REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université 08 Mai 45 Guelma
Faculté des Sciences et de l'Ingénierie

Département d'Informatique

Ecole Doctorale d'Informatique de l'Est (Pôle : Annaba)
Option : Génie logiciel

Mémoire présenté pour l'obtention du diplôme de :
Magister en Informatique

Titre du Mémoire :

Une approche pour la prise en compte des préférences des utilisateurs et du contexte pour l'adaptation et la personnalisation des systèmes d'information ubiquitaires

Réalisé par :

Benselim Mohamed Salah

Composition du jury :

Président :	Pr Hamid SERIDI	Professeur	Université 08 Mai 45, Guelma
Rapporteur :	Dr Hassina SERIDI	M.C	Université Badji Mokhtar, Annaba
Examinateurs :	Dr Allaoua CHAOUI	M.C	Université Mentouri, Constantine
	Dr Fadila ATIL	M.C	Université Badji Mokhtar, Annaba

JUILLET 2009

REMERCIEMENTS

Je remercie très particulièrement mon encadreur Mme SERIDI Hassina, maître de conférences à l'université de Annaba, pour sa grande disponibilité et son immense intérêt. Grâce à ses conseils, ses orientations et son appui moral, j'ai pu surmonter les moments difficiles et de doute rencontrés durant la préparation de ce travail.

Je tiens à remercier Mr SERIDI Hamid, professeur à l'université de Guelma, pour avoir accepté d'être président du jury, ainsi que pour ses efforts incessants au profit du développement de la formation de post-graduation à l'université de Guelma.

Je remercie également Mr CHAOUI, maître de conférences à l'université de Constantine, ainsi que Mme ATIL maître de conférences à l'université de Annaba pour avoir accepté d'examiner mon travail.

Je profite également pour remercier tous les responsables qui ont contribué au lancement et à l'encadrement de l'école doctorale d'informatique de l'est.

Je suis très reconnaissant à Mr Kolovos, de l'université de York (Grande Bretagne), pour le temps et l'intérêt qui m'a accordé lors de l'installation et l'utilisation de la plate-forme « Epsilon ».

Je ne dois pas oublier de remercier tous mes collègues pour leur soutien, compréhension et encouragement.

DÉDICACES

A la mémoire de ma mère,

A mon cher père pour son sacrifice
et son aide,

A mon épouse "WIDAD" pour sa
patience et son appui,

A mon fils bijou "YAHIA",

A mes frères et sœurs qui m'ont
tant soutenu et encouragé,

Et à tous mes amis.

بسم الله الرحمن الرحيم

SOMMAIRE

Sommaire

RÉSUMÉ

Résumé

L'émergence colossale des nouvelles technologies du Web ainsi que l'évolution de l'informatique ubiquitaire nécessitent beaucoup plus d'efforts quant à la conception et le développement des systèmes d'information. En effet, d'une part, les données proviennent de plusieurs sources d'informations très hétérogènes et distantes, et d'autre part, les utilisateurs sont de plus en plus nomades et utilisent de multiples dispositifs pour accéder à une information. Actuellement, le génie logiciel s'oriente vers le développement d'applications distribuées et ubiquitaires. Cette tendance sera contrainte par l'intégration des paramètres de l'informatique ubiquitaire (mobilité, hétérogénéité,…). Le grand défi des concepteurs sera la prise en compte des changements continuels de la situation courante de l'utilisateur. Ainsi, les nouvelles applications devront être capables de s'adapter avec la variation du contexte d'utilisation et de satisfaire les préférences des utilisateurs. L'objectif de notre étude est d'offrir aux concepteurs une démarche de développement d'applications sensibles au contexte capable de garantir la contextualisation des futurs systèmes d'information. Notre proposition, basée sur les principes de la MDA (Model Driven Architecture), est définie par trois étapes. Premièrement, la séparation des aspects contextuels en introduisant le processus 3TUP (3 Track Unified Process) et le processus de développement en « psi : Ψ». Deuxièmement, la modélisation du contexte en utilisant une approche graphique basée sur le langage UML (Unified Modeling Language). Et troisièmement, l'intégration de ce modèle contextuel dans le cycle de vie de la MDA en utilisant l'opération de fusion de modèles.

Mots-clés : Adaptation, contexte d'utilisation, fusion de modèles, génie logiciel, MDA, modélisation du contexte, préférences de l'utilisateur, séparation des aspects.

Abstract

The colossal emergence of new Web technologies and the evolution of ubiquitous computing require a lot of efforts as for design and development of information systems. Indeed, on the one hand, data come from different sources of very heterogeneous and distant information, and on the other hand, the users are more and more nomads and use multiple mobile devices to reach information. Nowadays, software engineering moves toward the development of distributed and ubiquitous applications. This tendency is constrained by parameters such as mobility and heterogeneity that characterize the current situation of the user. This work aims to propose a development approach which be able to take into account the variation of context of use during the development process of an application. Our proposal is based on MDA (Model Driven Architecture) principles and it is defined by three steps. First, the separation of contextual aspects by introducing the 3TUP (3 Track Unified Process) process and a new development process named « psi : Ψ». Second, the context modelling by using UML (Unified Modeling Language). And third, the integration of this contextual model into the MDA process by using the models merging operation.

Keywords: Adaptation, concerns separation, context modeling, context of use, MDA, Model merging, software engineering, user's preferences.

ملخــص

إن بروز تقنيات جديدة خاصة بالإنترنت و تطور ميادين الإعلام الآلي يتطلبان الكثير من الجهود من أجل إنجاز و تطوير أنظمة الإعلام. حيث أن معظم المعلومات تستخرج من مصادر مختلفة و متباعدة من جهة، و من جهة أخرى أن أغلب المستخدمين أصبحوا يتميزون بكثرة التنقل مع استعمال أجهزة متنوعة. حاليا، تتوجه هندسة البرامج نحو تطوير تطبيقات موزعة و متوفرة مهما كانت وضعية المستعمل عبر الزمان و المكان. يتميز هذا التوجه بضرورة إدماج الخصوصيات الجديدة مثل تنقلات المستعمل، خيارات المستعمل و تنوع الأجهزة المستعملة و التي تمثل «سياق الاستعمال». إذن هناك تحدي كبير سيواجهه مهندسو البرامج و المتمثل في الأخذ بعين الاعتبار للتغييرات المتكررة التي تميز الوضعية الحالية للمستعمل. هذا ما سيسمح للتطبيقات الجديدة بالتأقلم و التكيف تماشيا مع تحولات سياق الاستعمال و بالتالي تلبية رغبات و خيارات المستعمل.
الهدف من هذه الدراسة هو توفير طريقة منهجية لتطوير و إنجاز تطبيقات قادرة على إدماج سياق المستعمل في أنظمة الإعلام المتباعدة. من خلال هذا العمل نقترح نظرة جديدة ترتكز على أسس الـ MDA (Model Driven Architecture) وهي طريقة تعتمد على هندسة النماذج.
يتلخص اقتراحنا في ثلاث مراحل رئيسية. أولا، نقوم بفصل العناصر المكونة لسياق الاستعمال عن كل العناصر الأخرى (الوظيفية و التكنولوجية)، لأجل ذلك نقترح استعمال طريقة جديدة «بسي : Ψ » تعتمد على أسلوب جديد يسمى « 3TUP : 3 Track Unified Process ».
المرحلة الثانية تتمثل في إنشاء نموذج خاص يسمح بتمثيل عناصر السياق المفصولة سابقا.
أخيرا، نقوم بإدخال النموذج المتحصل عليه في دورة حياة الـ MDA و ذلك باستعمال عملية دمج النماذج.

كلمـــات مرشـــدة : هندسة البرامج، هندسة النماذج، تكييف، سياق الاستعمال، خيارات المستخدم، فصل العناصر، نمذجة السياق، دمج النماذج.

TABLE DES FIGURES

LISTE DES TABLEAUX

ABRÉVIATIONS ET ACRONYMES

2TUP	2 Track Unified Process
3TUP	3 Track Unified Process
ACID	Atomicité, Cohérence, Isolation et Durabilité
CC/PP	Composite Capabilities / Preference Profiles
CIM	Computation Independent Model
CM	Contextual Model
CoOL	Context Ontology Language
CSCW	Computer Supported Cooperative Work
CSOA	Context-aware Service Oriented Architecture
CWM	Common Warehouse Metamodel
ECA	Enterprise Computing Architecture
ECL	Epsilon Comparison Language
EDOC	Enterprise Distributed Object Computing
EMF	Eclipse Modelling Framework
EML	Epsilon Merging Language
EOL	Epsilon Object Language
Epsilon	Extensible Platform for Specification of Integrated Languages for mOdel maNagement
ETL	Epsilon Transformation Language
EVL	Epsilon Validation Language
HTML	HyperText Markup Language
IHM	Interaction Homme-Machine
KSOM	Kohonen Self-Organizing Map
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MM	Merged Model
MMMs	Méta-Méta-Modèles
MMs	Méta-Modèles
MOF	Meta Object Facility
OMG	Object Management Group
PDA	Personal Digital Assistant
PDM	Platform Description Model
PIM	Platform Independent Model

PSM	Platform Specific Model
PUMAS	Peer Ubiquitous Multi-Agent System
SECAS	Simple Environment for Context Aware Systems
SID	Systèmes d'Informations Distribués
TCAO	Travail Coopératif Assisté par Ordinateur
UML	Unified Modeling Language
UP	Unified Process
XMI	XML Metadata Interchange
XML	eXtended Markup Language
CORBA	Common Object Request Broker Architecture
J2EE	Java Enterprise Edition

INTRODUCTION GÉNÉRALE

i. Préambule

Les services offerts par le Web sont multiples, variés et en nette expansion. En effet, le Web nous permet de rechercher, d'accéder, d'exploiter et d'échanger les informations provenant de plusieurs sources d'information (hétérogènes et distantes) et en utilisant plusieurs dispositifs (fixes ou mobiles). Bénéficiant des avantages de l'informatique nomade, chaque utilisateur peut, de n'importe où et n'importe quand, interroger le système ou interagir avec une application pour avoir des réponses et des données bien définies. Ces réponses, même si elles sont justes, sont généralement très nombreuses et ne sont pas toutes aussi intéressantes et pertinentes; c'est-à-dire qu'elles ne répondent pas toutes aux souhaits de l'utilisateur. Celui-ci sera obligé de passer un temps considérable dans la recherche de l'information pertinente parmi les réponses proposées. L'objectif est de réduire la liste des réponses fournies par la prise en considération des souhaits de l'utilisateur et des conditions d'exécution de l'application. On parle, alors, de prise en compte des préférences de l'utilisateur et du contexte d'utilisation par les systèmes d'information.
Les concepteurs de systèmes d'information ne doivent pas négliger ni ignorer les contraintes contextuelles qui entourent l'exécution d'une application et l'interaction avec l'utilisateur. Ceci permettra d'avoir des applications sensibles au contexte qui, d'une part, fournissent des informations personnalisées et pertinentes et, d'autre part, s'adaptent à la variation des conditions d'exécution issue de l'informatique ubiquitaire. La question est comment une application peut-elle prendre en charge les variations et les changements des conditions d'exécution (contexte d'utilisation et préférences utilisateur), et à quel niveau (par rapport à l'application) faut-il introduire ou insérer les informations contextuelles capturées. Ceci nous ramène à réfléchir sur la possibilité d'étendre les approches de conception existantes ou essayer de proposer de nouvelles démarches pour la conception des systèmes d'informations adaptés au contexte d'utilisation (l'environnement, l'application et l'utilisateur).

ii. Motivations

L'utilisation colossale du Web nécessite une nouvelle vision des systèmes d'information distants. L'évolution croissante des technologies hard (réseaux sans fil, dispositifs mobiles, PC portables, téléphones mobiles,…) destinées pour l'accès à une information distante a poussé la communauté du génie logiciel à suivre cette allure de croissance en garantissant des systèmes d'information plus adaptés et plus personnalisés.
En effet, le besoin d'assurer la disponibilité de l'information n'importe où et n'importe quand a conduit vers une forte émergence de l'informatique ubiquitaire. Les nouvelles tendances s'orientent vers l'utilisation de la notion de « modèle » et l'ingénierie des modèles (MDE : Model Driven Engineering) qui présente de grandes facilités pour représenter, traiter, stocker

et exploiter les informations. Ainsi, la MDA (Model Driven Architecture) offre plusieurs opportunités dans le domaine du génie logiciel comme l'utilisation de modèles pour formaliser et gérer l'information, la séparation des différents aspects d'un système (fonctionnels et technologiques), la portabilité, l'interopérabilité et la réutilisation.
Malgré ses avantages, la MDA actuelle n'inclut pas d'une façon claire et explicite la notion de « contexte d'utilisation » dans son processus de développement. Cette notion, liée aux changements continuels de la situation courante de l'utilisateur, représente une partie cruciale pour la personnalisation des applications surtout lorsque nous évoluons dans un environnement nomade et ubiquitaire.

iii. Problématique

Dans un environnement ubiquitaire, l'utilisateur est nomade et de plus en plus exigeant quant à la qualité des informations demandées (nature, contenu, affichage,…). L'utilisation de différents dispositifs mobiles conduit à un changement perpétuel (temps, espace, environnement) de la situation courante de l'utilisateur. Ces paramètres affectent directement la nature et le mode d'exécution d'une tache. Dans ce cas, les développeurs auront en face le problème d'accès au même système en différentes situations et avec divers dispositifs. Aussi, ils seront contraints de prendre en charge et de résoudre le problème d'hétérogénéité (informations, sources d'information, matériels, logiciels).
La problématique de notre étude cherche à satisfaire les souhaits et les préférences des utilisateurs quelque soient les conditions environnantes de la situation courante de l'utilisateur. Pour cela, nous avons cherché à proposer une approche qui peut introduire la notion de contexte dans le processus de développement d'une application.

iv. Objectifs

L'objectif de notre étude est d'assurer l'adaptation et la personnalisation des systèmes d'information ubiquitaires par rapport au contexte d'utilisation. Cette adaptation permettra de fournir des informations pertinentes aux utilisateurs selon leurs préférences souhaitées. Notre but est de proposer une approche capable de prendre en compte les préférences utilisateurs et le contexte d'utilisation lors du processus de développement d'une application. Pour arriver à nos fins, nous avons d'abord séparé les aspects contextuels d'un système par rapport aux autres aspects (métiers et techniques). Ensuite, nous avons modélisé ces aspects dans un nouveau modèle contextuel. Enfin, ce modèle contextuel est intégré dans le processus de développement de l'approche MDA par une opération de fusion de modèles.

v. Contributions

Cette étude nous a permis de contribuer à la satisfaction des utilisateurs mobiles et à la personnalisation des systèmes d'informations ubiquitaires. Cette contribution englobe la notion de séparation d'un nouveau type d'aspects qui est l'aspect contextuel, la proposition d'un nouveau type de processus appelé : 3 Track Unified Process (3TUP), la présentation et la description d'un nouveau processus de développement appelé « **psi : Ψ** », la construction d'un nouveau modèle contextuel qui formalise tous les éléments pouvant influencer le contexte d'utilisation et enfin une méthode basée sur la fusion de modèles pour pouvoir intégrer le modèle contextuel dans le processus de développement de l'approche MDA.

vi. Organisation du document

La suite de ce manuscrit comporte sept chapitres qui s'articulent en trois parties. La première partie, composée des trois premiers chapitres, introduit les concepts et un état de l'art du domaine d'étude. La seconde présente les contributions apportées à travers cette étude. Tandis que la troisième partie expose un exemple d'application des travaux réalisés sous forme d'un cas d'étude.

Première partie
Dans la première partie, le premier chapitre présente un état de l'art sur la notion du contexte d'utilisation ainsi que la sensibilité au contexte. Le chapitre 2 décrit l'architecture et le fonctionnement des systèmes d'information et en particulier ceux évoluant dans le domaine de l'informatique ubiquitaire. Le dernier chapitre de cette partie présente une description globale de l'approche MDA.
Dans cette première partie constituant l'état de l'art des travaux relatifs à notre sujet, nous présentons seulement les concepts et principes nécessaires à la compréhension de notre étude et nous citons les références des sources permettant au lecteur d'approfondir ses connaissances sur le sujet.

Deuxième partie
La deuxième partie, quant à elle, comporte trois chapitres décrivant nos contributions via ce travail. Nous introduisons cette partie par une discussion globale qui décortique les opportunités offertes par la MDA pour prendre en compte le contexte d'utilisation et les préférences des utilisateurs. Le chapitre 4 aborde la première étape (séparation des aspects) de l'approche proposée. La seconde étape (modélisation du contexte) est traitée dans le chapitre 5. La dernière étape (intégration du modèle contextuel) de notre proposition est abordée dans le chapitre 6.
Notre proposition, basée en particulier sur les approches de l'ingénierie dirigée par les modèles nous a permis, dans un premier temps, de séparer les informations contextuelles, et ensuite de les modéliser dans un modèle indépendant. Ce modèle contextuel fera l'objet d'une opération de fusion avec le modèle métier (PIM) avant d'être transformé vers un modèle spécifique (PSM).

Troisième partie
La troisième partie, comportant le chapitre 7, expose une expérimentation d'un cas d'étude pour démontrer la faisabilité de l'approche proposée. Ainsi, toutes les étapes de cette approche sont réalisées sous la plate-forme « Eclipse » et en utilisant les langages spécifiques fournis par la plate-forme « Epsilon ».

Conclusion et perspectives
Enfin, ce document s'achève par un bilan concluant le travail réalisé et une présentation de quelques perspectives.

Partie I

Etat de l’art

CHAPITRE 1

LES SYSTÈMES D'INFORMATION UBIQUITAIRES

Introduction

Le système d'information représente le noyau de toute organisation qui peut lui garantir plusieurs qualités comme la rapidité, la fiabilité et la pertinence. Il se rapporte à toutes les données et tous les modes de traitement des informations utilisées dans cette organisation et il ne se limite pas à ce qui est informatisé. En effet, c'est un élément primordial du pilotage d'une telle organisation et de ses activités pour aider à la prise de décision. Pour faire face à la concurrence grandissante, les entreprises s'appuient sur la qualité et l'efficacité de leurs systèmes d'information pour évoluer et satisfaire les clients de plus en plus exigeants.

Aujourd'hui, le grand défi des systèmes d'information est de prendre en charge la mobilité des utilisateurs et l'hétérogénéité des dispositifs utilisés.
Le développement rapide de l'informatique mobile (téléphone mobile, assistants personnels, ordinateur portables…) permet d'envisager la mise en œuvre de systèmes d'information souples et dynamiques appelés : les systèmes d'information ubiquitaires (ou pervasifs). A travers ces systèmes, l'utilisateur dispose à tout moment et à tout endroit d'un accès à l'information.

L'ubiquité des systèmes d'information peut être atteinte par la personnalisation et l'adaptation de ces systèmes avec les préférences des utilisateurs et le contexte d'utilisation de chaque service. Ainsi, ces systèmes satisferont les caractéristiques, les préférences et les objectifs des utilisateurs.

Les systèmes d'information ubiquitaires représentent une extension des systèmes d'information classiques et qui doivent vérifier les contraintes d'ubiquité comme la distribution, la mobilité, l'hétérogénéité, l'adaptation et la personnalisation.

1.1 L'informatique ubiquitaire

La notion d'informatique ubiquitaire (Ang.: ubiquitous computing ou ubicomp), est introduite pour la première fois par Mark Weiser en 1991 [Weiser, 1991]. C'est un paradigme assez récent qui permet à un utilisateur d'accéder aux données où qu'il soit et quand il le désire ; c'est-à-dire que le déplacement dans l'espace et la variation du temps ne soient pas un obstacle envers l'accès aux informations même si celles-ci présentent des caractéristiques

hétérogènes et proviennent, parfois, de sources d'information distantes et distribuées. L'ubiquité fait appel aux réseaux ambiants (ubiquitous networks) qui se caractérisent par des entités mobiles (terminaux, routeurs, PDA, téléphones cellulaires,…etc.) communicantes et parfois de très petite taille. L'utilisation de ces réseaux sera contrainte par les problèmes de mobilité, de sécurité et de sûreté (confidentialité des données, fiabilité des applications), de continuité de services (tolérance aux pannes) et de qualité de service.

1.1.1 Les éléments de l'informatique ubiquitaire

L'interactivité des différentes machines nécessite, pour s'adapter aux multiples situations, plusieurs types de principes lorsqu'il s'agit de développer une application [Banavar, 2002].

Le matériel :
En plus qu'il soit un lieu de stockage et de traitement des données, le matériel représente un portail pour s'introduire dans une application ou dans un espace de données.
L'application :
Une application n'est pas seulement un logiciel destiné à exploiter les capacités d'un matériel, mais, c'est aussi un moyen efficace qui permet à l'utilisateur d'améliorer les performances d'une tache.
L'environnement :
Un environnement d'exécution ne se réduit pas à un espace virtuel qui existe pour stocker et exécuter un logiciel, mais englobe, aussi, toutes les informations des objets environnants d'un utilisateur.

1.1.2 Cycle de vie de l'informatique ubiquitaire

G. Banavar [Banavar, 2000] propose un cycle de vie formé de trois étapes :
Temps de conception (design-time)
C'est le temps nécessaire à un développeur pour créer, maintenir ou optimiser une application.

Temps de chargement (load-time)
C'est un temps durant lequel le système compose, adapte et charge les composants d'une application.

Temps d'exécution (run-time)
C'est le temps pendant lequel l'utilisateur exploite l'application et utilise ses fonctionnalités.

1.1.3 Clefs de recherche dans le domaine de l'informatique ubiquitaire [Banavar, 2000]

Espace intelligent (smart spaces)
C'est une zone bien définie, ouverte ou enclose, qui renferme plusieurs systèmes incorporés (computers, senseurs, interfaces utilisateur et les infrastructures de services).
Transparence (invisibility)
Les utilisateurs ne sont pas, nécessairement, préoccupés par leur interaction avec les technologies de l'informatique ubiquitaire.
Ascension située (localized scalability)
S'occupe de la gestion des échanges d'information entre les utilisateurs et leur entourage (objets environnants).

1.2 Notions sur les systèmes d'information

1.2.1 Définition d'un système d'information

Un système d'information est l'ensemble des moyens et procédures utilisés en vue de restituer aux utilisateurs une information directement utilisable au bon moment.

Le système d'information a pour mission de récolter, stocker, traiter, mettre en forme et diffuser l'ensemble des informations qui sont nécessaires pour atteindre les objectifs opérationnels et stratégiques d'une organisation.

1.2.2 Architecture d'une organisation

Comme illustré dans la figure1, une organisation peut se décomposer en 3 sous-systèmes qui s'interagissent entre eux :
- Le **système de décision ou système de pilotage** qui réfléchit et décide. Il reçoit du système d'information toutes les informations nécessaires au pilotage de l'organisation
- Le **système opérant ou système opérationnel** qui traite les activités productives. Il essaie de réaliser les objectifs définis par le système de pilotage
- Le **système d'information** qui génère, mémorise, traite et communique les informations. Il assure le lien et le couplage entre système opérant et système de pilotage. Le système d'information fournit au système opérationnel les informations détaillées nécessaires à l'accomplissement de ses missions. Les flux entrants et sortants sont constitués exclusivement d'informations.

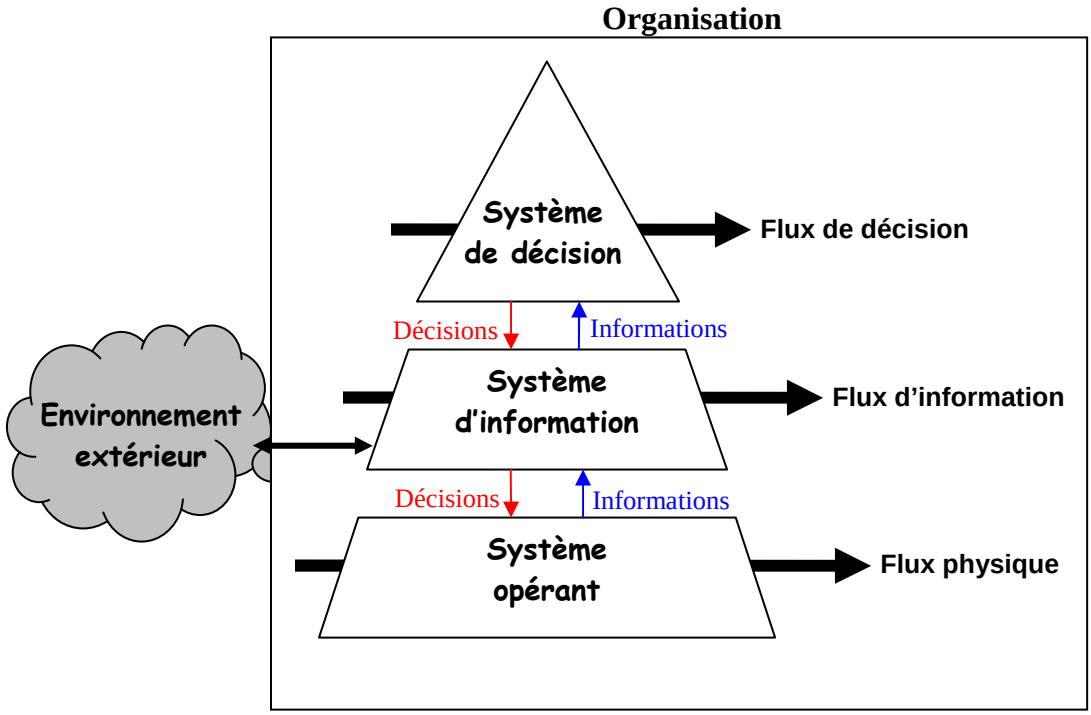


Figure 1. Niveaux de complexité d'une organisation

1.2.3 Objectifs d'un système d'information

Le rôle principal d'un système d'information est :

- D'assurer l'échange d'information entre :
  - Le système de décision et le système opérant.
  - L'organisation et l'environnement extérieur.
- De fournir au système de décision des informations concernant :
  - l'état de fonctionnement du système opérant afin de prendre les décisions nécessaires.
  - l'environnement extérieur pour prévoir les décisions d'adaptation.
  - L'état global du fonctionnement de l'organisation.
- De fournir au système opérant les informations nécessaires à son fonctionnement.

1.2.4 Fonctions d'un système d'information

Les principales fonctions du système d'information sont (figure 2):

- Collecte de l'information
- Saisie de l'information
- Stockage et mémorisation de l'information.
- Traitement de l'information.
- Restitution de l'information
- Transmission et diffusion de l'information.

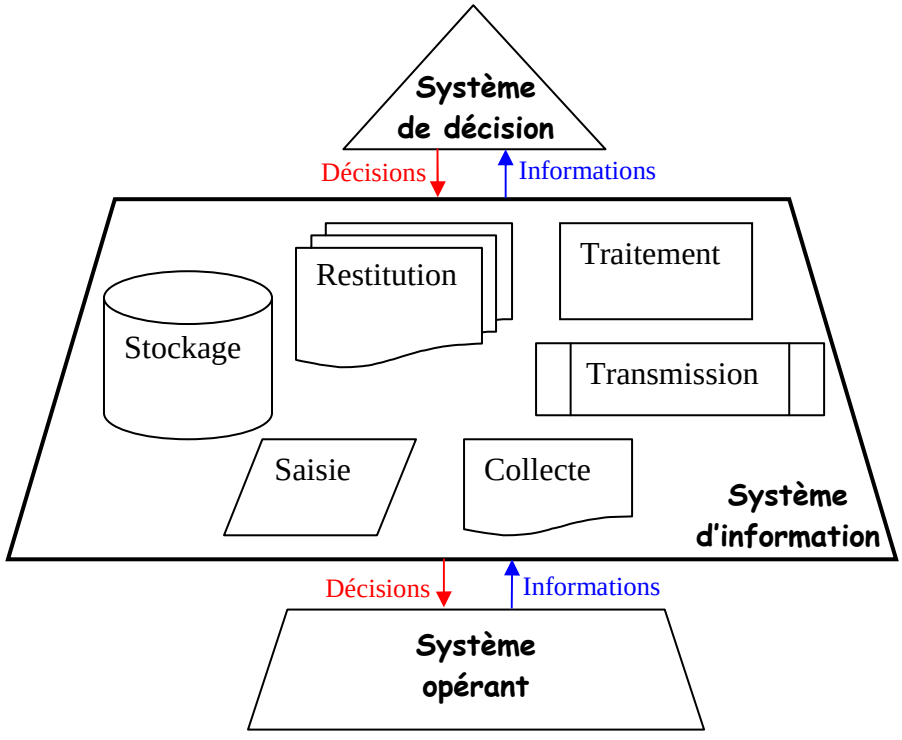


Figure 2. Fonctions d'un système d'information

1.2.5 Conception d'un système d'information

En général, la conception d'un système d'information peut être organisée en trois couches (figure 3):

- Présentation
- Logique d'application
- Gestion de ressources

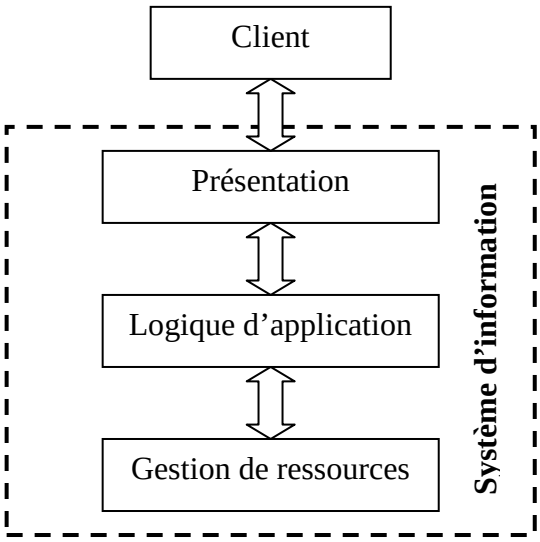


Figure 3. Les différentes couches d'un système d'information

En général, les couches peuvent être définies comme suit [Alonso, 2004] [Frankel, 2003]:

Présentation : cette couche assure la présentation des informations aux entités externes et l'interaction avec elles. Ces entités externes représentent les utilisateurs ou les autres machines qui peuvent communiquer avec le système d'information. Par exemple, dans le cas du Web, la couche de présentation peut être implémentée par une servlet Java qui fait la création de pages HTML (HyperText Markup Language).

Logique d'application : avant de délivrer les résultats, le système d'information doit d'abord traiter les données. Ce traitement, assuré par la couche logique d'application, fait appel à un programme qui implémente les opérations demandées par le client à travers la couche de présentation. Ces programmes représentent souvent les services fournis par le système d'information.

Gestion de ressources : cette couche est responsable de la gestion de tous les dépôts d'un système d'information. Un dépôt peut être une base de données, un système de fichier ou tout autre type de dépôt qui peut contenir les données nécessaires au fonctionnement du système d'information.

La conception d'un système d'information peut être du type top-down ou bottom-up.
La figure 4 illustre les deux cas et met en évidence les relations entre les étapes et les couches.

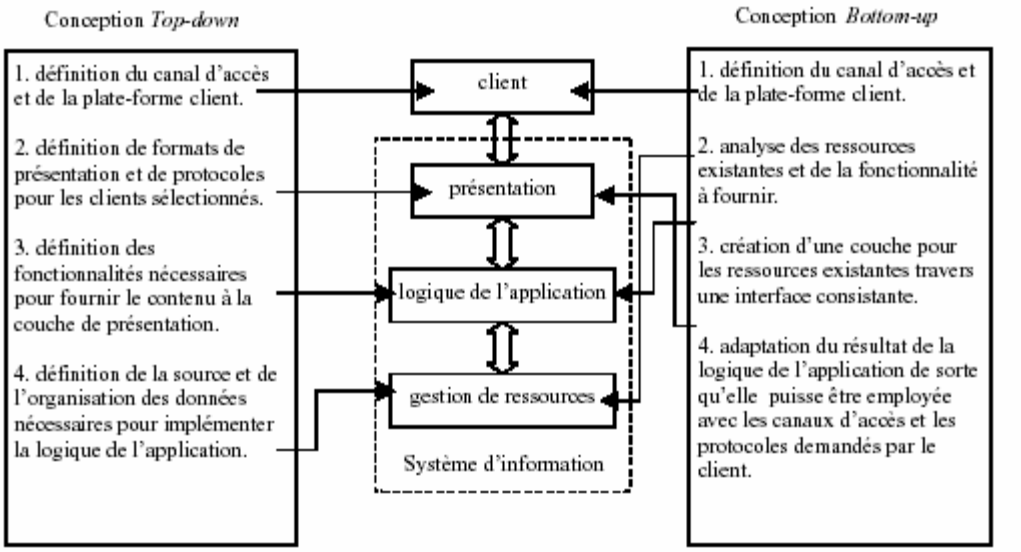


Figure 4. La conception du type top-down et bottom-up [Alonso, 2004]

1.2.6 Défis des systèmes d'information modernes

La conception des SI modernes s'articule autour de quelques points sensibles :

- La flexibilité des architectures logicielles
- La distribution massive des composants
- La forte hétérogénéité des sources d'information
- La forte hétérogénéité des ressources utilisées
- L'utilisation massive des données
- Mobilité, disponibilité et sécurité.

1.3 Les systèmes d'informations distribués (SID)

Le domaine des échanges d'information a connu, ces dernières années, l'émergence et l'utilisation très excessive de la messagerie électronique, ce qui justifie l'importance de l'étude des mécanismes et des outils d'aide au travail coopératif. Ces outils, qui ressortent du domaine du Travail Coopératif Assisté par Ordinateur (TCAO ou CSCW pour Computer Supported Cooperative Work), sont chargés de la gestion de flux de données et consistent à restructurer les organisations initialement centralisées au profit d'autres organisations distribuées dont les acteurs sont physiquement éloignés (notion d'entreprise distribuée) [Adam, 2000].

Un réseau coopératif d'entreprise représente la structure générale des systèmes d'information distribués qui existent dans les entreprises.

1.3.1 Architecture d'un réseau coopératif

L'architecture des réseaux d'entreprise peut être décrite sous la forme de graphes [Adam, 2000] ayant des nœuds et des arcs. Les données, qui sont de nature textuelle, graphique ou animée, peuvent être des documents uniques (rapports, brevets,…) ou des recueils de données (bases de données). Ces données circulent le long des arcs et subissent des transformations à chaque passage par les nœuds.
Les nœuds correspondent aux lieux où les informations sont traitées. Ces nœuds sont constitués d'un ou de plusieurs acteurs (humains, logiciels, matériels,…), chaque acteur

effectue des taches ou des activités (transformations de données) correspondant à son rôle dans le réseau.
Les arcs (relations ou transitions) représentent les échanges de données entre les nœuds. Ces échanges s'effectuent de façon synchrone ou asynchrone et se font dans le cadre d'un objectif fixé qui peut être soit la réalisation d'une tache simple, soit la mise en œuvre d'une procédure (ensemble de taches interdépendantes).
La structure organisationnelle du réseau peut être découpée selon les rôles joués par les acteurs, ou selon des sous objectifs ; on parle alors de l'articulation de l'organisation, correspondant à la connexité du graphe.
Ainsi, à chaque procédure correspond un réseau structuré composé de nœuds entre lesquels circulent des données.

1.3.2 Coopération dans un réseau d'entreprise

La coopération entre les acteurs d'un réseau résulte de leur capacité à collaborer, co-décider et coordonner leurs activités [De Michelis, 1994]. Pour être efficace, cette coopération requiert des connaissances sur les mécanismes de négociation.

En général, un mécanisme de négociation est caractérisé par l'enchaînement de trois phases (figure 5):

- La proposition
- Les revendications
- L'accord

Ces phases peuvent être décrites par la figure suivante :

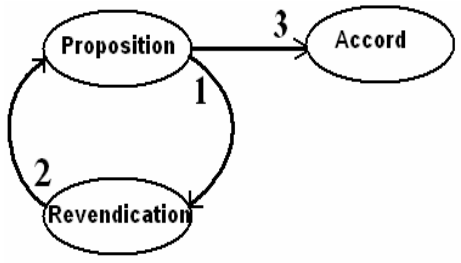


Figure 5. Mécanisme de négociation [Adam, 2000]

1- un acteur propose une solution,
2- les autres acteurs approuvent cette solution ou proposent des modifications,
3- les modifications peuvent être refusées ou prises en compte pour la proposition de nouvelles solutions.

La coopération est, par définition [Adam, 2000], l'aptitude à la communication, la coordination et la collaboration d'un ensemble d'acteurs pour la réalisation d'un objectif commun ; donc elle s'appuie, fondamentalement, sur les trois notions (communication, coordination et collaboration).

La communication : pour qu'il y est une certaine coopération entre deux acteurs, ils doivent se communiquer par échanges d'informations, soit directement, soit indirectement par modification de quelques paramètres de leur environnement commun.
La coordination : dans une organisation qui peut être représentée par un réseau à faible degré structuration, la réalisation d'un objectif commun n'est possible que par une action coordonnée de l'ensemble des acteurs.
La collaboration : représente l'activité commune d'un ensemble d'acteurs, et est utilisée pour exprimer une activité synchrone.

1.3.3 Architectures d'organisation de travail

Les entreprises sont généralement structurées autour de diverses architectures d'organisation de travail telles que l'architecture fonctionnelle, l'architecture divisionnelle et l'architecture matricielle [Alberto, 1993].

1.3.3.1 Architecture fonctionnelle

L'organisation est découpée en plusieurs unités, chaque unité correspondra à une fonction spécifique (administrative, commerciale,…). Ce qui nécessite, pour les systèmes d'information, que chaque unité aura ses propres logiciels et ses propres bases de données ; d'où la difficulté de la réalisation d'un système de gestion global cohérent (figure 6).

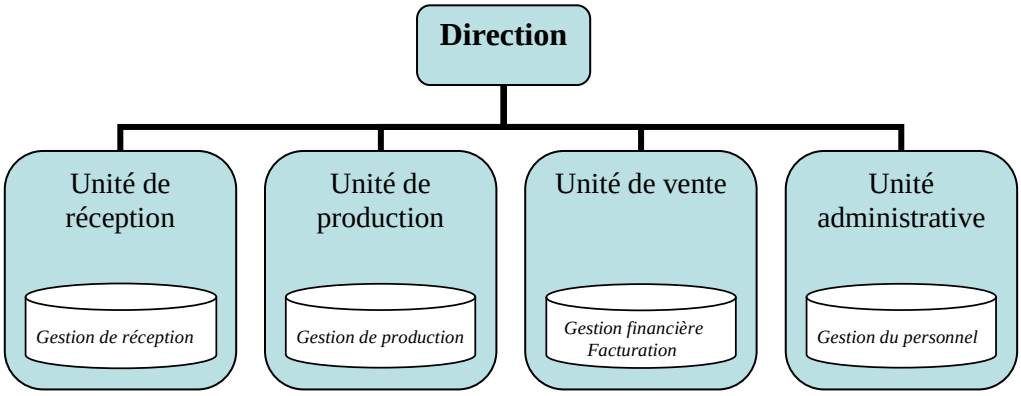


Figure 6. Exemple de structure fonctionnelle

1.3.3.2 Architecture divisionnelle

La structure de l'entreprise est divisée en unités physiquement éloignées, mais dans ce cas, le découpage se fait en fonction des marchés qu'elle occupe. Chaque unité dispose d'une organisation appropriée qui engendre des systèmes d'information correspondants (logiciels, bases de données,…) couvrant, ainsi, l'ensemble du cycle de vie d'un projet ou d'un produit (figure 7).

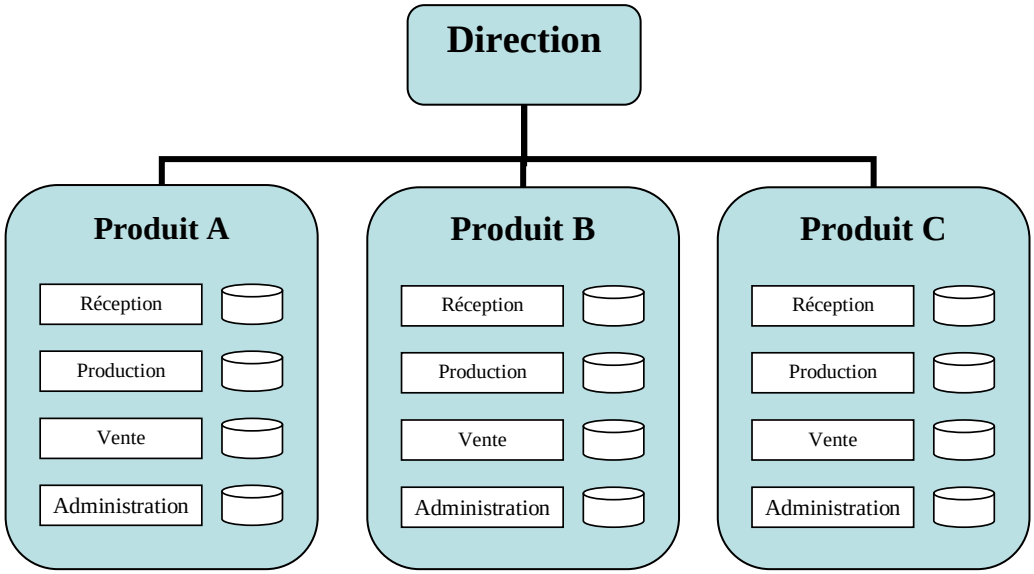


Figure 7. Exemple de structure divisionnelle

2.1.1.1 Architecture matricielle

L'entreprise est découpée suivant deux axes orthogonaux. Le premier axe désigne les unités fonctionnelles, tandis que le deuxième représente les produits ou les projets. Dans ce type d'architecture, chaque acteur effectue ses activités sous la responsabilités d'au moins deux dirigeants (le responsable d'unité et le chef de projet) ; ce qui entraînera l'apparition des problèmes de négociation entre les deux responsables. Les systèmes d'information relatifs à cette organisation reposent, surtout, sur la gestion des projets et doivent offrir un module d'aide à la décision qui permettra aux responsables de mieux exploiter les ressources humaines et logicielles (figure 8).

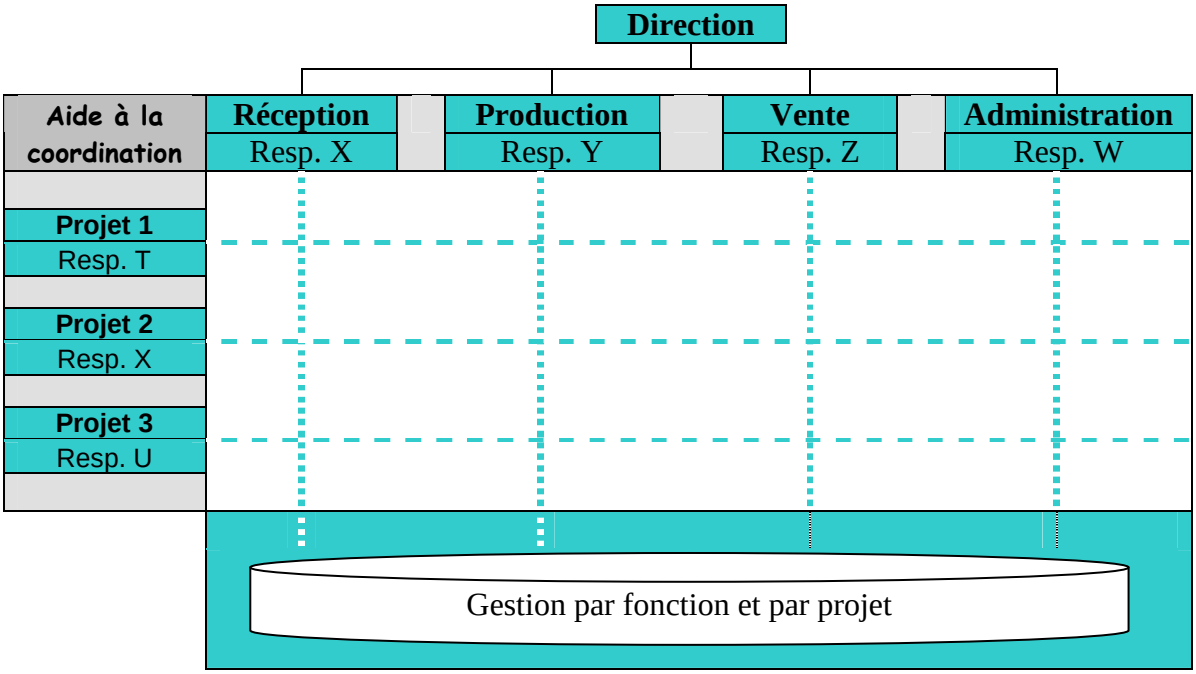


Figure 8. Exemple de structure matricielle

1.4 Les systèmes d'information ubiquitaires

Aujourd'hui, l'accès à distance aux informations et aux services est devenu très important. Ceci exige que les procédés et les outils qui gèrent l'information soient de nature ubiquitaire. Les systèmes d'information ubiquitaires se manifestent considérablement dans les domaines de développement d'applications, de recherche d'informations, d'organisation, de gestion des entreprises,…etc. Ces systèmes doivent être fortement disponibles à tous les utilisateurs, n'importe où, n'importe quand et sous n'importe quelle circonstance. Par conséquent, les concepteurs des systèmes d'information ubiquitaires doivent prendre en compte toutes les contraintes relatives à l'ubiquité comme l'hétérogénéité, la mobilité, la sécurité,…etc.
L'ubiquité des systèmes d'information peut être atteinte par la personnalisation et l'adaptation de ces systèmes avec les préférences des utilisateurs et le contexte d'utilisation de chaque service. Ainsi, ces systèmes satisferont les caractéristiques, les préférences et les objectifs des utilisateurs toujours exigeants.
Les systèmes d'information ubiquitaires représentent une extension des systèmes d'information classiques et qui doivent vérifier les contraintes d'ubiquité comme la distribution, la mobilité, l'hétérogénéité, l'adaptation et la personnalisation.

1.4.1 Distribution

Lors de l'utilisation très vaste du Web et des codes mobiles, une application a besoin d'exécuter des codes téléchargés sur le client et les serveurs. Le téléchargement se fait à partir de sources diverses et probablement distants. L'utilisateur de l'application devient nomade, c'est-à-dire qu'il aura la possibilité d'accéder aux données de cette application à n'importe quel moment et de n'importe quel endroit. Ceci peut être assuré par l'intermédiaire de différents moyens d'accès présentés sous formes de terminaux nomades (un téléphone GSM associé à un PDA, une Set Top Box : TV interactive ou WebTV).

La conception d'application doit, donc, prendre en considération de nouveaux paramètres (sécurité, disponibilité et mobilité) qui peuvent influencer la distribution des systèmes d'information.

La sécurité :
Les applications utilisées par les systèmes d'information font appel à de nombreux composants fournis et administrés par de multiples organisations dont le degré de confiance n'est pas toujours à la hauteur. Il convient donc d'établir un espace de confiance afin d'avertir l'application [Donsez, 1999]. Cet espace de confiance est créé pendant l'étape d'initialisation de l'application et est maintenu jusqu'à l'étape de terminaison. La grande sûreté et l'assurance absolue de l'espace de confiance conduisent logiquement à une meilleure sécurité des systèmes d'informations distribués, et ce, indépendamment des conditions de fonctionnement et des contraintes d'utilisation. Les informations peuvent être sécurisées par la mise en place de mécanismes de contrôle d'accès qui ont la possibilité de restreindre l'utilisation des systèmes d'information seulement aux utilisateurs autorisés (vérification de l'authenticité). Aussi, il faut toujours garder la trace des informations échangées et de leurs sources, ceci permettra un meilleur contrôle (en terme d'intégrité) des flux d'informations.

La disponibilité :

L'exécution d'une application nécessite la disponibilité de quelques composants seulement et non pas de tous ceux possibles. Le concepteur peut définir quel est le type de fonctionnement d'une application quand la configuration ne comporte que quelques composants disponibles, et quelles sont alors les conditions d'utilisation [Donsez, 1999].

1.4.2 Mobilité

La notion de mobilité désigne surtout la capacité de se déplacer continuellement dans l'espace. Dans les systèmes d'information ubiquitaires, chaque utilisateur est considéré comme "nomade", donc, il peut changer sa localisation (endroit) selon ses besoins et ses désirs (préférences). La prise en charge de la mobilité consiste à assurer, à tout moment, la capacité d'identification de la nouvelle localisation d'un utilisateur après ses mouvements ou ses déplacements, et offrir, ainsi, les chemins d'accès appropriés de cette nouvelle localisation. La variation de la localisation entraîne un changement du contexte d'utilisation, par conséquent, les systèmes d'information doivent être capables de rétablir les paramètres du contexte d'utilisation (liés à l'ancienne localisation) et les faire adapter avec la nouvelle localisation (dès que l'utilisateur reprenne la main sur l'application).

Gestion de la mobilité

La gestion de la mobilité nécessite l'identification et la définition de quelques besoins en terme de bases de données de localisation (localisation de téléphones cellulaires, gestion de flottes de véhicules). Pour définir ces besoins, il faut identifier les dimensions relatives au problème de la mobilité dans les accès aux données [Canals, 2002] :

Gestion de données spatio-temporelles : l'objectif des bases de localisation est de déterminer clairement l'endroit et l'emplacement d'un utilisateur ou du moins l'approximer avec une faible marge d'erreur. Cette approximation permet une meilleure modélisation des données, une expression claire des requêtes et une évaluation efficace de ces mêmes requêtes.

Modèles d'accès à l'information : la mobilité nécessite d'envisager de nouveaux modèles d'accès aux données plus souples et plus efficaces que les modèles existants (par exemple, le modèle client-serveur). En effet, un modèle de souscription permet à un utilisateur d'être notifié automatiquement lorsqu'un événement satisfaisant son abonnement est publié. Un modèle de diffusion permet à un serveur de diffuser de façon répétitive une information qui peut être captée par les utilisateurs (clients) intéressés. Autres modèles basés sur une diffusion épidémique de l'information peuvent être envisagés pour permettre le passage à l'échelle [Canals, 2002].

Synchronisation et cohérence transactionnelle : la prise en charge de la mobilité nécessite souvent un travail en mode déconnecté. Celui-ci provoque généralement des problèmes d'hétérogénéité et de dissemblance entre la version originale (souvent hébergée sur le réseau fixe) et ses différentes copies (hébergées sur des dispositifs mobiles). Lors des reconnexions, des protocoles complexes de synchronisation/réconciliation doivent être appliqués pour rétablir la convergence des différentes copies tout en respectant les règles de causalité et préservation de l'intention de l'utilisateur. La mobilité nécessite aussi la mise en œuvre des propriétés transactionnelles ACID (Atomicité, Cohérence, Isolation et Durabilité) parce que certaines applications ont des besoins plus forts de cohérence qui nécessitent le respect de ces propriétés.

Gestion des données embarquées : une bonne gestion de la mobilité pousse les concepteurs à concevoir produire des composants bases de données destinés à être embarqués sur des dispositifs mobiles ultra-légers. L'architecture de chaque dispositifs se caractérise par des

propriétés bien définies (portabilité, autonomie électrique, coût, sécurité …) et doit tenir compte des contraintes matérielles imposées (taille, débit, capacité mémoire, …).
Confidentialité des données : l'objectif majeur des utilisateurs mobiles est d'accéder plus facilement aux données. C'est pourquoi celles-ci sont stockées sur un serveur fixe et gérer, par exemple, par un hébergeur de données sur l'Internet (Database Service Provider). C'est alors que surgit le problème de confidentialité et de sécurité de ces données. Des modèles de sécurité de bout en bout restent à définir afin de résister à tous d'attaques et de menaces provenant d'un intrus ou de l'hébergeur lui-même.
Adaptabilité : l'objectif de l'adaptabilité est d'offrir à l'utilisateur, quelle que soit sa localisation et son type de terminal, un service aussi proche que possible de celui fourni dans un contexte fixe. A cet effet, toute information téléchargée sur un dispositif mobile doit dépendre, par exemple, des capacités d'affichage, de stockage et du temps de connexion nécessaire au téléchargement.

1.4.3 Hétérogénéité

L'hétérogénéité est la principale propriété qui caractérise les systèmes d'information modernes. En effet, cette caractéristique marque la majorité des éléments intervenant dans un système d'information. Nous distinguons :

- L'hétérogénéité des données,
- L'hétérogénéité des sources d'information,
- L'hétérogénéité des ressources utilisées,
- L'hétérogénéité des interfaces,
- L'hétérogénéité des logiciels.

1.4.4 Adaptation et personnalisation

L'évolution des systèmes d'information s'oriente actuellement vers plus d'adaptation et de personnalisation dans un environnement informatique marqué par l'hétérogénéité des sources d'information et par la mobilité incessante des utilisateurs.
Avec l'apparition de l'informatique ubiquitaire, le véritable challenge est d'accéder à de l'information pertinente dans environnement caractérisé par un contexte mobile et fortement dynamique. La personnalisation permet d'améliorer la pertinence de l'information retournée en prenant en considération tous les facteurs susceptibles d'influencer l'exécution d'une situation particulière.
Dans ce sens, nous avons élaboré une approche qui peut assurer l'adaptation et la personnalisation des systèmes d'information ubiquitaires suivant le contexte d'utilisation et les préférences des utilisateurs. Notre approche propose une solution qui sépare les contraintes contextuelles et les considère comme étant un type d'aspects indépendants. C'est-à-dire que nous nous basons sur la séparation entre trois types d'aspects : métiers, techniques et contextuels. Cette approche sera détaillée dans la partie « contributions ».

Conclusion

Dans ce premier chapitre, nous avons présenté un état de l'art qui décrit les principales notions des systèmes d'information. Ainsi, nous avons rappelé l'architecture et la conception

des systèmes d'information classiques, ensuite nous avons montré comment ces systèmes ont évolué pour s'adapter à des environnements distribués et hétérogènes. A cet effet, nous avons décrit la nouvelle tendance de ces systèmes à devenir ubiquitaires tout en garantissant la vérification des principales caractéristiques de cette ubiquité, à savoir : mobilité, hétérogénéité, distribution, adaptation et personnalisation.

CHAPITRE 2

LE CONTEXTE D'UTILISATION

Introduction

Aujourd'hui, l'accès aux informations distantes se fait à travers différents dispositifs tels que les ordinateurs de bureaux, les ordinateurs portables ou même des dispositifs mobiles comme les téléphones portables ou les PDA (Personal Digital Assistant).

Les informations requises par l'utilisateur doivent être adaptées, au plus possible, à son profil d'utilisation et à ses préférences. Ces dernières sont considérées comme des paramètres variables à cause de la diversité des comportements des utilisateurs, de la variation des éléments de l'environnement ainsi que l'évolution des ressources disponibles. Par conséquent, le résultat attendu va être certainement influencé par le changement de ces paramètres ; c'est-à-dire qu'il doit prendre en compte le contexte de l'utilisateur et de l'utilisation.

L'étude du contexte permet de déceler la nature des interactions entre l'utilisateur, les applications et l'environnement. Ces interactions comprendront non seulement les états physiques et logiques, mais aussi, les états émotionnels et les comportements de tous les objets environnants.

Les approches actuelles destinées au développement des applications se trouvent obligées de s'adapter avec les nouvelles contraintes relatives à la mobilité des utilisateurs et à l'hétérogénéité des dispositifs utilisés. Ces paramètres affectent directement l'état de la situation courante de l'utilisateur. En effet, les changements imprévus et continuels subis par une situation peuvent entraver la bonne exécution des taches. Donc, à un instant donné, chaque situation est caractérisée par des éléments bien définis qui peuvent influencer l'exécution des taches de cette situation. Ces éléments constituent le contexte d'utilisation de la situation courante d'un utilisateur.

Le contexte d'utilisation inclut tous les objets, physiques ou morales, qui peuvent agir ou interagir avec l'utilisateur, l'application ou les deux (y compris l'utilisateur et l'application eux même). Généralement, la notion de contexte d'utilisation comprend des éléments tels que la localisation, l'utilisateur (préférences, comportement,…), le temps, les ressources disponibles, les personnes avoisinantes, etc.

Les applications qui utilisent le contexte d'utilisation s'appellent : applications sensibles au contexte. Ces applications cherchent à s'adapter avec les changements du contexte d'utilisation et suivant les préférences imposées par l'utilisateur.

2.1 Le contexte

2.1.1 Définitions

Plusieurs définitions du terme « contexte » ont été introduites.
Schilit et Theimer [Schilit, 1994b] ont défini le contexte comme étant l'endroit, l'identité des personnes voisins et objets et les changements de ces objets.
Pas loin de la définition précédente, Brown et Chen [Brown, 1997] montrent que le contexte représente l'endroit, l'identité des gens qui entourent l'utilisateur, le temps, la saison, la température,…etc.
Ryan [Ryan, 1997] définit le contexte comme l'endroit de l'utilisateur, l'environnement, l'identité et le temps.
Dey [Dey, 1998], pour sa part, fournit une définition plus large : le contexte représente l'état émotionnel de l'utilisateur, la concentration, l'endroit, la date, le temps, les objets et les personnes présentes dans l'environnement de l'utilisateur.

Il existe d'autres définitions plus simplistes qui ont été citées par les chercheurs.
Brown [Brown, 1996] définit que le contexte est l'ensemble des éléments de l'environnement de l'utilisateur à partir duquel la machine de cet utilisateur peut acquérir des informations.
Franklin et Flaschbart [Franklin, 1998] affirment que le contexte représente, tout simplement, la situation de l'utilisateur.
Ward et Hooper [Ward, 1997] disent que le contexte est l'état de l'entourage d'une application.
Dey [Dey, 1999a] donne une autre définition : le contexte symbolise l'état de l'utilisateur et cet état peut être du type physique, social, émotionnel ou informationnel.
Pascoe [Pascoe, 1998] définit le contexte comme un sous-ensemble d'états physiques et conceptuels relatifs à une entité particulière.

Pour résumer ces définitions, Dey, Abowd et Salber [Dey, 2001] donnent une définition plus significative du contexte :

" 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 applications themselves".

Ce qui peut être traduit par :

Le contexte est toute information utilisable pour caractériser la situation d'une entité. Une entité peut être une personne, un endroit, ou tout objet considéré pertinent pour l'interaction entre l'utilisateur et l'application incluant l'utilisateur et l'application eux-mêmes.

2.1.2 Catégorisation du contexte

La situation d'une entité particulière est caractérisée par l'entourage dans lequel cette entité est produite ; cet entourage ou « contexte » représente l'ensemble des différentes

caractéristiques ambiantes ou environnantes de l'entité. Ce qui signifie que le changement d'une caractéristique peut affecter directement l'exécution d'une tache ou influencer le résultat souhaité. D'où la nécessité d'étudier cette variation du contexte en étudiant la catégorisation du contexte. Ces catégories doivent fournir des réponses aux questions qui, quoi, quand et où ? (who, what, when and where ?) d'une entité et relatives à l'exécution d'une tache tache par l'utilisateur pour pouvoir, ensuite, déterminer le pourquoi ? (why ?) de l'apparition d'une situation quelconque.
La diversité des types (ou catégories) du contexte découle de la grande variation du sens attribué par les chercheurs à la notion de « contexte ».

D'après les définitions précédentes, on peut ressortir les différents types du contexte. Ainsi, et en définissant le contexte, Ryan [Ryan, 1997] propose quatre types de contexte qui sont : l'endroit, l'environnement, l'identité et le temps. De leur part, Dey et Abowd [Dey, 1999b], ont repris ces mêmes types de contexte sauf qu'ils ont remplacé l'environnement par un nouveau type qui est l' « activité ». Pour Schilit [Schilit, 1994a], les catégories du contexte tournent autour des idées (Où êtes-vous ? , Avec qui vous êtes ? et Quelles sont les ressources (dans le voisinage) ?) , mais ils excluent ou ne prennent pas en compte d'autres informations plus importantes comme l'activité ou le temps.

Pour récapituler l'étude de la catégorisation du contexte, nous présentons l'étude réalisée par Dey et Abowd dans [Dey, 1999b]. Cette étude montre qu'il existe des types primaires et des types secondaires parce que en ayant un type de contexte (primaire) qui sera considéré comme index, on peut déduire des indices sur d'autres informations du contexte (secondaire). Par exemple, en sachant l'identité d'une personne (type primaire), on peut connaître facilement d'autres informations comme les numéros de téléphones, les adresses, la liste des amis, les autres personnes de l'environnement,…etc. Aussi, en ayant l'endroit d'une entité, par exemple une voiture, on peut ressortir d'autres informations pouvant être pertinentes car le contexte de l'entité « voiture » change et diffère lorsque cette voiture est stationnée dans un garage ou bien si elle est (circule) sur autoroute.

Les entités envisagées peuvent être réparties en trois classes :
- Les places : sont des régions de l'espace géographique
- Les personnes : peuvent être sous forme d'individus ou groupes, unies ou réparties
- Les objets : peuvent être des objets physiques ou des composants logiciels.

Les parties constituantes de l'environnement sont [Schilit, 1994a]:
- Env. informatique : processeurs disponibles, machines disponibles, réseaux, …
- Env. de l'utilisateur : endroit, personnes voisins, situation sociale,…
- Env. physique : niveau de lumière, niveau de son,…

Les principaux éléments qui peuvent constituer le contexte sont :

- L'endroit : caractérise la position et la situation d'un utilisateur ou d'une entité particulière par rapport aux éléments avoisinants et à l'espace

- L'environnement (ou l'activité) : est l'ensemble de tous les éléments et tous les événements qui peuvent avoir lieu dans une situation

- L'identité : représente la nature (définition) exacte d'une entité ainsi que sa description comportementale.

- ❖ Le temps : décrit le moment (instant) du déroulement d'une tache

Ces types de contexte seront considérés comme des types primaires (premier niveau), et à partir desquels, on peut dériver d'autres types de second niveau qui seront utiles pour mieux décortiquer le contexte d'une entité particulière.

2.1.3 Domaines du contexte

Les domaines de contexte pouvant être prises en compte s'articulent autour de plusieurs dimensions qui influencent l'exécution d'une situation [Ranganathan, 2005] [Privat, 2007]:

Contexte physique

- Spatial (localisation, position, orientation, attitude, etc.)
- Temporel (heure, jour, date, semaine, saison, …)
- Environnemental (éclairage, niveau de bruit, température, pression, et plus généralement toutes variables physiques pertinentes pouvant être acquises à partir de l'environnement)

Contexte utilisateur

- Profil (rôle, préférences, permissions)
- Capacités (connaissances du domaine, niveau d'expertise)
- Activité (marche, travail, voiture), tache au sens procédure (niveau en dessous d'une activité)
- Présence disponibilité, …

Contexte du groupe

- Activité du groupe
- Relations et liens sociaux,
- Autres personnes

Contexte social

- Activité sociale en cours
- Contexte organisationnel, hiérarchique, politique, etc.
- Normes sociales implicitement acceptées et partagées

Contexte de l'application

- Emails reçus
- Sites Web visités
- Fichiers précédemment ouverts

Contexte « système »

- Logiciel (middleware et applications), bases de connaissances
- Procédures d'action, existence de stratégies vicariantes
- Nature et état du réseau utilisé
- Etat matériel (capacités permanentes et actuelles, performances, charge, disponibilité, niveaux de batterie, etc.)

2.1.4 Modes d'utilisation possibles du contexte

Le contexte peut être utilisé selon plusieurs façons possibles [Privat, 2007] :

- Déclenchement ou arrêt automatique du système
- Adaptation automatique instantanée du système
- Apprentissage incrémental incorporé dans le système
- Information annexe fournie en sortie : état du système ou contexte distant
- Contrôle de l'activité externe du système

2.1.5 Qualité du contexte

Une information contextuelle doit présenter des propriétés de qualité très particulières pour pouvoir être prise en compte d'une manière efficace [Ranganathan, 2005] :
Confiance : c'est la probabilité pour que le contexte capturé soit correct.
Précision (exactitude) : représente le pourcentage d'erreur des contextes capturés ou déduits.
Fraîcheur : c'est l'intervalle de temps qui sépare les différentes interprétations d'un certain type de contexte.
Résolution (détermination) : c'est un intervalle qui délimite l'information de localisation.

2.1.6 Bénéfices internes

Les bénéfices internes attendus de la prise en compte du contexte sont [Privat, 2007] :

- Désambiguïser l'information issue des inputs primaires,
- Fiabiliser un système,
- Augmenter les performances adaptatives d'un système.

2.1.7 Bénéfices pour l'utilisateur

Pour l'utilisateur final, les bénéfices attendus de la prise en compte du contexte sont [Privat, 2007]:

- Amélioration du confort d'utilisation
- Rendre l'application plus proche d'un agent humain, permettre de lui conférer du « bon sens »
- Prise en compte de la performance des utilisateurs par rapport au système
- Prendre en compte des besoins latents, même implicites et non formulés par l'utilisateur
- Adaptation intelligente à l'utilisateur par apprentissage incrémental, apprentissage de préférences des utilisateurs
- Eviter de demander des entrées d'information inutiles ou redondantes
- Eviter de présenter des informations en sortie inutiles ou redondantes
- Fournir des informations en sortie plus pertinentes, adaptées et personnalisées
- Adapter la présentation des informations aux conditions de consultation (terminaux disponibles)
- S'adapter à l'environnement de l'utilisateur et à l'utilisateur lui-même, à son activité, sa situation, son état, etc.
- Resituer le contexte distant et enrichir la conscience de l'utilisateur en situation de communication distante (adaptation non automatique)

2.2 La sensibilité au contexte (context-aware)

2.2.1 Définitions

Après avoir répondu aux questions (qui, quoi, quand et où), et une fois on a identifié les types de contexte, on peut, maintenant, aborder la question « comment ? », et ce, pour parvenir à mieux comprendre le mode d'utilisation du contexte dans le monde de l'informatique nomade et plus particulièrement dans la conception des applications de l'informatique ubiquitaire.
Le terme « context-aware » est utilisé pour mettre en évidence l'utilisation et la sensibilité au contexte et sa prise en compte par les applications dans différents domaines. Ce terme englobe toutes les connaissances et les informations relatives à l'utilisation du contexte par les développeurs d'applications.
La notion de l'informatique sensible au contexte « context-aware computing », introduite par Schilit et Theimer en 1994 [Schilit, 1994b], désigne tout logiciel qui peut s'adapter avec les changements des utilisateurs et des objets dans un endroit d'utilisation précis et pour un temps donné. Cette définition exige, donc, que les applications soient auto-adaptables avec le contexte.
Toutes les définitions de l'informatique sensible au contexte convergent vers deux catégories : l'utilisation du contexte et l'adaptation au contexte [Dey, 1999b].
Dans le cadre de l'utilisation du contexte, Hull [Hull, 1997] et Pascoe [Pascoe, 1998a] [Pascoe, 1998b] résument le sens de l'informatique de contexte en l'aptitude d'assumer les quatre taches caractéristiques: la détection, la perception, l'interprétation et la réponse vis-à-vis de l'environnement de l'utilisateur. Par contre, Dey [Dey, 1998] réduit cette définition à la relation de l'application avec l'interface homme-machine. Salber [Salber, 1998] décrit la sensibilité au contexte comme la capacité de fournir un maximum de flexibilité d'un service basé sur la perception et la capture du contexte en temps réel. Pour la catégorie de l'adaptation au contexte, on parlera plutôt d'applications sensibles au contexte.

2.2.2 Les applications sensibles au contexte

De point de vue adaptation au contexte, les chercheurs [Schilit, 1994a], [Ward, 1997], [Korteum, 1998], [Brown, 1997], [Davies, 1998], [dey, 1997], [Abowd, 1997a] admettent que les applications sensibles le contexte sont des applications qui changent dynamiquement ou bien qui adaptent leur comportement suivant le contexte de l'application et de l'utilisateur.
Aussi, Brown [Brown, 1998] définit les applications utilisant le contexte comme étant les applications qui fournissent automatiquement l'information et/ou décident de l'action à prendre selon le contexte détecté.
En s'inspirant des différentes définitions précédentes, Dey et Abowd [Dey, 1999b] ont fourni une définition très significative et plus récapitulative du context-aware, et que nous allons adopter dans notre étude :

" 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".

Ce qui peut être traduit par :

Un système est sensible au contexte s'il utilise ce contexte pour fournir une information pertinente et/ou des services à l'utilisateur, sachant que cette pertinence dépend des taches de l'utilisateur.

Cette définition est plus générale que celles citées précédemment. En effet, si on considère une application où le comportement est invariable mais qui utilise les éléments du contexte seulement en affichage ou en consultation, et on veut connaître si cette application est context-aware ou bien non, alors dans ce cas le résultat ne sera pas le même en appliquant les différentes définitions. C'est-à-dire qu'une application peut ne pas être context-aware pour les définitions prétendues réduites (ou étroites) et qu'elle peut l'être d'autres définitions qui soient plus générales et plus vastes (par exemple la définition de Dey et Abowd [Dey, 1999b]).

2.2.3 Catégorisation des applications sensibles au contexte

En se basant sur la définition du context-aware, on peut ressortir toutes les informations pertinentes relatives au contexte. Maintenant, la question est de savoir comment utiliser ces informations dans le développement des applications. Pour cela, les chercheurs ont proposé la catégorisation des applications utilisant le contexte pour une meilleure prise en compte de cette notion.
En effet, Schilit [Schilit, 1994c] s'appuie sur l'idée que la prise en compte du contexte doit vérifier les aspects les plus importants : Où êtes-vous ? , Avec qui vous êtes ? Et Quelles sont les ressources (dans le voisinage) ? ; et il démontre que le contexte n'est pas fonction seulement de l'endroit (l'espace) mais aussi de tous les objets mobiles et susceptibles d'être changeants et qui se manifestent prés de l'utilisateur, il inclut même des paramètres comme la lumière, le son, le réseau, la communication, la situation sociale…etc. D'où, il propose une catégorisation du contexte basée sur la nature la tache, c'est-à-dire si cette tache perçoit une information ou exécute une commande, ou bien qu'elle s'effectue manuellement ou automatiquement. Ce qui revient à dire qu'on peut avoir quatre catégories d'applications utilisant le contexte obtenues par le croisement de deux axes orthogonaux ; le premier est lié à la nature de la tache (information ou commande), tandis que le deuxième désigne le mode d'exécution (manuel ou automatique). Ces catégories sont résumées dans le tableau suivant [Schilit, 1994c] :

	Manuel	Automatique
Information	Proximate selection & contextual information	Automatic contextual reconfiguration
Commande	Contextual commands	Context-triggered actions

Tableau 1. Les dimensions des applications sensibles au contexte [Schilit, 1994c]

a. Sélection de proximité :
C'est une technique d'interaction utilisateur qui consiste à proposer tous les objets proches se trouvant dans le voisinage de l'utilisateur.

b. Reconfiguration contextuelle automatique :

C'est un processus qui permet d'ajouter de nouveaux composants, de supprimer des composants déjà existants ou de modifier les liens entre les composants suivant les changements du contexte.

c. Commandes contextuelles :
Selon le contexte d'exécution, une même commande peut produire des résultats différents.

d. Déclencheurs contextuels :
Ce sont des simples règles SI-ALORS utilisées pour spécifier comment les systèmes utilisant le contexte seront adaptés.

Une autre forme de catégorisation du contexte a été proposée par Chen et Kotz [Chen, 2000]. En effet, ils introduisent la notion d'activité et de passivité pour la prise en compte du contexte ; par conséquent, ils définissent deux catégories d'applications sensibles au contexte :

- Utilisation du contexte actif : adapter automatiquement une application à un nouveau contexte en changeant le comportement de cette application.
- Utilisation du contexte passif : une application propose un contexte (nouveau ou modifié) à l'utilisateur et/ou sauvegarde ce contexte pour une éventuelle utilisation ultérieure.

De sa part, Pascoe [Pascoe, 1998a] a proposé une autre catégorisation des applications sensibles au contexte ; cette catégorisation s'appuie sur l'identification des principales caractéristiques et des facteurs qui influent la prise en compte du contexte dans le développement des applications. Ces caractéristiques sont :

- Perception (ou Capture) contextuelle : c'est la possibilité de détecter les informations contextuelles et de les présenter à l'utilisateur.
- Adaptation contextuelle : c'est la possibilité d'exécuter ou de modifier automatiquement un service en s'appuyant sur le contexte courant.
- Découverte des ressources contextuelles : les applications utilisant le contexte devront être capables de localiser et d'exploiter les ressources et les services supposés pertinents au contexte de l'utilisateur.
- Augmentation contextuelle : c'est la possibilité d'associer des données numérisées avec les éléments du contexte de l'utilisateur.

Pour résumer les différentes catégorisations des applications utilisant le contexte, nous présentons, ci après, les trois catégories proposées par Dey et Abowd [Dey, 1999b] ; ces catégories montrent les caractéristiques et les fonctions essentielles qu'une application context-aware doit vérifier.

a. présentation de l'information et des services :
Consiste à présenter les éléments du contexte (y compris les services) à l'utilisateur sous forme d'information.

b. exécution automatique d'un service :
Décrit les applications qui déclenchent une commande ou reconfigurent le système relatif à l'utilisateur en tenant compte des changements du contexte.

c. étiquetage du contexte pour une future réutilisation :

Les applications associent (ou attachent) les données capturées aux informations pertinentes du contexte, c'est-à-dire qu'elles doivent créer des étiquettes (labels) de correspondance qu'on peut faire appel pour une éventuelle réutilisation à l'avenir.

2.2.4 Architecture générale d'une application sensible au contexte

En se basant sur l'étude de l'architecture générale du projet SECAS (Simple Environment for Context Aware Systems), on peut spécifier les cinq étapes nécessaires à la sensibilité au contexte [Chaari, 2005].

a. Capture du contexte :
Les informations contextuelles sont capturées à l'aide de senseurs physiques et sont, ensuite, traduites en données brutes. Celles-ci ne seront pas directement utilisables par l'application.

b. Interprétation du contexte :
Consiste à transformer les données capturés (sous une forme brute) en éléments plus faciles à exploiter par l'application. Par exemple, convertir les coordonnées géographiques (GPS) d'un endroit en une adresse postale.

c. Stockage du contexte :
Les valeurs des informations contextuelles sont stockées et échangées à l'aide de représentations XML. Chaari, Laforest et Flory [Chaari, 2005] proposent cinq axes pour modéliser le contexte (mode de communication, utilisateur, terminal, localisation et environnement) ; un élément XML correspondra à chaque axe et l'ensemble des paramètres constituant chaque axe est défini par le concepteur de l'application en respectant des grammaires standards (comme CC/PP [Indulska, 2003] pour les axes utilisateur et terminal). Le concepteur doit définir la structure la structure de chaque paramètre (ensembles, valeurs atomiques…) [Chaari, 2005].

d. Dissémination du contexte :
Le contexte est un ensemble de plusieurs éléments disjoints. Ces éléments peuvent être utilisés séparément par les applications sensibles au contexte. Les paramètres contextuels pertinents sont transmis pour chaque service de l'application. Ensuite, les services choisissent ceux (les paramètres contextuels) qui modifient leurs comportements et leurs méthodes de dissémination (synchrone ou asynchrone) [Chaari, 2005].

e. Adaptation au contexte :
L'adaptation des services au contexte concerne trois niveaux différents de l'application [Chaari, 2005] :

- Adaptation de contenu : flux de données entre les services et l'utilisateur
- Adaptation de présentation : interface utilisateur
- Adaptation de comportement : fonctionnement des services offerts à l'utilisateur

2.3 Perception et capture du contexte (sensing the context)

La manipulation de n'importe quel outil exige l'existence et la disponibilité de cet outil. De même, pour que les applications soient en mesure de bien prendre en compte les changements du contexte, il faut leur fournir tous les éléments susceptibles d'influencer le contexte. C'est-à-dire qu'il faut d'abord capturer et acquérir ces éléments pour ensuite permettre aux applications de les utiliser. Pour cela, plusieurs mécanismes ont été élaborés pour capturer et percevoir le contexte courant et le présenter sous forme d'information.

Parmi ces mécanismes, nous présentons ceux établis par Chen et Kotz dans [Chen, 2000] :

2.3.1 Perception de l'endroit

Le lieu de l'utilisateur est un élément très important du contexte du moment que le déplacement de cet utilisateur d'un emplacement vers un autre implique automatiquement un changement du lieu et par conséquent la modification du contexte. Pour capturer l'endroit, on peut utiliser un système qui va garder la trace des mouvements de l'utilisateur dans l'espace. Par exemple, on oblige l'utilisateur à insérer son badge (ou carte d'identité magnétique) dans le lecteur approprié ou bien appuyer à l'aide d'un doigt sur une zone de reconnaissance automatique des empreintes. Une autre solution pour déterminer la localisation d'un utilisateur consiste à utiliser le Global Positioning System (GPS) qui peut nous fournir les coordonnées spatiales de n'importe quel endroit. L'inconvénient du système GPS est qu'il présente des insuffisances et des défaillances lorsqu'il s'agit de localiser des utilisateurs se trouvant à l'intérieur des constructions parce que le signal est si faible qu'il ne peut s'infiltrer ou traverser les obstacles solides telles que les murs des constructions. Pour remédier au problème de localisation des endroits clos, les chercheurs ont essayé d'expérimenter de nouvelles solutions telles que l'utilisation des ondes infrarouges dans les systèmes Active Badge d'Olivetti [Want, 1992] et ParcTab de Xerox [Want, 1996] et le projet Cyberguide [Abowd, 1997b]. D'autres firmes ont utilisé les émetteurs-récepteurs basés sur les fréquences radio comme c'est le cas chez AT&T [Asthana, 1994] et Hewlett Packard [Hull, 1997]. Notons aussi l'existence d'autres modes de localisation tels que les cameras vidéos, l'ultrason, les sols sensibles, les radars, les domaines des réseaux,…etc.

2.3.2 Perception des contextes de bas niveau

On appelle contexte de bas niveau toute information contextuelle qui soit dans un état brut et qui peut être perçu ou mesuré directement ; par exemple, on cite le temps, la lumière, les sons, la température,…etc.
Le temps : cette information contextuelle est généralement obtenu à partir de l'heure système de la machine, c'est le cas dans les systèmes Active Badge [Want, 1992], ParcTab [Want, 1992] et Cyberguide [Abowd, 1997b]. Néanmoins, il existe d'autres formes de temps à percevoir telles que les jours de la semaine, les jours du mois, les mois de l'année, les quatre saisons, les zones du faisceau horaire, … etc.
Les objets voisins : tous les objets et les personnes proches peuvent influencer l'environnement de l'utilisateur et par conséquent modifient le contexte d'utilisation; d'où, le besoin d'enregistrer, périodiquement, la variation de l'état de ces objets (y compris les personnes).

L'orientation : la direction est aussi une information contextuelle très importante qu'il faut mesurer pour, ensuite, la fournir aux applications. Par exemple, le système Newton MessagePad [Schmidt, 1998] utilise un simple senseur de direction basé sur deux commutateurs de mercure.

Il existe d'autres éléments du contexte qu'il faut capturer à l'aide de techniques et d'outils appropriés. A cet effet, on utilise la photodiode pour détecter le niveau de la lumière, on fait appel à deux accéléromètres pour nous fournir les mesures de l'inclinaison et de la vibration, on peut placer des senseurs infrarouges pour détecter la présence humaine, on met un microphone pour capter les sons, ainsi que des senseurs pour mesurer la température, la pression, volume de gaz dégagé…etc.

2.3.3 Perception des contextes de haut niveau

Les contextes de haut niveau sont étroitement liés à l'activité courante de l'utilisateur et à son comportement social. Dans ce cas, on peut capturer ces modes de contextes en utilisant un système de surveillance (par vision) se basant sur la technologie de la vidéo (camera) associée aux méthodes de traitement d'images. On peut, aussi, consulter le calendrier de taches d'un utilisateur pour extraire ses activités prévues à un temps donné, mais cette solution renferme un inconvénient majeur qui surgit lorsque l'utilisateur lui-même ne respecte pas son calendrier ou bien omet ou néglige de planifier quelques taches.

Une autre solution pour la perception des contextes de haut niveau consiste à appliquer les techniques de l'intelligence artificielle sur les informations contextuelles déjà capturées. C'est-à-dire que nous pouvons reconnaître un contexte complexe à partir de la combinaison des autres senseurs de contextes de bas niveau [Schmidt, 1998]. Ainsi, le projet TEA [TEA, 1998] montre comment peut-on savoir si un téléphone ou un PDA (Personnal Digital Assistant) se trouve dans la main de l'utilisateur, sur la table ou bien dans la valise, et ce, en utilisant trois senseurs (un pour la lumière et deux axes X et Y pour l'accélération); dans ce cas, les signaux rentrant sont assemblés à l'aide d'un réseau de neurone appelé « Kohonen Self-Organizing Map (KSOM) » pour avoir, à la sortie, un résultat bien ordonné qui sera traité et classé par une machine d'état fini (probabiliste) [Van Laerhoven, 1999]. Les systèmes à base de règles rencontrent des difficultés relatives aux frontières des objets, les modèles indéfinis, ambiguïté,…etc. Pour le projet TEA, par exemple, on ne peut savoir si le téléphone se trouve dans la valise ou bien dans la poche de l'utilisateur.

2.3.4 Perception des changements du contexte

La liste des informations contextuelles proposées aux applications ne doit pas être figée mais, par contre, elle doit être actualisée après chaque modification du contexte. Il ne suffit pas, donc, de capturer les changements du contexte, mais d'essayer, aussi, de les prévoir en utilisant les techniques de sondage ou de simulation. Le sondage permet de prévoir le délai et la périodicité pour lesquels la liste des contextes sera mise à jours. Brewington et Cybenko [Brewington, 2000] ont développé un model formel pour afficher les informations contextuelles, et ils utilisaient les délais des informations observées précédemment pour estimer le rythme des changements, et par conséquent, le système peut décider sur le temps idéal auquel il faut actualiser la liste des informations.

2.4 Modélisation du contexte

La modélisation des informations du contexte est une formalisation de données pour exprimer ces informations sous des formes bien adaptées (des modèles) pour être utilisées ou étudiées.
La diversité des propriétés des éléments du contexte entraîne une grande variété de modèles, la majorité d'eux se penche vers la modélisation des informations basée sur l'endroit, d'autres font appel aux structures de données pour modéliser ces informations.

2.4.1 Modèles liés à l'endroit

Ce sont les modèles qui permettent de manier facilement la mobilité des objets dans l'espace et d'avoir la localisation plus ou moins exacte de ces objets.

Modèle symbolique : l'endroit est représenté par des symboles abstraits ou des noms ;
Exemple : un système basé sur l'accès par carte magnétique (Active Badge).
Modèle géométrique : l'endroit est représenté par des coordonnés géométriques ;
Exemple : Global Positioning System (GPS).
Modèle combiné : l'endroit peut être représenté, à la fois, par des noms symboliques et des coordonnés géométriques ; les deux représentations peuvent se convertir l'une à l'autre par l'application de prédicats prédéfinis [Leonhardt, 1998].

2.4.2 Modèles liés aux structures de données [Strang, 2004], [Chaari, 2005]

Paires (clé, valeur) : consiste à associer une valeur ou un attribut à chaque clé pour former une paire qui servira à la modélisation des informations contextuelles. Par exemple, Spreitzer [Spreitzer, 1993] fait correspondre une variable d'environnement (clé) à une variable qui contient le contexte courant (valeur). Aussi, le « context toolkit » de Dey [Dey, 2001] utilise cette approche.

Production (ou Génération) d'étiquette : consiste à utiliser des étiquettes et des champs correspondants, ces mêmes champs peuvent contenir d'autres étiquettes et les champs correspondants, ce qui ramène le système vers un déroulement récursif.

Les RDF (Ressource Description Framework) : la représentation la plus connue dans cette approche est une extension du profil W3C CC/PP [Indulska, 2003] proposé par Held dans [Held, 2002].

Les modèles graphiques : par exemple l'utilisation des diagrammes UML indiqués dans [Bauer, 2003].

Modèles orienté objet : un objet est caractérisé par ses états et ses méthodes. Les informations contextuelles seront exprimées par les états de l'objet qui nous fournira les méthodes appropriées pour pouvoir accéder et modifier ces états. Ces modèles sont, par exemple, utilisés dans le projet TEA [TEA, 1998]

Modèles basé sur la logique : les données du contexte sont exprimées sous formes de membres (ou facteurs) dans un système à base de règles. L'avantage est qu'il est possible d'ajouter de nouvelles règles à la base de données à chaque besoin. Un exemple de ces modèles est proposé dans [McCarty, 1993] pour formaliser le contexte.

Les ontologies : par exemple, le modèle CoOL [Strang, 2003] qui représente le contexte comme un ensemble d'entités ayant des aspects décrivant leurs caractéristiques.

2.5 Adaptation des applications au contexte

L'adaptation d'une application au contexte implique l'adaptation des éléments qui la composent, c'est-à-dire essayer d'adapter le comportement, le contenu et la présentation (figure 9).

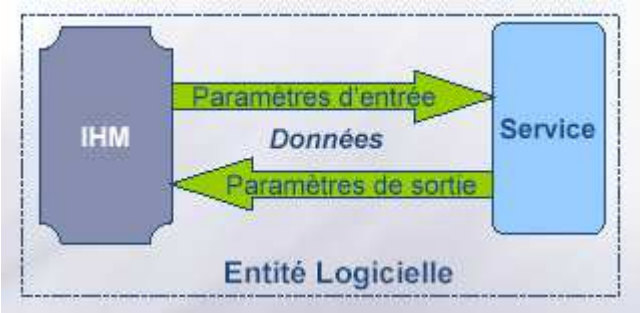


Figure 9. Vue « utilisateur » d'une application [Chaari, 2005]

2.5.1 Adaptation de comportement (services)

Le comportement peut être représenté par un graphe de dépendance où les nœuds symbolisent les services et les arcs désignent les relations entre les services. L'adaptation de comportement consiste à transformer un graphe de dépendance de services en un autre graphe de dépendance de services adaptés. Cette transformation porte sur les nœuds ainsi que sur les arcs. Une solution [Chaari, 2005] est d'intégrer une couche d'adaptation sur les services de l'application en utilisant plusieurs opérateurs comme les opérateurs de filtrage (*projection, sélection, augmentation, union*), les opérateurs sur les instances de services (*SélectionVersion, AjoutVersion*) ou les opérateurs d'adaptation du graphe (*VerrouillerService, AjouterService, InsérerService*).

2.5.2 Adaptation de contenu (données)

C'est la modification de quelques propriétés des informations (les données) proposées à l'utilisateur de façon à satisfaire ses préférences. Par exemple, on peut citer :

- **a.** L'adaptation de format : modifier complètement ou partiellement le format d'une donnée (ex. réduire le nombre de couleurs d'une image)
- **b.** La traduction : convertir un texte suivant les préférences de l'utilisateur
- **c.** La compression de données : peu être une compression brute (ex. zip), une compression multimédia (ex. jpeg) ou une compression sémantique (ex. résumé d'un texte).
- **d.** La décomposition de données : sélectionner seulement une partie d'un objet
- **e.** L'agrégation de données : c'est le rassemblement de plusieurs données différentes. Elle peut être de deux types : agrégation spatiale (ex. un objet image plus un objet texte) ou agrégation temporelle (ex. montage de séquences vidéos).

2.5.3 Adaptation de présentation (IHM)

Consiste à assouplir l'interface utilisateur de façon à ce qu'elle soit maniable plus facilement et selon les préférences de cet utilisateur. Par exemple, si l'utilisateur souhaite des formes d'affichages spécifiques (ex. les courbes graphiques au lieu des données tabulaires). L'adaptation de présentation consiste aussi à fournir des facilités de navigation entre les entités logicielles (services+interfaces+données) pour pouvoir bénéficier de tous les services de l'application [Chaari, 2005].

2.6 Présentation du context toolkit de « Dey »

2.6.1 Notions requises (Besoins)

L'utilisation du contexte requiert la compréhension de quelques notions essentielles. Nous citons ci-dessous les plus importantes [Dey, 2001]:

Séparation des préoccupations **:** consiste à faire la distinction entre la façon d'acquérir le contexte et la façon de son utilisation. Cette distinction permettra aux applications d'exploiter les éléments du contexte sans avoir besoin de connaître les détails des senseurs ainsi que la manière de capture des informations contextuelles.

Interprétation du contexte : l'information contextuelle est initialement capturée sous une forme brute. Donc, pour être exploitable par les applications, cette information doit être interprétée c'est-à-dire qu'elle soit sous forme de données formalisées ou modélisées.

Modes de communications distribuées et transparentes **:** L'informatique ubiquitaire repose sur le principe de fonctionnement de systèmes largement distribués et très hétérogènes. Dans ce cas, le traitement du contexte sera confronté au problème de la diversité des senseurs et des capteurs installés sur différents sites distants. Pour assurer ces modes de communications distribuées, il faut mettre en place un dispositif adéquat basé sur les réseaux de communication et qui garantira une certaine transparence quant aux senseurs et aux applications. (Ex. : les protocoles, utilisation de décodeurs, synchronisation d'horloges,…).

Disponibilité absolue du contexte **:** les applications ne savent pas, à priori, le moment dont elles auront besoin d'une information contextuelle ; par conséquent, les composants qui capturent le contexte devront s'exécuter indéfiniment pour être apte à fournir cette information à chaque demande si besoin.

Sauvegarde et historique du contexte **:** les composants qui capturent le contexte doivent sauvegarder, d'une façon périodique, les changements de l'information contextuelle ; cela permettra de constituer un fichier historique du contexte et qui peut être présenté sous forme d'une boite de dialogue. L'historique du contexte peut permettre, aussi, de prévoir les valeurs futures du contexte.

Découverte de ressources **:** un mécanisme de découverte des ressources est indispensable pour les différentes applications. En effet, ces applications doivent être en mesure de connaître la nature de l'information contextuelle capturée par les senseurs ainsi que sa localisation et le mode de communication. Ceci peut être assuré, par exemple, à l'aide du nom de la machine hôte (hostname) ou le port de la machine exécutant le senseur de contexte. Le mécanisme se chargera de trouver tous les composants capables de capturer le contexte et de fournir les chemins d'accès correspondants.

2.6.2 Composants du toolkit

En général, la sensibilité au contexte doit passer par trois étapes [Dey, 2001] :

- Capturer le contexte
- Interpréter le contexte capturé pour avoir des données exploitables par l'application

- Fournir les informations interprétées à l'application

Pour garantir le cheminement de ces trois étapes, dey [Dey, 2001] a conçu son toolkit à l'aide des éléments suivants :

Les widgets de contexte **:** ce sont des composants logiciels qui peuvent fournir les informations contextuelles capturées par les senseurs aux applications, et ce, à partir de leur environnement d'exécution. Les widgets récupèrent les informations sur l'état de l'environnement via les senseurs (en entrée).

Les interpréteurs **:** on utilise les interpréteurs de contexte pour convertir les informations brutes en données exploitables par les applications; c'est-à-dire hausser le niveau d'abstraction des données. Par exemple, tout endroit peut être repéré à l'aide des coordonnées géographiques (bas niveau d'abstraction) ou bien en connaissant l'adresse postale et le nom de la rue (haut niveau d'abstraction).

Les agrégateurs **:** ce sont des éléments qui permettent de réunir les multiples parties de l'information contextuelle. L'agrégation est applicable surtout lorsque les senseurs sont distribués. L'information capturée par les différents senseurs est acheminée par les widgets de contexte avant d'être agrégée dans le but de fournir une information harmonieuse relative au contexte capturé.

Les services **:** ce sont des composants chargés de l'exécution des actions. Ces actions peuvent intervenir après l'opération d'acquisition du contexte (via les widgets, les interpréteurs et les agrégateurs). Les services de contexte assurent le contrôle et la modification des informations de l'état de l'environnement (en sortie).

Les découvreurs **:** consistent à assurer un suivi permanent des capacités disponibles et offertes par les différents composants (widgets, interpréteurs, agrégateurs et services). Ainsi, toutes les applications peuvent consulter (à l'aide de requêtes) les découvreurs pour rechercher un composant particulier, ces applications utilisent aussi les découvreurs pour localiser un composant de contexte dont elles ont besoin.

2.6.3 Fonctionnement du toolkit

Comme décrit précédemment, le toolkit « boite à outils » de dey comprend cinq composants qui assurent des fonctions distinctes. Les widgets de contexte acquiert l'information contextuelle. Les interpréteurs transforment et augmentent le niveau d'abstraction de l'information capturée et peuvent combiner de divers éléments du contexte pour produire une donnée contextuelle de haut niveau. Les agrégateurs collectent et réunissent les informations contextuelles relatives à une entité particulière pour faciliter son exploitation par les applications. Les services utilisent le contexte capturé pour assurer l'exécution des actions et des comportements relatifs à l'environnement. Enfin, les découvreurs proposent et fournissent aux applications toutes les informations possibles concernant la disponibilité des ressources et des composants de l'environnement.

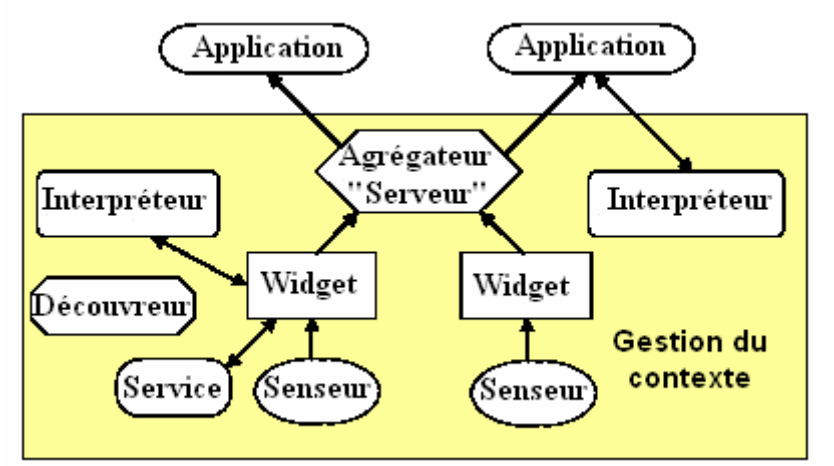


Figure 10. Exemple d'architecture du toolkit de dey [Dey, 2001]

La figure 10 présente un exemple qui utilise les différents composants du toolkit de dey. En effet, il permet la mise en place de deux senseurs, deux widgets, un agrégateur (serveur), deux interpréteurs, un service, un découvreur et deux applications.
Le fonctionnement du toolkit débute lorsqu'un élément de contexte surgit, cette nouvelle opportunité (modification de l'environnement) est sauvegardée par un découvreur. Celui-ci prévient les agrégateurs pour aller chercher les widgets et les interpréteurs appropriés, et invite les applications à choisir les meilleurs agrégateurs, les widgets et les interpréteurs.
Le senseur fournit les données au widget qui sauvegarde le contexte, le widget fait appel à un interpréteur pour convertir ces données et en obtenir des informations de haut niveau d'abstraction, ce qui rendra l'information contextuelle plus disponible et mieux exploitable par les applications ou les autres composants. L'agrégateur recueille les parties du contexte à partir des différents widgets et prépare les réponses appropriées aux requêtes formulées par les applications [Dey, 2001].
Pour mettre en évidence l'intérêt et l'importance du toolkit de dey dans la pratique du développement des applications, nous présentons, ci-dessous, deux exemples d'architectures d'applications utilisant cette boite à outils (figure 11, figure12):

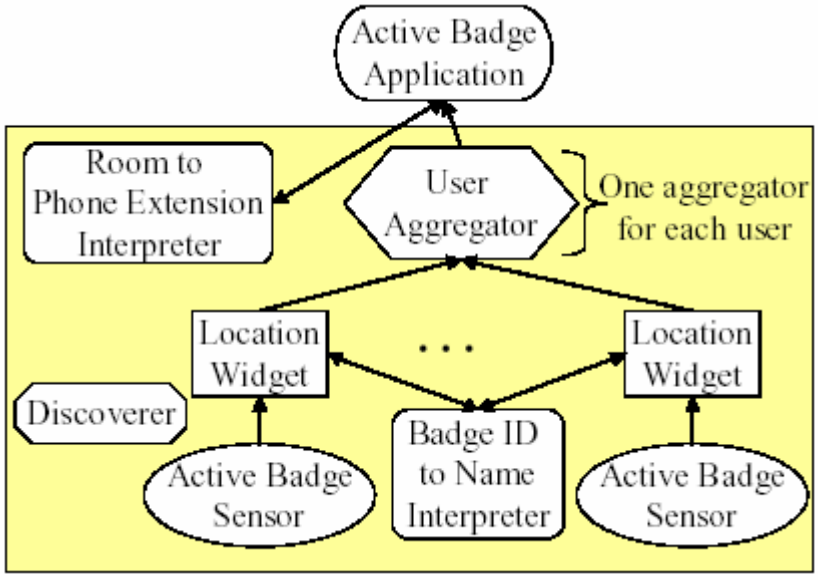


Figure 11. Architecture d'une application Active Badge Call-Forwording [Dey, 2001]

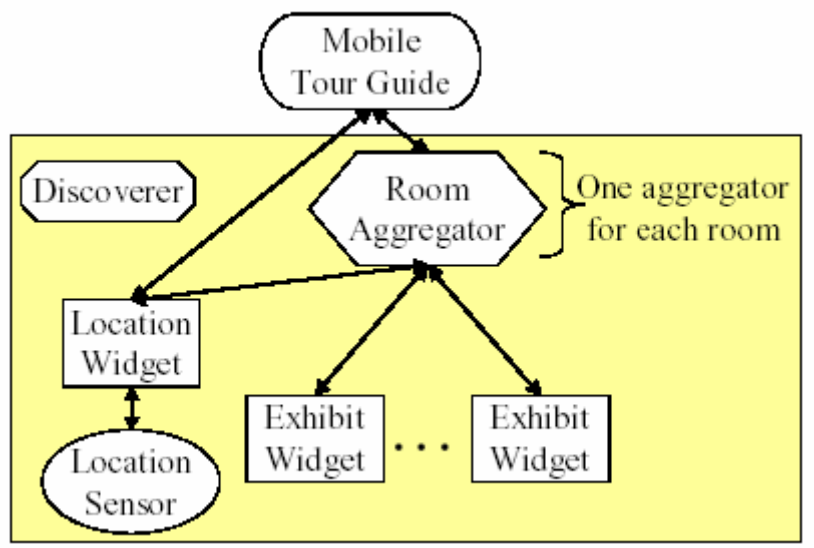


Figure 12. Architecture d'une application Mobile Tour Guide [Dey, 2001]

Conclusion

Ce deuxième chapitre nous a permis d'étudier, de comprendre et de maîtriser la notion de contexte d'utilisation. En effet, nous avons collecté la majorité des définitions et des catégorisations présentées par d'autres auteurs. Cette étude nous a permis, par conséquent, de faire ressortir tous les éléments qui constituent le contexte d'utilisation ainsi que tous les facteurs qui peuvent influencer la situation courante d'un utilisateur. Ce chapitre nous a aidé à mieux cerner le sujet à étudier pour tenter d'apporter un plus dans le domaine de la prise en compte du contexte d'utilisation.

CHAPITRE 3

L'APPROCHE MDA

Introduction

Le développement des logiciels a été, pendant plusieurs années, un champ d'utilisation des paradigmes de composants et de l'orienté objet. Mais ces technologies ne semblent pas offrir les meilleures solutions pour bien assurer la création et la maintenance de ces systèmes logiciels. La complexité de ces logiciels a poussé la communauté du génie logiciel vers le retour à l'utilisation d'un ancien paradigme basé sur les modèles mais avec une nouvelle notion représentée par l'approche de l'ingénierie dirigée par les modèles (MDE : Model Driven Engineering). A cet effet, l'OMG (Object Management Group) a défini l'approche MDA (Model Driven Architecture) (figure 13).

L'approche dirigée par les modèles (MDA) est une spécification basée sur un ensemble de standards tels que MOF (Meta Object Facility), UML (Unified Modeling Language), XMI (XML Metadata Interchange) et CWM (Common Warehouse Metamodel) . la MDA a été conçu pour élaborer, visualiser, échanger, transformer et stocker des modèles de logiciels compréhensibles par la machine, développés indépendamment des technologies d'implémentation, et séparant les contraintes techniques des contraintes fonctionnelles.

La MDA propose plusieurs outils qui ont pour objectif de manipuler des modèles logiciels, de vérifier leur cohérence, de les raffiner, pour finalement les transformer automatiquement en squelettes de code, tests d'intégrations et scripts de déploiement, pour des plates-formes diverses (J2EE, .NET, CORBA, …).

Le principe de base de l'approche MDA est la séparation entre les aspects métiers et les aspects technologiques. Cette séparation représente l'apport majeur de cette approche ; néanmoins, elle présente d'autres avantages tels que la portabilité, l'interopérabilité, l'évolutivité, et la documentation afin d'augmenter la productivité dans le processus de développement logiciel.

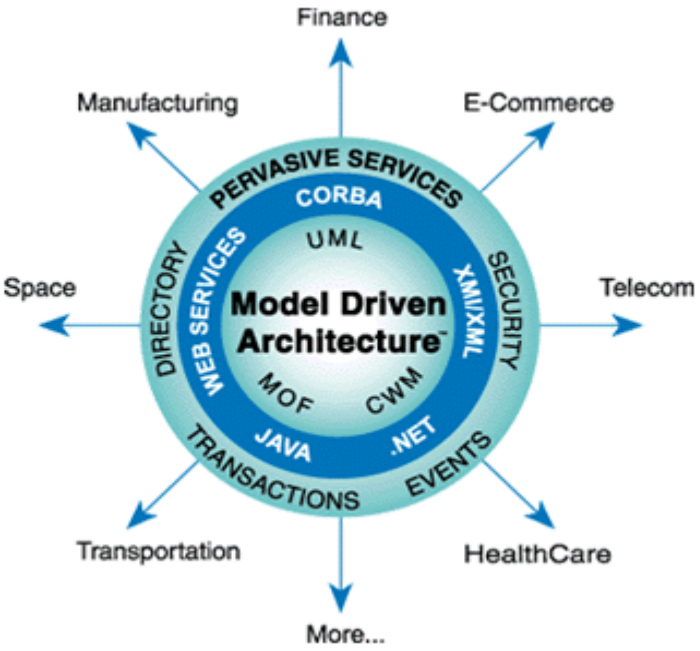


Figure 13. Architecture de la MDA [OMG, 2003b]

3.1 Les niveaux d'abstraction

La MDA présente une architecture basée sur quatre niveaux : M0, M1, M2 et M3 (figure 14):

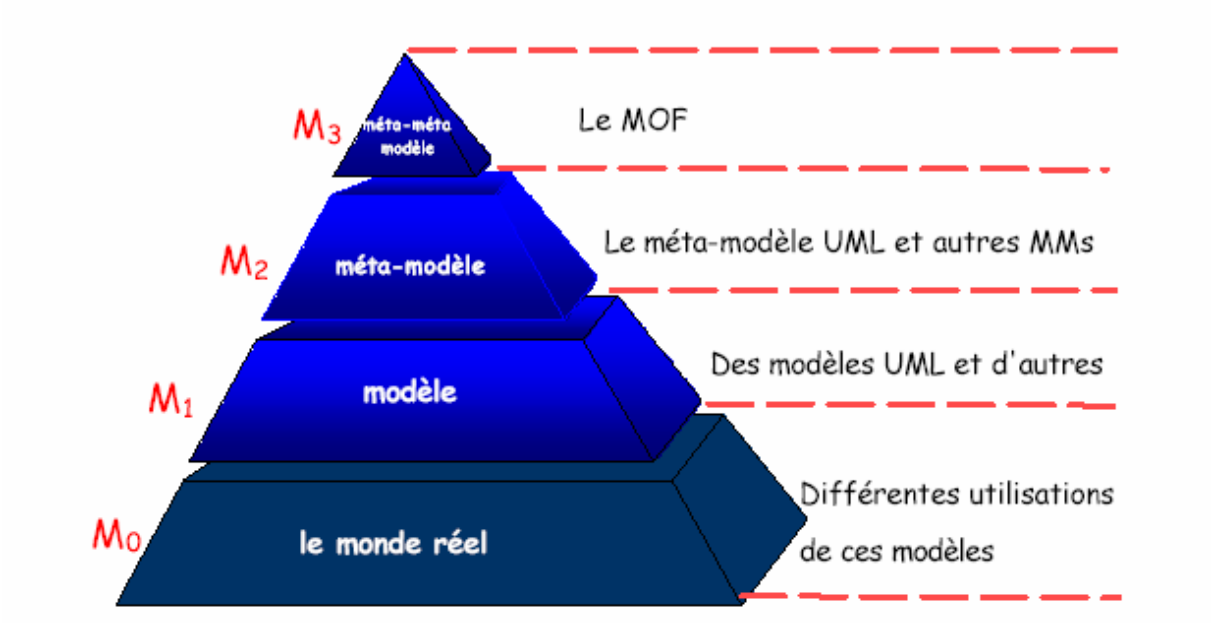


Figure 14. Architecture à quatre niveaux

- Le niveau M0 représente les différents objets de l'univers de discours ou bien les informations du monde réel. Ce niveau M0 correspond au niveau concret, aux instances des objets qui sont utilisées dans le système.
- Le niveau M1 représente les différents modèles de chaque univers de discours par la description et la spécification du système. Un modèle peut être décrit par la

combinaison de dessins et de textes où le texte est exprimé par un langage de modélisation spécifique ou bien en langage naturel. Le niveau M1 correspond aux entités utilisées pour modéliser les instances présentes dans un système : les classes, leurs attributs, leurs opérations, et leurs associations avec d'autres classes.

- Le niveau M2 représente les méta-modèles spécifiques à chaque domaine : un méta-modèle pour chaque domaine d'intérêt pertinent pour les modèles de niveau M1. Le niveau M2 comprend les entités utilisées pour décrire une application orientée objet. On trouve, par exemple, les notions de classe, de relation, de méthode et d'attribut.
- Le niveau M3 correspond au méta-méta-modèle, c'est à dire le modèle utilisé pour décrire tout méta-modèle et doit être conçu pour permettre la définition de tous les concepts nécessaires pour la modélisation de méta-modèles et pour leur unification dans un cadre de référence commun. Un méta-méta-modèle est indépendant du domaine, il contient des méta-caractéristiques pour des méta-modèles spécifiques. Dans le monde MDA, ce niveau d'abstraction est dénommé MOF (Meta Object Facility).

3.2 Concepts de base de la MDA

La MDA utilise plusieurs concepts et notions qui représentent le noyau de cette approche. Ces concepts sont définis comme suit [OMG, 2003b] :

- Le système : un système peut être un programme, une entreprise, un groupe de personnes,…etc.

- Le modèle : un modèle est une description ou une spécification formelle d'une fonction, structure et/ou comportement d'une application ou d'un système.

- Dirigé par les modèles (Model Driven) : une approche est dite dirigée par les modèles si elle s'appuie sur la force des modèles pour développer un système, et par conséquent elle permet de guider la compréhension, la conception, la construction, le déploiement, la maintenance et la modification des systèmes développés.

- L'architecture : en général, l'architecture d'un système est une spécification des parties et des liens du système et les règles d'interaction des parties utilisant les liens. L'architecture MDA est un cas particulier qui propose l'utilisation de certains types de modèles et comment ces modèles seront construits ainsi que les relations qui existent entre les différents types de modèles.

- Le point de vue (Viewpoint) : un point de vue sur un système est une technique d'abstraction qui utilise un groupe de concepts et de règles dans le but de se concentrer sur un aspect particulier de ce système. La MDA spécifie trois points de vue sur un système :

- Computation independent viewpoint : se focalise sur l'environnement et les besoins du système sans se préoccuper ni de sa structure ni de son fonctionnement.
- Platform independent viewpoint : se focalise sur les opérations du système (aspect fonctionnel) sans rentrer dans les détails d'une plate-forme particulière.
- Platform specific viewpoint : applique la combinaison du platform independent viewpoint avec les détails d'utilisation d'une plate-forme spécifique.

- La vue : une vue (ou modèle de point de vue) d'un système est une représentation de ce système à partir de la perspective d'un point de vue choisi.

- La plate-forme : une plate-forme est une union de sous-systèmes et de technologies qui fournit un ensemble de fonctionnalités à travers des interfaces et des patterns spécifiques. Chaque application de la plate-forme doit pouvoir utiliser ces outils sans être contraint de connaître le mode d'implémentation de ces fonctionnalités.

- L'application : une application fait référence à une fonctionnalité en cours de développement. Un système peut, donc, être décrit par une ou plusieurs applications supportées par une ou plusieurs plates-formes.

- L'indépendance de plate-forme : chaque modèle doit vérifier la qualité de l'indépendance de la plate-forme, c'est-à-dire que ce modèle soit indépendant des caractéristiques de la plate-forme.

- Le modèle de plate-forme : un modèle de plate-forme fournit un ensemble de concepts techniques qui représentent les différents composants de cette plate-forme et les services fournis par cette même plate-forme.

- La transformation de modèle : c'est le processus de conversion d'un modèle vers un autre modèle du même système.

- Le mapping : c'est la spécification d'un mécanisme pour transformer les éléments d'un modèle conforme à un méta-modèle vers des éléments d'un autre modèle conforme à un autre (peut être le même) méta-modèle.

- L'implémentation : une implémentation est une spécification qui fournit toutes les informations nécessaires à la construction d'un système et à le rendre opérationnel.

3.3 Modèles de la MDA

3.3.1 Computation Independent Model (CIM)

Aussi appelé modèle de domaine ou modèle métier, le CIM représente une vue d'un système à partir du computation independent viewpoint. Le modèle CIM ne montre pas les détails de la structure du système, par contre il fait ressortir des informations sur l'environnement et sur les besoins de ce système. Ce modèle correspond à la phase de capture des besoins.

3.3.2 Platform Independent Model (PIM)

C'est une vue d'un système à partir du platform independent viewpoint dans laquelle sera présenté l'aspect fonctionnel de ce système. Un PIM doit vérifier la contrainte de l'indépendance de plate-forme pour qu'il puisse être utilisé par différentes plates-formes. Ce modèle correspond à une phase d'analyse et de conception.

3.3.3 Platform Specific Model (PSM)

C'est une vue d'un système à partir du platform specific viewpoint. Un modèle PSM combine les spécifications du PIM avec les détails qui spécifient comment un système utilise une plate-forme particulière. Ce modèle correspond à une phase de conception et d'implémentation.

3.4 Le processus MDA

Principes

Pour pouvoir appliquer le processus de l'approche MDA, il faut satisfaire les conditions suivantes :

- Disposer de modèles de haut niveau consistants, précis et complets
- Disposer d'un langage de haut niveau pour écrire ces modèles
- Disposer de règles de transformation
- Disposer d'un langage pour écrire ces règles
- Disposer d'outils d'exécution de ces règles
- Disposer d'outils de transformation des modèles de haut niveau en modèles d'exécution

Fonctionnement

Une fois ces conditions remplies, une approche MDA de développement logiciel peut être appliquée, et son déroulement passe par quatre étapes majeures :

a. **Génération du Computation Independent Model (CIM)** :

Cette première étape correspond à la phase de capture de tous les besoins du système et de son environnement. Pendant cette étape, on ignore la structure et les opérations de fonctionnement du système.

b. **Construction du modèle PIM** :

Le PIM est un modèle de haut niveau d'abstraction et doit être indépendant de toute technologie d'implémentation. Pendant cette étape, on spécifie la nature des fonctionnalités et les types d'opérations sans préciser les détails techniques suivant lesquels le système va utiliser une plate-forme particulière. A ce niveau, on utilise UML comme langage de modélisation.

c. **Transformation du modèle PIM vers un modèle PSM** :

Le PIM construit est stocké en format MOF qui sera l'entrée de l'étape de mapping. Et c'est cette étape de mapping qui permettra de construire un PSM après transformation du PIM. L'opération de transformation peut être faite d'une façon complètement manuelle ou automatique à l'aide de patterns ou d'algorithmes spécifiques. Pendant cette étape, on choisit une plate-forme particulière pour introduire ses spécifications techniques dans l'opération de transformation.

d. **Génération du code** :

Pendant cette étape, on génère le code à partir du PSM en utilisant des outils spécifiques de transformation, et on crée, ainsi, l'implémentation de ce PSM pour la plate-forme choisie.

3.5 Approches de transformation de modèles

Généralement, ces approches sont utilisées lors des transformations PIM-PSM.

3.5.1 Marquage

Le principe de cette approche consiste à (figure 15):

- **a.** Choisir une plate-forme
- **b.** Si cette plate-forme ne possède pas de mapping, alors procéder à sa construction et sa préparation (le mapping doit contenir des annotations ou marques)
- **c.** Utiliser ces annotations pour marquer (choisir) les éléments du modèle PIM à transformer
- **d.** Transformer le PIM marqué pour produire un modèle PSM.

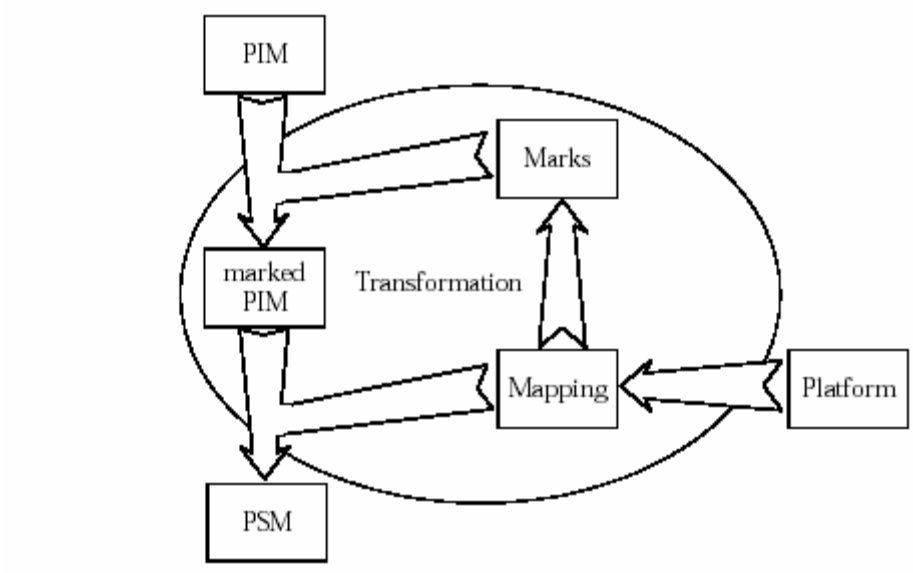


Figure 15. Transformation par marquage [OMG, 2003b]

3.5.2 Transformation de méta-modèle

Les étapes de cette approche sont (figure 16):

- **a.** Préparer un modèle en utilisant un langage indépendant de la plate-forme spécifié par un méta-modèle
- **b.** Choisir une plate-forme particulière
- **c.** Si on ne dispose pas des spécifications de transformation de cette plate-forme, alors on doit les préparer
- **d.** Les spécifications de transformation représentent le mapping entre les méta-modèles
- **e.** Transformer le PIM vers un PSM en utilisant le mapping des méta-modèles.

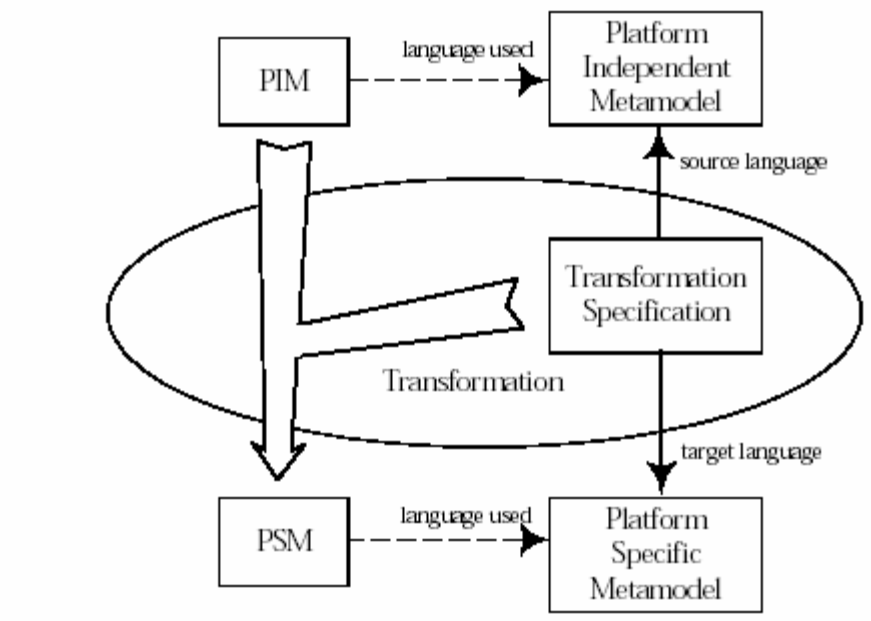


Figure 16. Transformation de méta-modèle [OMG, 2003b]

3.5.3 Transformation de modèle

Cette technique consiste à (figure 17):

- **a.** Préparer un modèle en utilisant des types (logiciels) indépendants de la plate-forme
- **b.** Les éléments du PIM sont les sous-types des types indépendants de la plate-forme
- **c.** Choisir une plate-forme particulière
- **d.** Si on ne dispose pas des spécifications de transformation de cette plate-forme, alors on doit les préparer
- **e.** Les spécifications de transformation représentent le mapping entre les types indépendants de la plate-forme et les types dépendants de la plate-forme
- **f.** Transformer le PIM vers un PSM en utilisant le mapping des types
- **g.** Les éléments du PSM sont les sous-types des types spécifiques de la plate-forme.

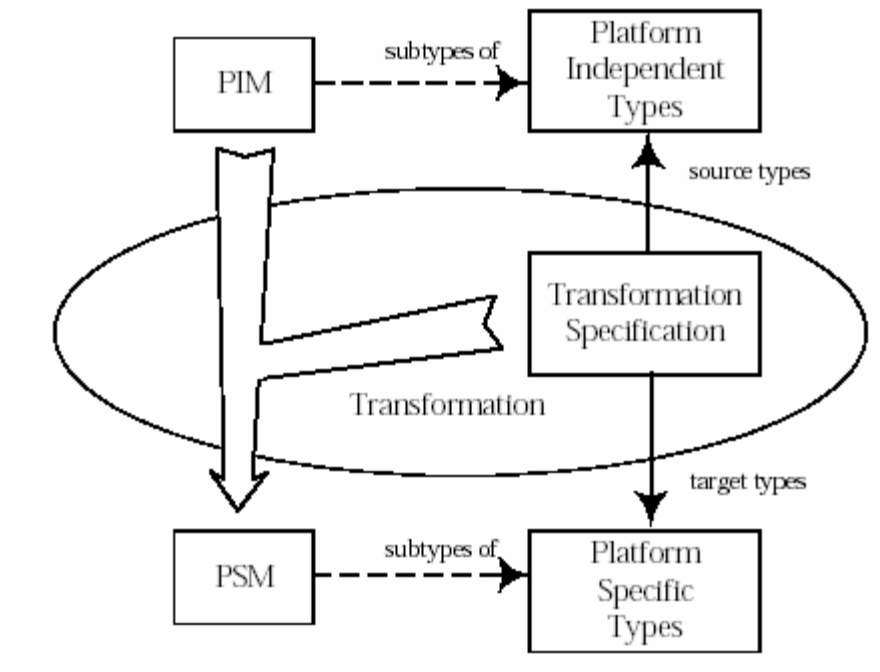


Figure 17. Transformation de modèle [OMG, 2003b]

3.5.4 Application de pattern

Cette approche représente une extension des deux approches précédentes. Le principe est le même sauf que des patterns sont associés aux types pour former le mapping et définir, ainsi, les spécifications de transformation (figures 18 et 19).

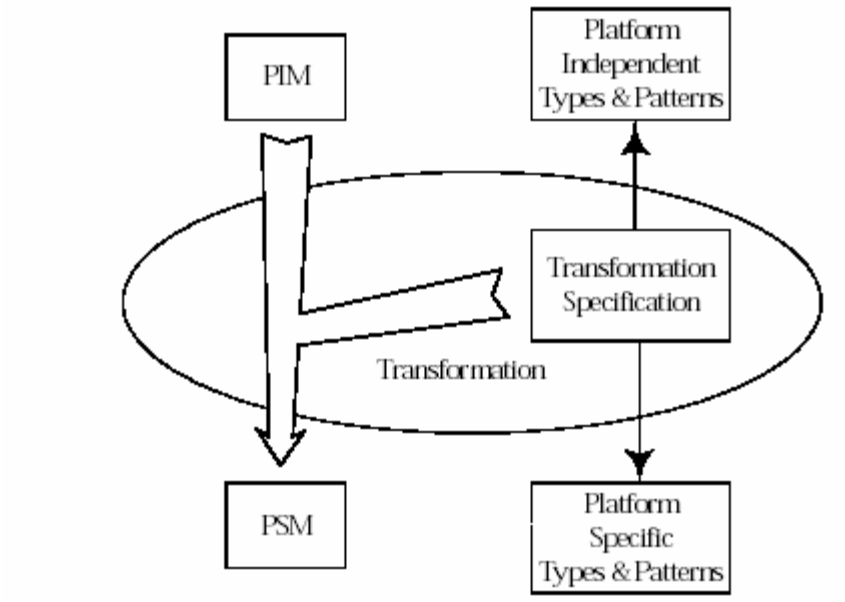


Figure 18. Application de pattern [OMG, 2003b]

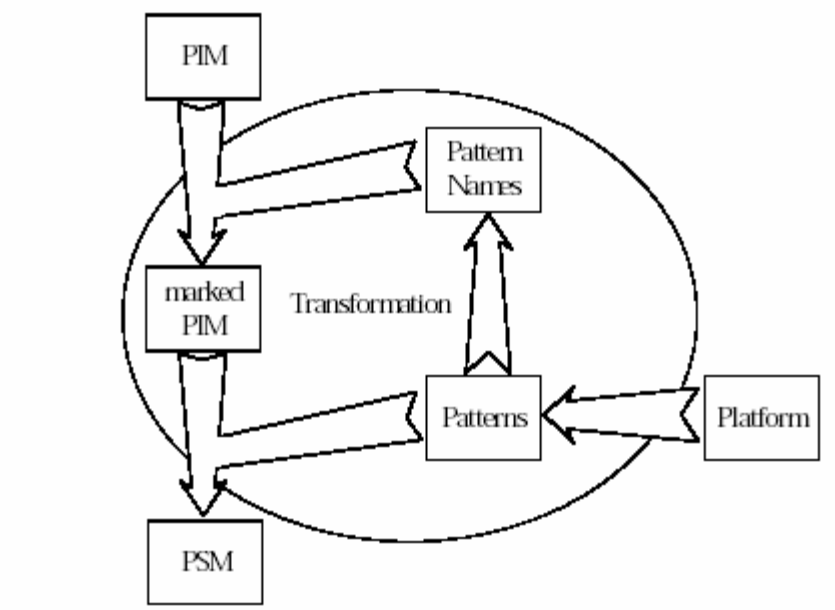


Figure 19. Autre possibilité d'utiliser les patterns [OMG, 2003b]

3.5.5 Fusion de modèles

Cette approche de transformation consiste à (figure 20):

- **a.** Définir et concevoir un modèle en utilisant une approche de modélisation particulière,
- **b.** Combiner le PIM avec le nouveau modèle en appliquant une opération de fusion, pour obtenir un modèle fusionné,
- **c.** Choisir une plate-forme particulière
- **d.** Si on ne dispose pas des spécifications de transformation de cette plate-forme, alors on doit les préparer
- **e.** Les spécifications de transformation représentent le mapping entre les modèles
- **f.** Transformer le modèle fusionné, obtenu lors de l'étape (b.), vers un PSM en utilisant le mapping des modèles.

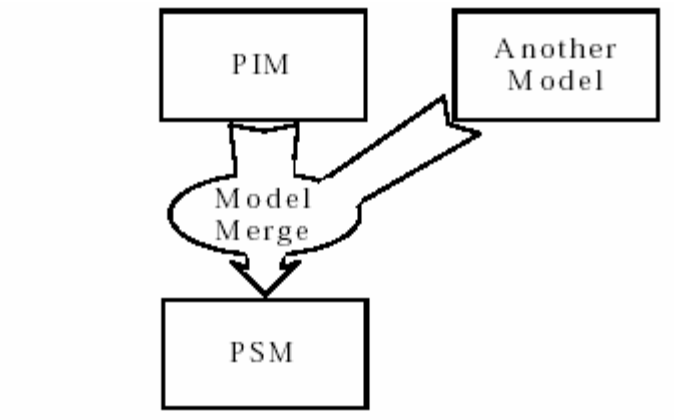


Figure 20. Fusion de modèles [OMG, 2003b]

3.5.6 Information supplémentaire

Cette approche permet d'introduire de nouvelles informations pendant la phase de transformation PIM-PSM. Ces informations peuvent porter sur des connaissances du domaine de l'application ou bien sur des connaissances relatives à la plate-forme (figure 21).

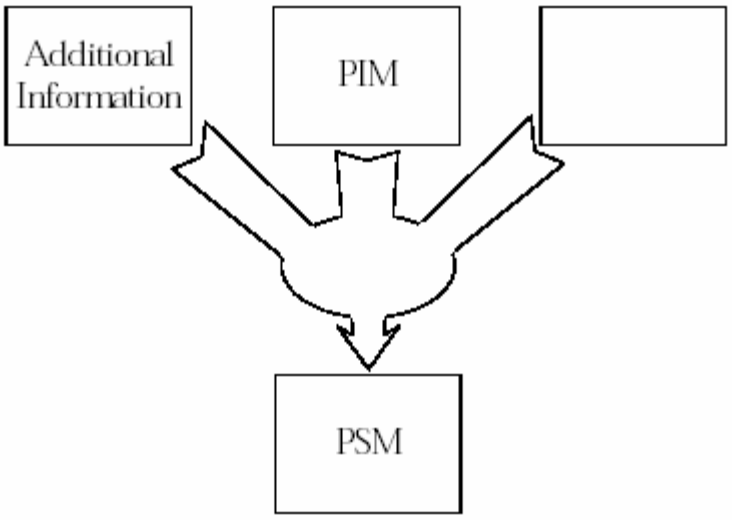


Figure 21. Information supplémentaire [OMG, 2003b]

3.6 Méthodes de transformation de modèles

La transformation PIM-PSM requiert la présence et la disponibilité d'un nombre considérable d'informations. Ces informations sont nécessaires pour accomplir l'opération de transformation et leur introduction peut se faire de plusieurs manières allant du manuel à l'automatique.

On distingue quatre méthodes de transformations de modèles :

a. Transformation manuelle

Toutes les décisions de conception sont établies et introduites d'une façon manuelle, c'est-à-dire que les informations nécessaires à la transformation PIM-PSM sont manuellement exploitées (extraites et introduites) par les concepteurs.

b. Transformation utilisant un profile de PIM

Un PIM peut être établi en utilisant un modèle indépendant en UML profile, et ce modèle peut être transformé vers un PSM exprimé utilisant un second modèle spécifique en UML profile. L'opération de transformation comporte le marquage du PIM en utilisant les marques fournies par le profile du modèle spécifique.

c. Transformation utilisant les patterns et les annotations

Les patterns peuvent être utilisés dans la spécification du mapping. Le mapping comprend un pattern et des annotations correspondantes à quelques éléments de ce pattern. Le principe de cette méthode de transformation consiste à utiliser ces annotation pour préparer (marquer) un PIM. Ensuite, les éléments du PIM marqué sont transformés selon le pattern pour construire le PSM.

d. Transformation automatique

Il existe des cas où le PIM peut fournir toutes les informations nécessaires à l'implémentation ; donc, la génération de code n'aura pas besoin d'opérations supplémentaires comme le marquage par les annotations ou l'introduction d'informations supplémentaires. Dans ce cas, on dit que le PIM est complet et contient tous les outils de transformation qui permettent de le transformer directement vers le code du programme sans intervention du développeur qui n'aura même pas besoin de voir le PSM.

3.7 Standards de la MDA

L'OMG fournit les principaux standards utilisés par l'approche MDA, à savoir:

a. Unified Modeling Language (UML)

C'est un langage graphique utilisé pour visualiser, spécifier et documenter les systèmes logiciels. La MDA utilise des modèles pouvant être décrits en langage UML.

b. Meta Object Facility (MOF)

C'est un langage standard qui sert à exprimer les méta-modèles en permettant la définition et la manipulation des méta-données. (Une méta-donnée est une donnée qui représente un modèle). L'intérêt principal du MOF est surtout qu'il permet la définition d'opérations globales non spécifiques applicables à tous les méta-modèles.

c. XML Metadata Interchange (XMI)

C'est un standard qui aide à définir, échanger et manipuler les données et les objets XML. Le XMI assure la jonction entre le monde des modèles et le monde XML. XMI se base sur le MOF et permet la génération de schémas XML à partir de méta-modèles.

d. Common Warehouse Metamodel (CWM)

C'est un standard de l'OMG qui traite des entrepôts de données. Il couvre toutes les étapes nécessaires à l'élaboration, à la création et à la transformation des entrepôts de données. CWM est formé d'un ensemble d'interfaces qui servent à échanger les méta-données dans des environnements distribués et hétérogènes.

3.8 Avantages de la MDA

- La séparation entre les spécifications métiers et les spécifications techniques, ce qui assure la protection de la logique métier contre les changements ou l'évolution des technologies
- La portabilité
- L'interopérabilité
- La réutilisation et la composition de modèles rendant possible la construction de systèmes complexes.
- Un modèle PIM peut être transformé vers plusieurs modèles PSM. C'est-à-dire que le même PIM peut être utilisé pour créer plusieurs modèles spécifiques appartenant à des plates-formes différentes.
- La possibilité d'extraire plusieurs vues d'un même système.
- L'évolution simultanée des modèles métiers et des technologies.
- La protection contre les facteurs inhérents au développement manuel de systèmes.
- L'augmentation de l'interaction et de la migration entre les espaces technologiques différents.
- Les nouvelles applications présentent une grande robustesse, une meilleure maintenabilité et une génération de code moins coûteuse.

Conclusion

Dans ce chapitre, nous avons présenté les principaux concepts de l'approche MDA. Nous avons insisté sur l'importance des techniques de transformation et leur rôle primordial dans le processus de développement des nouveaux systèmes informatiques qui sont devenus plus complexes. A cet effet, la MDA peut être une solution adaptée pour gérer la complexité de ces systèmes en proposant plusieurs avantages comme la séparation des aspects, la portabilité, l'interopérabilité,…etc. Nous avons mis l'accent également sur les transformations de modèles.

TRAVAUX VOISINS

Travaux voisins

Plusieurs études ont été menées dans le domaine de recherche sur le contexte d'utilisation et surtout concernant la prise en compte des changements contextuels qui environnent l'exécution d'une application ainsi que les préférences souhaitées d'un utilisateur. Rares sont les travaux qui se sont orientés vers l'utilisation de l'approche MDA pour concevoir des applications adaptées et obtenir ainsi des informations personnalisées.

Dans ce sens, Vale et Hammoudi démontrent dans [Vale, 2008] l'importance de la MDA dans la modélisation du contexte et dans le développement des applications sensibles au contexte, et pour cela, ils ont conçu le métamodel contextuel CSOA (Context-aware Service Oriented Architecture) utilisant les principes de la MDA et se basant sur les viewpoints de EDOC-ECA. Dans l'étude citée dans [Ou, 2006], les auteurs ont conçu un modèle contextuel utilisant les ontologies dans le but de modéliser les informations contextuelles, ensuite ils proposent une nouvelle architecture MDIA (Model Driven Integration Architecture) consacrée à l'implémentation des applications sensibles au contexte.

La majorité des autres travaux considèrent l'application comme un produit fini sur lequel seront appliquées les différentes méthodes de contextualisation, néanmoins les outils et les approches élaborés représentent une issue inévitable pour la compréhension des principes de l'adaptation contextuelle des applications et des systèmes d'information.

Par exemple, A.K. Dey présente dans [Dey, 2001] une boite à outils « context toolkit »qui peut servir comme support pour adapter les applications au contexte. En effet, il propose trois étapes nécessaires pour la sensibilité au contexte (capturer le contexte, l'interpréter et le fournir à l'application)

Une autre étude présentée par Chen [Chen, 2000] consiste à capturer les différentes informations contextuelles qui vont être traitées et modélisées (modèle de localisation, structures de données). Les auteurs proposent deux types d'approches, l'une basée sur une architecture centralisée utilisant un serveur de contexte centralisé qui fournit les informations contextuelles aux applications, et l'autre repose sur une architecture distribuée dans laquelle l'information contextuelle, requise par les applications, est hébergée sur plusieurs sites distribués.

Le projet PUMAS (Peer Ubiquitous Multi-Agent System) est un framework basé sur les agents (un agent est une entité autonome et proactive qui fournit à un utilisateur des fonctions spécifiques). PUMAS sélectionne de l'information recherchée à partir d'une ou plusieurs sources d'information, et l'adapte aux caractéristiques de l'utilisateur et à celles du DM (dispositif mobile) [Chaari, 2005]

Le projet SECAS (Simple Environment for Context Aware Systems) s'intéresse à l'adaptation des applications au contexte d'utilisation (préférences et environnement de l'utilisateur, terminal utilisé,…). Son objectif est de rendre les applications adaptables aux différents contextes d'utilisation sur leurs trois composantes : données, services et présentation. Pour assurer une telle adaptation, ce projet fait appel aux services Web [Chaari, 2005].

Par notre étude, nous ésperons apporter des contributions qui permettent de faire évoluer les méthodes de développement d'applications sensibles au contexte, et par conséquant, facilter l'adaptation des sytèmes d'information ubiquitaires. Notre étude tente de démontrer l'importance de la séparation entre trois types d'aspects, à savoir : les aspects métiers, les aspects contextuels et les aspects techniques. Notre objectif est de construire un modèle contextuel à partir des contraintes contextuelles précédemment séparées, et de l'intégrer, ensuite, dans le processus de développement de l'approche MDA en utilisant l'opération de fusion de modèles.

Partie II

Contributions

CONTEXTE VS MDA

Comment les systèmes d'information peuvent-ils prendre en compte le contexte d'utilisation?

En général, l'adaptation des systèmes d'information aux préférences de l'utilisateur ou au contexte d'utilisation se base sur l'accouplement des sources d'information (systèmes d'information ubiquitaires) et des informations contextuelles (contexte d'utilisation). Les sources d'information sont exprimées sous forme de fichiers XML. De même, le contexte d'utilisation est décrit par des fichiers XML. Ces fichiers XML seront traduits en faits utilisables pour effectuer des raisonnements à base de règles. Une autre copie de ces fichiers XML sera stockée pour former des bases de connaissances qui serviront pour la mise en œuvre de l'adaptation. Généralement, la prise en compte du contexte d'utilisation et des préférences utilisateurs par les systèmes d'information ubiquitaires doit passer par plusieurs étapes :

- **a.** Capture de l'information contextuelle (collecte d'informations via les capteurs et les senseurs).
- **b.** Traitement de l'information capturée (interprétation, stockage, …).
- **c.** Modélisation des données et formulation des règles contextuelles.
- **d.** Adaptation des systèmes d'information en utilisant les nouveaux modèles de données (on fera appel à une approche de conception des SI sur laquelle seront insérés les nouveaux modèles de données).

La majorité des travaux menés dans le domaine de la contextualisation [Dey, 2001], [Chen, 2000] appliquent l'adaptation contextuelle sur les applications déjà conçues et finies. Notre étude (Figure 22) se distingue de ces travaux par le fait que les informations contextuelles seront prises en compte lors des phases de conception et du développement de l'application et non pas après [Benselim, 2009a].

Opportunités de l'approche MDA

Nous allons utiliser une approche basée sur les modèles (MDA : Model Driven Architecture) [OMG, 2003b] comme approche de conception. La MDA consiste à appliquer des transformations successives sur les modèles (CIM – PIM – PSM) pour aboutir à la génération du code exécutable d'une application [OMG, 2003b]. La notion de "contexte d'utilisation" n'est pas clairement prise en charge par le cycle de vie de la MDA même si dans la définition du modèle CIM [OMG, 2003b] on fait inclure les informations portant sur l'environnement, mais ceci demeure insuffisant, d'une part, pour faire face aux gigantesques variations subies

par cet environnement à cause des grandes opportunités offertes par l'informatique nomade, et d'autre part, pour répondre à bien aux préférences des utilisateurs de plus en plus exigeants.

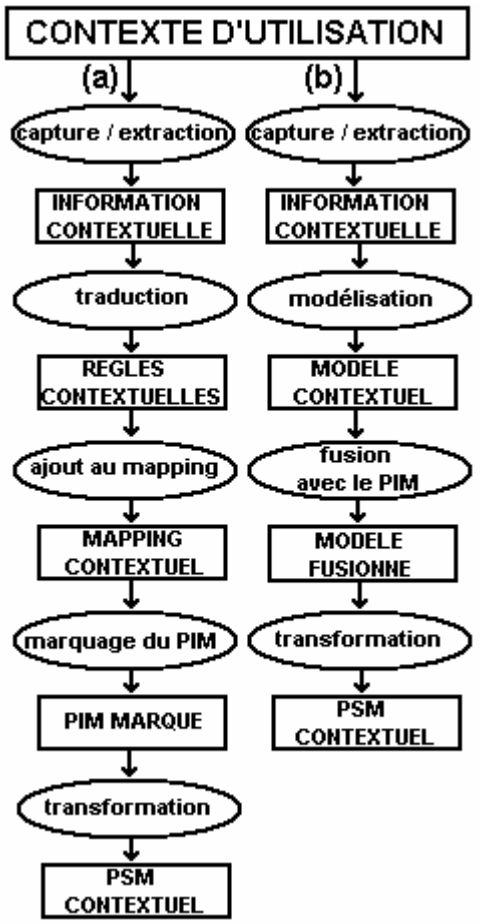


Figure 22. Architecture proposée [Benselim, 2009a]

Pour cela, nous envisageons d'introduire ou bien d'insérer les informations contextuelles dans le cycle de vie de la MDA, cette insertion peut se faire à plusieurs niveaux [Benselim, 2009a]:

a. Lors de l'élaboration du modèle CIM:

On peut augmenter la définition du modèle CIM [OMG, 2003b] pour englober l'aspect contextuel de la situation courante (application et utilisateur). Cette solution est réalisable par la substitution du terme "environnement" dans la définition du CIM par le terme "contexte d'utilisation" qui réunira, en plus de l'environnement, l'application et l'utilisateur. Cette substitution permet de définir clairement chacun des paramètres du contexte d'utilisation et d'inclure l'aspect comportemental et les préférences d'un utilisateur. Par exemple, les concepteurs des systèmes d'information peuvent introduire, dès la préparation du modèle CIM, des variables (éléments du modèle CIM) réservées pour les paramètres régionaux (langue, date, heure, nombres, unités de mesure,…) détectables automatiquement selon la localisation de l'utilisateur (élément de l'environnement). L'environnement, dans ce cas, ne garantit pas la personnalisation des résultats car si cet utilisateur est un touriste étranger qui utilise son PDA à partir de la localisation détectée et qui présente un profil différent (langue, mode d'affichage,..), il ne sera pas satisfait.

b. Lors de la transformation PIM-PSM (Figure 22):

Pour insérer les informations contextuelles, on peut utiliser les différentes approches (ou techniques) de transformations des modèles MDA proposées par l'OMG dans [OMG, 2003b].

Parmi ces techniques nous pouvons choisir deux qui conviennent le plus (simples, claires, faciles à mettre en œuvre,…) à nos fins; et qui sont :

- Technique de marquage de modèles [OMG, 2003b]: c'est une technique qui consiste à transformer un modèle source (PIM) vers un modèle résultat (PSM) en utilisant les annotations. Celles-ci sont extraites à partir du "Mapping" (ensemble de règles de transformation relatives aux contraintes technologiques d'une plateforme), et sont, ensuite, appliquées sur les éléments du PIM. On obtient ainsi un nouveau modèle appelé "PIM marqué" qui renferme les éléments marqués (transformables) et non marqués du PIM initial. Enfin, chacune des règles du "Mapping" est appliquée sur l'élément marqué correspondant (appartenant au PIM marqué) pour avoir un élément du PSM.
- Technique de fusion de modèles [OMG, 2003b]: cette technique consiste à fusionner le modèle à transformer (PIM) avec un autre modèle (à définir) pour obtenir un modèle résultat (PSM).

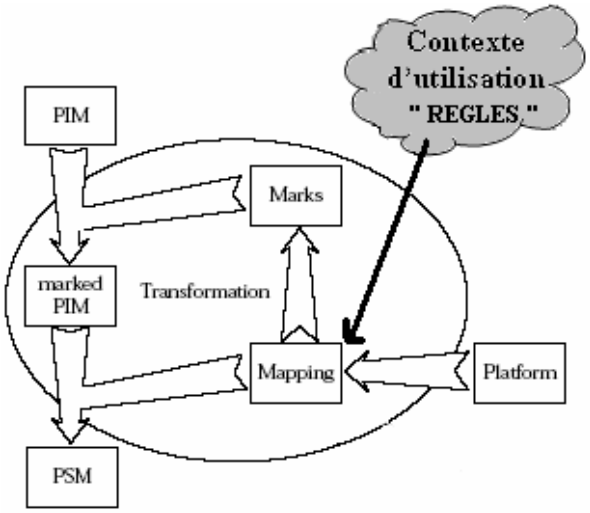


Figure 23. Architecture proposée d'une transformation par marquage [Benselim, 2009a]

Solutions envisagées

Dans la figure 22, nous présentons l'architecture de notre proposition et nous montrons les étapes des deux variantes possibles (a) et (b) pour la prise en compte du contexte par la MDA.
(a) En utilisant la technique de marquage de modèles (Figure 23), nous proposons que l'ensemble des règles de transformation "Mapping" puisse être enrichi par les informations contextuelles (sous forme de règles contextuelles), c'est-à-dire que ce "mapping contextuel" (Figure 24) s'alimentera de deux sources différentes à savoir la plate-forme et le contexte d'utilisation. A partir du mapping contextuel, on fait ressortir les annotations (ou marques) qui seront utilisées pour marquer les éléments du PIM à transformer. On obtient alors un "PIM marqué" sur lequel on appliquera les règles du mapping contextuel (plate-forme et contexte d'utilisation) du même système pour avoir enfin un modèle PSM ayant les spécifications techniques de la plate-forme et aussi les spécifications contextuelles et comportementales du système.

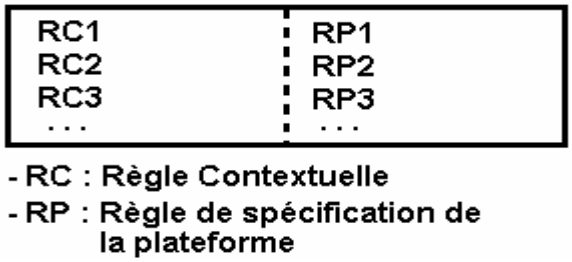


Figure 24. Contenu du "mapping contextuel"

(b) En utilisant l'autre technique de transformation qui consiste à fusionner le modèle PIM avec un autre modèle, nous proposons que cet autre modèle soit un nouveau modèle contextuel qui contiendra toutes les informations contextuelles du système. Nous envisageons, donc, d'ajouter une étape préliminaire qui sera consacrée à la modélisation du contexte et qui aura comme résultat le modèle contextuel (Figure 25).

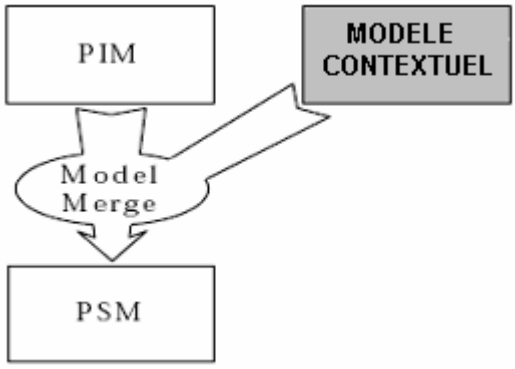


Figure 25. Architecture proposée d'une transformation par fusion [Benselim, 2009a]

Dans ce nouveau modèle contextuel, on va formaliser toutes les contraintes contextuelles d'utilisation et les préférences des utilisateurs en données contextuelles. L'étape suivante consiste à faire la combinaison (fusion) de ce modèle contextuel avec le modèle PIM pour assurer la transformation vers le modèle PSM. Le principe de la fusion des deux modèles est décomposé en quatre phases: [Kolovos, 2006a]

- comparaison: identification des correspondances (accords) entre les éléments des modèles à fusionner
- test: vérification de la conformité des éléments correspondants identifiés dans le but d'éliminer les conflits entre eux
- fusion: application de l'outil de fusion (algorithmes)
- restructuration: nettoyage du modèle obtenu de toutes les inconsistances et les anomalies.

Solution choisie

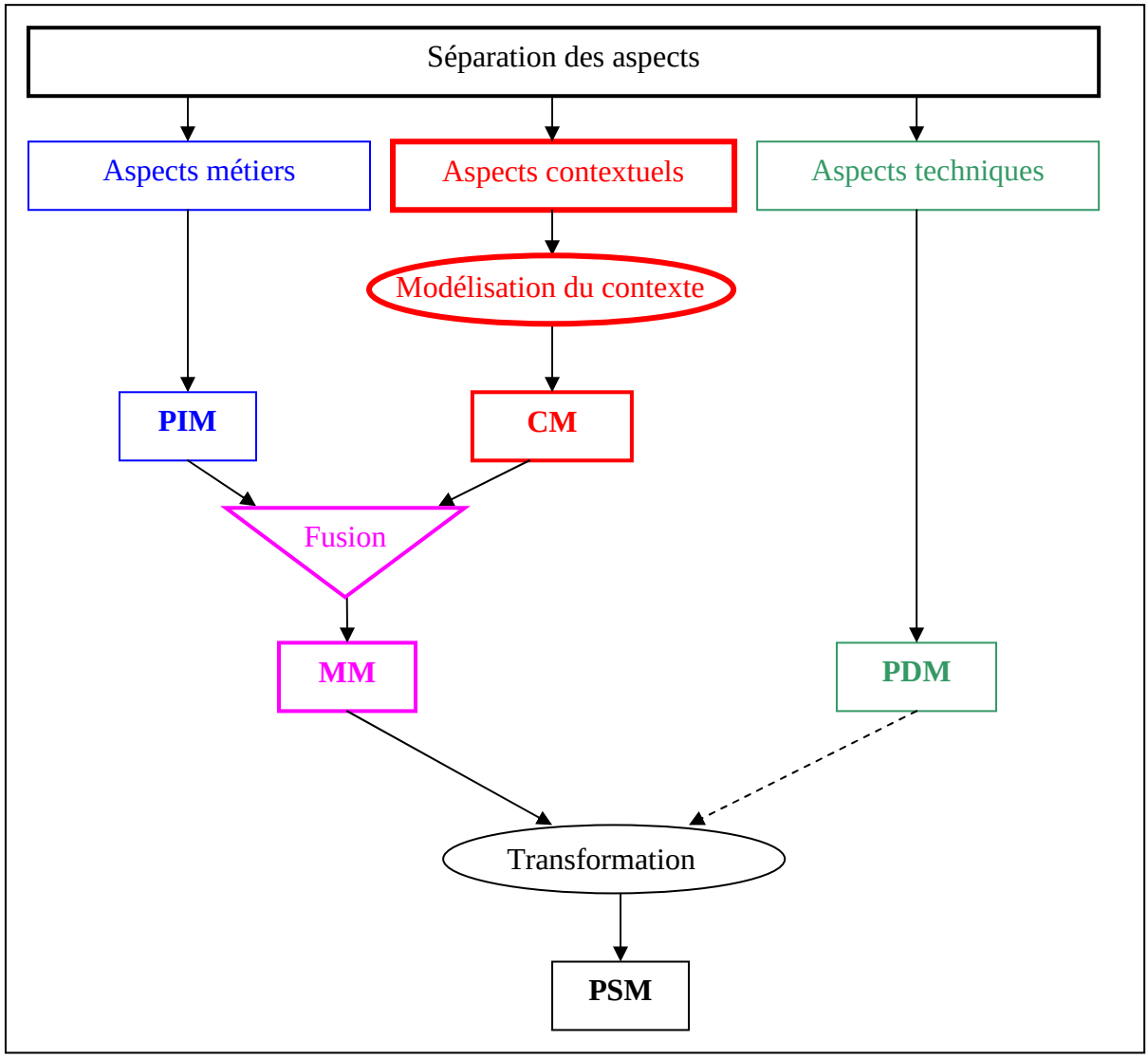
Dans la suite de cette partie, nous présentons les détails de l'approche proposée pour la prise en compte du contexte et des préférences des utilisateurs par les systèmes d'information ubiquitaires. Cette approche [Benselim, 2009b] se base essentiellement sur le principe de la

séparation des préoccupations et plus particulièrement la séparation des aspects contextuels. Ainsi, cette séparation va permettre d'étudier les contraintes contextuelles d'une façon indépendante des autres contraintes imposées au système d'information (contraintes fonctionnelles et contraintes technologiques). La notion d'ubiquité nécessite fortement que le contexte d'utilisation soit étudié séparément pour mieux prendre en charge l'évolution et les changements du contexte d'une part, et d'autre part pour ne pas remettre en cause tout le processus de développement à cause de ces changements perpétuels.
Après séparation de l'aspect contextuel, toutes les informations contextuelles seront représentées à l'aide d'une approche graphique en utilisant le langage de modélisation UML. Ensuite, le modèle contextuel obtenu sera intégré dans le cycle de vie de l'approche MDA en utilisant l'opération de fusion de modèles.

Notre approche peut être résumée en trois étapes [Benselim, 2009b]:

- 1ère étape : séparation des aspects contextuels
- 2ème étape : modélisation du contexte
- 3ème étape : Intégration du modèle contextuel.

La figure 26 illustre la position des étapes de notre approche par rapport à l'architecture générale de l'approche MDA.



PIM : Platform Independant Model
CM : Contextual Model (modèle contextuel)
MM : Merged Model (modèle fusionnée PIM+CM)
PDM : Platform Description Model (modèle de description de la plate-forme)
PSM : Platform Specific Model

Figure 26. Architecture proposée pour l'extension de la MDA [Benselim, 2009b]

Pour faciliter la compréhension de notre proposition, nous avons jugé utile de traiter et de présenter chaque étape de l'approche dans un chapitre à part. Ainsi les trois chapitres qui suivent correspondront aux trois étapes majeures de l'approche proposée :

- Chapitre 4 : Séparation des aspects contextuels
- Chapitre 5 : Modélisation du contexte
- Chapitre 6 : Intégration du modèle contextuel

CHAPITRE 4

SÉPARATION DES ASPECTS CONTEXTUELS

Introduction

Un système se définit par un ensemble hétérogène de caractéristiques. Ces dernières représentent les différents aspects de ce système. Pour l'étudier, il ne doit pas être vu comme étant un ensemble indivisible, parce que cette vue globale est généralement très complexe et difficile à comprendre.

Donc, au lieu d'étudier le système dans son intégralité, il est préférable et plus pratique de le décomposer en plusieurs sous-systèmes simples. Chacun de ces sous-systèmes sera consacré sur l'étude d'un aspect particulier du système et fait abstraction des autres aspects.
La séparation des aspects d'un système permet de développer chacun de ces aspects d'une façon simultanée et indépendante les uns par rapport aux autres et évite, par conséquent, la propagation des problèmes et des difficultés rencontrés par un type d'aspects envers les autres.

Le principe de la séparation des aspects offre une grande souplesse dans le domaine de développement logiciel. En effet, cette séparation permet, comme le cas de la MDA, de définir un modèle métier indépendant de toute plate-forme technique et de générer automatiquement du code vers la plate-forme choisie. Ainsi, le processus de développement ne sera pas mis en cause à cause de l'évolution possible des supports technologiques.

L'apport de l'informatique ubiquitaire demande une étude spécifique et minutieuse des changements subis par le contexte d'utilisation d'une situation. Les éléments de ce contexte doivent être traités indépendamment des autres contraintes parce qu'ils présentent des caractéristiques spécifiques et très variables (mobilité, hétérogénéité, …).

Notre étude propose de créer un nouveau type d'aspects à séparer des autres (métiers et techniques). Ce nouveau type sera consacré pour l'étude des contraintes contextuelles. Ainsi, nous aurons un processus de développement qui s'articule autour de trois branches indépendantes en entrée (branche fonctionnelle, branche technologique et branche contextuelle). A ce propos, nous avons introduit les notions de processus 3TUP (3 Track Unified Process) et le processus de développement en « **psi : Ψ** » pour substituer, respectivement, aux notions de processus 2TUP et le processus de développement en « Y ».

4.1 Rappels sur le processus de développement en Y

Une meilleure description de l'architecture du processus de développement en «Y» a été détaillée par Roques et Valée dans le livre « UML en action » [Roques, 2003] et dont nous présentons quelques notions élémentaires.

4.1.1 Processus de développement logiciel

Un processus de développement représente une séquence d'étapes ordonnées qui coopèrent entre elles dans le but de créer un nouveau système logiciel ou bien de mettre à jour un système existant. L'objectif d'un tel processus est la production de meilleurs logiciels qui satisferont pleinement les besoins des utilisateurs.

4.1.2 Processus Unifié (UP :Unified Process)

Un processus unifié est un type de processus de développement logiciel qui est construit sur UML. Il est itératif et incrémental, centré sur l'architecture, conduit par les cas d'utilisation et piloté par les risques.

La gestion d'un processus UP est organisée suivant les quatre (4) phases suivantes:

- Préétude
- Elaboration
- Construction
- Transition.

Un processus UP doit répondre aux caractéristiques suivantes :

- Il est itératif et incrémental
- Il est piloté par les risques
- Il est construit autour de la création et de la maintenance d'un modèle
- Il est orienté composant
- Il est orienté utilisateur.

Les activités de développement d'un processus UP sont définies par cinq (5) disciplines fondamentales qui décrivent :

- La capture des besoins
- L'analyse
- La conception
- L'implémentation
- Le test et le déploiement.

4.1.3 Le processus 2 Track Unified Process (2TUP)

Un processus 2TUP est un processus UP qui doit satisfaire toutes les contraintes de changements subies par les systèmes d'information de l'entreprise tout en contrôlant les capacités d'évolution et de correction des tels systèmes. D'après l'architecture en «Y», tous les changements imposés à un système informatique peuvent être classés en deux types de contraintes: les contraintes fonctionnelles et les contraintes techniques (figure 27). C'est-à-dire que le processus peut suivre deux chemins de développement indépendants. D'où l'appellation de « 2 Track » qui signifie littéralement « 2 chemins ».

Figure 27. Le système d'information soumis à deux types de contraintes

Le principe du processus 2TUP consiste à traiter toute évolution imposée au système d'information sur deux axes distincts et parallèles qui sont l'axe fonctionnel (aspects métiers) et l'axe technique (aspects technologiques). Ces deux axes qui forment deux branches d'entrée seront fusionnés dans une branche commune qui représente la phase de réalisation du processus global. Cette fusion conduit à l'obtention d'un processus de développement en forme de Y (figure 28) [Roques, 2003].

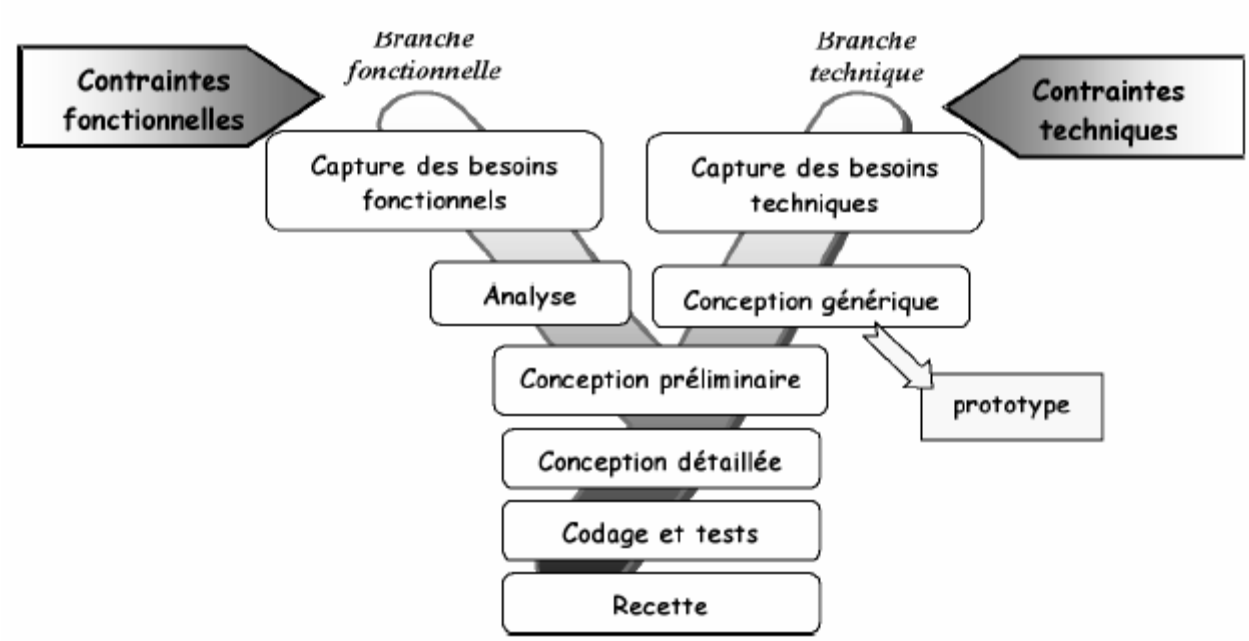


Figure 28. Le processus de développement en « Y » [Roques, 2003]

4.2 Processus de développement en « psi : Ψ »

4.2.1 Le processus 3 Track Unified Process (3TUP)

Le processus 3TUP peut être défini comme suit [Benselim, 2009b]:
Un processus 3TUP est un processus UP qui prend en charge toutes les contraintes de changement subies par les systèmes d'information ubiquitaires tout en assurant le contrôle des capacités d'évolution et de correction de ces systèmes. La particularité majeure des contraintes imposées à un système d'information ubiquitaire réside dans le changement continuel du contexte d'utilisation de la situation courante d'un utilisateur.

Les éléments du contexte d'utilisation (préférences de l'utilisateur, localisation, temps, ressources et environnement) ne peuvent appartenir ni aux spécifications fonctionnelles ni aux spécifications technologiques.

4.2.2 Architecture du processus de développement en psi «Ψ»

Comme illustré en figure 29, le processus de développement en psi «Ψ» s'articule autour de quatre (4) phases:

- Une branche technique
- Une branche fonctionnelle
- Une branche contextuelle
- Une phase de réalisation

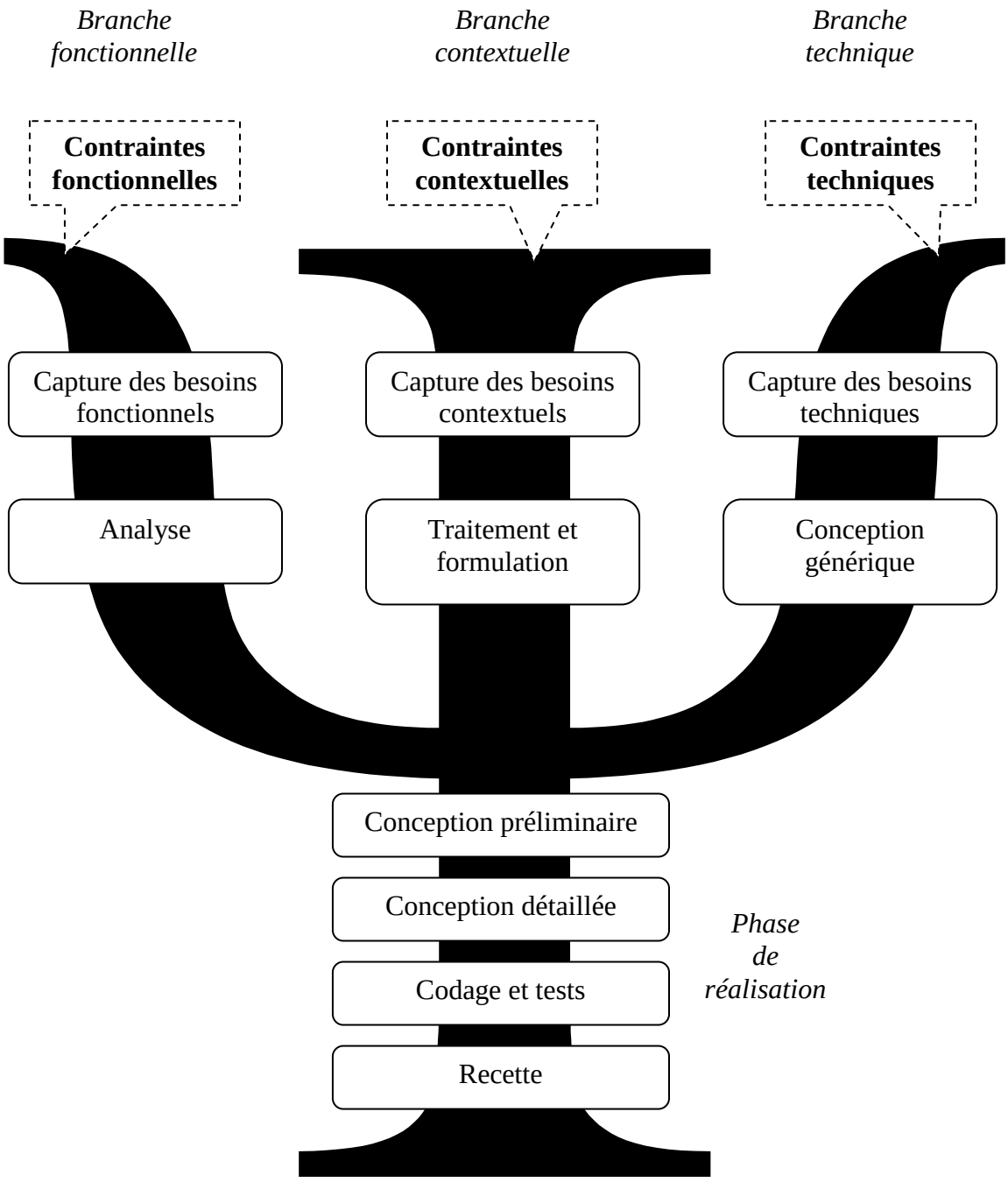


Figure 29. Le processus de développement en psi «Ψ»

4.3 Pourquoi séparer l'aspect contextuel ?

En effet, pour répondre à cette question, il faut démontrer que chacun des éléments contextuels ne peut être traité ni par la branche fonctionnelle ni par la branche technique. Par exemple, dans l'informatique ubiquitaire, les notions de temps et d'espace sont très importantes et elles couvrent l'ubiquité (n'importe où et n'importe quand) des systèmes d'information et des utilisateurs de plus en plus nomades. Aussi, le changement des objets environnants peut affecter directement l'état de la situation courante d'un utilisateur. D'autre part, chaque utilisateur peut présenter des préférences particulières quant au contenu, à la présentation ou à l'affichage des informations demandées.

A partir des définitions précédentes de la notion de « contexte » (chapitre 2), nous pouvons extraire quelques éléments qui constituent les composants principaux du contexte d'utilisation. Ces éléments sont : l'utilisateur, la localisation, le temps et l'environnement. Dans le domaine de l'informatique ubiquitaire, ces éléments présentent une caractéristique commune qui est le changement et la variation, et c'est sur ce point que se base notre démonstration.

a. L'utilisateur

L'entité « utilisateur » inclut l'identité, les activités et le profil (préférences). Ces éléments changent continuellement et sont spécifiques à chaque utilisateur. Celui-ci a besoin de personnaliser ses activités ou bien les informations demandées en présentant quelques préférences ou des souhaits particuliers. Donc, l'entité « utilisateur » ne peut pas être une contrainte fonctionnelle du système ni une contrainte technique.

b. La localisation

La localisation désigne l'endroit et l'emplacement d'une situation particulière. L'endroit est changeable suivant l'espace d'exécution de cette situation et suivant la mobilité des utilisateurs et des ressources. A partir de la localisation, le système peut fournir des valeurs par défaut de quelques paramètres comme la langue, la monnaie, unités de mesure,…et qui ne sont pas spécifiés par l'utilisateur. La variation incessante de l'endroit fait de lui un élément qui ne pas appartenir aux fonctionnalités du système ni, bien sûr, aux spécifications technologiques.

c. Le temps

Cet élément contextuel inclut le temps, la date, les saisons…etc. cet éléments peut fournir plusieurs indications sur le climat, la température, la pression…et qui représentent des informations pertinentes pour les utilisateurs nomades. Le temps est une variable primordiale dans les systèmes d'information ubiquitaires, donc sa prise en charge ne doit pas être effectuée ni parmi les contraintes fonctionnelles ni parmi les contraintes techniques.

d. L'environnement

Cet élément possède une large signification en renfermant les ressources (devices), les réseaux, les personnes avoisinantes ainsi que tout objet capable d'agir ou d'interagir avec l'utilisateur, l'application ou bien les deux. Chacun de ces éléments de l'environnement peut affecter et modifier l'exécution d'une situation suivant ses propres changements.

L'informatique ubiquitaire offre plusieurs possibilités d'accès à l'information comme l'utilisation aléatoire, en temps et en espace, des dispositifs mobiles par des utilisateurs de plus en plus exigeants et nomades. A cause de la mobilité des utilisateurs, non seulement le temps et l'endroit qui changent constamment mais aussi les objets environnants (y compris les personnes avoisinantes). Donc l'utilisateur se trouve confronté à une situation différente, c'est-à-dire dans un nouveau contexte d'utilisation et dont il doit s'y adapter.

La modification ou le changement d'un des éléments du contexte d'utilisation implique la transformation de ce contexte vers un nouvel état de la situation courante de l'utilisateur. La prise en charge de ces changements ne doit pas remettre en cause tout le processus de développement, mais seulement la partie qui traite ce type de contraintes.

D'où l'utilité de regrouper toutes les spécifications contextuelles qui représente le contexte d'utilisation dans une nouvelle branche indépendante appelée : **branche contextuelle**.

Cette séparation de la branche contextuelle permettra de développer l'aspect contextuel d'un système d'une façon indépendante, d'une part, des fonctionnalités propres de ce système et d'autre part, des contraintes technologiques choisies (plate-forme).

Le processus 3TUP propose un cycle de développement en Ψ, qui sépare entre les aspects techniques, les aspects fonctionnels et les aspects contextuels.

Ceci nous amène à répartir les contraintes subies par un système d'information ubiquitaire en trois types distincts (figure 30):

- Contraintes fonctionnelles
- Contraintes techniques
- Contraintes contextuelles

C'est-à-dire que le processus peut suivre trois chemins de développement indépendants. D'où l'appellation de « **3 Track** » qui signifie littéralement « 3 chemins ».

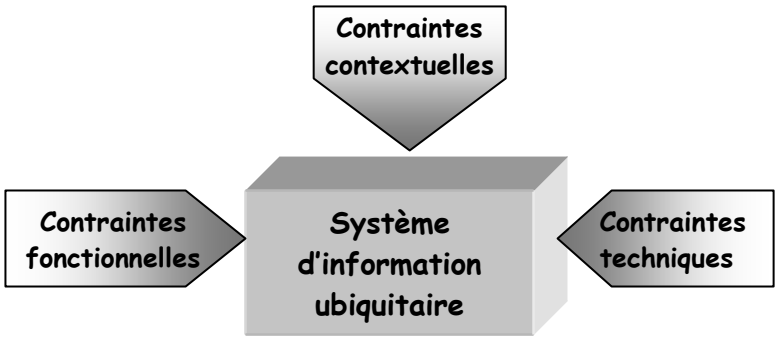


Figure 30. Le système d'information soumis à trois types de contraintes [Benselim, 2009b]

4.4 Description de la branche contextuelle

La branche contextuelle représente l'ensemble des contraintes relatives au contexte d'utilisation d'un système ou d'une application. Cette branche est très cruciale puisque l'informatique ubiquitaire offre de grandes opportunités pour l'utilisation nomade des dispositifs et des services. Ceci aura un impact direct sur le contexte d'utilisation d'une situation quelconque.
Le développement de la branche contextuelle passe par deux étapes majeures : capturer les besoins contextuels et ensuite les interpréter.

a. Capture des besoins contextuels : cette étape consiste à capturer et collecter tous les éléments susceptibles d'influencer sur la situation courante d'un utilisateur. Ces éléments représentent les composants du contexte d'utilisation incluant l'utilisateur, l'application et l'environnement.

b. Traitement et formulation des données : les éléments capturés sont traités et transformés en données conceptuelles. Cette étape sert à préparer les données contextuelles pour être exploitables lors de l'étape de la conception préliminaire (1ère étape de la phase de réalisation).

4.5 Avantages de la séparation de l'aspect contextuel

- Eliminer la dépendance qui peut exister entre les fonctionnalités propres du système et les contraintes contextuelles.
- Prise en compte du contexte d'utilisation indépendamment de l'évolution technologique des plates-formes choisies.
- La possibilité de développement en parallèle de plusieurs axes à la fois.
- L'évolution de la technologie ne peut pas affecter tout le processus de développement.
- Protéger les fonctionnalités du système de l'évolution et des changements subis par le contexte d'utilisation.
- Augmenter le degré de maintenabilité.
- Minimiser les risques de remise en cause de tout le processus de développement.
- Intégration facile des nouvelles contraintes contextuelles qui peuvent apparaître.

Conclusion

Dans ce chapitre, nous avons présenté la première étape de notre approche proposée pour la prise en compte du contexte d'utilisation. Cette étape consiste à séparer et faire isoler les aspects contextuels du système à étudier. La séparation des aspects contextuels permet d'étudier tous les changements des contraintes contextuelles indépendamment des autres contraintes soumises au système (fonctionnelles et techniques). A cet effet, nous avons introduit un nouveau type de processus appelé : 3 TUP, qui considère trois genres d'aspects (métiers, techniques et contextuels). Aussi, nous avons proposé une nouvelle démarche, appelée : « **psi : Ψ** », destinée pour le développement d'applications plus adaptées à l'environnement ubiquitaire.

Le chapitre suivant est réservé à la seconde phase importante de notre proposition dédiée à la modélisation des informations contextuelles.

CHAPITRE 5

LA MODÉLISATION DU CONTEXTE

Introduction

Actuellement, l'utilisation de la notion de « modèle » marque son grand retour avec l'apport de l'ingénierie des modèles (MDE : Model Driven Engineering) qui présente de grandes facilités pour représenter, traiter, stocker et exploiter les informations.

Les informations contextuelles, déjà identifiées par séparation des aspects, doivent être modélisées à l'aide d'un langage de modélisation. Ceci permettra d'avoir un modèle contextuel qui formalise toutes les contraintes du contexte d'utilisation.

Parmi les approches existantes destinées pour la modélisation du contexte, nous avons choisi la modélisation graphique utilisant le langage UML (Unified Modeling Language).

Ce choix est argumenté par les multiples avantages que présente UML comme la disponibilité de notations simplifiées et non ambiguës, la formalisation précise des concepts et la couverture importante des besoins en modélisation.

5.1 La représentation du contexte

Plusieurs auteurs [Bauer, 2003] [Chaari, 2005] estiment qu'actuellement il n'existe pas une méthode standard qui montre comment modéliser un contexte ni comment ce contexte peut-il influencer la conception d'un système d'information.

L'utilisation d'une information contextuelle nécessite sa description et sa représentation. A cet effet, Bauer propose plusieurs possibilités [Bauer, 2003]:

- Le langage humain
- Les diagrammes UML
- La composition de modèles

Pour décrire une information contextuelle, chaque élément contextuel doit faire l'objet d'une description textuelle contenant les informations qui ne peuvent pas être exprimées par des diagrammes, et qui contient aussi :

- Une description générale de la situation

- Les détails de l'information supportée
- Le processus de la distribution de l'information
- Les processus décrivant l'utilisation de l'information

Les procédures qui interviennent et qui sont liées à l'environnement doivent être exprimées par des diagrammes. Le langage UML (Unified Modelling Language) représente le meilleur outil standard pour la modélisation de l'information. UML propose plusieurs diagrammes qui peuvent décrire les différents aspects ou vues d'un système.

La représentation d'un contexte dépend étroitement de son utilisation [Bauer, 2003]. Par exemple, si le contexte est utilisé pour décrire l'état d'une entité alors ce contexte doit être exprimé par un diagramme de classes (représentation par diagrammes statiques). Par contre, si le contexte est utilisé pour spécifier une information contextuelle dépendante de l'interaction de l'utilisateur alors le contexte sera exprimé à l'aide de diagramme approprié (représentation par diagrammes statiques).
Donc les diagrammes UML peuvent être utilisés pour représenter le contexte. Par conséquent, toutes les informations relatives au contexte (composants, procédure, taches, …) peuvent être exprimées et modélisées par l'un de ces diagrammes.

5.2 Méthode d'utilisation du contexte

Généralement, l'utilisation du contexte passe par trois étapes [Bauer, 2003]:

a. Identification et analyse du contexte
Cette étape consiste à décrire clairement les informations contextuelles et de les étudier pour faire ressortir tous les éléments constituants le contexte ainsi que leurs propriétés.

b. Abstraction des informations contextuelles
Les éléments du contexte précédemment identifiés et analysés sont soumis à un processus d'abstraction destiné à mettre en évidence les aspects les plus importants du contexte. Ceci doit tenir compte de quelques paramètres :

- L'information requise par l'utilisateur
  - Structure du modèle : par exemple, le diagramme de classes
  - Type du support d'information : exprimé par un diagramme de séquences
- La présentation de l'information
- La source d'information

c. Modélisation et conception
La modélisation des systèmes d'information sensibles au contexte se base essentiellement sur la notion de contexte. Ces systèmes agissent comme étant un assistant qui est capable de fournir l'information demandée malgré les changements continuels subis par la situation courante de l'utilisateur.

5.3 Approches de modélisation des informations contextuelles

Comme présenté dans la partie de l'état de l'art (chapitre 2), la classification des approches de modélisation du contexte peut être comme suit :

a. Modèles liés à l'endroit

- Modèle symbolique,
- Modèle géométrique,
- Modèle combiné.

b. Modèles liés aux structures de données

- Paires (clé, valeur),
- Production (ou Génération) d'étiquette,
- Les RDF (Ressource Description Framework),
- Les modèles graphiques,
- Modèles orienté objet,
- Modèles basé sur la logique,
- Les ontologies.

5.4 Modèles et langages de modélisation

5.4.1 Définition d'un modèle

Nous citons ci-dessous quelques définitions du terme « modèle » et sa signification dans le monde de l'informatique.

- Un modèle est une représentation d'une partie de la fonctionnalité, de la structure et/ou du comportement d'un système [OMG, 2001].

- Un modèle est une simplification de quelque chose qui nous permet de voir, manipuler, et raisonner sur le sujet étudié, et qui nous aide à en comprendre la complexité inhérente [Mellor, 2004].

- Un modèle est une abstraction d'un système [Frankel, 2003]. Une abstraction est une description de quelque chose qui omet certains détails non pertinents.

- Un modèle est une abstraction d'un système physique qui distingue ce qui est pertinent de ce qui ne l'est pas dans le but de simplifier la réalité. Un modèle contient tous les éléments nécessaires à la représentation d'un système réel [Muller, 2004].

- Un modèle est une simplification d'un système créé avec un but spécifique. Le modèle doit être capable de répondre aux demandes à la place du système étudié. Les réponses fournies par le modèle doivent être les mêmes que celles fournies par le système, à condition qu'elles restent dans la limite du domaine défini par le but général du système [Bézivin, 2001].

- Un modèle est une description d'un système dans un langage bien défini, et qui respecte un format précis (une syntaxe) et une signification (une sémantique). Cette description doit être convenable pour une interprétation automatisée par un ordinateur [Kleppe, 2003].

Pour récapituler les différentes définitions citées plus haut, nous dirons que la signification du terme modèle s'articule autour d'une notion qui désigne la représentation, la simplification, l'abstraction et la description limitée d'un système.

Donc, la construction d'un modèle a pour objectif de représenter une partie particulière d'un système et selon un contexte bien précis. Cette construction est faite à l'aide d'un langage bien défini qui fournit aux concepteurs une syntaxe et une sémantique spécifiées pour réglementer la création des éléments et leurs relations. Le langage utilisé s'appelle, alors, langage de modélisation.

Un langage de modélisation est une spécification formelle bien définie qui fournit les éléments et les concepts de base pour aider à la construction des modèles.

Dans notre étude et pour pouvoir construire un modèle contextuel, nous avons choisi d'utiliser le langage de modélisation UML (Unified Modeling Language).

5.4.2 Unified Modeling Language (UML)

Le langage unifié de modélisation (UML - Unified Modeling Language) [OMG, 2003a] a été proposé par Grady Booch, Ivar Jacobson et Jim Rumbaugh.
UML a été conçu pour unifier les différents langages de modélisation proposés indépendamment et aussi pour résoudre le problème de l'interopérabilité des outils de modélisation issue de la grande diversité des langages proposés.
Le principe de UML est fondé sur l'orienté objets, la notion de classe, l'objet, les attributs, les méthodes et les relations entre les classes ou les objets.
UML est un langage visuel destiné à la modélisation et à la représentation des informations.
La spécification de UML définit un langage graphique pour visualiser, spécifier, construire et documenter les artefacts des systèmes distribués orientés objets.

5.4.2.1 Concepts de base du langage UML

- Le concept d'objet
- Le concept de classe
- Le concept d'entité

a. Le concept d'objet
Un objet est un élément matériel ou immatériel qui satisfait les contraintes de distinction, de permanence et d'activité. Un objet possède une identité, un état et un comportement.
Tout objet peut être manipulé ou utilisé, c'est-à-dire qu'il a la capacité de communiquer avec les utilisateurs à l'aide d'une interface qui lui est propre.
L'interface d'un objet comprend :

- Les services qu'il peut rendre à ses utilisateurs
- Les ressources qu'il peut mettre à la disposition de ses utilisateurs

b. Le concept de classe
Une classe est un ensemble d'objets ayant des similitudes. Ces ressemblances sont caractérisées par la façon d'identifier les objets, par leurs types d'états possibles et par le rôle qu'ils jouent.

c. Le concept d'entité

Une entité est un ensemble d'objets ayant la même structure et qui sont gérés de la même façon.

5.4.2.2 Les diagrammes UML

La version UML 2.0 [OMG, 2003c] propose un ensemble de treize types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Ces types de diagrammes sont répartis en deux groupes : les diagrammes structurels et les diagrammes comportementaux

• Les diagrammes structurels :
– Diagramme de classes : décrit les éléments de base statiques comme les classes, les associations, les interfaces, les attributs, les opérations, les généralisations, etc.
– Diagramme d'objets : montre les instances des éléments structurels et leurs liens à l'exécution.
– Diagramme de packages : représente l'organisation logique du modèle et les relations entre packages.
– Diagramme de structure composite : représente l'organisation interne d'un élément statique complexe.
– Diagramme de composants : montre des structures complexes avec leurs interfaces fournies et requises.
– Diagramme de déploiement : décrit le déploiement physique des « artefacts » sur les ressources matérielles.

• Les diagrammes comportementaux :
– Diagramme de cas d'utilisation : montre les interactions fonctionnelles entre les acteurs et le système.
– Diagramme de vue d'ensemble des interactions : combine les diagrammes d'activité et de séquence pour former des fragments d'interaction avec des décisions et des flots.
– Diagramme de séquence : décrit la séquence verticale des messages passés entre objets au niveau d'une interaction.
– Diagramme de communication : montre la communication entre objets dans le plan au niveau d'une interaction.
– Diagramme de temps : combine les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps.
– Diagramme d'activité : décrit l'enchaînement des actions et décisions au niveau d'une activité.
– Diagramme d'états : représente les différents états et transitions possibles des objets d'une classe.

5.4.2.3 Domaines d'utilisation de UML

- UML définit un langage pas un processus
  - Le processus est indépendant de l'organisation et du problème à traiter;
  - UML définit en premier lieu le méta- modèle (la sémantique);
  - UML définit ensuite une notation (la syntaxe);
  - Les auteurs encouragent un processus guidé par les Use Cases;
- UML est un langage public et standardisé
  - L'OMG a retenu UML comme langage de modélisation objet standard, pour la description et la conception;

5.4.2.4 Les points forts de UML

- Notations simplifiées et non ambiguës
- Formalisation précise des concepts
- Couverture importante des besoins en modélisation

5.4.2.5 Les objectifs de UML

- Fournir un langage de modélisation graphique expressif
- Fournir des mécanismes d'extension et de spécialisation des concepts de base
- Être indépendant des langages et du processus de développement
- Fournir une définition formelle du langage
- Favoriser le développement des outils de modélisation
- Intégrer les meilleures pratiques

5.5 Modélisation graphique du contexte

5.5.1 Principe

Pour modéliser graphiquement le contexte d'utilisation, nous proposons un processus de cinq (5) phases :

1. Identification : définir tous les objets et les contraintes susceptibles d'influencer la situation courante d'un utilisateur ;
2. Nomenclature : Attribuer un nom (ou stéréotype) à chaque objet ;
3. Hiérarchisation : Organiser les objets selon les dépendances et la hiérarchie du système;
4. Sélection du diagramme : Choisir le diagramme adéquat pour représenter les objets qui contiennent des informations contextuelles ;
5. Construction du modèle : Schématiser tous les constituants du modèle à partir des objets identifiés et en utilisant les concepts du diagramme choisi.

5.5.2 Mise en œuvre

1ère phase : Identification

La situation courante d'un utilisateur peut subir plusieurs facteurs et contraintes répartis comme suit :

a. Les contraintes relatives à l'utilisateur lui-même
- Identité
- Profil
- Préférences

b. Les contraintes relatives à l'application
- Logiciels
- Matériels
- Réseaux

c. Les contraintes relatives à l'environnement
- Localisation
- Temps
- Climat (température, pression, …)
- Objets environnants (Personnes, ressources, …)

2ème phase : Nomenclature

La nomenclature des objets collectés est :

- ✓ Utilisateur
- ✓ Personne
- ✓ Localisation
- ✓ Temps
- ✓ Device
- ✓ ressource
- ✓ PC
- ✓ Laptop
- ✓ PDA
- ✓ Téléphone mobile «Mobile»
- ✓ Session
- ✓ Profil
- ✓ Préférence
- ✓ Contenu
- ✓ Présentation
- ✓ Comportement
- ✓ Rue
- ✓ Bâtiment
- ✓ Aéroport
- ✓ Gare de train «Gare»
- ✓ Bureau
- ✓ Maison
- ✓ Cyber

3ème phase : Hiérarchisation

La figure 31 représente les liens et les dépendances hiérarchiques entre les différents éléments du contexte d'utilisation.

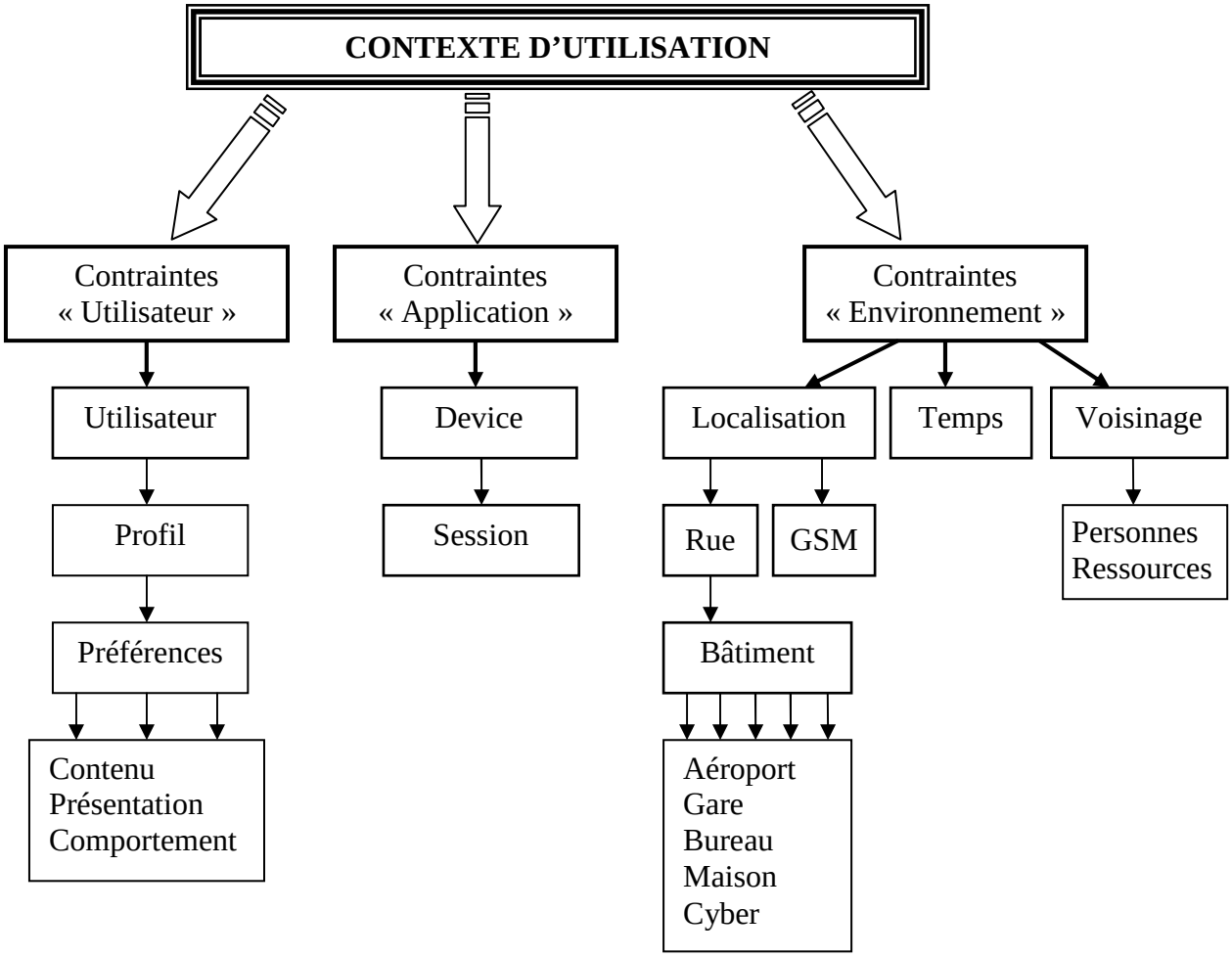


Figure 31. Hiérarchie des dépendances des classes

4ème phase : Sélection du diagramme

Pour représenter le modèle contextuel, nous avons opté pour le diagramme de classes proposé par le langage UML. Ce type de diagramme représente la structure statique des classes d'un logiciel ainsi que les relations qui existent entre ces classes. Après avoir fixé l'ensemble des éléments contextuels du système, le diagramme de classe représente un excellent formalisme de base pour modéliser ces éléments. La notation est assez simple et met en œuvre des rectangles contenant les classes et des flèches montrant soit un lien d'héritage soit une association.

5ème phase : construction du modèle

Avant d'entamer la construction du modèle contextuel, nous allons, d'abord, décrire toutes les classes en précisant leurs attributs et leurs opérations (méthodes). Ensuite, nous présentons les associations utilisées dans notre modèle ainsi que leurs types et leurs cardinalités.

Le tableau 2 définit les attributs et les opérations de chaque classe.

Nom de la classe	Attributs	Opérations
Utilisateur	IdUser Username Password	Créer ( ), Modifier ( ), Supprimer ( )
Personne	Identité Email	Créer ( ), Modifier ( ), Supprimer ( )
Localisation	IdRue : integer IdBatiment : integer Longitude : string Latitude : string Hauteur : string Adresse : string	Créer ( ), Modifier ( ), Supprimer ( )
Temps	Date : string Heure : string Saison : string Température : string Pression : string	Créer ( ), Modifier ( ), Supprimer ( )
Device	IdDevice : integer	Créer ( ), Modifier ( ), Supprimer ( )
PC	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Laptop	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
PDA	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Mobile	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Session	IdSession : integer IdUser : integer IdDevice : integer EtatSession : string HeureDebut : string Duree : string Date : string	Créer ( ), Modifier ( ), Supprimer ( ), Ouvrir ( ), Fermer ( ), Suspendre ( ), interrompre ( ), reprendre ( )
Profil	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Préférence	IdPreference : integer	Créer ( ), Modifier ( ), Supprimer ( )
Contenu	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Présentation	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Comportement	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Rue	IdRue : integer	Créer ( ), Modifier ( ), Supprimer ( )
Bâtiment	IdBatiment : integer	Créer ( ), Modifier ( ), Supprimer ( )

Aéroport	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Gare	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Bureau	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Maison	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )
Cyber	Attribut : string Valeur : string	Créer ( ), Modifier ( ), Supprimer ( )

Tableau 2. Description des attributs et des opérations

Le tableau 3 définit les associations entre les différentes classes.

Nom association	Type association	Classes concernées	cardinalités
Utiliser	Simple	Utilisateur Device	**0..*** **0..***
Occuper	Simple	Utilisateur Session	**0..*** **1**
Est un	Généralisation	Utilisateur Personne	**1** **1..***
proximité	Simple	Utilisateur Ressource	**0..*** **0..***
Voisinage	Réflexive	Personne Personne	**0..*** **0..***
Présenter	Simple	Utilisateur Profil	**0..*** **0..***
Est un	Généralisation	PC Device	**1** **1..***
Est un	Généralisation	Laptop Device	**1** **1..***
Est un	Généralisation	PDA Device	**1** **1..***
Est un	Généralisation	Mobile Device	**1** **1..***
Se trouver à	Simple	Device Temps	**1** **0..***
Caractérisé par	Simple	Device Localisation	**0..1** **0..***
Situer	Simple	Localisation Rue	**0..1** **1..***
Situer	Simple	Localisation Bâtiment	**0..1** **1..***
Renfermer	Simple	Rue Bâtiment	**0..*** **1**
Est un	Généralisation	Aéroport Bâtiment	**1** **1..***
Est un	Généralisation	Bureau	**1**

		Bâtiment	**1..***
Est un	Généralisation	Gare Bâtiment	**1** **1..***
Est un	Généralisation	Maison Bâtiment	**1** **1..***
Est un	Généralisation	Cyber. Bâtiment	**1** **1..***
Composer	Agrégation	Préférence profil	**1..*** **1..***
Est un	Généralisation	Contenu Préférence	**1** **1..***
Est un	Généralisation	Présentation Préférence	**1** **1..***
Est un	Généralisation	Comportement Préférence	**1** **1..***

Tableau 3. Description des associations

En ayant ces éléments (classes, associations) qui représentent les composantes principales d'un diagramme de classe, nous pouvons construire notre modèle contextuel qui servira comme entrée pour l'opération de fusion de modèles.

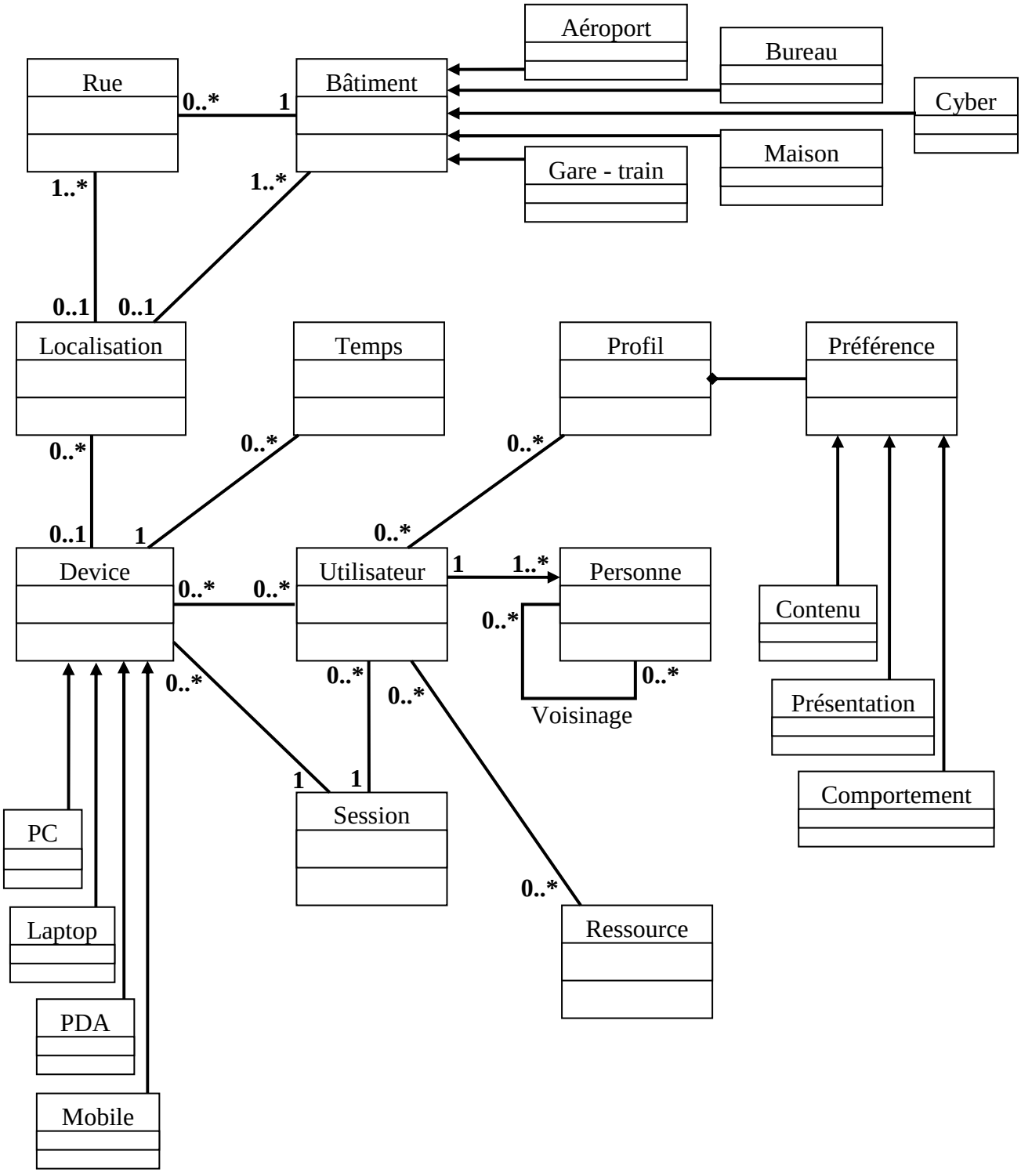


Figure 32. Diagramme de classes du modèle contextuel [Benselim, 2009b]

Le modèle contextuel (figure 32) met en évidence la prise en charge des principaux éléments constituant le contexte d'utilisation, à savoir : la localisation, le temps, le dispositif utilisé (device), les préférences de l'utilisateur, les personnes avoisinantes et les objets (ou ressources) environnants.

Conclusion

Dans ce chapitre réservé à la deuxième étape de notre approche, nous avons essayé de modéliser le contexte d'utilisation à partir des éléments contextuels séparés lors de l'étape précédente. Pour cela, nous avons utilisé une approche de modélisation graphique basée sur le langage UML. Ce choix est argumenté par les multiples avantages de ce langage comme la disponibilité de notations simplifiées et non ambiguës, la formalisation précise des concepts et la couverture importante des besoins en modélisation. Le langage UML propose plusieurs types de diagrammes parmi lesquels nous avons choisi le diagramme de classes pour représenter les aspects contextuels. Notre contribution s'illustre par la construction d'un modèle contextuel « prototype » qui aidera les concepteurs à développer les applications sensibles au contexte.
Le modèle contextuel ainsi construit, sera intégré dans le processus de développement de l'approche MDA pour être pris en considération. Cette intégration représente la troisième étape de notre approche proposée et fera l'objet du chapitre suivant.

CHAPITRE 6

L'INTÉGRATION DU MODÈLE CONTEXTUEL

Introduction

L'ingénierie de modèles (MDE : Model Driven Engenering) se base sur la gestion et la manipulation des modèles pour garantir les objectifs de conception et de développement.
La MDA est une approche qui utilise la notion de "modèles" pour concevoir une application. En MDA, toute chose (tout objet) peut être considéré comme un modèle. Un modèle est une description formelle d'un objet. Les modèles, qui représentent les éléments de base dans l'approche MDA, peuvent subir plusieurs types d'opérations comme la création, la suppression, la modification, la fusion, la transformation,…etc. Ainsi, la fusion de modèles représente l'une de ces opérations les plus délicates vu la spécificité des contraintes que doivent vérifier tous les éléments nécessaires à l'exécution de cette opération. Par exemple, des contraintes sur le mode de représentation des modèles en entrée, sur la composante de l'ensemble des règles de mapping (qui inclut les règles de fusion et les règles de transformation), sur les détails de déroulement du processus de fusion (union et combinaison) et à quel niveau d'abstraction se fera cette fusion (modèle, méta-modèle ou méta-méta-modèle). Le développement des applications sensibles au contexte rencontre un grand défi quant à l'intégration du contexte d'utilisation.

Notre étude se distingue de la majorité des autres travaux par la prise en compte des contraintes contextuelles pendant les étapes du processus de développement d'une application et non pas après. Le modèle contextuel, obtenu après séparations des aspects contextuels et modélisation, doit faire l'objet d'une prise en charge par une approche de développement. Notre proposition envisage une extension de l'approche MDA dans laquelle sera intégré le modèle contextuel. Pour accomplir cette intégration, nous avons utilisé une opération de fusion de modèles qui va combiner les éléments du modèle métier (PIM) avec ceux du modèle contextuel (CM : Contextual Model). Le processus de fusion passe par quatre étapes : comparaison des paires d'éléments, vérification de conformité des éléments comparés, fusion et combinaison des éléments vérifiés et, enfin, restructuration des éléments combinés. Le résultat de l'opération de fusion est un modèle fusionné (MM : Merged Model) qui sera transformé vers un modèle spécifique (PSM) suivant les spécifications de la plate-forme choisie.

6.1 Définition de l'opération de fusion

La fusion de deux ou plusieurs modèles est la combinaison de tous leurs composants (entités, attributs, associations) suivant des contraintes et des règles de passage bien définies dans le but de construire un nouveau modèle cible.

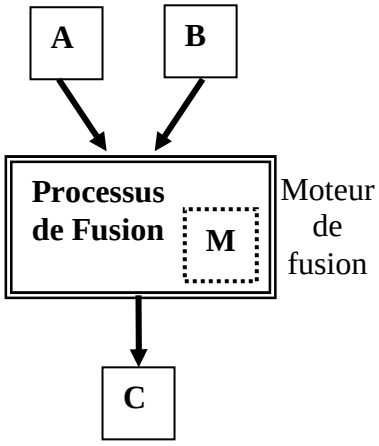


Figure 33. Schéma général d'une opération de fusion

La figure 33 représente une opération de fusion de deux modèles en entrée A et B pour construire un modèle résultat C suivant le mapping M, avec:

- Le modèle en entrée **A** : représente le modèle métier du système.
- Le modèle en entrée **B** : désigne le modèle contextuel du système.
- Le modèle résultat **C** : c'est le modèle obtenu après combinaison et transformation des modèles en entrée.
- Le processus de fusion : représente les étapes de fusion à suivre et/ou l'algorithme de fusion à appliquer.
- Le mapping **M** : c'est l'ensemble de toutes les correspondances d'éléments et les règles qui définissent la nature de la relation entre les modèles A et B. Le mapping représente une partie de tout le moteur de fusion.

6.2 Principes de la fusion

L'opération de fusion nécessite la définition de quelques notions élémentaires [Pottinger, 2003]:

6.2.1 Représentation des modèles

Les modèles peuvent être représentés selon la terminologie conventionnelle des niveaux d'abstraction de la MDA à savoir :

- le modèle,
- le méta-modèle et
- le méta-méta-modèle.

Pour notre étude, nous avons opéré au niveau d'abstraction M1, c'est-à-dire le niveau « modèle ».

6.2.2 Les entrées de la fusion

Les éléments nécessaires au déroulement d'une opération de fusion sont :

- Un modèle métier noté A qui représentent toutes les fonctionnalités du système à étudier,
- Un modèle contextuel noté B qui formalise toutes les informations relatives au contexte d'utilisation,
- Une désignation optionnelle pour choisir le modèle privilégié (A ou B) en cas de correspondance parfaite et pour résoudre les conflits.
- Un mapping M, qui se compose de :

    - Les éléments de mapping : expriment les correspondances simples entre les éléments des deux modèles en entrée. Ces correspondances peuvent être de type «égalité» ou de type «similitude». l'«égalité» est définie lorsque le même élément origine peut être en relation simple avec deux éléments destinations distincts mais égaux. Si cette relation est complexe, on dit qu'on a une «similitude».
    - Les relations : expriment les liaisons complexes qui existent entre les éléments ne présentant pas de correspondances simples.
    - Les éléments de non-mapping : ces éléments n'ont pas de représentation réelle ni dans A ni dans B mais qui peuvent aider à décrire la relation des éléments de A et B.

6.2.3 Le résultat de la fusion

Le modèle résultat C obtenu par fusion des deux modèles A et B doit satisfaire les conditions suivantes :

(C1) : Préserver les éléments : chaque élément en entrée possède un élément correspondant dans C.
(C2) : Préserver l'égalité : les éléments en entrée sont transformés vers le même élément de C si et seulement si ils sont de types ″égalité″ dans le mapping.
(C3) : Préserver les relations : chaque relation en entrée existe d'une façon explicite ou implicite dans C.
(C4) : Préserver la similitude : les éléments déclarés similaires (mais non égaux) dans le mapping conservent leur état de similitude dans C.
(C5) : Satisfaire la contrainte méta-méta-model : le modèle C doit satisfaire toutes les contraintes du méta-méta-model en introduisant les éléments et les relations nécessaires.
(C6) : Outres les éléments et les relations cités ci-dessus, aucun élément ou relation supplémentaire n'existe dans C.
(C7) : Préserver les propriétés : chaque propriété d'un élément du modèle C doit figurer comme propriété d'un élément de l'un des modèles en entrée.
(C8) : Préférence de valeur : la valeur d'une propriété d'un élément de C est choisie parmi les éléments du mapping correspondant à cet élément, sinon à partir du modèle privilégié, sinon à partir d'un élément qui lui correspond.

6.3 Gestion des conflits

6.3.1 Types de conflits

- Conflits de représentation : ce sont des conflits qui apparaissent au niveau « modèle » et sont causés par les différentes représentations du même concept du monde réel. Par exemple, deux modèles qui décrivent le même concept par différentes façons.
- Conflits de méta-modèle : ces conflits apparaissent au niveau « méta-modèle » et ils se présentent lorsque les spécifications de ce méta-modèle ne sont pas respectées par le résultat de la fusion.
- Conflits de fond (principe) : ces conflits apparaissent au niveau « méta-méta-modèle » et sont causés par la violation de contraintes dans ce méta-méta-modèle. Par exemple, les conflits causés par la restriction du type unique ou la limitation (min, max) des cardinalités des relations.

6.3.2 Résolution de conflits

- Pour les conflits de représentation : la résolution nécessite une intervention manuelle de l'utilisateur en connectant les éléments des modèles provoquant le conflit vers le même élément du mapping dont le type est « égalité ».

Soit a un élément du modèle A, soit b un élément du modèle B. Si a et b décrivent un même concept, alors nous aurons un conflit de représentation. Pour résoudre ce conflit, nous devons cherché une correspondance (a=b) dans le mapping M telle que : l'élément a peut être connecté à l'élément a' et l'élément b peut être connecté à l'élément b'.

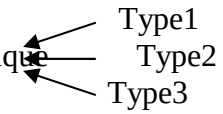
Exemples : a = identité , b = nom & prénom , identité = nom & prénom.
a = adresse , b = localité , adresse = localité.

- Pour les conflits de méta-modèle : la solution est de créer un nouveau opérateur qui va forcer (obliger) le modèle à respecter l'ensemble des contraintes. Cet opérateur sera une spécification du méta-modèle.

Exemple : écrire une règle de contrôle qui vérifie le résultat de la fusion.

- Pour les conflits de fond (principe) : pour résoudre le conflit du type unique, il faut créer un nouveau type qui va s'intercaler entre l'élément est ses types. Le nouveau type va hériter des types déclarés d'un côté, et va replacer une liaison avec l'élément d'un autre côté.

Exemple : a : Type1
a : Type2 devient a : TypeUnique avec TypeUnique
a : Type3



6.4 Algorithme de fusion

Cet algorithme est inspiré de l'étude menée par Pottinger et présentée dans [Pottinger, 2003].
C'est un algorithme qui permet de fusionner deux modèles A et B vers un modèle résultat C suivant les règles et les correspondances contenues dans le mapping M.
Pour accomplir cette fusion, l'algorithme démarre par une opération de comparaison « matching » qui permet d'identifier toutes les relations existantes entre les éléments de A et B et de chercher les éléments (correspondances) du mapping M qui justifient ces relations. L'existence de ces relations et de ces correspondances permet de produire les nouveaux éléments du modèle résultat en combinant les éléments liés par une correspondance du

mapping. Le modèle ainsi obtenu va subir une certaine validation par la vérification des propriétés des nouveaux éléments, la vérification des relations et la résolution de conflits.

Les étapes de cet algorithme peuvent être résumées comme suit :

Soit A un modèle métier,
Soit B un modèle contextuel,
Soit le mapping M,
Soient les éléments a, b, c et m tels que : a est un élément de A, b est un élément de B, c est un élément de C et m est un élément de M.
Le modèle C est le résultat de la fusion des deux modèles A et B suivant le mapping M.

Les étapes de l'algorithme de fusion sont :

1. **Initialisation :** mettre le modèle résultat C à Ø
2. **Matching des éléments :** déterminer une relation d'équivalence en groupant les éléments de A, B et M. Initialement, chaque élément se trouve dans son propre groupe. Ensuite :
   - **a.** Si une relation existe entre un élément de A ou de B avec un élément de type « égalité » appartenant à M, alors combiner les groupes correspondants.
   - **b.** Répéter (a) jusqu'à l'arrêt ensuite créer, pour chaque groupe, un nouveau élément dans C.
3. **Propriétés de l'élément :** soit c un élément de C relativement au groupe G1.
   - **a.** Les propriétés de c (sauf la propriété de type) sont formées à partir de la réunion de toutes les propriétés des éléments appartenant au groupe G1.
   - **b.** La propriété de type d'un élément de C est définie si et seulement cet élément est un élément de mapping, et sa valeur est déterminée par la condition C8 (préférence de valeur).
4. **Relations**
   - **a.** Si un élément c de C correspond à un élément de mapping m de type «similitude» alors remplacer chaque relation R dont l'origine est m par une relation qui a comme origine c et comme destination l'élément de C qui correspond au groupe de la destination de R.
   - **b.** Les relations provenant d'un élément sont classées dans l'ordre suivant :
     - celles qui correspondent aux relations dans le mapping,
     - ensuite celles qui correspondent aux relations dans le modèle privilégié mais pas dans le mapping,
     - et enfin toutes les autres relations.
   - **c.** Enlever les relations implicites du modèle C.
5. **Résolution de conflits :** la résolution d'un conflit peut, d'une part, créer un nouveau conflit, et d'autre part, elle peut s'interférer avec une autre résolution.
   Pour chaque conflit :
   - **a.** Définir une stratégie spécifique de résolution
   - **b.** Appliquer la stratégie choisie
   - **c.** Sinon, appliquer la stratégie prédéfinie par défaut.

6.5 Processus d'intégration

Le processus d'intégration (figure 34) du modèle contextuel dans l'approche MDA se base essentiellement sur l'opération de fusion de modèles. Cette opération permet de combiner les éléments du modèle métiers (PIM) avec ceux du modèle contextuel (CM). La fusion peut être décomposée en quatre phases : comparaison, test de conformité, fusion et restructuration [Kolovos, 2006a]. Le résultat de l'opération de fusion sera un modèle fusionné (MM) qui va subir, à son tour, une transformation vers un modèle spécifique (PSM) suivant le mapping de la plate-forme choisie.

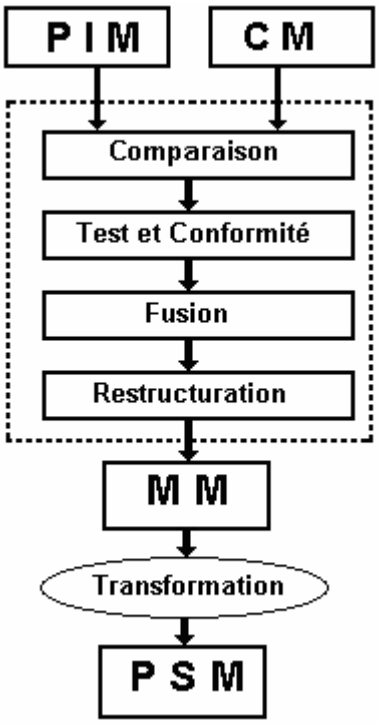


Figure 34. Etapes du processus d'intégration [Benselim, 2009b]

6.5.1 Phase de comparaison

C'est une phase de «matching» des éléments par paires, c'est-à-dire l'identification de toutes les correspondances (accords) qui existent entre les éléments équivalents des modèles en entrée. Ceci permet d'éliminer la redondance des éléments dans le modèle résultat. Les approches de comparaison peuvent être manuelles ou automatiques. Le principe de matching repose sur l'exécution de règles appelées « match-rules ». Chaque « match-rule » peut comparer des paires d'éléments appartenant à deux modèles différents et donner, ensuite, une décision sur le résultat de cette comparaison. Cette décision, de type logique, montre si les deux éléments comparés sont assortis entre eux ou non.
La comparaison porte sur tous les éléments qui constituent le modèle à fusionner. C'est-à-dire que chaque élément du modèle PIM sera comparé à un autre élément du modèle contextuel CM à condition que ces deux éléments soient du même type.
Par exemple, UML propose l'utilisation des notions de classe, opération, attribut… Donc, pour comparer deux modèles UML, il faut appliquer l'opération de «matching» sur tous leurs éléments constituants comme les classes, les opérations, les attributs…, ceci nous ramène à

écrire plusieurs règles de comparaison (match-rules) dont chacune permet de décrire les éventuelles correspondances existantes entre les paires d'éléments de même type.

La syntaxe générale d'une règle de comparaison est de la forme :

rule *<nom_de_la_règle>*
match *PIM : UML!<élément_PIM>*
with *CM : UML!<élément_CM>*
extends *<portée>*
compare *<expression de comparaison>*

Cette règle permet de comparer un élément du PIM avec un autre élément du CM où les deux modèles (PIM et CM) sont conformes au même méta-modèle (UML). Le champ d'application (portée) de cette règle est défini par la partie « **extends** ». La partie « **compare** » comprend une expression de comparaison entre les deux éléments à comparer et doit retourner une décision de type logique qui affirme ou non la correspondance de ces deux éléments. Donc, si « **compare** » retourne la valeur 'vrai' on dit que les deux éléments comparés sont correspondants.

6.5.2 Phase de vérification de la conformité

Pendant cette phase, tous les éléments identifiés lors de la phase de comparaison (matching) sont testés et vérifiés. La vérification porte sur la conformité des éléments les uns par rapport aux autres, et permet l'identification d'éventuels conflits qui peuvent rendre la fusion non réalisable.
De même que pour l'opération de comparaison, la vérification de la conformité des éléments comparés doit être appliquée sur tous les éléments constituants le modèle. La conformité est, aussi, réalisée à l'aide de « match-rules » qui décident si les deux éléments comparés sont conformes ou non. Sauf que cette conformité soit définie dans une nouvelle partie appelée « **conform** ».

Donc, une règle de comparaison et de vérification de conformité sera de la forme :

rule *<nom_de_la_règle>*
match *PIM : UML!<élément_PIM>*
with *CM : UML!<élément_CM>*
extends *<portée>*
compare *<expression de comparaison>*
conform *<tests de conformité>*

La partie « **conform** » comprend des instructions décrivant les conditions à remplir pour vérifier la conformité des éléments à comparer. Cette partie doit retourner une décision de type logique (vrai ou faux) qui nous indique si ces les éléments comparés sont bien conformes ou non.

6.5.3 Phase de fusion

C'est la phase de l'union et de la combinaison des éléments issus des deux phases précédentes pour avoir les nouveaux éléments du modèle résultat. Dans cette phase, il existe deux démarches possibles, et ce, suivant le résultat du matching. En effet, si les éléments comparés

sont jugés « correspondants », alors on procède à leur fusion dans le modèle cible MM ; et si ces éléments comparés sont jugés « non correspondants », alors on leur applique une transformation vers le modèle MM. La correspondance des éléments comparés est définie sur la base des décisions logiques retournées par les parties « **compare** » et « **conform** ».
Pour exécuter ces opération, deux types de règles sont utilisées : les règles de fusion « merge-rules » ou les règles de transformation « transform-rules ».
Ainsi, si « **compare** » retourne 'vrai' et si « **conform** » retourne 'vrai', alors nous pouvons conclure que les deux éléments comparés sont correspondants (égaux et conformes). Dans ce cas, nous devons appliquer une règle de type « merge-rule » qui va permettre de fusionner (assembler) ces éléments correspondants dans un même élément du modèle cible. Cette combinaison permet d'éliminer la redondance des mêmes éléments dans le modèle cible MM.
La syntaxe générale utilisée pour décrire une règle de fusion « merge-rule » est la suivante :

```
rule <nom_de_la_règle>
    merge  PIM : UML!<élément_PIM>
    with CM : UML!<élément_CM>
    into MM : UML!<élément_MM>
   {
     < instructions de fusion>
   }
```

Sinon, si l'une ou l'autre des deux parties « compare » et « conforme » retourne 'faux', alors nous déduisons que les éléments comparés ne sont pas complètement correspondants. Dans ce cas, nous devons déplacer chacun de ces éléments vers le modèle cible à l'aide de règles de type « transform-rules ».

La syntaxe de ce type de règles est de la forme :

Si l'élément appartient au modèle source PIM,

```
rule <nom_de_la_règle>
    transform PIM : UML!<élément_PIM>
    to MM : UML!<élément_MM>
    {
     < instructions>
    }
```

Ou bien, si l'élément appartient au modèle source CM,

```
rule <nom_de_la_règle>
    transform CM : UML!<élément_CM>
    to MM : UML!<élément_MM>
    {
     < instructions>
    }
```

Pour garder la trace et l'origine des éléments fusionnées ou transformés, nous étiquetons chaque élément du modèle cible par une mention de type « stéréotype ».

Les stéréotypes sont définis préalablement dans la partie « **pre** » de la façon suivante :

```
Pre
{
def mergedStereotype : new MM !stereotype ;
mergedStereotype.name:='merged';
def PIMStereotype : new MM !stereotype ;
PIMStereotype.name:='PIM';
def CMStereotype : new MM !stereotype ;
CMStereotype.name:='CM';
}
```

6.5.4 Phase de nettoyage et de restructuration

Le modèle résultat, obtenu après la phase de fusion, peut contenir des inconsistances ou des anomalies. Pour le rendre fini et valide, il doit subir une opération de nettoyage et/ou une opération de restructuration afin de remettre l'ordre dans sa structure finale.
Cette opération est surtout applicable lorsque les modèles à fusionner ne sont pas conformes au même méta-modèle, ce qui n'est pas le cas pour nos modèles en entrée (PIM et CM).

6.5.5 Phase de transformation

La dernière étape du processus d'intégration du modèle contextuel consiste à transformer le modèle fusionné vers un modèle spécifique en utilisant les spécifications techniques de la plate-forme choisie.

6.6 Les stratégies

Le langage EML (Epsilon Merging Language) introduit la notion de stratégies pour accomplir l'opération de fusion d'une façon souple et efficace. Une stratégie est un type d'algorithme qui peut être rattaché à une spécification EML pour implémenter une fonctionnalité nécessaire à l'exécution d'une règle quelconque (matching, fusion ou transformation). Il existe trois types de stratégies dans EML, une pour chaque type de règle.

a. Stratégie de matching : le principe consiste à comparer deux éléments de modèles et retourner une instance « match » qui contient l'information indiquant si ces éléments correspondent ou non. Cette stratégie comprend deux types de démarches : *MofIdMatchingStrategy* et *EmfIdMatchingStrategy* qui comparent les éléments du MOF (MetaObject Facility) et les modèles basés EMF (Eclipse Modelling Framework) en examinant leur nature XMI (XML Metadata Interchange). On applique cette stratégie lorsque les modèles à comparer découlent d'un modèle commun.

b. Stratégie de fusion : cette stratégie possède deux méthodes : *appliesTo(leftElement: Object, rightElement : Object) : Boolean* qui décide si le résultat de la stratégie appliquée est une valeur logique, et *autoMerge(leftElement :Object, rightElement : Object)* qui spécifie la logique qui fusionne les deux instances. D'autre part, cette stratégie renferme deux démarches *CommonMofMetamodelMergingStrategy* et *CommonEmfMetamodelMergingStrategy* qui permettent la fusion d'éléments qui proviennent du MOF et de modèles EMF appartenant à un métamodel commun. Dans ce cas un algorithme de fusion est nécessaire.

c. Stratégie de transformation : c'est une stratégie qui consiste à transformer un élément source vers un élément équivalent dans le modèle cible. A un module EML s'associent deux stratégies de transformation : *leftTransformationStrategy* et *rightTransformationStrategy*.

6.7 Exemple de fusion de modèles

Cas de deux modèles UML

Pour illustrer le processus de fusion de modèles (PIM & CM: Modèle Contextuel), on va choisir des modèles UML qui représente chacun un type de spécification bien déterminé. C'est à dire assurer la séparation entre les spécifications fonctionnelles d'un système (PIM) et les spécifications contextuelles de ce même système (CM:Modèle Contextuel). Les modèles utilisés ont été simplifiés pour mieux mettre en évidence le processus de fusion.

Soient deux modèles UML développés d'une façon complètement indépendante.

1. Le premier modèle UML (Figure 35) schématise le modèle PIM d'une application qui consiste à assurer les consultations d'informations médicales. A l'aide de cette application, une société pharmaceutique fabrique des médicaments et propose des informations relatives au domaine médical (maladies et médicaments). Toutes ces informations sont sauvegardées et actualisées dans des bases de données gérées par un système d'information correspondant.

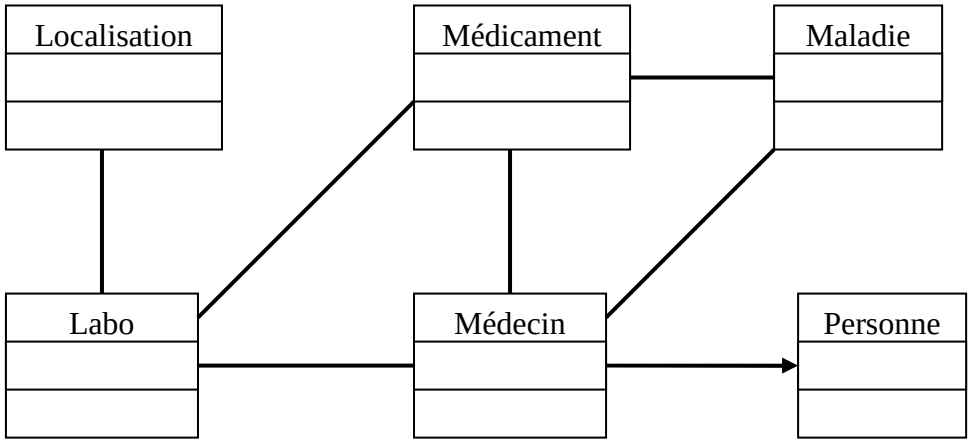


Figure 35. Diagramme de classes d'un PIM

- Un Médecin est une personne
- Un Médecin appartient à un Labo
- Un Labo est situé dans une Localisation
- Un Labo fabrique des médicaments
- Un Médecin identifie les maladies
- Un Médecin prescrit les médicaments
- Un Médicament correspond à une maladie

2. Le deuxième modèle UML (Figure 36) schématise le modèle contextuel dans lequel sera représenté le contexte d'utilisation (device utilisé, temps, localisation, profil utilisateur,…).

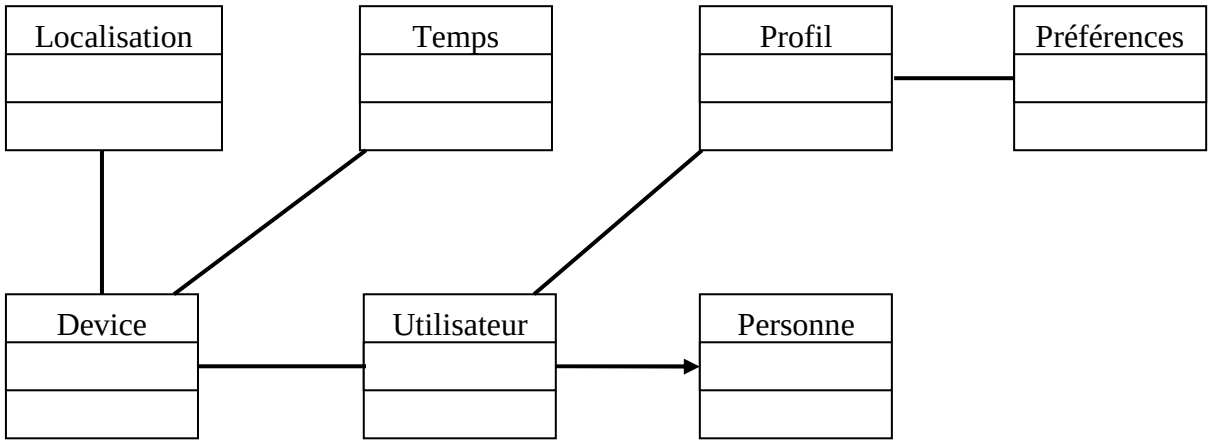


Figure 36. Diagramme de classes d'un modèle contextuel (CM)

- Un utilisateur est une personne
- Une personne présente un profil particulier
- Un profil est composé de plusieurs préférences
- Un utilisateur utilise un device
- Un device se trouve dans une localisation précise

Pour procéder à l'opération de fusion des deux modèles UML (PIM & CM), il faut appliquer les quatre phases du processus de fusion.

1.Phase de comparaison : lors de cette phase, toutes les classes des deux modèles en entrée (PIM & CM) sont comparées deux à deux. La comparaison de chaque paire de classes doit retourner un résultat sur la nature de la correspondance qui existe entre les classes comparées. A la fin de cette phase, on aura identifier les deux types de classes : celles qui correspondent et celles qui ne correspondent pas. Pour notre exemple :

- Les classes correspondantes sont : Personne et Localisation
- Les classes non correspondantes sont : Médecin, Médicament, Maladie, Labo, Utilisateur, Device, Temps, Profil et Préférences.

2.Phase de vérification de la conformité : pendant cette phase, on teste la conformité des classes jugées correspondantes pour décider sur la nature du matching des classes comparées.

3.Phase de fusion : c'est au niveau de cette phase qu'on procède à l'opération de fusion proprement dite pour construire les nouvelles classes du modèle cible. Ces nouvelles classes se distingueront des classes en entrée par l'ajout de stéréotypes appropriés qui indiqueront l'origine de la classe.

En réalité, on applique deux types d'actions selon la décision du matching des classes comparées.

- En effet, si ces classes sont jugées correspondantes et conformes alors on applique une union des deux classes dans une nouvelle classe dans le modèle cible. Dans ce cas, on utilise une règle de type "merge-rule". La nouvelle classe portera la mention de stéréotype « merged » qui montre que c'est une classe issue de l'union (la fusion) de deux classes provenant des deux modèles d'entrée (figure 37).

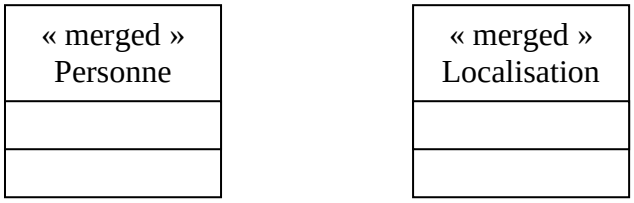


Figure 37. Exemples de classes fusionnées

- Par contre, si les classes comparées sont jugées non correspondantes alors on applique une transformation par copie (translation) de chacune de ces classes dans le modèle cible. Dans ce cas, on utilise une règle de type "transform-rule". Chaque classe issue de cette transformation portera le stéréotype « PIM » si elle provient du modèle PIM ou bien le stéréotype « CM » si elle découle du modèle contextuel (figure 38).

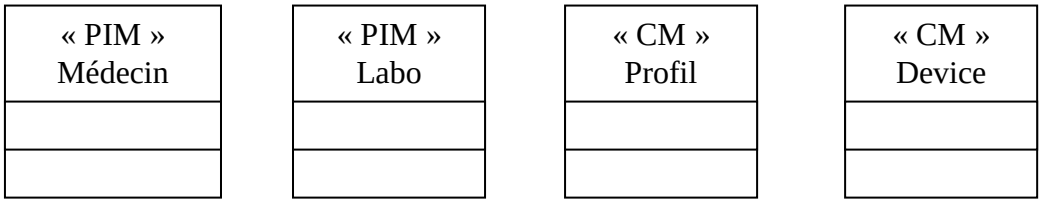


Figure 38. Exemples de classes translatées

4.Phase de restructuration : après construction du modèle cible, il est nécessaire de vérifier qu'il n'y a pas d'anomalies dans composante du modèle par rapport aux modèles d'entrée. S'il y a lieu, il faut nettoyer toutes les inconsistances trouvées et restructurer le modèle cible tout en respectant l'intégrité des classes et des associations.

Après application du processus de fusion des deux modèles de l'exemple, notre modèle résultat ainsi obtenu est le suivant (figure 39):

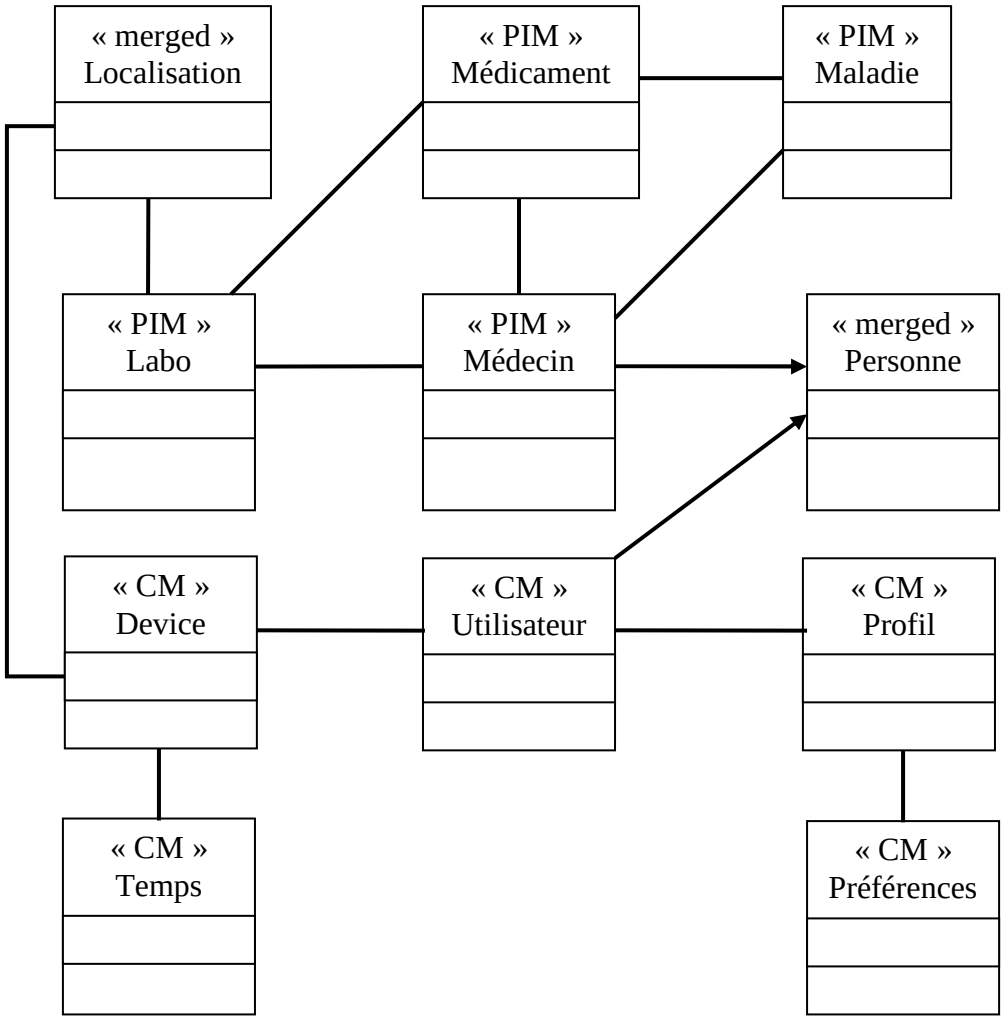


Figure 39. Diagramme de classes d'un modèle fusionné (PIM+CM)

Conclusion

Ce chapitre représente la dernière étape de l'approche proposée qui consiste à intégrer le modèle contextuel, issu de l'étape précédente, dans le processus de développement de l'approche MDA. Cette intégration permet de prendre en charge les informations contextuelles lors des étapes de conception d'une application. Le processus d'intégration se base essentiellement sur l'opération de fusion de modèles. Cette opération permet de combiner les éléments du modèle métiers avec ceux du modèle contextuel. La fusion est décomposée en quatre phases : comparaison, test de conformité, fusion et restructuration. Le résultat de l'opération de fusion est un modèle fusionné qui est, à son tour, transformé vers un modèle spécifique suivant les spécifications techniques de la plate-forme choisie.

Partie III

Expérimentation

CHAPITRE 7

ETUDE DE CAS

Introduction

Pour mettre en œuvre notre approche, une étude de cas illustrative dans le domaine médical a été entamée. Nous avons utilisé la plate-forme « Epsilon » (Extensible Platform for Specification of Integrated Languages for mOdel maNagement) qui opère sous « Eclipse » parce que nous avons trouvé tous les langages spécifiques pour implémenter les différentes phases de notre approche. En plus de UML 2.0, Epsilon propose les langages suivants : Epsilon Comparison Language (ECL), Epsilon Transformation Language (ETL), Epsilon Merging Language (EML) et Epsilon Validation Language (EVL) (Kolovos, 2008). Pour implémenter cette application pratique, nous avons suivi les étapes suivantes:

— préparation du modèle CIM,
— Construction du modèle métier PIM en utilisant UML 2.0,
— Construction du modèle contextuel CM en utilisant UML 2.0,
— Construction du modèle fusionné MM (PIM+CM),
    - Comparaison des éléments des modèles (PIM et CM) en utilisant ECL,
    - Vérification des éléments comparés en utilisant ECL,
    - Fusion des éléments comparés et vérifiés en utilisant EML,
    - Restructuration et nettoyage du modèle obtenu en utilisant EVL,
— Transformation du modèle MM vers un PSM en utilisant ETL.

7.1 Enoncé du cas d'étude

Notre application s'intitulant "consultations médicales" consiste à collecter et à dispatcher tous les renseignements relatifs aux différentes maladies.

Les informations médicales portent sur:

- L'origine d'une maladie
- Les symptômes d'une maladie
- La liste des médicaments
- La liste des médecins
- La liste des laboratoires pharmaceutiques

Les utilisateurs de l'application peuvent être:

- Les malades

- Les médecins
- Les étudiants
- Autres personnes

L'application doit garantir la disponibilité de l'information quelque soit la situation courante de l'utilisateur. Dans un environnement ubiquitaire, cette situation est caractérisée par des propriétés très spécifiques comme la mobilité des utilisateurs, les préférences d'un utilisateur, l'hétérogénéité des dispositifs utilisés, …etc. Ces paramètres constituent l'aspect contextuel de notre application et peuvent être définis comme suit :

- Les propriétés de l'utilisateur (identité, préférences,…)
- La localisation
- Le temps
- Le dispositifs utilisé
- Les personnes avoisinantes
- Les ressources disponibles

Notre application constitue une partie d'un système d'information qui sera chargé de la collecte, la saisie, le stockage, le traitement et la restitution de l'information et qui évolue dans un environnement ubiquitaire caractérisé par les changements perpétuels du contexte d'utilisation. Pour accéder à une information, un utilisateur peut utiliser n'importe quel dispositif qui lui est disponible et peut, aussi, se déplacer d'un endroit à un autre (ex. de la maison vers le bureau de travail). Le rôle de notre application est d'assurer la disponibilité de l'information demandée suivant la situation courante de l'utilisateur qui représente le contexte d'utilisation. C'est-à-dire que l'information fournie doit être adaptée à chaque changement du contexte d'utilisation.
Par exemple, le déplacement d'un utilisateur de sa maison vers son bureau de travail entraîne un changement considérable dans les paramètres du contexte d'utilisation. En effet, le contexte de la maison se caractérise par l'utilisation d'un téléphone mobile, par l'absence d'autres ressources (imprimantes) et par l'absence d'autres personnes avoisinantes. Par contre, le contexte du bureau de travail est caractérisé par l'utilisation d'un PC de bureau relié à d'autres ressources (terminaux et imprimantes) via un réseau local et en présence de ses collègues. Dans ce cas, l'application doit prendre en compte cette variation du contexte d'utilisation en adaptant l'affichage des résultats selon l'endroit et le dispositif utilisé par l'utilisateur, et aussi en lui offrant la possibilité de partager, par exemple, ses taches d'impression sur plusieurs imprimantes ou en partageant ses taches de traitement sur ses collègues.

7.2 Outils utilisés

7.2.1 Description de « Epsilon »

Pour exécuter toutes les taches requises dans le processus de fusion de modèles, on fait appel à la plate-forme « EPSILON » (Extensible Platform for Specification of Integrated Languages for mOdel maNagement) ainsi que tous les outils offerts par cette plate-forme (éditeurs, langages,…etc.).

La figure 40 schématise l'architecture générale de la plate-forme Epsilon et présente les différents outils proposés.

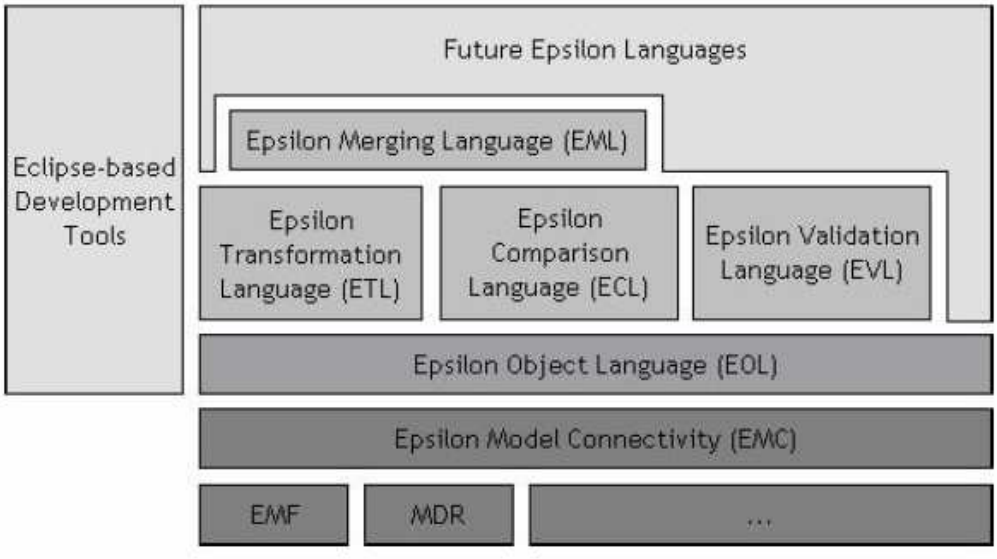


Figure 40. Architecture de la plate-forme "Epsilon"[Kolovos, 2006b]

Epsilon est une plate-forme qui fournit l'environnement nécessaire et adéquat pour le développement des langages spécifiques afin de supporter toutes les taches nécessaires à la gestion des modèles (transformation, fusion, comparaison et validation) [Kolovos, 2006b].

7.2.2 Les langages spécifiques

En effet, Epsilon propose plusieurs langages spécifiques dont chacun peut être utilisé pour exécuter une étape du processus de fusion de modèles. Ainsi, le langage ECL (Epsilon Comparison Language) sera utilisé dans les étapes de comparaison et de vérification de la conformité, le langage EML (Epsilon Merging Language) pour assurer les taches de l'étape de fusion et enfin le langage EVL (Epsilon Validation Language) qui permettra de nettoyer et valider le modèle résultat lors de l'étape de restructuration. Ces langages utilisent, de leur part, un langage objet propre à la plate-forme Epsilon appelé EOL (Epsilon Object Language) [Kolovos, 2008].

7.2.2.1 Epsilon Object Language (EOL)

C'est un langage objet qui est construit sur les bases de OCL (Object Constraint Language) et qui fournit des grandes possibilités quand à la gestion des modèles.

7.2.2.2 Epsilon Comparison Language (ECL)

C'est un langage basé règles qui permet d'exécuter l'opération de comparaison de modèles. Le principe de ECL repose sur l'utilisation des règles « match-rules » qui sont capables de comparer des paires d'éléments appartenant aux modèles d'entrée. Chaque « match-rule » comprend deux parties : « compare » et « conform » qui indiquent si les éléments à comparer sont correspondants ou non. Le corps de « compare » et « conform » est écrit en langage EOL.

7.2.2.3 Epsilon Merging Language (EML)

C'est un langage qui réutilise les concepts de « match-rules » de ECL et « transform-rules » de ETL mais en rajoutant un concept spécifique « merge-rules » qui permet d'exécuter l'opération de fusion d'une façon plus claire et plus structurée. La langage EML commence par l'exécution d'une règle « match-rule » pour définir la correspondance qui existe entre deux éléments ; et suivant la décision de cette règle, une autre règle est déclenchée, soit une « merge-rule » dans le cas d'une correspondance conforme ou une « transform-rule » dans le cas d'une non correspondance.

7.2.2.4 Epsilon Validation Language (EVL)

C'est un langage qui permet d'améliorer la façon par laquelle seront exprimées les contraintes de validités d'un modèle.

7.2.2.5 Epsilon Transformation Language (ETL)

C'est un langage de transformation basé sur les règles. ETL consiste à utiliser le concept de « transform-rules » qui permet de convertir les éléments d'un modèle source vers des éléments d'un modèle cible. Chaque « transform-rule » comprend deux parties « déclarative » et « impérative » écrites en langage EOL.

7.3 Implémentation

7.3.1 Construction du PIM en utilisant UML 2.0

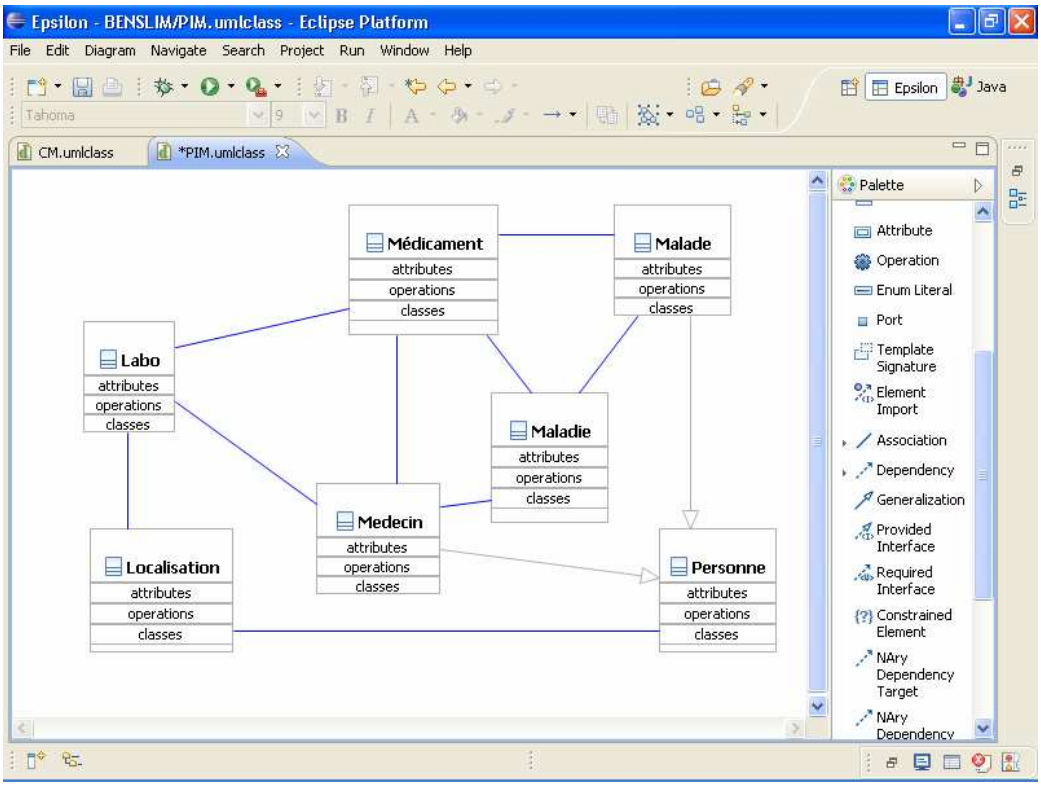


Figure 41. Le modèle PIM sous Epsilon

La figure 41 illustre le diagramme de classes du modèle métiers PIM. Ce diagramme est construit en utilisant le langage UML proposé par la plate-forme Epsilon. Ce PIM représente les principales entités de notre application ainsi que les relations liant ces entités.

7.3.2 Construction du CM en utilisant UML 2.0

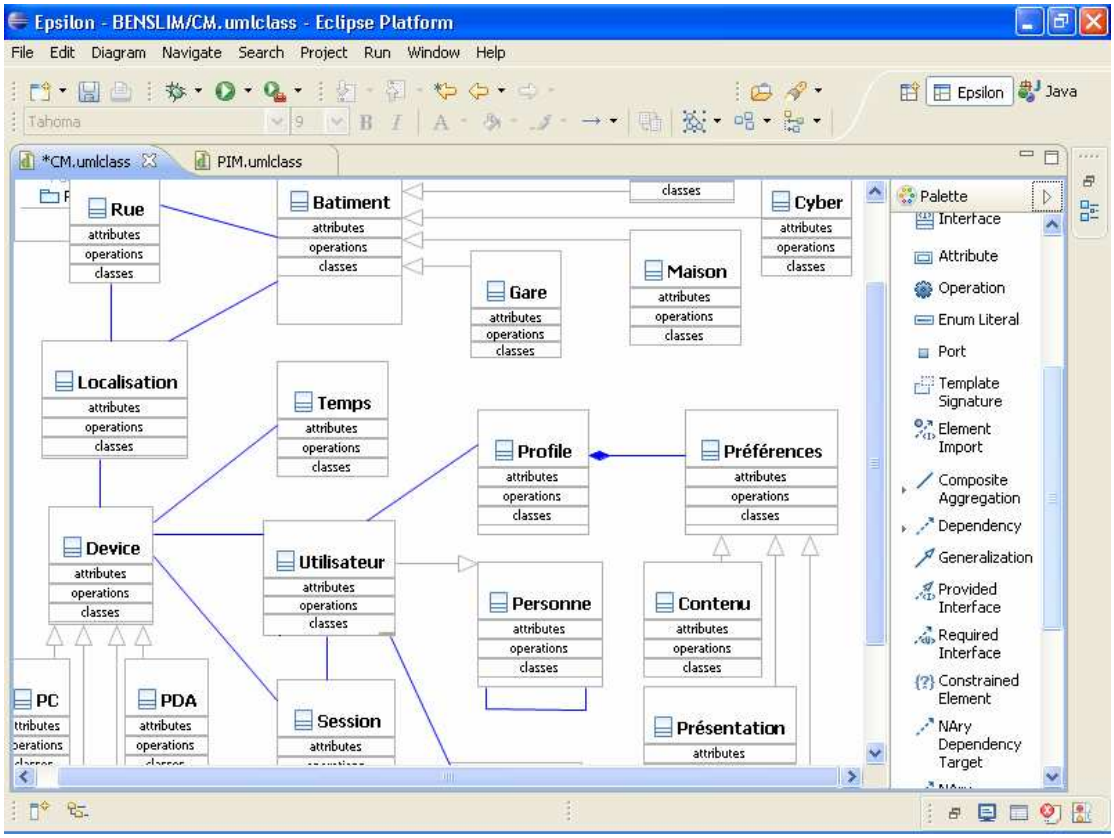


Figure 42. Le modèle contextuel CM sous Epsilon

La figure 42 schématise le modèle contextuel CM, décrit dans la section 5.5, à l'aide d'un diagramme de classes en utilisant le langage UML proposé par la plate-forme Epsilon. Ce modèle contextuel peut être considéré comme étant un modèle prototype qui sert à décrire et représenter les informations contextuelles d'un système. En effet, le CM met en évidence et prend en charge les principaux éléments constituants le contexte d'utilisation.

Pour notre application, ce modèle contextuel servira de modèle d'entrée pour accomplir l'opération de fusion nécessaire au processus d'intégration des informations contextuelles dans l'approche MDA.

7.3.3 Construction du modèle fusionné MM (PIM+CM)

La combinaison des deux modèles sources PIM et CM doit respecter les étapes du processus d'intégration présenté dans la section 6.5.
A titre de rappel, ce processus passe par cinq phases:

- Phase de comparaison,
- Phase de vérification de la conformité,
- Phase de fusion,
- Phase de restructuration,
- Phase de transformation.

Pour chacune de ces phases, Epsilon propose un langage particulier destiné à exécuter les taches spécifiques de cette phase. Tous ces langages sont écrits à base du même langage objet EOL (Epsilon Object Language). Ceci nous offre la possibilité de regrouper plusieurs phases du processus d'intégration dans le même programme (code). Ainsi, les phases de comparaison et de vérification sont réunies dans un seul programme qui va nous retourner des décisions booléennes sur le résultat de la comparaison (correspondance et conformité). Aussi, les phases de fusion et de restructuration sont regroupées dans le même programme.

7.3.3.1 Comparaison et vérification

La phase de comparaison consiste à identifier toutes les correspondances existantes entre les paires d'éléments des deux modèles comparés. Cette identification permet de déceler les éventuelles relations de similarité ou de différence qui peuvent exister entre les éléments comparés. Le rôle de cette phase est d'éliminer la redondance d'éléments similaires dans le modèle cible (modèle fusionné).

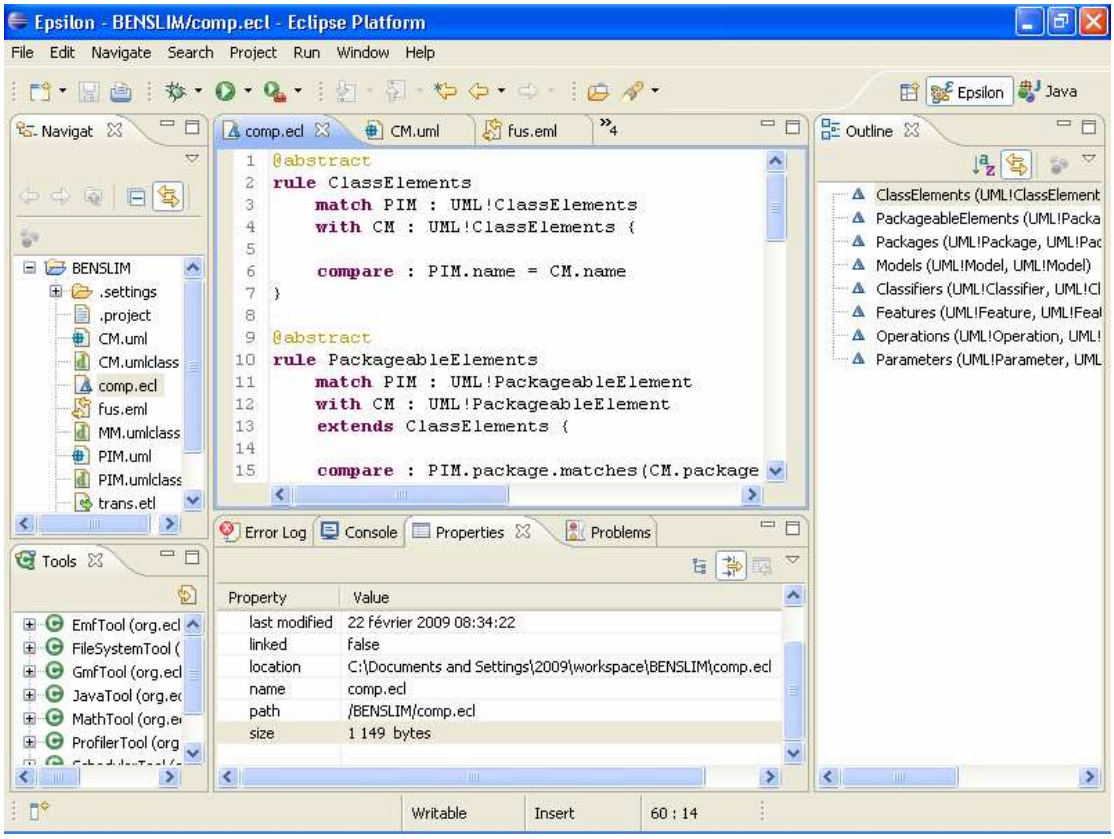


Figure 43. La phase de comparaison sous Epsilon

Pratiquement sous la plate-forme Epsilon et en utilisant le langage ECL (figure 43), la comparaison se base sur l'utilisation des règles de matching "matching-rules" qui permettent de comparer les paires d'éléments provenant des deux modèles en entrée (modèle PIM et modèle contextuel). Cette comparaison porte sur tous les éléments constituant les modèles à fusionner comme les classes, les attributs, les opérations, les associations, …
Pour comparer nos deux modèles, nous avons appliqué la règle "***match*** *elt1* ***with*** *elt2*" qui peut identifier les ressemblances existantes entre les éléments *elt1* et *elt2* appartenant respectivement aux modèles PIM et CM (modèle contextuel). Le résultat de l'opération de comparaison se trouve dans la partie "***compare***" de la règle de matching et qui est du type logique. Ainsi, si ***compare*** retourne *true* alors nous saurons que les deux éléments comparés *elt1* et *elt2* présentent une correspondance entre eux.

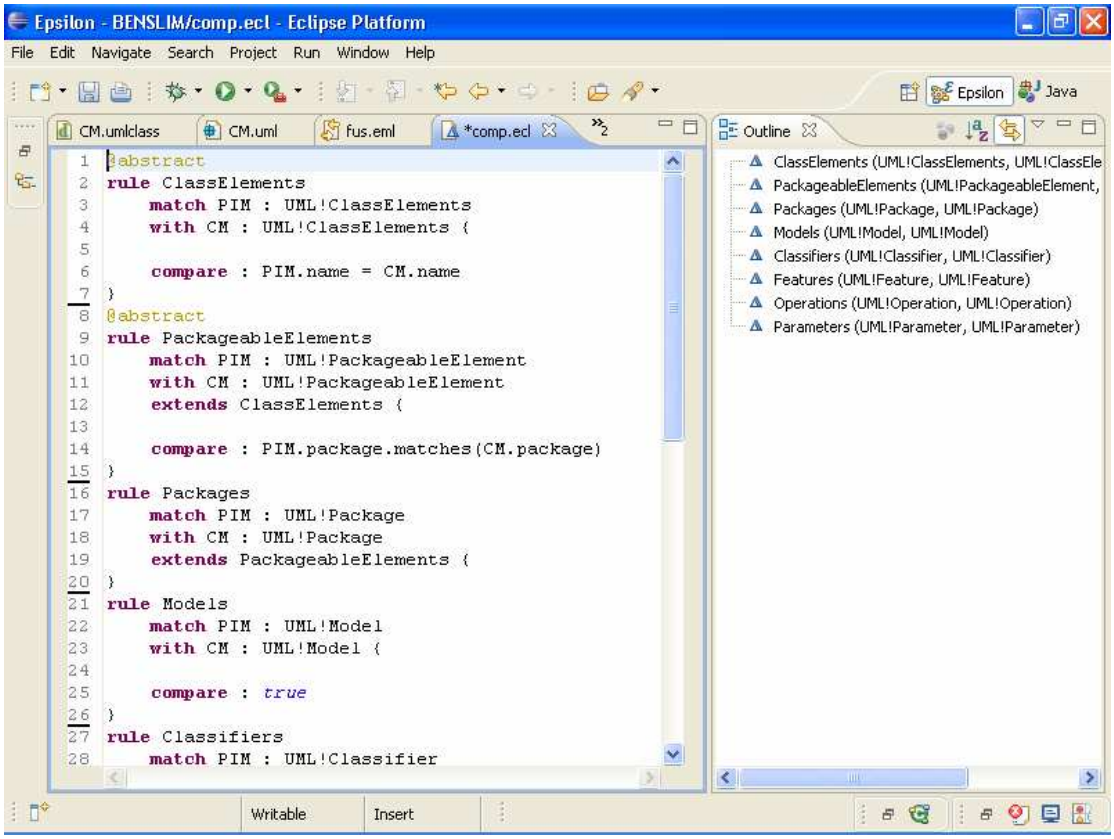


Figure 44. Extrait du programme de comparaison "*comp.ecl*"

La figure 44 illustre un extrait du programme de comparaison (*comp.ecl*) faisant ressortir les règles de matching utilisées pour comparer les modèles à fusionner.

La phase de vérification consiste à décider sur la conformité des correspondances identifiées lors de la phase de comparaison.
Cette phase utilise, à son tour, les règles de matching du type "***match*** *elt1* ***with*** *elt2*" mais en précisant si les éléments comparés sont conformes ou non. La décision sur la conformité des éléments se trouve dans la partie "***conform***" de la règle de matching et qui est du type logique. La décision retournée par la partie "***conform***" permettra de choisir le type de règles à appliquer lors de la phase suivante (fusion).

7.3.3.2 Fusion et restructuration

La phase de fusion consiste à combiner (fusionner) les modèles sources (PIM et CM) vers un modèle cible (MM). Cette combinaison est contrainte par les décisions de comparaison et de conformité fournies par les parties "***compare***" et "***conform***".
Parmi les stratégies de fusion précédemment citées dans la section **6.6**, nous avons choisi celles qui s'adaptent le plus à notre cas d'étude. Ce choix doit respecter la nature des méta-modèles pour lesquels les modèles traités sont conformes.
Nos deux modèles d'entrée (PIM et CM) ainsi que le modèle cible (MM) sont conformes au même méta-modèle (UML 2.0), donc, nous avons appliqué la stratégie de "*AllCommonMofMetamodelStrategy*" qui permet de combiner (fusionner) les éléments présentant des correspondances (matchings). Pour les autres éléments qui ne sont pas similaires (ni correspondances, ni conformité), nous avons appliqué la stratégie "CommonMofMetamodelTransformationStrategy" permettant de copier (transformer) ces éléments vers le modèle cible.

Pratiquement, la phase de fusion fait appel au langage EML (figure 45) qui propose l'application de l'un ou l'autre des deux types de règles: "*merge-rules*" ou "*transform-rules*", et ce, suivant la nature du matching existant entre les éléments à fusionner. Ainsi, si ces éléments sont correspondants (*compare=true*) et conformes (*conform=true*) alors nous appliquons une *merge-rule* de la forme "***merge*** *elt1* ***with*** *elt2* ***into*** *elt3*" où *elt1*, *elt2* et *elt3* appartiennent respectivement aux modèles PIM, CM et MM. Sinon, si les éléments comparés ne sont pas correspondants (*compare=false*) et/ou ne sont pas conformes (*conform=false*) alors nous appliquons une *transform-rule* de la forme "***transform*** *elt1* ***to*** *elt3*" où *elt1* est un élément du modèle source (PIM ou CM) et *elt3* est un élément du modèle cible (MM).

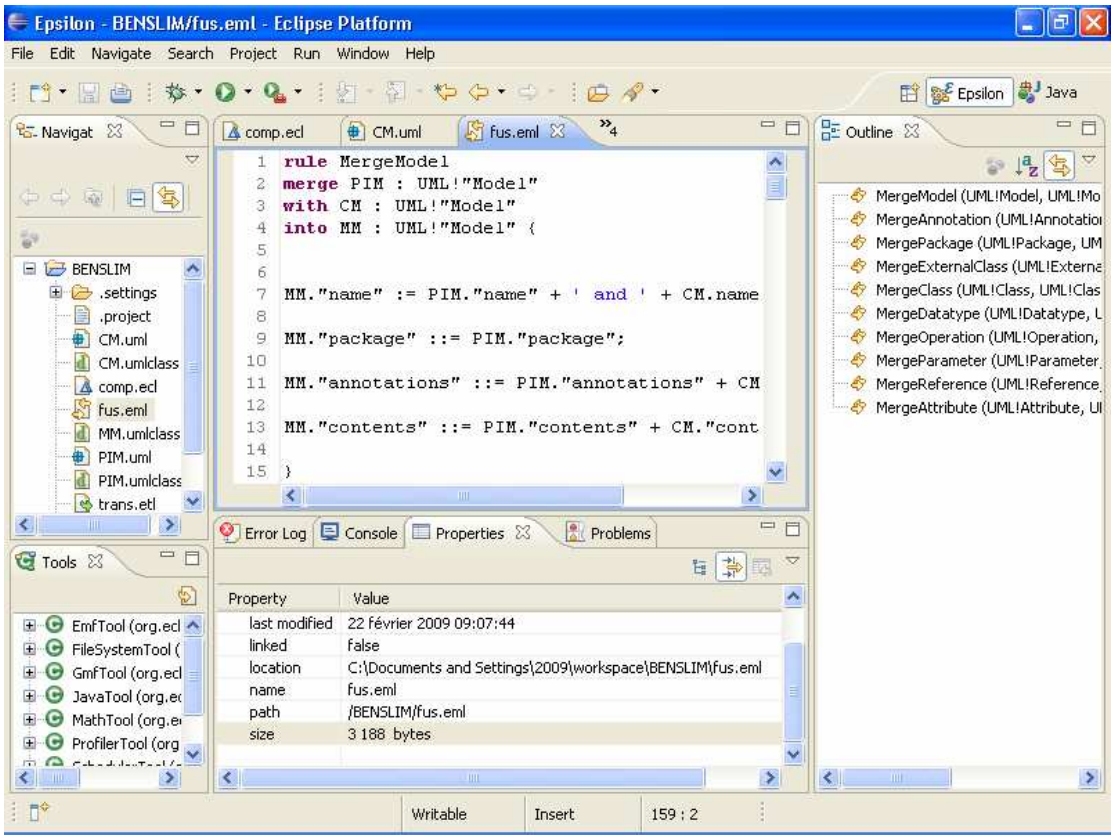


Figure 45. La phase de fusion sous Epsilon

La figure 46 illustre un extrait du programme de fusion (*fusion.eml*) faisant ressortir quelques règles nécessaires à la combinaison des éléments des deux modèles sources (PIM et CM) pour permettre la construction du nouveau modèle fusionné (MM).

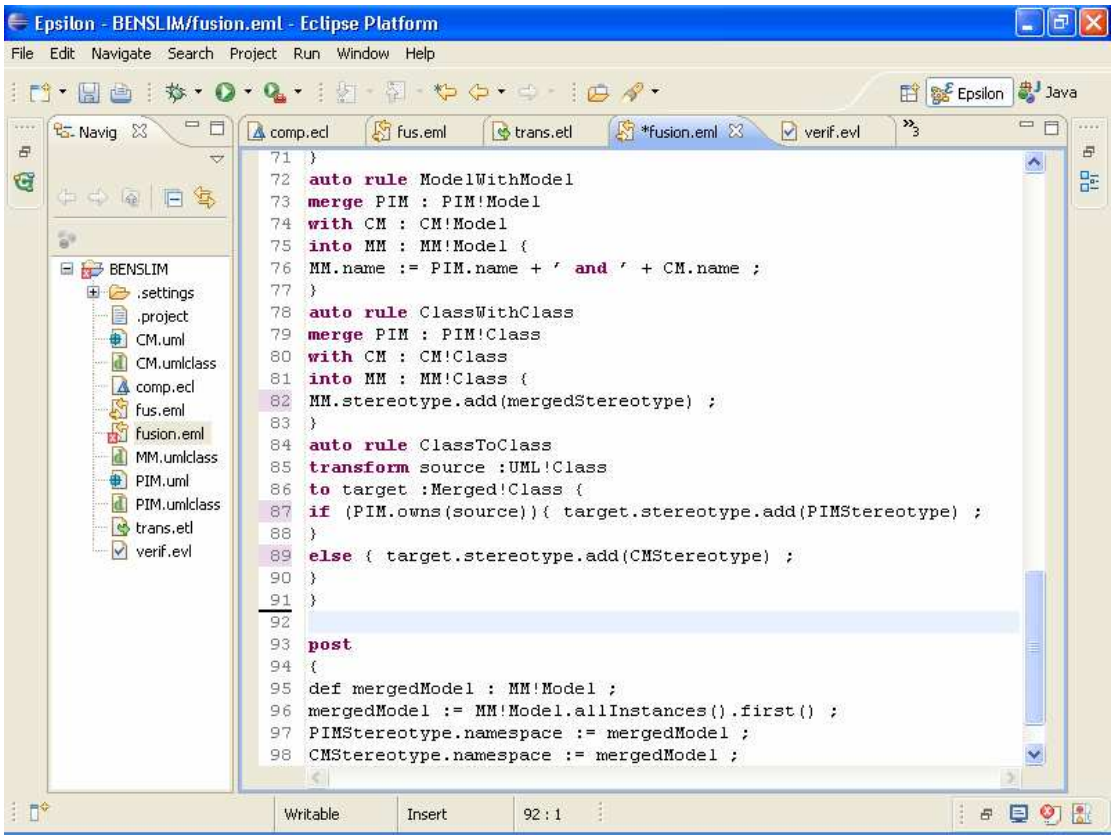


Figure 46. Extrait du programme de fusion "*fusion.eml*"

Chaque élément du modèle cible doit comprendre une étiquette de type "stéréotype" pour indiquer le modèle source duquel il provient. Ceci permet de garder la trace des éléments nouvellement créés et leur origine. Les stéréotypes sont créés dans la partie "***pre***" qui précède le corps des règles "*merge-rules*" et "*transform-rules*" et sont, ensuite, rattachés à l'élément correspondant dans la partie "***post***" qui suit ces règles.

Les blocs "***pre***" et "***post***" peuvent être écrits comme suit:

```
pre
{
def mergedStereotype : new MM!Stereotype ;
mergedStereotype.name := 'merged ' ;
def PIMStereotype : new MM!Stereotype ;
PIMStereotype.name := 'PIM ' ;
def CMStereotype : new MM!Stereotype ;
CMStereotype.name := 'CM' ;
}

…
```

```
post
{
def mergedModel : MM!Model ;
mergedModel := MM!Model.allInstances().first() ;
PIMStereotype.namespace := mergedModel ;
CMStereotype.namespace := mergedModel ;
mergedStereotype.namespace := mergedModel ;
}
```

Après application et exécution des programmes *comp.ecl* et *fusion.eml*, nous pouvons générer automatiquement le modèle fusionné MM (figure 47). Ce modèle est du même type que les modèles en entrée (diagramme de classes) et appartient au même méta-modèle (UML). Dans le modèle MM, les classes sont construites à partir de celles des modèles en entrée soit par fusion (si elles sont correspondantes et conformes) soit par transformation (si elles ne sont pas correspondantes et/ou ne sont pas conformes).

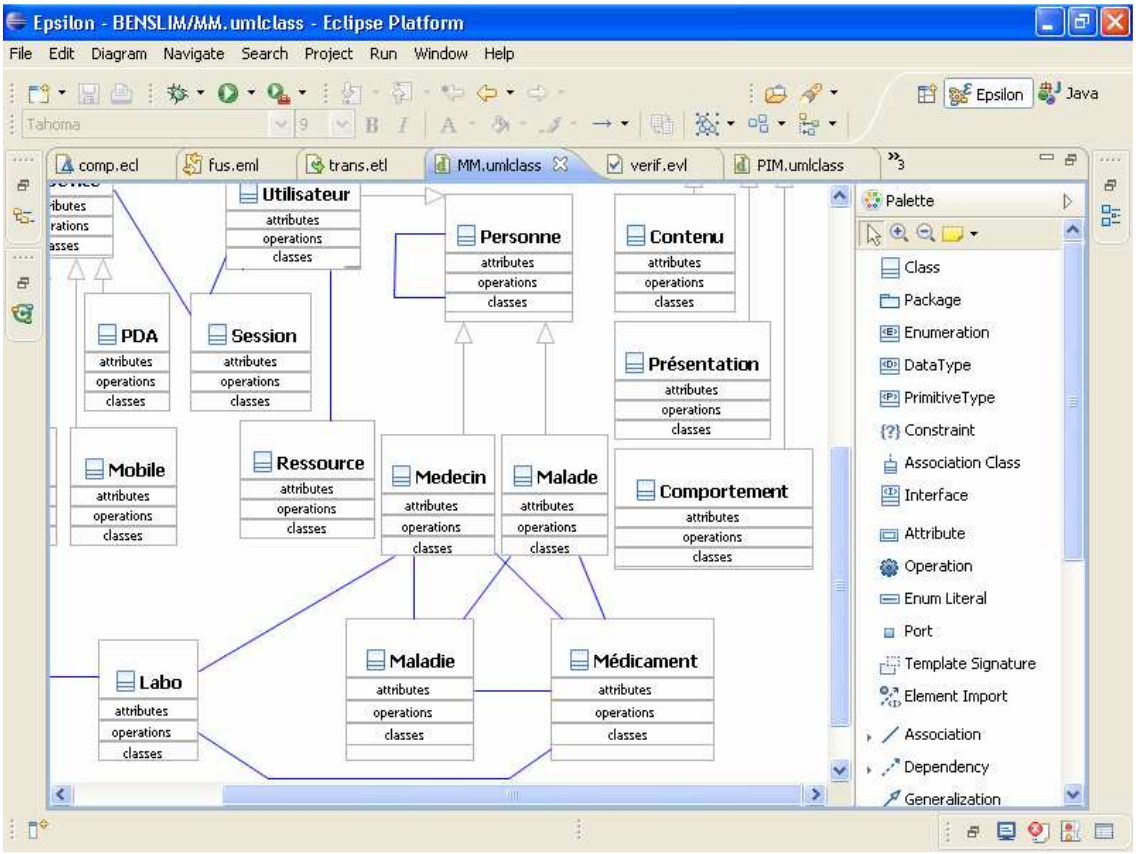


Figure 47. Le modèle fusionné MM sous Epsilon

7.3.3.3 Transformation

La phase de transformation consiste à convertir le modèle fusionné MM vers un modèle spécifique PSM en utilisant le langage ETL proposé par la plate-forme Epsilon et en respectant les spécifications techniques de la plate-forme technologique choisie.

Le langage ETL se base essentiellement sur l'application des règles de transformation de la forme "***transform*** *elt1* ***to*** *elt2*" où elt1 est un élément du modèle fusionné MM et elt2 est un élément du modèle cible PSM.

7.3.4 Discussion

Le nouveau modèle obtenu, après le processus d'intégration, permettra aux utilisateurs de l'application "consultations médicales" d'avoir les informations demandées quelque soit l'état de la situation courante. Ainsi, un médecin ou un malade sera considéré (par les nouveaux liens du modèle MM) comme un simple utilisateur soumis aux différents facteurs qui peuvent influencer l'exécution de ses taches. Ces facteurs représentant le contexte d'utilisation sont issus du modèle source CM et permettront, par conséquent, aux médecins et aux malades d'agir dans un environnement ubiquitaire et d'exploiter les différentes opportunités offertes. Notre application aura la particularité de pouvoir s'adapter aux changements continuels de la situation courante d'un utilisateur. Ce qui conduit automatiquement à la personnalisation des systèmes d'information ubiquitaires. Par exemple, un médecin aura la possibilité de consulter les dossiers de ses malades à partir de son cabinet, de sa maison ou de l'hôtel (en cas d'urgence). Les informations retournées seront personnalisées suivant les paramètres du contexte d'utilisation. Ainsi, le contenu diffère suivant la nature de l'utilisateur (médecin, étudiant,…), le mode d'affichage varie selon le dispositif utilisé, les paramètres régionaux sont fixés suivant la localisation et/ou les préférences de l'utilisateur, …etc.

Conclusion

Dans cette partie, nous avons présenté les grandes lignes de l'expérimentation d'un cas d'étude. L'objectif de cette expérimentation est de démontrer la faisabilité de notre proposition et de tenter de mettre en œuvre les concepts utilisés. Nous avons utilisé la plate-forme "Epsilon" qui propose plusieurs outils nécessaires pour mettre en pratique les différentes phases de notre approche proposée. Ainsi, le modèle métiers PIM a été construit en utilisant le langage UML. La modélisation du contexte nous a permis de construire un modèle contextuel (CM) en utilisant le langage UML. Les différentes phases du processus d'intégration du modèle contextuel dans le cycle de vie de la MDA ont été implémentées en utilisant les langages spécifiques offerts par Epsilon. Pour la comparaison et la vérification des éléments des deux modèles (PIM et CM), nous avons utilisé le langage spécifique ECL qui permet, à l'aide des "match-rules", d'identifier les correspondances qui existent entre ces éléments et de décider s'ils sont conformes ou non. Cette décision sur le type de matching nous permet d'appliquer l'opération de fusion à l'aide du langage EML en utilisant soit les "Merge-rules" soit les "Transform-rules". Le modèle fusionné, ainsi obtenu, sera transformé vers un modèle spécifique PSM suivant les spécifications technologiques de la plate-forme choisie (par exemple en C++).

CONCLUSION GÉNÉRALE ET PERSPECTIVES

Conclusion générale

La prise en compte des informations contextuelles pendant la conception des applications nous permet de cerner toutes les variations du contexte d'utilisation qui influencent l'exécution d'une tache par un utilisateur nomade et, aussi, de prévoir les conditions contextuelles de plusieurs situations jusque là inconnues. En effet, par la modélisation des données contextuelles et la formalisation d'un modèle purement contextuel, on peut assurer la prise en considération du contexte d'utilisation d'une application à travers les transformations subies par ce nouveau modèle.

Dans cette étude, nous avons présenté une nouvelle vision de l'approche MDA qui est capable d'introduire le contexte d'utilisation dans son cycle de vie et, par conséquent, produire des applications sensibles au contexte. Ce qui permettra de concevoir et réaliser des systèmes d'information adaptés et personnalisés qui peuvent, d'une part, vérifier les nouvelles caractéristiques de l'informatique ubiquitaire et, d'autre part, satisfaire les exigences des utilisateurs nomades.

Notre proposition passe par trois étapes. La première consiste à séparer les aspects contextuels des autres aspects (métiers et techniques). Cette séparation s'appuie sur les deux notions introduites, à savoir le processus 3TUP et le processus de développement en « **psi : Ψ** ». La seconde étape a pour objectif de formaliser les informations contextuelles sous forme d'un modèle. Pour la construction du modèle contextuel, nous avons utilisé une approche graphique basée sur le langage UML. La dernière étape propose une méthode qui permet d'intégrer le modèle contextuel dans le processus de développement de l'approche MDA, et ce, en utilisant l'opération de fusion de modèles.

Ce travail nous a permis de mieux comprendre l'importance de la prise en compte du contexte d'utilisation dans le développement des futures applications. Cette importance découle des changements continuels subis par la situation courante de l'utilisateur de plus en plus mobile et exigeant.

Perspectives

Notre vision s'étend jusqu'à la réalisation de systèmes d'informations « contextuellement paramétrables », c'est-à-dire que les applications conçues doivent renfermer des variables représentant toutes les informations contextuelles possibles et qui seront introduites selon les contraintes et les profils des utilisateurs. Les systèmes d'information ainsi obtenus pourront satisfaire les conditions d'adaptation et seront capables de fournir des informations personnalisées et pertinentes. Ce qui permet à l'utilisateur de s'adapter lui-même (son profil et ses préférences) et adapter l'application au contexte d'utilisation (environnement, utilisateurs voisins,…).

Nous prévoyons poursuivre cette étude par la construction d'un « mapping contextuel ». Celui-ci comprendra de nouvelles règles de transformation (relatives au contexte d'utilisation) qui guideront le passage d'un modèle PIM vers un modèle PSM.

Aussi, nous envisageons de construire un « méta-modèle contextuel ». Ce méta-modèle permettra aux concepteurs de s'y référencer pour produire des modèles contextuels plus conformes et plus structurellement homogènes.

BIBLIOGRAPHIE

[Abowd, 1997a] Abowd G.D., Dey A.K., Orr R., Brotherton J. « Context-Awareness in Wearable and Ubiquitous Computing ». 1st International Symposium on Wearable Computers (1997), pp 179-180, 1997.

[Abowd, 1997b] Abowd G.D., Atkeson C.G., Hong J., Long S., Kooper R., Pinkerton M. « Cyberguide: A mobile context-aware tour guide ». ACM Wireless Networks, 3(5): pp 421-433, October 1997.

[Adam, 2000] Adam E. «Modèle d'organisation multi-agent pour l'aide au travail coopératif dans les processus d'entreprise : application aux systèmes administratifs complexes», 2000.

[Alberto, 1993] Alberto T., Combemale P. «Comprendre l'entreprise. Théorie, gestion relations sociales». Paris : Nathan, 1993.

[Alonso, 2004] Alonzo G., Casati F., Kuno H., Machiraju V. « Web Services: Concepts, Architectures and Applications ». Springer-Verlag, 1rst édition, 2004.

[Asthana, 1994] Asthana A., Cravatts M., Krzyzanowski P. «An indoor wireless system for personalized shopping assistance». In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pp 69-74, Santa Cruz, California, December 1994. IEEE Computer Society Press. 1994.

[Banavar, 2000] Banavar G., Beck J., Gluzberg E., Munson J., Sussman J., Zukowski D. «Challenges :an application model for pervasive computing». In proceedings of the 6th annual international conference on mobile computing and networking, pp 266-274, ACM Press, 2000.

[Banavar, 2002] Banavar G., Bernstein A. « Software infrastructure and design challenges for ubiquitous computing applications» commun. ACM, 45(12) pp 92-96, 2002.

[Bauer, 2003] Bauer J. «Identification and modeling of contexts for different information Scenarios in air traffic», PhD thesis. Mar. 2003.

[Benselim, 2009a] Benselim M.S., Seridi-Bouchelaghem H. «Contextual adaptation of ubiquitous information systems», In Proceedings of the 2nd International Conference on Multimedia Computing and Systems (ICMCS'09) Morocco, April, 2009.

[Benselim, 2009b] Benselim M.S., Seridi-Bouchelaghem H. «Une approche pour le développement d'applications sensibles au contexte», Congrès INFORSID 2009, Toulouse, France, Mai 2009.

[Bézivin, 2001] Bézivin J., Gerbé O. « Towards a Precise Definition of the OMG/MDA Framework ». In Proceedings of the 16th International Conference on Automated Software Engineering, pp 273–280, November 2001.

[Brewington, 2000] Brewington B., Cybenko G. «Keeping up with the changing Web». IEEE Computer, 33(5):52-58, May 2000.

[Brown, 1996] Brown P.J. «The Stick-e Document: a Framework for Creating Context-Aware Applications». Electronic Publishing '96 (1996), pp 259-272, 1996.

[Brown, 1997] Brown P.J., Bovey J.D., Chen X. «Context-Aware Applications: From the Laboratory to the Marketplace». IEEE Personal Communications, 4(5) , pp 58-64, 1997.

[Brown, 1998] Brown P.J. « Triggering Information by Context ». Personal Technologies, 2(1) (1998), pp 1-9, 1998.

[Canals, 2002] Canals G., Nigay L., Pucheral P. « Mobilité : accès aux données et interaction homme-machine » Actes des deuxièmes assises nationales du GdR I3, Information – Interaction – Intelligence. Nancy, France, Décembre 2002. Cépadues Éditions, 2002.

[Chaari, 2005] Chaari T., Laforest F., Flory A. « Adaptation des applications au contexte en utilisant les services web ». Proceedings of the 2nd French-speaking conference on mobility and ubiquity computing, ACM International Conference Proceeding Series, Vol. 120 pp 111-118, Grenoble, France, 2005.

[Chen, 2000] Chen G., Kotz D., « A survey of context-aware mobile computing research ». Dartmouth Computer Science Technical Report TR2000-381, 2000.

[Davies, 1998] Davies N., Mitchell K., Cheverst K., Blair G. « Developing a Context Sensitive Tourist Guide ». 1st Workshop on Human Computer Interaction with Mobile Devices, GIST Technical Report G98-1, 1998.

[De Michelis, 1994] De Michelis G. «From the analysis of cooperation within work-processes to the design of CSCW Systems». Proceedings of the 15th Interdisciplinary Workshop on Informatics and Psychology: Interdisciplinary approaches to system analysis and design, Schaerding, Austria, 24-26 May 1994.

[Dey, 1997] Dey A.K., Abowd G.D. « CyberDesk: The Use of Perception in Context-Aware Computing ». 1st Workshop on Perceptual User Interfaces (1997), pp 26-27, 1997.

[Dey, 1998] Dey A.K. «Context-Aware Computing: The CyberDesk Project». AAAI 1998 Spring Symposium on Intelligent Environments, Technical Report SS-98-02, pp 51-54, 1998.

[Dey, 1999a] Dey A.K., Abowd G.D., Wood A. «CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services». Knowledge-Based Systems, 11, pp 3-13, 1999.

[Dey, 1999b] Dey A.K., Abowd G.D. «Towards a Better Understanding of context and context-awareness». Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, June 1999.

[Dey, 2001] Dey A.K., Abowd G.D., Salber D. «A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications». Human-Computer Interaction vol.16, 2001, pp 97-166, 2001.

[Donsez, 1999] Donsez D., Jean S., Lecomte S. « Evolution du rôle de la carte à microprocesseur dans les systèmes d'information distribués ». Technical Report 99-11, LIFL, Université Lille 1, France, Juin 1999.

[Frankel, 2003] Frankel D.S. «Model Driven Architecture: Applying MDA to Enterprise Computing». Wiley and OMG Press, 2003.

[Franklin, 1998] Franklin D., Flaschbart J. «All Gadget and No Representation Makes Jack a Dull Environment». AAAI 1998 Spring symposium on Intelligent Environments, Technical Report SS-98-02 (1998) pp 155-160, 1998.

[Held, 2002] Held A., Buchholz S., Schill A. «Modeling of Context Information for Pervasive Computing Applications». In Proc. 6th World Multiconference (SCI2002), Orlando, FL, 2002.

[Hull, 1997] Hull R., Neaves P., Bedford-Roberts J. «Towards situated computing». In Proceedings of the First International Symposium on Wearable Computers, Cambridge, Massachusetts, October 1997. IEEE Computer Society Press. 1997.

[Indulska, 2003] Indulska J., Robinson R., Rakotonirainy A., Henricksen K. «Experiences in Using CC/PP in context-Aware Systems ». In Proceedings 4th International Conference On Mobile Data Management, Melbourne. Australia, pp. 247-261, January 2003.

[Kleppe, 2003] Kleppe A., Warmer J., Bast W. « MDA Explained: The Model Driven Architecture: Practice and Promise ». Addison-Wesley, 1st édition, August 2003.

[Kolovos, 2006a] Kolovos D.S, Paige R.F., Polack F.A.C. «Merging Models with the Epsilon Merging Language (EML) ». In Proc. ACM/IEEE 9th International Conference on MDE Languages and Systems (Models/UML 2006), Italy, 2006.

[Kolovos, 2006b] Kolovos D.S, Paige R.F., Polack F.A.C. « eclipse Development Tools for Epsilon » In Eclipse Summit Europe, Eclipse Modeling Symposium, Esslingen, Germany, October 2006.

[Kolovos, 2008] Kolovos D.S, Paige R.F., Rose L.M., Polack F.A.C. «Epsilon». Epsilon Book From Department of Computer Science, University of York, UK. September 11, 2008.

[Korteum, 1998] Korteum G., Segall Z., Bauer M. « Context-Aware, Adaptive Wearable Computers as Remote Interfaces to "Intelligent" Environments ». 2nd International Symposium on Wearable Computers (1998) pp58-65, 1998.

[Leonhardt, 1998] Leonhardt U. «Supporting location-awareness in open distributed systems». PhD thesis, Imperial College of Science, Technology and Medicine, University of London, May 1998.

[Lopez, 2005] Lopez D. C. P. « Etude et applications de l'approche MDA pour des plates-formes de Services Web ». PhD thesis. 2005.

[McCarty, 1993] McCarty J. «Notes on formalizing contexts». In Proc. of the 13 international joint conference on artificial intelligence, Bajcsy Ed. M.Kaufmann, pp. 555-560, California, 1993.

[Mellor, 2004] Stephen J. Mellor S.J., Scott K., Uhl A., Weise D. «MDA Distilled: Principles of Model-Driven Architecture ». Addison-Wesley, 1st édition, March 2004.

[Muller, 2004] Pierre-Alain Muller P-A., Gaertner N. « Modélisation Objet avec UML ». Eyrolles, 514, 2004.

[OMG, 2001] OMG. «Model Driven Architecture (MDA) ». Document number ormsc/2001-07-01, July 2001.

[OMG, 2003a] OMG. « UML v 1.5, formal ». 2003-03-01, March 2003.

[OMG, 2003b] OMG. «MDA Guide V1.0.1», June 2003.

[OMG, 2003c] OMG. «Unified Modeling Langage: Superstructure version 2.0 Final Adopted Specification», OMG ptc/03-08-02. OMG, August 2003.

[Ou, 2006] Ou S., Georgalas N., Azmoodeh M., Yang K., Sun X. «A Model Driven Integration Architecture for Ontology-Based Context Modelling and CAA Development» . A. Rensink and J. Warmer (Eds.): ECMDA-FA 2006, LNCS 4066, pp. 188 – 197, 2006.

[Pascoe, 1998a] Pascoe J. «Adding Generic Contextual Capabilities to Wearable Computers». 2nd International Symposium on Wearable Computers (1998), pp 92-99, 1998.

[Pascoe, 1998b] Pascoe J., Ryan N.S., Morse D.R. «Human-Computer-Giraffe Interaction – HCI in the Field». Workshop on Human Computer Interaction with Mobile Devices. 1998.

[Pottinger, 2003] Pottinger R.A., Bernstein P.A. « Merging Models Based on Given Correspondences ». Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[Privat, 2007] Privat G., Ramparany F. « les interfaces contextuelles ». Publications France Télécoms R&D, 2007.

[Ranganathan, 2005] Ranganathan A., Al-Muhtadi J., Biehl J., Ziebart B., Campbell R., Bailey B. « Towards a pervasive computing benchmark, PerWare '05 Workshop on Support for Pervasive Computing ». Third IEEE International Conference on Pervasive Computing and Communications (PerCom 2005), pp 194–198, 2005.

[Roques, 2003] Roques P., Vallée F. « UML en action : De l'analyse des besoins à la conception en Java ». Eyrolles, 2003.

[Ryan, 1997] Ryan N., Pascoe J., Morse D. «Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant». Gaffney,V., van Leusen, M., Exxon, S. (eds.) Computer Applications in Archaeology, 1997.

[Salber, 1998] Salber D., Dey A.K., Abowd G.D. «Ubiquitous Computing: Defining an HCI Research Agenda for an Emerging Interaction Paradigm». Georgia Tech GVU Technical Report GIT-GVU-98-01 (1998).

[Schilit, 1994a] Schilit B., Adams N., Want R. «Context-Aware Computing Applications». 1st International Workshop on Mobile Computing Systems and Applications. pp 85-90, 1994.

[Schilit, 1994b] Schilit B., Theimer M. «Disseminating Active Map Information to Mobile Hosts». IEEE Network, 8(5) (1994) 22-32, 1994.

[Schilit, 1994c] Schilit B., Adams N., Want R. « Context-aware computing applications ». In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pages 85-90, Santa Cruz, California, December 1994. IEEE Computer Society Press. 1994.

[Schmidt, 1998] Schmidt A., Beigl M., Gellersen H.W. «There is more to context than location». In Proceedings of Workshop on Interactive Applications of Mobile Computing (IMC'98), Rostock, Germany, November 1998. Neuer Hochschulschriftverlag. 1998.

[Spreitzer, 1993] Spreitzer M., Theimer M. «Providing location information in a ubiquitous computing environment». In Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, pages 270-283, Asheville, NC, December 1993. ACM Press. 1993.

[Strang, 2003] Strang T., Linnhoff-Popien C., «Service Interoperability on Context Level in Ubiquitous Computing Environments». In Intl. Conf. on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w), L'Aquila, Italy, 2003.

[Strang, 2004] Strang T., Linnhoff-Popien C. «A context modelling survey» workshop on advanced context modeling, reasoning and management as part of ubicomp 2004, the 6th intl. conf. on ubiquitous computing, pp 33-40, sept 2004.

[TEA, 1998] TEA project, Esprit project 26900: Technology for enabled awareness (TEA), 1998.

[Vale, 2008] Vale S., Hammoudi S. «Towards Context Independence in Distributed context-aware applications by the Model Driven Approach». ACM SIPE'08, Sorrento, Italy, July 7, 2008.

[Van Laerhoven, 1999] Van Laerhoven K. «Online adaptive context awareness, starting with low-level sensors». Licentiaats thesis at the University of Brussels, Brussels, Belgium, May 1999.

[Want, 1992] Want R., Hopper A., Falcão V., Gibbons J. « The Active Badge location system ». ACM Transactions on Information Systems, 10(1):91-102, January 1992.

[Want, 1996] Want R., Schilit B.N., Norman I. Adams, Gold R., Petersen K., Goldberg D., Ellis J.R., Weiser M. «The ParcTab Ubiquitous Computing Experiment». In Tomasz Imielinski and Henry F. Korth, editors, Mobile Computing, chapter 2, Kluwer Academic Publishers, 1996.

[Ward, 1997] Ward A., Jones A., Hopper A. «A New Location Technique for the Active Office». IEEE Personal Communications 4(5) (1997) pp 42-47, 1997.

[Weiser, 1991] Weiser M. « The computer of the 21st century ». Scientific American, 265(3):66–75, September 1991.