoins d'un utilisateur nomade, puisque les préférences et les besoins de l'utilisateur peuvent varier en fonction de son contexte courant (cf. chapitre 4). Par conséquent, un même ensemble de stratifications, qui, dans une situation, semble parfaitement adapté aux attentes de l'utilisateur, peut s'avérer inadéquat dans d'autres situations. Par exemple, des stratifications définies en supposant l'utilisation d'un ordinateur fixe se montreront probablement inadéquates lors que l'utilisateur emploie un ordinateur de poche, dont les capacités d'affichage et de mémoire sont nettement réduites.

Dans le présent travail, nous pallions ce problème en associant le modèle d'accès progressif au modèle de contexte à travers la notion de profil (voir section 7.2.3), au sein d'un processus de filtrage guidé par ce même modèle de contexte. Néanmoins, avant de présenter le modèle de profil, il est important de discuter le modèle de contenu sur lequel le MAP est appliqué. Ce modèle, qui représente l'information de conscience de groupe, est discuté dans la prochaine section.

7.2.2 Modèle de contenu

Le processus de filtrage que nous proposons ici utilise un modèle de contenu spécifique pour représenter les informations de conscience de groupe. Ce modèle repose sur une approche basée sur la notion d'*événement*. Dans cette approche, adoptée par [Liechti 2000], [KirschPi 2003a] et [BSCW 2005], le mécanisme de conscience de groupe s'organise autour de la création et de la notification des événements. Un ***événement***[85] représente un élément d'information de conscience de groupe. De manière plus générale, un événement représente un ensemble d'informations sur un sujet donné relatif au processus de coopération. De façon schématique, un événement décrit une information relative à une action qui se déroule (ou qui s'est déroulée, ou encore qui est prévue) dans un groupe de travail. Il porte donc des informations concernant des entités qui participent au processus de coopération : par exemple, la fin d'une l'activité ou encore le fait qu'un membre du groupe se connecte. Ainsi, au fur et à mesure qu'un groupe développe son travail, que les acteurs de ce groupe interagissent à l'aide du collecticiel, les événements correspondants à ces actions sont générés

[85] À ne pas confondre avec le concept homonyme de la *programmation par événements*.

par le collecticiel. L'ensemble d'événements produits par le collecticiel correspond à l'ensemble d'informations de conscience de groupe disponibles.

L'objectif de ce modèle de contenu est de permettre la description des événements qui peuvent s'avérer utiles pour un utilisateur nomade participant au processus de coopération. Par exemple, l'arrivée d'un membre du groupe sur une certaine zone géographique peut donner lieu à la production de l'événement correspondant ; ou encore, dans le cas d'un processus d'édition collaborative, le fait de modifier un document peut être vu comme une classe d'événement, et une modification apportée à un rapport donné sera donc un objet de cette classe d'événement spécifique. L'ensemble des classes d'événements possibles est identifié par le concepteur du système selon les caractéristiques et les fonctionnalités du système, et les besoins attendus des utilisateurs.

Notre modèle de contenu est donc centré sur une classe abstraite nommée *Événement*, laquelle représente cette notion. Par ailleurs, nous proposons un certain nombre d'attributs pour cette classe que nous considérons comme élémentaires pour une description minimale de l'information de conscience de groupe dans un collecticiel. Néanmoins, il est important d'observer que cette classe doit être spécialisée par les concepteurs d'un collecticiel qui utilise ce modèle. Le concepteur doit redéfinir cette classe en considérant les informations qui peuvent s'avérer intéressantes pour les utilisateurs lorsqu'ils coopèrent à travers le système en question. Ainsi, l'ensemble que nous considérons comme minimal pour la description d'un événement est constitué des attributs suivants (voir Figure 50) :

- un `identifiant`, qui attribue un nom à l'événement (par exemple, *'présence Bernard'*) ;
- une `description`, qui décrit l'action représentée par l'événement (par exemple, *'Bernard est en ligne et se trouve dans le bureau D322'*) ;
- un champ `détails` qui décrit de manière plus détaillée l'action (par exemple, *'Bernard s'est connectée à 8h30 et se trouve au bureau D322, 3 étage - bâtiment D'*) ;
- un `intervalle` auquel se réfère l'information portée par l'événement (par exemple, l'intervalle dans lequel l'utilisateur Bernard est en ligne : *de 8h30 du matin jusqu'à maintenant*) ;
- un champ `médias` qui sert notamment à apporter des informations complémentaires à l'événement (par exemple, *une photo de Bernard)*.

Nous considérons également que chaque événement concerne un ou plusieurs éléments de notre modèle de contexte, puisqu'un événement véhicule une information à propos de ces éléments. La relation existant entre l'événement et les éléments qu'il concerne se présente dans le modèle de contenu sous la forme d'une association entre la classe `Événement` et un ou plusieurs objets `Élément_de_Contexte` (voir association `concerne` dans la Figure 50). Par exemple, dans le cas de l'événement « *présence Bernard* » décrit ci-dessus, nous pouvons imaginer que cet événement se réfère à l'objet de la classe `Membre` qui représente l'utilisateur `Bernard`.

De plus, nous considérons que chaque objet de la classe (ou d'une sous-classe de) `Événement` est associé à un objet `Description_de_Contexte` (voir association `est_produit_dans` dans la Figure 50), qui représente le contexte dans lequel cet événement a été (ou devra être) produit. Puisque ce modèle de contenu vise la conception de collecticiels en tant que systèmes sensibles au contexte, nous pouvons imaginer que la réalisation d'une action dans un tel collecticiel peut être attachée à un contexte donné, notamment celui de l'auteur de l'action.

Dans le cas de l'exemple précédent, l'événement « *présence Bernard* » peut être associé à la localisation de Bernard au moment de la création de l'événement (voir Figure 50).

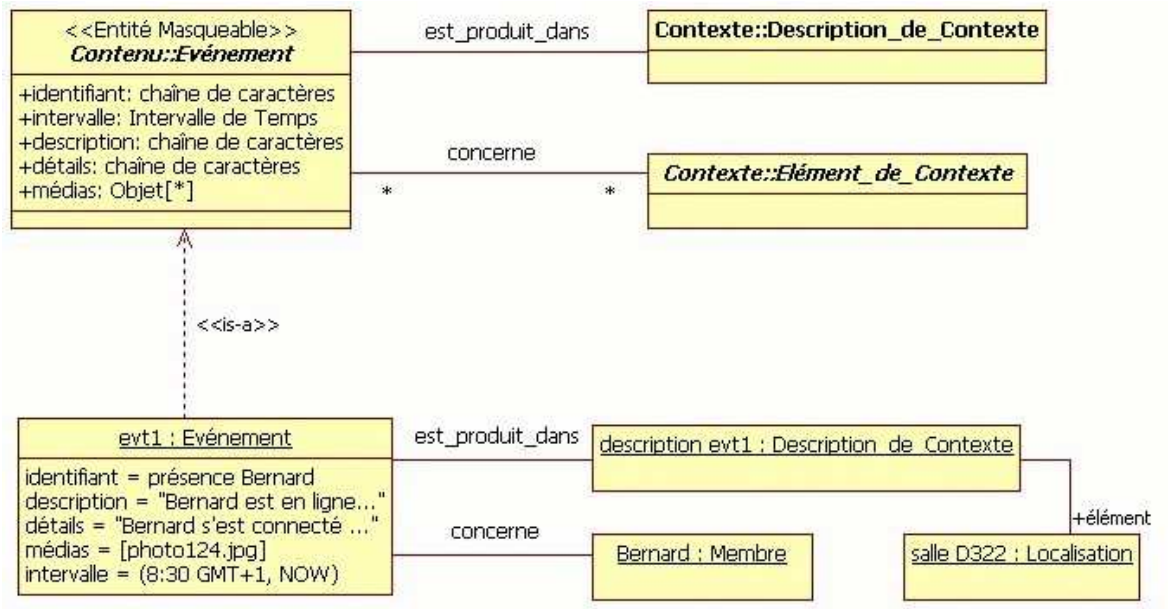


Figure 50. La classe événement représentant un type d'information de conscience de groupe et un exemple d'instance de cette classe.

Outre un événement pour indiquer la présence d'un membre du groupe, comme l'exemple ci-dessus l'indique, d'autres classes d'événements sont possibles selon le type de collecticiel. Par exemple, un collecticiel pour le partage de données peut définir une classe d'événements pour indiquer l'ajout d'un nouvel `Objet_Partagé`, ou encore une autre classe pour indiquer qu'une nouvelle version d'une ressource est disponible. D'ailleurs, on retrouve ces deux classes d'événement aussi bien sur le collecticiel *BSCW* [BSCW 2005] que sur *LibreSource* [Forest 2005a], deux collecticiels pour le partage de documents. Celles-ci ne sont pas les seules classes d'événement possibles. D'autres types de collecticiels auront besoin d'autres classes. Par exemple, un système de « geonotes » (respectivement un collecticiel qui utilise des annotations pour la communication asynchrone entre les membres du groupe) peut définir une classe pour indiquer qu'une nouvelle note est disponible sur une certaine zone géographique (respectivement sur une ressource donnée). En somme, pour chaque collecticiel, il est possible de définir plusieurs classes d'événements. Chacune de ces classes est une spécialisation de la classe `Événement`, proposée dans ce modèle. Cette classe est au centre du mécanisme de conscience de groupe conçu à l'aide de ce modèle.

Par ailleurs, en vue de l'application du Modèle d'Accès Progressif, nous considérons la classe `Événement`, ainsi que ses sous-classes, comme des *Entités Masquables potentielles.* En d'autres termes, les stratifications seront définies pour cette (ces) classe(s) et ses (leurs) instances. Nous pouvons, conformément à la spécification du MAP, définir aussi bien des stratifications intensionnelles que des stratifications extensionnelles. Par exemple, nous pouvons définir une stratification intensionnelle sur l'ensemble d'attributs de la classe `Événement` S_1^{int} = {{*identifiant, description*}, {*intervalle, détails*}, {médias}}, ou encore une stratification en extension qui organise les instances de cette classe selon la valeur de l'attribut `intervalle` S_2^{ext} = {{*événement.intervalle pendant* JOURNEE}, {*événement.intervalle pendant* SEMAINE}}. Dans la section suivante, nous présentons le modèle de profil qui combine ce modèle de contenu au MAP.

7.2.3 Modèle de profil

Le modèle de profil que nous proposons sert à représenter les préférences (et les besoins) de l'utilisateur vis-à-vis de l'information de conscience de groupe. L'objectif est d'associer les préférences à des situations précises afin qu'on puisse exprimer quelles classes d'information de conscience de groupe sont pertinentes pour l'utilisateur et dans quelles situations. Il s'agit de permettre des constructions du genre « lorsque l'utilisateur se trouve dans telle ou telle situation, il préfère que telle et telle information lui soit délivrée, organisée de cette manière ».

Les préférences des utilisateurs sont représentées, dans ce modèle, à travers la notion de ***profil***. Un profil représente les préférences et les contraintes que le système doit satisfaire pour un utilisateur donné dans un certain contexte. Pour cela, un profil est constitué notamment par la description d'un ou plusieurs *contexte(s) potentiel(s)* et de la spécification de *règles de filtrage*. Un contexte potentiel (aussi appelé *contexte d'application*) décrit une situation dans laquelle l'utilisateur peut se trouver au moment où il accède au système. Les préférences exprimées par le profil s'appliquent à ces contextes potentiels : ils représentent les situations dans lesquelles les règles de filtrages sont valides et doivent être utilisées. Les règles de filtrage correspondent donc à celles à appliquer lorsque le contexte courant d'utilisation observé concorde avec au moins un des contextes potentiels décrits par le profil. Cependant, ces règles ne sont pas explicitement portées par une classe dans ce modèle. Ces règles résultent de la combinaison de plusieurs éléments, dont un ensemble de *stratifications* et un ensemble d'*événements*. Elles reflètent les préférences du propriétaire du profil par rapport aux contextes d'application donnés. En d'autres termes, un profil décrit quelles sont les informations attendues par ou souhaitables pour l'utilisateur (et leur organisation) lorsqu'il se trouve dans une situation donnée.

Cette définition de la notion de profil est fondée sur l'hypothèse selon laquelle les préférences d'un utilisateur nomade, par rapport aux informations de conscience de groupe, peuvent changer en fonction du contexte d'utilisation. Ces différentes préférences sont, pour nous, une conséquence directe de la variation du contexte d'utilisation et des différents besoins qui peuvent en découler. Par exemple, nous pouvons imaginer qu'un même utilisateur n'aura pas les mêmes besoins (ni les mêmes préférences) lorsqu'il accède à un système à partir d'un ordinateur fixe dans son bureau, et lorsqu'il accède au même système à partir d'un ordinateur de poche dans le métro, ou encore lorsqu'il le fait à partir de son ordinateur portable à l'aéroport. Ces environnements sont très distincts et, naturellement, l'utilisateur développe différentes préférences pour chacun au fur et à mesure qu'il s'habitue à ces situations. Nous croyons qu'un utilisateur nomade acquiert souvent l'habitude d'utiliser un collecticiel majoritairement dans des situations connues et qui peuvent être caractérisées par certains éléments de contexte (cf. chapitre 4). Par exemple, un utilisateur qui voyage souvent peut prendre facilement l'habitude d'accéder au système à partir de l'aéroport ou d'une gare.

À partir du moment où l'utilisateur développe des habitudes pour des situations qu'il peut caractériser, l'utilisateur peut définir des profils à travers lesquels il peut exprimer ses préférences pour ces situations. Ceci permet, d'une part, d'impliquer davantage l'utilisateur dans le processus d'adaptation, puisque c'est lui qui exprime quelles informations il considère comme pertinentes lorsqu'il se trouve dans un contexte donné et leur organisation. D'autre part, la définition des profils pour les situations le plus souvent rencontrées par l'utilisateur permet une adaptation plus appropriée du système, tant au contexte de l'utilisateur qu'à ses préférences, dans la majorité des cas d'utilisation. Nous pouvons donc nous attendre à ce que le système satisfasse plus souvent les attentes de l'utilisateur.

Ainsi, dans le modèle que nous proposons, les profils sont des instances d'une classe appelée `Profil` (voir Figure 51 et Figure 52). De manière générale, nous pouvons percevoir chaque profil comme un ensemble composé par :

a) un `propriétaire` qui établit le profil (association `est_établi_par` dans la Figure 51) ;

b) au moins un `contexte_d'application`, qui représente les contextes potentiels associés au profil auxquels on comparera le contexte courant de l'utilisateur afin de juger l'applicabilité (ou validité) du profil et, finalement, de sa sélection ou non lors de la première étape du processus de filtrage (cf. section 7.3).

c) l'ensemble de sous-classes de la classe `Événement` dont les instances peuvent être sélectionnées (on dit que le profil *est abonné à* ces événements), traduisant ainsi le contenu informationnel considéré comme pertinent par le propriétaire du profil.

d) l'ensemble des `stratifications` définies par le propriétaire du profil. Ces stratifications ainsi que les événements abonnés par le profil constituent les *règles de filtrage* de celui-ci (voir les associations `est_abonné_à` et `est_ordonné_par` dans la Figure 52).

e) l'ensemble de conditions applicables à l'ensemble des événements retenus (Figure 53).

Figure 51. Un profil est établi par son propriétaire et s'applique à un contexte donné.

Comme le montre la Figure 52, l'association `est_abonné_à` met en relation le profil et les événements auxquels le profil est abonné, tandis que l'association `est_ordonné_par` met en relation le profil et les stratifications définies pour ces événements, et l'élément `cible` de ces stratifications. Ainsi, un profil décrit, pour un `contexte_d'application` donné, les `stratifications` définies par le `propriétaire`, lesquelles organisent les `événements` considérés comme pertinents dans ce contexte. Par le biais de la multiplicité de ces associations, un utilisateur (représenté par un objet `Membre`) peut disposer de plusieurs stratifications pour une classe d'`Événement` donnée en les associant à différents profils (et donc à divers contextes d'application). De même, il peut définir plusieurs stratifications (en extension et en intension) agissant sur les événements auxquels un profil donné est abonné. De plus, il convient d'observer dans la Figure 52 que les stratifications associées à un profil sont, par définition, ordonnées. En d'autres termes, au moment de la définition du profil, les stratifications sont ordonnancées selon les préférences du `propriétaire` du `profil`.

Le `propriétaire` d'un profil (voir association `est_établi_par` dans la Figure 51) est celui qui détient tous les droits sur ce profil. C'est lui qui établit le profil, en définissant notamment les stratifications qui le composent. Ces stratifications sont définies en considérant la classe `Événement` et ses sous-classes comme leurs *entités masquables* (voir association `est_ordonné_par` dans la Figure 52). Par ailleurs, ces stratifications visent l'élément `cible`, qui correspond à la notion de « *catégorie d'utilisateur* » du modèle d'accès progressif (cf. section

7.2.1). Autrement dit, les stratifications sont pensées pour cet élément, qui devra recevoir les informations sélectionnées par l'ensemble de stratifications. Cette cible est, le plus souvent, un objet de la classe `Membre` (c'est-à-dire, un *utilisateur*). De plus, la `cible` et le `propriétaire` peuvent correspondre à un même objet. C'est d'ailleurs le cas lorsque c'est l'utilisateur lui-même qui définit ses propres profils. Néanmoins, ceci n'est pas obligatoire. Le modèle de profil prévoit la possibilité que la `cible` et le `propriétaire` soient représentés par deux objets distincts. Nous pouvons avoir un propriétaire, typiquement un membre du groupe avec un rôle particulier (le coordinateur du groupe, ou l'administrateur du système), qui définit un profil pour quelqu'un d'autre (pour un rôle ou pour un groupe donné, ou encore pour un certain dispositif).

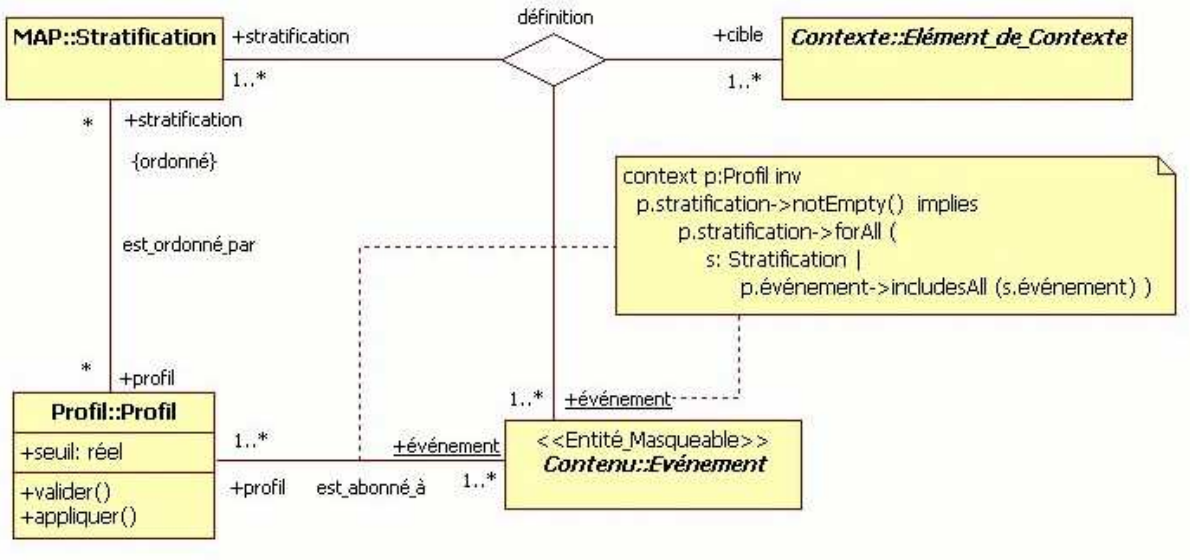


Figure 52. Un profil est abonné à un ensemble d'événements, organisés selon un ensemble de stratifications définies pour un élément cible.

L'idée est que l'administrateur du système, mais aussi chaque utilisateur ou coordinateur de groupe, puisse définir de tels profils (les contextes d'applications et les règles associées). Ainsi, chaque utilisateur peut éventuellement définir quelle information de conscience de groupe il considère comme pertinente et dans quelles circonstances elle l'est, puis l'organiser en plusieurs niveaux de détail. De même, l'administrateur du système peut proposer des profils pour les dispositifs admis par le système, selon les capacités de chaque dispositif, ou encore définir des profils spécifiques à certains rôles, tel que le rôle de coordinateur d'un groupe, selon les besoin de ces rôles par rapport à l'information de conscience de groupe. Dans le premier cas, le propriétaire et la cible sont représentés par le même objet, l'utilisateur. Dans les autres cas, nous avons un propriétaire (l'administrateur du système) qui définit des profils pour différentes cibles (un dispositif dans l'un, un rôle dans l'autre exemple).

Par ailleurs, on observe, par les exemples ci-dessus, que le profil n'est pas un concept dont nous restreignons l'utilisation aux seuls utilisateurs : un profil peut s'appliquer à tout autre élément (groupe, rôle, dispositif…) de notre modèle de contexte pour décrire des règles de filtrage à appliquer dans des contextes prédéfinis. Le modèle de profil, à travers les associations `est_établi_par` (Figure 51) et `définition` (Figure 52), définit aussi bien le `propriétaire` que la `cible` d'un profil comme étant un `Élément_de_Contexte`, et non nécessairement un objet de la classe `Membre`. Par conséquent, il est possible d'exprimer, à

travers les profils, des contraintes que le processus de filtrage doit respecter pour un élément quelconque et ceci par rapport à un contexte d'utilisation spécifique (le contexte d'application).

Ainsi, nous pouvons imaginer que, par exemple, le concepteur d'un éditeur coopératif sur le Web (comme *AllianceWeb* [Salcedo 1998] ou encore celui proposé par [Guerrero 2004]) peut définir plusieurs classes d'événement, parmi lesquelles une classe appelée « *description document* », dont les instances décrivent les documents partagés par le groupe, une classe nommée « *modifications document* », dont les instances décrivent les modifications réalisées sur un document, et une classe « *commentaire* », dont les instances contiennent des commentaires qui ont été formulés par les membres groupe sur un document donné. Dans ce cas, l'utilisateur 'Alain' peut définir un profil qui s'applique lorsqu'il travaille avec son ordinateur de poche (contexte d'application du profil) et qui est abonné à la troisième classe d'événement (*commentaire*), dont les instances sont organisées selon la stratification S_1^{int} = {{*identifiant, description*}, {*intervalle, détails*}, {*médias*}}. On peut également imaginer que, dans ce même système, l'administrateur peut définir un profil spécifique au rôle de coordinateur de l'équipe (la cible du profil est donc un objet de la classe `Rôle`), lequel s'applique lorsque l'utilisateur qui joue ce rôle est engagé dans une activité d'édition (le contexte d'application contient ici un objet de la classe `Activité` représentant l'édition d'un document). Un tel profil peut être abonné aux classes « *modifications document* » et « *commentaire* » et les organiser selon les stratifications S_2^{int} = {{*identifiant, intervalle*}, {*détails*}, {*médias*}} et S_3^{ext} = {{*événement.intervalle pendant* JOURNEE}, {*événement.intervalle pendant* SEMAINE}}.

La possibilité de définir des profils propres à des rôles spécifiques est particulièrement intéressante pour les collecticiels, car la bonne exécution d'un rôle dans une équipe demande des informations de conscience de groupe qui soient appropriées aux droits et aux responsabilités liées au rôle. En d'autres termes, chaque rôle a des besoins distincts en ce qui concerne le mécanisme de conscience de groupe (voir section 2.2 et [KirschPi 2001]). Ceci est le cas notamment du coordinateur d'une équipe. Un coordinateur (ou le *leader*) d'une équipe est celui qui doit organiser la coopération et doit veiller au bon fonctionnement de l'équipe. Sa responsabilité est de guider le groupe vers l'obtention d'un résultat qui satisfait les objectifs prédéfinis par ce même groupe. Par conséquent, celui qui joue le rôle de coordinateur nécessite des informations de conscience de groupe qui lui permettent d'avoir une vue globale de l'état de la coopération, ce qui n'est pas forcément nécessaire à tous les autres participants.

Outre les stratifications et l'abonnement aux classes d'événements, le modèle de profil permet également l'expression des conditions, qu'on dit contextuelles, sur les éléments de contexte liés aux événements. Selon le modèle de contenu présenté dans la section 7.2.2, chaque événement est produit dans un contexte donné et concerne un certain nombre d'éléments de contexte. Les conditions contextuelles exploitent ces relations entre événement et contexte. Ainsi, alors que les stratifications associées à un profil se concentrent sur la classe `Événement` (tant en intension qu'en extension), les conditions contextuelles se concentrent uniquement sur les associations `est_produit_dans` et `concerne` qui lient les *événements* au modèle de contexte (voir Figure 50).

Nous avons défini deux types de *conditions contextuelles* (voir Figure 53). Une première condition, nommée *condition de production*, s'applique sur le contexte de production d'un événement. Cette condition détermine que, pour qu'une instance d'événement donnée soit sélectionnée par le profil, il faut que, en plus de l'abonnement à sa classe d'appartenance, l'instance soit produite dans un contexte qui correspond à celui exprimé par la condition de production. Autrement dit, il faut que la situation décrite par le contexte de production soit

également rencontrée dans le contexte de production de l'événement. Cette condition de production est représentée, dans le modèle de profil, par l'association « `exige_contexte` » (voir Figure 53), laquelle lie un `Profil` à une `Description_de_contexte`.

La deuxième condition contextuelle que nous avons définie agit sur les éléments de contexte concernés par un événement. À travers cette condition, un profil peut demander la présence de certains éléments de contexte dans l'action décrite par l'événement. En d'autres termes, pour qu'une instance d'événement soit sélectionnée, elle doit concerner les éléments décrits dans cette condition, c'est-à-dire, l'événement doit être associé à ces éléments. De la même manière que pour la condition de production, cette condition sur les éléments s'exprime, dans le modèle de profil, à travers une association « `demande` » qui lie la classe `Profil` à la classe `Élément_de_Contexte` (voir Figure 53).

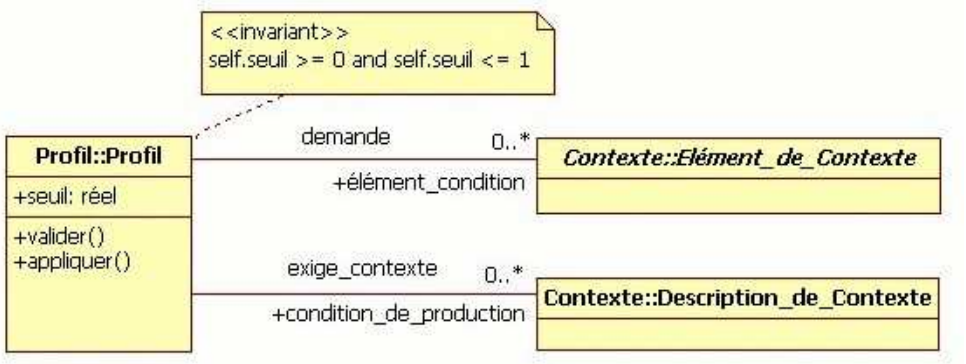


Figure 53. Conditions contextuelles établies par les profils.

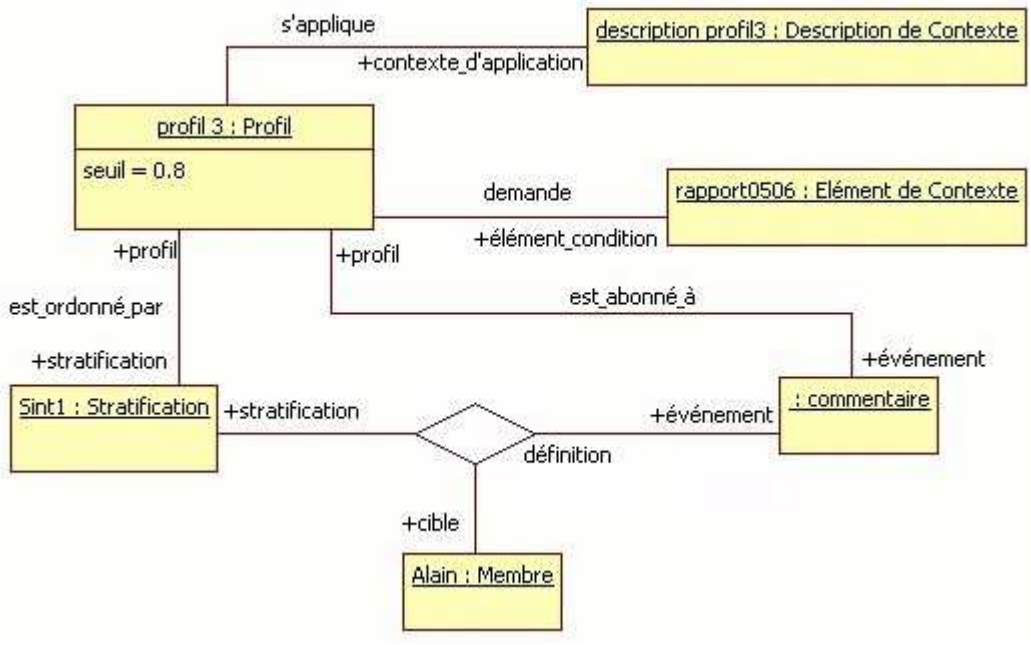


Figure 54. Exemple de profil défini pour l'utilisateur Alain comptant une condition contextuelle.

La définition de ces conditions contextuelles permet la sélection des événements en fonction de leur contexte. Nous pouvons ainsi exprimer des conditions indiquant que les événements doivent concerner un objet partagé donné (un rapport, un plan partagé par le groupe...), ou encore être produit dans une certaine localisation (une salle, une région en

particulier) ou par un certain collègue. Nous pouvons imaginer, par exemple, que l'utilisateur '*Alain*' peut définir un profil, montré dans la Figure 54, qui s'applique lorsqu'il se trouve avec son ordinateur portable dans la salle de réunion (contexte d'application représenté précédemment par la Figure 43, page 130) et qui sélectionne, selon une stratification S_1^{int}, les événements de la classe « *commentaire* » concernant le rapport sur lequel il travaille. Ceci signifie que seulement les commentaires formulés par les collègues qui concernent ce document en particulier devront être mis à disposition d'Alain, suivant l'ordre défini par la stratification S_1^{int}. Les commentaires concernant d'autres documents ne doivent pas être sélectionnés par ce profil. Nous pouvons aussi imaginer la définition d'un profil qui choisit seulement les instances d'une classe d'événement « *présence* » qui ont été produits dans une localisation donnée. Dans ce cas, le profil s'intéresse à la présence des collègues à une certaine localisation (par exemple, les médecins dans l'hôpital, dans le cas d'un système comme celui proposé par [Muñoz 2003]).

Par ailleurs, on observe dans la Figure 53 que la multiplicité lié aux associations `exige_contexte` et `demande` n'impose pas que des conditions contextuelles soient définies pour tous les profils. Ceci est nécessaire puisque ces conditions représentent des contraintes fortes sur les événements. Leur sémantique peut être comparée à un « *et* » logique qui s'ajoute aux règles déterminées par les stratifications et par l'abonnement du profil : pour qu'un événement soit choisi par le profil, il doit appartenir à l'extension d'une classe abonnée par le profil *et* satisfaire, à la fois, les conditions de production *et* les conditions sur les éléments. On observe également dans la Figure 53 la présence d'un attribut `seuil` dans la classe `Profil`. Cet attribut s'applique aux mesures de similarité et aux autres opérations définies sur le modèle de contexte (voir chapitre 6), lors de leur utilisation pour de telles conditions contextuelles, pouvant être exprimées à l'aide de ce modèle.

7.3 Processus de Filtrage

Le processus de filtrage guidé par le contexte que nous proposons pour adapter l'information de conscience de groupe repose sur les modèles sous-jacents présentés ci-dessus (voir Section 7.2) et sur le modèle de contexte (voir chapitre 5). Ce dernier est utilisé de deux façons. D'une part, il permet de représenter le *contexte courant* de l'utilisateur. D'autre part, il est utilisé par les autres modèles, notamment pour la description du (ou des) *contexte(s) d'application* de chaque `Profil`, lequel exprime également des *règles de filtrage* (cf. section 7.2.3). En se basant sur ces modèles, l'objectif du processus de filtrage est de filtrer les informations de conscience de groupe (les `Événements`) disponibles selon le contexte courant de l'utilisateur et selon ses préférences pour ce contexte. Ce processus cherche à sélectionner et à organiser les événements générés par le système en fonction de leur pertinence pour l'utilisateur, cette pertinence étant déterminée par la spécification des profils pour le contexte actuel. Nous espérons ainsi mettre à disposition de l'utilisateur seulement les événements qu'il considère comme pertinents, étant donnée la situation dans laquelle il se trouve.

De plus, le processus de filtrage guidé par le contexte que nous proposons vise notamment les collecticiels sur le Web, car ces systèmes doivent, dès maintenant, être munis d'un tel type de filtrage. Ce processus est donc conçu pour être implanté au sein du mécanisme de conscience de groupe du collecticiel. Les collecticiels sur le Web, comme tout système Web, repose notamment sur une architecture client-serveur, qui suppose un serveur dit « lourd », c'est-à-dire, un serveur avec d'importantes capacités de calcul et de mémoire. D'ailleurs, l'utilisation croissante des dispositifs mobiles aux capacités réduites renforce ce

scénario d'un client léger face à un serveur lourd, qui s'occupe de la plupart des calculs. C'est donc le serveur qui accueille le processus de filtrage.

Ce processus de filtrage s'organise en deux étapes. Dans la première étape, le processus de filtrage sélectionne, parmi tous les profils disponibles, ceux dont l'application est possible. C'est-à-dire, ceux dont le contexte d'application correspond au contexte courant de l'utilisateur. Ces profils sélectionnés, nommés « valides », sont ensuite exploités à la seconde étape du processus. Celle-ci procède à l'ordonnancement, par priorité, des profils et à l'application des règles qui sont décrites dans chaque profil (à savoir, l'abonnement aux événements, les stratifications et les conditions contextuelles).

Le reste de cette section s'organise comme suit : la section 7.3.1 présente la première étape du processus de filtrage, tandis que la seconde est présentée dans la section 7.3.2.

7.3.1 Étape 1 – sélection des profils

La première étape du processus de filtrage consiste à sélectionner des profils qui sont valides par rapport au contexte courant de l'utilisateur. En d'autres termes, nous cherchons, parmi les profils disponibles pour l'utilisateur, ceux qui sont applicables à la situation dans laquelle se trouve cet utilisateur. Les profils disponibles sont, en principe, ceux dont l'utilisateur en question est la cible, ainsi que tous les profils dont la cible se trouve dans le contexte courant de l'utilisateur. Par exemple, un profil dont la cible est un rôle doit être disponible pour tous les utilisateurs qui jouent ce rôle précis.

La sélection des profils est réalisée suite à une comparaison entre les *contextes d'application* associés à chaque profil disponible et le *contexte courant* de cet utilisateur. Nous rappelons que ces deux types de contexte sont des instances de la classe `Description_de_Contexte` (cf. section 5.1.2 et section 7.2.3). Pour chaque profil, cette étape du processus de filtrage teste si au moins un de ces contextes d'application a un contenu qui est un sous-ensemble du contenu du contexte courant. En d'autres termes, nous vérifions si toutes les circonstances décrites dans un contexte d'application du profil sont présentes dans le contexte actuel de l'utilisateur. Si cette vérification s'avère positive, alors le profil est sélectionné, puisque l'utilisateur se trouve effectivement dans une situation décrite par le profil.

L'approche que nous adoptons est donc comparable à la recherche d'une *relation d'inclusion* entre deux instances de la classe `Description_de_Contexte`. Ainsi, la première étape du processus de filtrage cherche la correspondance entre les contextes d'application et le contexte courant de l'utilisateur à travers l'opérateur *contains* que nous avons proposé précédemment (voir section 6.2). Étant `C` est le contexte courant d'un utilisateur et `C'` un contexte d'application d'un profil donné, si `C` contient `C'` (si $\mathcal{G}(C)$ *contains* $\mathcal{G}(C')$), alors le profil est dit valide, et il sera donc sélectionné par cette étape du processus de filtrage. Ainsi :

- si C_u est le contexte courant de l'utilisateur `U` ;
- si `P` est un profil disponible pour l'utilisateur `U` et $\{C_{i_P}\}_{i\geq 1}$ est l'ensemble de contextes d'application associés au profil `P` ;
- alors si C_u contient au moins un contexte d'application C_{i_P}, `P` est valide.

$$\exists C_{i_P}\left(\mathcal{G}(C_u)\ contains\ \mathcal{G}(C_{i_P})\right)\rightarrow Valide(P)$$

Nous défendons l'utilisation ici des versions les plus souples des opérateurs *contains* et *equals*, c'est-à-dire, l'opérateur *contains* sous une fonction de condition (section 6.2.3) et

l'opérateur *equals* par poids[86] (section 6.1.2). Nous défendons cette utilisation car la nature même de l'information contextuelle demande une approche plus souple. Conformément aux chapitres 5 et 6, au fur et à mesure que la situation d'un utilisateur évolue, le processus d'acquisition de contexte met à jour les instances du modèle qui représentent ce contexte. Néanmoins, d'une part, ce processus d'acquisition n'est pas infaillible, et d'autre part, les caractéristiques des certains attributs de la classe `Élément_de_Contexte`, telle que la `précision`, rend difficile la comparaison entre les instances. Ceci demande donc une approche plus souple pour la comparaison d'instances du modèle de contexte, d'où l'utilisation des versions les moins contraignantes de ces opérateurs.

Afin d'illustrer cette étape, nous prenons l'exemple d'un utilisateur appelé `Alain`, qui utilise un collecticiel de partage de documents, à l'instar de *LibreSource* [Forest 2005a] ou encore *BSCW* [BSCW 2005], pour collaborer avec deux groupes distincts : un `comité` de programme dans lequel `Alain` joue le rôle de `président` du comité, et une `commission` d'évaluation, dans laquelle Alain est `rapporteur`. À un moment donné, nous supposons que l'utilisateur `Alain` accède à l'application de `répertoire_partagé` à partir d'une salle de réunion (`salle_D102`), avec son ordinateur portable (`latitude_X1`), et s'engage dans l'`édition` d'un document précis (`rapport0506`). À ce moment de l'interaction d'Alain avec le collecticiel, nous pouvons supposer que le contexte d'Alain est celui représenté par l'objet `Description_de_Contexte` montré dans la Figure 55a, dans lequel nous apercevons les objets représentant le dispositif et la localisation d'Alain, ainsi que les groupes auxquels il participe et ses rôles dans ces groupes.

Puis, nous pouvons supposer également qu'à ce même moment quatre profils sont disponibles pour l'utilisateur Alain (voir Figure 56) : un premier profil pour les situations dans lesquelles il est engagé dans l'activité d'édition (`profil1`) ; un deuxième profil conçu par l'administrateur du système pour le rôle de rapporteur, qu'Alain joue dans la commission d'évaluation (le `profil2` dans la Figure 56) ; un troisième profil applicable lorsqu'Alain se trouve à la salle de réunion avec son ordinateur portable (`profil3`), et un dernier dirigé vers l'utilisation de son téléphone cellulaire (`profil4`). Parmi ces profil, on note que trois d'entre eux (`profil1`, `profil3` et `profil4`) ont été prédéfinis par l'utilisateur lui-même, comme le montre la présence des tuples de l'association `est_établi_par` liés dans la Figure 56. Les objets `Description_de_Contexte` qui représentent les contextes d'application associés à chacun de ces profils (`description_profil1`, `description_profil2`, `description_profil3`, `description_profil4`) sont présentés dans la Figure 56. En les comparant avec le contexte courant de l'utilisateur Alain, représenté dans la Figure 55, nous pouvons supposer que l'objet `Description_de_Contexte` représentant ce dernier (`contexte_Alain`) contient les *contextes d'application* liés aux trois premiers profils (`profil1`, `profil2` et `profil3`). Ces profils sont donc sélectionnés par la première étape du processus de filtrage. En revanche, le dernier profil ne sera pas sélectionné, puisque son contexte d'application ne correspond pas au contexte courant de l'utilisateur. Ce dernier ne contient pas le précédent, notamment car l'objet `treo650`, présent dans le contexte d'application, ne trouve pas un objet équivalent dans l'objet `contexte_Alain`.

Pour cet exemple, nous supposons l'utilisation de l'opérateur *contains* sous une fonction de condition (celle proposée dans la section 6.2.3, laquelle tient compte de la valeur de l'attribut `précision`), avec une « *ignore list* » qui inclut l'objet `default` de la classe `Membre`. Nous pouvons donc observer que l'utilisation d'une version moins souple de cet opérateur entraînera la non sélection du profil conçu pour le rôle de rapporteur (dont le contexte d'application est l'objet `description_profil2`). Cet exemple montre bien l'impact du choix des

[86]Nous rappelons que celui-ci est utilisé au sein de l'opérateur précédent.

opérateurs : plus l'opérateur utilisé par le système est contraignant, plus difficile sera le choix d'un profil qui correspond au contexte courant d'utilisation.

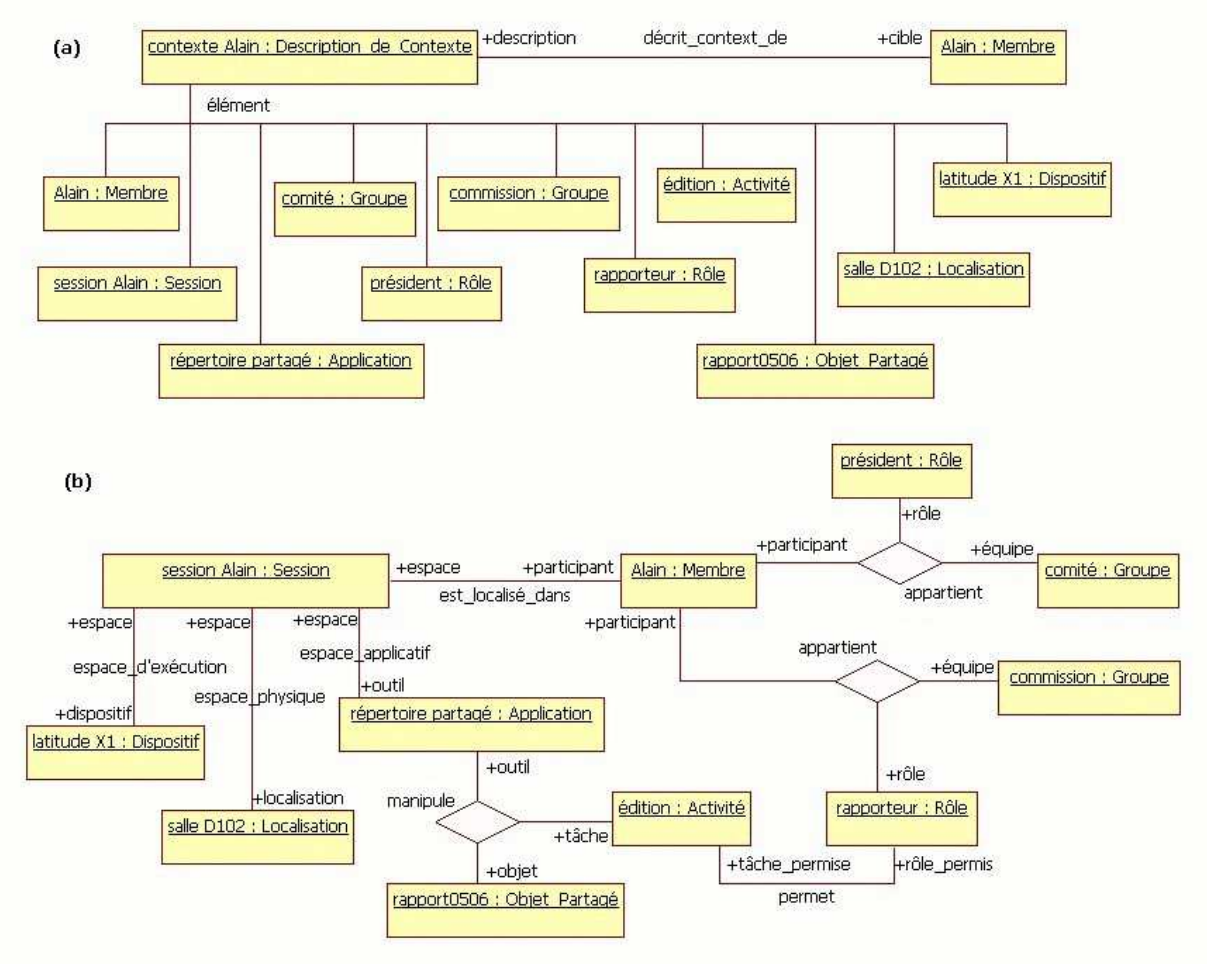


Figure 55. Un exemple du contexte d'un utilisateur, avec tous les objets qui le composent en (a), et en (b), les relations qui lient ces objets entre eux.

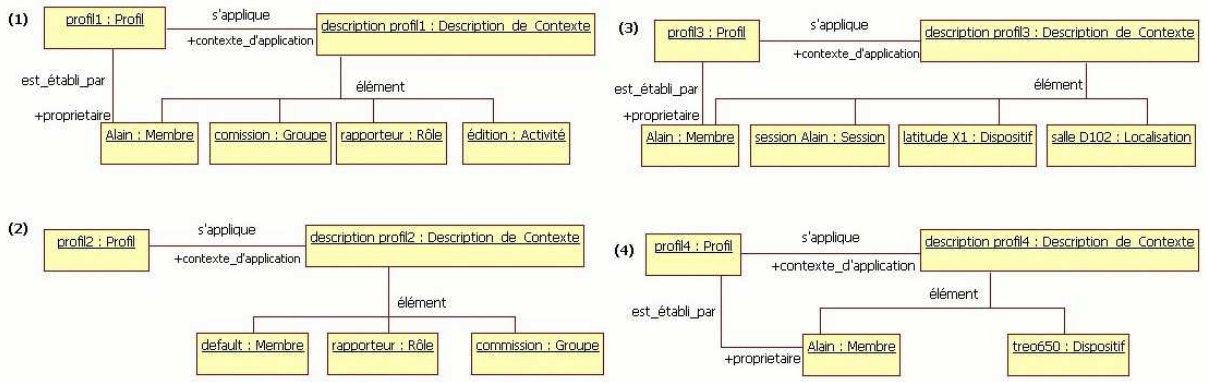


Figure 56. Les contextes d'application associés aux profils disponibles pour un utilisateur donné[87].

À travers cet exemple, nous constatons également l'importance de la définition du (ou des) contexte(s) d'application lié(s) à un profil. Celui-ci guide cette étape du processus de

[87]Pour une meilleure clarté, les attributs des objets et les relations entre eux sont omises de la représentation. Par ailleurs, les objets égaux sont représentés avec les mêmes noms.

filtrage et conditionne le résultat attendu par ce processus. De manière générale, nous avons deux cas de figure : dans un premier cas, des profils plus « généraux », qui sont conçus pour des situations assez générales, et dans un deuxième cas, des profils plus « spécifiques » sont conçus pour des situations très précises. Dans le premier cas, illustré, par exemple, par l'objet `description_profil4` dans la Figure 56, la description d'une situation assez générale (celle de l'utilisation d'un dispositif donné) se traduit par un contexte d'application composé par un nombre réduit d'éléments de contexte. Dans le deuxième cas, une description plus complète d'une situation précise (telle que l'objet `description_profil3` dans la Figure 55) se traduit par un contexte d'application composé par un nombre plus important d'éléments de contexte. Par conséquent, plus le contexte d'application d'un profil est complet, plus il y aura d'éléments qui doivent trouver un équivalent dans le contexte courant de l'utilisateur, et donc plus réduites seront les possibilités de sélection de ce profil par rapport à un profil plus général. En revanche, l'utilisation d'un contexte d'application bien précis permet une meilleure définition des préférences pour ce contexte, puisque les règles (notamment les stratifications) associées à un tel profil sont conçues en tenant compte de ce contexte en particulier. Un certain équilibre dans la définition des profils et des leurs contextes d'application est donc nécessaire car de cet équilibre dépend le succès de cette étape du processus de filtrage.

Néanmoins, il est important d'observer que la définition des profils plus généraux permet la définition des *profils par défaut*, des profils qui seront choisis même dans des situations où il n'y a pas de profils spécifiques prédéterminés. Un tel profil par défaut peut être défini à l'aide d'un contexte d'application minimal composé, par exemple, d'un seul objet `Élément_de_Contexte` : un objet `Membre` pour définir le profil *default* d'un utilisateur, un objet `Rôle` pour le profil d'un rôle donné, ou un objet `Dispositif` pour un profil applicable lors qu'un certain dispositif est employé. Nous pouvons même envisager, en théorie, un contexte d'application "vide", c'est-à-dire un contexte d'application représenté par un objet `Description_de_Contexte` qui ne fait référence aucun `Élément_de_Contexte`. Un tel contexte d'application est toujours valide, car, par définition, tout autre objet `Description_de_Contexte` contient une description vide. Ceci permet donc la définition d'un profil par défaut applicable dans toutes les situations. Un tel profil présente l'avantage de garantir un minimum d'information de conscience de groupe sera disponible. En revanche, un profil aussi général présente également un sérieux désavantage : il ne sera pas réellement adapté au contexte courant de l'utilisateur.

Par ailleurs, cette étape du processus de filtrage tire profit d'une importante caractéristique du modèle de profil : celle de permettre la définition de plusieurs contextes d'application pour un même profil. Ces contextes sont considérés, par définition, comme complémentaires. Par conséquent, la première étape du processus de filtrage peut sélectionner un profil dès qu'au moins un contexte d'application correspond au contexte courant de l'utilisateur. Ceci signifie que les contextes d'application d'un profil sont sémantiquement liés entre eux par un « *ou* » logique : il suffit qu'un de ces objets soit contenu par le contexte de l'utilisateur pour que le profil soit sélectionné.

Enfin, l'Annexe II présente l'algorithme que réalise cette étape du processus de filtrage. Le résultat de cet algorithme est une liste de profils sélectionnés à l'aide de l'opérateur *contains.*

7.3.2 Étape 2 – filtrage des événements

Une fois la sélection des profils achevée, la seconde étape du processus de filtrage est amorcée. Cette étape compare tous les critères définis dans ces profils (les classes

d'événement, les stratifications et les conditions contextuelles) aux informations portées par chaque événement disponible. Ainsi, parmi tous les événements disponibles, le processus ne va retenir que ceux qui correspondent à ces critères. En d'autres mots, cette étape du processus va appliquer les "règles" définies par chaque profil sélectionné.

Cette étape applique donc les stratifications qui ont été définies dans les profils sélectionnés. Ces stratifications filtrent et organisent les événements disponibles. Cette procédure est réalisée en cinq étapes par un algorithme (présenté dans l'Annexe II) :

(a) Tout d'abord, l'algorithme procède à un *ordonnancement* de tous les `profils` sélectionnés par ordre de priorité ;

(b) Ensuite, pour chacun de ces profils, l'algorithme sélectionne, parmi les événements disponibles, ceux qui appartiennent à une classe à laquelle le profil est *abonné* et qui n'ont pas été sélectionné par un profil précédent ;

(c) Après ceci, l'algorithme applique, pour chaque profil, les stratifications qu'il a définies, en commençant par les stratifications extensionnelles, puis les stratifications intensionnelles ;

(d) Pour chacun des événements choisis précédemment, l'algorithme procède à la vérification de l'ensemble des *conditions contextuelles* associées au profil. Ceci signifie que, pour chaque objet `Événement`, l'algorithme vérifie si le `contexte_de_production` associé à celui-ci satisfait les `conditions_de_production` exprimées par l'objet `Profil` (par exemple, s'il a été produit par un certain membre du groupe ou à une localisation donnée). Il vérifie aussi si les conditions sur les objets `Élément_de_Contexte` concernés par l'`Événement` sont également satisfaites (par exemple, si l'`Événement` concerne une `Localisation` donnée ou un certain `Objet_Partagé`) ;

(e) Enfin, l'algorithme rend disponible une liste d'événements organisés en plusieurs niveaux de détails, selon les stratifications extensionnelles appliquées, ainsi qu'une liste ordonnées des stratifications intensionnelles définies et applicables sur l'ensemble d'événements choisis et une liste également ordonnée des stratifications extensionnelles appliquées.

Notons que, selon le *modèle de profil* (cf. section 7.2.3), l'*Entité Masquable* de ces stratifications est la classe `Événement` et ses sous-classes auxquelles le `Profil` est abonné (voir, dans la Figure 52, la contrainte fixée entre les associations `est_abonné_à` et `définition`). Afin de respecter ces stratifications, nous devons appliquer un profil à la fois. Pour cela, l'algorithme procède à l'*ordonnancement* des profils en respectant leur ordre de priorité. Cette ordre de priorité est donné par l'application de la *relation de similarité* Sim (voir section 6.3.3) entre le *contexte courant de l'utilisateur* et le *contexte d'application* de chaque profil. Cette mesure estime quelle proportion du graphe décrit par le contexte de l'utilisateur est constituée par des éléments (objets et tuples) qui ont des équivalents (c'est-à-dire, des objets ou des tuples égaux) dans le graphe décrit par le contexte d'application. Cette mesure favorise donc les profils les plus spécifiques (*i.e.* les profils dont le contexte d'application est composé par un plus grand nombre d'éléments de contexte, décrivant en détail une situation en particulier), au détriment des profils les plus généraux, dont le contexte d'application est composé par un nombre plus réduit d'éléments. Ainsi :

- Soit C_u le contexte courant de l'utilisateur, et $\{C_{i_p}\}_{i \geq 1}$ l'ensemble des contextes d'application C_i associés à un profil `P`, la priorité du profil `P` est donné par :

$$\mathcal{P}riorité(P) = \mathtt{Max}\left(Sim(C_u, C_{i_p}) \right)_{i \geq 1}$$

En donnant la priorité aux profils dont le contexte d'application est le plus proche de celui dans lequel l'utilisateur se trouve, nous voulons favoriser les préférences exprimées par le propriétaire du profil pour ce contexte en particulier. L'objectif est de mettre au premier plan les informations indiquées par les profils qui correspondent au mieux à la situation courante. Par exemple, par rapport au contexte d'utilisation représenté dans la Figure 55, les profils montrés dans la Figure 56 sont ordonnés comme suit : `profil3` (*Sim* `(contexte Alain, description profil3) = 0, 38`), `profil1` (*Sim* `(contexte Alain, description profil1) = 0, 35`) et `profil2` (*Sim* `(contexte Alain, description profil2) = 0, 16`)[88]. Nous pouvons observer, dans cet exemple, que les profils dont le contexte d'application est plus précis sont prioritaires par rapport au profil le plus général.

Une fois les profils ordonnés, la seconde étape du processus de filtrage va les appliquer dans l'ordre établi. Ceci signifie que, pour chaque profil, cette étape du processus va analyser et appliquer les "règles" intrinsèquement représentées dans le profil. Ces règles sont exprimées à travers l'*abonnement* du profil aux classes d'`Événements`, les *stratifications* définies sur ces classes, et les *conditions contextuelles* éventuellement définies. Dans un premier temps, l'application du profil va se concentrer sur les *abonnements*. L'algorithme qui réalise cette étape (voir Annexe II) filtre d'abord les événements disponibles afin de séparer les instances des classes d'`Événement` dont le profil est abonné à d'autres instances. Ce premier tri vise notamment à réduire l'ensemble d'événements sur lequel les stratifications et les conditions contextuelles sont appliquées : à partir de ce moment, l'algorithme ne manipule que les événements dont le profil est abonné, et non plus la totalité des événements disponibles.

Les *stratifications* associées au profil sont définies en considérant comme entité masquable les classes d'`Événement` auxquelles le profil est abonné (cf. section 7.2.3). Elles s'appliquent donc sur l'ensemble d'événements qui a été trié précédemment. Par ailleurs, ces stratifications sont ordonnées, par le propriétaire du profil, lors de sa définition (voir Figure 52). Par conséquent, nous pouvons les appliquer par type et dans l'ordre établi par le propriétaire. L'algorithme (voir Annexe II) applique d'abord les stratifications en extension sur l'ensemble d'instances triées précédemment. Ceci donne lieu à une liste d'événements en plusieurs niveaux, correspondant aux niveaux de détails déterminés par les stratifications extensionnelles. Cette liste, illustrée par la Figure 57, organise donc les événements triés précédemment, et elle peut éventuellement ne contenir qu'un sous-ensemble de ces événements, puisque les stratifications en extension peuvent réduire encore plus cet ensemble en cachant certaines instances.

Par exemple, supposons que, parmi les classes d'événements définis dans le collecticiel utilisé par l'utilisateur `Alain`, nous retrouvons les classes suivantes : une classe « `Nouveau_Document` » qui représente l'ajout d'un nouveau objet partagé dans le système ; une classe « `Modification_Document` » qui représente le fait qu'un document a été modifié ; une classe « `Commentaire` » qui correspond à l'existence d'un commentaire déposé par un membre du groupe ; et une classe « `Présence` », laquelle indique la présence d'un membre du groupe dans le système. À partir de ces classes, nous pouvons imaginer que le `profil3` défini par l'utilisateur `Alain` (voir figures 55 et 56) est abonné aux classes `Modification_Document` et `Commentaire`. Pour ces classes, le profil pourrait porter deux stratifications en extension : S_3^{ext} = {{*événement.intervalle pendant* SEMAINE}, {*événement.intervalle pendant* SEMAINE-1}} définie sur la première classe d'événement abonnée, et S_4^{ext} = {{*événement.intervalle pendant* HEURE}, {*événement.intervalle pendant* JOURNEE}, {*événement.intervalle pendant* SEMAINE}} sur la deuxième classe abonnée. Ainsi, en supposant que les instances e_1 à e_5, représentées dans la Figure 57, appartiennent à l'extension de la classe `Modification_Document`, et les instances e_9 à e_{19} appartiennent à l'extension de la classe

[88]Nous rappelons que le calcul de la mesure de similarité *Sim* tient compte des tuples unissant les objets, même si les tuples sont omis de la représentation faite dans la Figure 56.

`Commentaire`, la Figure 57 représenterait la liste résultant de l'application de ces stratifications S_3^{ext} et S_4^{ext}. Par ailleurs, dans ce même exemple, nous pouvons supposer que l'ensemble d'événements disponibles est supérieur à ces instances. La Figure 57 suggère, par exemple, l'existence d'un sous-ensemble d'instances e_{20} à e_{30} qui ne sont pas présélectionnées puisqu'elles n'appartiennent pas à l'extension d'une des classes dont le profil est abonné. De plus, l'application même des stratifications en extension peut réduire l'ensemble d'événements présélectionnés, comme le suggère la Figure 57, où les instances e_6 à e_8 appartiennent à la présélection, mais elles ne sont pas représentées dans la liste finale d'événements. Nous pouvons donc supposer que ces événements ont été produits dans un `intervalle` antérieur à la semaine précédente (représenté par l'indication « `semaine - 1` »).

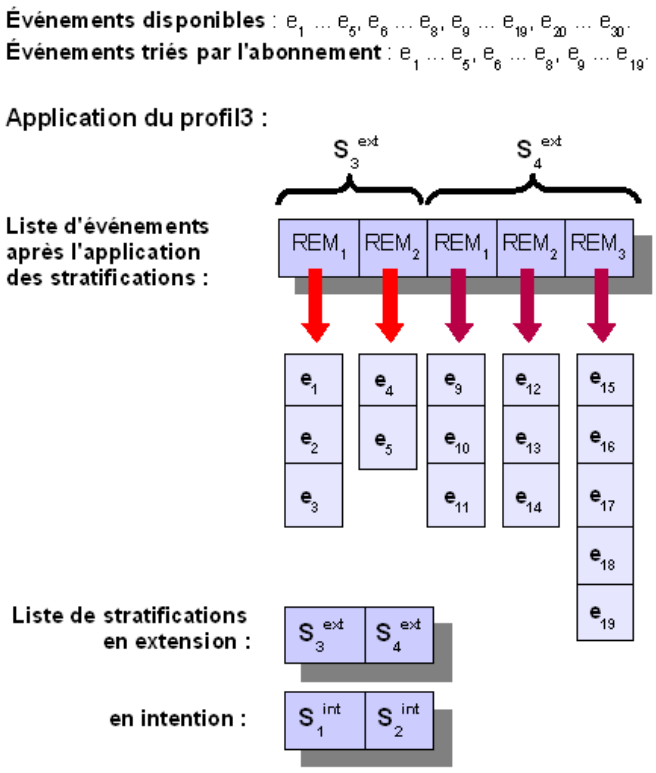


Figure 57. Exemple d'application d'un profil sur un ensemble d'événements.

Les stratifications sont ensuite organisées en deux listes : une pour les stratifications intensionnelles et une autre pour les stratifications extensionnelles. Ces listes sont nécessaires pour l'application des opérations de navigation sur ces stratifications (cf. section 7.2.1.2). Ainsi, au moment où les événements sont délivrés à l'utilisateur, celui-ci peut naviguer dans les niveaux définis par les stratifications à l'aide des opérations *masquer/dévoiler* et *avancer/retourner*, ainsi qu'entre les stratifications, par le biais des opérations *suivant/précédent*.

Enfin, il convient d'observer que, en l'absence des stratifications d'un type, le contenu de tous les événements présélectionnés par l'abonnement est donné à un seul niveau de détail. Ainsi, si un profil ne comporte pas de stratification en extension, seulement des stratifications en intension, toutes les instances des classes d'`Événements` abonnées sont organisées dans un seul niveau. De même, lorsque seulement des stratification en extension sont disponibles. Dans ce cas, tous les attributs des événements sont rendus disponibles dans un seul niveau de détail. Ceci équivaut, en quelque sorte, à créer, de manière automatique, une "stratification" avec une seule REM.

Outre l'abonnement et les stratifications, chaque profil peut également contenir une ou plusieurs *conditions contextuelles* (cf. section 7.2.3), qui s'appliquent tant au contexte de production de l'événement (*condition de production*) qu'aux éléments concernés par l'événement (*condition sur les éléments*). La seconde étape du processus de filtrage analyse donc ces conditions et les comparent à tous les événements sélectionnés par l'abonnement et par les stratifications. Cependant, la manière dont ces conditions sont analysées varie en fonction de son type. Les conditions de production sont comparées au contexte de production de chaque événement à l'aide de l'opérateur *contains*, tandis que les conditions sur les éléments sont confrontées aux éléments concernés par chaque événement (voir section 7.2.2) à travers la mesure de similarité *SimO*.

Le modèle de contenu que nous proposons (voir section 7.2.2) prévoit que chaque objet `Événement` est associé à un objet `Description_de_Contexte`, qui représente le contexte dans lequel l'événement est produit. Les conditions de production définies par un profil (voir section 7.2.3) s'appliquent directement sur ce contexte de production. L'objectif de ces conditions est de restreindre les événements sélectionnés à ceux qui ont été produits dans un contexte comparables à celui fixé par les conditions. Ceci correspond à déterminer l'existence d'une relation d'inclusion entre le contexte de production de l'événement et les conditions décrites par le profil. Ainsi, lorsqu'un profil présente une telle condition, le processus de filtrage va comparer l'objet `Description_de_Contexte` représentant cette condition à l'objet représentant le `contexte_de_production` associé à chaque objet `Événement`. Cette comparaison se fait à l'aide de l'opérateur *contains* de manière à ce que l'événement ne soit sélectionné que si son contexte de production contient au moins une des conditions de production. En d'autres termes, si le contexte de production de l'événement *contient* la situation décrite par une des conditions de production, alors l'événement sera sélectionné. Sinon, il sera rejeté et ne sera pas mis à disposition de l'utilisateur. Ainsi :

- soit `P` un profil et $\{ CP_{i_P} \}_{0 \le i \le n}$ l'ensemble des conditions de production déterminées par ce profil, et `e` un événement dont le contexte de production est C_e,
- alors si $\exists CP_{i_P} \ (\mathcal{G}(C_e)\ contains\ \mathcal{G}(CP_{i_P}))$, l'événement `e` est présélectionné et continue à suivre le processus de filtrage.

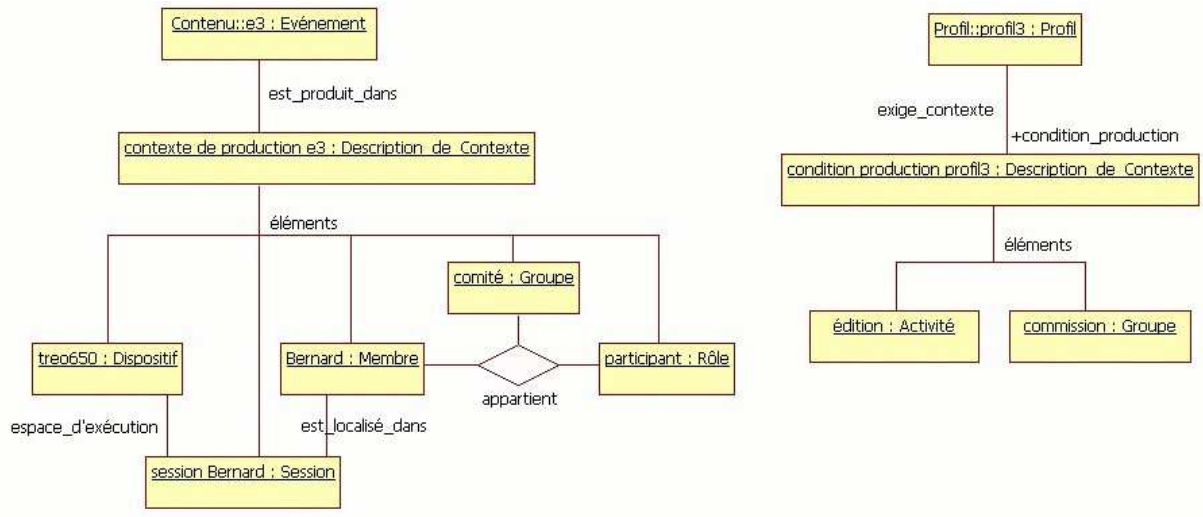


Figure 58. Comparaison entre le contexte dans lequel un événement a été produit et une condition de production sur un profil.

Par exemple, supposons que le profil3 décrit par l'utilisateur Alain dans l'exemple précédent ait comme condition de production le fait que l'événement doit être produit dans le cadre de l'activité d'édition au sein du groupe commission (voir la Description_de_Contexte dans la Figure 58). Dans ce cas, si nous supposons que l'événement e_3, présélectionné par ce profil dans la Figure 57, a été produit par l'utilisateur Bernard, lequel est membre du groupe comité, cet événement sera retiré de la liste d'événements sélectionnés, car son contexte de production ne correspond pas à la condition de production fixée par le profil, comme le montre la Figure 58 (l'objet représentant le contexte_de_production ne contient pas celui représentant la condition_production).

Le second type de condition contextuelle agit sur les *éléments concernés* par l'événement. L'objectif est de sélectionner seulement les événements qui concernent les éléments de contexte fixés par le profil (les objets élément_condition). Néanmoins, puisque l'acquisition de ces éléments de contexte n'est pas un processus dont on peut garantir le résultat, la comparaison entre les conditions sur les éléments et les éléments proprement dits doit être la plus souple possible. Ainsi, nous proposons l'utilisation de la mesure de similarité *SimO* pour comparer ces objets dans cette seconde étape du processus de filtrage. À l'aide du seuil défini dans la classe Profil (voir Figure 53), le processus de filtrage compare les éléments conditions aux éléments concernés par chaque événement. Pour chaque élément de condition, le processus le compare, à travers la mesure *SimO*, aux éléments concernés par l'événement. Si, pour *tous* les *éléments conditions*, le résultat de la mesure *SimO* pour au moins un des éléments concernés à l'événement est supérieur au seuil défini par le profil, cet événement pourrait être sélectionné. Dans le cas contraire (c'est-à-dire, aucun des éléments concernés par l'événement est suffisamment similaire à un élément condition donné), l'événement est écarté de la liste d'événements sélectionnés. Ainsi :

- soit P un profil et $\{ E_{i_P} \}_{i \geq 0}$ l'ensemble d'éléments conditions définis par P et seuil le seuil de similarité déterminé sur P,
- soit e un événement et $\{ E_{j_e} \}_{j \geq 0}$ l'ensemble d'éléments de contexte concernés par l'événement e,
- alors, si $\forall E_{i_P} \exists E_{j_e} \left(SimO(E_{i_P}, E_{j_e}) \geq seuil \right)$ l'événement e reste admis dans la *liste d'événements sélectionnés*.

Afin d'illustrer cette dernière sélection, nous retournons à l'exemple précédent. Dans cet exemple, l'utilisateur Alain dispose de plusieurs profils (voir Figure 56), dont trois sélectionnés par la première étape du processus de filtrage. Parmi ces profils, le mieux classé est le profil3, qui a été défini pour une situation (le contexte d'application du profil) dans laquelle l'utilisateur Alain se trouve dans la salle de réunion (salle_D102) avec son ordinateur portable (latitude_X1). Ce profil est abonné aux classes d'événements Modification_Document et Commentaire à travers plusieurs stratifications en extension (S_3^{ext} et S_4^{ext}, dont l'application est représentée par la Figure 57) et en intension (S_1^{int} et S_2^{int}). Cependant, à travers les conditions contextuelles, l'utilisateur Alain a fixé, dans ce profil, qu'il n'est intéressé que par les événements produits dans le contexte d'une activité d'édition du groupe commission (condition de production illustrée par la Figure 58) et qui concernent l'objet partagé rapport0506, dont son équipe s'occupe depuis quelques semaines. Dans cet exemple, l'objet rapport0506 se présente comme un élément_condition sur les éléments concernés par chaque événement. Ainsi, l'ensemble d'événements présélectionnés par les abonnements et les stratifications (ensemble représenté dans la Figure 57) est réduit par rapport aux conditions de production (représentées dans la Figure 58) et par l'élément de condition. La Figure 59 illustre les éléments de contexte concernés par certains de ces événements. Parmi eux, nous illustrons

un événement e_{12} qui appartient à la classe `Commentaire` et qui concerne une localisation géographique en particulier (`zone_test_1`). Cet événement sera, par la suite, écarté de la liste d'événements sélectionnés, car aucun élément qu'il concerne n'est similaire à l'élément condition (`rapport0506`). En revanche, un événement e_{16} de la même classe, et également illustré par la Figure 59, sera maintenu dans la liste d'événements sélectionnés, car un des éléments qu'il concerne (`rapport`) est suffisamment similaire à l'élément condition (`profil3.seuil = 0,8` et *SimO* `(rapport0506, rapport) ≥ 0,8`).

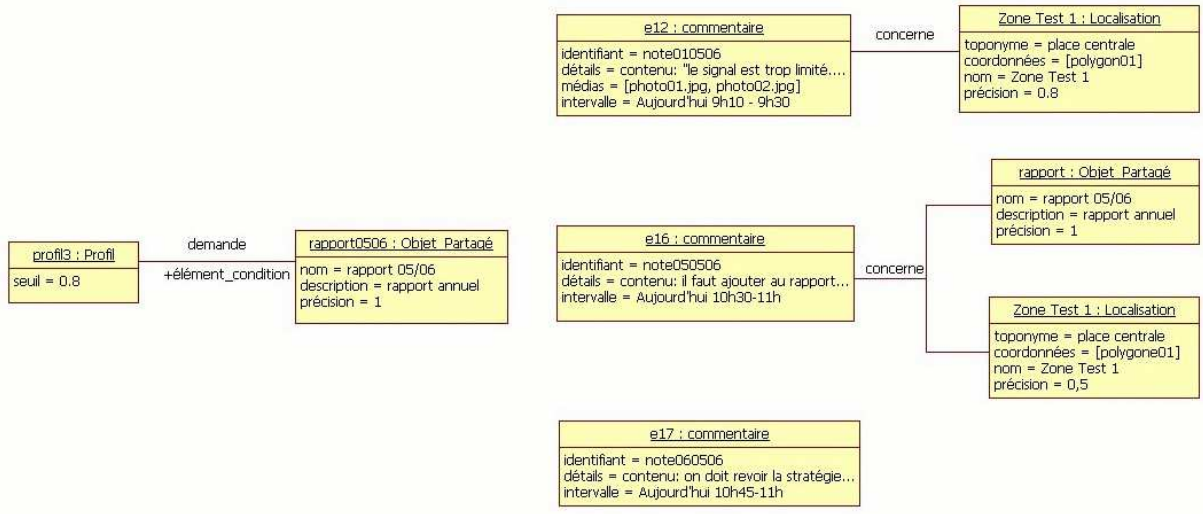


Figure 59. Comparaison entre un élément condition associé à un profil et les éléments de contexte concernés par certains événements.

Par ailleurs, lors que le processus de filtrage applique les conditions contextuelles éventuellement déterminées par un profil, il est nécessaire de considérer le cas des événements qui ne disposent pas d'informations contextuelles associées, c'est-à-dire, des événements qui n'ont pas d'information sur leur contexte de production, ni ne disposent des éléments de contexte qu'ils concernent. Dans ce cas, nous ne pouvons rien affirmer vis-à-vis de leur correspondance aux conditions contextuelles. Pour répondre à cette question, nous avons choisi la même approche déjà adoptée lors de la définition des opérations sur le modèle de contexte (cf. chapitre 6), laquelle consiste à n'utiliser que les informations réellement disponibles. Ceci signifie que, lorsqu'un objet ne possède pas d'élément qui pourrait être comparés aux conditions contextuelles, celles-ci ne s'appliquent pas à cet objet. Ainsi, les événements présélectionnés qui ne disposent pas de contexte de production ou qui n'ont pas d'élément de contexte associé par la relation `concerne` resteront sur la liste d'événements sélectionnés, même si l'algorithme de filtrage n'est pas en mesure de vérifier leur correspondance aux conditions contextuelles. Dans l'exemple illustré par la Figure 59, l'événement e_{17} est présélectionné et ne contient pas d'objet `Élément_de_Contexte` associé capable d'être comparé à l'`élément_condition` lié au profil. Dans ce cas, l'événement e_{17} reste dans la liste d'événement sélectionné.

Une fois que toutes les "règles" définis par un profil donné sont appliquées, la seconde étape du processus de filtrage passe au profil suivant, selon l'ordre de priorité établie entre les profils. L'application du profil suivant engendre l'ajout de nouveaux niveaux dans la liste d'événements sélectionnés, ainsi que l'ajout d'autres stratifications aux listes de stratifications employées. Néanmoins, il convient d'observer que, lorsqu'un événement se trouve déjà dans la liste d'événements sélectionnés et un profil le présélectionne à nouveau, il ne sera pas rajouté une deuxième fois à la liste (cf. algorithme dans l'Annexe II). Ceci signifie que lorsque deux

profils sélectionnent une même information de conscience de groupe, l'ordre de priorité entre les profils est respecté, et l'information est délivrée selon les indications du profil prioritaire. Par conséquent, il n'y a pas de duplication d'information, un même événement n'étant pas présent simultanément sur deux niveaux différents de la liste d'événements sélectionnés.

Ainsi, dans l'exemple de l'utilisateur `Alain`, une fois que l'application du profil prioritaire (`profil3`) est terminée, le processus de filtrage passe à l'application du deuxième profil dans l'ordre, qui est le `profil1` (dont le contexte d'application est illustré dans la Figure 56). Ce dernier a été défini par `Alain` pour les situations dans lesquelles il est engagé dans l'activité d'`édition` dans le groupe `commission`. Ce profil peut donc être abonné à la classe d'événements `Modification_Document`, suivant une stratification en extension (S_5^{ext} = {{*événement.intervalle pendant* SEMAINE}, {*événement.intervalle pendant* MOIS}}) et une stratification en intension (S_1^{int} = {{*identifiant, description*}, {*intervalle, détails*}, {*médias*}}). Dans ce cas, le deuxième profil détermine également la sélection des événements de la classe `Modification_Document` qui ont été déjà sélectionnés par le profil précédent. Ces événements en commun entre les deux profils ne seront pas réintroduits dans la liste d'événements sélectionnés qui sera mise à disposition de l'utilisateur. Seuls les événements de cette classe qui n'ont pas été choisis par le premier profil, mais qui le sont par le second, seront introduit par l'application du second profil. Ceci est notamment le cas de l'événement e_3, qui a été écarté par les conditions contextuelles du `profil3` (voir Figure 58), mais qui est sélectionné par le `profil1`, puisque celui-ci, contrairement au profil précédent, ne contient pas de conditions contextuelles. De cette manière, après l'application de profils sélectionnés dans l'ordre, l'utilisateur `Alain` disposera des événements représentés dans la Figure 60.

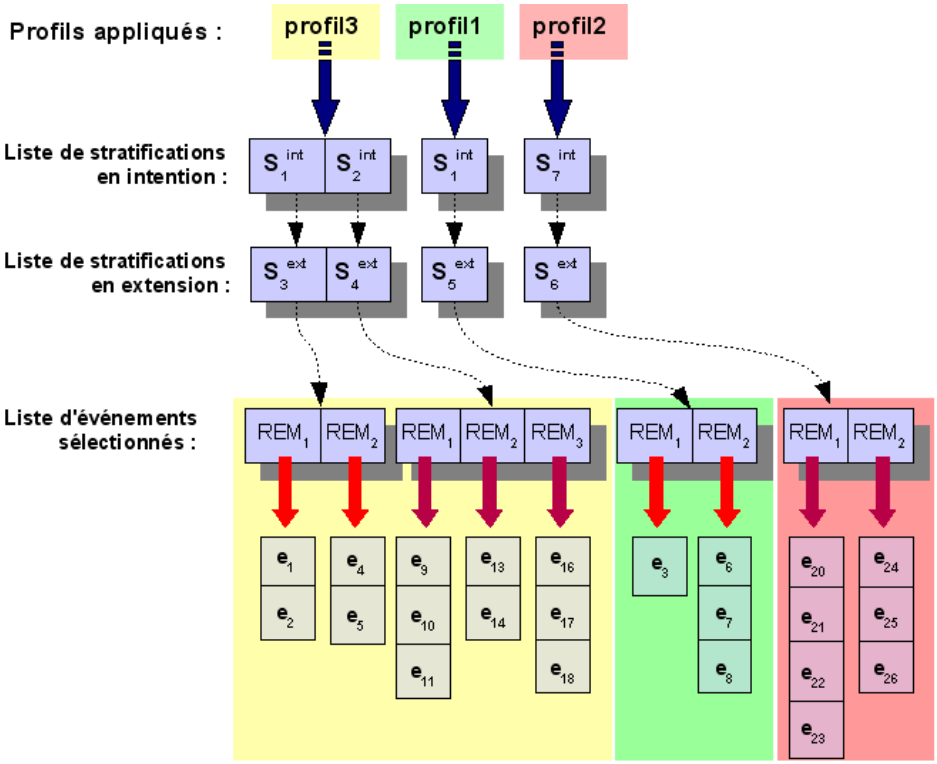


Figure 60. Structures résultant de la seconde étape du processus de filtrage.

À la fin du processus de filtrage, une liste d'événements, organisée en plusieurs niveaux, comme le montre la Figure 60, est rendue disponible pour l'utilisateur. Cette liste respecte les règles définies par les profils, dans l'ordre de priorité établie par la seconde étape du processus. L'utilisateur peut ensuite naviguer sur cette liste, suivant les stratifications

définies par les profils, lesquelles sont également organisées en deux listes. Ceci permet d'appliquer correctement les stratifications à l'ensemble d'événements auquel elles font référence. Chaque *stratification* est applicable sur une classe d'événements donnée (l'*entité masquable* de la stratification), et plus particulièrement, à un ensemble d'instances de cette classe qui ont été sélectionnées par le profil auquel la stratification est associée. Par exemple, la stratification S_2^{int} associée au `profil3` (voir Figure 60) a la classe `Commentaire` comme EM. Elle s'applique donc aux événements e_9 à e_{18}, qui appartiennent à cette classe et qui ont été sélectionnés par le même profil. Elle ne s'applique pas aux événements d'autres classes (par exemple, les événements e_1 à e_5), ni aux événements de la même classe sélectionnés par d'autres profils. Cette mesure garantit que les préférences du propriétaire du profil seront respectées lors de l'application de celui-ci.

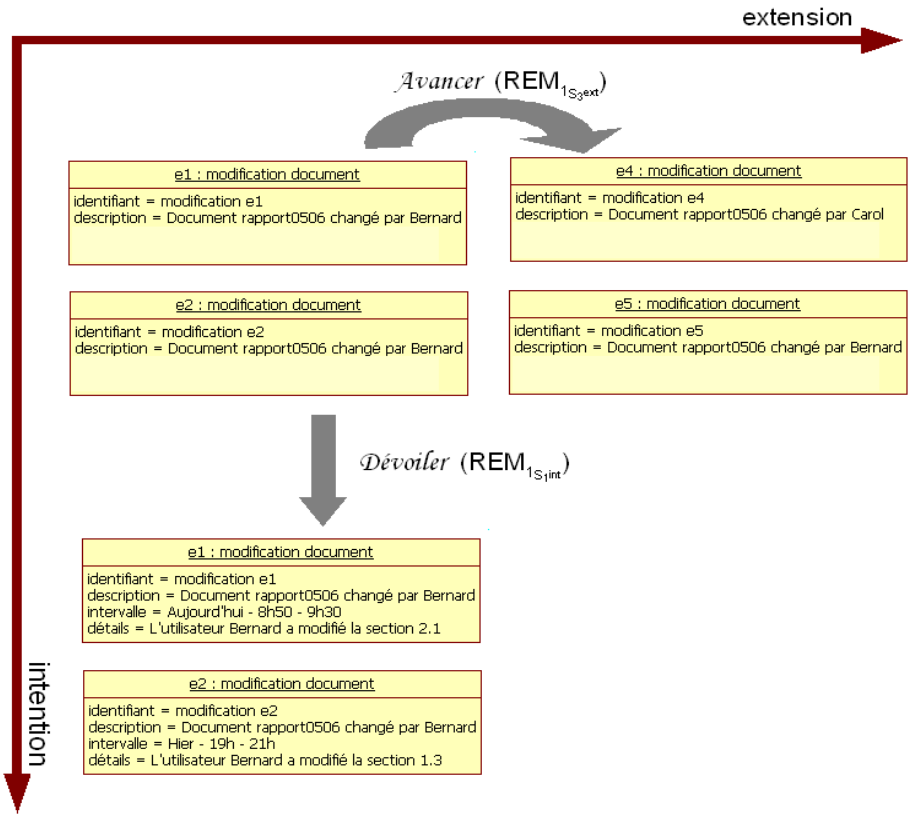


Figure 61. Exemple de l'application des opérations sur les stratifications en extension et en intension.

Afin d'illustrer le résultat du processus de filtrage, prenons l'exemple de l'utilisateur `Alain`, présenté dans la Figure 60. Cet utilisateur disposera, dans un premier niveau, des informations contenues dans les événements e_1 et e_2 de la classe `Modification_Document` (REM_1 de la stratification extensionnelle S_3^{ext}). Ces informations sont organisées selon la stratification intensionnelle S_1^{int}, ce qui signifie qu'Alain aura à sa disposition seulement les attributs `identifiant` et `description` des événements e_1 et e_2. À partir de ce moment, l'utilisateur peut naviguer sur l'ensemble d'événements disponibles, en utilisant pour cela les opérations définies sur le modèle d'accès progressif (cf. section 7.2.1.2). Il peut, par exemple, consulter le prochain niveau en extension, en appliquant l'opération *avancer* sur la stratification S_3^{ext}, ce qui rendra visible les événements e_4 et e_5 ($REM_{2_{S_3^{ext}}}$), ou encore demander plus de détails sur les mêmes objets en appliquant l'opération *dévoiler* sur la stratification intensionnelle S_1^{int}, ce qui rend également visible les attributs `intervalle` et `détails` ($REM_{2_{S_1^{int}}}$). Ces mouvements sont illustrés par la Figure 61.

Par ailleurs, l'utilisateur peut également naviguer entre les stratifications, à l'aide de l'opération *suivant*. Il peut donc passer directement de la stratification S_3^{ext} à la stratification S_4^{ext} en navigant en extension, et du même en intension, en passant directement de la stratification S_1^{int} à la S_2^{int}. Cependant, même si on peut naviguer en extension et en intension de manière apparemment indépendante, l'application des stratifications doit toujours respecter leur définition et les indications portées par les profils. Cela implique que certaines opérations ne seront pas permises, faute d'éléments sur lesquels les stratifications s'appliquent. Par exemple, lorsque l'utilisateur `Alain` navigue dans les stratifications S_1^{int} et S_5^{ext}, associées au second profil (`profil1`), l'application de l'opération *suivant* en intension par cet utilisateur ne peut se faire que si l'utilisateur avance d'abord en extension. Ceci est nécessaire puisque la prochaine stratification dans la liste, la stratification S_7^{int} associée au troisième profil (voir Figure 60), s'applique aux événements de la classe `Présence`, pour laquelle elle a été définie. Il faut donc que des instances de cette classe soient également disponibles pour que cette stratification puisse être appliquée. Ceci est possible à travers la stratification extensionnelle S_6^{ext}, laquelle sélectionne et organise les événements de la classe `Présence` pour le `profil2`.

Nous pouvons donc observer que les listes des stratifications disponibles à l'utilisateur ne sont pas complètement étrangères les unes des autres. Les stratifications définies pour un même profil sont liées : elles s'appliquent aux classes d'événements dont le profil est abonné. Il est donc normal que leurs applications soient, en quelque sorte, inter-dépendantes. En réalité, l'application d'une stratification en intension dépend de la mise à disposition à priori de l'extension de l'EM à laquelle la stratification se réfère. En d'autres termes, l'application des stratifications extensionnelles d'un profil doit précéder l'application des stratifications intensionnelles du même profil. Par ailleurs, nous pouvons observer que les stratifications dépendent du profil auquel elles sont associées. Ceci signifie donc que la navigation à l'intérieur de la liste d'événements produite par le processus de filtrage dépend indirectement des profils appliqués et est directement guidée par les listes des stratifications générées par ce même processus. Les profils déterminent, en quelque sorte, des "méta-niveaux" d'information à l'intérieur desquels l'utilisateur applique l'ensemble de stratifications définies par le profil. La Figure 60 représente, par des arrière-plans de couleurs distinctes, les "méta-niveaux" qui s'appliquent dans l'exemple de l'utilisateur Alain.

En somme, suite aux deux étapes du processus de filtrage, l'information de conscience de groupe est filtrée et organisée en plusieurs niveaux. Le résultat de ce processus est donc l'ensemble filtré et organisé d'informations de conscience de groupe (la liste d'événements) qui doit être mis à disposition de l'utilisateur par le collecticiel, ainsi que les "instructions" pour la navigation sur cet ensemble (les listes de stratifications).

7.4 Conclusions

À l'issue du processus de filtrage[89], un ensemble ordonné d'événements à délivrer à l'utilisateur nomade est donc identifié. Cet ensemble sera très probablement plus approprié au contexte courant de l'utilisateur que l'ensemble d'événements disponibles, puisque cet ensemble a été sélectionné selon les profils de l'utilisateur (et donc ses préférences) définis pour un tel contexte. De plus, cet ensemble sera probablement inférieur à l'ensemble de tous les événements disponibles, ce qui réduit considérablement le risque de surcharge cognitive. Ce risque est un risque majeur dans les collecticiels, surtout en ce qui concerne les utilisateur nomades (cf. section 2.2 et chapitre 4). Réduire le risque d'une surcharge d'informations de

[89]Le processus de filtrage abordé dans ce chapitre a été également traité par les publications [1], [2], [4], [6] et [9] indiquées dans l'Annexe IV.

conscience de groupe et rendre ces informations plus appropriées à l'utilisateur et à son contexte d'utilisation est l'objectif de ce processus de filtrage.

Ce processus diffère des autres travaux de la thématique du Travail Coopératif (cf. chapitre 2) et de ceux dans la thématique de l'Informatique Sensible au Contexte (cf. chapitre 3) à la fois par l'utilisation d'un modèle de contexte, qui a été formalisé et qui s'étend au-delà de la simple localisation et du dispositif de l'utilisateur, et par la prise en compte des préférences de l'utilisateur pour ce contexte. L'utilisation d'un modèle formel pour la notion de contexte permet la libre évolution de ce modèle et de la notion de contexte, sans que le processus de filtrage soit affecté. Même si les éléments de contexte ou leur structure interne (leurs attributs) changent, le processus de filtrage que nous proposons reste valable, puisqu'il repose sur l'essentiel du modèle de contexte (la représentation du contexte par une description de contexte et la notion d'élément de contexte) et sur les opérations définies sur ce modèle. Ceci n'est pas forcément possible sur des architectures comme *Rover* [Banerjee 2002] ou *MoCA* [Rubinsztejn 2004], ni sur des applications comme celle proposé par Muñoz *et al*. [Muñoz 2003], puisque, dans ces travaux, la notion de contexte n'est pas formalisée.

Par ailleurs, l'association entre les préférences de l'utilisateur et les contextes potentiels à travers les profils implique davantage l'utilisateur sur le processus de filtrage. L'utilisateur peut exprimer ses préférences pour un contexte donné, ce qui permet une meilleure adaptation de l'information de conscience de groupe à la fois au contexte d'utilisation et aux attentes de l'utilisateur. Ceci n'est pas le cas des systèmes tels que ceux proposés par Hibino *et al.* [Hibino 2002], par Bardram et Hansem [Bardram 2004], ou encore par Muñoz *et al.* [Muñoz 2003], dans lesquels l'utilisateur ne peut pas associer ses préférences personnelles à une situation donnée. Les modèles que nous proposons dans ce chapitre permettent une telle association, sans pour autant empêcher la spécification des profils par défaut.

Grâce à l'utilisation du modèle de contexte, il est possible aux utilisateurs de concevoir des profils applicables à des situations spécifiques, tant par rapport à leur *contexte collaboratif* (des profils applicables à un groupe, à un rôle, à une activité donnée, etc.), que par rapport à leur *contexte physique* (un profil applicable à un dispositif ou à une localisation donnée, ou encore un profil qui sélectionne seules les informations relatives à une localisation, etc.). Cette possibilité d'avoir des profils liés tant à des aspects physiques qu'à des aspects collaboratifs est un atout nécessaire pour la nouvelle génération des collecticiels sur le Web. Ces collecticiels sont désormais naturellement accessibles aux utilisateurs nomades, et doivent donc se comporter en tant que systèmes sensibles au contexte, tout en restant des systèmes coopératifs, dont l'objectif est de favoriser la coopération.

La définition des profils reste un élément important du processus de filtrage que nous proposons. La qualité des informations fournies par le processus de filtrage dépend directement de la définition des profils. Un profil qui n'a pas été bien défini, soit en relation à son contexte d'application, soit en ce qui concerne les règles de filtrage, peut entraîner des résultats inattendus. Par exemple, le profil peut être choisi dans une situation différente de celle imaginée par le propriétaire du profil, ou encore il peut entraîner la sélection d'informations également différentes de celles attendues. Comme dans tous les systèmes qui utilisent le principe d'un profil pour l'adaptation, la définition du profil reste un point sensible dont dépend le résultat de l'adaptation. Le processus de filtrage que nous proposons n'est pas une exception à cette règle. Par ailleurs, il est important d'observer qu'un certain équilibre entre la définition des profils spécifiques et généraux est nécessaire. D'une part, plus un profil est général, plus importantes seront ses chances d'être choisi et appliqué, mais l'information qu'il sélectionne sera probablement moins adaptée au contexte courant de l'utilisateur. D'autre part, plus un profil est spécifique, plus l'information qu'il sélectionne sera adaptée à la situation dans laquelle se trouve l'utilisateur lors de son application, et plus réduites sont les

chances qu'il soit appliqué en dehors de cette situation. La spécification de ces deux types de profils est nécessaire pour le bon fonctionnement du mécanisme de conscience de groupe.

De plus, la spécification des profils par défaut est l'élément clé pour le traitement des situations inattendues. Même si un collecticiel sur le Web est utilisé majoritairement dans des situations connues, dont on peut caractériser certains éléments de contexte, il est toujours possible que l'utilisateur se trouve face à une situation inconnue. Dans ces cas, la définition des profils par défaut peut aider l'utilisateur, au moins en lui garantissant l'accès à un minimum d'informations de conscience de groupe. Nous préconisons la spécification d'au moins un profil par défaut pour chaque type de dispositif supporté par le système, afin de garantir aux utilisateurs un minimum d'informations, même lorsqu'ils ne disposent pas d'un profil spécifique à la situation dans laquelle ils se trouvent. Ceci est nécessaire étant donnée l'importance de l'information de conscience de groupe pour le succès du travail coopératif (cf. section 2.2). Un utilisateur, surtout un utilisateur nomade, peut facilement perdre le contact avec les collègues et leurs activités. Il est important pour lui d'avoir à sa disposition les informations concernant ses collègues et l'évolution du travail de son équipe. Le collecticiel doit donc garantir qu'au moins un minimum d'informations de conscience de groupe sera disponible, même lorsque l'utilisateur se trouve dans une situation inconnue, d'où l'importance de la définition des profils par défaut.

En ce qui concerne son fonctionnement, le processus de filtrage que nous proposons comporte deux étapes, elles mêmes composées d'autres pas. Certains peuvent observer que ce processus peut se montrer très complexe lorsqu'un grand nombre d'événements et de profils sont disponibles, et encore plus lorsque ces profils contiennent plusieurs conditions contextuelles. Cependant, il ne faut pas oublier que ce processus se positionne dans le cadre de la conception des collecticiels sur le Web. Il est donc conçu pour une architecture Web typique, dans laquelle, un serveur de calcul (le ou les serveurs Web) est disponible et héberge les principales fonctionnalités du système. Dans le cadre de cette architecture, le processus de filtrage se trouve naturellement au sein d'un serveur de calcul, capable donc de supporter la charge imposée par le mécanisme de filtrage.

Pour conclure, il est important d'observer que, même si tout le processus de filtrage guidé par le contexte que nous proposons vise à adapter l'information de conscience de groupe au sein d'un collecticiel sur le Web, nous pouvons envisager son application pour d'autres type d'information, grâce à l'utilisation du modèle de contenu, lequel est centré sur la notion abstraite d'événement. Il suffit de spécialiser la notion d'événement au type d'information voulu. Par exemple, nous pouvons imaginer qu'au lieu de représenter une action dans le cadre d'une coopération, les nouvelles sous-classes d'événement pourront représenter l'occurrence des différents type d'événement dans le cadre de la gestion de risques naturel (l'occurrence d'un éboulement, d'une avalanche, etc.). Puisque le processus de filtrage se base sur le modèle de contenu de manière générale, ce type de spécialisation est envisageable et permettrait l'application du mécanisme de filtrage pour d'autres types d'information, non seulement la conscience de groupe. De même en ce qui concerne l'application du processus à d'autres types de systèmes sensibles au contexte. Même si le processus à été conçu pour les collecticiels, l'application de ses principes peut se faire en dehors du cadre d'un système coopératif.

8 Implémentation du Processus de Filtrage : Le Canevas BW-M

Dans ce chapitre, nous proposons une implémentation pour le processus de filtrage (chapitre 7) à travers un canevas nommé BW-$\mathcal{M}$[90]. Dans ce travail, le modèle de contexte (chapitre 5), avec les opérations définies sur ce modèle (chapitre 6), ainsi que le mécanisme de filtrage proposés se positionnent surtout à un niveau conceptuel. Le canevas BW-$\mathcal{M}$ vise à concrétiser cette proposition, par une mise en œuvre des éléments proposés. Pour cela, le canevas BW-$\mathcal{M}$ est conçu comme un mécanisme de support à la conscience de groupe sensible au contexte.

Nous avons vu dans le chapitre 2 qu'il n'existe pas une architecture typique qui caractériserait les collecticiels. Nous avons, le plus souvent, une architecture répartie, suivant différents modèles, dont le modèle client-serveur propre aux systèmes Web. Même si l'architecture de ces systèmes varie, nous pouvons les concevoir comme étant construits à l'aide de différents composants (ou services), dont un composant responsable de la gestion de l'information de conscience de groupe. C'est le cas, par exemple, du canevas *Ants* [Lopez 2003]. Les systèmes sensibles au contexte n'ont également pas une architecture standard, mais on peut désormais constater la présence de certains types de composants, notamment pour l'acquisition du contexte, à l'instar de l'architecture *MoCA* [Rubinsztejn 2004]. Néanmoins, puisque la majorité des systèmes sensibles au contexte n'adoptent pas un modèle pour la représentation des informations contextuelles, la gestion de contexte se limite à ce composant d'acquisition.

Le canevas BW-$\mathcal{M}$ se positionne, dans cette logique, à la fois en tant que composant de support de la conscience de groupe pour les collecticiels, et également en tant que composant de gestion (et non d'acquisition) de contexte, pour un système sensible au contexte. Le canevas BW-$\mathcal{M}$ se présente donc comme une sorte de composant pour le support *sensible au contexte* à la conscience de groupe. Par ailleurs, ce canevas est conçu pour être utilisé au sein des collecticiels sur le Web, notamment ceux supportant le travail asynchrone et nomade. Les utilisateurs de ces systèmes ont besoin d'un support à la conscience de groupe qui soit adapté à leur condition nomade. Le canevas BW-$\mathcal{M}$ s'attaque donc à cette question.

Ce chapitre s'organise comme suit : la section 8.1 présente la structure du canevas ; la section 8.2 discute sa méthodologie d'application ; et la section 8.3 apporte nos conclusions à propos du canevas BW-$\mathcal{M}$.

8.1 Structure

Le canevas BW-$\mathcal{M}$ se base sur les principes posés par nos travaux précédents, notamment sur le canevas BW [KirschPi 2003a] [KirschPi 2003b]. Dans ces travaux, nous avons proposé un mécanisme de support à la conscience de groupe, le canevas BW, qui utilise une approche basée sur la notion d'événement et qui structure le support à la conscience de groupe en trois couches : interface, contrôle et stockage. Ces mêmes principes sont adoptés par le canevas BW-$\mathcal{M}$. Nous adoptons donc ici une approche par événement, notamment à travers le modèle de contenu (cf. section 7.2.2), laquelle se caractérise également par un cycle de vie en trois phases. Le traitement de ces trois phases implique les différents composants du

[90]Le terme BW-$\mathcal{M}$ est un acronyme pour « *Big Watcher - Mobile* ».

canevas BW-*M*, lesquels s'organisent en respectant l'architecture en trois couches. Par ailleurs, ces composants sont implémentés en plusieurs paquetages, à travers le langage Java. Dans les prochaines sections nous présentons l'approche par événements (section 8.1.1), l'architecture logicielle adoptée par le canevas (section 8.1.2), ainsi que les paquetages en langage Java qui composent cette implémentation (section 8.1.3).

8.1.1 Approche par événement

Conformément au modèle de contenu (cf. section 7.2.2), nous adoptons dans ce travail une *approche par événement*[91]. Selon cette approche, l'information de conscience de groupe est représentée à travers les événements. Chaque événement représente, le plus souvent, une action qui a eu lieu dans le collecticiel et dont la connaissance peut contribuer à la coordination du travail en équipe. Les classes d'événements observables sont définies par les concepteurs du système, à travers la spécialisation du modèle de contenu, selon le type de système et les activités que les utilisateurs peuvent accomplir au sein du système.

Hormis la définition des événements, cette approche de la conscience de groupe repose également sur un *cycle de vie* en trois phases, dans lesquelles l'information de conscience de groupe est produite et mise à disposition de l'utilisateur. Ces trois phases sont les suivantes (voir Figure 62 et Figure 63) : *(i)* d'abord, une première phase d'*enregistrement*, dans laquelle les classes d'événements observables sont enregistrées ; *(ii)* puis, une deuxième phase d'*observation*, dans laquelle le collecticiel observe l'occurrence des événements ; *(iii)* et une troisième phase de *notification*, dans laquelle on notifie les utilisateurs sur l'occurrence de ces événements.

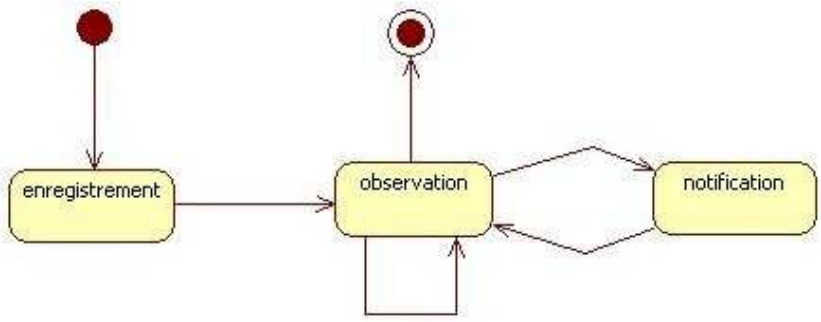


Figure 62. Diagramme d'états relatif au cycle de vie du canevas BW-*M*.

Le fonctionnement du canevas BW-*M* est donc guidé par ce cycle de vie en trois phases (Figure 62). Dans ce diagramme, on observe que le canevas entre dans la deuxième phase dès que l'occurrence d'un événement est constatée, et dans la troisième phase, dès qu'il reçoit la demande de notification, suite à laquelle le canevas BW-*M* procédera au filtrage des événements disponibles.

Dans la phase d'*enregistrement*, le collecticiel informe le canevas BW-*M* des classes d'`Événement` qui sont observées par le collecticiel pendant son exécution (voir Figure 63). En d'autres termes, c'est dans cette phase que l'on détermine l'ensemble d'événements qui concerneront le mécanisme de conscience de groupe. Au moment de la conception d'un collecticiel, les événements qui peuvent être produits par lui sont déterminés par les concepteurs. Ceci se fait, d'une part, à travers la spécialisation de la classe `Événement` dans le

[91]En anglais, cette approche est référencée par le terme « *event-based awareness* ».

modèle de contenu, et, d'autre part, par l'enregistrement de ces sous-classes auprès du canevas. Par exemple, lorsqu'on utilise le canevas BW-$\mathcal{M}$ pour la construction d'un collecticiel de partage de documents, les concepteurs peuvent enregistrer, auprès du canevas, les classes d'événements « `présence` », pour indiquer la présence d'un collègue sur l'espace de travail partagé, et « `Modification_Document` », indiquant qu'une nouvelle version d'un objet partagé est disponible, parmi d'autres classes d'événements. Ceci signifie que, durant l'exécution du système, celui-ci va observer l'occurrence de ces actions (la présence d'un utilisateur sur le système et les modifications apportées aux documents) afin d'alimenter le mécanisme de conscience de groupe avec ces informations.

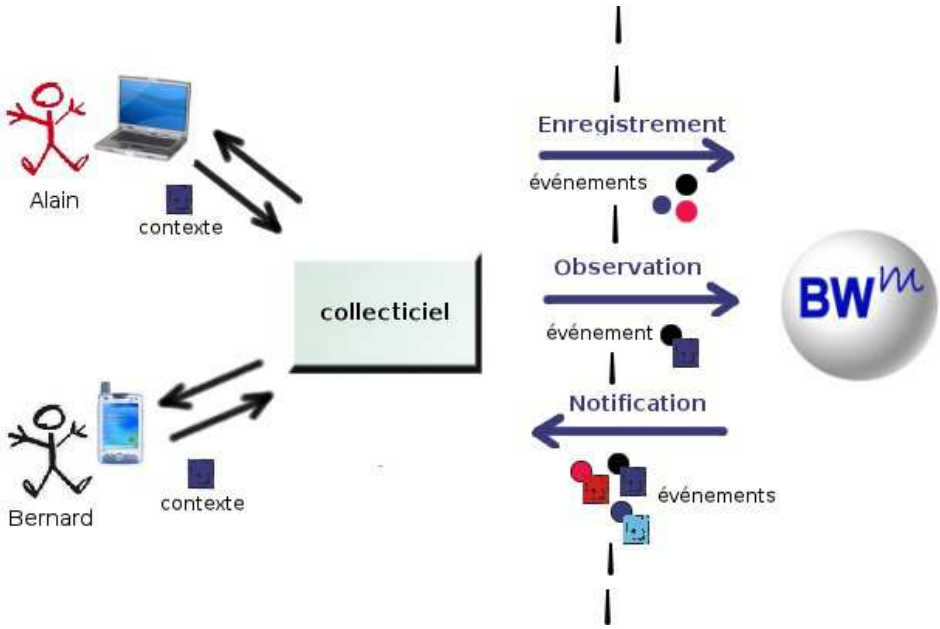


Figure 63. Les phases de fonctionnement du canevas BW-$\mathcal{M}$.

Ainsi, au fur et à mesure que les utilisateurs interagissent avec le collecticiel, celui-ci les observe et produit les événements correspondants à leurs actions. C'est la deuxième phase du cycle, la phase d'*observation*. Dans cette phase, le système informe le canevas de l'occurrence d'un événement, en lui transmettant les objets qui correspondent à l'événement et à son contexte de production (voir Figure 63), conformément au modèle de contenu (cf. section 7.2.2). Par exemple, l'entrée de l'utilisateur `Bernard` dans le système est à l'origine d'un événement de la classe `Présence` décrivant la connexion de `Bernard` au système. Le système renvoie au canevas BW-$\mathcal{M}$ cet objet `Événement`, ainsi que les objets liés à celui-ci, notamment le contexte de production et les objets `Élément_de_Contexte` lui concernant. À partir de ce moment, le canevas BW-$\mathcal{M}$ peut traiter cette occurrence, en vérifiant d'abord si l'événement envoyé appartient à une classe d'événement enregistrée précédemment. Si c'est le cas, le canevas incorpore cet objet `Événement` à sa base de connaissances, et dans le cas contraire, l'événement n'est pas pris en considération par le canevas. Par ailleurs, durant la phase d'observation, le collecticiel informe au canevas BW-$\mathcal{M}$ tout changement survenu dans le contexte courant des utilisateurs connectés au système. Par exemple, si l'utilisateur `Alain` change de localisation pendant sa connexion au système (il passe d'une salle à une autre), le collecticiel informe le canevas de ce changement en lui informant la nouvelle localisation de cet utilisateur.

La troisième phase du cycle de vie, la phase de *notification*, est déclenchée dès lors qu'un utilisateur demande l'information de conscience de groupe, ou dès que le système

décide de la lui envoyer. L'objectif est de rendre compte aux utilisateurs des événements qui se sont produits à l'intérieur du collecticiel et qui sont pertinents pour lui. Ainsi, à partir du moment que le canevas BW-$\mathcal{M}$ reçoit la demande de notification, il doit procéder au filtrage des événements en fonction des profils disponibles pour l'utilisateur cible et de son contexte courant. Pour cela, il implémente le processus de filtrage présenté dans la section 7.3.

Nous pouvons observer que le canevas BW-$\mathcal{M}$ se présente comme un composant à part du collecticiel (voir Figure 63). L'idée est de permettre l'utilisation du canevas dans différents collecticiels. Ceux-ci peuvent être conçus à l'aide du canevas BW-$\mathcal{M}$ sans que, pour autant, le contenu réel du canevas lui soit connu. Par conséquent, les concepteurs peuvent se concentrer sur la définition des classes d'événements et sur d'autres questions propres au collecticiel, et laisser la gestion et le filtrage de l'information de conscience à charge du canevas BW-$\mathcal{M}$.

Cette indépendance entre le canevas BW-$\mathcal{M}$ et le collecticiel est l'avantage le plus significatif de l'approche par événement. L'information de conscience de groupe est naturellement très liée au collecticiel. La modélisation de cette information à travers la notion d'événement permet sa définition par le biais d'un concept plus abstrait, celui de l'événement, ce qui garanti une certaine indépendance entre le mécanisme de conscience de groupe et le collecticiel. Par ailleurs, l'utilisation d'un cycle de vie en trois phases permet l'observation de différentes classes d'événements et leur utilisation au sein du mécanisme de support.

8.1.2 Architecture logicielle

Le canevas BW-$\mathcal{M}$ a été fortement inspiré par nos travaux précédents [KirschPi 2003a] [KirschPi 2003b], dans lesquels nous proposons que le support de la conscience de groupe soit organisé selon une architecture en trois couches : *interface*, *contrôle* et *stockage*. Les éléments de la couche d'*interface* s'occupent de la présentation de l'information de conscience de groupe aux utilisateurs. Ils interagissent directement avec les utilisateurs du collecticiel, en leur présentant l'information que la couche de *contrôle* a été sélectionnée. Celle-ci est responsable de la gestion proprement dite de l'information de conscience de groupe. Les éléments de la couche de contrôle gèrent les requêtes faites par le collecticiel, notamment lors de l'enregistrement et de l'observation des événements, ainsi que celles réalisées par les utilisateurs eux-mêmes, à travers la couche d'interface. Par ailleurs, la couche de contrôle est également responsable du filtrage de l'information de conscience de groupe. Enfin, la couche de *stockage* est responsable du stockage des informations nécessaires au mécanisme de conscience de groupe. Ces informations incluent non seulement les événements qui représentent l'information de conscience de groupe proprement dite, mais également les informations relatives aux utilisateurs, tel que leurs profils.

Cette organisation en trois couches permet notamment le partage des responsabilités entre les différentes couches, et par la même occasion, une réduction de la complexité pour les développeurs, puisque les problèmes propres à chaque couche peuvent être traités de manière indépendante. Ainsi, les fonctionnalités du canevas BW-$\mathcal{M}$ se concentrent notamment sur la couche de *contrôle*, puisque ce canevas répond aux responsabilités propres à cette couche, qui sont la gestion de l'information de conscience de groupe et du contexte.

Le canevas BW-$\mathcal{M}$ adopte donc cette architecture en trois couches (voir Figure 64). Dans cette figure, nous pouvons observer que les différents composants du canevas se trouvent notamment au sein de la couche de contrôle. Le canevas BW-$\mathcal{M}$ est composé par trois unités de gestion centrales : la *gestion de conscience de groupe*, la *gestion de contexte* et la *gestion de connaissances*. La première unité est responsable de la gestion des informations

de conscience de groupe et des requêtes liées à ces informations. Cette unité inclut également une partie de la gestion du Modèle d'Accès Progressif (MAP), laquelle est partagée entre la couche de contrôle et la couche d'interface. La seconde unité du canevas BW-$\mathcal{M}$ est responsable de la gestion des informations contextuelles, notamment les informations relatives au contexte courant de chaque utilisateur connecté au système à un moment donné. Enfin, la troisième unité de gestion, la gestion des connaissances, s'occupe notamment du stockage des connaissances manipulées par les autres unités de gestion, à savoir les informations de conscience de groupe et contextuelles. Cette unité contrôle donc la base des connaissances qui contient les informations utilisées par le canevas BW-$\mathcal{M}$. Dans les prochaines sections nous détaillons chacune de ces unités.

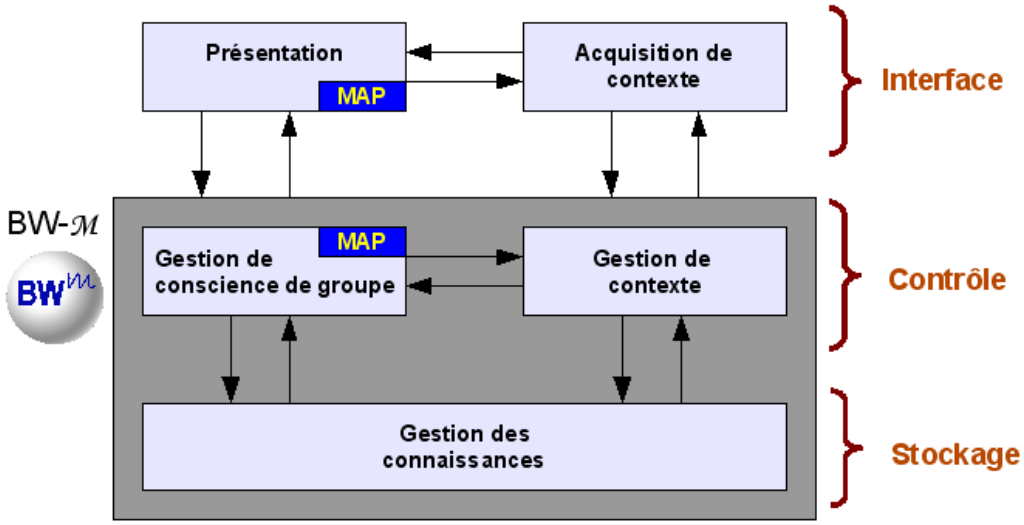


Figure 64. Architecture adoptée par le canevas BW-$\mathcal{M}$.

8.1.2.1 La gestion de l'information de conscience de groupe

L'unité de gestion de conscience de groupe du canevas BW-$\mathcal{M}$ intervient dans les trois phases du cycle de vie. Dans la phase d'enregistrement, cette unité reçoit et traite les demandes venues du collecticiel. Ensuite, dans la phase d'observation, l'unité de gestion de conscience de groupe reçoit du collecticiel les événements qui se sont produits et les traite en fonction des classes d'événements enregistrées auparavant. Puis, dans la phase de notification, cette unité de gestion réalise le filtrage des événements en fonction des profils disponibles pour l'utilisateur cible et de son contexte courant, qu'elle obtient auprès de l'unité de gestion de contexte (section 8.1.2.2).

Pour accomplir ses fonctions, l'unité de gestion de conscience de groupe est elle-même composée par différentes sous-unités (voir Figure 65), qui se partagent les responsabilités. Parmi ces unités, nous pouvons observer, dans la Figure 65, la présence de plusieurs sous-unités de façade. Celles-ci correspondent aux interfaces visibles à l'extérieur de l'unité de gestion et à travers lesquelles cette unité communique avec les autres unités du canevas BW-$\mathcal{M}$ et avec le collecticiel. L'utilisation de façades s'inspire du patron de conception *façade* [Gamma 1994]. Celui-ci vise à simplifier l'accès aux éléments internes à un paquetage (ou à un composant) à travers une interface visible et reconnue à l'extérieur du même. Ceci isole le contenu du paquetage de l'extérieur et réduit ainsi les risques que l'évolution des éléments à l'intérieur du paquetage puisse affecter davantage les éléments à l'extérieur qui l'utilisent.

Ainsi, nous séparons l'accès à l'unité de gestion de l'information de conscience de groupe en quatre types de façades, selon le type d'opération que ces façades comportent (voir Figure 65) : (*i*) une *façade de contrôle*, qui admet l'enregistrement des classes d'événements et l'occurrence d'un événement précis ; (*ii*) une *façade de filtrage*, qui acceptent les demandes de notification venues de l'utilisateur, à travers la couche d'interface, ou du collecticiel ; (*iii*) une *façade de contexte* à travers laquelle l'unité de gestion de conscience de groupe interagit avec l'unité de gestion de contexte ; (*iv*) et une *façade de stockage*, à travers laquelle cette unité sollicite les services de l'unité de gestion de connaissances.

Hormis les sous-unités de façade, la gestion de conscience de groupe compte trois éléments principaux (voir Figure 65) : le *moniteur*, le *filtrage* et le *MAP*. Le moniteur est la sous-unité responsable tant du traitement des demandes d'enregistrement de classes d'événements que de l'observation de ces événements. Pour sa part, le filtrage opère pendant la phase de notification. Cette sous-unité réalise la sélection des événements pertinents pour l'utilisateur, selon le processus de filtrage que nous avons défini dans le chapitre 7. Enfin, le MAP contrôle l'application du modèle d'accès progressif. Cette sous-unité réalise la gestion des stratifications qui sont associées aux profils.

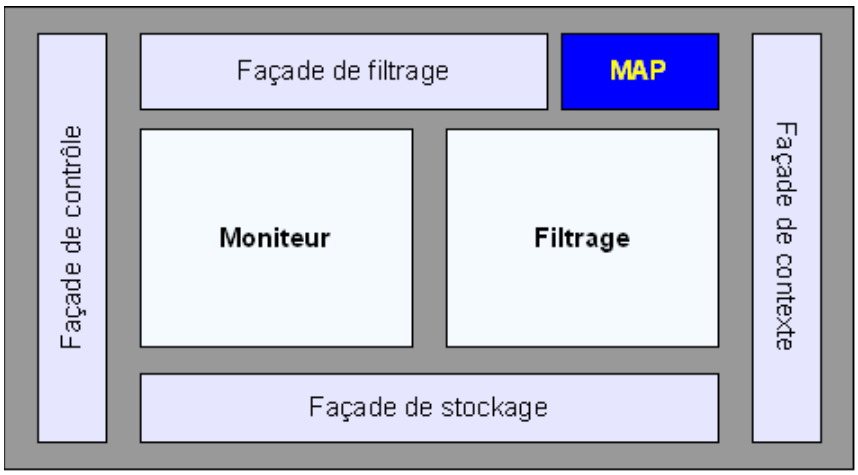


Figure 65. Structure interne de l'unité de gestion de conscience de groupe.

Le moniteur

Le moniteur est l'élément du canevas BW-$\mathcal{M}$ responsable du traitement des requêtes relatives à l'*enregistrement* des classes d'événements qui seront observées. Il reçoit également les informations relatives à l'*occurrence* d'un événement au sein du collecticiel. Pour cela, le moniteur interagit directement avec les façades de contrôle et de stockage. La façade de contrôle, qui est mise en œuvre par deux interfaces (`RegisterFacade` et `MonitorFacade` dans la Figure 66), permet au collecticiel de solliciter au canevas les services correspondants à l'enregistrement et à l'occurrence des événements. D'autre part, la façade de stockage (interface `StorageFacade` dans la Figure 66) permet au moniteur de solliciter les services de stockage de l'unité de gestion de connaissances.

Le traitement des demandes d'*enregistrement* d'une nouvelle classe d'événement se fait à travers la classe de gestion nommée `Register` (voir Figure 66). Celle-ci organise un registre contenant la liste de classes d'`Événement` enregistrées par le collecticiel auprès du canevas BW-$\mathcal{M}$. Elle reçoit les demandes d'enregistrement du collecticiel par le biais de la façade de contrôle. Pour chaque demande, la classe `Register` vérifie d'abord la validité de la demande et introduit ensuite les informations relatives à celle-ci au registre (voir Figure 67). Une

demande d'enregistrement est valide si la classe d'événement concernée par la demande respecte le modèle de contenu présenté dans la section 7.2.2 et si elle n'a pas été enregistrée auparavant. Une fois que la demande est vérifiée et s'avère correcte, son traitement consiste à inclure au registre le nom de la nouvelle classe d'événement enregistrée, afin que le moniteur puisse reconnaître les instances de cette classe lors de l'occurrence des événements. La Figure 67 illustre le traitement d'une demande d'enregistrement.

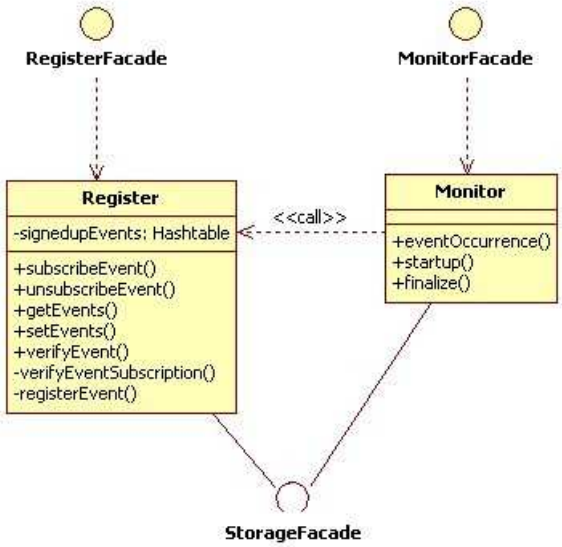


Figure 66. Classes composant le moniteur.

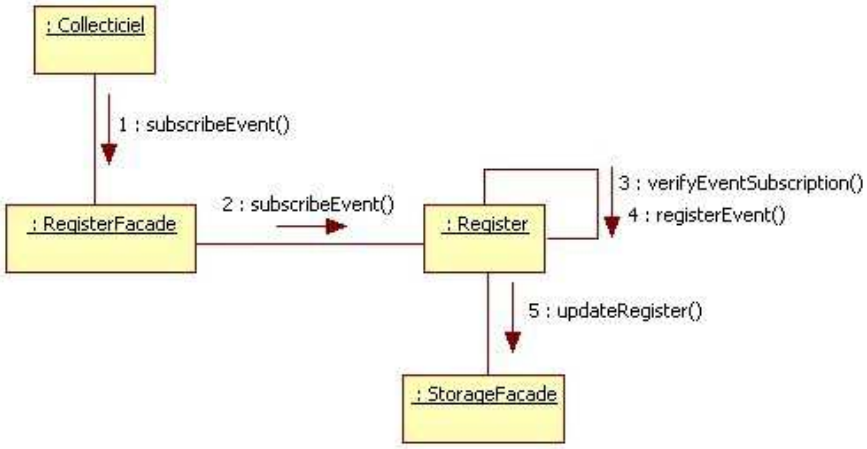


Figure 67. Processus d'enregistrement d'une nouvelle classe d'événement.

En dehors des demandes d'enregistrement, le moniteur est également la sous-unité du canevas BW-*M* responsable du traitement des requêtes concernant l'*occurrence* d'un événement au sein du collecticiel. Lorsque le collecticiel constate la réalisation d'une action relative à une classe d'événement observée, il génère l'objet `Événement` correspondant à cette action précise. Nous considérons que le collecticiel est en meilleure position pour collecter les informations relatives à cette action, grâce au contact direct qu'il entretient avec chaque

utilisateur connecté au système et grâce au contrôle qu'il a des informations relatives aux équipes et aux utilisateurs. Grâce à ces informations et aux informations sur le contexte courant de l'utilisateur auteur de l'action, le collecticiel est donc en mesure de produire l'instance de la sous-classe d'`Événement` relative à l'action en question, ainsi que les instances du modèle de contexte relatives au contexte de production de l'événement et aux éléments concernés par celui-ci (cf. section 7.2.2). Toutes ces instances sont communiquées au moniteur par le biais de la façade de contrôle et traitées par une classe nommée `Monitor` (voir Figure 66).

Lorsque le moniteur est informé de l'occurrence d'un événement, il vérifie si celui-ci correspond à une classe d'`Événement` enregistrée par le collecticiel. Ceci est réalisé à travers la classe `Register`, qui vérifie l'appartenance d'un objet `Événement` aux classes maintenues dans le registre. Si l'objet `Événement` appartient effectivement à l'extension d'une classe souscrite par le collecticiel, le moniteur introduit celui-ci à la base d'événements disponibles. Celle-ci fait partie des connaissances maintenues par le canevas BW-$\mathcal{M}$ à travers l'unité de gestion de connaissances. La classe `Monitor`, qui traite l'occurrence de l'événement, sollicite donc à la façade de stockage l'introduction du nouvel objet `Événement` dans la base de connaissances gérée par la couche de stockage. Ce processus est illustré par le diagramme de séquence présenté dans la Figure 68.

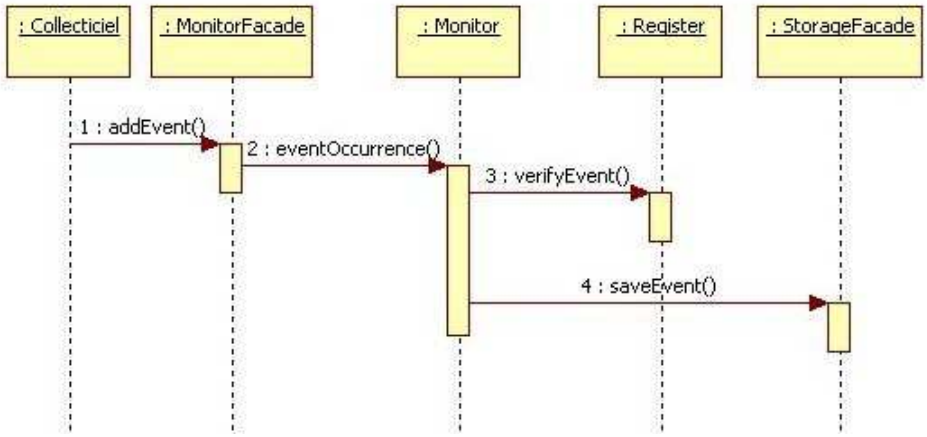


Figure 68. Traitement de l'occurrence d'un événement.

Le filtrage

Le filtrage est la sous-unité de gestion de conscience de groupe responsable de la réalisation du processus de filtrage des événements dans le canevas BW-$\mathcal{M}$. Ce processus est centralisé par une classe `Filter` (voir Figure 69), laquelle implémente le processus décrit dans le chapitre 7. Cette classe reçoit de la façade de filtrage, représentée par l'interface `FilteringFacade` (Figure 69), les demandes de filtrage. Chaque demande est réalisée pour un utilisateur cible, informé par la façade de filtrage au moment de la demande. Il est important de noter que l'utilisateur cible peut être à l'origine de la demande, si le collecticiel lui permet de solliciter les informations de conscience de groupe. Qu'il soit ou non à l'origine, la connaissance de l'utilisateur cible guide le processus de filtrage, puisque tant la sélection des profils que le filtrage des événements se fait en tenant compte de cet utilisateur, son contexte courant et les profils dont il dispose.

La classe `Filter` implémente le processus de filtrage (cf. section 7.3), lequel utilise le contexte courant de l'utilisateur cible comme paramètre. Ce processus utilise également plusieurs des opérations définies sur le modèle de contexte (voir chapitre 6). L'unité de

gestion de contexte (voir section 8.1.2.2) est responsable tant de la gestion des informations concernant le contexte courant des utilisateurs actifs dans le système, que de la réalisation de opérations sur les instances du modèle de contexte. Par conséquent, afin de récupérer les informations contextuelles nécessaires et d'effectuer les opérations sur ces informations, le filtrage utilise les services de l'unité de gestion de contexte, à travers la façade de contexte, représentée par l'interface `ContextFacade` (voir Figure 69). Par ailleurs, la classe `Filter` sollicite également les services d'autres unités du canevas BW-$\mathcal{M}$, telles que l'unité de gestion de connaissances, par le biais de la façade de stockage (interface `StorageFacade` dans la Figure 69), et de la sous-unité de gestion du modèle d'accès progressif, à travers l'interface `PAMFacade`.

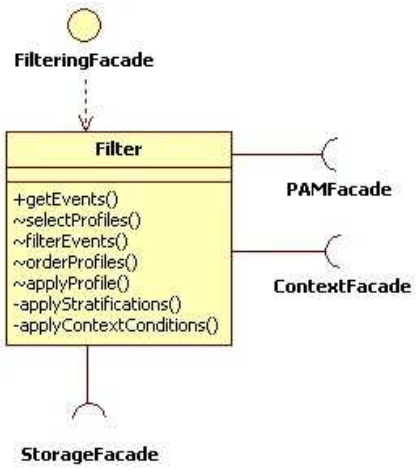


Figure 69. Éléments du canevas BW-$\mathcal{M}$ intervenant sur le filtrage des événements.

Ainsi, dès lors la classe `Filter` reçoit la demande, le processus de filtrage est déclenché, comme le montre la Figure 70. Dans cette figure, un diagramme de collaboration résume les principales interactions entre la classe `Filter` et les façades qui sont accessibles depuis cette classe. Le processus de filtrage commence donc par une requête « `getEvents` ». Cette requête déclenche, dans la classe `Filter`, la réquisition du contexte courant de l'utilisateur cible. La réquisition est faite à la façade de contexte, qui interagit directement avec l'unité de gestion de contexte. Une fois que le contexte courant de l'utilisateur est connu, la première étape du processus de filtrage peut débuter (action n°4 dans le diagramme de la Figure 70). Dans cette étape, la classe `Filter` récupère les profils disponibles auprès de la façade de stockage afin de sélectionner ceux dont le contexte d'application permet leur utilisation. Cette sélection de profils suit le processus indiqué dans la section 7.3.1, lequel est illustré par la Figure 71.

Une fois que les profils applicables au contexte courant de l'utilisateur ont été sélectionnés, la seconde étape du processus de filtrage peut démarrer (cf. section 7.3.2). Cette étape, représentée par l'opération `filterEvents` (action n° 7 dans la Figure 70) vise à filtrer les événements disponibles selon les règles déterminées par les profils sélectionnés dans l'étape précédente. Ainsi, la classe `Filter`, lors de l'exécution de cette deuxième étape, va ordonnancer les profils sélectionnés et appliquer chaque profil de manière indépendante. L'application de chaque profil implique : (*i*) la récupération, auprès de la façade de stockage, des événements pour lesquels le profil est abonné ; (*ii*) l'application des stratifications, à l'aide du modèle d'accès progressif ; (*iii*) et l'application des conditions contextuelles fixées par le

profil, à l'aide de l'unité de gestion de contexte. Le diagramme représenté dans la Figure 72 résume cette étape du processus.

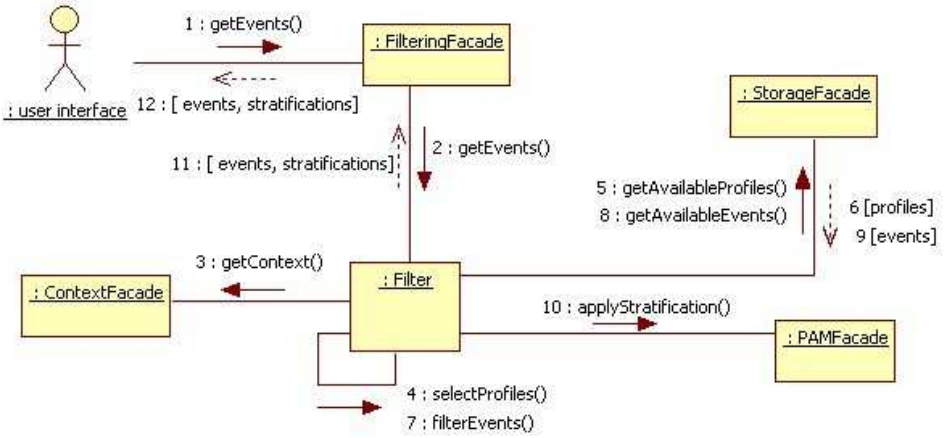


Figure 70. Diagramme de collaboration résumant le filtrage des événements.

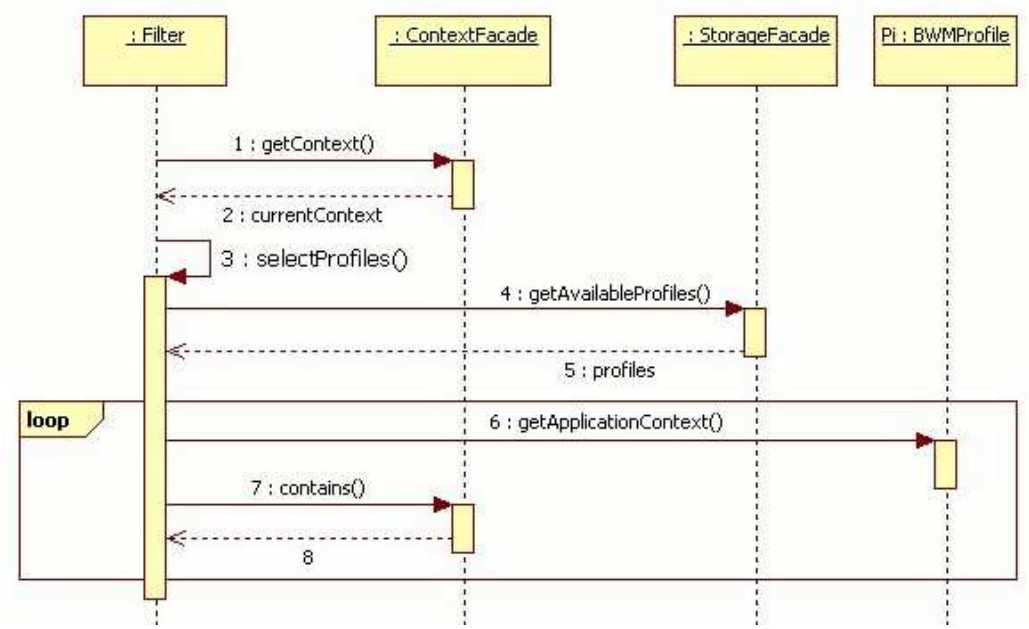


Figure 71. Diagramme de séquence mettant en évidence les collaborations qui ont lieu durant la sélection des profils.

Dans la Figure 72, nous pouvons observer que l'application de chaque profil implique la collaboration de plusieurs objets : le profil lui-même (représenté par l'objet `Pi` dans la Figure 72) ; les événements (représentés par l'objet `e`), dont les informations contextuelles sont comparées aux conditions contextuelles (opérations `11` à `17` de la Figure 72) ; et les façades de stockage (`StorageFacade`), de contexte (`ContextFacade`), et celle du modèle d'accès progressif (`PAMFacade`). Cette dernière donne accès aux fonctionnalités mises en œuvre par la sous-unité de gestion du modèle d'accès progressif, dont le contenu est présenté ci-dessous.

Le résultat du processus de filtrage (voir Figure 73) est un ensemble ordonné d'événements et les stratifications applicables à ces événements (cf. section 7.3.2). La liste de

stratifications est transmise afin que le MAP, dans le couche de présentation, puisse contrôler l'application des opérations (*masquer/dévoiler*, *avancer/retourner*, etc.) sur ces stratifications. Par ailleurs, les événements sont ordonnés par l'application des stratifications extensionnelles, qui organisent les instances en plusieurs niveaux.

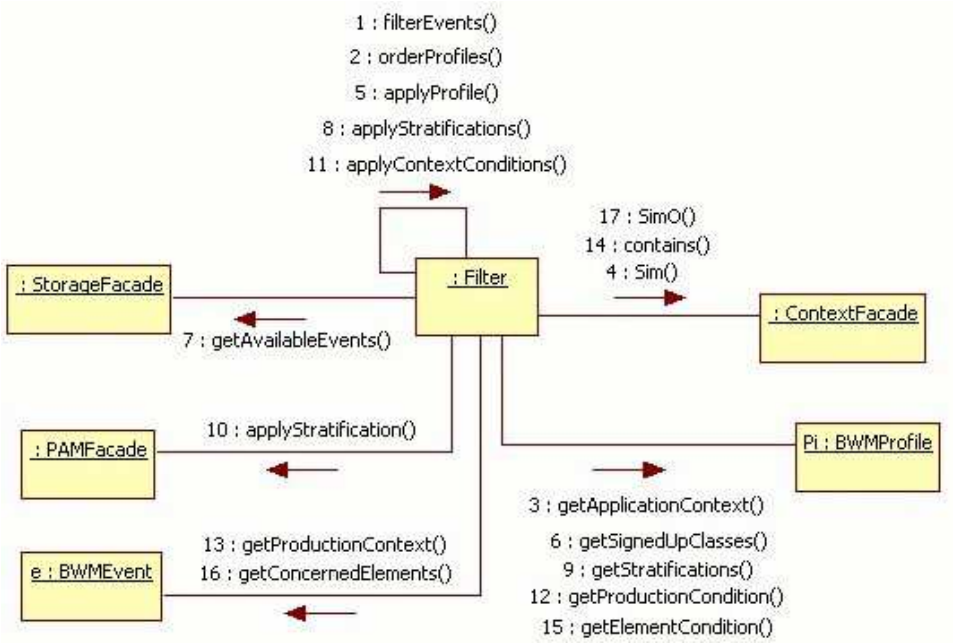


Figure 72. Collaborations mises en œuvre durant la seconde étape du processus de filtrage.

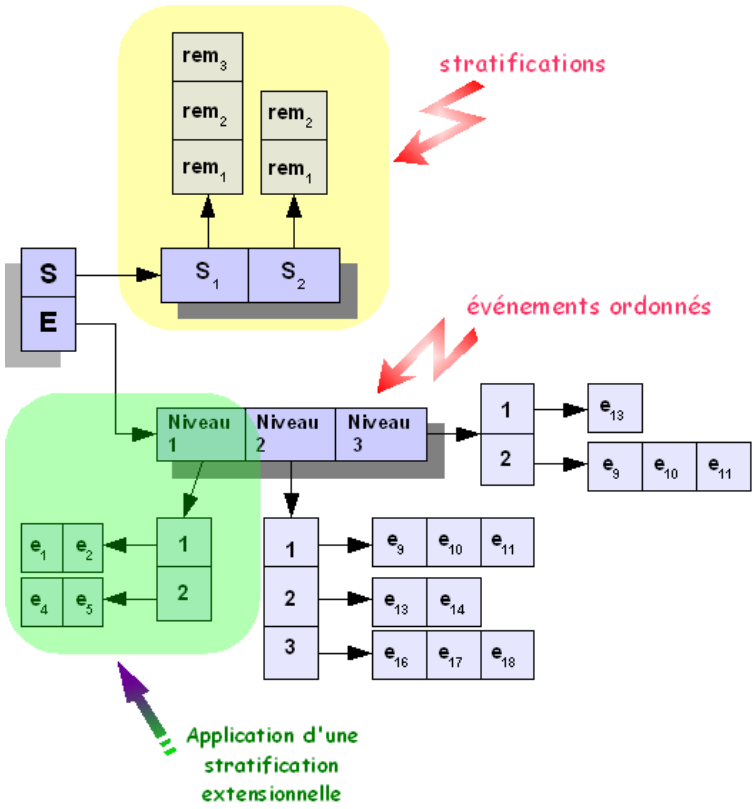


Figure 73. Structure de données retournée par le processus de filtrage.

Le MAP

La dernière sous-unité qui compose l'unité de gestion de conscience de groupe est celle responsable de la gestion du *modèle d'accès progressif* (MAP). Cette sous-unité implémente les concepts présentés dans la section 7.2.1 (voir Figure 74). Nous avons donc la notion de stratification (classe `PAMStratification` dans la Figure 74), qui est spécialisée en stratification intensionnelle et extensionnelle, et la notion de représentation d'entité masquable (classe `PAMRoME`).

La notion de stratification extensionnelle a été particulièrement spécialisée pour l'application du MAP au modèle de contenu. Cette spécialisation (classe `EventStratification` dans la Figure 74) est accompagnée de la spécialisation de la notion de représentation d'entité masquable propre aux événements (classe `EventExtRoME` dans la Figure 74). Ces spécialisations permettent notamment la définition de stratifications extensionnelles sur la valeur de l'attribut `intervalle` de la classe `Événement`. Cette spécialisation, telle que nous l'implémentons, permet la création, à travers l'application d'une telle stratification, de la structure d'événements en plusieurs niveaux nécessaire au processus de filtrage. En d'autres termes, l'application de la méthode `apply` de la classe `EventStratification` génère la structure d'événements mise en évidence dans la Figure 73.

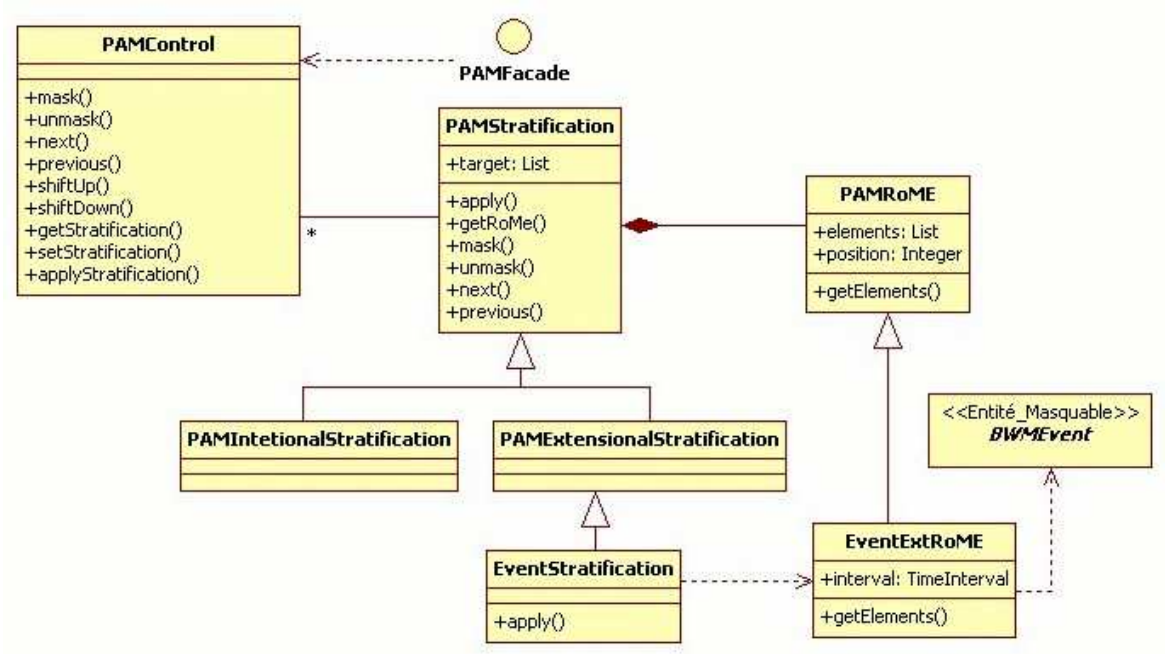


Figure 74. Implémentation du modèle d'accès progressif.

De plus, cette sous-unité de gestion du MAP contrôle la validité des stratifications, c'est-à-dire, si les stratifications appliquées respectent les règles imposées par le modèle d'accès progressif (voir [Villanova 2002]). Ceci est réalisé par la classe `PAMControl` (voir la Figure 74), qui contrôle la définition et la validité des stratifications, ainsi que l'application des opérations définies par le MAP.

Par ailleurs, la sous-unité de gestion du MAP possède une particularité par rapport aux autres unités qui composent le canevas BW-*M*. Contrairement aux autres unités, qui se focalisent sur une seule couche de l'architecture (contrôle ou stockage), le MAP participe à deux couches distinctes : la couche de contrôle et celle d'interface (Figure 64). Ceci est nécessaire car, afin de présenter correctement les informations, la couche d'interface doit

également manipuler les concepts introduits par le modèle d'accès progressif. La présentation de l'information de conscience de groupe doit respecter les stratifications définies par les profils en application. Ceci implique pouvoir manipuler ces stratifications et appliquer les opérations définies dans la section 7.2.1. Par conséquent, il faut que l'unité de gestion de l'accès progressif soit également présente auprès des composants de gestion d'interface. Ces composants doivent donc s'appuyer sur la gestion du MAP pour contrôler la présentation des événements selon les stratifications.

8.1.2.2 La gestion de l'information contextuelle

L'unité de *gestion de contexte* est l'élément du canevas BW-*M* responsable de la gestion des informations contextuelles. Autrement dit, cette unité maintient notamment les informations relatives au contexte courant de chaque utilisateur actif dans le système. Elle reçoit du collecticiel, plus particulièrement des composants responsables de l'acquisition de contexte, les informations relatives à chaque utilisateur et les traite, en respectant le modèle de contexte (chapitre 5). Par ailleurs, l'unité de gestion de contexte est également responsable de la gestion du modèle de contexte, qui est ainsi capable d'évoluer au cours de temps. De plus, la gestion du modèle passe également par l'implémentation des opérations qui ont été définies dans le chapitre 6. Afin d'accomplir ses responsabilités, cette unité de gestion se divise, à l'instar de l'unité de gestion de conscience groupe, en plusieurs sous-unités (voir Figure 75) : (*i*) *façades* d'*acquisition*, de *contexte* et de *stockage* ; (*ii*) *interprétation* ; (*iii*) *gestion des instances* ; et (*iv*) *gestion du modèle*. Ces sous-unités se partagent les responsabilités, de manière à ce que chacune d'entre elles s'occupe d'un aspect précis de la gestion du contexte.

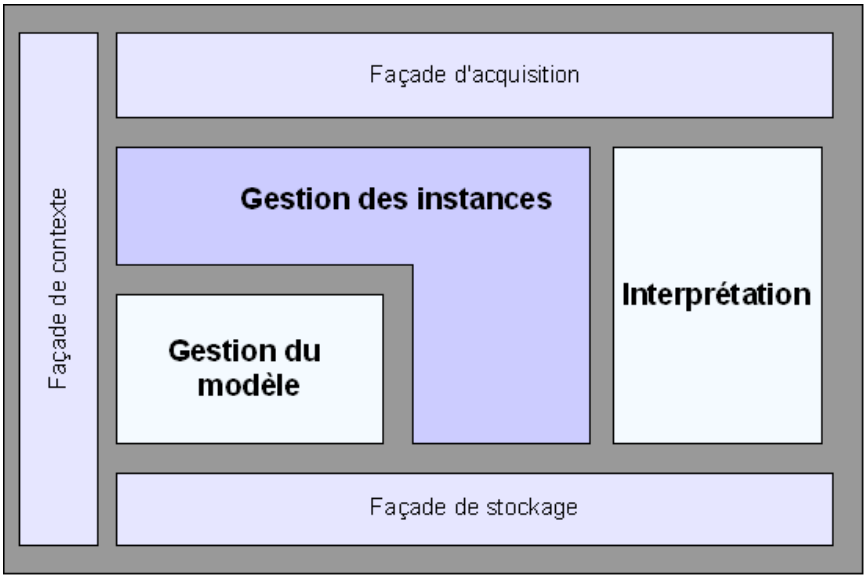


Figure 75. Éléments de l'unité de gestion de contexte du canevas BW-*M*.

Les façades ont, dans l'unité de gestion de contexte, la même fonction que dans l'unité de gestion de conscience de groupe. Elles isolent le contenu des unités et rendent uniforme l'accès à leurs fonctionnalités. D'ailleurs, nous pouvons observer que l'unité de gestion de conscience de groupe et cette unité de gestion de contexte ont deux façades en commun (voir Figure 65 et Figure 75) : celle qui donne accès à l'unité de gestion de connaissances (la *façade de stockage*), et celle qui donne accès à l'unité de gestion de contexte à partir de l'unité de

gestion de conscience de groupe (la *façade de contexte* que nous avons discuté dans la section 8.1.2.1). Néanmoins, nous pouvons observer, dans la Figure 75, que l'unité de gestion de contexte possède une façade propre à elle, la *façade d'acquisition*. Celle-ci sert d'interface entre le canevas BW-$\mathcal{M}$ et le système, et plus particulièrement entre les composants d'acquisition de contexte du système et la gestion du contexte proposée par le canevas. C'est donc à travers cette interface que le canevas BW-$\mathcal{M}$ reçoit les informations contextuelles capturées par le processus d'acquisition de contexte mise en place par le système.

Outre les façades, l'unité de gestion de contexte comporte trois sous-unités principales (Figure 75). L'unité d'*interprétation* est conçue pour admettre l'interprétation des informations contextuelles et l'extraction de nouvelles connaissances à partir des informations reçues (par exemple, la correspondance entre un ensemble de coordonnées et une zone géographique pré-définie). D'autre part, la sous-unité de *gestion des instances* manipule les instances du modèle de contexte disponibles dans la base de connaissances (laquelle est maintenue par l'unité de gestion de connaissances (cf. section 8.1.2.3). Cette sous-unité contrôle donc les instances de la classe `Description_de_Contexte` qui représentent le contexte courant de chaque utilisateur actif. Elle est également responsable du processus d'instanciation de la classe `Élément_de_Contexte`. Enfin, la sous-unité de *gestion du modèle* traite de la structure du modèle, permettant que des modifications soient apportées au modèle de contexte en cours d'exécution. Par ailleurs, en parallèle à la manipulation du modèle, cette sous-unité implémente également les opérations décrites dans le chapitre 6. Nous détaillons ensuite chacune de ces sous-unités.

L'interprétation

L'intérêt de l'interprétation des informations contextuelles réside notamment sur la possibilité d'acquérir des nouvelles connaissances à partir de celles acquises durant la détection du contexte. Le cas le plus connu d'interprétation est la mise en correspondance entre un ensemble de coordonnées obtenues par un GPS (latitude et longitude) et une adresse (nom et n° de rue, quartier, etc.). Néanmoins, nous pouvons également imaginer d'autres exemples d'interprétation possibles, tel que la déduction de l'activité de l'utilisateur à partir de sa localisation et de son emploi de temps.

À travers les exemples ci-dessus, nous pouvons observer que l'interprétation dépend, en partie, du système dans lequel elle est appliquée. Le canevas BW-$\mathcal{M}$ se doit donc de présenter une solution générique qui peut être spécialisée lors de l'application du canevas à un système précis. Ainsi, nous proposons, dans cette sous-unité, un gestionnaire d'interprétation, représenté de la classe `InterpretationHandler` (Figure 76), lequel met en œuvre le patron de conception « *chaîne de responsabilité* »[92] [Gamma 1994]. L'objectif de ce patron est d'éviter le couplage direct entre l'émetteur d'une requête et les récepteurs, à travers la définition d'une chaîne de récepteurs potentiels à la requête. La requête est donc transmise d'objet à objet dans la chaîne, jusqu'à ce que l'un d'entre eux la traite effectivement. L'application de ce patron de conception permet non seulement que le client (celui qui fait la requête) puisse ignorer quel sera le récepteur exact de sa requête, mais elle permet également d'introduire dynamiquement de nouveaux récepteurs à la chaîne. Ceci est particulièrement intéressant pour l'interprétation des informations contextuelles, car chaque nouveau récepteur dans la chaîne fournit une interprétation distincte des instances d'`Élément_de_Contexte`.

Ainsi, la sous-unité d'interprétation est constituée par une chaîne d'interprètes, chaque interprète pouvant analyser les informations contextuelles et déduire des nouvelles connaissances, qui seront, à leur tour, introduites dans la base de connaissances accessible à

[92]De l'original en anglais, « *chain of responsibility* ».

travers la façade de stockage (interface `StorageFacade` dans la Figure 76). Les concepteurs d'un collecticiel utilisant le canevas BW-$\mathcal{M}$ sont donc en mesure d'introduire de nouveaux interprètes à travers la spécialisation de la classe `InterpretationHandler` et à travers leur insertion dans la chaîne de responsabilité. À ce titre, le canevas BW-$\mathcal{M}$ propose, par définition, un interprète pour les associations, lequel est représenté par la classe `AssociationInterpreter` (voir Figure 76). Cet interprète cherche à déduire l'existence des tuples reliant les objets qui composent le contexte d'un utilisateur et qui n'ont pas été détectés par le processus d'acquisition. Le modèle de contexte présenté (chapitre 5) propose un certain nombre d'associations reliant les sous-classes d'`Élément_de_Contexte`. Cependant, nous savons que certains tuples de ces associations peuvent n'être pas détectés au moment de l'acquisition de contexte. L'interprète proposé par le canevas BW-$\mathcal{M}$ agit dans ces cas où le processus d'acquisition capture les informations relatives aux éléments de contexte, mais non celles relatives aux associations liant ces éléments.

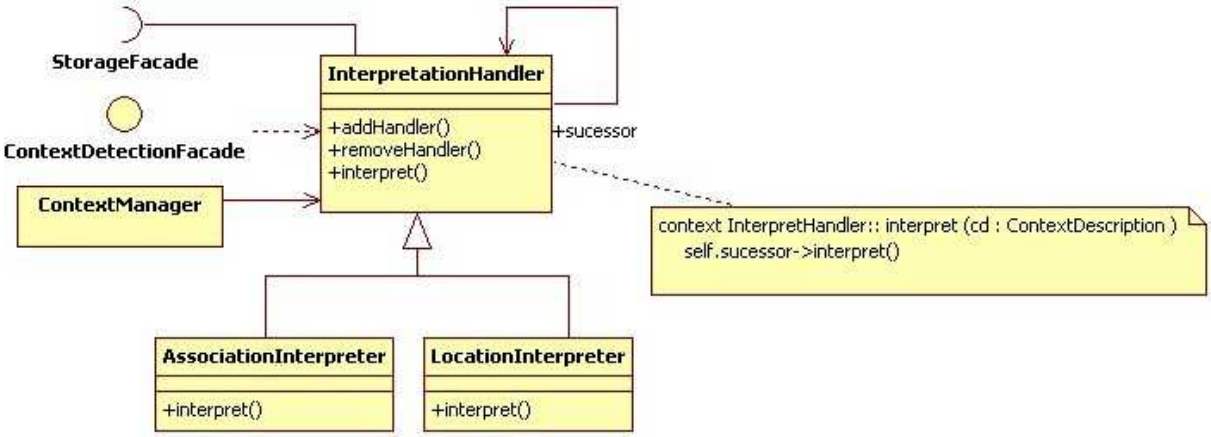


Figure 76. Structure responsable de l'interprétation des informations contextuelles.

L'interprète `AssociationInterpreter` focalise particulièrement sur les associations qui composent le contexte collaboratif, et notamment l'association `appartient`, qui relie les classes `Groupe`, `Membre` et `Rôle` (voir Figure 20, page 83). L'objectif est de déduire l'existence de ces tuples à partir des connaissances disponibles. Par exemple, lorsque le processus d'acquisition détecte la présence d'un utilisateur précis, l'interprète cherche, à l'intérieur de la base de connaissances qui est disponible à travers la façade de stockage, les groupes auxquels le membre appartient, et les rôles qu'il joue dans chaque groupe. À partir des informations qu'il retrouve, l'interprète introduit dans la description du contexte courant de l'utilisateur les tuples de l'association `appartient` et les objets relatifs à ces groupes et aux rôles joués par cet utilisateur. L'interprète utilise ainsi les connaissances statiques à propos des utilisateurs et des groupes qui sont emmagasinées par l'unité de gestion de connaissances du canevas BW-$\mathcal{M}$ (cf. section 8.1.2.3).

Hormis cet interprète pour les associations, d'autres interprètes sont possibles. Nous préconisons, par exemple, la définition d'un interprète capable de traiter les informations relatives à la localisation. Nous croyons qu'une interprétation possible pour la localisation serait de traduire un ensemble de coordonnées GPS vers une zone géographique prédéterminée. En somme, le système dispose d'un cadastre de zones géographiques les plus appréciées par les utilisateurs (ou une liste de zones où l'utilisation du système est permise). Dans un tel cadastre, chaque zone est identifiée par le polygone qui la représente (suivant ainsi les indications du consortium *OpenGis*[93]). Ainsi, l'interprète vérifie si le point

[93] *Open Geospatial Consortium (OGC)* - http://www.opengeospatial.org/.

représentant la localisation d'un utilisateur se trouve à l'intérieur d'une zone géographique, et donc déduit que l'utilisateur lui-même s'y trouve. Dans ce cas, à partir d'une information de bas niveau (les coordonnées GPS), l'interprétation déduit une information de plus haut niveau (la zone géographique dans laquelle un utilisateur se trouve en ce moment).

De plus, il est important d'observer que les fonctionnalités de la sous-unité d'interprétation sont disponibles la façade d'acquisition (interface `ContextDectionFacade`) et la sous-unité de *gestion des instances*, à travers la classe `ContextManager`. La première sert de liaison entre le collecticiel et le sous-unité d'interprétation, ce qui permet au premier d'utiliser les fonctions d'interprétation pendant le processus d'acquisition de contexte, et non seulement au moment où le canevas BW-$\mathcal{M}$ reçoit les informations capturées par ce processus. La seconde est utilisée pour la gestion des instances à chaque fois qu'un changement dans le contexte courant d'un utilisateur actif est signalé par le collecticiel. Un changement peut impliquer des nouvelles connaissances ou une restructuration dans les connaissances disponibles à propos d'un utilisateur, ce qui peut donner lieu à de nouvelles interprétations.

La gestion des instances

La sous-unité de gestion d'instances est responsable de la manipulation des instances du modèle de contexte. Les éléments qui composent cette sous-unité réalisent la gestion du contexte des utilisateurs actifs et manipulent les instances du modèle de contexte. Pour cela, cette sous-unité est composée par trois éléments (voir Figure 77) : (*i*) la gestion des éléments actifs, réalisée par la classe `ActiveElementsManager` ; (*ii*) le contrôle de connaissances contextuelles, réalisé par la classe `ContextManager` ; et (*iii*) les constructeurs d'instances, mis en œuvre par les classes `ContextDescriptionBuilder`, `ContextElementBuilder` et `ContextAssociationBuilder`. La *gestion des éléments actifs*, réalisée par la classe `ActiveElementsManager`, consiste à maintenir à jour le contexte courant de chaque élément actif dans le système. Un élément actif est un élément dont le contexte est observé par le système. Il s'agit typiquement des utilisateurs qui sont connectés à un moment donné. La *gestion de connaissances* est responsable notamment du traitement des besoins relatifs au modèle de contexte des autres unités du canevas BW-$\mathcal{M}$. Par ailleurs, la classe `ContextManager`, qui réalise cette gestion, est également responsable de l'interface avec la couche de stockage. Elle détermine la création, la sauvegarde et la suppression des instances du modèle de contexte. La création proprement dite des instances est la responsabilité des *constructeurs* qui peuvent, à partir de la structure du modèle de contexte, créer et modifier les instances de la classe `Description_de_Contexte` (classe `ContextDescriptionBuilder` dans la Figure 77), de chaque sous-classe d'`Élément_de_Contexte` (classe `ContextElementBuilder`) et de chaque association (classe `ContextAssociationBuilder`) reliant ces éléments.

La classe `ActiveElementManager`, qui réalise la gestion des éléments actifs, reçoit du système, à travers la façade d'acquisition (interface `ContextDetectionFacade` dans la Figure 77), les informations contextuelles qui ont été observées par les composants d'acquisition de contexte du système. Chaque utilisateur connecté au système (membre d'un ou plusieurs groupes) est, par définition, un *élément* actif observé par celui-ci. Cependant, le canevas BW-$\mathcal{M}$ admet qu'autres éléments (un dispositif, une application, etc.) puissent également être observés par le système. Lorsque le système détecte un nouvel élément qui devient actif ou un changement dans le contexte d'un élément déjà en observation, il informe le canevas BW-$\mathcal{M}$ de ce changement, à travers la façade d'acquisition. Celle-ci repasse la requête à la classe `ActiveElementManager`, laquelle met immédiatement à jour la liste d'éléments actifs qu'elle maintient et déclenche, auprès de la classe `ContextManager`, les mesures nécessaires pour la création ou la modification des instances de `Description_de_Contexte` qui représentent le contexte courant de l'élément actif en question.

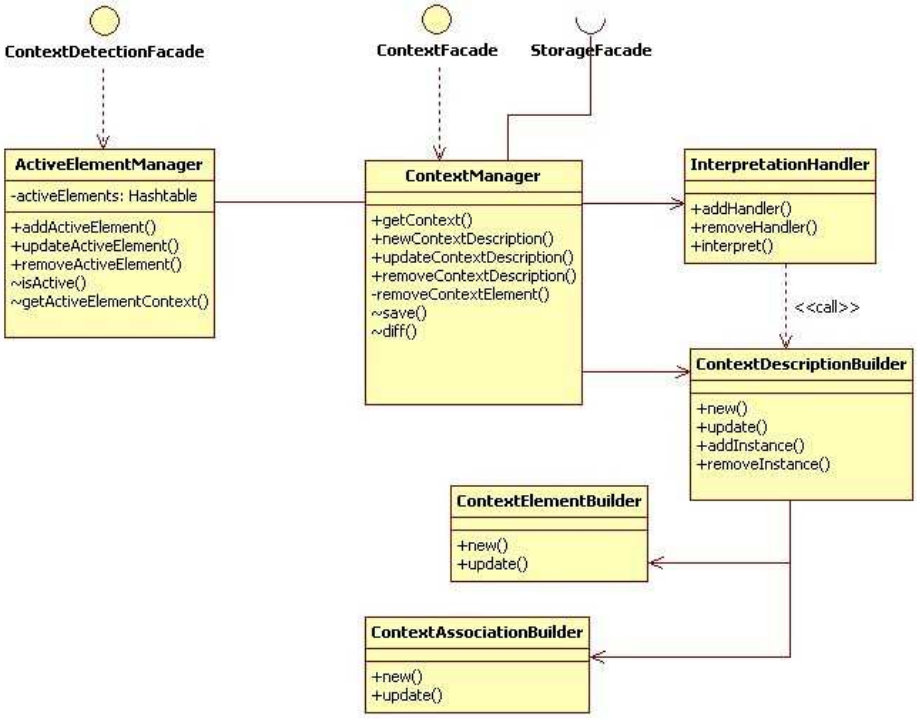


Figure 77. Classes qui composent la gestion des instances.

La Figure 78 illustre l'introduction d'un nouvel élément actif. Lorsque le système détecte le contexte d'un nouvel élément actif (un utilisateur qui vient de se connecter au système, par exemple), il informe le canevas BW-$\mathcal{M}$ à travers la façade d'acquisition. Celle-ci passe la requête à la classe `ActiveElementManager`, laquelle vérifie s'il s'agit vraiment d'un nouvel élément. Si c'est le cas, elle demande au `ContextManager` la création d'une nouvelle instance de `Description_de_Contexte` avec les informations fournies par le système. Ces informations sont transmises à travers la façade d'acquisition en format XML, suivant le schéma défini par le Système de Représentation de Connaissances par Objets AROM[94] (voir section 5.2.2), et elles sont ensuite traduites vers les objets et les tuples qui leur correspondent par la classe `ContextDescriptionBuilder`. Une fois que l'objet `Description_de_Contexte` est créé, le `ContextManager` ordonne sa sauvegarde dans la base de connaissances, par le biais de la façade de stockage (interface `StorageFacade`).

La procédure lorsqu'il y a un changement dans le contexte d'un élément actif (par exemple, lorsqu'un utilisateur change d'activité ou de localisation) est similaire à celle présentée dans la Figure 78. En revanche, au lieu de créer une nouvelle instance de `Description_de_Contexte`, la gestion des éléments actifs demande la mise à jour de la description préexistante. Ainsi, lorsqu'un changement est observé, le système informe le canevas, à travers la façade d'acquisition, des modifications survenues dans le contexte de l'élément actif. Selon l'ampleur des changements, le système transmet soit le nouveau contexte entièrement, soit seules les nouvelles informations dont il dispose. La classe `ContextManager` analyse ces informations transmises avec la demande de mise à jour. Dans le cas où le système aurait transmis une nouvelle description de contexte, la classe `ContextManager` remplace tout simplement la description préexistante par la nouvelle. Dans le cas où seules les instances qui ont été modifiées ont été transmises, cette même classe compare l'instance

[94]Le schéma XAROM, proposé par le système AROM, est présenté dans l'Annexe III.

`Description_de_Contexte` qui représente le contexte courant de l'élément et les nouvelles informations. À partir de cette comparaison, le contexte courant de l'élément en question est mis à jour par l'introduction des nouvelles instances (objets `Élément_de_Contexte` et tuples), par la modification du contenu des instances déjà liées à cette instance de `Description_de_Contexte`, ou même par leur suppression. La Figure 79 illustre les collaborations entre les éléments de gestion de contexte qui sont nécessaires pour la mettre en œuvre.

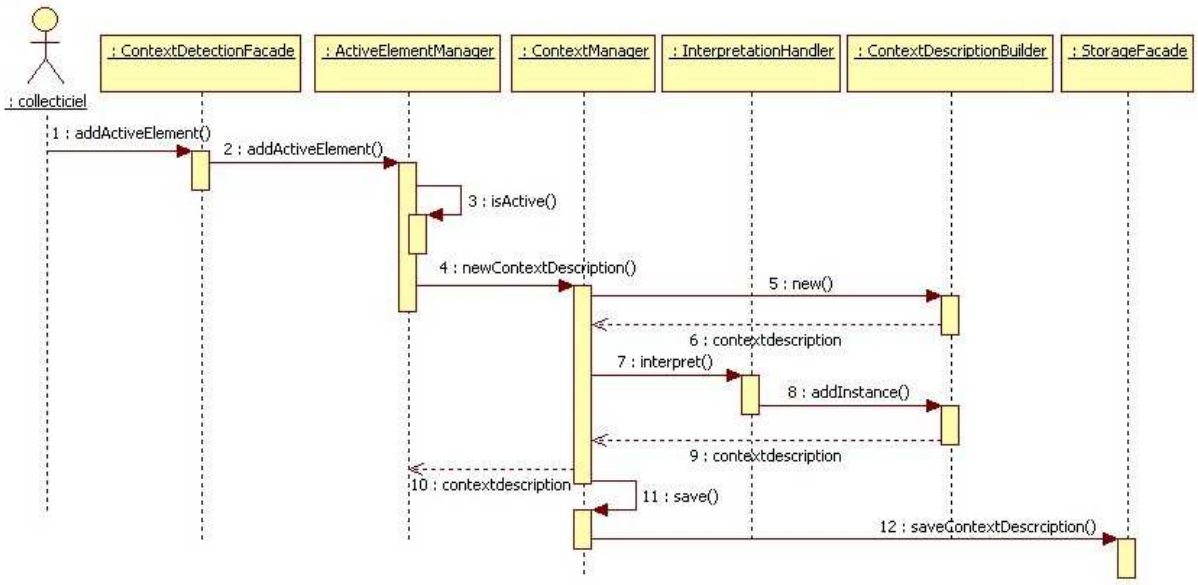


Figure 78. Diagramme de séquence représentant le processus déclenché à l'intérieur du canevas BW-*M* par l'introduction d'un nouvel élément actif dans le système.

Parmi les collaborations mises en évidence par la Figure 79, nous pouvons observer qu'afin de modifier le contenu d'une instance de `Description_de_Contexte`, le `ContextManager` compte avec l'aide des constructeurs. Chaque constructeur est capable de créer et de modifier les instances du modèle de contexte. Ce travail d'instanciation est donc isolé au sein des constructeurs, qui s'occupent de la création des nouveaux objets (à travers la classe `ContextElementBuilder`) et tuples (à travers la classe `ContextAssociationBuilder`), et plus particulièrement de la création des nouvelles instances de `Description_de_Contexte` (classe `ContextDescriptionBuilder`).

Par ailleurs, la sous-unité de gestion des instances reçoit également de requêtes venues des autres unités du canevas BW-*M*. Ceci est notamment le cas de la sous-unité de filtrage de l'information de conscience de groupe (cf. section 8.1.2.1). Celle-ci utilise les services de la gestion d'instances, à travers la *façade de contexte* (voir Figure 65), afin de consulter le contexte courant d'un utilisateur actif durant le processus de filtrage. La sous-unité de filtrage, à travers la classe `Filter`, sollicite à la façade de contexte (interface `ContextFacade`) le contexte courant de l'élément actif visé par le processus de filtrage. La façade repasse cette demande à la classe `ContextManager`, qui vérifie auprès de la classe `ActiveElementManager` s'il s'agit effectivement d'un élément actif en ce moment. Si c'est le cas, le contexte courant de cet élément est envoyé de retour au processus de filtrage. En revanche, si la classe `ActiveElementManager` ignore cet élément (c'est-à-dire l'élément n'est plus considéré comme actif au moment de la requête), le `ContextManager` demande à la façade de stockage le dernier contexte connu pour cet élément, et si celui-ci est toujours disponible, il sera envoyé à l'unité de gestion de conscience de groupe. La Figure 80 représente le traitement d'une telle demande.

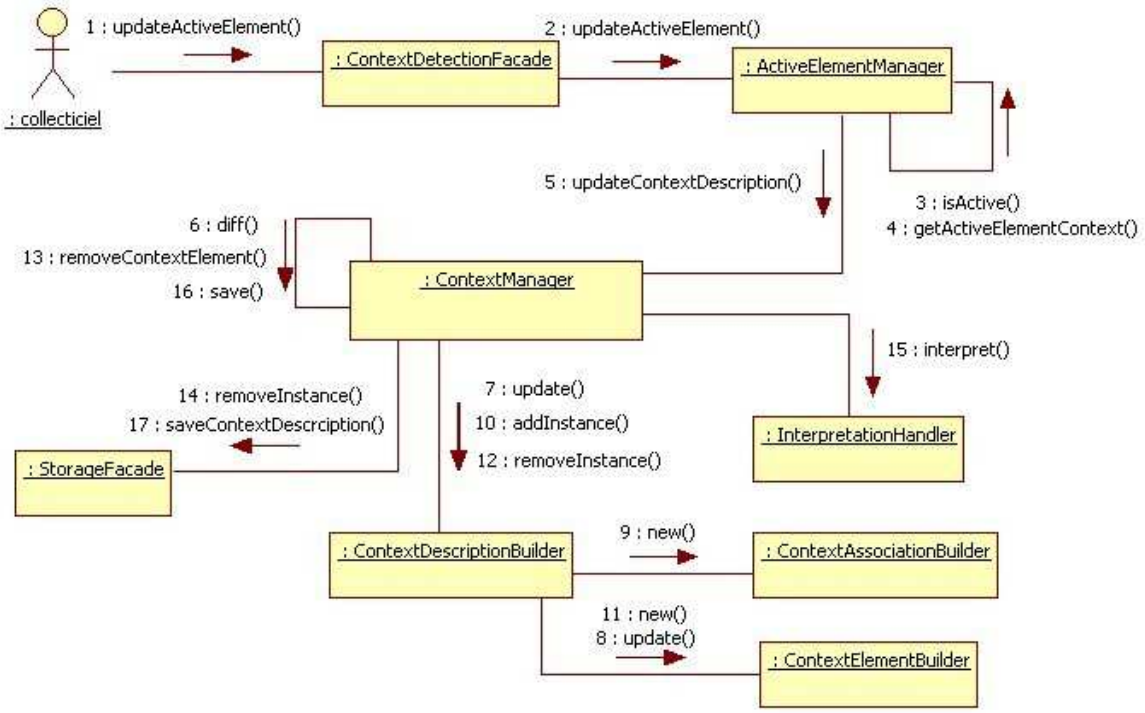


Figure 79. Diagramme de collaboration présentant un exemple de mise à jour du contexte courant d'un élément actif.

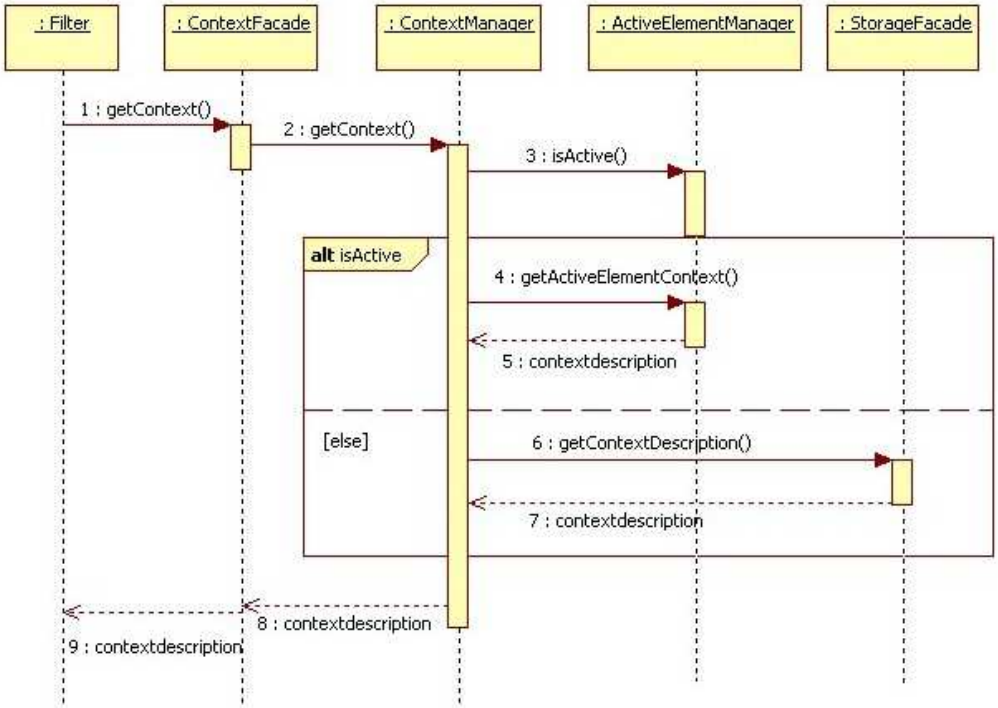


Figure 80.Diagramme de séquence illustrant une requête émanant de la sous-unité de filtrage.

Enfin, il convient d'observer que, d'une part, la façade de contexte sert principalement à la communication interne entre les éléments du canevas BW-*M*. D'autre part, la façade

d'acquisition permet au collecticiel d'informer le canevas des informations contextuelles qu'il a pu capturer. Cependant, en aucun cas, la façade d'acquisition ne permet la récupération des informations contextuelles qui sont maintenues par le canevas BW-$\mathcal{M}$, telle que la façade de contexte le permet. Par conséquent, les informations relatives au contexte courant d'un utilisateur maintenues par le canevas BW-$\mathcal{M}$ ne sont accessibles qu'aux éléments du canevas. Celui-ci ne les rend pas disponibles à l'extérieur. Il les utilise certes pour le filtrage des informations de conscience de groupe, mais il ne les transmet pas à d'autres composants du collecticiel. Ceci est le résultat d'un souci de respect de la vie privée. Néanmoins, le respect de la vie privée doit également être garanti par les composants d'acquisition du contexte au sein du collecticiel. Ces composants doivent non seulement garder le caractère privé de l'information contextuelle, mais également s'assurer qu'aucun élément de contexte n'est capturé sans le consentement de l'utilisateur. Le canevas BW-$\mathcal{M}$ ne peut pas garantir ceci, puisqu'il ne participe pas directement au processus d'acquisition de contexte. Par contre, le canevas garantit que les informations contextuelles qu'il reçoit ne sont pas utilisées pour des finalités autres que le filtrage des informations de conscience de groupe.

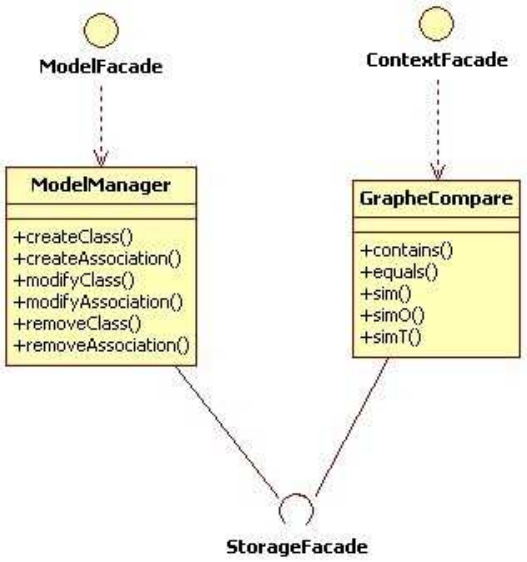


Figure 81. Classes participant à la gestion du modèle.

La gestion du modèle

La sous-unité de gestion du modèle est responsable de la maintenance du modèle de contexte. Le modèle de contexte (chapitre 5) est implémenté par le canevas BW-$\mathcal{M}$ à travers une base de connaissances AROM, laquelle est gérée par l'unité de gestion de connaissances (cf. section 8.1.2.3). Le système AROM permet que la structure de la base de connaissances soit altérée en cours d'exécution (cf. section 5.2). La sous-unité de gestion du modèle contrôle donc les modifications qui peuvent être apportées au modèle de contexte. Les fonctionnalités de la sous-unité de gestion du modèle sont fournies par la classe `ModelManager` (Figure 81). Cette classe est accessible à travers l'interface `ModelFacade`, laquelle participe à la *façade de contexte*. À travers cette interface, le collecticiel est capable de modifier le modèle de contexte et de l'adapter à ses besoins, même pendant l'exécution.

La sous-unité de gestion du modèle inclut également une implémentation des opérations définies sur le modèle (cf. chapitre 6). Cette implémentation est réalisée par la classe `GrapheCompare` (voir Figure 81), laquelle est accessible à travers la façade de contexte

(`ContextFacade`). Les opérations mises en œuvre par cette classe sont particulièrement utilisées par l'unité de gestion de conscience de groupe, qui les emploie pour le filtrage des événements (cf. section 8.1.2.1).

8.1.2.3 La gestion des connaissances

L'unité de gestion des connaissances est en charge du stockage de toutes les connaissances manipulées par le canevas BW-$\mathcal{M}$. Ces connaissances sont organisées dans une base de connaissances, laquelle contient non seulement les classes et les instances du modèle de contexte, mais également celles des autres modèles utilisés par le canevas BW-$\mathcal{M}$: modèle de contenu, modèle de profil et modèle de accès progressif. Ainsi, la base de connaissances maintenue par cette unité comporte les connaissances sur le contexte courant des utilisateurs actifs, les profils associés aux utilisateurs et les événements qui ont été produits au cours des travaux dans les équipes.

Par ailleurs, la base de connaissances maintenue par cette unité de gestion est constituée à la fois par des connaissances statiques et par des connaissances dynamiques. Les connaissances statiques sont celles dont la mise à jour ne se fait pas régulièrement (c'est-à-dire, ces connaissances changent assez rarement). Ceci est le cas notamment de la structure de chaque modèle utilisé (modèle de contexte, modèle de contenu, modèle de profil et modèle d'accès progressif) et de leurs spécialisations. Les connaissances statiques incluent les classes et les associations directement liées au collecticiel et à l'environnement de travail qu'il supporte. Il s'agit, par exemple, des connaissances relatives au *processus* de coopération mis en place par le collecticiel, les *rôles* et les *activités* proposées par le collecticiel, les classes d'événements qu'il observe, ainsi que les informations plus statiques sur les *membres* du groupe et leurs *profils*. Toutes ces connaissances viennent de la spécialisation par l'instanciation des modèles proposés suite à l'application du canevas BW-$\mathcal{M}$ à un collecticiel donné.

Parmi ces connaissances statiques, nous retrouvons donc les profils qui ont été définis pour chaque utilisateur. Ceci implique les connaissances sur le profil, les classes abonnées par le profil, ainsi que les stratifications et les contextes potentiels associés à chaque profil. Les contextes potentiels sont ainsi des instances du modèle de contexte (voir section 7.2.3) qui ont un caractère statique : elles décrivent une situation dans laquelle un utilisateur peut potentiellement se trouver, et pas forcément la situation dans laquelle il se trouve actuellement.

Le contexte courant de chaque utilisateur actif dans le système fait partie des connaissances dynamiques maintenues dans la base. Ces instances du modèle de contexte sont couramment mises à jour au cours de l'interaction entre les utilisateurs et le système. Les connaissances dynamiques sont composées également par les instances d'événements qui sont produites à l'intérieur du collecticiel au fur et à mesure que le travail dans les équipes évolue. Cependant, contrairement au contexte des utilisateurs, dont les instances peuvent être écartées une fois que l'utilisateur ne sera plus actif, les événements restent dans la base de connaissances. La dynamique ici réside dans l'introduction fréquente de nouveaux événements dans la base de connaissances.

Afin de bien gérer cette base, l'unité de gestion de connaissances utilise le Système de Représentation de Connaissances par Objets (SRCO) AROM [Page 2000] [Page 2001]. Ce système présente une API qui permet la manipulation des bases de connaissances à partir d'une application (voir [Page 2000] [Bruley 2003]). L'unité de gestion interagit directement

avec cette API (voir Figure 82), afin de garantir l'exécution des toutes les demandes venues des autres éléments du canevas BW-*M*.

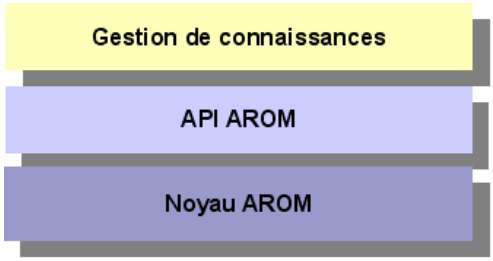


Figure 82. Organisation de la couche de stockage dans le canevas BW-*M*.

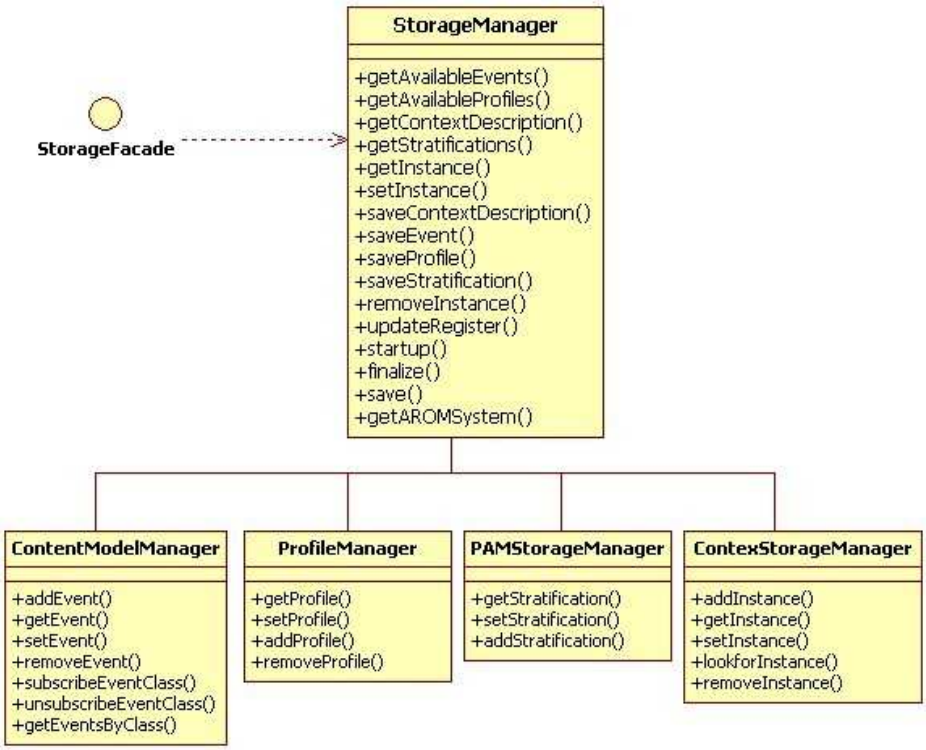


Figure 83. Les classes qui composent la gestion de connaissances du canevas BW-*M*.

Tant l'unité de gestion de conscience de groupe (cf. section 8.1.2.1), que l'unité de gestion de contexte (cf. section 8.1.2.2) dépendent des services de la façade de stockage, notamment à travers l'interface `StockageFacade`. Cette façade repasse les requêtes reçues des autres unités au contrôleur de l'unité de stockage (voir classe `StorageManager` dans la Figure 83). Ce contrôleur traite ces demandes, tout en déléguant la manipulation des instances proprement dites à un ensemble de classes qui sont spécialisées dans les modèles manipulés par le canevas BW-*M*. Ainsi, le contrôleur de l'unité de stockage s'appuie sur quatre classes qui traitent chacune les instances d'un modèle en particulier (Figure 83) : le modèle de contenu (classe `ContentModelManager`), le modèle de profil (classe `ProfilManager`), le modèle d'accès progressif (`PAMStorageManager`) et le modèle de contexte (`ContextStorageManager`). Ces classes remplissent de fonctionnalités similaires (l'insertion de nouvelles instances, la modification et la récupération d'instances préexistantes, ou encore la suppression

d'instances), mais chacune d'entre elles s'intéresse à un modèle en particulier. Par conséquent, le traitement des instances d'un modèle précis est isolé au sein d'une de ces classes de traitement.

8.1.3 Les Paquetages

Le canevas BW-$\mathcal{M}$ est écrit en langage Java et, comme tel, s'organise en différents paquetages. Ces paquetages organisent les éléments de l'architecture du canevas que nous avons présenté dans la section précédente. Ainsi, nous retrouvons dans le canevas BW-$\mathcal{M}$ quatre paquetages principaux, représentés dans la Figure 84 : `control`, `PAM`, `context` et `storage`. Le premier paquetage, `control`, englobe l'unité de gestion de conscience de groupe. Le second, `PAM`, encapsule la gestion du modèle d'accès progressif. Le troisième paquetage, `context`, présente l'unité de gestion de contexte, et le dernier, `storage`, englobe l'unité de gestion de connaissances. Chacun de ces paquetages contient les classes et les interfaces présentées dans les sections ci-dessus, souvent organisées en "sous-paquetages". De cette manière, l'unité de gestion de conscience de groupe, dans le paquetage `control`, se divise encore en deux paquetages, `control.filtering` et `control.monitoring`, qui contiennent respectivement les éléments de filtrage et de moniteur (voir Figure 84). De même, le paquetage `context`, qui contient l'unité de gestion de contexte, se divise en plusieurs paquetages, selon les fonctionnalités des classes lui appartenant (voir Figure 84) : un paquetage `control.interpretation` contient les classes d'interprétation ; un paquetage `control.manager` comporte les éléments de contrôle (gestion des instances et gestion du modèle) ; le paquetage `control.factory` englobe les constructeurs ; et le paquetage `control.util` contient l'implémentation des opérations sur le modèle de contexte.

Cependant, la Figure 84 présente également un cinquième paquetage qui ne correspond pas aux éléments de l'architecture du canevas BW-$\mathcal{M}$. Il s'agit du paquetage `kernel`. Ce paquetage est là pour représenter les informations qui sont échangées entre les paquetages. Tous les éléments qui composent le canevas BW-$\mathcal{M}$ manipulent plus ou moins les mêmes informations : les événements, les profils, les éléments de contexte, etc. En somme, les informations représentées dans les modèles de contenu, de profil et de contexte, et qui sont stockées par l'unité de gestion de connaissances. Le paquetage `kernel` fournit une représentation homogène pour ces informations, permettant leur échange entre les différents éléments du canevas.

Le paquetage `kernel` est composé essentiellement par des « interfaces » Java (voir Figure 85). Elles définissent les méthodes de manipulation des informations contenues par chaque classe, mais elles ne donnent pas une implémentation précise pour ces méthodes. Ceci est particulièrement intéressant pour leur utilisation au sein du système dans lequel le canevas BW-$\mathcal{M}$ est appliqué. Le contenu du paquetage `kernel` est visible au système, qui manipule ainsi les mêmes informations. Même si le canevas BW-$\mathcal{M}$ propose dans les façades des méthodes d'échange de données en XML, l'utilisation d'objets Java peut être préférée en certaines situations, pour des raisons de performance. Ainsi, nous proposons, dans le paquetage `kernel`, les interfaces Java pour chaque modèle utilisé au sein du canevas. Par ailleurs, nous proposons également une implémentation, à travers la classe `ConcretBWMElement`, laquelle implémente l'interface `BWMElement`. Cette classe se présente comme un « *wrapper* » pour un objet AROM représentant l'élément. En d'autres termes, chaque objet de cette classe contient une référence à l'objet dans la base de connaissances AROM qui contient les informations.

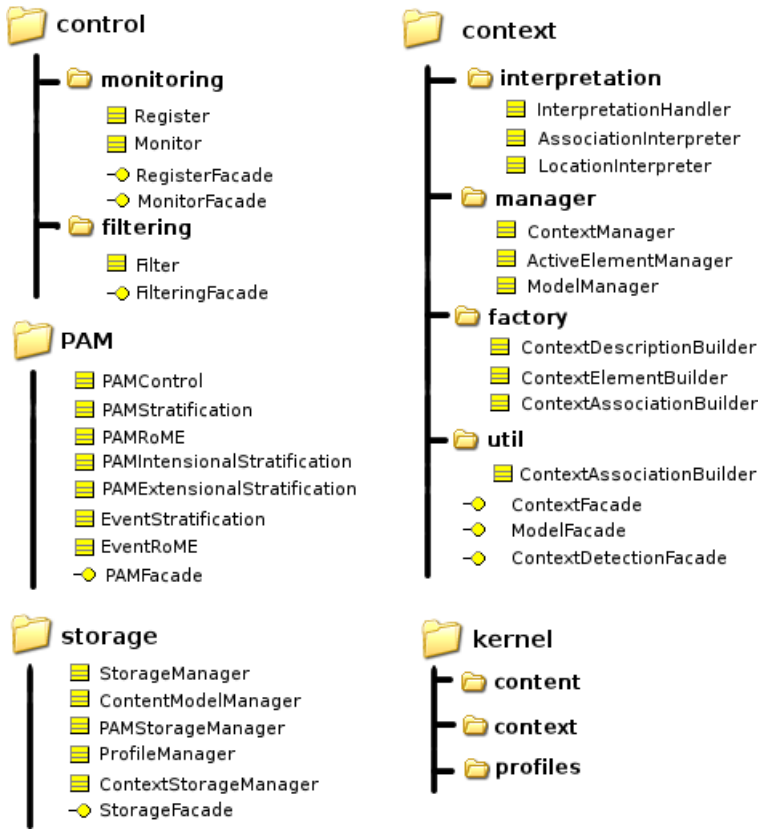


Figure 84. Paquetages Java du canevas BW-$\mathcal{M}$.

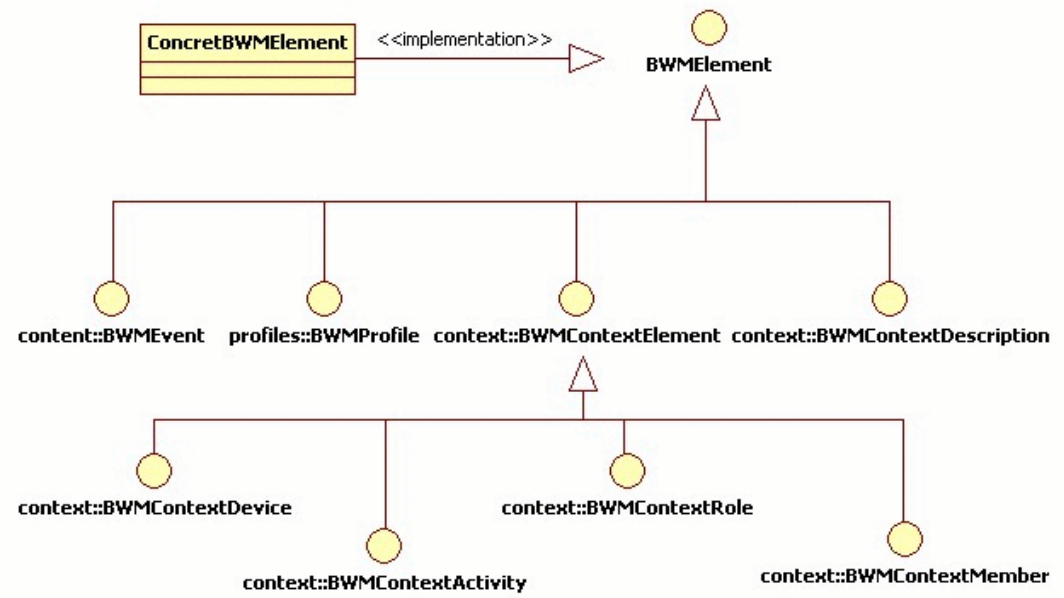


Figure 85. Contenu du paquetage `kernel`.

8.2 Méthodologie d'Application du Canevas

Le canevas BW-$\mathcal{M}$, comme tous les canevas, est conçu indépendamment d'un système précis et doit être couplé à un système pour être réellement opérationnel. Dans cette section

nous présentons les démarches nécessaires pour réaliser ce couplage entre le canevas BW-$\mathcal{M}$ et un collecticiel. Ces démarches visent à aider les concepteurs d'un collecticiel sur le Web qui désirent utiliser le canevas dans leur système. Elles expliquent comment ces concepteurs doivent appliquer le canevas BW-$\mathcal{M}$ à leur cas précis. En plus des démarches, nous présentons également quelques illustrations d'une possible application du canevas BW-$\mathcal{M}$ à de systèmes réels. Ainsi, la section 8.2.1 présente les démarches possibles pour l'application du canevas BW-$\mathcal{M}$, et la section 8.2.2 discute quelques illustrations d'application.

8.2.1 Les démarches

Le canevas BW-$\mathcal{M}$ peut être appliqué lors de la conception d'un collecticiel sensible au contexte de deux manières distinctes : d'abord à travers une démarche plus traditionnelle, dans laquelle le code Java du canevas est introduit directement dans le système qui l'applique. Puis, la seconde démarche consiste à utiliser le canevas BW-$\mathcal{M}$ en tant que service Web. Nous avons encapsulé le canevas BW-$\mathcal{M}$ à l'intérieur d'un service Web, lequel rend accessibles les fonctionnalités du canevas à travers les standards propres aux services Web (voir [Ferris 2003]), notamment le WSDL[95]. Dans les prochaines sections, nous détaillons ces démarches.

8.2.1.1 Démarche traditionnelle

La démarche traditionnelle du canevas BW-$\mathcal{M}$ consiste à utiliser directement les classes et les paquetages Java qui composent le canevas dans la conception d'un nouveau collecticiel sensible au contexte. Celle-ci est la démarche traditionnelle pour tout canevas Java, dans laquelle on importe les classes du canevas dans le produit en conception. L'interaction entre le collecticiel et le canevas BW-$\mathcal{M}$ se fait, dans ce cas, directement par les objets Java, et notamment par d'appels à de méthodes appartenant aux façades du canevas.

Ainsi, lors de l'utilisation du canevas BW-$\mathcal{M}$, les concepteurs d'un collecticiel sur la Web doivent établir les « points d'entrée », qui connectent le canevas BW-$\mathcal{M}$ au système en conception et ainsi pouvoir atteindre les fonctionnalités du canevas. Le canevas BW-$\mathcal{M}$ compte trois points d'entrée principaux qui doivent être implémentés : les *événements*, le *contexte* et l'*interface*.

Les événements

Le premier de ces points d'entrée réside donc sur la définition et l'enregistrement des événements. Initialement, les concepteurs du système doivent établir les classes d'*événements* qui seront observées par le système. Cela correspond à déterminer les types d'information de conscience de groupe qui sont livrés aux utilisateurs. Ceci revient également à définir les informations liées au processus de coopération qui font partie de chaque classe d'*événement*. Ainsi, si nous supposons un système d'édition collaborative sur le Web, ce système peut définir de classes d'*événements* pour de situations telles que la modification du document, l'ajout de commentaires ou encore la disponibilité des utilisateurs.

La définition des classes d'événements implique la spécialisation du modèle de contenu (cf. section 7.2.2). Les concepteurs sont donc invités à spécialiser la classe `Événement` selon les besoins prévus pour le collecticiel en conception. Une fois définies les sous-classes d'événements, il faut que celles-ci soit enregistrées auprès du canevas, suivant le cycle de vie

[95] « *Web Services Description Language* » - http://www.w3.org/TR/wsdl.

du canevas BW-$\mathcal{M}$. Les concepteurs déterminent donc non seulement les classes d'événements, mais également le moment à partir duquel celles-ci seront observées, en les enregistrant auprès du canevas.

Ensuite, une fois les classes d'événements définies et le moment de leur enregistrement déterminé, il faut encore mettre en place la production des événements. Les concepteurs doivent déterminer à quels moments de l'interaction entre les utilisateurs et le système les événements seront produits. Par exemple, si une classe d'événement `Présence` est observée, un nouvel événement de cette classe doit être produit à chaque fois qu'un utilisateur se connecte au système. Il s'agit donc de déterminer le moment idéal pour l'acquisition des informations de conscience de groupe qui composent chaque événement. L'événement est ainsi produit dans le collecticiel et ensuite retransmis au canevas, par le biais de la façade de contrôle, et, plus particulièrement, à travers le `MonitorFacade` (cf. section 8.1.2.1).

Le contexte

Le deuxième point d'entrée pour l'application du canevas BW-$\mathcal{M}$ réside dans l'acquisition du contexte. Afin de garder son indépendance par rapport aux technologies utilisées et aux éléments observés, le canevas BW-$\mathcal{M}$ n'interfère pas dans le processus d'acquisition du contexte. Cependant, il dépend de ce processus puisqu'il utilise les informations contextuelles pour le filtrage de l'information de conscience de groupe.

Ainsi, comme en tout système sensible au contexte, les concepteurs d'un collecticiel sensible au contexte doivent concevoir le processus d'acquisition du contexte. Ceci inclut le choix des éléments de contexte qui seront observés et des technologies qui seront utilisées pour les capturer. En ce qui concerne le choix des éléments, le modèle de contexte (cf. chapitre 5) suggère une série d'éléments particulièrement applicables aux collecticiels sur le Web. Les concepteurs sont libres de choisir, parmi ces éléments, ceux qui seront effectivement utilisés dans le système en conception, et même d'introduire d'autres éléments qui ne se trouvent pas originellement dans le modèle proposé.

Le choix des éléments observés dépend en partie des technologies disponibles pour leur capture. Les concepteurs iront choisir les éléments dont ils ont les moyens de capture à un coût acceptable pour eux. Il faut savoir que pour certains éléments de contexte, notamment la localisation, plusieurs technologies sont disponibles (utilisation de GPS [Samama 2005], du réseaux sans fil [Muñoz 2003] ou des étiquettes RFID [Anne 2005], etc.). Le choix entre une et l'autre réside dans leur disponibilité et leur coût pour les concepteurs. Même si une technologie s'avère plus performante en certaines occasions, son utilisation peut ne pas être possible si son coût d'application s'avère prohibitif pour le système en conception. Ainsi, les concepteurs d'un collecticiel sur le Web qui emploient le canevas BW-$\mathcal{M}$ doivent déterminer les éléments de contexte observés et les technologies qui seront utilisées pour cela, en tenant compte de leur intérêt pour le système et de leur coût d'application.

Il convient également de rappeler que certains travaux, tels que Dey [Dey 2000] et Rey et Coutaz [Rey 2004], proposent des architectures pour le processus d'acquisition du contexte. L'utilisation de telles architectures est conseillée, puisqu'elles permettraient une meilleure organisation du processus d'acquisition. Dans tous les cas, le collecticiel doit être en mesure d'informer le canevas BW-$\mathcal{M}$ du contexte courant de chaque élément actif.

Hormis le processus d'acquisition, les concepteurs peuvent également proposer de nouveaux interprètes qui pourront s'ajouter à la chaîne mise en place par la sous-unité d'interprétation (cf. 8.1.2.2). Il faut remarquer que la définition des nouveaux interprètes dédiés au système en conception dépend également des éléments de contexte observés. Selon les éléments observés et leur intérêt dans le système en conception, la définition d'un

interprète approprié, par la spécialisation de la classe `InterpretationHandler` (voir Figure 76), devient plus ou moins importante. Par exemple, l'élément de contexte `Activité` est typiquement un élément qui peut être sujet à une interprétation. Selon l'interaction de l'utilisateur avec le système et la situation dans laquelle il se trouve, le système peut essayer de déterminer l'activité courante de l'utilisateur. Pour cela, il faut que les concepteurs définissent un interprète pour l'élément de contexte `Activité`.

L'interface

Le troisième point d'entrée du système au canevas BW-$\mathcal{M}$ concerne l'*interface* avec l'utilisateur. Le canevas BW-$\mathcal{M}$ ne propose pas directement de composants pour l'interface avec l'utilisateur. Plusieurs raisons motivent ce choix. D'abord, la volonté d'indépendance du canevas par rapport au collecticiel dans lequel il est appliqué. Chaque système possède sa propre interface, et la présentation des informations de conscience de groupe doit s'intégrer parfaitement à cette interface. Par ailleurs, nous pouvons trouver, dans la littérature, la proposition de plusieurs *widgets* dans le but de faciliter la présentation de ces informations (voir, par exemple, [Boyle 2002] et [Hill 2004]). De plus, le canevas BW-$\mathcal{M}$ a été conçu pour une architecture client-serveur typique du Web, dans laquelle il se trouve sur le serveur. Par conséquent, il n'est pas assuré d'avoir un quelconque contact direct avec l'interface utilisateur.

Ainsi, l'interface avec l'utilisateur reste sous la responsabilité du collecticiel, et donc sous l'entière responsabilité des concepteurs du collecticiel. Cependant, l'interface proposée doit permettre à l'utilisateur de demander volontairement les informations de conscience de groupe (et par conséquent, le filtrage des événements disponibles). Les concepteurs doivent connecter le canevas BW-$\mathcal{M}$ à l'interface du système en conception à travers notamment la façade de filtrage. Celle-ci doit recevoir les demandes faites par l'utilisateur, et retourner à l'interface avec l'utilisateur le résultat du processus de filtrage.

Le résultat du processus de filtrage est donc retransmis, par le canevas BW-$\mathcal{M}$, à l'interface avec l'utilisateur. Celle-ci est responsable de l'affichage des informations rendues disponibles par le canevas. Ceci signifie que l'interface avec l'utilisateur doit également gérer l'application des stratifications, et plus particulièrement, donner à l'utilisateur l'opportunité d'appliquer les opérations sur ces stratifications. Afin d'aider le collecticiel dans cette tâche, le canevas BW-$\mathcal{M}$ propose que la sous-unité de gestion du modèle d'accès progressif soit également présente dans la couche de présentation (cf. section 8.1.2.1).

Par ailleurs, à travers la couche de présentation, les concepteurs doivent également donner aux utilisateurs la possibilité de définir leurs propres profils. Ceci est nécessaire pour que les utilisateurs puissent décrire leurs préférences et leurs besoins dans les situations qu'ils retrouvent le plus souvent (leurs contextes potentiels les plus récurrents). Néanmoins, nous encourageons les concepteurs à proposer également un ensemble de profils par défaut et de profils standards, qui pourront être modifiés et adaptés par les utilisateurs. Nous croyons que l'utilisateur doit être capable de créer/modifier un profil de manière à définir les situations dans lesquelles ce profil doit être appliqué et l'ordre dans laquelle les événements seront exhibés (ceci à travers les stratifications).

Après avoir analyser chacun des points d'entrée, nous pouvons affirmer que l'application du canevas BW-$\mathcal{M}$ à un système en particulier consiste à combler ces entrées, en fournissant au canevas les informations dont il a besoin (les événements et le contexte) et les moyens pour la présentation des informations. L'interaction entre le système et le canevas se fait à travers les façades, notamment la façade de contrôle (interfaces `RegisterFacade`,

`MonitorFacade` et `PAMFacade`), la façade filtrage (`FilterFacade`) et la façade d'acquisition (`ContextDetectionFacade`). Dans cette démarche traditionnelle, la façade de contexte est également accessible au collecticiel (à travers les interfaces `ContextFacade` et `ModelFacade`), ce qui donne aux concepteurs une gamme d'action plus importante. Ils peuvent utiliser toutes les fonctionnalités liées à la gestion du contexte (y compris la gestion du modèle) que le canevas BW-*M* propose.

8.2.1.2 Démarche Service Web

L'autre démarche d'application possible du canevas BW-*M* s'effectue à travers l'utilisation d'un service Web qui englobe le canevas. Dans cette approche, le collecticiel n'interagit pas directement avec le canevas, mais il interagit avec ses façades, qui sont accessibles par le biais du service Web. Ce service isole le canevas, qui se comporte comme un composant isolé du reste du collecticiel. La communication entre les deux (collecticiel et BW-*M*) se fait entièrement à travers les standards propres aux services Web, et donc les informations sont échangées naturellement en XML (et non plus par des appels à de méthodes Java). L'application du canevas BW-*M* devient ainsi plus simple, puisqu'elle est réduite à l'utilisation du service Web qui l'englobe.

Ce service Web BW-*M* rend accessible les façades de *contrôle*, d'*acquisition* et de *filtrage* du canevas. Leurs fonctionnalités deviennent ainsi des méthodes que le service Web implémente et rend publique à travers sa description WSDL. La Figure 86 illustre une partie de cette description, dans laquelle nous apercevons la méthode `addActiveElement` de la façade d'acquisition, la méthode `addEvent` de la façade de contrôle, et la méthode `getEvent` de la façade de filtrage. La façade de contexte est, quant à elle, complètement masquée. Elle reste interne au canevas BW-*M* et n'est pas visible à l'extérieur de celui-ci (donc, il n'y a pas de méthode dans le service Web donnant accès à ses fonctionnalités). En revanche, certaines fonctionnalités de la façade de stockage sont rendues publiques par le service Web, notamment celles qui permettent l'introduction dans la base de connaissances des nouveaux utilisateurs, des profils et même des stratifications. La base de connaissances initiale utilisée par le service Web est, par ailleurs, fournie par les concepteurs. Celle-ci contient les connaissances statiques nécessaires au démarrage du service et du système. Une fois démarré, le service Web restreint l'accès à la base de connaissances, qui ne devient accessible qu'à travers la manipulation du canevas, laquelle est, pour sa part, restreinte aux interfaces implémentées par le service.

Même si cette démarche masque certaines façades du canevas BW-*M*, les points d'entrée que nous avons vus dans la démarche traditionnelle restent les mêmes (*événements*, *contexte* et *interface*) dans la démarche par service Web. Seulement leur traitement diffère entre les démarches. Les concepteurs sont toujours obligés, dans la démarche par service Web, de définir les classes d'événements qui seront observées et le moment dans lequel leur occurrence est capturée. Cependant, au lieu d'avoir à spécialiser le modèle de contenu, les concepteurs se limitent à un appel au service Web, à travers lequel le collecticiel transmet ces informations en XML, en utilisant, pour cela, le schéma XAROM (voir Annexe III). La Figure 87 présente un exemple d'utilisation du canevas BW-*M* à travers le service Web proposé. Dans la Figure 87, la méthode `addEvent` du service Web, définie par la façade de contrôle du canevas BW-*M*, est utilisée et reçoit les informations relatives à l'occurrence d'un événement (la présence de l'utilisateur `Bernard` dans le système).

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="urn:BWMfacade/wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ns2="urn:BWMfacade/types" name="BWMfacade"
targetNamespace="urn:BWMfacade/wsdl">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="urn:BWMfacade/types" xmlns:soap11-
enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="urn:BWMfacade/types">
      <complexType name="addActiveElement">
        <sequence>
          <element name="String_1" type="string"
nillable="true"/></sequence></complexType>
      <complexType name="addActiveElementResponse">
        <sequence>
          <element name="result"
type="boolean"/></sequence></complexType>
      <complexType name="addEvent">
        <sequence>
          <element name="String_1" type="string"
nillable="true"/></sequence></complexType>
      <complexType name="addEventResponse">
        <sequence>
          <element name="result"
type="boolean"/></sequence></complexType>
      <complexType name="getEvents">
        <sequence>
          <element name="String_1" type="string"
nillable="true"/></sequence></complexType>
      <complexType name="getEventsResponse">
            <sequence>
          <element name="result" type="string"
nillable="true"/></sequence></complexType>
(...)
    </schema></types>
  <message name="BWMfacadeSEI_addActiveElement">
    <part name="parameters"
element="ns2:addActiveElement"/></message>
  <message
name="BWMfacadeSEI_addActiveElemeuse=ntResponse">
    <part name="result"
element="ns2:addActiveElementResponse"/></message>
  <message name="BWMfacadeSEI_addEvent">
    <part name="parameters"
element="ns2:addEvent"/></message>
  <message name="BWMfacadeSEI_addEventResponse">
    <part name="result"
element="ns2:addEventResponse"/></message>
  <message name="BWMfacadeSEI_getEvents">
    <part name="parameters"
element="ns2:getEvents"/></message>
  <message name="BWMfacadeSEI_getEventsResponse">
    <part name="result"
element="ns2:getEventsResponse"/></message>
(...)
```

```
(...)
  <portType name="BWMfacadeSEI">
    <operation name="addActiveElement">
      <input message="tns:BWMfacadeSEI_addActiveElement"/>
      <output
message="tns:BWMfacadeSEI_addActiveElementResponse"/></op
eration>
    <operation name="addEvent">
      <input message="tns:BWMfacadeSEI_addEvent"/>
      <output
message="tns:BWMfacadeSEI_addEventResponse"/></operation>
    <operation name="getEvents">
      <input message="tns:BWMfacadeSEI_getEvents"/>
      <output
message="tns:BWMfacadeSEI_getEventsResponse"/></operation>
(...)
  </portType>
  <binding name="BWMfacadeSEIBinding"
type="tns:BWMfacadeSEI">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <operation name="addActiveElement">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/></input>
      <output>
        <soap:body use="literal"/></output></operation>
    <operation name="addEvent">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/></input>
      <output>
        <soap:body use="literal"/></output></operation>
    <operation name="getEvents">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/></input>
      <output>
        <soap:body use="literal"/></output></operation>
(...)
  </binding>
  <service name="BWMfacade">
    <port name="BWMfacadeSEIPort"
binding="tns:BWMfacadeSEIBinding">
      <soap:address
location="http://condado.mshome.net:8080/BWM/BWMfacade"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"/></port></service>
</definitions>
```

Figure 86. Description WSDL du service Web qui englobe le canevas BW-$\mathcal{M}$.

En ce qui concerne le point d'entrée contexte, le constat est le même. De la même manière que dans la démarche précédente, l'application du canevas BW-$\mathcal{M}$ à travers la démarche par service Web nécessite que les informations sur le contexte soient transmises par le système au canevas. Le système doit toujours s'occuper du choix des éléments du contexte et de l'acquisition de ces éléments. Une fois acquises, les informations contextuelles sont transmises, en XML, au canevas, par le biais de la façade d'acquisition, qui est rendue accessible par le service Web. Ainsi, dans la Figure 88, nous pouvons apercevoir un appel à la méthode `addActiveElement` du service Web. Dans cet appel, les informations sur le contexte d'un nouvel utilisateur actif dans le système (l'utilisateur `Alain`) sont envoyées au canevas, qui les traite comme nous l'avons décrit précédemment. Ces informations sont transmises par une chaîne de caractères XML, suivant le schéma XAROM. Dans ce schéma, les objets sont décrits par leurs variables, à travers les étiquettes XML `Object` et `Variable`, tandis que les

tuples sont décrits par les rôles qui forment leurs liens (étiquettes XML `Tuple` et `Role`, mises en évidence dans la Figure 88).

```
[#|2006-06-27T19:44:14.734+0200|INFO|sun-appserver-pe8.2|javax.enterprise.system.tools.deployment|_ThreadID=13;|
DPL5306:Servlet Web Service Endpoint [BWMfacade] listening at address
[http://condado.mshome.net:8080/BWM/BWMfacade]|#]
[#|2006-06-27T19:44:14.828+0200|INFO|sun-appserver-pe8.2|javax.enterprise.system.tools.deployment|_ThreadID=13;|
DPL5110: EJBC - END of EJBC for [BWM]|#]
(...)
[#|2006-06-27T19:44:15.921+0200|INFO|sun-appserver-pe8.2|javax.enterprise.system.container.web|_ThreadID=14;|
WEB0100: Loading web module [BWM] in virtual server [server] at [/BWM]|#]
[#|2006-06-27T19:44:16.015+0200|INFO|sun-appserver-pe8.2|javax.enterprise.resource.webservices.rpc.server.http|
_ThreadID=14;|JAXRPCSERVLET14 : Initialisation du servlet JAX-RPC|#]
[#|2006-06-27T19:44:16.140+0200|INFO|sun-appserver-pe8.2|javax.enterprise.system.tools.admin|_ThreadID=14;|
ADM1042:Status of dynamic reconfiguration event processing:[success]|#]

[#|2006-06-27T19:44:34.734+0200|INFO|sun-appserver-pe8.2|javax.enterprise.system.stream.out|_ThreadID=15;|Log
message: Tue Jun 27 19:44:34 CEST 2006--addEvent String_1:<Object name="#12F4D0"> <Variable
name="identifiant">Bernard logged in</Variable> <Variable name="description">Location: bureau D322</Variable>
<Variable name="details">Logged in at 8:30; Device: IBM Thinkpad; Location: bureau D322; Location details: 3rd
floor</Variable> <Variable name="time interval">8:30--</Variable> <Variable
name="media">http://jangada/photos/Bernard.jpg</Variable> </Object> |#]

(...)
```

Figure 87. Exemple d'appel au service Web (méthode `addEvent`), extrait du journal du serveur.

```
[#|2006-06-27T19:44:14.734+0200|INFO|sun-appserver-pe8.2|javax.enterprise.system.tools.deployment|_ThreadID=13;|
DPL5306:Servlet Web Service Endpoint [BWMfacade] listening at address
[http://condado.mshome.net:8080/BWM/BWMfacade]|#]
[#|2006-06-27T19:44:14.828+0200|INFO|sun-appserver-pe8.2|javax.enterprise.system.tools.deployment|_ThreadID=13;|
DPL5110: EJBC - END of EJBC for [BWM]|#]
(...)
[#|2006-06-27T21:30:36.609+0200|INFO|sun-appserver-pe8.2|javax.enterprise.system.stream.out|_ThreadID=15;|Log
message: Tue Jun 27 21:30:36 CEST 2006--addActiveElement String_1:<Object name="treo650" isa="Device">
<Variable name="memory"> 18.0</Variable> <Variable name="energy">82.0</Variable> <Variable
name="ccppProfileURL">http://jangada/ccpp/treo650.xml </Variable> <Variable name="precision">0.6</Variable>
<Variable name="name">treo650</Variable> </Object> <Object name="contexte_Alain" isa="ContextDescription">
</Object> <Object name="Alain" isa="Member"> <Variable name="login"> duponta</Variable> <Variable
name="email">Alain.Dupont@host.fr</Variable> <Variable name="name">Alain Dupont</Variable> </Object> <Tuple
isa="description_composition"> <Role name="element">Alain</Role> <Role name="descriptor">
contexte_Alain</Role> </Tuple> <Tuple isa="description_composition"> <Role name="element">treo650</Role>
<Role name="descriptor">contexte_Alain</Role> </Tuple> <Tuple isa="contextualize"> <Role
name="descriptor">contexte_Alain </Role> <Role name="target">Alain</Role> </Tuple> |#]
(...)
```

Figure 88. Exemple d'utilisation de la méthode `addActiveElement` du canevas BW-$\mathcal{M}$ à travers le service Web (extrait du journal du serveur.)

Enfin, en ce qui concerne la présentation, celle-ci reste sous la responsabilité du collecticiel qui utilise le service Web BW-$\mathcal{M}$. Par exemple, lorsque celui-ci est sollicité pour le filtrage des informations, il renvoie au client qui l'appelle les événements sélectionnés et ordonnés, ainsi que les stratifications qui doivent être utilisées pour leur présentation. Toutes ces informations sont transmises par une chaîne de caractères XML en suivant la structure déterminée par la sous-unité de filtrage (cf. section 8.1.2.1). La Figure 89 présente un exemple de ces informations que le service Web BW-$\mathcal{M}$ fournit en retour à un appel `getEvents`. Il s'agit donc d'un texte XML pur, dont la présentation à l'utilisateur doit être faite par le collecticiel qui le reçoit. Ceci est, par ailleurs, le procédé normal pour tout service Web : l'application client est en charge de l'interface avec l'utilisateur.

L'avantage de l'utilisation de la démarche par service Web par rapport à la démarche traditionnelle réside dans la non utilisation des paquetages Java qui composent le canevas

BW-$\mathcal{M}$. À travers l'utilisation du service Web BW-$\mathcal{M}$, les concepteurs ne sont pas tenus d'utiliser directement ces paquetages dans le système en conception. Les concepteurs peuvent même utiliser un langage autre que le Java pour la construction du système, et utiliser les fonctionnalités du canevas BW-$\mathcal{M}$ à travers son service Web.

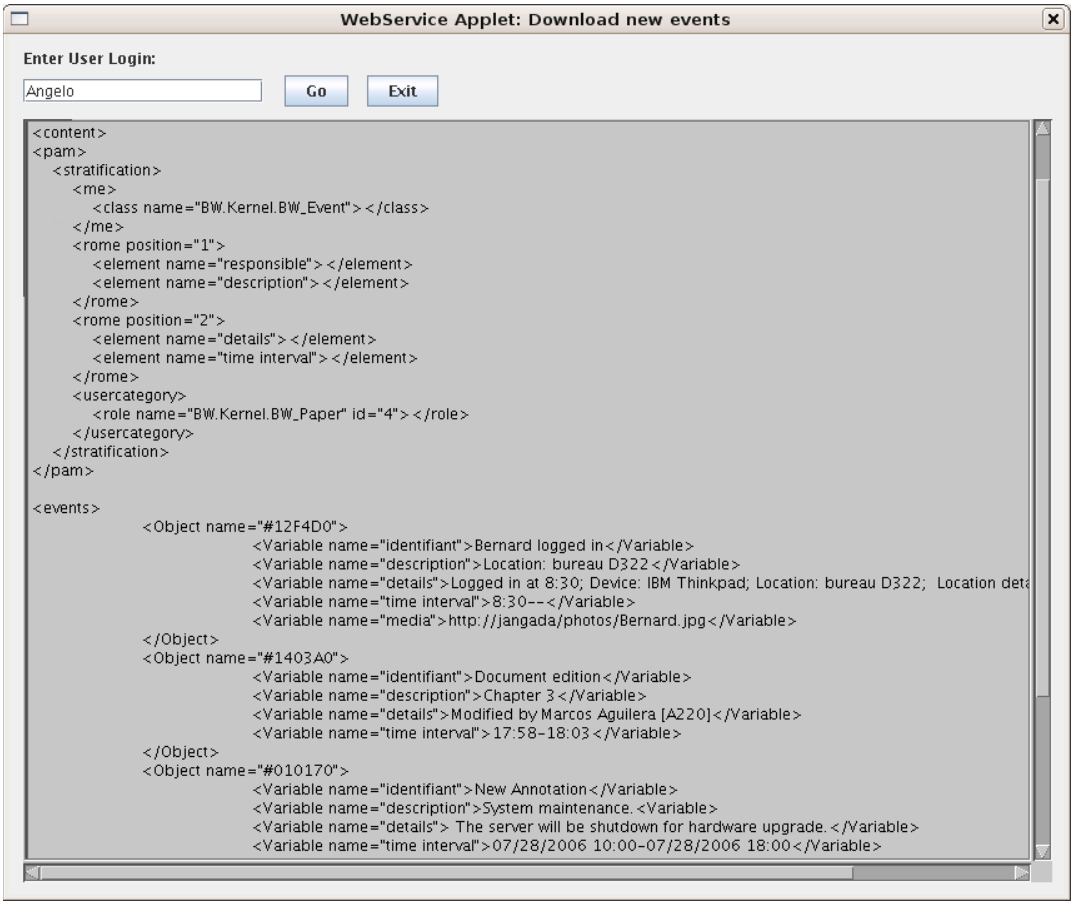


Figure 89. Exemple de retour de l'appel `getEvents` au service Web BW-$\mathcal{M}$.

8.2.2 Illustrations

Dans cette section, nous illustrons l'application du canevas BW-$\mathcal{M}$ par quelques exemples de la façon dont le canevas serait utilisé dans de systèmes existant dans la littérature. Nous illustrons donc l'application du canevas par sa possible intégration à deux systèmes distincts : l'éditeur collaboratif *AllianceWeb* et l'espace de travail partagé *LibreSource.*

8.2.2.1 AllianceWeb

AllianceWeb [Martínez 2002b] [Salcedo 1998] est un éditeur collaboratif sur la Web qui permet à un groupe d'auteurs d'éditer de manière asynchrone de documents structurés. *AllianceWeb* est considéré comme un collecticiel asynchrone, puisque les utilisateurs ne sont pas obligés de travailler de manière toujours synchrone. D'ailleurs, cet éditeur permet que plusieurs auteurs puissent avoir le droit de changer un même document dans un même

intervalle de temps. Le document est divisé automatiquement en plusieurs fragments, et chaque auteur modifie un fragment distinct. Le support à la conscience du groupe dans l'éditeur *AllianceWeb* est réalisé par la notification des coauteurs sur les changements observés dans les fragments qui composent le document en cours d'édition pendant leur période d'activité. Accessoirement, cet éditeur permet également l'affichage des informations sur les coauteurs. Finalement, le support à la conscience de groupe permet à chaque utilisateur de configurer le niveau de notification souhaité (mise à jour automatique, notification des changements ou mode sans surveillance) pour chaque fragment auquel l'utilisateur accède [Martínez 2002b] [Salcedo 1998].

Comme tout autre collecticiel sur le Web, *AllianceWeb* est maintenant accessible à partir des dispositifs mobiles capables d'accéder au Web, notamment les ordinateurs de poche et les ordinateurs portables. L'intérêt d'un tel système pour un utilisateur nomade réside non seulement sur la possibilité d'éditer collaborativement un document n'importe où, dès qu'un accès réseaux est disponible (ce qui est particulièrement intéressant pour les utilisateurs munis d'ordinateurs portables), mais également sur la possibilité de consulter les annotations faites sur un document (*AllianceWeb* propose un outil d'annotation) et les modifications effectuées par les coauteurs. Dans ces situations le support à la conscience de groupe de *AllianceWeb* ne semble pas adapté aux utilisateurs nomades, étant donné que l'information présentée est peu détaillée et limitée aux modifications sur le document en édition sur le moment. Par exemple, les utilisateurs disposent d'une indication sur le fait qu'une nouvelle version d'un fragment est disponible, mais ils sont incapables de savoir combien de modifications ont été réalisées depuis leur dernière connexion au système. Par conséquent, si un utilisateur désire savoir si un collègue a modifié un document ou a fait des annotations dans d'autres fragments que ceux visualisés par l'utilisateur, celui-ci doit consulter la totalité du document. Un tel procédé peut s'avérer inadapté à un utilisateur nomade, muni d'un dispositif mobile, étant donné les contraintes qui habituellement accablent ce type d'utilisateur.

Ainsi, nous pouvons imaginer la proposition d'un nouveau service de conscience de groupe pour l'éditeur *AllianceWeb* à partir de l'application du canevas BW-*M*. Dans ce cas spécifique, le canevas BW-*M* interagit avec l'intergiciel *Piñas*, sur lequel l'éditeur *AllianceWeb* est basé. L'intergiciel *Piñas* est utilisé par l'éditeur *AllianceWeb* notamment pour la gestion des documents partagés et des messages du système [Martínez 2002b] [Decouchant 2001]. Cependant, *Piñas* et *AllianceWeb* ont été écrits en langage C, ce qui nous dirige vers la démarche par service Web pour l'application du canevas BW-*M*. Nous avons donc besoin d'un nouveau composant lié à l'intergiciel *Piñas*, qui serait responsable de la capture des informations transmises par *Piñas* entre les instances de l'éditeur *AllianceWeb*. Ce composant traite ainsi les informations qu'il capture et les envoie au service Web BW-*M*. Il agit donc comme un *médiateur* entre l'éditeur *AllianceWeb* et le service Web.

Suivant ainsi la démarche d'application du canevas BW-*M*, il faut satisfaire les trois « points d'entrée » du canevas (les événements, le contexte, et l'interface). Le premier de ces points fait référence aux événements. Il faut donc définir les classes d'événements qui seront observées et comment cette observation aura lieu. Nous proposons alors, pour l'éditeur *AllianceWeb*, l'enregistrement de cinq classes d'*événement*, dont trois liées aux fragments qui composent les documents, une relative aux annotations et une relative aux utilisateurs : (*i*) une classe pour indiquer la disponibilité d'une nouvelle version d'un fragment (« `ChangedFragment` ») ; (*ii*) une deuxième classe pour indiquer l'union de deux fragments (« `MergedFragments` ») ; (*iii*) une troisième classe d'événement pour indiquer la division d'un fragment en deux (« `SplitFragments` ») ; (*v*) une classe d'événement appelée « `NewAnnotation` » pour indiquer qu'une nouvelle annotation a été attachée à un document ; et (*iv*) une classe d'événement « `UserSession` » pour indiquer la présence d'un coauteur dans

l'environnement de travail. L'occurrence de ces événements est observée par le médiateur qui reconnaît leur occurrence à partir des messages qui sont transmises par l'intergiciel *Piñas*. Le travail du médiateur consiste ici à traduire les messages qu'il capture vers des événements, qui sont représentés en XML et qui sont ensuite envoyés au service Web.

Le service Web BW-$\mathcal{M}$ reçoit et traite l'occurrence de ces événements comme nous l'avons décrit précédemment. Ces événements font désormais partie de la base de connaissances du service BW-$\mathcal{M}$. Par conséquent, les utilisateurs d'*AllianceWeb* auront désormais à leur disposition des informations relatives à l'évolution du processus d'édition sur le document, puisque même les événements qui se sont produits lorsqu'un utilisateur était absent seront disponibles pour lui à son retour. Par exemple si un utilisateur a été absent pendant une certaine période, il pourra recevoir les événements qui décrivent les modifications effectuées sur le document partagé durant cette période, pouvant ainsi suivre l'évolution du processus d'édition du document.

Le second point d'entrée considère le contexte. Nous devons introduire dans l'éditeur *AllianceWeb* quelques fonctions d'acquisition de contexte, de manière à lui permettre d'exploiter toutes les fonctionnalités du canevas BW-$\mathcal{M}$. Ceci commence par la détermination des classes d'`Élément_de_Contexte` qui seront observées. Nous préconisons, pour l'éditeur *AllianceWeb*, l'observation d'au moins les éléments de contexte suivants : les concepts de `Membre`, `Groupe`, `Rôle` et `Activité`, proposés par la notion de contexte collaboratif (cf. chapitre 5), et le concept de `Dispositif` employé par les utilisateurs. La majorité de ces éléments peuvent être capturés facilement par un composant d'acquisition de contexte situé auprès de chaque instance de l'éditeur *AllianceWeb* et communicant avec l'intergiciel *Piñas*. Par ailleurs, l'intergiciel sert également de moyen de communication entre ce composant d'acquisition de contexte et le médiateur.

Pour le troisième point d'entrée, l'interface, nous proposons le développement de nouvelles interfaces adaptées aux dispositifs de taille réduite comme les ordinateurs de poche. Une telle interface pourrait permettre à l'utilisateur d'ouvrir un document pour l'édition ou de consulter les informations de conscience de groupe sans l'avoir ouvrir. Cette dernière interface devrait être simple, comme, par exemple, une interface textuelle qui présente la liste et le contenu des instances d'événements, selon l'organisation établie par les stratifications sélectionnées par le processus de filtrage.

Profil	Contexte d'application	Classes d'événements abonnées	Stratifications
Profile1	{laptop: *Device*}	*ChangedFragment* *MergedFragment* *SplitFragment*	S_1^{ext} = { {event.interval **during** DAY}, {event.interval **during** WEEK } } S_2^{int}= { {name, description}, {interval, details} {medias} }
Profile2	{manager: *Role*}	*NewAnnotation* *UserSession*	S_3^{ext} = { {event.interval **during** WEEK}, {event.interval **during** (WEEK - 1) } } S_2^{int} = { {name, description}, {interval, details} {medias} }

Tableau 2. Profils par défaut prévus pour l'éditeur *AllianceWeb*.

D'un autre côté, l'utilisation du canevas BW-$\mathcal{M}$ requiert aussi l'implémentation de quelques profils généraux et d'une interface qui permet aux utilisateurs de définir leurs propres profils. Nous proposons le développement de profils pré-définis pour les situations les

plus courantes, de manière à fournir aux utilisateurs des exemples qui pourront être utilisés pour construire leurs propres profils. Ainsi, deux situations sont particulièrement intéressantes pour la définition des profils par défaut : lorsque l'utilisateur utilise un ordinateur portable, et lorsque l'utilisateur joue le rôle de « *manager* » pour le document courant. La première situation est fortement caractérisée par le travail en mode déconnecté, car les utilisateurs munis d'ordinateurs portables alternent souvent entre périodes d'absence et périodes de connexion. Ces utilisateurs ont besoin d'informations sur les modifications effectuées sur les fragments d'un document pendant les périodes d'absence de l'utilisateur. Dans la seconde situation, l'utilisateur joue un rôle important dans le groupe, qui peut être associé au rôle de coordinateur. Par conséquent, il est important que cet utilisateur soit tenu au courant de la présence des autres coauteurs et des discussions (à travers l'échange des annotations) entre ces coauteurs. Le Tableau 2 présente les profils pré-définis pour ces situations.

8.2.2.2 LibreSource

LibreSource [Forest 2005a] [Forest 2005b] est un collecticiel de partage de documents sur le Web. Son objectif est double : d'abord supporter le développement coopératif de logiciels (à l'instar de *SourceForge*[96]), et puis se présenter comme une plate-forme pour la gestion de communautés distribuées (à l'exemple de *Zope*[97]). *LibreSource* propose ainsi plusieurs services qui varient de l'*édition collaborative*, à travers le partage de documents et un outil de *Wiki*[98], au *contrôle de versions*, en passant par la *communication asynchrone*, à travers un forum et de listes de discussions. Tous ces services reposent sur une plate-forme fondée sur les technologies Web et sur J2EE[99], à travers l'implémentation *JOnAS*[100].

La plate-forme *LibreSource* est formée par un ensemble de composants J2EE qui s'articulent sur un noyau (Figure 90). Le noyau gère les droits d'accès et la communication entre les composants. Cette communication se fait essentiellement par des messages asynchrones, à travers la technologie JMS[101]. Les composants sont des services Web ou des *servlets*, qui s'exécutent sur le serveur d'application *JOnAS*. Ainsi, *LibreSource* est accessible à travers un client Web traditionnel (c'est-à-dire, de pages Web générées par le serveur et visualisées sur un navigateur Web standard), comme le montre la Figure 91. Par ailleurs, certains services associés à la plate-forme *LibreSource* disposent de clients légers, conçus en Java et téléchargés automatiquement à travers le navigateur (des applications *Java Web Start*[102]).

En ce qui concerne la conscience de groupe, la plate-forme *LibreSource* se base sur la notion d'événement et sur la gestion d'historique (en ce qui concerne le contrôle de versions). Les événements sont produits au cours des interactions entre les utilisateurs et le système, et plus particulièrement, entre les membres d'un projet et les ressources appartenant au projet. La notification se fait notamment à travers la page principale du projet, que les coordinateurs du projet (ou les coauteurs autorisés) peuvent configurer pour que certaines classes d'événements soient présentées. Il s'agit d'une souscription à ces classes d'événements faite au nom de tout le groupe d'utilisateurs qui participe à un projet. Par exemple, dans la Figure 91 nous pouvons

[96] http://sourceforge.net/.
[97] http://www.zope.org/.
[98] Un *wiki* est définie comme « un système de gestion de contenu de site Web qui rend les pages Web librement et également modifiables par tous les visiteurs autorisés. Les wikis sont utilisés pour faciliter l'écriture collaborative de documents avec un minimum de contrainte » (http://fr.wikipedia.org/wiki/Wiki).
[99] « Java Enterprise Edition » - http://java.sun.com/javaee/.
[100] http://jonas.objectweb.org/.
[101] « Java Message Service » - http://java.sun.com/products/jms/.
[102] http://java.sun.com/products/javawebstart/

remarquer la présence d'un tableau contenant les derniers événements produits liés au projet en question.

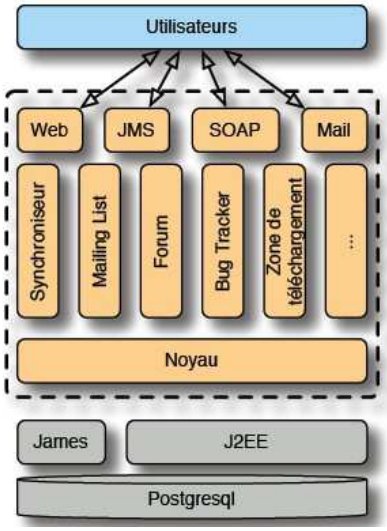


Figure 90. Structure de la plate-forme *LibreSource* (d'après [Forest 2005b]).

LibreSource, ainsi comme *AllianceWeb*, apporte un réel intérêt aux utilisateurs nomades, puisqu'ils peuvent désormais continuer à accéder au système même en déplacement et à partir de différents dispositifs. Ils peuvent ainsi suivre l'évolution du travail dans l'équipe, même lorsqu'ils sont loin de leur lieu de travail. Cependant, pour ces utilisateurs nomades, souvent confrontés à plusieurs contraintes (cf. chapitre 3), la plate-forme *LibreSource* ne fournit pas un mécanisme de conscience de groupe qui soit personnalisé et sensible au contexte. L'insertion du canevas BW-$\mathcal{M}$ dans la plate-forme devient ainsi intéressante.

Étant donnée la structure de la plate-forme *LibreSource* (voir Figure 90), l'application du canevas BW-$\mathcal{M}$ s'oriente clairement vers la démarche par service Web, puisque elle est facilement intégrable à la plate-forme. Néanmoins, ceci n'élimine pas la nécessité d'avoir un *médiateur* qui puisse observer la plate-forme et ainsi capturer le contexte et les événements. La démarche d'application suit donc le même schéma du cas précédent : d'abord définir les événements, puis traiter les questions liées au contexte, et ensuite celles liées à l'interface.

Concernant les événements, le premier « point d'entrée » du canevas BW-$\mathcal{M}$, nous préconisons l'utilisation des mêmes classes d'événements déjà proposée par *LibreSource*. Ces classes concernent des actions telles que l'introduction d'un nouveau document, ou encore la disponibilité d'une nouvelle version d'un logiciel en développement (ces deux classes d'événements, nommées `Upload` et `Commit`, sont visibles dans la Figure 91). Leur occurrence est constatée par le médiateur à travers l'observation des messages échangés entre les composants et le noyau. Une fois capturée l'occurrence d'un nouveau événement, celui-ci est en XML et envoyé au service Web BW-$\mathcal{M}$ qui le stocke dans sa base de connaissances.

En ce qui concerne le second point d'entrée, le contexte, il est nécessaire d'introduire dans la plate-forme *LibreSource* soit un nouveau composant pour l'acquisition de contexte, soit de le faire à travers le médiateur. Les éléments de contexte qui peuvent être facilement observés restent assez similaires au cas précédent : les éléments lié au contexte collaboratif (`Membre`, `Groupe`, `Rôle`, `Activité` et `Objet_Partagé`), plus les éléments `Dispositif` et `Application`. L'acquisition de ces éléments de contexte est réalisée également à travers l'observation des messages échangés entre les composants appartenant à la plate-forme. Néanmoins, une seconde possibilité serait de munir le composant d'acquisition de contexte

d'un client léger que les utilisateurs pourraient télécharger à travers *LibreSource* (cette pratique est déjà largement répandue dans *LibreSource*, qui l'utilise notamment pour le contrôle de versions). Cette approche aurait l'avantage d'être visible à l'utilisateur, qui, en conséquence, devient conscient du processus d'acquisition de contexte pour lequel il devra donner son accord. Ainsi, si un utilisateur n'est pas d'accord avec l'observation de son contexte courant par le système, il peut éviter ceci tout simplement en refusant le téléchargement de l'application qui effectue l'acquisition de contexte localement.

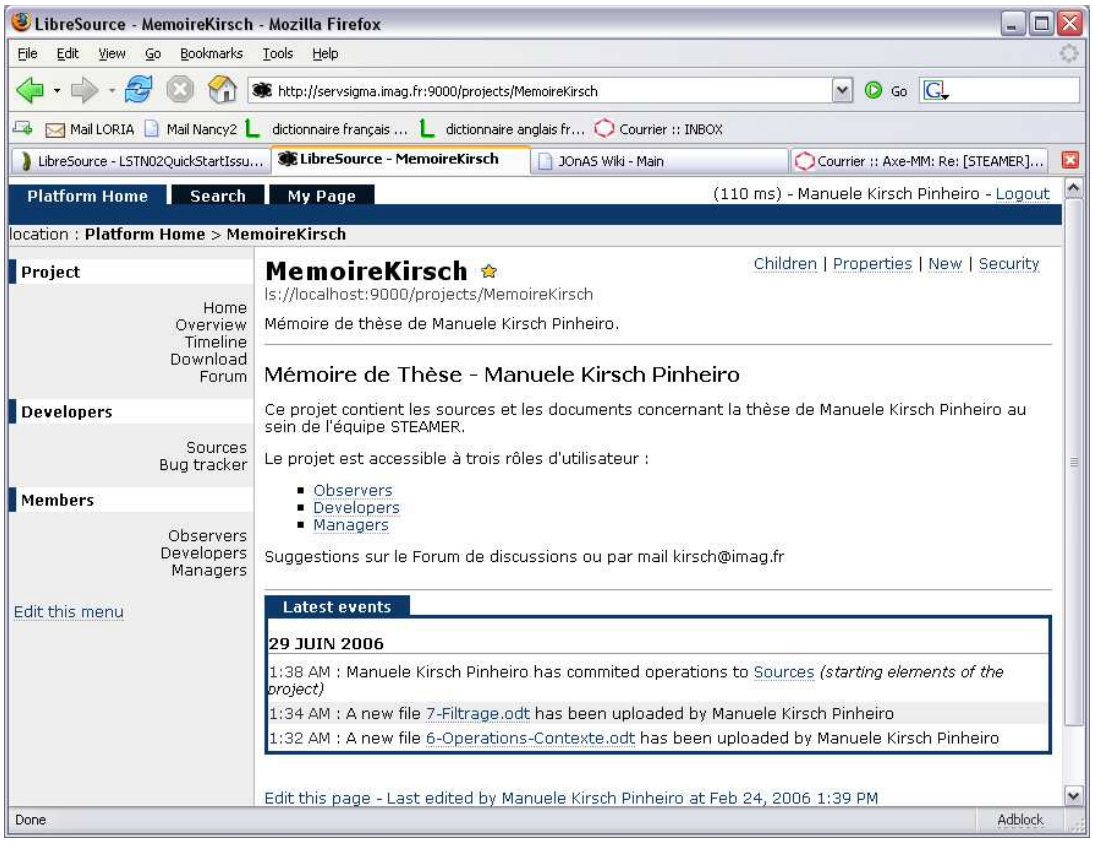


Figure 91. Page d'entrée d'un projet sur *LibreSource*, dans laquelle on retrouve la liste de derniers événements produits.

Cette approche par client léger qui est téléchargé par l'utilisateur peut également être utilisée pour le troisième point d'entrée du canevas, l'interface. Nous pouvons donc définir une application Java qui communique directement avec le service Web. Une telle application fournit non seulement une interface pour la présentation de l'information de conscience de groupe suivant les définitions du modèle d'accès progressif, mais également fournir une interface pour la définition des nouveaux profils pour l'utilisateur.

Par ailleurs, pour chaque projet, nous pouvons associer un profil par défaut pour les utilisateurs d'ordinateurs de poche, lequel présenterait un résumé de tous les modifications apportées, soit par la mise à jour de codes sources développés par le groupe (événements `Commit`), soit par l'introduction de nouveaux documents (événements `Upload`). De même pour le rôle de coordinateur du projet (*manager*), lequel intéresse également les nouvelles contributions déposées sur le forum de discussions. Le premier cas consisterait en un profil qui présente un résumé de la situation actuelle du groupe (idéale donc pour les utilisateurs qui ne disposent pas suffisamment de temps et de ressources pour analyser tous événements liés

au projet). Le second donne, au contraire, une vue plus générale de la coopération, nécessaire au coordinateur du groupe. Le Tableau 3 illustre ces deux exemples de profils.

Profil	Contexte d'application	Classes d'événements abonnées	Conditions contextuelles	Stratifications
Profile-Proj1	{laptop: *Device*}	*update* *commit*	EC[103]= {*project1:* Group}	S_1^{ext} = { {event.interval **during** DAY}, {event.interval **during** (DAY- 1) } } S_2^{int}= { {name, interval }, { details } }
Profile-manage	{manager: *Role*}	*update* *commit* *user session* *new post*	EC = {*project1:* Group}	S_3^{ext} = { {event.interval **during** DAY}, {event.interval **during** WEEK}, {event.interval **during** (WEEK - 1) } } S_4^{int} = { {name, description}, {interval, details} {medias} }

Tableau 3. Exemples de profils par défaut possibles pour la plate-forme *LibreSource*.

8.3 Conclusions

Dans ce chapitre[104], nous avons proposé une implémentation pour le processus de filtrage que nous avons discuté dans le chapitre 7. Cette implémentation se présente sous la forme d'un canevas en langage Java, nommé BW-$\mathcal{M}$, dont l'objectif est de fournir un support sensible au contexte à l'information de conscience de groupe.

Nous avons proposé deux démarches d'application pour le canevas BW-$\mathcal{M}$. D'abord, une démarche traditionnelle, laquelle propose l'utilisation directe des paquetages Java qui composent le canevas dans le système auquel il est appliqué. Ensuite, une démarche qui propose l'utilisation d'un service Web qui encapsule le canevas. Dans ces deux démarches, l'application du canevas doit satisfaire trois points : la définition des événements, l'acquisition de contexte, et l'interface avec l'utilisateur. Les illustrations présentées ci-dessus (section 8.2.2) montrent notamment la démarche par service Web, qui semble plus simple que la démarche traditionnelle, même si cette dernière garantit un accès complet aux fonctionnalités définies par les façades du canevas.

Pour conclure, il est important de remarquer que l'objectif du canevas BW-$\mathcal{M}$ est de démontrer l'applicabilité du processus de filtrage et du modèle de contexte que nous avons proposé dans ce travail. À travers ce canevas, nous montrons que tous les modèles proposés dans ce travail peuvent être implémentés et mis à contribution à travers un composant logiciel tel qu'un canevas ou un service Web. Ce dernier est particulièrement intéressant puisque la nouvelle génération de collecticiels sur le Web sera probablement conçue en utilisant cette technologie. Par ailleurs, cette nouvelle génération doit être conçue non seulement en tant que collecticiel, mais également en tant que système sensible au contexte, étant donnée la disponibilité grandissante des technologies mobiles donnant accès au Web.

[103]EC = éléments concernés.

[104]Les sujets abordés dans ce chapitre ont été traités par les publications [1], [3] et [4] indiquées dans l'Annexe IV.

9 Conclusions et Perspectives

Le présent travail se positionne à l'interface entre deux thématiques de recherche : le *Travail Coopératif Assisté par Ordinateur* (cf. chapitre 2) et l'*Informatique Sensible au Contexte* (cf. chapitre 3). Dans ces thématiques, nous nous sommes concentrés sur la question du support à la conscience de groupe pour des utilisateurs nomades, dans un système sur le Web à la fois coopératif et sensible au contexte. Nous faisons donc face à la nouvelle génération de collecticiels sur le Web qui doivent désormais être conçus également en tant que systèmes sensibles au contexte. Dans ce chapitre, nous présentons nos conclusions et nos perspectives pour le travail présenté dans ce document.

9.1 Rappel de la Problématique

La problématique à l'origine de ce travail de thèse part d'un constat simple : l'utilisation des nouvelles technologies mobiles et la mobilité des utilisateurs qui en découle doivent être prises en compte lors de la conception des nouveaux collecticiels sur le Web. L'évolution et la démocratisation des nouvelles technologies mobiles représentées, entre autres, par les ordinateurs de poche, les téléphones cellulaires et par les réseaux *WiFi*, a donné plus de liberté aux utilisateurs qui peuvent désormais se déplacer librement tout en accédant au Web dès lors qu'une connexion réseau est disponible. Ces technologies donnent à l'utilisateur une mobilité pour l'accès à l'information dont il ne disposait pas auparavant. Cette mobilité représente un atout important pour le travail coopératif, qui n'est plus cantonné au bureau. Par conséquent, les collecticiels sur le Web, qui sont dès maintenant accessibles à travers les nouvelles technologies, doivent tenir compte des besoins de ces nouveaux utilisateurs qu'on dit nomades.

À partir de ce constat, nous avons focalisé notre travail sur quatre problèmes de base :

(1) Les collecticiels, doivent désormais présenter des capacités d'adaptation au contexte.

Les nouvelles technologies mobiles présentent des capacités limitées par rapport aux technologies traditionnelles : capacités d'affichage et de mémoire réduites, processeurs moins performants, durée de vie restreinte des batteries, débit plus faible des réseaux sans fil, problèmes de déconnexions, etc. De plus, ces contraintes varient dans le temps avec les différentes situations dans lesquelles l'utilisateur peut se trouver au moment de son accès au système. Par conséquent, il devient crucial de doter les systèmes sur le Web de capacités d'adapter leur offre, en termes de diffusion d'information (par le choix du contenu et de la présentation) et de mise à disposition de services, à ces utilisateurs nomades en tenant compte du contexte d'utilisation courant.

(2) Les contraintes auxquelles sont confrontés les utilisateurs nomades aggravent les risques de surcharge cognitive.

Les utilisateurs nomades ont besoin d'informations de *conscience de groupe*, même (et peut-être principalement) lorsqu'ils ne sont pas dans leurs bureaux respectifs. Le nomadisme favorise la perte de contact entre les participants du groupe, et les collecticiels doivent compenser cette possible perte par des mécanismes de conscience de groupe adaptés.

Cependant, l'utilisation des nouvelles technologies rend difficile l'application des techniques traditionnellement utilisées pour le support à la conscience de groupe. Les mécanismes de conscience de groupe ne peuvent plus exploiter les techniques de visualisation très élaborées, ni envoyer des quantités importantes d'information par le réseau. De plus, un utilisateur nomade ne dispose pas du même temps pour interpréter les informations de conscience de groupe, et il ne s'intéresse pas forcément aux mêmes informations que lorsqu'il était dans un environnement fixe (dans son bureau, par exemple). Ses intérêts peuvent changer en fonction de son activité, de sa localisation, et même des ressources matérielles dont il dispose. Les mécanismes de conscience de groupe, au sein des collecticiels sur le Web, doivent donc adapter leur offre aux utilisateurs nomades dans leur contexte d'utilisation courant. Ils doivent devenir sensibles au contexte.

(3) La notion de contexte d'utilisation, à la base des systèmes sensibles au contexte, doit être élargie pour les collecticiels, afin de prendre en considération les aspects collaboratifs naturels à ces systèmes.

Un collecticiel qui se veut sensible au contexte doit, au contraire d'autres systèmes sensibles au contexte, considérer l'utilisateur nomade en tant que participant d'un ou plusieurs groupes de travail, dans lesquels il collabore avec d'autres membres. Par conséquent, outre les éléments qui décrivent la situation physique de l'utilisateur en tant qu'individu (tels que sa localisation et son dispositif actuel), plusieurs éléments relatifs au groupe et au processus coopératif doivent également faire partie du contexte de cet utilisateur nomade. Il faut donc élargir la notion de *contexte d'utilisation* pour prendre en compte, dans un processus d'adaptation, tout ce qui relève de l'information relative au processus collaboratif dans lequel les utilisateurs sont impliqués.

(4) Les préférences de l'utilisateur varient en fonction du contexte dans lequel il se trouve et elles ne sont pas suffisamment prises en considération par les systèmes sensibles au contexte.

En ce qui concerne l'information de conscience de groupe, il est important d'impliquer davantage l'utilisateur dans le processus d'adaptation. La majorité des systèmes sensibles au contexte ne tiennent que très peu compte des préférences de l'utilisateur pour l'adaptation, tandis que les collecticiels prennent habituellement en compte les préférences de l'utilisateur sans les mettre en relation avec le contexte d'utilisation. Or, un utilisateur nomade peut se trouver dans des contextes variés, aux contraintes également distinctes, autant du point de vue technique (utilisation de dispositifs aux capacités distinctes, par exemple), que du point de vue social (on n'agit pas de la même façon au bureau, à la maison, ou dans un train, etc.). Les préférences d'un utilisateur par rapport à l'information de conscience de groupe peuvent donc changer en fonction de l'environnement dans lequel il se trouve. Par ailleurs, les utilisateurs nomades ne s'intéressent qu'aux informations de conscience de groupe qui peuvent contribuer à leurs activités courantes. Ces utilisateurs s'attendent donc à ce que seules les informations de conscience de groupe les plus pertinentes leur soient fournies. De plus, l'analyse de ces informations ne doit pas exiger plus d'attention ou de temps que ce dont dispose l'utilisateur. Il faut donc que les informations les plus pertinentes pour le contexte courant de l'utilisateur soient rendues disponibles avant les moins pertinentes.

9.2 Bilan du Travail Réalisé

À partir de la problématique exposée ci-dessus, nous avons développé, dans cette thèse, quatre axes de travail interconnectés :

(1) la proposition d'un modèle de représentation par objets du contexte capable de formaliser la notion de contexte élargie pour les collecticiels ;

(2) la proposition d'un ensemble d'opérations capables de manipuler les instances de ce modèle ;

(3) la définition d'un processus de filtrage des informations de conscience de groupe capable de sélectionner et d'organiser ces informations selon le contexte d'utilisation et les préférences de l'utilisateur ;

(4) la construction d'un canevas pour la mise en œuvre des propositions ci-dessus.

L'approche adoptée par ce travail a été donc celle de l'adaptation de l'information de conscience de groupe par le filtrage de ces informations. Ce filtrage est guidé à la fois par le contexte d'utilisation et par les préférences de l'utilisateur. Cette approche vise le groupe à travers les individus qui le composent. Nous croyons que, pour améliorer la performance du groupe, il faut d'abord améliorer les conditions de travail de chaque membre du groupe. Les performances d'un individu dans le groupe ont une répercussion dans la performance du groupe dans l'ensemble, puisque les difficultés d'un membre peuvent être à l'origine de ruptures dans la réalisation d'activités et dans la circulation d'informations qui pénaliseront nécessairement le travail d'autres membres. Cependant, le fait de considérer chaque membre du groupe individuellement ne signifie pas, pour autant, que nous avons traité les individus de manière isolée. Nous les avons considéré comme partie d'un collectif, le groupe, à travers notamment la représentation de contexte que nous avons proposée. Celle-ci tient compte à la fois des aspects physiques du contexte (la localisation de l'utilisateur, par exemple), et des aspects directement liés à la coopération et au groupe (tel que les rôles joués par l'utilisateur).

Nos contributions dans chaque axe de travail sont discutées ci-dessous :

(1) Le modèle de contexte

Dans cet axe de travail, nous avons proposé un modèle par objets simple et extensible qui représente la notion de contexte. Ce modèle se positionne à un niveau conceptuel, qui doit être spécialisé afin d'être correctement appliqué à un système précis. Cette spécialisation réside notamment dans le choix des éléments de contexte qui composent une description de contexte.

À travers ce modèle, nous avons pu représenter, par un formalisme à objets, une notion de contexte élargie (par rapport à celle que les systèmes sensibles au contexte utilisent), qui tient compte des aspects collaboratifs propres aux collecticiels. Nous avons considéré, comme étant partie de la notion de contexte, certains éléments liés au processus de coopération qui relie les membres d'une équipe, en plus des éléments liés au contexte physique d'utilisation du système. Une telle notion de contexte s'applique particulièrement aux collecticiels : les utilisateurs de ce type de système sont en effet engagés dans un processus de coopération qui influence, en retour, le contexte d'utilisation du collecticiel. Cette prise en compte des aspects collaboratifs démarque le modèle de contexte que nous avons proposé des autres représentations de contexte proposées dans la littérature.

Une autre particularité de notre modèle de contexte est la possibilité de représenter une information contextuelle incomplète ou même ambiguë. Ceci est un atout important pour le modèle de contexte, puisque le processus d'acquisition de contexte qui alimente le modèle n'est pas infaillible et peut être à l'origine d'informations incomplètes (par l'incapacité de capter certaines informations) ou ambiguës (suite, par exemple, à un problème d'interprétation).

Par ailleurs, une description de contexte est pour nous composée par un ensemble d'éléments de contexte *et* par un ensemble de relations que relient entre eux ces éléments. Ceci permet une description plus complète de la situation dans laquelle un utilisateur peut se trouver. Enfin, à travers ce modèle, il est possible de représenter le contexte courant d'un utilisateur nomade, ainsi que la représentation des contextes potentiels dans lesquels cet utilisateur peut se trouver pendant les interactions avec le collecticiel.

(2) Opérations sur le modèle

Grâce à l'utilisation d'un formalisme par objets, le modèle de contexte est naturellement sujet aux mécanismes d'exploitation propres à ce formalisme, tels que l'instanciation, l'héritage et la classification. Cependant, nous avons également défini une série d'opérateurs qui permettent de comparer les instances de ce modèle, et ainsi d'exploiter de différentes manières le contenu de ces instances. Nous avons défini trois types d'opérations complémentaires (*equals*, *contains* et *simO*, *simT*, *Sim*). Chaque type d'opération analyse un type différent de relation (relation d'égalité, relation d'inclusion, relation de similarité, respectivement). Ces opérations permettent également la comparaison entre les instances de ce modèle tant de manière isolée, qu'en tenant compte de leurs relations avec d'autres instances, comme dans un graphe. Il s'agit donc d'un mécanisme d'exploitation complémentaire aux mécanismes traditionnellement associés au formalisme par objets.

Par ailleurs, il est important d'observer que, même si originellement ces opérations ont été définies pour le modèle de contexte, elles ont été définies de manière générique, en ce basant sur les concepts utilisés par le Système de Représentation de Connaissances par Objets (SRCO) AROM. Ceci permet l'application de ces opérations à tout autre modèle objet.

(3) Processus de filtrage guidé par le contexte

En nous basant sur le modèle de contexte et les opérations définies sur ce modèle, nous avons proposé un processus de filtrage guidé par le contexte de l'information de conscience de groupe. Ce processus est guidé à la fois par le contexte d'utilisation courant et par les préférences de l'utilisateur définies pour ce contexte, ce qui le démarque des autres propositions dans les thématiques étudiées par ce travail.

Le processus de filtrage que nous avons proposé s'appuie sur un ensemble de modèles interconnectés qui représentent l'information de conscience de groupe (modèle de contenu), l'organisation de cette information en plusieurs niveaux de détails (modèle d'accès progressif), et les préférences de l'utilisateur pour un contexte donné (modèle de profil). À travers le modèle de profil, les utilisateurs peuvent exprimer quelles informations de conscience de groupe ils considèrent comme pertinentes dans un contexte potentiel donné, et dans quel ordre ces informations doivent être mises à leur disposition. Cette association entre les préférences de l'utilisateur et les contextes potentiels dans lesquels elles s'appliquent permet que l'utilisateur soit impliqué davantage dans le processus de filtrage. Les utilisateurs peuvent désormais décrire leurs préférences pour les situations qu'ils rencontrent le plus souvent, ce

qui peut rendre l'adaptation de l'information de conscience de groupe plus appropriée à la fois au contexte d'utilisation et aux attentes de l'utilisateur.

(4) Le canevas BW-$\mathcal{M}$

Le dernier axe de notre travail est représenté par la proposition du canevas BW-$\mathcal{M}$. Nous avons proposé le canevas BW-$\mathcal{M}$ en tant que mise en œuvre des propositions précédentes. En d'autres termes, le canevas BW-$\mathcal{M}$ implémente le processus de filtrage, les opérations sur le modèle de contexte et le modèle de contexte, pour lequel ce canevas utilise une base de connaissances dans le système AROM. Le canevas BW-$\mathcal{M}$ est réalisé en langage Java et peut être utilisé de deux formes distinctes : directement à travers la manipulation des paquetages Java qui le composent (ce qui correspond à la démarche traditionnelle d'utilisation d'un canevas en langage Java), et à travers un service Web que nous avons proposé et qui encapsule le canevas. Le canevas BW-$\mathcal{M}$ peut donc être utilisé pour la conception des nouveaux collecticiels sur le Web sensibles au contexte à travers les deux démarches que nous avons mises en place.

9.3 Perspectives

Nos propositions ouvrent plusieurs perspectives scientifiques à court et à plus long termes. Nous soulignons ici certaines de ces perspectives qui, pour nous, présentent un intérêt plus marqué pour l'évolution des propositions réalisées. Nous organisons ces perspectives en quatre catégories, présentées ci-dessous :

(1) Évaluation approfondie du canevas BW-$\mathcal{M}$

À court terme, une de nos perspectives consiste à réaliser des tests plus approfondis sur le canevas BW-$\mathcal{M}$, visant deux publics distincts : les concepteurs de systèmes sensibles au contexte, et les utilisateurs de ces systèmes. Nous envisageons donc d'évaluer d'abord la facilité avec laquelle différents concepteurs utilisent le canevas BW-$\mathcal{M}$. Ceci pourra nous conduire à améliorer les deux démarches d'application du canevas. L'idée est d'observer les difficultés rencontrées par les concepteurs qui ne sont pas familiers avec le canevas BW-$\mathcal{M}$ lors de son utilisation pour la conception d'un système.

En ce qui concerne les utilisateurs, nous envisageons d'étudier leurs réactions face au processus de filtrage. L'idée est d'analyser si les utilisateurs perçoivent les résultats du processus de filtrage face aux préférences qui ont été associées à leur contexte courant. Une telle étude permettrait d'évaluer le degré de satisfaction des utilisateurs face aux résultats, et notamment si, pour eux, les résultats apportés par le processus de filtrage compensent le temps dépensé pour la définition des profils. Une telle étude demande la définition d'une méthodologie de test précise, car l'observation d'utilisateurs nomades est rendue difficile par le caractère ubiquitaire de leurs interactions avec le système.

(2) L'exploitation de l'aspect temporel des profils

Par ailleurs, l'association d'un aspect temporel aux profils se présente également comme une perspective intéressante pour ce travail. L'idée ici serait de permettre la définition d'un profil pour un intervalle de temps donné. Chaque profil aurait ainsi un temps de vie

limité au delà duquel il ne serait plus valide ni applicable. Cette perspective rendrait possible, par exemple, la définition d'une politique de contrôle d'accès valable pour une période donnée. L'idée ici serait d'utiliser les stratifications associées au profil pour montrer ou cacher certaines informations pour certains utilisateurs durant un certain temps.

(3) Application du processus de filtrage

Une autre perspective intéressante pour ce travail considère l'application du processus de filtrage en dehors du cadre des collecticiels sur le Web. Même si tout le processus de filtrage que nous avons proposé vise à adapter l'information de conscience de groupe au sein d'un collecticiel sur le Web, nous pouvons envisager son application à d'autres types d'information. Ceci est possible grâce à l'utilisation du modèle de contenu qui est centré sur la notion abstraite d'événement. Il suffirait donc de spécialiser la notion d'événement en un type d'information spécifique à un domaine. De même en ce qui concerne l'application du processus à d'autres systèmes sensibles au contexte. Même si le processus a été conçu pour les collecticiels, l'application de ses principes peut se faire en dehors du cadre d'un système coopératif. Pour cela, l'adaptation des éléments de contexte aux concepts manipulés par le système en question nous semble essentielle.

Dans ce sens, une perspective serait l'application du processus de filtrage dans une architecture pair-à-pair[105] (P2P). Ces architectures sont de plus en plus utilisées pour la conception des collecticiels. Elles se démarquent par leur caractère totalement distribué qui contraste avec le caractère centralisé de l'architecture client/serveur typique du Web. Une architecture P2P pose plusieurs défis liés au partage, à la diffusion et à l'accès aux informations entre les pairs. De plus, l'architecture P2P accentue les questions liées au respect de la vie privée qui doivent être prises en compte dès lors qu'on parle de la diffusion d'informations contextuelles. La diffusion des informations relatives au contexte de chaque individu dans le réseau de pairs doit donc être assujettie à une politique de contrôle, visant le respect de la vie privée. Une telle politique pourra influencer le processus de filtrage, qui devrait s'adapter à cette architecture et à ces politiques.

(4) Évolution du modèle de contexte

Une dernière perspective que nous voulons souligner ici consiste à étudier la possibilité de déduire de nouvelles connaissances à partir du modèle de contexte. L'idée consiste à étudier les possibilités d'interprétation des informations représentées dans le modèle. Pour cela, nous devons analyser la possibilité de définir des contraintes supplémentaires sur les instances du modèle, afin de guider un processus d'interprétation. Un tel processus serait particulièrement intéressant pour l'extraction de connaissances à partir d'un ensemble d'informations incomplètes ou ambiguës. L'idée serait donc d'enrichir les connaissances sur le contexte par la déduction de nouvelles connaissances à partir de celles disponibles.

[105]« peer-to-peer »

Références

[Agostini 2000] Agostini, A., Michelis, G., « A light workflow management system using simple process models », *Computer Supported Cooperative Work (CSCW)*, vol. 9, n° 3-4, 2000, Kluwer Academic Publishers, pp. 335 – 363.

[Alarcón 2004] Alarcón, R., Collazos, C., Guerrero, L.A. « Distributed shared contexts ». In : Karmouch, A., Korba, L. and Madeira, E. (Eds.), *LNCS 3284 - 1st International Workshop on Mobility Aware Technologies and Applications - MATA 2004*, Florianópolis, Brésil, Springer-Verlag, 2004, pp. 27-36.

[Alarcón 2002] Alarcón, R., Fuller, D., « Intelligent awareness in support of collaborative virtual work groups ». In : Haake, J.M., Pino, J.A. (Eds.), *LNCS 2440 - 8th International Workshop on Groupware: Design, Implementation and Use - CRIWG 2002*, Springer-Verlag, 2002, pp. 168-188.

[Allen 1983] Allen, J. F., « Maintaining Knowledge about Temporal Intervals », *Communications of the ACM,* vol. 26, n° 11, nov. 1983, pp. 832-843.

[Anne 2005] Anne, M., Crowley, J.L., Devin, V., Privat, G., « Localisation intra-bâtiment multi-technologies : RFID, Wifi et vision », In : Coutaz, J., Lecomte, S. (Eds.), *Actes des deuxièmes journées francophones : Mobilité et Ubiquité 2005 (UbiMob'05)*, 31 mai – 3 juin 2005, Grenoble, France, pp. 29-35.

[Appelt 2001] Appelt, W., « What groupware functionality do users really use? Analysis of the usage of the BSCW system », *9th Euromicro Workshop on PDP 2001*, 2001, IEEE Computer Society.

[Austin 2004] Austin, D., Barbir, A., Ferris, C., Garg, S., « Web Services Architecture Requirements », W3C Working Group Note, 11 February 2004, W3C. Disponible sur http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211 (août 2005).

[BBC 2005] BBC News, « *Online drives entertainment sales* ». Disponible sur http://news.bbc.co.uk/2/hi/entertainment/4119806.stm (22 juin 2005).

[Banerjee 2002] Banerjee, S., Agarwal, S., Kamel, K., Kochut, A., Kommareddly, C., Nadeem, T., Thakkar, P., Trinh, B., Yossef, A., Larson, R.L., Shankar, A.U., Agrawala, A., « Rover: scalable location-aware computing », *Computer*, vol. 35, n°10, octobre 2002, IEEE Computer Society, pp. 46-53.

[Bardram 2004] Bardram, J.E., Hansen, T.R., « The AWARE architecture: supporting context-mediated social awareness in mobile cooperation », *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW'04)*, Chicago, USA, Novembre 6-10, 2004, ACM Press, pp.192 – 201.

[Bardram 2005] Bardram, J.E., « The Java Context Awareness Framework (JCAF) – a service infrastructure and programming framework for context-aware applications », In : Gellersen, H.W, Want, R., Schmidt, A. (Eds), *LNCS 3468 - Third International Conference in Pervasive Computing (Pervasive'2005)*, Munich, Germany, May 8-13, 2005, Springer-Verlag, pp. 98-115.

[Beineke 1988] Beineke, L.W., Wilson, R.J., *« Selected topics in graph theory: 3 »*, Academic Press Limited, Cambridge, 1988. pp. 210.

[Benali 2002] Benali, K., Bourguin, G., David, B., Derycke, A., Ferraris, C., « Collaboration/Coopération ». In : Le Maitre, J. (Ed.), *Actes des 2e Assises Nationales du GdR I3 : Information – Interaction – Intelligence*, Nancy, France, décembre 2002, Cépadues Éditions, pp. 79-94.

[Benerecetti 2001] Benerecetti, M., Bouquet, P., Bonifacio, M., « Distributed context-aware systems », *Human-Computer Interaction*, vol. 16, n° 2-4, 2001, pp. 231-228.

[Berge 1970] Berge, C., « *Graphes et hypergraphes* », Dunod, Paris, 1970. pp. 474.

[Berge 1988] Berge, C., « Hypergraphes », In : Beineke, L.W., Wilson, R.J. (Eds.), « *Selected topics in graph theory: 3* », Academic Press Limited, Cambridge, 1988. pp. 189-206.

[Billsus 2002] Billsus, D., Brunk, C.A., Evans, C., Gladish, B., Pazzani, M., « Adaptative interfaces for ubiquitous web access », *Communications of the ACM*, vol. 45, n° 5, mai 2002, ACM Press, pp. 34-38.

[Bisson 1995] Bisson, G., « Why and how to define a similarity measure for object based representation systems », In : *Mars, N. (Ed.), Towards very large knowledge bases, 2nd International Conference on Building and Sharing Very Large-Scale Knowledge Bases (KBKS)*, Enschede, Pays Bas, avril 1995, IOS press, pp. 236-246.

[Borges 2004] Borges, M.R.S., Brézillon, P., Pino, J., Pomerol, J.-C., « Bringing context to CSCW », *Computer Supported Cooperative Work in Design 2004 (CSCWD'2004)*, International Academic Publishers / Beijing World Publishing Corporation, pp. 161-166.

[Bouthier 2004] Bouthier, C., « Mise en contexte de la conscience de groupe : adaptation et visualisation », Thèse de Doctorat, Institut National Polytechnique de Lorraine, Nancy, France, 2004.

[Boyle 2002] Boyle, M., Greenberg, S., « GroupLab Collabrary: a toolkit for multimedia groupware », In : Patterson, J. (Ed.), *ACM CSCW 2002 Workshop on Network Services for Groupware*, 2002. Disponible sur http://grouplab.cpsc.ucalgary.ca/papers/2002.html (juillet 2005).

[Boyle 2005] Boyle, M., Greenberg, S., « The Language of Privacy: Learning from Video Media Space Analysis and Design », *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 12, n° 2, juin 2005, ACM Press, pp. 328-370.

[Brézillon 2002a] Brézillon, P., « Modeling and using context: past, present and futur », Rapport de Recherche LIP6 2002/010, 2002. Disponible sur http://www.lip6.fr/reports/lip6.2002.010.html. (mai 2003)

[Brézillon 2002b] Brézillon, P., « Expliciter le contexte dans les objets communicants », In : Kintzig, C., Poulain, G., Privat, G., Favennec, P-N. (Eds.), *Objets Communicants*, Hermes Science Publications, Paris, 2002, pp. 293-303.

[Brown 1997] Brown, P.J., Bovey, J.D., Chen, X., « Context-aware applications: from the laboratory to the marketplace », IEEE Personal Communications, vol. 4, n° 5, octobre 1997, pp. 58-64.

[Bruijn 2003] Bruijn, J. de, « Using Ontologies: Enabling Knowledge Sharing and Reuse on the Semantic Web », DERI Technical Report DERI-2003-10-29, octobre 2003. Disponible sur http://deri.ie/publications/techpapers/documents/DERI-TR-2003-10-29.pdf (octobre 2005).

[Bruley 2003] Bruley, C., Genoud, P., Dupierris, V., « Guide utilisateur AROM v2.0 », mars 2003. Disponible sur ftp://ftp.inrialpes.fr/pub/romans/logiciels/arom2/ug.pdf (juin 2005).

[BSCW 2005] http://bscw.fit.fraunhofer.de/ (juillet 2005)

[Bucur 2005] Bucur, O., Beaume, P., Boissier, O., « Définition et représentation du contexte pour des agents sensibles au contexte », In : Coutaz, J., Lecomte, S. (Eds.), *Actes des deuxièmes journées francophones : Mobilité et Ubiquité 2005 (UbiMob'05)*, 31 mai – 3 juin 2005, Grenoble, France, pp. 13-16.

[Burrell 2002] Burrell J., Gray G.K., Kubo K., Farina, N., « Context-aware computing: a text case », In : Borriello G., Holmquist L.E. (eds.), *LNCS 2498 - 4th International Conference on Ubiquitous Computing*, 2002, Springer-Verlag, pp. 1-15.

[Canals 2002] Canals, G., Nigay, L., Pucheral, P., « Mobilité : accès aux données et interaction homme-machine », *Actes des 2e Assises Nationales du GdR I³ : Information – Interaction – Intelligence*, Nancy, France, décembre 2002, Cépadues Éditions, pp. 119-139.

[Capponi 1995] Capponi, C., « Identification et exploitation des types dans un modèle de connaissances à objets », Thèse de Doctorat, Université Joseph Fourier, Grenoble, France, octobre 1995.

[Capponi 2005] Capponi, C., Chabalier, J., « Cycles of classification in the knowledge representation system AROM », article fourni directement par l'auteur, juin 2005.

[Carroll 2003] Carroll, J.M., Neale, D.C., Isenhour, P.L., Rosson, M.B., McCrickard, D.S., « Notification and awareness : synchronizing task-oriented collaborative activity », *International Journal of Human-Computer Studies (IJHCS)*, vol. 58, 2003, Elsevier, pp. 605-632.

[Chaari 2004] Chaari, T., Laforest, F., Celentano, A., « Design of context-aware applications based on web services », Technical Report RR-2004-033, LIRIS, Lyon, octobre 2004. Disponible sur http://liris.cnrs.fr/publis/rr_html (décembre 2004).

[Chaari 2005] Chaari, T., Laforest, F., Flory, A., « Adaptation des applications au contexte en utilisant les services Web », In : Coutaz, J., Lecomte, S. (Eds.), *Actes des deuxièmes journées francophones : Mobilité et*

Ubiquité 2005 (UbiMob'05), 31 mai – 3 juin 2005, Grenoble, France, pp. 111-118.

[Chabalier 2003] Chabalier, J., Fichant, G., Capponi, C., « La classification récursive dans AROM », *Actes 9ème conférence francophone Langages et Modèles Objets (LMO'03), RSTI – L'objet*, vol. 9, n°1-2, 2003, pp. 167-181, 2003.

[Chabalier 2004] Chabalier, J., « Acquisition incrémentale et représentation des systèmes intégrés bactériens par une approche orientée-objet », Thèse de Doctorat, Université de Provence, 2004.

[ChalmersD 2002] Chalmers, D., « Contextual mediation to support ubiquitous computing », Thèse de Doctorat, University of London, 2002.

[ChalmersM 2002] Chalmers, M., « Awareness, Representation and Interpretation », *Computer Supported Cooperative Work*, vol. 11, n° 3-4, septembre 2002, Kluwer Academic Publishers, pp. 389-409.

[Cheverest 2002] Cheverest, K., Mitchell, K., Davies, N., « The role of adaptive hypermedia in a context-aware tourist guide », *Communication of ACM*, vol. 45, n° 5, mai 2002, ACM Press, pp. 47-51.

[Coppola 2003] Coppola, P., Mea, V.D., Gaspero, L. Di, Mizzaro, S., « The concept of relevance in mobile and ubiquitous information access », In : Frestani, F., Dunlop, M., Mizzaro, S. (Eds.), *LNCS2954 - International Workshop on Mobile and Ubiquitous Information Access (Mobile HCI 2003)*, Udine, Italy, Springer-Verlag, 2004, pp. 1-10.

[Corcho 2000] Corcho, O., Gómez-Pérez, A., « Evaluating knowledge representation and reasoning capabilities of ontology specification languages », *14th European Conference on Artificial Intelligence (ECAI'00) – Workshop on Application of Ontology and Problem Solving Methods*, Berlin, Germany, 2000. Disponible sur http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/3.ps (avril 2003).

[Coutaz 2003] Coutaz, J., Crowley, J.L., Dobson, S., Garlan, D., « Context is the key », *Communication of the ACM*, vol. 48, n° 3, mars 2003, ACM Press, pp. 49-53.

[David 2001] David, B., « IHM pour les collecticiels », *Réseaux et Systèmes Réparties (RSR-CP)*, vol. 13, 2001, Hermes Science, pp. 169-206.

[DavidB 2001] David, J.M.N., Borges, M.R.S., « Improving the selectivity of awareness information in groupware applications », *The Sixth International Conference on Computer Supported Cooperative Work in Design (CSCWD'2001)*, 2001, IEEE Computer Society, pp. 41-46.

[Decouchant 2001] Decouchant, D., Favela, J., Martinez-Enriquez, A.M., « PINAS: A Middleware for Web Distributed Cooperative Authoring », *Symposium on Applications and the Internet (Saint'2001)*, janvier 2001, IEEE Computer Society, pp. 187-193.

[Dey 2000] Dey, A.K., « Providing Architectural Support for Building Context-Aware Applications », PhD Thesis, Georgia Institute of Technology, 2000.

[Dey 2001] Dey, A.K. « Understanding and using context ». *Personal and Ubiquitous Computing*, vol. 5, n° 1, 2001, pp. 4-7.

[Dias 1998] Dias, M.S., « COPSE: Um ambiente de suporte ao projeto cooperativo de software », Dissertation de Master (Dissertação de mestrado), COPPE - Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brésil, 1998 (langue : portugais).

[Dias 1999] Dias, M.S., Borges, M.R.S., « Development of groupware systems with the COPSE infrastructure », *Proceedings of International Workshop on Groupware (CRIWG'99),* IEEE Computer Society, Cancun, 1999, pp.278-285.

[Dourish 1992] Dourish, P., Bellotti, V., « Awareness and Coordination in Shared Workspaces », *Proceedings of ACM Conference on Computer-Supported Cooperative Work* (CSCW'92). ACM Press, pp. 107-114.

[Dourish 2001] Dourish, P., « Seeking a foundation for context-aware computing », *Human Computer Interaction*, vol. 16, n° 2-4, 2001, pp. 229-241.

[Dourish 2004] Dourish, P., « What we talk about when we talk about context », *Personal and Ubiquitous Computing*, vol. 8, n° 1, 2004, pp. 19-30.

[ECOO 2001] Projet ECOO, « Projet ECOO : environnements et coopération », Rapport d'activité, 2001, INRIA. Disponible sur http://www.inria.fr/rapportsactivite/RA2001/ecoo/ecoo.pdf (juillet 2005).

[ECOO 2004] Project-Team ECOO, « Middleware for supporting cooperative work through Internet », Rapport d'activité, 2004, INRIA. Disponible sur http://www.inria.fr/rapportsactivite/RA2004/ecoo2004/ecoo.pdf (julliet 2005).

[Elliott 2005] Elliott, P., « *Warcraft sets gaming standards* ». Disponible sur http://news.bbc.co.uk/2/hi/technology/4253647.stm (14 février 2005).

[Ellis 1991] Ellis, C. A., Gibbs, S.J., Rein, G.L., « Groupware: Some issues and experiences », *Communications of the ACM*, vol. 34, n° 1, janvier 1991, ACM Press, pp. 38-58.

[Espinosa 2000] Espinosa, A., Cadiz, J., Rico-Gutierrez, L., Kraut, R., Scherlis, W., Lautenbacher, G., « Coming to the wrong decision quickly: why awareness tools must be matched with appropriate tasks », *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2000)*, The Hague, The Netherlands, avril 2000, ACM Press, pp. 392-399.

[Espinoza 2001] Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E., Bylund, M., « GeoNotes: Social and Navigational Aspects of Location-Based Information Systems », In : Abowd, G.D., Brumitt, B., Shafer, S. (Eds.), *LNCS 2201 - Third International Conference on Ubiquitous Computing (Ubicomp 2001)*, Atlanta, USA, 2001, Springer-Verlag, pp. 2-17.

[Farschian 2001] Farschian, B. A., « Integrating geographically distributed development teams through increased product awareness », *Information System Journal*, vol. 26, nº 3, 2001, pp. 123-141.

[Fernández 2002] Fernández, A., Haake, J.M., Goldberg, A., « Tailoring group work », In : Haake, J.M, Pino, J.A. (Eds.), *LNCS 2440 - International Workshop on Groupware (CRIWG 2002)*, 2002, Springer-Verlag, pp. 232-242.

[Ferris 2003] Ferris, C., Farrel, J., « What are Web services? », *Communication of the ACM*, vol. 46, n° 6, juin 2003, ACM Press, pp. 31.

[Forest 2005a] Forest, J., « LibreSource: Overview and Quick Start », Technical Documentation. Disponible sur http://dev.libresource.org/home/community/documentation/pdf/tn02 (mars 2006).

[Forest 2005b] Forest, J., Jourdain, S., Jouille, F., « LibreSource Plate-forme Libre de Travail Collaboratif », Rencontres Mondiale du Logiciel Libre (RMLL 2005), présentation. Disponible sur http://dev.libresource.org/home/community/documentation/pdf/presRMLL2005 (mars 2006).

[Gamma 1994] Gamma, E., Helm, R., Johnson, R., Vlissides, J., « *Design patterns: elements of reusable object-oriented software* », Addison-Wesley, 1994, pp. 394.

[Genoud 2000] Genoud, P., Dupierris, V., Page, M., Bruley, C., Ziébelin, D., Gensel, J., Bardou, D., « From AROM, a new object based knowledge representation system, to WebAROM, a knowledge bases server », *Workshop Application of Advanced Information Technologies to Medecine, AIMSA 2000*, septembre 2000. Disponible sur ftp://ftp.inrialpes.fr/pub/romans/logiciels/arom/aimsa2000pg.pdf (décembre 2005).

[Gensel 1995] Gensel, J., « Contraintes et représentation de connaissances par objets : application au modèle TROPES », Thèse de Doctorat, Université Joseph Fourier, Grenoble, France, octobre 1995.

[Gensel 2006] Gensel, J., Capponi, C., Genoud, P., Ziébelin, D., « Vers une intégration des relations Partie-Tout en AROM », *Langages et Modèles à Objets (LMO'2006)*, Nimes, mars, 2006.

[Goland 1999] Goland, Y., Whitehead, E., Faisi, A., Jensen, D., « HTTP Extensions for Distributed Authoring - WebDAV », RFC 2518. IETF, février 1999. Disponible sur http://andrew2.andrew.cmu.edu/rfc/rfc2518.html. (septembre, 2002).

[Greenberg 1996] Greenberg, S., « Peepholes: Low cost awareness of one's community », *ACM SIGCHI'96 Conference on Human Factors in Computing System (CHI'96), Companion Proceedings*, 1996, pp. 205-215. Disponible sur http://grouplab.cpsc.ucalgary.ca/papers/1996/96-ShortPapers.CHI/2-Peepholes/peepholes.pdf (juillet 2005).

[Greenberg 2001] Greenberg, S., « Context as a dynamic construct », *Human-Computing Interaction*, vol. 16, n° 2-4, 2001, pp. 257-268.

[Grinter 2000] Grinter, R.E., « Workflow systems: occasions for success and failure », *Computer Supported Cooperative Work*, vol. 9, n° 2, 2000, Kluwer Academic Publishers, pp. 189-214.

[Grudin 1994] Grudin, J., « Computer-Supported Cooperative Work: History and Focus », *Computer*, vol. 27, n° 5, mai 1994, IEEE Computer Society, pp. 19-26.

[Grudin 2001] Grudin, J., « Desituating action: digital representation of contexte », *Human-Computing Interaction*, vol. 16, n° 2-4, 2001, pp. 269-286.

[Guerrero 2004] Guerrero, L.A., Piño, J.A, Collazos, C.A., Inostroza, A., Ochoa, S.F., « Mobile support for collaborative work », In : Vreede, G.-J., Guerrero, L., Raventós, G.M. (Eds.), *LNCS 3198 - X International Workshop on Groupware (CRIWG'04)*, San Carlos, Costa Rica, septembre 2004, Springer-Verlag, pp. 363-375.

[Gutwin 2002] Gutwin, C., Greenberg, S., « A Descriptive Framework of Workspace Awareness for Real-Time Groupware », *Computer Supported Cooperative Work (CSCW)*, vol. 11, n° 3-4, septembre 2002, Kluwer Academic Publishers, pp. 411 – 446.

[Heflin 2004] Heflin, J., « OWL Web ontologie language use cases and requirements », W3C Recommandation, 10 février 2004. Disponible sur http://www.w3.org/TR/webont-req/ (octobre 2005).

[Henricksen 2002] Henricksen, K., Indulska, J., Rakotonirainy, A., « Modeling context information in pervasive computing systems », In : Mattern, F., Naghshineh, M., (Eds.), *LNCS 2414 - First International Conference in Pervasive Computing (Pervasive'2002)*, Zürich, Switzerland, août 2002, Springer-Verlag, pp. 167-180.

[Heusse 2003] Heusse, M., Rousseau, F., Berger-Sabbatel, G., Duda, A., « Performance anomaly of 802.11b », *Proceedings of Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, San Francisco, USA, mars 2003, pp. 836-843, vol. 2. Disponible sur http://drakkar.imag.fr/IMG/pdf/perfAnomaly-infocom.pdf (novembre 2005).

[Hibino 2002] Hibino S., Mockus, A., « handiMessenger: awareness-enhanced universal communication for mobile users », *LNCS 2411 - 4th International Symposium on Mobile Human-Computer Interaction (Mobile HCI 2002)*, 2002, Springer-Verlag, pp. 170-183.

[Hill 2004] Hill, J., Gutwin, G., « The MAUI toolkit: groupware widgets for group awareness », *Computer Supported Cooperative Work (CSCW)*, vol. 13, n° 5-6, 2004, Springer-Verlag, pp. 539–571.

[Horridge 2004] Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C., « A Practical Guide To Building OWL Ontologies With The Protégé-OWL Plugin », University of Manchester, août 2004. Disponible sur : http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf (janvier 2006).

[Horrocks 2003] Horrocks, I., Patel-Schneider, P.F., van Harmelen, F., « From SHIQ and RDF to OWL: The making of a web ontology language », *Journal of Web Semantics*, vol. 1, n° 1, 2003, pp. 7-26. Disponible sur : http://www.cs.man.ac.uk/%7Ehorrocks/Publications/download/2003/HoPH03a.pdf (janvier 2006).

[Jeantet 1998] Jeantet, A., « Les objets intermédiaires dans la conception. Eléments pour une sociologie des processus de conception », *Sociologie du Travail*, vol. 40, n° 3, 1998, Dunod, Paris. pp. 291-316.

[Jiang 2001] Jiang, J., Shi, M., « What should transactions provide for CSCW applications? », *Proceedings of International Conferences on Info-tech and Info-net (ICII 2001)*, Beijing, China, 29 octobre - 1 novembre 2001, vol. 5, IEEE Computer Society, pp.264-269.

[Jing 1999] Jing, J., Helal, A.S., Elmagarmid, A., « Client-server computing in mobile environments », *ACM Computer Surveys*, vol. 31, n° 1, juin 1999, ACM Press, pp.117-157.

[Keidl 2004] Keidl, M., Kemper, A. « Towards Context-Aware Adaptable Web Services », *Proceedings of the 13th World Wide Web Conference (WWW 2004)*, New York, USA, mai 2004. Disponible sur http://www2004.org/proceedings/docs/contents.htm (août 2005).

[Kirsh 2001] Kirsh D., « The context of work », *Human Computer Interaction*, vol. 16, n° 2-4, 2001, pp. 305-322.

[KirschPi 2001] Kirsch-Pinheiro, M., Lima, J.V., Borges, M.R.S., « Awareness em Sistemas de Groupware », *Proceedings of IDEAS'01*, Centre de Información Tecnológica (CIT), San Diego, Costa Rica, 2001, pp 323-335 (langue : portugais). Disponible sur http://www-lsr.imag.fr/Les.Personnes/Manuele.Kirsch-Pinheiro/AwarenessV2.pdf (juillet, 2005).

[KirschPi 2003a] Kirsch-Pinheiro, M., Lima, J.V., Borges, M.R.S., « A Framework for Awareness Support in Groupware Systems », *Computer in Industry*, vol. 52, n° 1, septembre 2003, Elsevier, 2003, pp. 47-57.

[KirschPi 2003b] Kirsch-Pinheiro, M., Villanova-Oliver, M., Gensel, J.; Lima, J.V., Martin, H., « Providing a Progressive Access to Awareness Information », In : Meersman, R., Tari, Z., Schmidt, D. C. (Eds.), *LNCS 2888 - CoopIS/DOA/ODBASE 2003*, Catânia, Italie, 2003, Springer-Verlag, pp. 336-353.

[Klyne 2004] Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butter, M.H., Tran, L., « Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0 », W3C Recommendation, 15 janvier 2004. Disponible sur http://www.w3.org/TR/CCPP-struct-vocab/ (octobre 2004).

[Laurillau 2002] Laurillau, Y., « Conception et réalisation logicielles pour les collecticiels centrées sur l'activité de groupe : le modèle et la plate-forme Clover », Thèse de Doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, septembre 2002.

[Larousse 1992] « *Larousse de Poche* », Larousse, 1992.

[Leiva-Lobos 2002] Leiva-Lobos, E.P., Covarrubias, E., « The 3-ontology: a framework to place cooperative awareness », In : Haake, J.M. and Pino, J.A. (Eds.), *LNCS 2440 - 8th Int. Workshop on Groupware: Design, Implementation and Use - CRIWG 2002*, Springer-Verlag, 2002, pp. 189-199.

[Lemlouma 2004a] Lemlouma T., « Architecture de négociation et d'adaptation de Services Multimédia dans des Environnements Hétérogènes », Thèse de Doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, avril 2004.

[Lemlouma 2004b] Lemlouma T., Layaïda N., « Context-Aware Adaptation for Mobile Devices », *IEEE International Conference on Mobile Data Management*, 2004, IEEE Computer Society, pp. 106-111.

[Liechti 2000] Liechti, O., « Awareness and the WWW: an overview », *Proceedings of ACM Conference on Computer-Supported Cooperative Work (CSCW'00), Workshop on Awareness and the WWW*. Disponible sur http://www2.mic.atr.co.jp/dept2/awareness/ (juillet 2005)

[Lopez 2003] Lopez, P.G., Skarmeta, A.F.G., « ANTS Framework for cooperative work environments », *Computer*, vol. 36, n° 3, IEEE Computer Society, mars 2003, pp. 56-62.

[LopezVel 2005] Lopez-Velasco, C., « AWSDL : une extension de WSDL pour des services Web adaptés », 23° congrès INFORSID 2005, Grenoble, 24-27 mai, 2005, pp. 133-148.

[Manheim 1998] Manheim, M.L., « Beyond Groupware and Workflow : The theory of cognitive informatics an dits implications for a people-based enterprise information architecture », White Paper, 1998. Disponible sur http://www.e-workflow.org/White_Papers/index.htm (novembre 2004).

[Manola 2004] Manola, F., Miller, E., McBride, B., « RDF Primer », W3C Recommendation, 10 février 2004. Disponible sur http://www.w3.org/TR/rdf-primer/ (octobre 2005).

[Marchand 2004] Marchand, C., « Mise au point d'algorithmes répartis dans un environnement fortement variable, et expérimentation dans le contexte des pico-réseaux », Thèse de Doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, 2004.

[Martínez 2002a] Martínez-Enríquez, A.M., Decouchant, D., Morán, A.L, Favela, J., « An adaptative cooperative web authoring environment », In : De Bra, P., Brusilovsky, P., Conejo, R. (Eds.), *2nd International Conference on Adatative Hypermedia and Adaptive Web-based Systems (AH 2002), 2002, LNCS 2347*, Springer-Verlag, pp. 535-538.

[Martínez 2002b] Martínez-Enríquez, A.M., Muhammad, A., Decouchant, D., Favela, J., « An inference engine for web adaptive cooperative work », In : Coello Coello, C.A., Albornoz, A., Sucar, L.E., Battistutti, O.C. (Eds.), *LNAI 2313 - 2nd Mexican International Conference on Artificial Intelligence: Adavances in Artificial Intelligence (MICAI 2002)*, Springer-Verlag, pp. 526-535.

[McEwan 2005] McEwan, G., Greenberg, S., « Supporting social worlds with the Community Bar », *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (Group 2005)*, 6-9 novembre 2005, Sanibel Island, Florida,USA, ACM Press, pp. 31-30. Disponible sur http://grouplab.cpsc.ucalgary.ca/papers/2005/05-CommunityBar/CommunityBar-report2005-789-20.pdf (août 2005).

[Miron 2006] Miron, A.D., « Représentation de Connaissances par Objets pour le Web sémantique ou comment rapprocher AROM de OWL », Rapport de Master 2 Recherche en Informatique, 21 juin 2006, Université Joseph Fourier, Grenoble.

[Moisuc 2004] Moisuc, B., Gensel, J., Martin, H., « Représentation de connaissances par objets pour les SIG à représentations multiples », *7ème Conférence du GDR SIGMA : Géomatique et Analyse Spatiale - CASSINI'04*, Grenoble, 2-4 juin 2005, pp. 95-101.

[Moisuc 2005] Moisuc, B., Davoine, P.-A., Gensel, J., Martin, H., « Design of Spatio-Temporal Information Systems for Natural Risk Management with an Object-Based Knowledge Representation Approach », *Geomatica*, vol. 59, n° 4, 2005.

[Molli 2001] Molli, P., Skaf-Molli, H., Bouthier, C., « State treemap: an awareness widget for multi-synchronous groupware », *7th Int. Workshop on Groupware (CRIWG'01)*, Darnstadt, Germany, 2001, IEEE Computer Society, pp. 106-114.

[Moran 2001] Moran, T., Dourish, P., « Introduction to this special issue on context-aware computing », *Human-Computer Interaction*, vol. 16, n° 2-3, 2001.

[Mostéfaoui 2004] Mostéfaoui, K., Pasquier-Rocha, J., Brézillon, P., « Context-aware computing: a guide for the pervasive computing community », *Proceedings of the IEEE/ACS International Conference on Pervasive Services (IPCS'04*), IEEE Computer Society, 2004, pp. 39-48.

[Muñoz 2003] Muñoz M., A., Rodríguez M., Favela J., Martinez-Garcia A.I., Gonzalez V.M., « Context-aware mobile communication in hospitals », *Computer*, vol. 36, n° 9, 2003, IEEE Computer Society, pp. 38-46.

[Napoli 2004] Napoli, A., Carré, B., Ducournau, R., Euzanat, J., Rechenmann, F., « Objets et représentation, un couple en devenir », In : Briot, J.-P. (Ed.), *L'Objet : logiciel, bases de données, réseaux - Des octets aux modèles, vingt ans après : où en sont les objets ?, RSTI série L'Objet*, vol. 10, N° 4, 2004, pp. 61-81.

[ObjectWeb 2005] ObjectWeb, « *What is Middleware* », disponible sur http://middleware.objectweb.org/ (juillet, 2005).

[O'Hare 2002] O'Hare, G., O'Grady, M., « Addressing mobile HCI needs through agents », In : Paternò, F. (Ed.), *LNCS 2411 - 4th International Symposium on Mobile Human-Computer Interaction (Mobile HCI 2002)*, Springer-Verlag, 2002, pp. 311-314.

[OMG 2004] OMG, « Unified Modeling Language (UML) Specification: Infrastructure: version 2.0 », novembre 2004. Disponible sur http://www.omg.org/docs/ptc/04-10-14.pdf (décembre 2005).

[Page 2000] Page, M., Gensel, J., Capponi, C., Bruley, C., Genoud, P., Ziébelin, D., « Représentation de connaissances au moyen de classes et d'associations : le système AROM », *6èmes Journées Langages et Modèles à Objets 2000*, janvier 2000, pp. 91-106. Disponible sur http://www-lsr.imag.fr/Les.Groupes/sigma/articles/page00a.pdf (mars 2005).

[Page 2001] Page, M., Gensel, J., Capponi, C., Bruley, C., Genoud, P., Ziébelin, D., Bardou, D., Dupierris, V., « A New Approach in Object-Based Knowledge Representation: The AROM System », In: Monostori, L., Váncza, J., Ali, M. (Eds.), *LNAI 2070 - 14th International Conference*

on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2001), Budapest, Hongrie, juin 2001, Springer-Verlag, pp. 113-118.

[Perry 2001] Perry, M., O'Hara, K., Sellen, A., Brown, B., Harper, R. « Dealing with mobility: understanding access anytime, anywhere », *ACM Transactions on Computer-Human Interaction*, vol. 8, n° 4, ACM Press, 2001, pp. 323-347.

[Prakash 1999] Prakash, A., Shim, H.S., Lee, J.H., « Data management issues and trade-offs in CSCW systems », *IEEE Transactions on Knwoledge and Data Engineering*, vol. 11, n° 1, jan./fev. 1999, IEEE Computer Society, pp. 213-227.

[Prante 2004] Prante, T., Streitz, N.A., Tandler, P., « Roomware: computers disappear and interaction evolves », *Computer*, vol. 37, n° 12, décembre 2004, IEEE Computer Society, pp. 47-54.

[PWC 2005] PricewaterhouseCoopers (PWC), « Global Entertainment and Media Outlook: 2005-2009 ». Disponible sur http://www.pwc.com/extweb/industry.nsf/0/6BB1D7B2F2463E1885256CE8006C6ED6?opendocument&vendor=none (juillet 2005).

[Pujolle 2003] Pujolle, G., « *Les réseaux : édition 2003* », Eyrolles, 2003.

[Renevier 2004] Renevier, P., « Systèmes Mixtes Collaboratifs sur Supports Mobiles : Conception et Réalisation », Thèse de Doctorat, Université Joseph Fourrier, Grenoble, France, 2004.

[Rey 2004] Rey, G., Coutaz, J., « Le contexteur : capture et distribution dynamique d'information contextuelle », *Mobilité & Ubiquité'04 (UbiMob'04)*, Nice, France, 2004. pp. 131-138.

[Roseman 1996] Roseman, M., Greenberg, S. « Building real time groupware with GroupKit, a groupware toolkit », *ACM Transactions on Computer Human Interactions*, vol. 3, n° 1, mars 1996, ACM Press, pp. 66-106.

[Rubinsztejn 2004] Rubinsztejn, H.K., Endler, M., Sacramento, V., Gonçalvez, K., Nascimento, F., « Support for context-aware collaboration » In : Karmouch, A., Korba, L., Madeira E. (Eds.), *LNCS 3284 - 1st International Workshop on Mobility Aware Technologies and Applications - MATA 2004*, Florianópolis, Brésil, Springer-Verlag, 2004, pp.37-47.

[Ryden 2005] Ryden, K. (ed.), « OpenGIS® Implementation Specification for Geographicinformation - Simple feature access - Part 2: SQL option », OpenGIS® Implementation Specification, novembre 2005. Disponible sur http://portal.opengeospatial.org/files/?artifact_id=13228 (décembre 2005).

[Salcedo 1998] Salcedo, M. R., « Alliance sur l'Internet : support pour l'édition coopérative des documents structurés sur un réseau à grande distance », Thèse de Doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, 1998.

[Samama 2005] Samama, N., Vervish-Picois, A., Fraçois, M., « La localisation en intérieur à l'aide de répéteurs GPS : vers un système de positionnement universel ? », In : Coutaz, J., Lecomte, S. (Eds.), *Actes des deuxièmes*

journées francophones : Mobilité et Ubiquité 2005 (UbiMob'05), 31 mai – 3 juin 2005, Grenoble, France, pp. 21-28.

[Sarker 2003] Sarker, S., Weels, J.D., « Understanding mobile handheld device use and adoption », *Communication of ACM*, vol. 46, n° 12, décembre 2003, ACM Press, pp. 35-40.

[Schilit 1994a] Schilit, B.N., Theimer, M.M., « Disseminating active map information to mobile hosts », *IEEE Network*, vol. 8, n°5, septembre/octobre 1994, pp. 22-32.

[Schilit 1994b] Schilit, B.N., Adams, N., Want, R., « Context-Aware Computing Applications », *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, USA, décembre 1994, pp. 85-90.

[Schilit 2002a] Schilit, B.N., Hilbert, D.M., Trevor, J., « Context-aware communication », *IEEE Wireless Communications*, vol. 9, n° 5, octobre 2002, IEEE Computer Society, pp. 37-45.

[Schilit 2002b] Schilit, B.N., Trevor, J., Hilbert, D.M., Koh, T.K., « Web interaction using very small Internet devices », *Computer*, vol. 35, n° 10, 2002, IEEE Computer Society, pp. 37-45.

[Schmidt 2002] Schmidt, K., « The problem with 'awareness': introductory remarks on 'Awareness in CSCW' », *Computer Supported Cooperative Work*, vol. 11, n° 3-4, Kluwer Academic Publishers, 2002, pp. 285-298.

[Skaf-Molli 2003] Skaf-Molli, H., Molli, P. Oster, G., Godard, C., « Toxic farm: a cooperative management platform for virtual teams and enterprises », *Proceedings of 5th Int. Conference on Enterprise Information Systems (ICEIS'03).* Disponivel sur http://www.loria.fr/~molli/rech/iceis03/ (juillet 2005).

[Skype 2005] « *Skype – The whole world can talk free* ». Disponible sur http://www.skype.com (juillet 2005).

[Smith 2004] Smith, M.K., Welty, C., McGuinness, D.L., « OWL Web Ontology Language Guide », W3C Recommendation, 10 février 2004. Disponible sur http://www.w3.org/TR/owl-guide/ (oct. 2005).

[Sohlenkamp 1998] Sohlenkamp, M. « Supporting group awareness in multi-user enviroment through perceptualization », Fachbereich Mathematik-Informatik der Universität - Gesamthochschule, 1998. Disponible sur http://orgwis.gmd.de/projects/POLITeam/poliawac/ms-diss/ (juillet 1999).

[Sohlenkamp 2000] Sohlenkamp, M., Mambrey, P., Prinz, W., Fuchs, L., Syri, A., Pankoke-Babatz, U., Klöckner, K., Kolvenbach, S., « Supporting the distributed German government with POLITeam », *Multimedia Tools and Applications*, vol. 12, n° 1, septembre 2000, Kluwer Academic Publishers, pp. 39-58.

[Tam 2004] Tam, J., Greenberg, S., « A framework for asynchronous change awareness in collaboratively-constructed documents », In : G.-J. de Vreede, L.A. Guerrero, G.M. Raventós (Eds.), *LNCS 3198 - International Workshop on Groupware: Design, Implementation and*

Use (CRIWG 2004), San Carlos, Costa Rica, septembre 2004, Springer-Verlag, pp. 67-83.

[Tang 2001] Tang, J.C., Yankelovich, N., Begole, J.B., Vankleike, M., « ConNexus to Awarenex: extending awareness to mobile users », *CHI Letters, CHI 2001*, vol. 3, n° 1, ACM Press, p. 221-229.

[TarpinBe 1997] Tarpin-Bernard, F., « Travail coopératif synchrone assisté par ordinateur : approche AMF-C », Thèse de Doctorat, Ecole Centrale de Lyon, Lyon, France, 1997.

[Teege 2000] Teege, G., « Users as composers: Parts and features as a basis for tailorability in CSCW systems », *Computer Supported Cooperative Work*, vol. 9, 2000, Kluwer Academic Publishers, pp. 101-122.

[Valtchev 1997] Valtchev, P., Euzenat, J., « Dissimilarity measure for collections of objects and values », In : Liu, X., Cohen, P., Berthold, M. (Eds.), *LNCS 1280 - 2nd International Symposium on Intelligent Data Analysis (IDA'97)*, Springer-Verlag, 1997, pp. 259-272.

[Valtchev 1999] Valtchev, P., « Building classes in object-based languages by automatic clustering », In : Hand, D., Kok, J., Berthold, M. (Eds.), *LNCS 1642 - 3rd International Symposium on Intelligent Data Analysis (IDA'99)*, Springer-Verlag, 1999, pp. 303-314.

[Villamil 2004] Villamil, M.-D.-P., Roncancio, C., Labbé, C., « PinS: peer to peer interrogation and indexing systems », *Proceedings of the International Database Engineering and Applications Symposium (IDEAS '04)*, juillet 2004, IEEE Computer Society, pp. 236- 245.

[Villanova 2002] Villanova-Oliver, M., « Adaptabilité dans les systèmes d'Information sur le Web : Modélisation et mise en œuvre de l'accès progressif », Thèse de Doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, décembre 2002. Disponible sur http://www-lsr.imag.fr/Les.Personnes/Marlene.Villanova/ (juillet 2005).

[Wang 2000] Wang, W., Haake, J.M., « Tailoring groupwares: the cooperative hypermedia approach », *Computer Supported Cooperative Work*, vol. 9, n° 1, 2000, Kluwer Academic Publishers, pp. 123-146.

[Warcraft 2005] « *World of Warcraft : community site* ». Disponible sur http://www.worldofwarcraft.com/ (juillet 2005).

[Weiser 1993] Weiser, M., « Some computer science issues in ubiquitous computing », *Communication of the ACM*, vol. 36, n° 7, juillet 1993, ACM Press, pp. 75-84.

[WfMC 2005] WfMC, « *Terminology & Glossary* », février 1999. Disponible sur http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf (juillet 2005).

Annexes

Annexe I – Algorithmes

Chapitre 6

Opérateur Equals

Algorithme 1 : Définition générale de l'opérateur Equals

En ce qui concerne deux objets :

```
EqualsV0_Case1
IN:
    OBJECT o1
    OBJECT o2
    CLASS C
OUT:
    BOOLEAN answer
BEGIN
    answer ← FALSE
    //o1 and o2 belongs to the extension of C
    IF o1 IN EXTENSION (C) AND o2 IN EXTENSION (C)
    THEN
        SET vars ← VARIABLES (C)
        BOOLEAN failed ← FALSE
        //o1 and o2 define the same values for the same variables?
        WHILE NOT failed AND HASNEXT (vars)
        DO
            VARIABLE ai ← NEXT (vars)
            IF o1.ai ≠ NULL AND o2.ai ≠ NULL AND FI (o1.ai, o2.ai) = 0
            THEN
                failed ← TRUE
            FI
        DONE
        IF failed
        THEN
            answer ← FALSE
        ELSE
            answer ← TRUE
        FI
    FI
END
```

En ce qui concerne deux tuples :

```
EqualsV0_Case2
IN:
  TUPLE t1
  TUPLE t2
  ASSOCIATION A
OUT:
  BOOLEAN answer
BEGIN
  answer ← FALSE
  //t1 and t2 are instances of the association A?
  IF t1 IN EXTENSION (A) AND t2 IN EXTENSION (A)
  THEN
      SET vars ← VARIABLES (A)
      BOOLEAN failed ← FALSE
      //t1 and t2 define the same values for the same variables?
      WHILE NOT failed AND HASNEXT (vars)
      DO
        VARIABLE ai ← NEXT (vars)
        IF t1.ai ≠ NULL AND t2.ai ≠ NULL AND FI (t1.ai, t2.ai) = 0
        THEN
          failed ← TRUE
        FI
      DONE
      IF NOT failed
      THEN
        //t1 and t2 have equal objects playing the same roles?
        SET roles ← ROLES (A)
        WHILE NOT failed AND HASNEXT (roles)
        DO
```

```
            ROLE rj ← NEXT (roles)
            IF NOT EQUALS (t1.rj, t2.rj)
            THEN
                failed ← TRUE
            FI
          DONE
        FI
        IF NOT failed
            THEN
                answer ← TRUE
            ELSE
                answer ← FALSE
        FI
  FI
END
```

Algorithme 2 : Définition de l'opérateur Equals par poids

```
EqualsV2
IN:
  INSTANCE t1
  INSTANCE t2
  REAL limit
  STRUCTURE S
OUT:
  BOOLEAN answer
BEGIN
  answer ← FALSE
  //t1 and t2 belongs to the extension of S
  IF t1 IN EXTENSION (S) AND t2 IN EXTENSION (S)
  THEN
      SET vars ← VARIABLES (s)
      REAL sum ← 0
      REAL sumwi ← 0
      FOR EACH ai IN vars
      DO
          IF t1.ai ≠ NULL AND t2.ai ≠ NULL
          THEN
              REAL wi ← WEIGHT (ai, s)
              sumwi ← sumwi + wi
              sum ← sum + (wi * FI (t1.ai, t2.ai))
          FI
      DONE
      IF ( sum ≤ ( limit  -  (1 - sumwi )) )
      THEN
          answer ← TRUE
      ELSE
          answer ← FALSE
      FI
  //if t1 and t2 are tuples, they must have equals objects playing the same roles
      IF IS_TUPLE(t1) AND answer = TRUE
      THEN
          SET roles ← ROLES (s)
          BOOLEAN failed  ← FALSE
          WHILE NOT failed AND HASNEXT (roles)
          DO
              ROLE rj  ← NEXT (roles)
              IF NOT EQUALS (t1.rj, t2.rj, limit)
              THEN
                  failed  ← TRUE
              FI
          DONE
          IF NOT failed
              THEN
                  answer ← TRUE
              ELSE
                  answer ← FALSE
              FI
      FI
  FI
END
```

Opérateur Contains

Algorithme 3 : Définition générale de l'opérateur Contains

```
ContainsV0
 IN:
        OBJECT set
        OBJECT subset
 OUT:
        BOOLEAN answer
 BEGIN
        //an optimistic approach: answer is initially true
        answer ←TRUE
        //get the tuples in which “set” or “subset” plays a role
        SET tuplesSet ← TUPLES(set)
        SET tuplesSubset ← TUPLES(subset)
        //Collect all tuples belonging to the graph defined by set
        FOR EACH t IN tuplesSet
        DO
          SET link ← OBJECTS (t)
          FOR EACH o IN link
          DO
            tuplesSet ←  UNION( tuplesSet, TUPLES(o))
          DONE
        DONE
        //Collect all tuples belonging to the graph defined by subset
        FOR EACH u IN tuplesSubset
        DO
          SET link ← OBJECTS (u)
          FOR EACH o IN link
          DO
            tuplesSubset ← UNION (tuplesSubset, TUPLES(o))
          DONE
        DONE
        FOR EACH u IN tuplesSubset
        DO //find in tuplesSet a tuple t corresponding (equals) to u
          TUPLE  t ← FIND_EQUAL (tuplesSet, u)
          IF t = NULL
          THEN //no corresponding tuple, answer false
            answer ← FALSE
          ELSE //t belongs to a match, eliminate it from the tuplesSet
            tuplesSet ← tuplesSet – { t }
          FI
        DONE
END
```

Algorithme 4 : Définition de l'opérateur Contains avec une « ignore list »

```
ContainsV1
 IN:
        OBJECT set
        OBJECT subset
        SET ignored
 OUT:
        BOOLEAN answer

 BEGIN
//an optimistic approach: answer is initially true
        answer ← TRUE
//first, we “enumerate” the arcs of the graph
//we collect  the tuples in which “set” or “subset” plays a role
        SET tuplesSet ← TUPLES(set)
        SET tuplesSubset ← TUPLES(subset)

//we collect all tuples belonging to the graph defined by set
        FOR EACH t IN tuplesSet
        DO
          SET link  ← OBJECTS (t)
          FOR EACH o IN link
          DO
```

```
            tuplesSet ← UNION (tuplesSet,  TUPLES(o))
          DONE
        DONE

//we collect all tuples belonging to the graph defined by subset
        FOR EACH u IN tuplesSubset
        DO
          SET link  ← OBJECTS (u)
          FOR EACH o IN link
          DO
            tuplesSubset ← UNION (tuplesSubset, TUPLES(o))
          DONE
        DONE

//we remove from the graph the tuples in the ignore list
        tuplesSet ←  tuplesSet- ignored
        tuplesSubset ← tuplesSubset - ignored
        BOOLEAN failed ← FALSE

WHILE NOT failed AND HASNEXT (tuplesSubset)
        DO
          TUPLE u ← NEXT (tuplesSubset)
//if u connect objects in the ignore list, we should ignore it
          SET linku  ← OBJECTS (u)
          SET intersectu ← INTERSECTION (linku, ignored)
          IF EMPTY (intersectu)       //no intersection with the ignore list
          THEN

//find in tuplesSet a tuple t that corresponds (equals) to u and that do not connect objects in the ignore list
                BOOLEAN found ← FALSE
                TUPLE t ← FIND_EQUALS (tuplesSet, u)
                WHILE NOT found  AND t ≠ NULL
                DO
//if t connect objects in the ignore list, we should ignore it and look for another tuple
                  SET linkt  ← OBJECTS (t)
                  SET intersectt ← INTERSECTION (linkt, ignored)
                  IF NOT EMPTY (intersectt)
                  THEN
//t connect objects in the ignore list, we should remove it from tuplesSet
                      tuplesSet ← tuplesSet – { t }
                  ELSE
//t matches u, we shouldn't use it anymore
                      found ← TRUE
                      tuplesSet ← tuplesSet – { t }
                  FI
                  t ← FIND_EQUALS (tuplesSet, u)
                DONE
                IF NOT found
                THEN
//no corresponding tuple, answer false, we may stop working
                  failed ← TRUE
                FI
          FI
        DONE
        IF failed
        THEN
          answer ← FALSE
        ELSE
          answer ← TRUE
        FI
END
```

Algorithme 5 : Définition de l'opérateur Contains sous une fonction de condition

```
ContainsV2
IN:
        OBJECT set
        OBJECT subset
        SET ignored
OUT:
        BOOLEAN answer
BEGIN
//an optimistic approach: answer is initially true
        answer ← TRUE
```

```
//first, we “enumerate” the arcs of the graph and collect  the tuples in which “set” or “subset” plays a role
        SET tuplesSet ← TUPLES(set)
        SET tuplesSubset ← TUPLES(subset)

//we collect all tuples belonging to the graph defined by set
        FOR EACH t IN tuplesSet
        DO
          SET link  ← OBJECTS (t)

//if t doesn't satisfy the FC condition, it should belong to ignored set
          IF FC (t) < 1
          THEN
            ignored ← UNION (ignored, { t } )
          FI

          FOR EACH o IN link
          DO
            SET tuplesO ←  TUPLES(o)
            tuplesSet ← UNION (tuplesSet,  tuplesO)
//if o doesn't satisfy the FC condition, it should belong to ignored set
            IF FC (o) < 1
            THEN
                ignored ← UNION (ignored,  { o, t } )
            FI
          DONE
        DONE

//we collect all tuples belonging to the graph defined by subset
        FOR EACH u IN tuplesSubset
        DO
          SET link  ← OBJECTS (u)
//if u  doesn't satisfy the FC condition, it should belong to ignored set
          IF FC (u) < 1
          THEN
            ignored ← UNION (ignored, { u )
          FI
          FOR EACH o IN link
          DO
            SET tuplesO ←  TUPLES(o)
            tuplesSet ← UNION (tuplesSet,  tuplesO)
//if o doesn't satisfy the FC condition, it should belong to ignored set
            IF FC (o) < 1
            THEN
                ignored ← UNION (ignored,  { o, u } )
            FI
          DONE
        DONE
//we remove from the graph the tuples in the ignore list
        tuplesSet ←  tuplesSet- ignored
        tuplesSubset ← tuplesSubset - ignored

        BOOLEAN failed ← FALSE
        WHILE NOT failed AND HASNEXT (tuplesSubset)
        DO
          TUPLE u ← NEXT (tuplesSubset)
//if u connect objects in the ignore list, we should ignore it
          SET linku  ← OBJECTS (u)
          SET intersectu ← INTERSECTION (linku, ignored)
          IF EMPTY (intersect)
          THEN
//find in tuplesSet a tuple t that corresponds (equals) to u and that do not connect objects in the ignore list
            BOOLEAN found ← FALSE
            TUPLE t ← FIND_EQUALS (tuplesSet, u)
            WHILE NOT found AND t ≠ NULL
            DO
                SET linkt  ← OBJECTS (t)
                SET intersectt ← INTERSECTION (linkt, ignored)
                IF NOT EMPTY (intersectt)
                THEN  // t connect objects in the ignore list,
//we should ignore it and remove it from the graph
                  tuplesSet ← tuplesSet – { t }
                ELSE // t matches u, we don't use it anymore
                  found ← TRUE
                  tuplesSet ← tuplesSet – { t }
                FI
                t ← FIND_EQUALS (tuplesSet, U)
```

```
                DONE

                IF NOT found
                    THEN
//no corresponding tuple, answer false, we may stop working
                        failed ← TRUE
                    FI
            FI
        DONE

        IF failed
        THEN
            answer ← FALSE
        ELSE
            answer ← TRUE
        FI

END
```

Mesures de Similarité

Algorithme 6 : Définition de l'opérateur SimO

```
SimO
IN:
    OBJECT o1
    OBJECT o2
OUT:
    REAL simo
BEGIN
    simo ← 0
    //o1 and o2 have a common structure?
    STRUCTURE  s ← COMMON_STRUCTURE (o1, o2)
    IF s ≠ NULL
    THEN
        REAL sum ← 0
        SET vars ← VARIABLES (s)

        FOR EACH ai IN vars
        DO
            IF t1.ai ≠ NULL AND t2.ai ≠ NULL
            THEN
                REAL wi ← WEIGHT (ai, s)
                sum ← sum + ( FI (o1.ai, o2.ai) * wi )
            FI
        DONE
        simo ← sum * FP (o1, o2)
    FI
END
```

Algorithme 7 : Définition de l'opérateur SimT

```
SimT
IN:
    TUPLE t1
    TUPLE t2
OUT:
    REAL simt
BEGIN
    simt ← 0
    //t1 and t2 have a common structure?
    STRUCTURE  s ← COMMON_STRUCTURE (t1, t2)
    IF s ≠ NULL
    THEN
        REAL sum1 ← 0
        REAL sum2 ← 0
        //first, we compare the variables
        SET vars ← VARIABLES (s)
```

```
        FOR EACH aj IN vars
        DO
            IF t1.aj ≠ NULL AND t2.aj ≠ NULL
            THEN
                REAL wj ← WEIGHT (aj, s)
                sum1 ← sum1 + ( FI (t1.aj, t2.aj) * wj )
            FI
        DONE
        //then, we compare the link
        SET roles ← ROLES (s)
        FOR EACH ri IN roles
        DO
            IF t1.ri ≠ NULL AND t2.ri ≠ NULL
            THEN
                REAL wi ← WEIGHT (ri, s)
                sum2 ← sum2 + ( SIMO (t1.ri, t2.ri) * wi )
            FI
        DONE
    sImt ← (sum1 + sum2) / 2
    FI
END
```

Algorithme 8 : Définition de l'opérateur Sim

```
Sim
 IN:
        OBJECT o1
        OBJECT o2
 OUT:
        REAL sim

 BEGIN
        SET To1 ← TUPLES(o1)
        SET Oo1 ← { o1 }
        INTEGER XT ← 0
        INTEGER XO ← 0

        //first, we  capture the element of the graph g(o1) = ( O(o1), T(o1))
        FOR EACH t IN To1
        DO
          SET link  ← OBJECTS (t)
          Oo1 ← UNION (Oo1, link)
          FOR EACH o IN link
          DO
            To1 ← UNION (To1, TUPLES (o))
          DONE
        DONE
        //we calculate Xo and XT
        SET To2 ← TUPLES(o2)
        FOR EACH u IN To2
        DO
          SET link  ← OBJECTS (u)
          IF FIND_EQUALS (To1, u) ≠ NULL
          THEN
            // the tuple u has an equal tuples in T(o1)
            XT ← XT + 1
          FI
          FOR EACH o IN link
          DO
            IF FIND_EQUALS (Oo1, o) ≠ NULL
            THEN
                // the object o has an equal object in O(o1)
                XO ← XO + 1
            FI
          To2 ← UNION (To2, TUPLES (o))
          DONE
        DONE

        //then we calculate Sim
        sim ← ( XO + XT) / ( SIZE (Oo1) + SIZE (To1))
 END
```

Annexe II – Algorithmes

Chapitre 7

Algorithme 1 : Première étape du processus de filtrage

```
Selection Profil
IN:
    ElementContexte cible
OUT:
    SET selection

BEGIN
    //  on récupère le contexte courant d'utilisation
    DescriptionContexte Cu ← CONTEXT (cible)
    // on récupère les profils disponibles
    SET candidats ← PROFILES (cible)

    FOR EACH Pi IN candidats
    DO
        // on récupère les contextes d'application associés à chaque profil
        SET contextApplication ← CONTEXT_APPLICATION (Pi)
        FOR EACH Cp IN contextApplication
        DO
            //si le contexte courant Cu contient au moins un contexte d'application Cp, le profil est sélectionné
            IF CONTAINS (Cu, Cp)
            THEN
                selection ← UNION (selection, Pi)
                BREAK
            FI
        DONE
    DONE
END
```

Algorithme 2 : Seconde étape du processus de filtrage

```
Filtrage Evenements
IN:
    ElementContexte cible
    SET profils
OUT:
    LIST stratificationsInt
    LIST stratificationsExt
    ListeEvenements evenements
BEGIN
    //  on récupère le contexte courant d'utilisation
    DescriptionContexte Cu ← CONTEXT (cible)
    // on met en ordre les profils
    SET profilsOrdonnes ← ORDER (profils, Cu)
    // on applique les profils en ordre
    FOR EACH Pi IN profilsOrdonnes
    DO
        LIST preselection ← { }
        LIST evtsOrdonnes ← { }
        // abonnement
        // on récupère les événements dont le profil est abonné
        SET classesEvts ← SIGNUP (Pi)
        FOR EACH Ei IN classesEvts
        DO
            preselection ← UNION (preselection, EVENTS (Ei))
        DONE
        //on enlève les événements qui sont déjà dans la liste d'événements
        LIST commun ←  INTERSECTION (preselection, evenements)
        preselection ← preselection – commun
        // stratifications
        // on applique les stratifications, ce qui va mettre en ordre les événements
        LIST stratsExtPi ← STRATIFICATIONS_EXT (Pi)
        LIST stratsIntPi ← STRATIFICATIONS_INT (Pi)
        FOR EACH Sj IN stratsExtPi
        DO
            ordonnes ← ADD (APPLY (preselection, Sj))
```

```
        DONE
        FOR EACH Si IN stratsIntPi
        DO
            ordonnes ← APPLY (ordonnes, Si)
        DONE
        stratificationsExt ← ADD (stratsExtPi)
        stratificationsInt ← ADD (stratsIntPi)
        // conditions contextuelles
        SET condProduction ← CONDITION_PRODUCTION (Pi)
        SET elementsCond ← CONDITION_ELEMENTS (Pi)
        // on vérifie les condition de production
        IF NOT EMPTY (condProduction)
        THEN
            FOR EACH evt IN ordonnes
            DO
                DescriptionContexte Ce ← CONTEXTE (evt)
                IF NOT MATCH_CONTAINS (Ce, condProduction)
                THEN
                    ordonnes ← ordonnes – {evt}
                FI
            DONE
        FI
        // on vérifie les éléments condition
        IF NOT EMPTY (elementsCond)
        THEN
            FOR EACH evt IN ordonnes
            DO
                SET elemConcernes ← CONCERN (evt)
                REAL seuil ← LIMIT (Pi)
                IF NOT MATCH_SIMO (elemConcernes, elementsCond)
                THEN
                    ordonnes ← ordonnes – {evt}
                FI
            DONE
        FI
        // on additionne un niveau à la liste d'événements
        evenements ← ADD (ordonnes)
    DONE
END


Match_Contains
IN:
    DescriptionContexte Ce
    SET condProduction

OUT:
    BOOLEAN match

BEGIN
    match ← FALSE

    FOR EACH Cp IN condProduction
    DO
        // si au moins un correspond, match est vrai
        IF CONTAINS (Ce, Cp)
        THEN
            match ← TRUE
            BREAK
            FI
    DONE
END


Match_SimO
IN:
    SET elemConcerne
    SET elemCondition
    REAL seuil
OUT:
    BOOLEAN match

BEGIN
    match ← TRUE

    FOR EACH Ep IN elemCondition
    DO
```

```
        // si aucun élément correspond, match est false
        BOOLEAN found
        FOR EACH Ee IN elemConcerne
        DO
            IF SIMO (Ep, Ee) ≥ seuil
            THEN
                found ← TRUE
                BREAK
            FI
        DONE

        IF NOT found
        THEN
            match ← FALSE
            BREAK
        FI
    DONE
END
```

Annexe III - XML Schéma XAROM

schema location: **Z:\work\XArom\src\schema\xarom\XArom.xsd**

Elements Complex types
KnowledgeBase **tKnowledgeBase**
schema location: **Z:\work\XArom\src\schema\xarom\Documentation.xsd**

Elements Complex types
Documentation **tDocumentable**
schema location: **Z:\work\XArom\src\schema\xarom\Structure.xsd**

Elements Complex types Simple types Attr. groups
Structure **tStructure** **tSuper** **aSuper**
schema location: **Z:\work\XArom\src\schema\xarom\Class.xsd**

Elements Complex types
Class **tClass**
schema location: **Z:\work\XArom\src\schema\xarom\Association.xsd**

Elements Complex types
Association **tAssociation**
schema location: **Z:\work\XArom\src\schema\xarom\Type.xsd**

Simple types Attr. groups
tDocumentationType **aBoundaryType**
tType **aRoleType**
tTypeByElement **aVariableDomainType**
tTypeCardinality **aVariableType**
tTypeHolder
schema location: **Z:\work\XArom\src\schema\xarom\Variable.xsd**

Complex types
tInstanceVariable
tStructureVariable
schema location: **Z:\work\XArom\src\schema\xarom\Domain.xsd**

Complex types Simple types
tRoleDomain **tBoundary**
tVariableDomain
schema location: **Z:\work\XArom\src\schema\xarom\Instance.xsd**

Elements Complex types
Instance **tInstance**
schema location: **Z:\work\XArom\src\schema\xarom\Object.xsd**

Elements Complex types
Object **tObject**
schema location: **Z:\work\XArom\src\schema\xarom\Tuple.xsd**

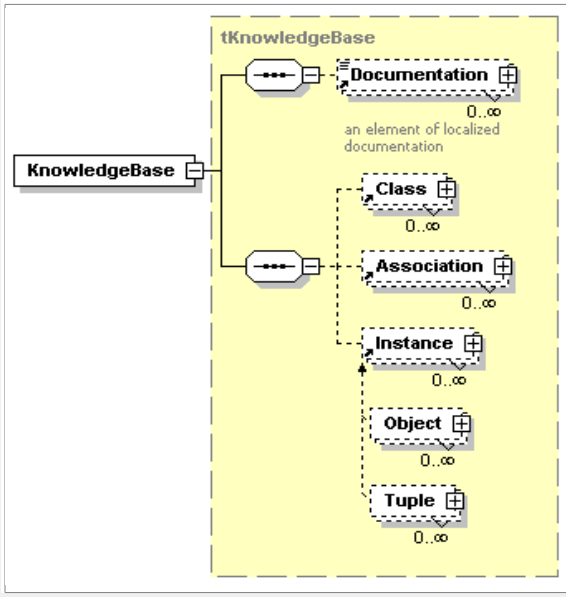
Elements Complex types
Tuple **tTuple**
schema location: **Z:\work\XArom\src\schema\xarom\Name.xsd**

types Attr. groups
tName **aName**
schema location: **Z:\work\XArom\src\schema\xarom\Role.xsd**

Complex types
tInstanceRole
tStructureRole
schema location: **Z:\work\XArom\src\schema\xarom\Attachment.xsd**

Attr. groups
aAttachment
element **KnowledgeBase**

diagram



type **tKnowledgeBase**

children **Documentation Class Association Instance**

attributes

Name	Type	Use	Default	Fixed	Annotation
name	tName	required			

identity constraints

	Name	Refer	Selector	Field(s)
unique	uStructure.name		.//Class|.//Association	@name
unique	uClass.name		.//Class	@name
unique	uInstance.name		.//Object	@name
keyref	krInstance.isa	uStructure.name	.//Object|.//Tuple	@isa
keyref	krStructure.super	uStructure.name	.//Class|.//Association	@super
keyref	krRole.class.name	uClass.name	.//Association/Role	@class
key	kAssociation.name		./Association	@name
keyref	krTuple.isa	kAssociation.name	./Tuple	@isa
keyref	krAssociation.super	kAssociation.name	.//Association	@super

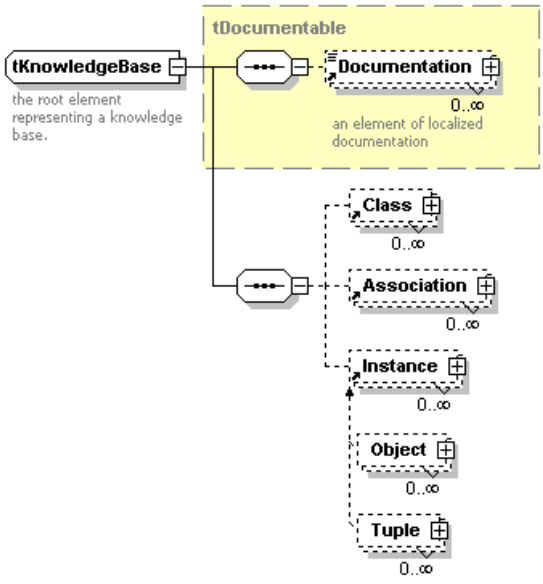
source

```
<xs:element name="KnowledgeBase" type="tKnowledgeBase">
  <xs:unique name="uStructure.name">
    <xs:annotation>
      <xs:documentation>unicity of Structure name</xs:documentation>
    </xs:annotation>
    <xs:selector xpath=".//Class|.//Association"/>
    <xs:field xpath="@name"/>
  </xs:unique>
  <xs:unique name="uClass.name">
    <xs:annotation>
      <xs:documentation>unicity of Structure name</xs:documentation>
    </xs:annotation>
    <xs:selector xpath=".//Class"/>
    <xs:field xpath="@name"/>
  </xs:unique>
  <xs:unique name="uInstance.name">
    <xs:annotation>
      <xs:documentation>unicity of Instance name</xs:documentation>
    </xs:annotation>
    <xs:selector xpath=".//Object"/>
    <xs:field xpath="@name"/>
  </xs:unique>
  <xs:keyref name="krInstance.isa" refer="uStructure.name">
    <xs:selector xpath=".//Object|.//Tuple"/>
```

```
      <xs:field  xpath="@isa"/>
   </xs:keyref>
   <xs:keyref  name="krStructure.super"  refer="uStructure.name">
      <xs:selector  xpath=".//Class|.//Association"/>
      <xs:field  xpath="@super"/>
   </xs:keyref>
   <xs:keyref  name="krRole.class.name"  refer="uClass.name">
      <xs:selector  xpath=".//Association/Role"/>
      <xs:field  xpath="@class"/>
   </xs:keyref>
   <xs:key  name="kAssociation.name">
      <xs:selector  xpath="./Association"/>
      <xs:field  xpath="@name"/>
   </xs:key>
   <xs:keyref  name="krTuple.isa"  refer="kAssociation.name">
      <xs:selector  xpath="./Tuple"/>
      <xs:field  xpath="@isa"/>
   </xs:keyref>
   <xs:keyref  name="krAssociation.super"  refer="kAssociation.name">
      <xs:selector  xpath=".//Association"/>
      <xs:field  xpath="@super"/>
   </xs:keyref>
   <!--xs:key  name="kKb.name">
         <xs:selector  xpath="KnowledgeBase"/>
         <xs:field  xpath="@name"/>
      </xs:key-->
   <!--xs:key  name="kAssociationVariable.name">
            <xs:selector  xpath="./Association/Variable"/>
            <xs:field  xpath="@name"/>
      </xs:key>
      <xs:keyref  name="krTupleVariable.name"  refer="kAssociationVariable.name">
            <xs:selector  xpath=".//Tuple/Variable"/>
            <xs:field  xpath="@name"/>
    </xs:keyref-->
   <!--  meta model constraints -->
   <!--  -->
   <!--  an Object wich is a Class instance must have a isa attribute wich value matches an existing Class -->
   <!--  a reference to a super Class must be an existing Class -->
    <!--  an Tuple wich is an Association instance must have a isa attribute wich value matches an existing
Association -->
   <!--  a reference to a super Association must be an existing Association -->
</xs:element>
```

complexType **tKnowledgeBase**

diagram



type extension of **tDocumentable**

children	**Documentation** **Class** **Association** **Instance**
used by	element **KnowledgeBase**

attributes	Name	Type	Use	Default	Fixed	Annotation
	name	tName	required			

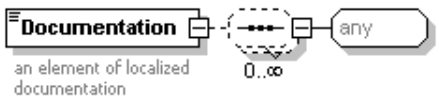
annotation	documentation	the root element representing a knowledge base.

source

```
<xs:complexType name="tKnowledgeBase">
  <xs:annotation>
    <xs:documentation>the root element representing a knowledge base.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="tDocumentable">
      <xs:sequence>
        <xs:element ref="Class" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>
                placeholder for classes definition.
              </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element ref="Association" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>
                placeholder for Association definition.
              </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element ref="Instance" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>
              placeholder for instance definition.
              </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attributeGroup ref="aName">
        <xs:annotation>
          <xs:documentation>
              naming attributes for the KB.
            </xs:documentation>
        </xs:annotation>
      </xs:attributeGroup>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

element **Documentation**

diagram



used by	complexType	**tDocumentable**

attributes	Name	Type	Use	Default	Fixed	Annotation
	type	tDocumentationType		text		
	lang	xs:language				documentation: the XML 'lang' attribute.

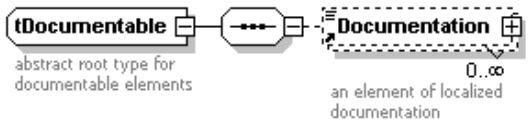
annotation	documentation	an element of localized documentation

source

```
<xs:element name="Documentation">
  <xs:annotation>
    <xs:documentation>an element of localized documentation</xs:documentation>
  </xs:annotation>
  <xs:complexType mixed="true">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:any processContents="lax"/>
    </xs:sequence>
    <xs:attribute name="type" type="tDocumentationType" default="text"/>
    <xs:attribute name="lang" type="xs:language">
      <xs:annotation>
        <xs:documentation>the XML 'lang' attribute.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

complexType **tDocumentable**

diagram



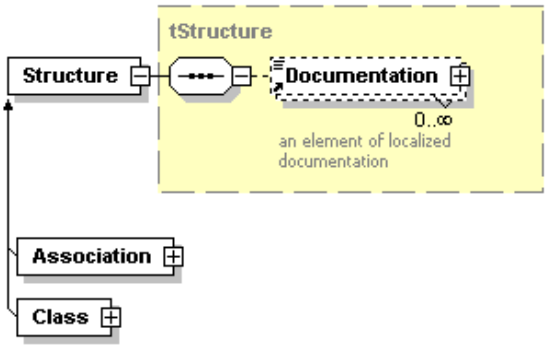
children	**Documentation**		
used by	complexTypes	**tInstance tKnowledgeBase tRoleDomain tStructure tStructureRole tStructureVariable tVariableDomain**	
annotation	documentation	abstract root type for documentable elements	

source

```
<xs:complexType name="tDocumentable" abstract="true">
  <xs:annotation>
    <xs:documentation>abstract root type for documentable elements</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element ref="Documentation" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

element **Structure**

diagram

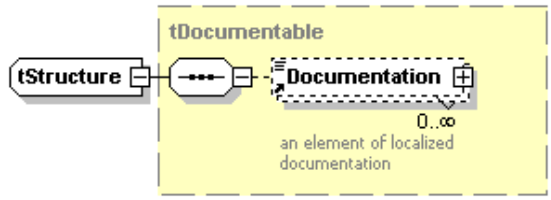


type	**tStructure**					
children	**Documentation**					
attributes	Name	Type	Use	Default	Fixed	Annotation
	name	tName	required			
	super	tSuper	optional			

source

```
<xs:element name="Structure" type="tStructure"/>
```

complexType **tStructure**

diagram	
type	extension of **tDocumentable**
children	**Documentation**
used by	element: **Structure** complexTypes: **tAssociation** **tClass**

attributes

Name	Type	Use	Default	Fixed	Annotation
name	tName	required			
super	tSuper	optional			

source

```
<xs:complexType name="tStructure" abstract="true">
  <xs:complexContent>
    <xs:extension base="tDocumentable">
      <xs:attributeGroup ref="aName"/>
      <xs:attributeGroup ref="aSuper"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

simpleType **tSuper**

type	**xs:token**
used by	attribute: **aSuper/@super**

source

```
<xs:simpleType name="tSuper">
  <xs:restriction base="xs:token"/>
</xs:simpleType>
```

attributeGroup **aSuper**

used by	complexType: **tStructure**

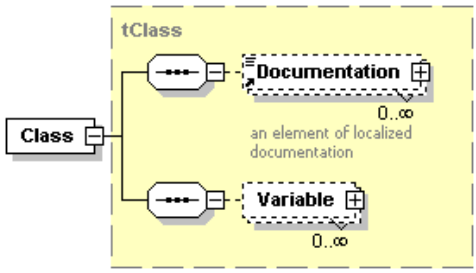
attributes

Name	Type	Use	Default	Fixed	Annotation
super	tSuper	optional			

source

```
<xs:attributeGroup name="aSuper">
  <xs:attribute name="super" type="tSuper" use="optional"/>
</xs:attributeGroup>
```

element **Class**

diagram	
type	**tClass**

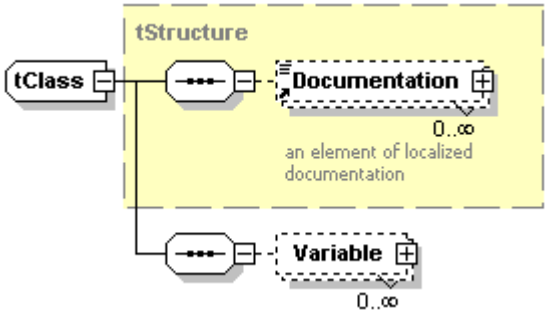
children	**Documentation Variable**					
used by	complexType	**tKnowledgeBase**				
attributes	Name	Type	Use	Default	Fixed	Annotation
	name	tName	required			
	super	tSuper	optional			
identity constraints		Name	Refer	Selector	Field(s)	
	unique	uClass.Variable.name		Variable	@name	

source

```
<xs:element name="Class" type="tClass" substitutionGroup="Structure">
  <xs:unique name="uClass.Variable.name">
    <xs:selector xpath="Variable"/>
    <xs:field xpath="@name"/>
  </xs:unique>
</xs:element>
```

complexType **tClass**

diagram



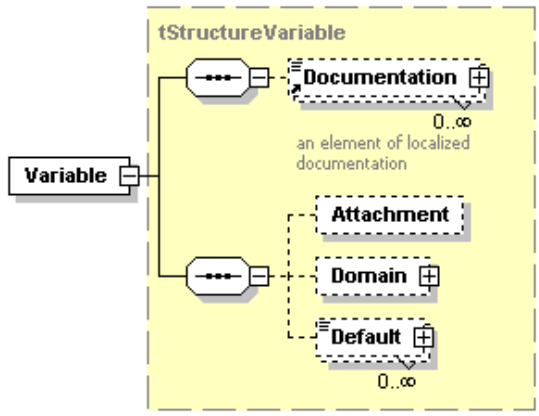
type	extension of **tStructure**					
children	**Documentation Variable**					
used by	element	**Class**				
attributes	Name	Type	Use	Default	Fixed	Annotation
	name	tName	required			
	super	tSuper	optional			

source

```
<xs:complexType name="tClass">
  <xs:complexContent>
    <xs:extension base="tStructure">
      <xs:sequence>
        <xs:element name="Variable" type="tStructureVariable" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

element **tClass/Variable**

diagram



type **tStructureVariable**

children **Documentation Attachment Domain Default**

attributes

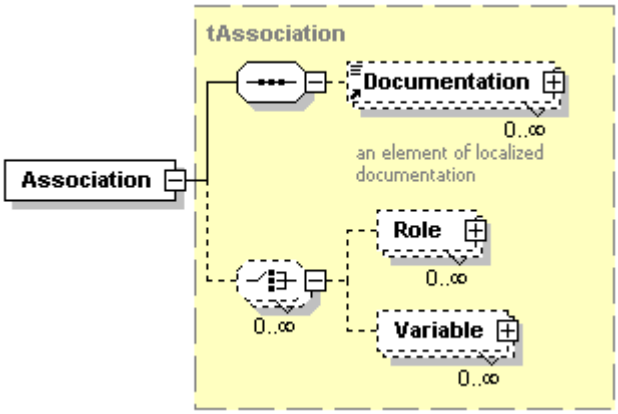
Name	Type	Use	Default	Fixed	Annotation
name	tName	required			
type	tType	optional			documentation: type is optional to allow redefinition of a variable in a derived Class or Association without having to specify the type again
holder	tTypeHolder	optional	scalar		
min	tTypeCardinality	optional			
max	tTypeCardinality	optional			

source

```
<xs:element name="Variable" type="tStructureVariable" minOccurs="0" maxOccurs="unbounded"/>
```

element **Association**

diagram



type **tAssociation**

children **Documentation Role Variable**

used by

complexType	tKnowledgeBase

attributes

Name	Type	Use	Default	Fixed	Annotation
name	tName	required			
super	tSuper	optional			

identity constraints

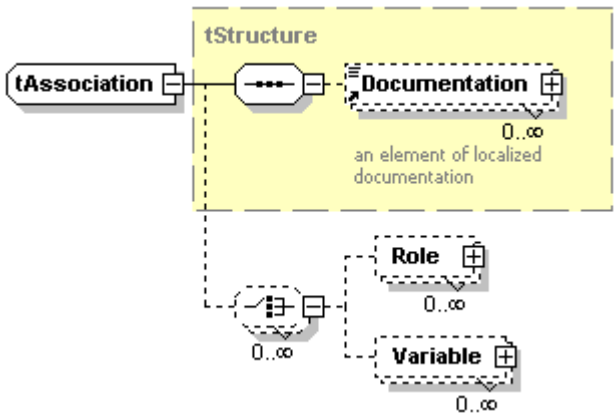
	Name	Refer	Selector	Field(s)
unique	uAssociation.Variable.name		./Variable	@name

source

```
<xs:element name="Association" type="tAssociation" substitutionGroup="Structure">
  <xs:unique name="uAssociation.Variable.name">
    <xs:annotation>
      <xs:documentation>unicity of Variable name within Structure</xs:documentation>
    </xs:annotation>
    <xs:selector xpath="./Variable"/>
    <xs:field xpath="@name"/>
  </xs:unique>
</xs:element>
```

complexType **tAssociation**

diagram



type: extension of **tStructure**

children: **Documentation** **Role** **Variable**

used by: element **Association**

attributes

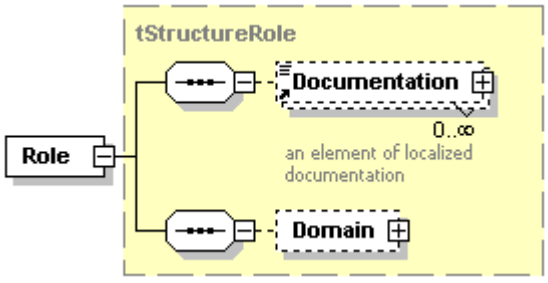
Name	Type	Use	Default	Fixed	Annotation
name	tName	required			
super	tSuper	optional			

source

```
<xs:complexType name="tAssociation">
  <xs:complexContent>
    <xs:extension base="tStructure">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Role" type="tStructureRole" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Variable" type="tStructureVariable" minOccurs="0" maxOccurs="unbounded"/>
        <!-- Role elements are optional since they can be inherited from a mother Association -->
      </xs:choice>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

element **tAssociation/Role**

diagram



type: **tStructureRole**

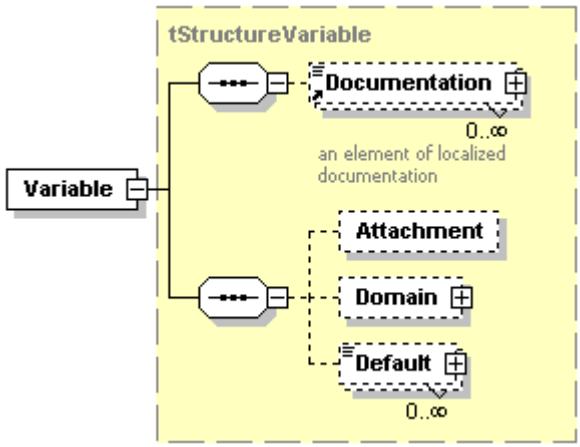
children: **Documentation** **Domain**

attributes	Name	Type	Use	Default	Fixed	Annotation
	name	tName	required			
	class	xs:string	required			
	min	tTypeCardinality	optional			
	max	tTypeCardinality	optional			

source
```
<xs:element name="Role" type="tStructureRole" minOccurs="0" maxOccurs="unbounded"/>
```

element **tAssociation/Variable**

diagram



type **tStructureVariable**

children **Documentation Attachment Domain Default**

attributes	Name	Type	Use	Default	Fixed	Annotation
	name	tName	required			
	type	tType	optional			documentation: type is optional to allow redefinition of a variable in a derived Class or Association without having to specify the type again
	holder	tTypeHolder	optional	scalar		
	min	tTypeCardinality	optional			
	max	tTypeCardinality	optional			

source
```
<xs:element name="Variable" type="tStructureVariable" minOccurs="0" maxOccurs="unbounded"/>
```

simpleType **tDocumentationType**

type restriction of **xs:token**

used by attribute **Documentation/@type**

facets
- enumeration: text
- enumeration: url

source
```
<xs:simpleType name="tDocumentationType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="text"/>
    <xs:enumeration value="url"/>
  </xs:restriction>
</xs:simpleType>
```

simpleType **tType**

type	union of (restriction of **xs:token**, restriction of **xs:string**)
used by	attribute **aVariableType/@type**

```
<xs:simpleType name="tType">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="boolean"/>
        <xs:enumeration value="integer"/>
        <xs:enumeration value="float"/>
        <xs:enumeration value="string"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="ctype:.+"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

simpleType **tTypeByElement**

type	**xs:boolean**
used by	attribute **aVariableDomainType/@byElement**

```
<xs:simpleType name="tTypeByElement">
  <xs:restriction base="xs:boolean"/>
</xs:simpleType>
```

simpleType **tTypeCardinality**

type	union of (**xs:nonNegativeInteger**, restriction of **xs:token**)
used by	attributes **aVariableType/@max aRoleType/@max aVariableType/@min aRoleType/@min**

```
<xs:simpleType name="tTypeCardinality">
  <xs:union memberTypes="xs:nonNegativeInteger">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="*"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

simpleType **tTypeHolder**

type	restriction of **xs:token**	
used by	attribute	**aVariableType/@holder**
facets	enumeration	scalar
	enumeration	set
	enumeration	list

```
<xs:simpleType name="tTypeHolder">
  <xs:restriction base="xs:token">
    <xs:enumeration value="scalar"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="list"/>
  </xs:restriction>
</xs:simpleType>
```

attributeGroup **aBoundaryType**

used by	elements	**tVariableDomain/lower** **tVariableDomain/upper**					
attributes	Name	Type	Use	Default	Fixed	Annotation	
	closed	xs:boolean	optional	true			
annotation	documentation	type attributes for interval boundary type.					

source

```
<xs:attributeGroup name="aBoundaryType">
  <xs:annotation>
    <xs:documentation>type attributes for interval boundary type.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="closed" type="xs:boolean" use="optional" default="true"/>
</xs:attributeGroup>
```

attributeGroup **aRoleType**

used by	complexType	**tStructureRole**					
attributes	Name	Type	Use	Default	Fixed	Annotation	
	class	xs:string	required				
	min	tTypeCardinality	optional				
	max	tTypeCardinality	optional				
annotation	documentation	type attributes for Roles.					

source

```
<xs:attributeGroup name="aRoleType">
  <xs:annotation>
    <xs:documentation>type attributes for Roles.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="class" type="xs:string" use="required"/>
  <xs:attribute name="min" type="tTypeCardinality" use="optional"/>
  <xs:attribute name="max" type="tTypeCardinality" use="optional"/>
</xs:attributeGroup>
```

attributeGroup **aVariableDomainType**

used by	complexType	**tVariableDomain**					
attributes	Name	Type	Use	Default	Fixed	Annotation	
	byElement	tTypeByElement	optional	true			
annotation	documentation	type attributes for Domains.					

source

```
<xs:attributeGroup name="aVariableDomainType">
  <xs:annotation>
    <xs:documentation>type attributes for Domains.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="byElement" type="tTypeByElement" use="optional" default="true"/>
</xs:attributeGroup>
```

attributeGroup **aVariableType**

used by	complexType	**tStructureVariable**					
attributes	Name	Type	Use	Default	Fixed	Annotation	

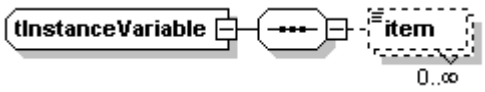
	type	tType	optional			documentation	type is optional to allow redefinition of a variable in a derived Class or Association without having to specify the type again
	holder	tTypeHolder	optional	scalar			
	min	tTypeCardinality	optional				
	max	tTypeCardinality	optional				
annotation	documentation	type attributes for Variables.					

source

```
<xs:attributeGroup name="aVariableType">
   <xs:annotation>
      <xs:documentation>type attributes for Variables.</xs:documentation>
   </xs:annotation>
   <xs:attribute name="type" type="tType" use="optional">
      <xs:annotation>
         <xs:documentation>type is optional to allow redefinition of a variable in a derived Class or Association
without having to specify the type again</xs:documentation>
      </xs:annotation>
   </xs:attribute>
   <xs:attribute name="holder" type="tTypeHolder" use="optional" default="scalar"/>
   <xs:attribute name="min" type="tTypeCardinality" use="optional"/>
   <xs:attribute name="max" type="tTypeCardinality" use="optional"/>
</xs:attributeGroup>
```

complexType **tInstanceVariable**

diagram



children **item**

used by

element	tInstance/Variable

attributes

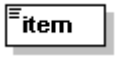
Name	Type	Use	Default	Fixed	Annotation
name	tName	required			

source

```
<xs:complexType name="tInstanceVariable" mixed="true">
   <xs:sequence>
      <xs:element name="item" type="xs:anySimpleType" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attributeGroup ref="aName"/>
</xs:complexType>
```

element **tInstanceVariable/item**

diagram



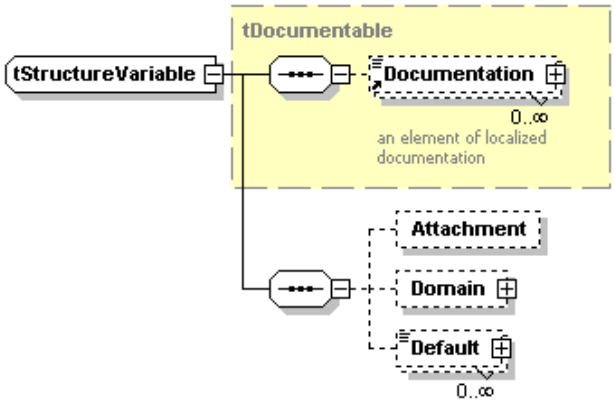
type **xs:anySimpleType**

source

```
<xs:element name="item" type="xs:anySimpleType" minOccurs="0" maxOccurs="unbounded"/>
```

ComplexType **tStructureVariable**

diagram



type: extension of **tDocumentable**

children: **Documentation Attachment Domain Default**

used by — elements: **tClass/Variable**, **tAssociation/Variable**

attributes

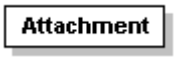
Name	Type	Use	Default	Fixed	Annotation
name	tName	required			
type	tType	optional			documentation: type is optional to allow redefinition of a variable in a derived Class or Association without having to specify the type again
holder	tTypeHolder	optional	scalar		
min	tTypeCardinality	optional			
max	tTypeCardinality	optional			

source

```
<xs:complexType name="tStructureVariable">
  <xs:complexContent>
    <xs:extension base="tDocumentable">
      <xs:sequence>
        <xs:element name="Attachment" minOccurs="0">
          <xs:complexType>
            <xs:attributeGroup ref="aAttachment"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="Domain" type="tVariableDomain" minOccurs="0"/>
        <xs:element name="Default" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType mixed="true">
            <xs:sequence>
                                    <xs:element name="item" type="xs:anySimpleType" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attributeGroup ref="aName"/>
      <xs:attributeGroup ref="aVariableType"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

element **tStructureVariable/Attachment**

diagram



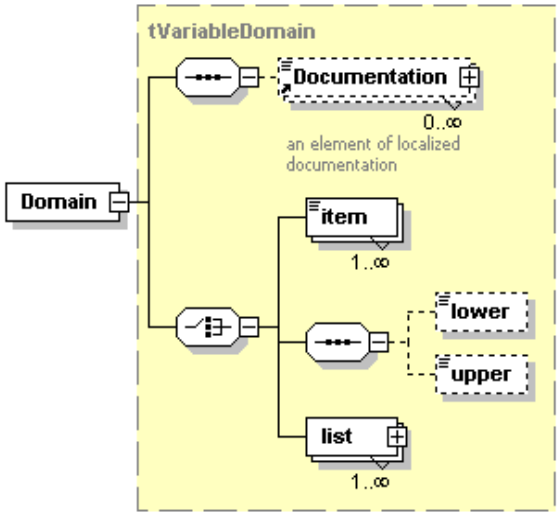
attributes	Name	Type	Use	Default	Fixed	Annotation
	class	xs:string				
	method	xs:string				

source

```
<xs:element name="Attachment" minOccurs="0">
  <xs:complexType>
    <xs:attributeGroup ref="aAttachment"/>
  </xs:complexType>
</xs:element>
```

element **tStructureVariable/Domain**

diagram



type **tVariableDomain**

children **Documentation item lower upper list**

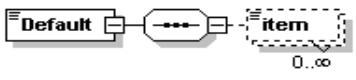
attributes	Name	Type	Use	Default	Fixed	Annotation
	byElement	tTypeByElement	optional	true		

source

```
<xs:element name="Domain" type="tVariableDomain" minOccurs="0"/>
```

element **tStructureVariable/Default**

diagram



children **item**

source

```
<xs:element name="Default" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="item" type="xs:anySimpleType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **tStructureVariable/Default/item**

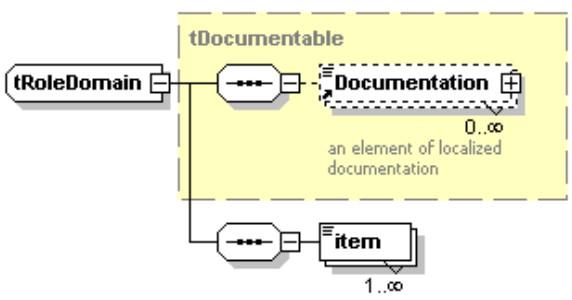
diagram

type **xs:anySimpleType**

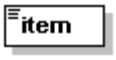
source

```
<xs:element name="item" type="xs:anySimpleType" minOccurs="0" maxOccurs="unbounded"/>
```

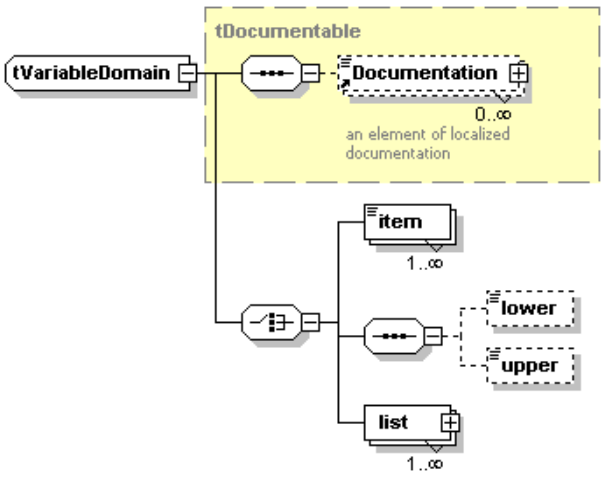
complexType **tRoleDomain**

diagram	
type	extension of **tDocumentable**
children	**Documentation item**
used by	element **tStructureRole/Domain**
source	<xs:complexType name="tRoleDomain"> <xs:complexContent> <xs:extension base="tDocumentable"> <xs:sequence> <xs:element name="item" type="xs:string" maxOccurs="unbounded"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

element **tRoleDomain/item**

diagram	
type	**xs:string**
source	<xs:element name="item" type="xs:string" maxOccurs="unbounded"/>

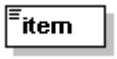
complexType **tVariableDomain**

diagram	
type	extension of **tDocumentable**
children	**Documentation item lower upper list**
used by	element **tStructureVariable/Domain**

attributes	Name	Type	Use	Default	Fixed	Annotation
	byElement	tTypeByElement	optional	true		

source
```
<xs:complexType name="tVariableDomain">
  <xs:complexContent>
    <xs:extension base="tDocumentable">
      <xs:choice>
        <xs:element name="item" type="xs:anySimpleType" maxOccurs="unbounded"/>
        <xs:sequence>
          <xs:element name="lower" minOccurs="0">
            <xs:complexType mixed="true">
              <xs:simpleContent>
                <xs:extension base="tBoundary">
                  <xs:attributeGroup ref="aBoundaryType"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
          <xs:element name="upper" minOccurs="0">
            <xs:complexType mixed="true">
              <xs:simpleContent>
                <xs:extension base="tBoundary">
                  <xs:attributeGroup ref="aBoundaryType"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:element name="list" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="item" type="xs:anySimpleType" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
      <xs:attributeGroup ref="aVariableDomainType"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

element **tVariableDomain/item**

diagram 

type **xs:anySimpleType**

source
```
<xs:element name="item" type="xs:anySimpleType" maxOccurs="unbounded"/>
```

element **tVariableDomain/lower**

diagram 

type extension of **tBoundary**

attributes

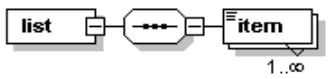
Name	Type	Use	Default	Fixed	Annotation
closed	xs:boolean	optional	true		

source
```
<xs:element name="lower" minOccurs="0">
  <xs:complexType mixed="true">
    <xs:simpleContent>
      <xs:extension base="tBoundary">
        <xs:attributeGroup ref="aBoundaryType"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

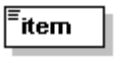
element **tVariableDomain/upper**

diagram	
type	extension of **tBoundary**
attributes	Name: closed; Type: xs:boolean; Use: optional; Default: true; Fixed: ; Annotation:
source	<xs:element name="upper" minOccurs="0"> <xs:complexType mixed="true"> <xs:simpleContent> <xs:extension base="tBoundary"> <xs:attributeGroup ref="aBoundaryType"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>

element **tVariableDomain/list**

diagram	
children	**item**
source	<xs:element name="list" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="item" type="xs:anySimpleType" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> </xs:element>

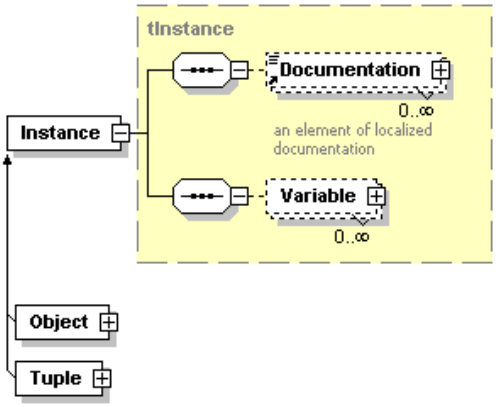
element **tVariableDomain/list/item**

diagram	
type	**xs:anySimpleType**
source	<xs:element name="item" type="xs:anySimpleType" maxOccurs="unbounded"/>

simpleType **tBoundary**

type	union of (**xs:integer**, **xs:float**, restriction of **xs:token**)
used by	elements: **tVariableDomain/lower** **tVariableDomain/upper**
source	<xs:simpleType name="tBoundary"> <xs:union memberTypes="xs:integer xs:float"> <xs:simpleType> <xs:restriction base="xs:token"> <xs:enumeration value="minusInfinite"/> <xs:enumeration value="infinite"/> </xs:restriction> </xs:simpleType> </xs:union> </xs:simpleType>

element **Instance**

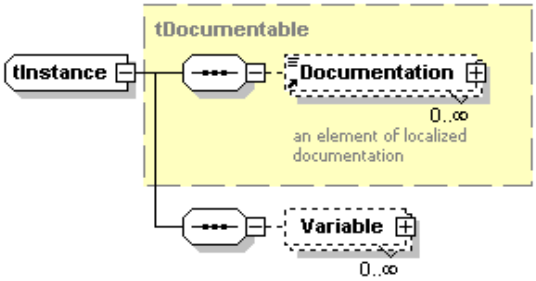
diagram	
type	**tInstance**
children	**Documentation** **Variable**
used by	complexType **tKnowledgeBase**

attributes	Name	Type	Use	Default	Fixed	Annotation
	isa	xs:string	required			

source

```
<xs:element name="Instance" type="tInstance"/>
```

complexType **tInstance**

diagram	
type	extension of **tDocumentable**
children	**Documentation** **Variable**
used by	element **Instance** complexTypes **tObject** **tTuple**

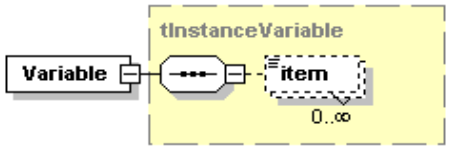
attributes	Name	Type	Use	Default	Fixed	Annotation
	isa	xs:string	required			

source

```
<xs:complexType name="tInstance" abstract="true" mixed="false">
  <xs:complexContent mixed="false">
    <xs:extension base="tDocumentable">
      <xs:sequence>
        <xs:element name="Variable" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="tInstanceVariable"/>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isa" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

element **tInstance/Variable**

diagram



type extension of **tInstanceVariable**

children **item**

attributes

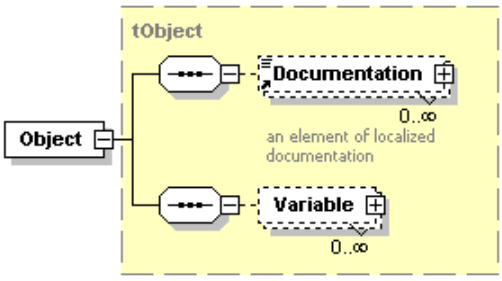
Name	Type	Use	Default	Fixed	Annotation
name	tName	required			

source

```
<xs:element name="Variable" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="tInstanceVariable"/>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
```

element **Object**

diagram



type **tObject**

children **Documentation Variable**

attributes

Name	Type	Use	Default	Fixed	Annotation
isa	xs:string	required			
name	tName	required			

identity constraints

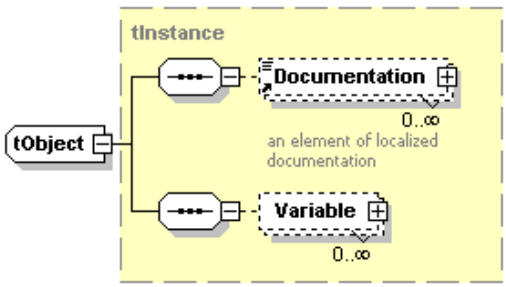
	Name	Refer	Selector	Field(s)
unique	uObject.Variable.name		./Variable	@name

source

```
<xs:element name="Object" type="tObject" substitutionGroup="Instance">
   <xs:unique name="uObject.Variable.name">
      <xs:selector xpath="./Variable"/>
      <xs:field xpath="@name"/>
   </xs:unique>
</xs:element>
```

complexType **tObject**

diagram



type	extension of **tInstance**
children	**Documentation Variable**
used by	element **Object**

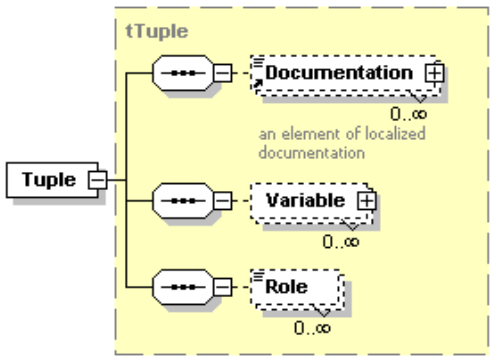
attributes	Name	Type	Use	Default	Fixed	Annotation
	isa	xs:string	required			
	name	tName	required			

source

```
<xs:complexType name="tObject">
  <xs:complexContent>
    <xs:extension base="tInstance">
      <xs:attributeGroup ref="aName">
        <xs:annotation>
          <xs:documentation>naming attributes for the Structure.</xs:documentation>
        </xs:annotation>
      </xs:attributeGroup>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

element **Tuple**

diagram



type	**tTuple**
children	**Documentation Variable Role**

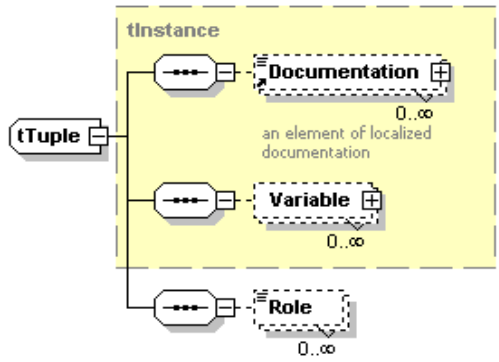
attributes	Name	Type	Use	Default	Fixed	Annotation
	isa	xs:string	required			

identity constraints		Name	Refer	Selector	Field(s)
	unique	uTuple.Variable.name		.//Variable	@name

source

```
<xs:element name="Tuple" type="tTuple" substitutionGroup="Instance">
  <xs:unique name="uTuple.Variable.name">
    <xs:selector xpath=".//Variable"/>
    <xs:field xpath="@name"/>
  </xs:unique>
</xs:element>
```

complexType **tTuple**

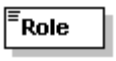
diagram	
type	extension of **tInstance**
children	**Documentation Variable Role**
used by	element **Tuple**

attributes	Name	Type	Use	Default	Fixed	Annotation
	isa	xs:string	required			

source

```
<xs:complexType name="tTuple">
   <xs:complexContent>
      <xs:extension base="tInstance">
         <xs:sequence>
            <xs:element name="Role" type="tInstanceRole" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

element **tTuple/Role**

diagram	
type	**tInstanceRole**

attributes	Name	Type	Use	Default	Fixed	Annotation
	name	tName	required			

source

```
<xs:element name="Role" type="tInstanceRole" minOccurs="0" maxOccurs="unbounded"/>
```

simpleType **tName**

type	**xs:token**
used by	attribute **aName/@name**

annotation	documentation	base type for naming Structures and Variables. note: simply an xsd:token for now. We should add AROM restrictions in the future.

source

```
<xs:simpleType name="tName">
   <xs:annotation>
      <xs:documentation>
            base type for naming Structures and Variables.
            note: simply an xsd:token for now. We should add AROM restrictions in the future.</xs:documentation>
   </xs:annotation>
   <xs:restriction base="xs:token"/>
</xs:simpleType>
```

attributeGroup **aName**

used by	complexTypes	**tInstanceRole tInstanceVariable tKnowledgeBase tObject tStructure tStructureRole tStructureVariable**					
attributes	Name	Type	Use	Default	Fixed	Annotation	
	name	tName	required				
annotation	documentation	naming attributes for Structures and Variables. at present only one required attribute called "name".					

source

```
<xs:attributeGroup name="aName">
  <xs:annotation>
    <xs:documentation>naming attributes for Structures and Variables. at present only one required attribute called "name".</xs:documentation>
  </xs:annotation>
  <xs:attribute name="name" type="tName" use="required"/>
</xs:attributeGroup>
```

complexType **tInstanceRole**

diagram

type extension of **xs:string**

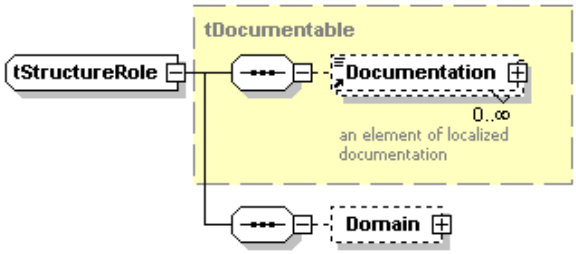
used by	element	**tTuple/Role**					
attributes	Name	Type	Use	Default	Fixed	Annotation	
	name	tName	required				

source

```
<xs:complexType name="tInstanceRole" mixed="true">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attributeGroup ref="aName"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

complexType **tStructureRole**

diagram



type extension of **tDocumentable**

children **Documentation Domain**

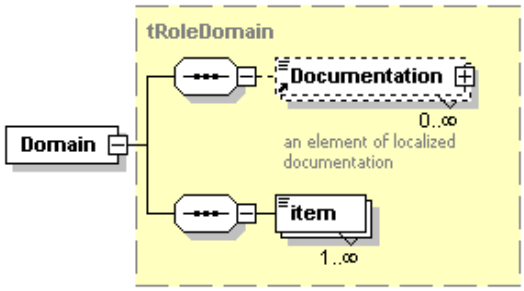
used by	element	**tAssociation/Role**					
attributes	Name	Type	Use	Default	Fixed	Annotation	
	name	tName	required				
	class	xs:string	required				
	min	tTypeCardinality	optional				
	max	tTypeCardinality	optional				

source

```
<xs:complexType name="tStructureRole">
```

```
    <xs:complexContent>
      <xs:extension base="tDocumentable">
        <xs:sequence>
          <xs:element name="Domain" type="tRoleDomain" minOccurs="0"/>
        </xs:sequence>
        <xs:attributeGroup ref="aName"/>
        <xs:attributeGroup ref="aRoleType"/>
      </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

element **tStructureRole/Domain**

diagram	
type	**tRoleDomain**
children	**Documentation item**
source	`<xs:element name="Domain" type="tRoleDomain" minOccurs="0"/>`

attributeGroup **aAttachment**

used by	element	**tStructureVariable/Attachment**					
attributes	Name	Type	Use	Default	Fixed	Annotation	
	class	xs:string					
	method	xs:string					
annotation	documentation	attachment attributes for Variables.					

source

```
<xs:attributeGroup name="aAttachment">
  <xs:annotation>
    <xs:documentation>attachment attributes for Variables.</xs:documentation>
  </xs:annotation>
  <xs:attribute name="class" type="xs:string"/>
  <xs:attribute name="method" type="xs:string"/>
</xs:attributeGroup>
```

Annexe IV – Liste de Travaux et Articles

Liste de Travaux et Articles

Liste de publications en relation avec ce travail de thèse.

[1] Kirsch-Pinheiro, M.; Villanova-Oliver, M.; Gensel, J.; Martin, H. « A Personalized and Context-Aware Adaptation Process for Web-Based Groupware Systems », *4th International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS'06), CAiSE'06 Workshop*, Luxembourg, June 5-6.

[2] Kirsch-Pinheiro, M.; Villanova-Oliver, M.; Gensel, J.; Martin, H. « Context-Aware Filtering for Collaborative Web Systems: Adapting the Awareness Information to the User's Context », *20th ACM Symposium on Applied Computing (SAC 2005) - Web Technologies and Applications (WTA)*, Santa Fe, New Mexico, Mars 2005, ACM Press, 2005, pp. 1668-1673.

[3] Kirsch-Pinheiro, M.; Villanova-Oliver, M.; Gensel, J.; Martin, H. « BW-M: A framework for awareness support in web-based groupware systems ». *9th International Conference On Computer Supported Cooperative Work In Design (CSCWD)*, 2005, Coventry, UK, pp. 240-246.

[4] Kirsch-Pinheiro, M. « Adaptation contextuelle de l'information de conscience de groupe dans les SIW collaboratifs ». *Actes du XXIIème Congrès Informatique des organisations et systèmes d'information et de décision – INFORSID 2005*, Grenoble, France, 2005. pp. 285-300.

[5] Kirsch-Pinheiro, M.; Villanova-Oliver, M.; Gensel, J.; Martin, H. « Une formalisation du cotexte dans les environnements coopératifs nomades ». *Actes des Deuxièmes Journées Francophones : Mobilité et Ubiquité 2005 (UbiMob'05)*, Grenoble, France, 2005. pp. 1-8.

[6] Kirsch-Pinheiro, M., Villanova-Oliver, M., Gensel, J., Martin, H. « A Context-Based Awareness Mechanism for Mobile Cooperative Users », In: R. Meersman, Z. Tari, A. Corsaro (Eds.), *LNCS 3292 - On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops: OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004*, octobre 2004. Springer-Verlag, 2004, pp. 9-10.

[7] Kirsch-Pinheiro, M., Gensel, J., Martin, H. « Representing Context for an Adaptative Awareness Mechanism », In: G.-J. de Vreede; L.A. Guerrero, G.M.Raventos (Eds.), *LNCS 3198 - X International Workshop on Groupware (CRIWG 2004)*, San Carlos, Costa Rica, septembre 2004, Springer-Verlag, 2004, pp. 339-348.

[8] Kirsch-Pinheiro, M., Gensel, J., Martin, H. « Awareness on Mobile Groupware Systems », In: A. Karmouch, L. Korba, E.R.M. Madeira (Eds.), *LNCS 3284 - 1st International Workshop on Mobility Aware Technologies and Applications (MATA 2004)*, Florianópolis, Brésil, octobre 2004. Springer-Verlag, 2004, pp. 78-87.

[9] Kirsch-Pinheiro, M.; Villanova-Oliver, M.; Gensel, J.; Lima, J.V.; Martin, H. « Providing a Progressive Access to Awareness Information ». In: R. Meersman et al. (Eds.), *LNCS 2888 - CoopIS/DOA/ODBASE 2003*, Springer-Verlag, 2003, pp. 336-353.

Adaptation Contextuelle et Personnalisée de l'Information de Conscience de Groupe au sein des Systèmes d'Information Coopératifs

Résumé

L'évolution des technologies mobiles (téléphones cellulaires, PDA, réseaux sans fil…) permet d'envisager leur utilisation pour un travail coopératif. Or, malgré les progrès effectués, on constate qu'il demeure encore des limitations en termes de capacités d'affichage, de batterie, ou encore de bande passante du réseau. C'est pourquoi, dans les systèmes d'information coopératifs sur le Web, l'adaptation du contenu délivré à un utilisateur nomade est un problème crucial. Cette adaptation est particulièrement importante pour le support à l'information de conscience de groupe. La conscience de groupe désigne la connaissance qu'un utilisateur a à propos de son groupe, de ses collègues et de leurs activités, et constitue un contexte pour les activités individuelles. Or, le nomadisme rendu possible par les nouvelles technologies favorise la perte de contact entre les participants du groupe, et les systèmes doivent compenser cette perte par des mécanismes de conscience de groupe adaptés. Par ailleurs, en raison des contraintes auxquelles sont exposés les utilisateurs nomades, l'adaptation doit désormais agir sur le contenu lui-même, en le sélectionnant et l'organisant selon sa pertinence pour l'utilisateur nomade. Cette pertinence peut changer en fonction du contexte dans lequel se trouve cet utilisateur, et donc la prise en compte du contexte d'utilisation devient aussi nécessaire pour l'adaptation de l'information de conscience de groupe que la prise en compte des préférences de l'utilisateur. Dans ce travail de thèse, nous proposons une nouvelle approche pour la question de l'adaptation de l'information de conscience de groupe au sein des collecticiels sur le Web. Nous proposons un mécanisme de filtrage de l'information qui tient compte à la fois du contexte de l'utilisateur et de ses préférences dans ce contexte d'utilisation précis. La notion de contexte est représentée ici par un modèle à objets qui intègre autant le contexte physique de l'utilisateur (sa localisation, le dispositif utilisé…) que le contexte coopératif dans lequel il évolue (notions de groupe, de rôle, d'activité…). Les préférences de l'utilisateur sont représentées par des profils qui permettent de délivrer à l'utilisateur des informations organisées en plusieurs niveaux de détails. Enfin, ce mécanisme de filtrage est implémenté au sein d'un canevas nommé BW-M, lequel utilise le système de représentation de connaissances par objets AROM pour l'implantation du modèle de contexte.

Personalized and Context-Aware Adaptation Process for Awareness Information on Web-Based Groupware Systems

Abstract

The evolution of mobile technologies, like web-enable cellphones, PDAs and wireless networks, makes it now possible to use these technologies for collaborative work through web-based groupware systems. However, due to the limitations of these technologies and to the mobility that these technologies grant to users, some adaptation is essential in these systems. In order to adapt their behaviour to the user's context, groupware systems must be built as context-aware systems. Moreover, this adaptation concerns particularly awareness mechanisms since nomadic users have more acute need for adapted awareness information. Dues to the constraints related to new technologies, adaptation ought to act on the contents itself, by selecting it and organizing it according to its relevance for nomadic users. Besides, this relevance depends on the user's current context, since user's preferences and needs may change accordingly to this context. In this work, we present a two-fold approach for adapting awareness information delivered to a nomadic user by web-based groupware systems, through a filtering mechanism which considers both the current context and preferences for this context. We propose an object-based representation for the notion of context, which takes into account the user's physical context as well as the user's collaborative context, including elements dedicated to the collaborative process in which the user is involved (notions of group, role, activity, etc.). The user's preferences are represented by a set of pre-defined profiles, which are exploited by the adaptation process in order to organize the delivered awareness information into several levels of detail, based on a progressive access model. Finally, we propose a framework, named BW-M, which implements our proposition based on an object-based knowledge representation system named AROM.