REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABOU-BEKR BELKAID - TLEMCEN

THÈSE

Présentée à :

FACULTE DES SCIENCES – DEPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

DOCTORAT EN SCIENCES

Spécialité: Informatique

Par :

BOUDAA Boudjemaa

Sur le thème

Approche Dirigée par les Modèles pour le Développement des Applications Sensibles au Contexte à Base de Services

---

Soutenue publiquement le 15/02/2017 à Tlemcen devant le jury composé de :

Pr. CHIKH Azedine	Professeur	Université de Tlemcen	Président
Pr. CHIKH Mohammed Amine	Professeur	Université de Tlemcen	Directeur de thèse
Pr. HAMMOUDI Slimane	Professeur	ESEO, Angers (France)	Co-Directeur de thèse
Pr. BELALEM Ghalem	Professeur	Université d'Oran1 (Ahmed Ben Bella)	Examinateur
Pr. BENSLIMANE Sidi Mohamed	Professeur	ESI, Sidi-Belabbes	Examinateur
Dr. DIDI Fedoua	Maitre de Conférences	Université de Tlemcen	Examinatrice
Dr. HADJILA Fethallah	Maitre de Conférences	Université de Tlemcen	Invité

BP 119, 13000 Tlemcen - Algérie

Remerciements

Ce travail a été réalisé sous la direction de Monsieur **CHIKH Mohammed Amine**, professeur à l'université d'Abou Bekr Belkaid de Tlemcen, auquel j'exprime ma profonde reconnaissance pour la confiance et la liberté d'action qu'il m'a accordées.

Je tiens à remercier également Monsieur **HAMMOUDI Slimane** qui a codirigé ma thèse, pour m'avoir orienté vers ce sujet de recherche, pour ses orientations scientifiques précieuses et pour son accueil durant mes stages de recherche. Comme je remercie Monsieur le Directeur et l'ensemble de l'équipe du Département Informatique de l'Ecole ESEO d'Angers (France) pour leur sympathie et leur gentillesse.

Je souhaite adresser mes sincères remerciements à Monsieur **CHIKH Azedine** qui m'a fait l'honneur de présider le jury de thèse, et aux personnes qui ont accepté la tâche délicate d'examiner ce mémoire et qui ont eu la patience d'évaluer ce travail, Monsieur **BELALEM Ghalem**, Monsieur **BENSLIMANE Sidi Mohamed,** Madame **DIDI Fedoua** et Monsieur **HADJILA Fethallah.**

Enfin, je tiens à remercier qui, de près ou de loin, ont collaboré à l'aboutissement de ce travail.

A la mémoire de mes défunts parents.
A mon très cher beau-père Monsieur Bouziane Ahmed.
A mon épouse et mes enfants Asma, Ahmed et Rahaf.
A toute ma famille et ma belle-famille.
A tous mes collègues du département Informatique.
A tous ceux qui m'aiment et que j'aime très fort.

Résumé

Contexte : Le développement des applications sensibles au contexte à base de services a été considéré parmi les domaines de recherche les plus étudiés dans la dernière décennie. L'objectif était d'accompagner l'évolution rapide de la technologie des appareils informatiques mobiles en fournissant des services personnalisés capables d'interagir avec différentes situations contextuelles dans un environnement pervasive. A cet effet, de nombreux travaux de recherche ont préconisé le développement dirigé par les modèles (MDD) pour construire des applications sensibles au contexte à base de services. Cependant, les approches proposées ont présenté des méthodologies spécifiques sans utiliser des normes générales de développement qui peuvent être suivies par les développeurs. En outre, la plupart d'entre eux ont ignoré l'aspect d'adaptation dynamique à l'exécution qui doit caractériser ce type d'applications et aucune stratégie d'adaptation n'a été prise en compte dans leurs propositions.

Objectif : La présente thèse a pour but de proposer une approche générique dirigée par les modèles pour l'ingénierie des applications sensibles au contexte à base de services avec une méthodologie de développement de logiciels incluant une boucle de reconfiguration pour réaliser l'adaptation dynamique de ces applications.

Méthode : Cette approche met l'accent sur la combinaison de MDD et la modélisation orientée aspect (AOM) pour tirer parti de leurs avantages. AOM encapsule les différentes logiques de sensibilité au contexte séparément dans des modèles d'aspect appelé ContextAspect qui peuvent facilement être tissés dans la logique métier du service en fonction de l'évolution du contexte au fil du temps. La méthodologie de développement proposée comprend quatre phases (modélisation, composition, transformation et adaptation) qui agissent en conformité avec la technologie MDA.

Résultats : Les principaux résultats obtenus à l'aide de l'approche actuelle sont la possibilité de combiner la technologie MDA avec le paradigme orienté aspect dans une méthodologie de développement générique pour les applications sensibles au contexte à base de services, et le traitement de leur adaptation dynamique au moment de l'exécution en fonction de l'évolution du contexte.

Conclusion: Le développement des applications sensibles au contexte est une tâche complexe, lourde, et qui prend du temps. Cependant, l'expérience atteinte en mettant en œuvre la méthodologie proposée nous amène à croire que la combinaison de MDD et AOM est nettement bénéfique pour surmonter certaines lacunes reconnues de plusieurs approches existantes et de rendre cette tâche plus simple, plus facile et plus rapide.

Mots clés : Application sensible au contexte à base de service; ContextAspect; méthodologie dirigée par les modèles; Tissage d'aspect; Adaptation dynamique.

Abstract

Context: Context-aware service-based applications development has been considered among the most studied research fields in the last decade. The objective was to accompany the rapid technology evolution of mobile computing devices by providing customized services able to interact with different contextual situations of a pervasive environment. For this purpose, many research works have advocated Model-Driven Development (MDD) for building context-aware service-based applications. However, the proposed approaches have presented specific methodologies without using development standards, which may be followed by developers. In addition, most of them have ignored the dynamic adaptation aspect at runtime that should characterize such kind of applications and no adaptation strategy was considered in their proposals.

Objective: This thesis aims to propose a generic model-driven approach for context-aware service-based applications engineering with a software development methodology including a reconfiguration loop to achieve the dynamic adaptation of these applications.

Method: This approach focuses on the combination of MDD and Aspect Oriented Modelling (AOM) to take advantage of their benefits. AOM encapsulates different context-awareness logics separately in aspect models called ContextAspect that can be easily woven into the service's business logic according to the changing context over time. The proposed development methodology includes four phases (modelling, composition, transformation and adaptation) which act in conformance with the MDA technology.

Results: The main results gained by using the present approach are the possibility to combine the MDA technology with the aspect-oriented paradigm in a generic development methodology for context-aware service-based applications, and the handling of their dynamic adaptation at execution time according to the changes in the context.

Conclusion: The development of context-aware applications is a complex, cumbersome, and time-consuming task. However, the experience reached by implementing the proposed methodology leads us to believe that the involvement of MDD and AOM is significantly beneficial to overcome some recognised shortcomings of several existing approaches and to make this task simpler, easier and faster.

Keywords: Context-aware Service-based Application; ContextAspect; Model-Driven Methodology; Aspect Weaving; Dynamic Adaptation.

الملخص:

السياق: يعتبر تطوير التطبيقات المستندة إلى الخدمات الحساسة للسياق من بين المجالات البحثية الأكثر دراسة في العقد الماضي. وكان الهدف هو مرافقة التطور التكنولوجي السريع لأجهزة الكمبيوتر المحمول من خلال تقديم خدمات مخصصة قادرة على التفاعل مع الحالات السياقية المختلفة في بيئة واسعة الانتشار. لهذا الغرض، العديد من الأعمال البحثية قد دعت إلى التنمية الموجهة بالنماذج **(MDD)** لبناء التطبيقات المستندة إلى الخدمات الحساسة للسياق، إلا أن ما قدم وفق هذا النهج المقترح لا يتعدى منهجيات محددة خالية من إستخدام معايير التنمية العامة التي يمكن إتباعها من قبل المطورين. وأيضا، فقد تجاهل معظمهم جانب التكيف الحيوي في وقت التشغيل التي يجب أن يميز هذا النوع من التطبيقات حيث لم يتم تقديم أي استراتيجية للتكيف في مقترحاتهم.

الهدف: تهدف الأطروحة الحالية إلى اقتراح نهج عام قائم على النموذج من أجل هندسة التطبيقات الحساسة للسياق المستندة إلى الخدمات مع منهجية تطوير البرمجيات بما في ذلك حلقة إعادة التشكيل لتحقيق التكيف الحيوي لهذه التطبيقات.

الطريقة: يركز هذا النهج على الجمع بين **MDD** و نمذجة الجانب المنحى (**AOM**) للاستفادة من فوائدهما. **AOM** يقوم بتغليف مختلف الحالات الحساسة للسياق بشكل منفصل في نماذج مستقلة تدعى **ContextAspect**، والتي يمكن أن تنسج بسهولة في منطق الأعمال والخدمات وفقا لتغير السياق مع مرور الوقت. وتتضمن منهجية التطوير المقترح أربع مراحل (النمذجة، التركيب، والتحول والتكيف) التي تعمل كلها في توافق مع تكنولوجية **MDA** .

النتائج: النتائج الرئيسية التي يمكن اكتسابها باستخدام النهج المقترح هي إمكانية الجمع بين تكنولوجية **MDA** و مفهوم الجانب المنحى في منهجية تطوير عامة للتطبيقات الحساسة للسياق و القائمة على الخدمات، والتعامل مع تكيفها الديناميكي في وقت التنفيذ وفقا للتغيرات التي قد تطرأ على السياق.

الخلاصة: تطوير التطبيقات الحساسة للسياق هو مهمة معقدة، ومرهقة، وتستغرق وقتا طويلا. ومع ذلك، فإن التجربة التي توصلنا إليها بتنفيذ المنهجية المقترحة تقودنا إلى الاعتقاد بأن إشراك **MDD** و**AOM** هو مفيد كثيرا للتغلب على بعض أوجه القصور المعترف بها في المناهج الموجودة وجعل هذه المهمة أسهل وأسهل وأسرع.

الكلمات المفتاحية: تطبيق حساس للسياق و مستند إلى الخدمة؛ **ContextAspect**؛ منهجية موجهة بالنموذج؛ نسج الجانب؛ التكيف الحيوي.

Liste des Publications Personnelles

Journal Publications

Boudjemaa Boudaa, Slimane Hammoudi, Leila Amel Mebarki, Abdelkader Bouguessa, Mohammed Amine Chikh: *An aspect-oriented model-driven approach for building adaptable context-aware service-based applications*. Science of Computer Programming (Journal Elsevier) 09/2016; DOI:10.1016/j.scico.2016.08.009. (Indexed Thomson Reuters, Scopus,…)

Boudjemaa Boudaa, Slimane Hammoudi, Abdelkader Bouguessa, Leila Amel Mebarki, Mohammed Amine Chikh: *On Sustaining Dynamic Adaptation of Context-Aware Services*. Journal EAI Endorsed Transactions on Context-aware Systems and Applications, Volume 3. 03/2015; 15(3-3). DOI:10.4108/casa.2.3.e4. (Indexed DBLP,…)

Boudjemaa Boudaa, Slimane Hammoudi, Mohammed Amine Chikh: *An Ontology-Based Context Model for Ubiquitous Applications*. Journal Progress in Machines and Systems. Volume 2, Issue 1, pages 18-28. 04/2013.

Boudjemaa Boudaa, Olivier Camp, Slimane Hammoudi, Mohammed Amine Chikh: *The Development and the Features of the Context-Aware Services*. Journal of Information Organization. Volume 2, Issue 3, pages 37-49. 03/2012.

Conference Proceedings

Boudjemaa Boudaa: *Towards a Model-Driven Requirements Specification of Context-Aware Services*. Proceedings of the 2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems, Morocco; 11/2014. (Indexed ACM, DBLP)

Boudjemaa Boudaa, Slimane Hammoudi, Abdelkader Bouguessa, Mohammed Amine Chikh: *Supporting Runtime Adaptation of Context-Aware Services*. Proceedings of the 3rd International Conference on Context-Aware Systems and Applications (ICCASA'2014), Dubai, United Arab Emirates; 10/2014. (Indexed ACM, DBLP)

Boudjemaa Boudaa, Slimane Hammoudi, Mohammed Amine Chikh: *Ontology-Based Context Modeling for Context-Aware Applications*. Proceedings of the 5th International Conference on Web and Information Technologies (ICWIT'13), Hammamet, Tunis, 5/2013.

Boudjemaa Boudaa, Slimane Hammoudi, Mohammed Amine Chikh: *ODM-based modeling for user-centered context-aware mobile applications*. Proceedings of the 3rd IEEE Information Technology and e-Services (ICITeS'2013), Sousse, Tunis; 03/2013. (Indexed IEEE Xplore)

Boudjemaa Boudaa, Olivier Camp, Slimane Hammoudi, Mohammed Amine Chikh: *Model Driven Development of Context Aware Services: Issues, techniques and review*. Proceedings of the 2nd IEEE International Conference on Information Technology and e-Services (ICITeS'2012); Sousse, Tunis; 03/2012. (Indexed IEEE Xplore)

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

1.1 Contexte et Motivation

De nos jours, le monde se dirige vers la dépendance totale sur les appareils modernes mobiles (Smartphone, GPS, ...) pour gérer la vie professionnelle et sociale, et cela comme a été prédit par Weiser [129] il y a plus de deux décennies en introduisant le terme «informatique ubiquitaire» ou «pervasive». Il a idéalisé un environnement intelligent doté de capteurs ayant pour objectif de réaliser certaines activités sans l'interférence humaine. La caractéristique essentielle des applications pervasives est d'anticiper le comportement de l'utilisateur en réagissant sans son intervention et de manière transparente. L'objectif premier d'un système pervasif est de permettre de répondre au besoin de l'utilisateur où il se trouve et à n'importe quel moment. L'importance croissante de ces applications fait que des nombreux travaux récents s'y intéressent.

Une sous classe des applications pervasives désignée par applications sensibles au contexte a conduit à des activités de recherche intense durant les 15 dernières années[72]. Malheureusement, ceux-ci soient ne donnent pas des réalisations concrètes de leurs approches ou ils incluent certaines détails techniques de la plateforme d'exécution pendant la conception de ce genre d'applications [10]. L'inclusion des détails techniques de la mise en œuvre dans le processus de développement depuis le début rend la conception des applications sensibles au contexte lourde et prendre beaucoup de temps. En outre, elle réduit la possibilité de la réutilisation de ces applications pour d'autres environnements technologiques d'exécution.

Pour combler cette lacune, récemment, certains travaux de recherche ont préconisé le développement dirigé par les modèles (MDD) [28, 109] en mettant l'accent sur les modèles [23]. Néanmoins, les approches proposées présentent des méthodologies spécifiques et ne se basent pas sur l'utilisation des standards de développement généraux qui peuvent être suivis par les développeurs, et donc, elles restent confinés à eux-mêmes. En outre, la plupart d'entre eux n'ont pas pris en compte l'adaptation dynamique à l'exécution qui caractérisent les applications sensibles au contexte et, en conséquence, l'absence de stratégies d'adaptation est constatée dans leurs propositions.

Les applications sensibles au contexte évoluent dans des environnements pervasifs connus par la richesse d'information de «contexte». Ces applications doivent adapter leurs comportements selon les variations des informations contextuelles en utilisant des mécanismes de «sensibilité au contexte».

Plusieurs définitions de contexte ont été fournies dans la littérature dont la majorité est résumée dans [13, 5]. La plus populaire est donnée par Dey et al., où le contexte est considéré comme « *Toute information qui peut être utilisée pour caractériser la situation des entités (une entité étant une personne, un lieu ou un objet) qui sont considérés comme pertinents pour l'interaction entre un utilisateur et une application, y compris l'utilisateur et l'application eux-mêmes. Le contexte est typiquement l'emplacement, l'identité et l'état des personnes, des groupes et des objets informatiques et physiques* »[47]. Dans ce travail, nous allons baser sur cette définition pour présenter notre approche de développement des applications sensibles au contexte.

De même, il y avait de nombreuses tentatives pour définir la terminologie de sensibilité au contexte introduite, pour la première fois, par Schilit et al. [108]. Pascoe dans [101] définit la sensibilité au contexte comme « *La capacité d'un programme ou d'un dispositif pour détecter ou capturer différents états de son environnement et de lui-même* ».

En fonction de cette définition, une application sensible au contexte et basée sur les services doit avoir la capacité de capturer le contexte et d'adapter son comportement suivant les changements de ce contexte pour fournir des informations et/ou des services pertinents à l'utilisateur final.

En conclusion, la réalisation de ce type d'applications a besoin des méthodologies de développement génériques garantissant d'une part, l'acquisition et la modélisation du contexte et d'autre part, le bon fonctionnement de l'adaptation au contexte de manière facile et rapide autant que possible.

La présente thèse rentre dans la cadre du domaine de l'ingénierie des d'applications sensibles au contexte, et vient dans le but de répondre à ce besoin par la proposition d'une approche de conception et de développement des applications sensibles au contexte à base de services.

1.2 Objectif et Étapes de Recherche

Notre travail a pour objectif de proposer une approche pilotée par les modèles pour l'ingénierie des applications sensibles au contexte à base de services avec une méthodologie de développement générique introduisant une boucle de reconfiguration afin de réaliser l'adaptation dynamique.

Cette approche met l'accent sur la combinaison du développement dirigé par les modèles (MDD) et la modélisation orientée aspect (AOM) [87] pour tirer partie de leurs avantages et trouver un bon compromis entre l'abstraction et l'exécution. La puissance des MDD réside dans l'utilisation des modèles et la capacité de transformer ces modèles pour concevoir des applications sensibles au contexte à base de services à haut niveau indépendamment de tout détail technique de plateforme d'exécution. AOM, d'autre part, est utile pour séparer les différents scénarios d'adaptation sensibles au contexte dans des modèles d'aspects prêts à être tissés dans le modèle de l'application de base, que ce soit au niveau de conception ou d'exécution en fonction du contexte qui est souvent inconstant.

Ces deux paradigmes fondamentaux (MDD et AOM) de génie logiciel sont utilisés pour soutenir la qualité de notre approche de développement des applications sensibles au contexte.

Avec une manière très succincte, nous citons, ci-après, les principales étapes de re-

cherche menées pour réaliser notre approche :

- La conception d'un modèle à base d'ontologie pour la représentation des informations de contexte de tout domaine d'application.
- L'élaboration d'une architecture logicielle des applications sensibles au contexte conforme à la technologie MDA de l'OMG [91].
- La proposition d'un métamodèle appelé « ContextAspect » afin d'encapsuler les différentes logiques sensibles au contexte en fonction des différentes situations contextuelles.
- La proposition d'une méthodologie dirigée par les modèles pour le développement des applications sensibles au contexte à base de services. Le cycle de vie de cette méthodologie comprend quatre phases (modélisation, composition, transformation et adaptation) agissent en conformité avec la technologie MDA.
- La focalisation sur la dernière phase pour son importance dans ce genre d'applications, et cela par l'intégration d'une boucle de contrôle et de reconfiguration pour réaliser l'adaptation dynamique des applications sensibles au contexte.
- Et en fin, la validation de notre approche par un exemple d'illustration sur une plateforme prometteuse tel que FraSCAti [112].

1.3 Organisation du mémoire

En plus de ce chapitre introductif, ce mémoire s'articule autour des huit autres chapitres. Les chapitres 2, 3 et 4 constituant la première partie de ce document et qui dressent les différents axes sur lesquels sera basé notre travail et un état de l'art sur les travaux de la littérature. Dans la deuxième partie qui contient les chapitres 5, 6, 7 et 8, nous présenterons en détail les différentes contributions attendues de ce travail. Et le chapitre 9 pour conclure cette thèse. Ci-après, les objectifs des huit chapitres :

Chapitre 2 : explique l'informatique pervasive et les applications sensibles au contexte dont les concepts de base sont définis avec des exemples.

Chapitre 3 : explicite le nouveau paradigme du Développement Dirigé par les Modèles (MDD) ainsi que ses fondements essentiels.

Chapitre 4 : expose un état de l'art des principaux travaux de développement des applications sensibles au contexte à base de services.

Chapitre 5 : présente un modèle à base d'ontologie selon la spécification ODM pour représenter les informations de contexte et qui peuvent être recueillies à travers des capteurs matériels ou logiciels.

Chapitre 6 : présente notre approche de développement des applications sensibles au contexte à base de services par la présentation d'une architecture logicielle conforme à MDA et un cycle de développement composé de quatre phases (modélisation, composition, transformation et adaptation). Les trois premières phases sont décrites en détail avec une étude de cas « SAMU ».

Chapitre 7 : détaille la dernière phase de l'adaptation qui est importante pour ce type d'applications en déroulant la boucle de contrôle MAPE-K sur un scénario de changement de comportement du service « Evacuation » de l'exemple SAMU.

Chapitre 8 : afin de valoriser et d'évaluer notre approche, ce chapitre présente les points forts et les limites de notre proposition en la comparant avec les principaux travaux de la

littérature.

Chapitre 9 : à la fin, ce chapitre donne une conclusion générale incluant une synthèse sur les contributions apportées par ce travail ainsi qu'un aperçu sur les perspectives de travaux futurs.

Première partie

Contexte Technologique et État de l'Art

Chapitre 2

L'informatique pervasive et les applications sensibles au contexte

2.1 Introduction

Ces dernières années ont vu apparaitre toute une gamme d'outils informatiques d'un type nouveau capables d'accompagner les activités quotidiennes de l'être humain. Par exemple, des téléphones portables utilisables comme cartes de crédit ou qui passent automatiquement en mode silencieux lors de l'entrée en réunion. Les applications ont également évolué pour s'adapter à ces nouveaux outils, qualifiés de pervasifs. L'informatique pervasive, également appelée ubiquitaire, s'occupe du développement des applications qui automatisent ces activités non traditionnelles. Le terme «informatique ubiquitaire» a été présenté pour la première fois en 1991, par Weiser [129]. Il a idéalisé un environnement intelligent doté de capteurs ayant pour objectif de réaliser certaines activités sans l'interférence humaine. La caractéristique essentielle des applications pervasive est d'anticiper le comportement de l'utilisateur en réagissant sans son intervention et de manière transparente.

L'importance croissante de l'informatique pervasive fait que des nombreux travaux récents s'y intéressent par la proposition d'un nouveau genre d'applications dites sensibles au contexte. La sensibilité au contexte concerne la perception de l'environnement pour interagir plus naturellement avec l'utilisateur. Cette perception passe par l'utilisation de capteurs de l'environnement physique, de matériels auto-descriptifs, de description des personnes, etc. Le contexte fournit un grand éventail d'informations qui permet au système pervasif d'agir de façon adaptée.

Dans ce chapitre, nous allons présenter toutes les notions de base de l'informatique pervasive et les applications sensibles au contexte.

2.2 L'informatique pervasive

En 1991, Mark Weiser a inventé l'expression de l'informatique pervasive (également appelée ubiquitaire, diffuse, ambiante, nomade,...). Il parlait alors d'une informatique invisible présente en tout lieu, à tout moment et en toute chose. Il a idéalisé un environnement intelligent doté de capteurs ayant pour objectif de réaliser certaines activités sans l'interférence humaine.

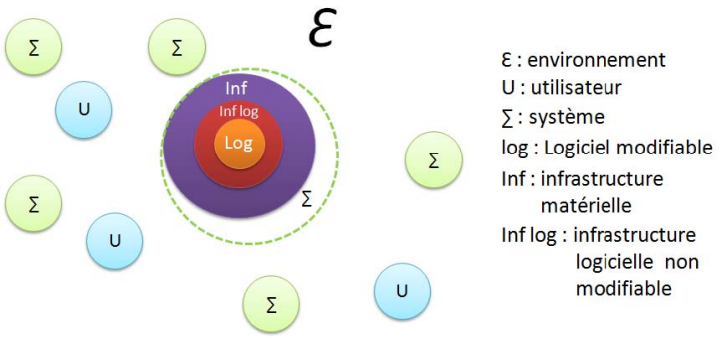


FIGURE 2.1 – Modèle général de l'Informatique Pervasive

Actuellement, avec la miniaturisation des matériels informatiques, de nombreux objets informatisés se trouvent intégrés dans notre quotidien et font peu à peu disparaitre l'idée de l'ordinateur personnel comme le seul assistant numérique informatisé. Ainsi, Gaëtan Rey et al., [105] ont défini un système d'informatique pervasive comme :

- « Multi-utilisateurs », puisque dans un même espace de nombreux utilisateurs peuvent être amenés à utiliser les mêmes ressources et à communiquer de manière concurrente.
- « Multi-dispositifs », car de nombreux objets physiques potentiellement informatisés et communicants peuvent nous entourer.
- « Multi-environnements », puisqu'ils sont mobiles tout comme les utilisateurs et peuvent changer d'emplacement et donc d'environnement.
- « Multi-applications », puisque de nombreuses applications reposant sur ces objets communicants peuvent être conçues.

L'informatique pervasive repose donc sur un ensemble d'entités environnementales en interaction (Figure 2.1). D'un point de vue informatique, ces entités sont de deux catégories [105] :

- Une entité ambiante peut être un « être vivant », c'est-à-dire un utilisateur ou plus généralement une entité dotée de capacités sensori-motrices, voire cognitives.
- Elle peut-être aussi un système informatisé. Ces systèmes reposent alors sur une infrastructure d'objets physiques appelés aussi dispositifs, qui peuvent offrir à leur tour une interface logicielle, appelée infrastructure logicielle, pour le reste du système informatisé. C'est sous ces contraintes que la partie logicielle modifiable du système permet *in fine* de mettre en œuvre de nouvelles fonctionnalités pour de nouvelles applications.

D'une façon générale et en se basant sur la définition de [129], « **L'informatique pervasive rend l'information/service disponible partout et à tout moment** ». Actuellement les applications informatiques fonctionnent dans une variété de nouveaux paramètres, par exemple, embarqués dans des voitures ou des appareils portables. Ils utilisent des informations sur leur contexte pour réagir et s'adapter aux changements dans l'environnement informatique. Ils deviennent de plus en plus pervasives et sensibles au contexte.

2.3 Les applications sensibles au contexte

L'objectif premier d'un système pervasif est de permettre de répondre au besoin de l'utilisateur où qu'il se trouve et à n'importe quel moment. La caractéristique essentielle des applications pervasives est d'anticiper le comportement de l'utilisateur en réagissant sans son intervention et de manière transparente. Par exemple, des téléphones portables utilisables comme cartes de crédit ou qui passent automatiquement en mode silencieux lors de l'entrée en réunion, des réfrigérateurs capables de commander par Internet des produits en fonction du besoin, des maisons qui adaptent la coloration de l'éclairage en fonction de l'humeur des habitants, des dispositifs GPS qui guident le conducteur en tenant compte non seulement du trajet mais également du trafic routier et de la dépense en carburant.

L'importance croissante de ces applications fait que des nombreux travaux récents s'y intéressent. Abowd et al. [3] proposent de classer ces travaux en quatre axes : les interfaces naturelles, la capture et l'accès automatisé des activités humaines, l'informatique du quotidien et l'informatique sensible au contexte. Le travail de cette thèse se situe dans ce denier axe. L'informatique sensible au contexte [5] se focalise sur le développement des applications qui prennent en compte le contexte d'utilisation et qui fournissent des solutions adaptées aux besoins des utilisateurs. Dans ce qui suit, nous allons aborder avec des exemples les concepts de base qui rentrent dans le développement de ces applications sensibles au contexte.

2.3.1 Contexte

Dans le domaine de l'informatique pervasive, de nombreuses recherches de développement des systèmes sensibles au contexte ont été proposées mais différemment et sans connaitre un consensus sur la définition de contexte et sa modélisation. Dans cette section vous voudrions donner les définitions de contexte les plus proches et les plus répandues avec quelques exemples.

2.3.1.1 Définitions

Qu'entend-t-on par « information de contexte » dans les systèmes sensibles au contexte ? Quelle est la signification du contexte qui a fait l'objet de nombreux travaux ? Diverses définitions du terme sont données dans la littérature et résumées dans [13].

Au début, les définitions ont été données pour des contextes d'applications spécifiques en énumérant les concrètes entités contextuelles. Par exemple, les auteurs de [30] définissent le contexte comme étant des informations sur la localisation, l'identité des personnes à proximité et les conditions physiques. Dans[107], Les auteurs ajoutent à cette définition la notion de temps.

D'autres définitions sont extrêmement large, la plus populaire est donnée par [46] : « **Le Contexte est toute information qui caractérise la situation d'une entité. Une entité étant une personne, un lieu ou un objet considéré comme pertinent relativement à une interaction entre un utilisateur et une application, incluant l'utilisateur et l'application eux-mêmes** ». Les auteurs donnent une définition générale qui peut être utilisée dans un large champ d'applications sensibles au contexte. Pour affiner leur définition, ils identifient quatre catégories de contexte qui leur

semblent pratiquement plus importantes que d'autres. Il s'agit de la localisation, l'identité (l'utilisateur), activité (Etat) et le temps [47].

Dans [132], l'auteur approuve cette dernière définition et affirme qu'elle couvre tous les travaux proposés dans le contexte. Cependant, il la considère comme étant une définition générale qui ne se limite pas au contexte. Ainsi, il propose sa propre définition dans laquelle limite le contexte à «**un ensemble d'informations, qui est structuré et partagé. Il évolue et est utilisé pour l'interprétation** »

La définition proposée dans [37] présente, également, le contexte comme étant organisé de façon hiérarchique. Dans cette proposition, les auteurs distinguent entre les informations environnementales qui déterminent le comportement des applications mobiles et qui sont pertinentes à l'application. Ils définissent ainsi le contexte comme « **l'ensemble des états environnementaux et les paramètres qui déterminent chacun un comportement d'une application ou dans lequel un évènement d'application se produit et qui est intéressant pour l'utilisateur** ».

Comme indiqué précédemment, il est difficile de donner une définition complète du terme contexte et, en fait, la notion de contexte n'est pas universelle mais relative à une situation et au domaine de l'application [30].

Typiquement, le contexte devrait contenir des informations sur l'identité de l'utilisateur, sur son activité, sur le moment où le contexte est capturé et sur la localisation du client terminal. Fondamentalement, il convient de répondre aux questions suivantes « Qui ? », « Quoi ? », « Quand ? » et « Où ? » et, idéalement, devrait permettre au système de répondre à une dernière question : « Pourquoi ? ».

2.3.1.2 Exemples

Quelques exemples d'entités et leurs contextes associés :

- Dispositif utilisé : bande passante de la connexion réseau, disponibilité de la mémoire, charge du processeur, taille d'écran et modes d'interaction avec l'utilisateur.
- Utilisateur : identité de l'utilisateur, sa langue maternelle, sa localisation géographique.
- Salle : niveau de lumière ou de bruit, température, ...

2.3.1.3 Types de contexte

Nous distinguons deux types de contexte selon sa nature :

Le contexte peut être **statique** (information qui ne devrait pas changer, par exemple, langue maternelle) ou **dynamique** (informations qui change au fil du temps, comme les conditions météorologiques). Donc, la prise en compte des deux types de contexte est crucial dans tous les propositions de développement des applications sensibles au contexte et, particulièrement, le contexte dynamique qui nécessite une adaptation au moment de l'exécution pour des réponses adéquates et rapides à l'utilisateur sans aucune intervention de sa part.

Une application sensible au contexte en cours d'exécution doit être adaptée de manière dynamique lors de l'exécution si un changement de contexte se produit, principalement dans les systèmes en temps réel où il n'y a pas de temps à perdre (soins de santé, circulation de l'air ; trafic aérien, ...). Par exemple, l'annulation d'un vol (comme une adaptation

sensible au contexte) doit être menée s'il y a une chute forte de neige (comme une information de contexte).

En conséquence, l'adoption d'une stratégie d'adaptation dynamique à l'exécution devient très essentiel dans le cycle de vie des applications sensibles au contexte (voir plus loin, la Sous-section 6.5.4et le Chapitre 7).

2.3.1.4 Modélisation du contexte

Dans la littérature de modélisation de contexte, nous trouvons plusieurs modèles contextuels proposés dans le domaine de l'informatique pervasive [120, 133, 16]. D'où, nous pouvons les classer comme suit :

Les premiers modèles étaient de type « clé-valeur » qui a emprunté le mécanisme de base de données (relationnelles ou XML) pour mettre la valeur de l'information contextuelle comme une variable d'environnement (ou clé).

Par la suite, les méthodes traditionnelles de génie logiciel ont été appliqué pour représenter le contexte : le modèle graphique comme ORM (Object-Role modeling) et UML, et le modèle orienté objet. Le modèle graphique fournit une vue claire et intuitive de contexte en décrivant les faits et les propriétés en tant que nœuds et les relations entre eux comme des arêtes. Le modèle orienté objet permet l'usage de leurs propres caractéristiques de l'encapsulation et de la réutilisation pour introduire l'abstraction et le mécanisme efficace de la classification pour la modélisation du contexte.

Par rapport à ces approches de modélisation contextuelles, le modèle logique ne pas se préoccuper de la façon dont le contexte est organisé ou représenté. Mais, il fournit un modèle formel et abstrait en utilisant le raisonnement avec une partie de potentiel disponible de contexte et la résolution de la compatibilité entre les différents contextes.

Cependant, les modèles ontologiques ont les avantages des approches orientées objet et de la logique de modélisation. ils fournissent un moyen formel pour modéliser le contexte dans des terminologies bien structurées, et soutiennent également le raisonnement formel. Selon [120, 16, 71], les ontologies offrent un meilleur formalisme pour construire les terminologies consensuelles du domaine de contexte d'une façon formelle afin qu'ils puissent être plus facilement partagées et ré-utilisées par les différents partenaires des environnements mobiles et ubiquitaires.

La modélisation avec les ontologiques ont des avantages évidents en ce qui concerne le soutien à l'interopérabilité et de l'hétérogénéité. Ces avantages permettent de considérer que les modèles de contexte à base d'ontologie sont les plus appropriés pour représenter les informations de contexte. Dans le chapitre 5, nous présenterons notre modèle de représentation de contexte en se basant sur les ontologies.

2.3.2 Sensibilité au contexte

La principale caractéristique de l'informatique pervasive est le changement dynamique de leurs environnements ou bien, plus précisément, leurs contextes. Pour mieux aider l'utilisateur dans ses tâches quotidiennes, les systèmes informatiques pervasifs doivent tenir compte du contexte global et adapter leurs services aux utilisateurs selon le changement de ce dernier. Cette aptitude est connue sous le nom de « sensibilité au contexte ».

2.3.2.1 Définitions

La terminologie de sensibilité au contexte a été évoquée pour la première fois par Schilit et al., dans [108] et présentée comme « **un logiciel qui s'adapte en fonction de son lieu d'utilisation, la collection des personnes et des objets à proximité, ainsi que les modifications apportées à ces objets dans le temps** ». Depuis lors, il y a eu de nombreuses tentatives pour définir cette terminologie. Dans [101], l'auteur définit la sensibilité au contexte comme « **la capacité d'un programme ou d'un dispositif à sentir ou capturer différents états de son environnement et de lui-même** ».

Compte tenu de ces deux définitions, une application sensible au contexte doit avoir la capacité de capturer les entités contextuelles nécessaires de son environnement d'exécution, les utiliser pour adapter son comportement et, enfin, présenter les services disponibles et appropriés à l'utilisateur.

Dans [46], les auteurs introduisent une autre définition dans laquelle ils insistent sur l'utilisation du contexte et de la pertinence des informations de contexte. Les auteurs estiment qu'« **un système est sensible au contexte s'il utilise le contexte pour fournir des informations pertinentes et/ou des services à l'utilisateur, où la pertinence dépend de la tâche de l'utilisateur** ».

Les applications sensibles au contexte doivent donc être capables de collecter des informations de contexte, de les interpréter, de les analyser et d'adapter, en conséquence, leur comportement aux changements de contexte pour satisfaire au mieux l'utilisateur final.

Ainsi, Dey [47] et Abowd [3] définissent trois catégories de fonctionnalités des applications sensibles au contexte, à savoir :

- Les informations et les services qui peuvent être présentés à l'utilisateur dans le contexte courant. Ceci inclut la sélection d'informations de proximité ("où est la banque la plus proche ?") et de services (interface modifiable) ;
- L'exécution automatique d'un service dans un certain contexte. Ceci inclut les actions initiées par des déclencheurs de contexte ("context triggered actions"), par exemple un téléphone qui change son volume d'écoute en fonction du bruit ambiant ;
- L'étiquetage du contexte à l'information pour une restitution ultérieure.

2.3.2.2 Exemples

Quelques exemples de la sensibilité au contexte :

- Avertissement lorsqu'une personne est à proximité.
- Affichage d'une position géographique.
- Ajustement de sa vitesse en fonction de la distance du véhicule qui est devant.
- Démarrage d'une synchronisation lorsque le réseau est rétabli.
- Modification de la température de la pièce en fonction des personnes présentes.
- Arrêt de la radio lors de la réception d'un appel téléphonique.
- Message SMS visuel ou vocal (si je suis dans ma voiture).

2.3.2.3 Architecture générale d'un système sensible au contexte

La séparation entre la détection et l'utilisation du contexte est nécessaire afin d'améliorer l'extensibilité et la réutilisabilité dans le système [118]. Dans cette perspective,

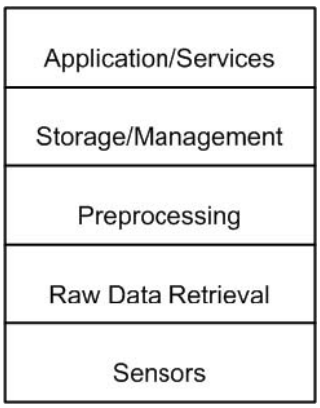


FIGURE 2.2 – Architecture conceptuelle pour les systèmes sensibles au contexte

la Figure 2.2 proposée dans [12] illustre une architecture en couches pour les systèmes sensibles au contexte, permettant de séparer les parties détection et utilisation, en y ajoutant la partie interprétation et raisonnement. Une description détaillée de ces différentes couches est donnée dans [12].

D'après plusieurs travaux [34, 74, 64], un système sensible au contexte a plusieurs composants, tel que un capteur de contexte, stockage du contexte, raisonneur du contexte et consommateur du contexte. Ces composants sont logiquement séparés des applications qu'ils supportent.

Ainsi, dans [124], les auteurs présentent et décrivent les éléments de base qui composent les systèmes sensibles au contexte dans l'environnement des services Web (voir Figure 2.3). Ils proposent une séparation explicite [118] entre :

- Les services sensibles au contexte et les applications : ils utilisent les informations du contexte et les outils supportés, afin de réagir "intelligemment" (être sensible au contexte).
- Les composants de la sensibilité au contexte : ils aident les applications et les services en captant et fournissant les informations du contexte. Des exemples de ces composants sont les capteurs, les enregistreurs de contexte et les moteurs de raisonnement sur le contexte.

2.4 Les applications sensibles au contexte et SOA

En observant dans les travaux proposés pour le développement des applications sensibles au contexte, nous trouvant que la majorité [74, 116, 115, 72, 73] adoptent et préconisent la réalisation de telles applications sur des plateformes basées sur l'architecture orientée services (calque de l'anglais service oriented architecture, SOA) pour les gains qu'elle apporte tel que le faible couplage et l'interopérabilité.

2.4.1 L'architecture orientée services (SOA)

Le SOA [50]est une architecture logicielle permettant la conception des systèmes à base d'un assemblage de services développés dans différents langages de programmation, déployés sur différentes plateformes et systèmes d'exploitation. Chaque service représente

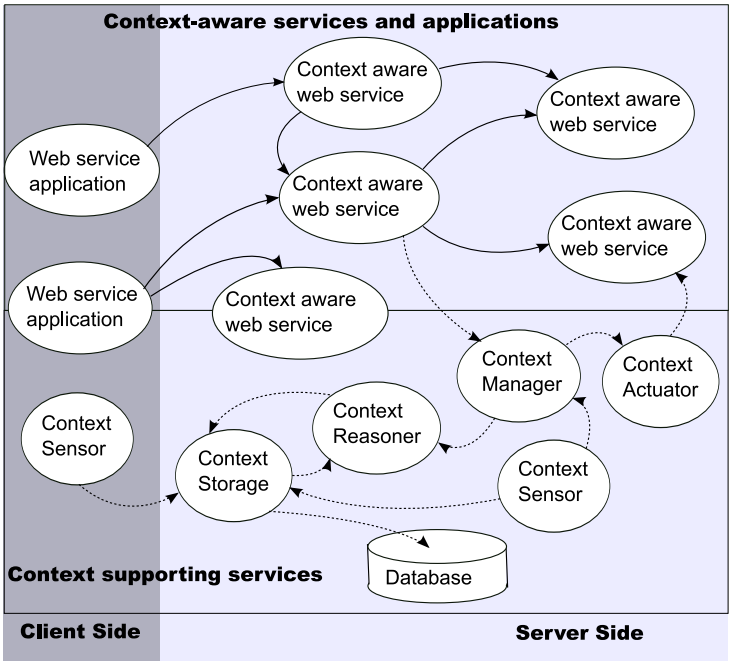


FIGURE 2.3 – Composants de base d'un système sensible au contexte dans l'environnement des services Web

un composant logiciel autonome de traitement et de gestion de données, communiquant avec son environnement à l'aide de messages.

Ce terme est apparu au cours de la période 2000-2001 et concernait à l'origine essentiellement les problématiques d'interopérabilité syntaxique en relation avec les technologies d'informatique utilisées dans les entreprises.

L'objet de base dans une architecture orientée services est la notion de service, c'est-à-dire une fonction encapsulée dans un composant que l'on peut interroger à l'aide d'une requête composée d'un ou plusieurs paramètres d'entrée et fournissant un ou plusieurs résultats. Avec SOA, chaque service doit être indépendant des autres afin de garantir sa réutilisabilité et son interopérabilité.

Plusieurs avantages seront gagnés par l'adoption d'une architecture SOA dans le domaine de développement logiciel [50], à savoir :

- Les services sont réutilisables : ils sont conçus de façon à ce qu'ils puissent être réutilisés ultérieurement.
- Les services interagissent à travers un contrat formel : pour pouvoir interagir, les services utilisent un contrat formel qui décrit chaque service et définit les termes de l'échange d'information.
- Les services sont faiblement couplés : ils sont conçus pour interagir avec le minimum d'interdépendances.
- Les services font abstraction de la logique métier : la seule partie visible du service est la partie exposée par le contrat de service (appelée interface publique). La logique sous-jacente du service est invisible et hors de portée du demandeur du service.
- Les services sont composables : les services peuvent être associés entre eux pour réaliser une fonction particulière (ex. processus métier). Ceci permet la représentation de la logique métier selon plusieurs niveaux de granularité et facilite la réutilisation et la création de plusieurs niveaux d'abstraction.

– Les services sont autonomes : les tâches accomplies par un service possèdent des limites. Le service possède un contrôle suivant cette limite et ne dépend pas d'autres services pour accomplir sa tâche.

La composition, le faible couplage et la flexibilité de réaction garanties par ces avantages permettent l'architecture orientée services de contribuer d'une manière efficience dans la construction des applications sensibles au contexte à base de services comme nous allons présenter dans les chapitres 6 et 7.

Aussi, le SOA peut être implémenté par plusieurs plateformes et technologies existantes telles que, les services Web avec les spécifications de WS-*[7], les services Web de type REST (REpresentational State Transfer) [51], OSGI Alliance [1], ESB (Enterprise Service Bus) [36].

Beaucoup de travaux de développement des applications sensibles au contexte à base de services utilisent la technologie des services Web dans leurs réalisations [116, 73, 103] comme une plateforme basée SOA. Les services Web reposent principalement sur des technologies basées sur XML pour la structure et le contenu de messages échangés entre services (SOAP), pour la description fonctionnelle et non fonctionnelle des services (WSDL), pour la découverte et le courtage des services (UDDI, WS-Discovery) et pour leurs orchestrations (BPEL). L'ensemble de ces technologies est appelé WS-* [7].

En revanche, écrire des applications SOA avec de nombreux services Web, n'est pas toujours chose aisée. Notamment, la mise en œuvre des services Web (WS-*, REST, etc.) demande du temps et surtout du code technique en plus des classes métiers.

Pour notre cas, nous proposons d'utiliser un autre modèle ouvert de la plateforme SOA fondé sur l'ingénierie logicielle à base de composants (CBSE), en l'occurrence, l'architecture à base de composant-service (SCA) qui s'avère prometteuse pour le développement des applications sensibles au contexte à base de services, notamment pour la facilité qu'elle offerte quant à l'adaptation en temps d'exécution de ces applications (voir Sous-section 6.5.4).

2.4.2 L'ingénierie logicielle à base de composants (CBSE)

A l'instar de la programmation orientée objet (POO), l'ingénierie logicielle à base de composants (notée CBSE pour component-based software engineering) définit le composant comme étant l'élément de base d'une application. Celui-ci peut être vu comme une entité indépendante encapsulant un code métier correspondant aux opérations qu'il peut effectuer [63]. La programmation par composant permet de définir plus aisément une architecture pour l'application. Elle permet aussi d'organiser l'interopération entre les objets formant chaque composant. En effet, l'accès au code métier d'un composant est limité à un ensemble d'interfaces appelées généralement les interfaces fournies ou requises. Les interfaces fournies représentent les interfaces par lesquelles un autre composant peut accéder aux fonctionnalités offertes par le composant et les interfaces requises correspondent à celles permettant au composant d'accéder aux fonctionnalités des autres composants. Ces interfaces clientes sont généralement nécessaires au bon fonctionnement du composant.

Parmi plusieurs modèles de composants proposés, nous nous sommes intéressés plus particulièrement par le modèle Fractal [31] qui est la base de notre plateforme cible FraSCAti et qui sera utilisée ultérieurement dans ce mémoire (voir Chapitres 6 et 7).

1. `https://www.osgi.org/`

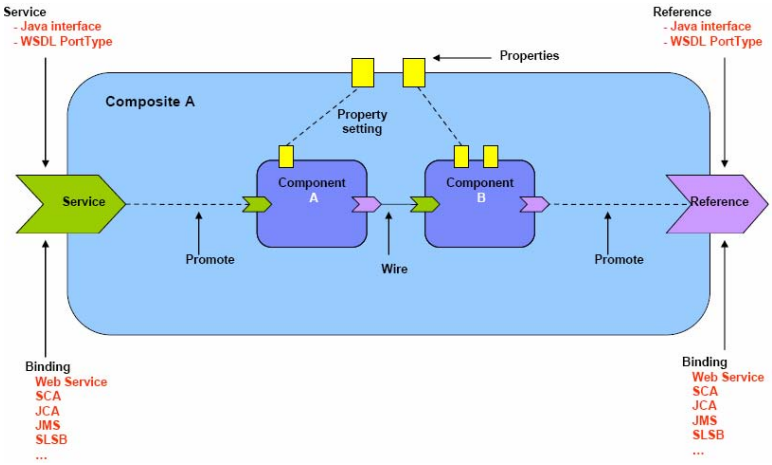


FIGURE 2.4 – SCA et l'assemblage des composants

2.4.3 L'architecture à base de composant-service (SCA)

L'architecture orientée services a besoin des plateformes logicielles plus appropriées et beaucoup plus flexibles pour la livraison, le support et la gestion d'applications distribuées conformes à ses principes. Le Service Component Architecture (SCA) [14] vise à accomplir ce besoin à l'aide des spécifications technologiques agnostiques soutenant la mise en place d'un service comme un composant logiciel.

SCA est un ensemble de spécifications pour la construction d'applications distribuées et des systèmes utilisant les principes de SOA [50] et CBSE [63]. Les spécifications SCA définissent la façon de créer et d'assembler les composants dans des applications complètes (voir Figure 2.4). Un composant implémente généralement une logique métier, qui est exposée sous forme d'un ou plusieurs interfaces « services ». Et également, un composant peut se servir des services fournis par d'autres composants, dans ce cas, il peut s'appuyer sur l'utilisation des interfaces « références ». Enfin, un composant peut aussi définir une ou plusieurs propriétés dont chacune contient une valeur qui peut être lue lorsque le composant est instancié.

Dans la deuxième partie de ce mémoire, nous allons voir comment exploiter les avantages de SCA afin de construire nos applications sensibles au contexte à base de services.

2.5 Conclusion

L'informatique pervasive a pour objectif de fournir les informations/services adéquats à l'utilisateur où il se trouve dans son environnement. Le nouveau type des applications sensibles au contexte permettent de satisfaire cet objectif en se basant sur les informations de contexte recueillies de l'environnement, l'utilisateur et l'application pour réaliser la sensibilité au contexte et déterminer le comportement de l'application que devrait prendre selon les différentes situations contextuelles.

Aussi, nous avons vu, dans ce chapitre, l'architecture orientée services (SOA) qui répond bien à la réalisation de ces applications sensibles au contexte avec la technologie des services Web. Actuellement, le SOA est renforcé en plus par la nouvelle architecture à base de composant-service (SCA) qui permet l'utilisation de plusieurs plateformes d'exécution y compris les services Web.

Dans le chapitre suivant, nous présenterons le développement dirigée par les modèles et qui sera utilisé, ultérieurement, pour proposer notre approche de développement des applications sensibles au contexte à base de services.

Chapitre 3

Le Développement Dirigé par les Modèles

3.1 Introduction

Le Développement Dirigé par les Modèles (MDD : Model-Driven Development) est une ingénierie de développement des systèmes logiciels dans lesquels les modèles prennent un rôle central, non seulement pour l'analyse de ces systèmes mais également pour leur construction. MDD a émergé à partir des initiatives de modélisation, le plus en évidence l'architecture dirigée par les modèles (MDA, Model-Driven Architecture) favorisée par le groupe (OMG, Object Management Group). Dans la portée de MDA, un couple des technologies a été développé qui sont devenues les pierres angulaires de MDD, comme la méta-modélisation et la transformation des modèles. MDD se fonde sur des langages pour définir des méta-modèles, comme le MOF (Meta-Object Facility) et Ecore (développé dans le cadre de Eclipse Modelling Framework- EMF), et des langages de spécifications de transformation comme QVT et ATL.

MDD a lieu, seulement, depuis quelques années et est sur le point de devenir une commodité dans le développement de logiciel dû à ses avantages (réduction des coûts de développement, amélioration de qualité de logiciel, réduction de coûts d'entretien et le soutien de l'évolution commandée de systèmes). Également, il a été appliqué dans beaucoup de domaines d'application, tels que le temps réel et les systèmes embarqué, et systèmes de télécommunication, et, plus récemment, au développement et à l'intégration des systèmes d'information d'entreprise, les systèmes sensibles au contexte.

Ce chapitre vise à présenter cette nouvelle ingénierie de développement ainsi que les fondements sur lesquelles elle se base.

3.2 L'ingénierie dirigée par les modèles

Avec le développement des réseaux et de l'avènement du Web, les systèmes d'information n'ont jamais été autant au centre de la stratégie des entreprises que durant la dernière décennie. Les fonctionnalités qu'ils offrent, leur facilité d'utilisation, leur fiabilité, leur performance et leur robustesse sont des qualités indéniables qui permettent aux entreprises d'innover et d'être compétitives. De nouvelles technologies et plateformes d'exécutions et de nouvelles versions de logiciels sont proposées continuellement afin de

faciliter la mise en œuvre de ces systèmes d'information tout en accroissant leurs qualités. La dernière en date de ces technologies et plateforme d'exécutions est certainement la plateforme orientée service et l'architecture logicielle associée désignée par l'architecture orientée service (notée SOA pour Services Oriented Architecture). Le vrai problème de ces nouvelles technologies et plateformes d'exécutions, est qu'elles sont toujours plus complexes et alourdissent considérablement les contraintes imposées aux équipes de développement. Cela met les entreprises devant un vrai dilemme, car elles hésitent entre adopter une nouvelle plateforme et subir le coût de la migration ou ne pas l'adopter et prendre le risque de voir les concurrents devenir plus compétitifs en ayant adoptée la nouvelle plateforme. Ce dilemme est un inconvénient majeur pour l'évolution des systèmes d'information des entreprises. La complexité des plateformes d'exécutions est la cause réelle de cet inconvénient, pourtant ces plateformes sont conçues pour améliorer la productivité et faciliter la maintenance des systèmes informatiques. Pour répondre à ce frein au développement des systèmes d'information, il était nécessaire de définir un autre type d'ingénierie du logiciel et approche permettant de faire face à la complexité. Cette ingénierie se devait d'être flexible et générique afin de pouvoir s'adapter à tout type de plateforme [118].

Aujourd'hui, l'ingénierie du logiciel s'oriente vers l'ingénierie dirigée par les modèles (IDM) et le principe du « tout est modèle » [19] au lieu de l'approche objet des années 80 et de son principe du « tout est objet », cette nouvelle approche peut être considérée à la fois en continuité et en rupture avec les précédents travaux. Tout d'abord en continuité car c'est la technologie objet qui a déclenché l'évolution vers les modèles [67]. En effet, une fois acquise la conception des systèmes informatiques sous la forme d'objets communicant entre eux, il s'est posé la question de les classifier en fonction de leurs différentes origines (objets métiers, techniques, etc.). L'IDM vise donc à fournir un grand nombre de modèles pour exprimer séparément chacune des préoccupations des utilisateurs, des concepteurs, des architectes, etc. C'est par ce principe de base fondamentalement différent que l'IDM peut être considérée en rupture par rapport aux travaux de l'approche objet[68].

En novembre 2000, l'Object Management Group (OMG) a proposé l'approche nommée Model Driven Architecture (MDA™) pour le développement et la maintenance des systèmes à prépondérance logicielle [91]. MDA applique la séparation des préoccupations entre la logique métier des systèmes informatiques et les plateformes utilisées et se fonde sur l'utilisation massive des modèles. Ainsi, la pérennité est l'objectif principal de MDA. Il s'agit de faire en sorte que la logique métier des applications ne soit plus mêlée aux considérations techniques de mise en production. Il devient dès lors possible de capitaliser les savoir-faire et d'être beaucoup plus réactif aux changements technologiques. Pour atteindre cet objectif, MDA vise à représenter sous forme de modèles toute l'information utile et nécessaire à la construction et à l'évolution des systèmes d'information. Les modèles sont au centre du cycle de vie des logiciels et des systèmes d'information.

Au-delà de la proposition spécifique MDA de l'OMG, l'Ingénierie Dirigée par les Modèles (IDM), ou Model Driven Engineering (MDE) en anglais, permet d'appréhender le développement de systèmes complexes en se concentrant sur une préoccupation plus abstraite que la programmation classique. Il s'agit d'une forme d'ingénierie générative dans laquelle tout ou partie d'une application est engendrée à partir de modèles. L'objectif de l'IDM est de définir une approche pouvant intégrer différents espaces technologiques. Ainsi, l'approche MDA devient une variante particulière de l'Ingénierie Dirigée par les

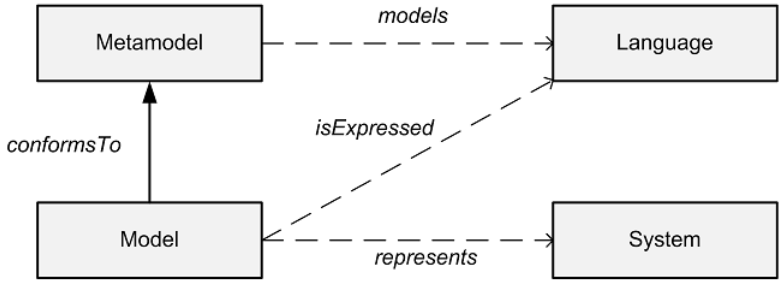


FIGURE 3.1 – Relations entre système, modèle, métamodèle et langage

Modèles [118].

3.2.1 Le Modèle

Alors que l'approche objet est fondée sur les relations essentielles d'instanciation et d'héritage, l'IDM est basée sur un autre jeu de concepts et de relations. Le concept central de l'IDM est la notion de modèle, pour laquelle il n'existe pas à ce jour de définition universelle. Néanmoins, à partir de différentes définitions proposées dans la littérature, on trouve les définitions suivantes :

Definition1 : « Un modèle est un ensemble de faits caractérisant un aspect d'un système dans un objectif donné. Un modèle représente donc un système selon un certain point de vue, à un niveau d'abstraction facilitant par exemple la conception et la validation de cet aspect particulier du système » [68].

Definition2 : Un modèle doit capturer les informations nécessaires et suffisantes pour permettre de répondre aux questions que l'on se pose sur un aspect du système qu'il représente, exactement de la même façon que le système lui-même aurait répondu [17, 110].

Aussi, dans [17, 110], un consensus se dégage sur la définition représentative suivante et qui a été proposée par [18] :

Definition3 : « **A model is a simplification of a system build with an intended goal in mind. The model should be able to answer questions in place of the actual system** » .

Cette dernière définition stipule qu'un modèle est une abstraction d'un système construit dans un but précis. De ce fait, le modèle contient un ensemble restreint d'informations sur un système et cet ensemble d'informations est choisi pour être pertinent vis a vis de l'utilisation qui sera faite du modèle. Nous disons dit alors que le modèle représente le système. Nous déduisons de cette définition la première relation fondamentale de l'IDM, entre le modèle et le système qu'il représente, nommée « **Represents** » dans [9]. Par analogie avec les langages de programmation, un programme exécutable représente le système alors que le code source de ce programme représente le modèle [118].

La Figure 3.2 illustre l'IDM par un exemple de cartographie. Dans cet exemple, une carte est un modèle (une représentation) de la réalité, avec une intention particulière (carte administrative, routière, des reliefs, etc.).

Notons que même si cette relation a fait l'objet de nombreuses réflexions, il reste toutefois difficile de répondre à la question « qu'est ce qu'un bon modèle ? » et donc de

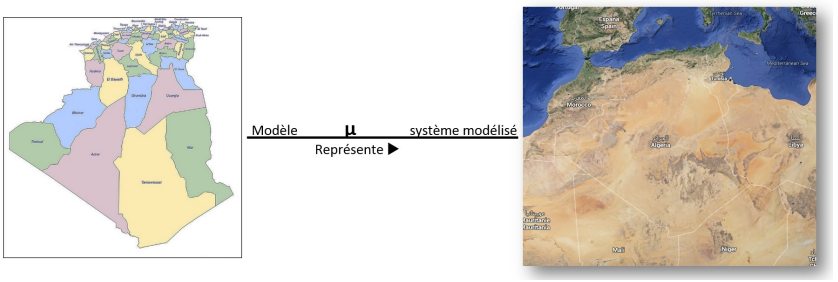


FIGURE 3.2 – Relation de représentation entre le système et le modèle

formaliser précisément la relation μ. Toute la difficulté de l'activité de modélisation est de trouver le bon niveau d'abstraction pour représenter un système selon un ensemble de modèles facilitant sa conception et sa validation. Un modèle doit donc par définition être une abstraction pertinente du système qu'il modélise pour un point de vue particulier, c'est-à-dire qu'il doit être suffisant et nécessaire pour répondre aux questions sous-jacentes à ce point de vue particulier, en lieu et place du système qu'il représente et exactement de la même façon que le système aurait répondu lui-même [118].

La notion de modèle dans l'IDM fait explicitement référence à la notion de langage de modélisation bien défini (Figure 3.1). En effet, pour qu'un modèle soit productif, il doit pouvoir être manipulé par une machine. Le langage dans lequel ce modèle est exprimé doit donc être clairement défini. De manière naturelle, la définition d'un langage de modélisation a pris la forme d'un modèle, appelé métamodèle [118].

3.2.2 Le Metamodèle

La notion de métamodèle est au cœur de l'IDM. La définition assez consensuelle est donnée dans [91] :

Définition : "A meta-model is a model that defines the language for expressing a model "

Un métamodèle est un modèle qui définit le langage d'expression d'un modèle. Il représente les concepts du langage de modélisation utilisé et la sémantique qui leur est associée. En d'autres termes, le métamodèle décrit le lien existant entre un modèle et le système qu'il représente. Cette notion de métamodèle conduit à l'identification d'une seconde relation, liant le modèle et le langage utilisé pour le construire, désignée par « **ConformsTo** » (voir Figure 3.1) et stipule en fait qu'un modèle est conforme à son métamodèle. En poursuivant l'analogie avec le monde de la programmation, un programme est écrit ou est exprimé dans un langage de programmation, un modèle est écrit ou est exprimé dans un langage de modélisation. Chacun des langages (programmation ou modélisation) définit les concepts et les règles à suivre (syntaxe) pour écrire un programme source conforme au langage de programmation, ou spécifier un modèle conforme au langage de modélisation (métamodèle).

Cette notion de conformité est essentielle à l'ingénierie dirigée par les modèles mais n'est pas nouvelle : un texte est conforme à une orthographe et une grammaire et un document XML est conforme à sa DTD. Un autre point lié à la notion de métamodèle est la distinction entre langage et métamodèle qui sont souvent confondus dans la littérature. Bien que ces deux concepts soient proches (voir la définition ci-dessus), ils sont néanmoins

différents [17].

Un langage est un système abstrait alors qu'un métamodèle est une définition explicite de ce langage. En pratique, un métamodèle permet de capitaliser un domaine de connaissances [68]. En reprenant la Figure 3.1 et le monde de la programmation, plusieurs programmes écrits en langage C sont conformes à la même grammaire du langage C. La grammaire est alors considérée comme un modèle représentant cet ensemble de programmes et auquel chacun doit se conformer. Sur l'exemple de la Figure 3.2, la carte doit, pour être utilisable, être conforme à cette légende qui représente son metamodèle. Ces deux relations permettent ainsi de bien distinguer le langage, qui joue le rôle de système, du métamodèle qui joue le rôle de modèle de ce langage [118].

Après l'acceptation du concept clé de métamodèle comme langage de description de modèle, de nombreux métamodèles ont émergés afin d'apporter chacun leurs spécificités dans un domaine particulier (développement logiciel, entrepôt de données, procédé de développement, etc.) [40].

Les concepts de système, modèle et métamodèle ainsi que les relations qui les lient, représentent les principes de base sur lesquels s'appuie l'IDM. A partir de ces principes, d'autres non moins importants peuvent être déduits afin d'assurer une complétude et une cohérence du cadre conceptuel de l'IDM. Ainsi, de la même manière qu'il est nécessaire d'avoir un métamodèle pour interpréter un modèle. Pour pouvoir interpréter un métamodèle il faut disposer d'une description du langage dans lequel il est écrit : un métamodèle pour les métamodèles. Le terme métamétamodèle est utilisé tout naturellement pour désigner ce métamodèle particulier. Un métamétamodèle définit les notions de base permettant l'expression des métamodèles et des modèles. Il permet d'exprimer les règles de conformités qui lient les entités du niveau modèle à celle du niveau métamodèle. Afin de limiter le nombre de niveaux d'abstraction (et ainsi éviter d'avoir à définir un métamétamétamodèle). Le métamétamodèle MOF [94] est conçu avec la propriété de méta-circularité, c-â-d. ; la capacité de se décrire lui-même. Dans le cas des grammaires, on spécifie ainsi EBNF par une grammaire et dans le cas d'XML, c'est une DTD XML pour les DTD qui joue le rôle de métamétamodèle.

Ainsi, l'IDM est fondée sur une architecture à quatre niveaux (Figure 3.3), initialement proposée dans le cadre du MDA [91], et qui fait maintenant l'objet d'un large consensus en IDM [110, 17]. Cette approche consistant à considérer une hiérarchie de métamodèles n'est pas propre à l'OMG, ni même à l'IDM, puisqu'elle est utilisée depuis longtemps dans différents domaines de l'informatique. Le domaine des langages de programmation par le biais des grammaires [4], est peut être, celui qui est le plus cité dans la littérature pour illustrer cette architecture à 4 niveaux. C'est sur les principes présentés dans cette section que se base l'organisation de la modélisation en IDM généralement décrite sous forme pyramidale (voir Figure 3.3) . Le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les métamodèles permettant la définition de ces modèles (UML, par exemple) constituent le niveau M2. Enfin, le métamétamodèle, unique et méta-circulaire, est représenté au sommet de la pyramide (niveau M3) [40].

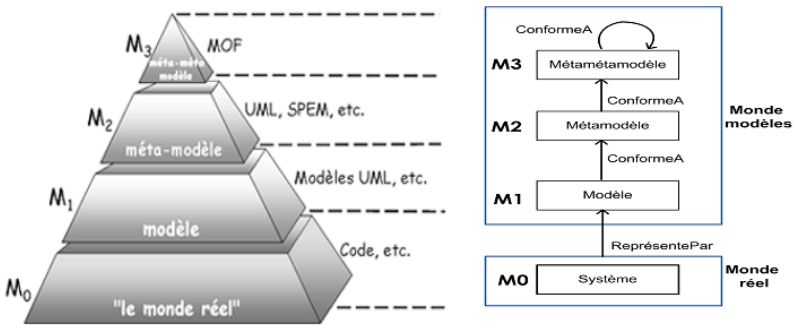


FIGURE 3.3 – Pyramide de modélisation à quatre niveaux

3.3 L'approche MDA

L'OMG a donné une définition détaillée de l'architecture MDA en 2003 [91] avec comme objectif de promulguer de bonnes pratiques de modélisation afin de déterminer comment, quand, quoi et pourquoi modéliser et d'exploiter pleinement les avantages des modèles.

MDA définit plusieurs formalismes standards de modélisation, notamment UML [96], MOF [94] et XMI [97], afin de promouvoir les qualités intrinsèques des modèles, telles que pérennité, productivité et prise en compte des plateformes d'exécution. Le principe clé de MDA consiste en l'utilisation de modèles aux différentes phases de développement d'une application en s'appuyant sur le standard UML. Plus précisément, MDA préconise l'élaboration des modèles :

- d'exigences (Computation Independant Model - CIM), dans lesquels aucune considération informatique n'apparait,
- d'analyse et de conception (Platform Independant Model - PIM)
- de code (Platform Specific Model - PSM).

Dans la section 3.3.1 nous discuterons ces trois types de modèles et leur place dans le processus de développement MDA.

L'objectif majeur de MDA [21], est l'élaboration de modèles pérennes (PIM), indépendants des détails techniques des plateformes d'exécution (J2EE, .Net, PHP, Oracle, ect.), afin de permettre la génération automatique de la totalité des modèles de code (PSM) et d'obtenir un gain significatif de productivité. Cette génération automatique est possible grâce à l'exécution des transformations de modèles. On comprend pourquoi le sucés de MDA et de l'IDM en général, repose en grande partie sur la résolution de problème de transformation de modèle. Cette problématique a donnée lieu ces dernières années à de nombreux travaux académiques [69, 66].

3.3.1 Processus de développement dans MDA

Nous avons vu aussi que MDA préconisait l'élaboration de différents modèles, modèle d'exigences CIM, modèle d'analyse et de conception abstraite PIM et modèle de code et de conception concrète PSM. Ces modèles de base seront présentés plus en détail dans les sous sections suivantes et qui sont indispensables dans tout processus de développement selon l'approche MDA (voir Figure 3.4). L'objectif de MDA est de décrire comment construire des systèmes logiciels de manière fiable et reproductible en utilisant les différents types

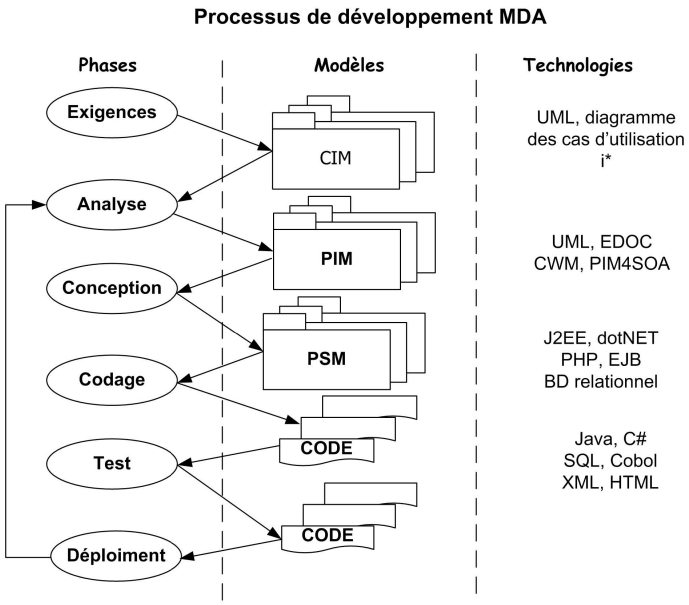


FIGURE 3.4 – Processus de développement selon l'approche MDA

de modèles.

3.3.1.1 Le modèle d'exigences CIM

Un modèle d'exigence doit représenter l'application dans son environnement afin de définir quels sont les services offerts par l'application et quelles sont les autres entités avec lesquelles il interagit. Le modèle d'exigence traduit les besoins du client et représente ainsi le point de départ dans le cycle de développement. Les modèles d'exigences peuvent être considérés comme des éléments contractuels, destinés à servir de référence lorsqu'on voudra s'assurer qu'une application est conforme aux demandes du client. Il est important de noter qu'un modèle d'exigences ne contient pas d'information sur la réalisation de l'application ni sur les traitements. C'est pourquoi, dans le vocabulaire MDA, les modèles d'exigences sont appelés les CIM (Computation Independant Model), "modèle indépendant de la programmation".

Avec UML, un modèle d'exigence peut se résumer à un diagramme de cas d'utilisation. Ce dernier contient en effet les fonctionnalités fournies par l'application (cas d'utilisation) ainsi que les différentes entités qui interagissent avec elles (acteurs) sans apporter d'information sur le fonctionnement de l'application.

En conclusion, le rôle des modèles d'exigences dans une approche MDA est d'être les premiers modèles pérennes. Une fois les exigences modélisées, elles sont censées fournir une base contractuelle avec le client. Grâce aux liens de traçabilité avec les autres modèles qui suivront dans le cycle de développement, un lien peut être créé depuis les exigences jusqu'au code final [118].

3.3.1.2 Le modèle PIM

Une fois le modèle d'exigences réalisé et validé par le client, le travail d'analyse et de conception peut démarrer. Dans l'approche MDA, cette phase utilise les modèles appelés PIM (Platform Independant Model), désignés aussi par modèle d'analyse et de conception

abstraite. En effet, ces modèles spécifiant généralement la logique métier de l'entreprise, sont indépendants de toute plateforme technique et ne doivent pas contenir d'informations sur les technologies qui seront utilisées pour déployer l'application. En intégrant les détails d'implémentations que très tard dans le cycle de développement, il est possible de maximiser la séparation des préoccupations entre la logique métier des applications et les techniques d'implémentation.

Aujourd'hui, UML s'est imposé comme la référence pour réaliser les modèles d'analyse et de conception. Il est important de noter que MDA ne fait que préconiser l'utilisation d'UML et qu'il n'exclut pas que d'autres langages puissent être utilisés. En effet, l'IDM, au contraire, favorise la définition de langages de modélisation dédiés à un domaine particulier [70] (Domain Specific Languages – DSL) offrant ainsi aux utilisateurs des concepts propres à leur métier et dont ils ont la maitrise. Ces langages sont généralement de petite taille et doivent être facilement manipulables, transformables, combinables, etc.

Dans ce contexte, l'OMG a proposé des langages standards dédiés à des domaines d'applications, tel que EDOC (Enterprise Distributed Object Computing) pour les systèmes distribués à base de composants, CWM pour les bases données, etc. Quels que soient les langages utilisés, le rôle des modèles d'analyse et de conception est de pérenniser la logique métier de l'entreprise et de faire le lien entre les modèles d'exigences et le code final de l'application. Les modèles PIM doivent être productifs, c'est à dire qu'ils doivent être suffisamment précis et contenir suffisamment d'information pour qu'une génération automatique de code soit envisageable ultérieurement [118].

3.3.1.3 Le modèle PSM

Cette phase, la plus délicate dans MDA, concerne la génération de code. Le modèle PSM (Platform Specific Model), appelé aussi le modèle de code ou de conception concrète, est alors mis en jeux. MDA considère que le code final d'application peut être facilement obtenu à partir du PSM. En effet, le code d'une application se résume à une suite de lignes textuelles, comme un fichier SQL, alors qu'un modèle de code est plutôt une représentation structurée incluant, par exemple, les concepts de table, clé primaire, clé étrangère, etc.

L'écriture de code à partir du modèle de code est donc une opération assez triviale. La principale différence entre un modèle de code PSM est un modèle d'analyse et de conception PIM réside dans le fait que le modèle de code est lié à une plateforme d'exécution. Ainsi, une caractéristique importante des modèles de code est qu'ils intègrent les concepts des plateformes d'exécution. Pour élaborer des modèles de code, MDA propose, entre autres, l'utilisation des profils UML. Un profil UML est une adaptation du langage UML à un domaine particulier [118].

3.3.1.4 Transformations du modèle CIM au modèle PSM

La mise en production de MDA passe par la mise en relation des trois principaux modèles CIM, PIM et PSM. Cette mise en relation se fait par les techniques de transformations qui sont omniprésentes et occupent une place centrale dans MDA. Ces techniques permettent de transformer un modèle CIM en un modèle PIM et d'obtenir un modèle PSM à partir d'un modèle PIM. La section 3.3.3 présentera les techniques de transformations dans l'espace MDA. La Figure 3.4 illustre le processus de développement dans MDA et montre les principales transformations de modèles suivantes [118] :

- **Transformation de modèles CIM vers PIM** : Permettent d'élaborer des PIM partiels à partir des informations contenues dans les CIM. L'objectif est de s'assurer que les besoins exprimés dans les CIM sont bien représentés dans les PIM. Ces transformations sont essentielles à la pérennité des modèles. Ce sont elles qui garantissent les liens de traçabilité entre les modèles et les besoins exprimés par le client.
- **Transformations de modèles PIM vers PIM** : Permettent de raffiner les PIM afin d'améliorer la précision des informations qu'ils contiennent. En UML, de telles transformations, peuvent être, par exemple, l'enrichissement de classes UML à partir des diagrammes de séquences et des diagrammes d'états transitions. Ces transformations sont omniprésentes dans les outils d'élaboration de modèles PIM. Elles permettent d'accélérer la production des PIM et donc de raccourcir le cycle de développement.
- **Transformations de modèles PIM vers PSM** : Permettent d'élaborer une bonne partie des modèles PSM à partir des modèles PIM. Ces transformations sont les plus importantes de MDA car elles garantissent la pérennité des modèles aussi bien que leur productivité et leur lien avec les plateformes d'exécution.
- **Transformations de modèles PSM vers code** : Permettent de générer la totalité du code. Ces transformations consistent en une traduction entre un formalisme structuré tel que un diagramme UML et un formalisme textuel représentant le code final.

3.3.2 Modélisation et métamodélisation dans MDA

Pour mettre en œuvre son objectif principal qui est la pérennité par les modèles, MDA définit une architecture à 4 niveaux (Figure 3.3), qui établit les concepts de base de l'ingénierie par les modèles et positionne chacun des standards qui font MDA, tel que MOF, UML, OCL, XMI. Cette section présente ces langages standards qui forment le socle de base pour la modélisation et la métamodélisation dans MDA [118].

Le succès industriel du langage UML [96] est unanimement reconnu, puisque c'est désormais le langage de spécification d'applications orientées objet le plus utilisé au monde. Le consensus sur ce langage fut décisif dans cette transition vers l'IDM et les techniques de production basées sur les modèles. Après l'acceptation du concept clé de métamodèle comme langage de description de modèle, de nombreux métamodèles ont émergés afin d'apporter, chacun, leurs spécificités dans un domaine particulier (développement logiciel, entrepôt de données, procédé de développement, etc.).

Devant la difficulté de voir émerger indépendamment et de manière incompatible une grande variété de métamodèles, il y avait un besoin urgent de donner un cadre général pour leur description. La réponse logique fut donc d'offrir un langage de définition de métamodèles qui prit lui-même la forme d'un modèle : ce fut le métamétamodèle MOF (Meta-Object Facility) [94]. Une des lacunes du langage UML est son incapacité à exprimer une sémantique opérationnelle. Il n'est pas évident pour UML d'exprimer par exemple précisément ce que fait une opération. Dans le métamodèle UML, une opération n'est définie que par son nom, ses paramètres et les exceptions qu'elle émet. Le corps de l'opération ne peut donc être défini.

Le langage OCL (Object Constraint Language) [92]a été précisément défini par l'OMG

pour combler cette lacune et permettre la modélisation du corps des opérations UML. OCL permet d'exprimer des contraintes sur tous les éléments des modèles UML.

Le standard XMI [97] permet de représenter les modèles sous forme de documents XML et favorise ainsi leur échange et donc leur pérennité. Le principe de fonctionnement de XMI consiste à générer automatiquement une spécification de structuration de balises XML (DTD ou XML schéma) à partir d'un métamodèle. Il est ainsi possible de bénéficier du mécanisme de validation des documents XML.

Pour permettre l'adaptation d'UML à d'autres domaines et pour préciser la signification de cette adaptation, l'OMG a standardisée le concept de profil UML. Un profil est un ensemble de techniques et de mécanismes permettant d'adapter UML à un domaine particulier. Cette adaptation peut se faire sur n'importe quel modèle UML et elle ne modifie en rien le métamodèle UML.

Enfin, l'OMG ne pouvait ignorer le franc succès de l'ingénierie des ontologies dans les domaines tel que le Web et le Web sémantique . Ainsi, l'OMG a proposé le métamodèle ODM (Ontology Definition Metamodel) [93] qui a pour objectif de permettre la modélisation d'une ontologie conformément au formalisme MOF. La modélisation d'ontologies en MOF permet un héritage de tout l'outillage de ce dernier. En effet, une fois qu'un langage de modélisation du Web Sémantique comme OWL est modélisé en MOF, ses utilisateurs peuvent utiliser les capacités de MOF pour la création, la gestion de modèles, la génération de code et l'interopérabilité avec des modèles MOF.

3.3.3 Transformation des modèles dans MDA

Après la modélisation et la métamodélisation, la transformation de modèles est la troisième problématique clé de MDA et de l'IDM en général. Elle permet de rendre les modèles productifs et d'obtenir une traçabilité entre des modèles très abstraits, proches des besoins exprimés par les utilisateurs, et des modèles très concrets, proches des plateformes d'exécutions. Nous avons vu dans la section 3.3.1 les différents types de transformations. Quel que soit son type (CIM vers PIM, PIM vers PSM, etc.), une transformation de modèles s'apparente toujours à une fonction qui prend en entrée un ensemble de modèles et qui fournit en sortie un ensemble de modèles.

La définition consensuelle de la transformation de modèle est donnée dans [78] : *« Une transformation est la génération automatique d'un modèle cible à partir d'un modèle source, selon une définition de transformation. Une définition de transformation est un ensemble de règles de transformation qui, ensemble, décrivent comment un modèle dans le langage source peut être transformé en un modèle dans le langage cible. Une règle de transformation est une description de la manière dont une ou plusieurs constructions dans le langage source peuvent être transformées en une ou plusieurs constructions dans le langage cible ».*

Pour qu'une transformation s'exécute au niveau des modèles, elle se spécifie au niveau métamodèles. Les métamodèles permettent dans le processus de transformation MDA de définir les structurations possibles des modèles source et cible et de servir de base pour la définition des règles de transformation (voir Figure 3.5).

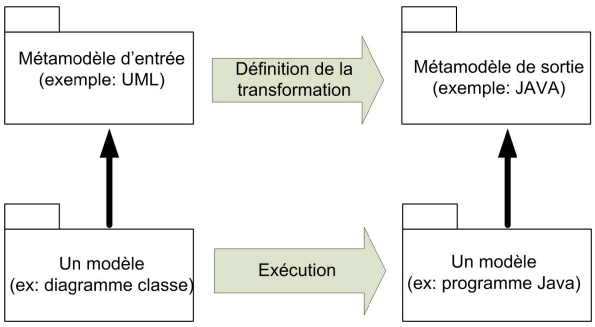


FIGURE 3.5 – Métamodèles et transformations de modèles

3.3.3.1 Les différentes approches de transformations de modèles

Trois principales approches ont été développées pour la mise en œuvre des transformations de modèles. La différence entre ces trois approches est liée à la manière dont les règles de transformations sont spécifiées [21, 118].

- **Approche par programmation** : Comme son son l'indique, un langage de programmation est mis en œuvre pour programmer une transformation de modèle de la même manière que l'on développe n'importe quelle application. Le langage de programmation est de type orienté objet afin de préserver l'uniformité conceptuelle avec les modèles UML et les outils MDA. Les transformations selon cette approche sont donc des applications qui ont la spécificité de manipuler des modèles. Ces applications utilisent des interfaces de gestion de modèles offrant les opérations nécessaires pour la manipulation des modèles. Les standards et framework MOF [94] et Ecore (développé dans le cadre de Eclipse Modelling Framework - EMF) [119] disposent de ces interfaces. Cette approche est très utilisée car elle bénéficie de la richesse des langages de programmation et des environnements de développement fortement outillés. Dans ce travail, nous avons basé sur ce type d'approche pour réaliser les différentes transformations proposées.
- **Approche par template** : L'approche par template consiste à définir des canevas pour les modèles cibles souhaités. Ces canevas sont des modèles cibles paramétrés ou modèles templates. L'exécution d'une transformation consiste à prendre un modèle template et à remplacer ses paramètres par les valeurs d'un modèle source. Cette approche par template est implémentée par exemple dans Softeam MDA Modeler [1].
- **Approche par modélisation** : Cette approche consiste, quant-à-elle, à appliquer les concepts de l'ingénierie des modèles aux transformations des modèles elles-mêmes. L'objectif est de modéliser les transformations de modèles et de rendre les modèles de transformation pérennes et productifs, en exprimant leur indépendance vis-à-vis des plateformes d'exécution. Le standard MOF 2.0 QVT [95] de l'OMG a été élaboré dans ce cadre et a pour but de définir un métamodèle permettant l'élaboration des modèles de transformation de modèles ainsi que le langage de transformation de modèles ATL [69]. L'approche par modélisation, même si elle est la plus complexe à mettre en œuvre, reste la plus prometteuse, car elle offre une solution favorisant la pérennité des transformations et facilite donc leur réutilisation.

1. `http://www.objecteering.fr/products_mda_modeler.php`

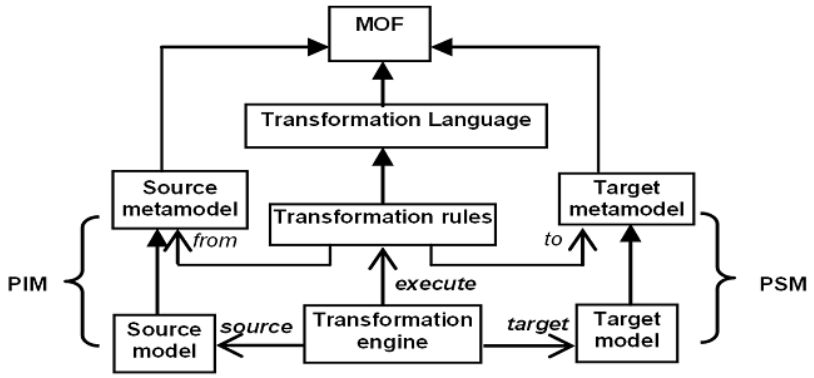


FIGURE 3.6 – Architecture du processus de transformation de modèles dans MDA

3.4 Le développement dirigé par les modèles (MDD)

3.4.1 Présentation

Cette nouvelle ingénierie pilotée par les modèles avec l'approche MDA a poussé l'émergence d'un nouveau type de développement des logiciels appelé le développement dirigé par les modèle (ou MDD : Model-Driven Development en anglais) [28]. Le MDD est une démarche de développement de logiciels qui propose d'employer des modèles compréhensibles à la machine à divers niveaux d'abstraction en tant que ses objets principaux. L'idée principale est de transformer automatiquement des modèles hautement abstraits en des modèles concrets à partir duquel une implémentation peut être générée de manière directe. L'approche MDD est soutenu par le MDA (Model Driven Architecture) initiative de l'OMG [91], qui a introduit, comme nous avons vu dans la section 3.3.1, la notion de PIM (Platform Independent Model) et PSM (Platform Specific Model). Un PIM est un modèle d'un système qui se concentre uniquement sur la logique métier de l'application et ne contient pas les détails techniques. D'autre part, Un PSM est une représentation du même système contenant tous les détails techniques nécessaires pour le réaliser sur une plateforme concrète de technologie. La correspondance entre les PIM et PSM est réalisée en utilisant des (semi-) automatiques transformations [10].

Grâce à la séparation claire de la logique métier exprimé dans le PIM et les détails techniques, contenues dans les règles de transformation qui génèrent le PSM, l'approche MDD facilite le développement d'applications pervasives pour de nombreux dispositifs. En effet, un seul PIM peut accommoder pour multiples PSM requis par des différents dispositifs et plateformes de technologies différentes. Ainsi, différents dispositifs pervasifs embarqués, avec des capacités et des exigences différentes, peuvent exécuter la même application à condition que le PIM est décrit et les règles de transformations sont données pour chaque appareil et chaque la plateforme.

Par conséquent, une autre issue principale dans MDD, est le processus de transformation des modèles. Aujourd'hui, il est bien reconnu que le processus de transformation des modèles est l'une des opérations les plus importantes de MDA.

Dans le contexte des quatre niveaux basiques de l'architecture de métamodélisation du MDA, différents scénarios de transformation de modèle à modèle ont été identifiés. La Figure 3.6 présente le scénario le plus commun de ces transformations, qui est compatible avec le standard MOF2.0/QVT [95]. Chaque élément présenté dans cette figure joue un

rôle important dans MDA. Les règles de transformation spécifient la façon de générer un modèle cible (c.-à-d. PSM) à partir d'un modèle source (c.-à-d. PIM).

Pour transformer un modèle donné dans un autre modèle nous devons exécuter des règles de transformation qui sont élaborées sur les correspondances trouvées entre les métamodèles source et cible. Les règles de transformation sont exprimées dans langages de transformation, tels que le standard QVT. Le moteur de transformation prend le modèle source en entrée, exécute les règles de transformation, et produit le modèle cible en sortie.

3.4.2 MDD et les applications sensibles au contexte

Dans l'étude de l'ingénierie des applications sensibles au contexte à base de services de [72], les auteurs définissent six classes d'approches pour gérer la sensibilité au contexte des applications, à savoir :

1. Les solutions basées sur les intergiciels et les plateformes de services dédiées,
2. L'utilisation d'ontologies,
3. Le raisonnement à base de règles,
4. Le code source (les langages de programmation et ses extensions),
5. Les approchés dirigées par les modèles,
6. Les messages d'interception.

En général, chacune de ces catégories d'approches a ses avantages et ses inconvénients. Par exemple, l'approche au niveau du code source peut donner plus de liberté aux développeurs de faire toutes sortes d'adaptation ou de sensibilité au contexte, mais cette approche n'a pas pu séparer à part les les préoccupations de l'adaptation (ou de sensibilité) au contexte, et souffre d'un coût de maintenance important .

Cependant, les approches pilotées par les modèles s'avèrent prometteuses par rapport aux autres approches proposées [72, 23]. Les approches de cette classe présentent beaucoup d'avantages (réduction des coûts de développement, amélioration de qualité de logiciel, réduction de coûts d'entretien et le soutien de l'évolution commandée de systèmes) pour le développement des applications/services sensibles au contexte, notamment son principe de séparation des différentes préoccupations dans des modèles.

Ce principe est le point clé le plus important pour traiter la sensibilité au contexte et assurer son indépendance avec la logique métier dans une application susceptible d'être sensible au contexte. Dans ce cadre de développement des applications sensibles au contexte à base de services, nous allons suivre une approche dirigée par les modèles sur une plateforme d'exécution cible orientée services pour bénéficier de touts les points forts du MDD et de l'architecture SOA (voir Sous-section 2.4.1).

3.4.3 Une autre plateforme PSM basée sur SOA

Dans le cadre de MDD, et pour le développement applications sensibles au contexte sur une plateforme basée SOA comme un PSM (Platform Specific Model), nous proposons d'utiliser l'architecture à base composant-service (voir Sous-section 2.4.3) et sa plateforme d'implémentation FraSCAti.

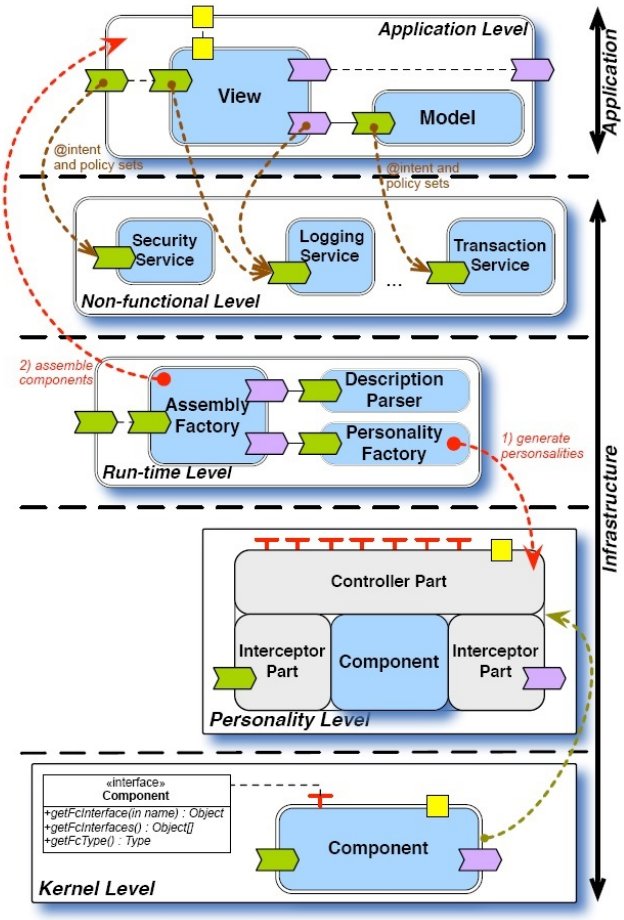


FIGURE 3.7 – Architecture de plateforme FraSCAti

3.4.3.1 FraSCAti

FraSCAti [2] est une plateforme open-source du consortium OW2. Elle fournit plusieurs fonctionnalités avancées, telles que le support de composants SCA réflexifs (permettant le changement de configuration à chaud des assemblages), des protocoles d'accès originaux comme UPNP, JNA ou encore des outils qui nous aiderons à observer l'état et administrer les assemblages de composants pendant leur exécution.

FraSCAti offre un regard réflexif des systèmes intergiciels (middleware) où la plateforme d'exécution, les services non fonctionnels et les applications métiers sont conçues et implémentés avec le même paradigme du modèle de composants SCA [111]. La Figure 3.7 montre une vue architecturale de la plateforme FraSCAti [112].

Cette architecture en couche contient 5 niveaux, la partie la plus haute, étiquetée «Application Level» correspond aux applications SCA conçues par les développeurs et destinées aux utilisateurs finaux. Les autres quatre couches sous-jacentes correspondent à l'infrastructure SCA, qui supportent le déploiement et l'hébergement et la reconfiguration en temps d'exécution (@run-time) de ces applications utilisateurs. Ces quatre couches sont détaillées dans [112].

2. `http://frascati.ow2.org`

Pourquoi la plateforme FraSCAti ? Dans le cadre de cette thèse, nous utiliserons la plateforme FraSCAti comme une implémentation de l'architecture SCA. Sa particularité est qu'elle facilite le processus d'adaptation dynamique des composants SCA qui est nécessaire pour notre domaine des applications sensibles au contexte (voir Chapitre 2). Ce paragraphe expose les motivations derrière notre choix de travailler avec cette plateforme.

1. FraSCAti est un modèle général, c'est-à-dire qu'il n'est pas spécifique à un domaine d'application [112].
2. Le modèle de base de FraSCAti, s'il n'est pas très innovant (il étend le modèle Fractal), contient les différents concepts de base communs par la plupart des autres modèles [34].
3. Le modèle FraSCAti est une implémentation d'une architecture orientée service (SCA).
4. Sur le plan technique, le modèle FraSCAti est très souple, permettant de nombreuses reconfigurations dynamiques et surtout est conçu à la base pour être facilement étendu et permettre l'ajout de nouvelles fonctionnalités.
5. Une application FraSCAti est vue comme un ensemble de composants et de services. Ces composants communiquent entre eux grâce à des références.

FraSCAti est donc un modèle très simple, complet, dynamique et extensible, qui nous offre toutes les fonctionnalités de base dont nous avons besoin pour bien mener notre recherche de développement des applications sensibles au contexte à base de services. Dans les chapitres 6 et 7 nous allons voir comment utiliser FraSCAti dans notre travail.

3.5 Conclusion

L'Ingénierie Dirigée par les Modèles (IDM) a permis d'améliorer significativement le développement de systèmes complexes, et cela par la concentration sur une préoccupation plus abstraite que la programmation classique. Il s'agit d'une forme d'ingénierie générative visant à automatiser la génération de tout ou partie d'une application à partir de modèles.

Dans ce chapitre, nous avons donné une présentation générale sur l'IDM qui est soutenue par l'approche MDA. Ce dernier offre un ensemble de standards de conception (MOF, XMI, UML, ...) et un processus de développement dirigée par les modèles (MDD). Le développement dirigée par les modèles selon MDA est basé sur la (méta) modélisation et la transformation des modèles pour générer les différents modèles d'une applications allant des exigences du client jusqu'à le code d'exécution (CIM, PIM et PSM), ce qui permet de garder une traçabilité cohérente servira à la validation et la maintenance de toute application développée. Aussi, l'engouement pour l'utilisation des technologies basées SOA, pour réaliser les applications sensibles au contexte, est maintenant renforcée par la plateforme prometteuse FraSCAti.

Dans le chapitre suivant, nous passerons en revue les travaux MDD proposés dans la littérature pour le développement des applications sensibles au contexte à base de services.

Chapitre 4

Approches MDD pour le développement des applications sensibles au contexte : État de l'art

4.1 Introduction

Les applications sensibles au contexte, qui s'adaptent en fonction des différents contextes, peuvent être réalisés à l'aide de l'Architecture Orientée Service (SOA). En effet, le faible couplage et l'interopérabilité inhérents à SOA contribuent de manière considérable à l'adaptation au contexte de ces applications et cela, par l'adoption de la technologie des services dans leurs développements [56].

Basé sur le paradigme SOA, divers travaux de recherche sur le développement des applications sensibles au contexte ont été réalisées en proposant différentes approches [72]. Cependant, la plupart des approches proposées ne présentent pas une méthodologie générique pour formaliser l'activité du développement de ce type d'applications et la rend très lourde et très longue.

Récemment, certains projets de recherche ont préconisé le Développement Dirigé par les Modèles (MDD) comme une approche de développement des applications sensibles au contexte à base de services. MDD permet la séparation entre l'information de contexte et la logique métier de l'application par l'utilisation des modèles d'abstraction et des techniques de transformation entre ces modèles pour traiter la sensibilité au contexte.

Dans ce chapitre, nous voulons passer en revue ces travaux de recherche basées sur les paradigmes SOA et MDD présentés dans la littérature et leurs principales caractéristiques.

4.2 Classification des approches

L'une des premières approches utilisant le MDD pour la modélisation de l'interaction entre le contexte et l'application basée sur les services se trouve dans ContextUML [113]. Ce travail a influencé plusieurs travaux dont quelques versions étendues de ContextUML ont été proposées. Ainsi, notre revue va discuter les approches suivantes :

1. L'approche ContextUML [113] ;
2. Les approches basées sur ContextUML [103, 29] ;

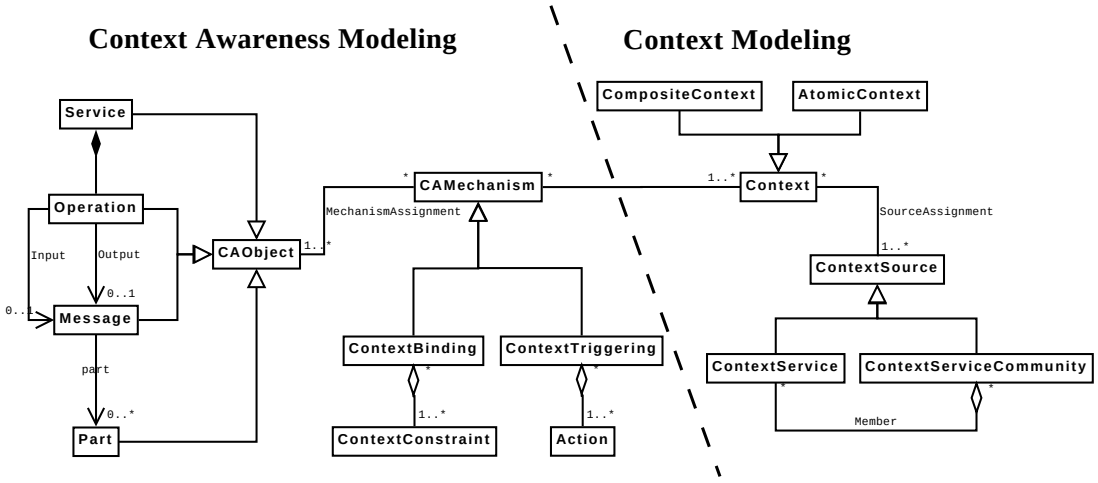


FIGURE 4.1 – Le métamodele ContextUML

3. Les approches basées sur UML et MOF [44, 73] ;
4. Les approches ad-hoc basées sur un Langage Spécifique de Domaine (DSL) [6, 127].

4.2.1 Approche de ContextUML

En utilisant le langage UML, le travail de Sheng et Benatallah [113] présente le métamodèle **ContextUML** pour la conception des services Web sensibles au contexte avec le développement dirigé par les modèles.

ContextUML est l'un des premiers modèles dédiés au domaine de la sensibilité au contexte. Il définit un métamodèle pour la modélisation de services Web sensibles au contexte. Les éléments propres aux services Web, tels que Service, Operation et Message sont représentés dans ce métamodèle ainsi que les mécanismes d'adaptation comme « Binding » ou « Triggering ». ContextUML met l'accent sur les mécanismes d'adaptation plutôt que sur la modélisation de contexte, et ne traite pas la qualité de contexte.

La philosophie principale de ContextUML est que les éléments de service sont en fait dépendants du contexte et peuvent être associés aux attributs de contexte décrits par un modèle de contexte approprié. Dans ce modèle, les attributs de contexte sont remplis par l'invocation des correspondants services fournissant des informations contextuelles [103].

ContextUML comprend à la fois un métamodèle et un langage de notation. Le métamodèle définit la syntaxe abstraite du langage, tandis que la notation définit la syntaxe concrète utilisée pour représenter ce langage [113]. La Figure 4.1 montre le métamodèle de ContextUML composé de deux aspects distincts : la modélisation du contexte et de la modélisation de la sensibilité au contexte (séparées par une ligne en pointillé sur la figure).

4.2.1.1 Modélisation du Contexte

La classe `Context` utilise deux sous-classes pour représenter les contextes atomiques et composites. Les contextes atomiques sont des contextes de bas niveau qui ne reposent pas sur d'autres contextes et peuvent être fournis directement par des sources de contexte. Alors que, les contextes composites sont de haut niveau et qui ne peuvent pas avoir une

contrepartie directe sur le côté de provision de contexte. Un contexte composite agrège multiples contextes, que ce soient atomiques ou composites.

La classe `ContextSource` modélise les ressources d'où les contextes sont récupérées. Elle est spécialisée par les sous classes `ContextService` et `ContextServiceCommunity`. `ContextService` est fourni par un fournisseur autonome pour la collecte, le raffinage, et la diffusion de l'information de contexte. Divers `ContextServices` peuvent fournir des informations de contexte très différentes et hétérogènes. Pour résoudre les problèmes d'hétérogénéité, `ContextServiceCommunity` agrège multiples services de contexte dans une communauté de services Web [83] pour fournir une interface unifiée sans se référer à un service de contexte spécifique. Un autre avantage résultant de l'utilisation des communautés de services de contexte est la provision dynamique des informations contextuelles. Lorsque le fonctionnement d'une communauté est invoquée, cette dernière est chargée de sélectionner le service de contexte le plus approprié (au moment de l'exécution) pour fournir l'information de contexte demandée. Les services de contexte peuvent rejoindre ou quitter les communautés à tout moment [113].

4.2.1.2 Modélisation de la Sensibilité au Contexte

`CAMechanism` est une classe qui formalise les mécanismes de la sensibilité au contexte. Deux catégories de mécanismes sont présentées : `ContextBinding` et `ContextTriggering`. Les mécanismes de sensibilité au contexte sont assignés à des objets sensibles au contexte, modélisés par des instances de `CAObject` à travers la relation `MechanismAssignment`, et indiquant quels objets ont quels types de mécanismes de sensibilité au contexte. `CAObject` est une classe de base pour tous les éléments du modèle ContextUML qui représente des objets sensibles au contexte. Il existe quatre sous-types de `CAObject` : `Service`, `Operation`, `Message` et `Part`. Un mécanisme de sensibilité au contexte peut être assigné à un service, une opération d'un service, messages d'entrée/sortie d'une opération, ou même une partie d'un message particulier.

La classe `ContextBinding` modélise la liaison automatique des contextes à des objets sensibles au contexte (par exemple, le paramètre d'entrée d'une opération de service).

La classe `ContextTriggering` modélise quand et comment l'adaptation contextuelle devrait se produire. Ce mécanisme de déclenchement de contexte contient, à la fois, un ensemble de contraintes sur le contexte (`ContextConstraint`) et un ensemble d'actions (`Action`), et précise que l'ensemble des actions sera exécuté lorsque toutes les contraintes de contexte ont la valeur vraie [113].

4.2.1.3 Avantages de ContextUML

ContextUML a plusieurs avantages [113] ; non seulement il fournit des mécanismes de traitement de contexte pour le développement des services sensibles au contexte, mais il offre aussi des primitives riches pour la modélisation de contextes et leur capture. Ce langage permet également le développement des services Web sensibles au contexte et offre une flexibilité de conception importante aux concepteurs de ce genre d'application. Ceci est réalisé par les aspects majeurs suivants :

Premièrement, la séparation entre la modélisation de contexte et la sensibilité au contexte de composants de service ; plusieurs mécanismes de sensibilité au contexte sont

abstraits et qui sont assignés à des composants de service appropriés pour atteindre la sensibilité au contexte. Cette séparation facilite à la fois le développement et la maintenance des services sensibles au contexte.

Deuxièmement, l'abstraction des communautés de services de contexte offre une grande flexibilité pour le provision du contexte par la liaison dynamique des services de contexte, et assure une haute qualité d'information de contexte par la sélection basée sur des politiques de qualité. Les communautés rendent également la provision de contexte plus robuste. Par exemple, si un service de contexte sélectionné à partir d'une communauté devient indisponible, un autre service de contexte de la même communauté peut être sélectionné pour assurer la haute disponibilité des services Web [82].

Enfin, les contextes composites améliorent la capacité de modélisation de l'information de contexte pour les concepteurs de services sensibles au contexte. En appliquant les contextes composites, les concepteurs de services peuvent modéliser tous les attributs de contexte de haut niveau qui sont utiles dans les services sensibles au contexte.

Toutefois, selon [117], même si ContextUML permet la modélisation de règles de dérivation, il ne comprend pas les moyens pour modéliser la vie privée des utilisateurs. En outre, comme ContextUML fournit une extension de poids lourd au métamodèle UML modifiant les métaclasses UML, il ne peut pas être utilisé par les outils standards de modélisation UML.

Suite à cela, Sheng et al. ont récemment présenté dans [114] la plateforme **Context-Serv** conçue pour le développement rapide des services Web sensibles au contexte. Context-Serv adopte le développement dirigé par les modèles (MDD) et utilise ContextUML pour la spécification des services sensibles au contexte. La plateforme offre également un ensemble d'outils automatisés pour la production et le déploiement des implémentations exécutables des services Web sensibles au contexte. Ainsi et dans [80], nous trouvons un prototype concret d'application Web sensible au contexte appelée «Smart Adelaide Guide» développée avec ContextServ, et qui a pour objectif d'aider les touristes d'Adélaïde (la capitale de l'Australie du Sud) à découvrir ses endroits intéressants, en fonction de leur localisation actuelle, les langues préférées et les conditions météorologiques. Cette application Web s'appuie sur la recherche des services Web d'attractions sensibles au contexte capturé.

4.2.2 Approches basées sur ContextUML

Pour bénéficier du travail de [113], diverses approches ont proposé des versions étendues et modifiées de ContextUML, nous pouvons citer ici deux types de travaux intéressants : les travaux de [103, 33, 56]qui présentent une approche basée sur la conception dirigée par les modèles et la programmation orientée aspect [49], et le travail de [29] qui traite l'information de contexte dans le projet des Services Mobiles Simples (SMS).

4.2.2.1 ContextUML et les Aspects

Prezerakos et al., dans [103], ont vu que le métamodèle ContextUML nécessite plusieurs modifications telles que :

1. Minimiser l'association entre les parties modélisant les services et le contexte,

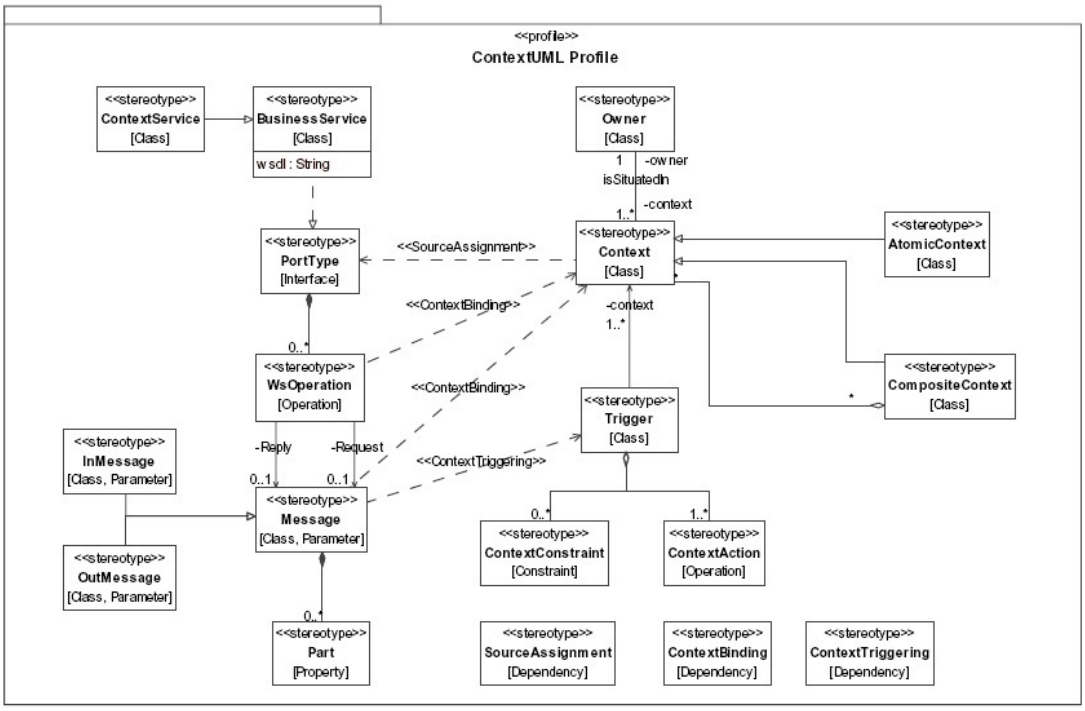


FIGURE 4.2 – Le métamodele modifié de ContextUML

2. Éliminer les relations non nécessaires qui existent entre les artéfacts du métamodèle ContextUML, et
3. Clarifier les sémantiques existantes du métamodèle afin de faciliter la transformation du modèle de service résultant en code par un outil de transformation de modèles.

D'où ils ont présenté une version modifié du modèle ContextUML [103]. Les différences entre l'original et le métamodèle ContextUML modifié se résument en deux grandes catégories : les différences dans les stéréotypes contenus dans les deux métamodèles (voir Figure 4.2) et les différences dans l'utilisation de ces stéréotypes lors de la création d'un modèle de service ou de contexte.

Ce travail est l'un des rares travaux qui ont traité le contexte avec la Programmation Orientée Aspect (AOP) [49], dont les auteurs ont basé sur l'idée qui considère la logique de base de service et la gestion de contexte comme des préoccupations distinctes au niveau de la modélisation ainsi que dans le code source résultant, où l'AOP encapsule le comportement dépendant du contexte dans des modules de code discrets (la fonctionnalité du service de base et la gestion de contexte peuvent être considéré comme deux problèmes distincts dont la gestion de contexte est transversale dans la fonctionnalité du service de base).

AOP est un paradigme de programmation visant à la manipulation de telles préoccupations transversales (ou aspects). Elle a été implémentée comme une extension des langages de programmation les plus populaires (AspectJ, AspectC++,... etc.) où il introduit les notions des `joinpoints` qui définissent les points de jonction du programme où les aspects peuvent introduire des comportements supplémentaires, `pointcuts` qui définissent des expressions pour détecter les joinpoints et des pièces de `advice` qui décrivent le code à appliquer sur les joinpoints. Les aspects sont, donc, constitués des pointcuts et des advices respectifs. AOP est idéal pour les préoccupations transversales homogènes, ce qui signifie que le même code peut être appliqué à des endroits différents. L'utilisation de

l'information de contexte suivant le paradigme AOP peut être traitée par les aspects qui interrompent l'exécution du service principal afin de réaliser l'adaptation au contexte des services [72].

Dans la phase conception de [103], Prezerakos et al. ont décrit un processus de modélisation constitué des étapes suivantes :

1. Découverte des services Web pertinents et leur transformation en diagrammes UML,
2. Conception des modèles de contexte et de la logique de service en se basant sur ces diagrammes UML et le métamodèle ContextUML,
3. Association des éléments de modèle de contexte avec des sources de contexte respectives,
4. Association des éléments logiques de service avec les éléments respectifs du modèle de contexte, et
5. L'assemblage manuel des services Web découverts dans un workflow composite en utilisant un diagramme d'activité UML.

Dans la phase de codage, les modèles UML de service et de contexte, résultants de la phase ci-dessus, sont exportés sous forme de fichiers XMI et introduits dans un outil de transformation de modèle qui, en raison des stéréotypes du modèle ContextUML, les transforme en source Java. L'utilisation d'un moteur AOP est supposé comme le mécanisme qui permet l'extension du comportement des services existants. Les informations de liaison de contexte fournies dans les modèles UML sont utilisées pour créer les pointcuts et les advices liés, ainsi que pour créer la liaison entre eux.

L'utilisation efficace des techniques de MDD, dans cette approche, permet aux développeurs de services de gérer la logique métier et le comportement des services dépendant du contexte comme deux préoccupations distinctes au sein du même modèle et donc, toute modification du comportement dépendant du contexte se fera sans graves répercussions sur la fonctionnalité principale du service. Aussi, la séparation des préoccupations peut être réalisée au niveau du code source et par conséquent pendant l'exécution de service par l'encapsulation de la fonctionnalité dépendante du contexte dans des aspects séparés dans un langage de programmation de haut niveau.

Pour obtenir les avantages de la combinaison du MDD avec l'AOSD (Aspect-Oriented Software Development), Carton et al., dans [33], présentent une autre approche qui considère que chaque application sensible au contexte (basée sur les services) possède un ensemble d'aspects. Le paradigme orienté aspect permet de concevoir et d'implémenter des préoccupations séparées selon l'information de contexte dont l'adaptation devrait se produire de manière indépendante de l'application de base. Dans cette approche, les préoccupations sont conçues par l'utilisation du Theme/UML, qui est une extension d'UML standard fournit des constructions pour supporter la spécification des comportements transversaux des aspects par rapport à des modèles abstraits déclencheurs avec un moyen de correspondance pour spécifier les éléments de l'application de base qui se lient à ces modèles déclencheurs. Pour l'implémentation de ces préoccupations, AspectJ [1] est utilisé et qui est une extension orientée aspect du langage Java pour offrir une programmation des constructions modélisant les aspects, et spécifier où et quand dans le code de base s'appliquent les aspects.

Grassi et Sindico décrivent dans [56] un autre modèle conceptuel pour un développement guidé par les modèles des applications sensibles au contexte similaire à ContextUML

[113]. Ce modèle conceptuel est structuré en deux parties principales, la modélisation du contexte et de la modélisation de l'adaptation au contexte. Dans la première, le contexte est catégorisé d'un contexte basé sur les états qui caractérisent la situation actuelle d'une entité et d'un contexte basé sur les évènements qui représentent les changements d'état de l'entité. Les contraintes sont utilisées sur les deux types de contexte pour déclencher diverses invocations : contraintes d'état référent à des points précis dans le temps, tandis que les contraintes d'évènement exploitent les données historiques des évènements de contexte. L'information de contexte peut être utilisée dans la dernière partie pour modifier la structure ou le comportement de l'application. Puis, ils ont identifié deux mécanismes fondamentaux pour introduire la sensibilité au contexte dans une application développée sur la base de l'architecture SOA (Service-Oriented Architecture), à savoir :

1. Par des liaisons sensibles au contexte dont les valeurs sont associés à des entités d'application selon le contexte.
2. Par des « inserts » sensibles au contexte, pour introduire d'autres éléments structurels ou comportementaux à des points spécifiques de l'application suivant le contexte. Le concept du « insert » sensible au contexte, est dérivé des domaine de la conception orientée aspects (AOD) et de la programmation orientée aspects (AOP).

L'information de contexte peut être utilisée pour être transformée à des valeurs spécifiques ou pour modifier la structure ou le comportement dune application. Dans ce travail, des éléments `«Moniteur»` sont responsables de récupérer les diverses informations de contexte, tandis que `«Adaptateurs»` adaptent l'application en se basant sur ces informations contextuelles, par exemple en provoquant des actions supplémentaires à effectuer. Ensuite, ce modèle UML est transformé à un code AspectJ, où les Adaptateurs et les Moniteurs sont implémentés comme des aspects [72].

4.2.2.2 Modélisation du Contexte pour la Réalisation des Services Mobiles Simples

Dans [29], les auteurs présentent un modèle de contexte basé sur UML pour le projet des Services Mobiles Simples (SMS). Le système de SMS se concentre sur la modélisation, la gestion et de fournir des informations de contexte afin de faciliter la création, l'approvisionnement et l'utilisation des services mobiles sensibles au contexte.

SMS considère deux niveaux pour la création de service :

1. Le premier, le niveau de la modélisation de haut niveau pour les programmeurs experts (concepteurs de logiciels) qui est basée sur un profil UML spécifique défini pour les SMS.
2. Le second, le niveau de l'assistant de création, est destiné aux utilisateurs ayant une expertise technique minimale.

Les deux niveaux proposent de réaliser la composition de services en rassemblant et en adaptant des éléments existants de service ou des services composants.

Pour intégrer l'information de contexte dans la création des services et la modélisation de leur comportement sensibles au contexte, un modèle de contexte pour les SMS a été développé. Ce modèle, inspiré des approches de ContextUML [113] et SIMPLICITY [1]

1. un projet prédécesseur du SMS

[106], comprend trois niveaux d'abstraction différents (méta-modèle, modèle et instance). Le modèle UML résultant peut être utilisé pour obtenir des modèles de contexte à l'aide d'autres langages, par exemple les standards liés à XML tels que 3GPP.

Le niveau Métamodèle du modèle de contexte de SMS définit les entités génériques pour la modélisation du contexte, de sa structure, les propriétés et les relations avec un niveau d'abstraction élevé et indépendamment de l'information de contexte concrète (par exemple, localisation, dispositif, ...). A ce niveau, l'approche de SMS utilise les classes `Context`, `AtomicContext` et `CompositeContext` de ContextUML, mais simplifie la modélisation des sources de contexte et ajoute sa propre classe pour décrire la qualité des paramètres de contexte [29]. Le niveau Modèle réalise le niveau Meta et inclut les différentes caractéristiques des informations de contexte atomique et composite tels que l'emplacement, l'utilisateur ou périphérique. Enfin, le niveau Instance comprend des instances concrètes d'informations de contexte à partir de niveau modèle et donne à un contexte individuel une «identité» en spécifiant des valeurs pour le modèle abstrait.

Ces niveaux d'abstraction peuvent être utilisés comme une base pour transformer le modèle résultant à des divers langages d'implémentation.

4.2.3 Approches basées sur UML et MOF

En plus de ContextUML, plusieurs autres approches ont été proposées en utilisant les standards UML et MOF de l'OMG. Les deux plus importantes de ces approches sont présentées ci-après. La première, décrite dans [44], propose un métamodèle MOF pour le développement des applications mobiles sensibles au contexte. La deuxième approche présentée dans [73] propose une méthodologie et préconise en faveur d'une séparation complète entre les fonctionnalités de l'application Web et l'adaptation au contexte dans toutes les phases de développement (analyse, conception, implémentation).

4.2.3.1 Un metamodèle MOF pour le Développement des Applications Mobiles Sensibles au Contexte

La spécification Meta Object Facility (MOF) [94] est la fondation de la stratégie de métamodélisation de l'OMG qui peut être utilisée pour concevoir la syntaxe abstraite de tout métamodèle. La syntaxe abstraite de MOF est représentée à l'aide de MOF lui-même et utilise un diagramme de classes UML comme sa syntaxe concrète. Donc, nous pouvons supposer que tout métamodèle MOF peut être représenté avec des outils UML, comme une instance d'un modèle MOF est également un diagramme de classes UML.

Dans [44], les auteurs présentent un métamodèle d'information contextuelle basé sur MOF pour le développement des applications sensibles au contexte. Le métamodèle est structuré selon cinq vues : vue de base, de service, de souscription, de services sensibles au contexte et de qualité où chaque vue est représentée par un ensemble de MOF. Les deux principales vues (les vues de base et de service) sont présentés ci-après.

Figure 4.3 montre le diagramme de classe de vue de base [44]. Le métamodèle de contexte est organisé autour de la métaclasse « `CWClassifer` », une structure abstraite qui généralise les notions de base de tout contexte : les entités (métaclasse « `CWEntity` ») et leurs attributs (métaclasse « `CWAttribute` »). Les entités correspondent à des objets physiques ou conceptuel à partir duquel l'information contextuelle est capturée, tels que : les

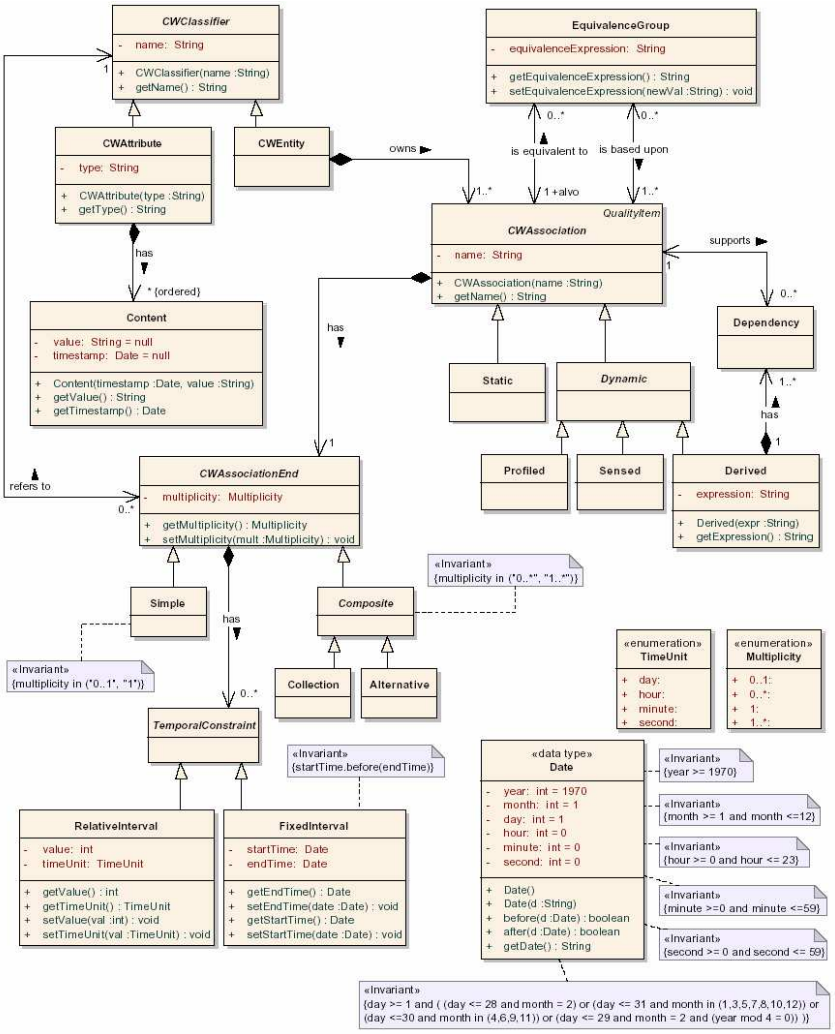


FIGURE 4.3 – La vue Core

personnes, organisations, dispositifs et les lieux. L'information contextuelle est généralement définit comme un attribut. Les attributs peuvent être ajustés d'une manière statique ou dynamique en utilisant des capteurs. Les états qu'un attribut peut assumer, pendant sa durée de vie, sont maintenus comme des instances de la métaclasse « `Content` ». La métaclasse abstraite « `Association` » est utilisée pour définir les relations entre les entités et les attributs et les relations entre les entités elles-mêmes. Chaque association appartient à une entité (« `CWEntity` » propriétaire de « `CWAssociation` »). Toute l'information contextuelle est classée soit statique ou dynamique, selon le type d'association entretenue avec son propriétaire. Les associations statiques (métaclasse `Static`) révèlent une relation à long terme entre l'entité et la valeur de son attribut, qui est resté inchangée au fil du temps. Les associations dynamiques (métaclasse « `Dynamic` ») sont classées selon trois types différents : senti (métaclasse « `Sensed` »), profilé (métaclasse « `Profiled` ») et dérivé (métaclasse « `Derived` »). Ces types d'associations sont utilisés pour souligner les différences en matière d'origine, persistance, niveau de confiance et temporalité, et sont utiles pour guider le fonctionnement de la plateforme sensible au contexte et les applications associées. Les autres concepts de cette vue sont détaillées dans [44].

L'interaction et l'interopérabilité entre les applications sensibles au contexte et la plateforme des services (architecture orientée services, dans ce cas) devrait être assurée par

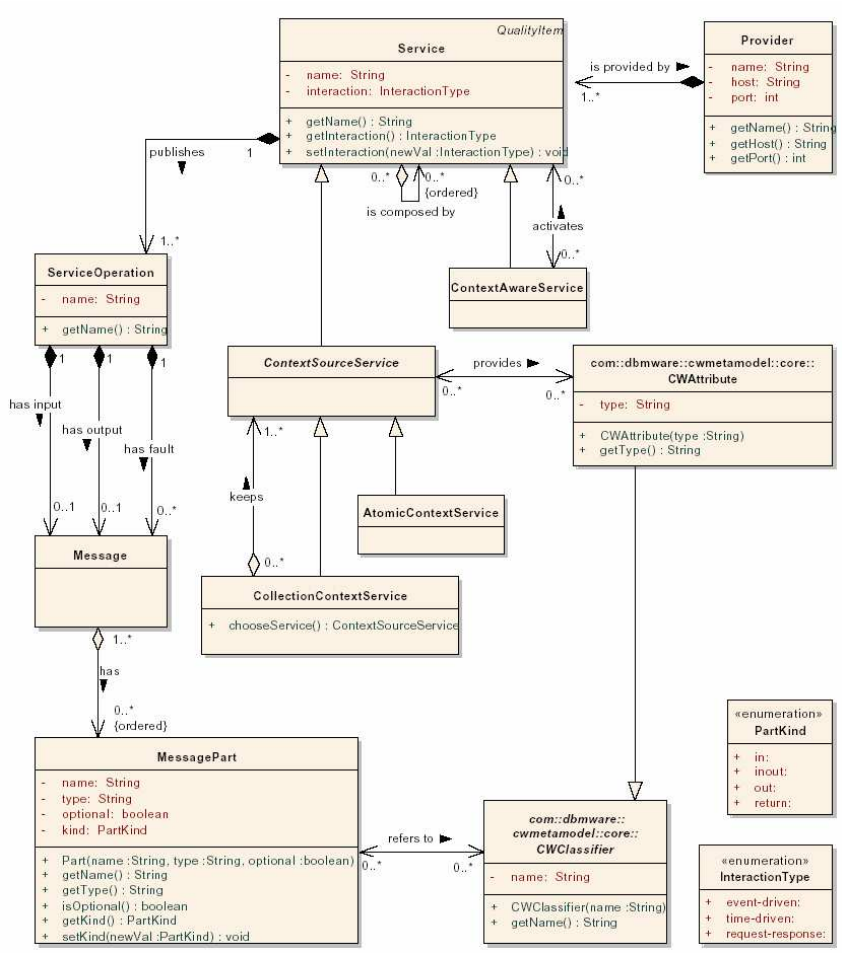


FIGURE 4.4 – La vue Service

des mécanismes standards. Ces derniers devraient se conformer aux exigences suivantes :

1. Assurer un couplage faible entre les applications et les plateformes,
2. Fournir un haut degré de flexibilité pour ajouter, mettre à jour et supprimer des demandes d'information contextuelle, sans impact sur la plateforme ou d'autres applications fonctionnant sur le même environnement.
3. L'adaptabilité à accepter les différents modèles de contexte à condition qu'ils partageant tous le même métamodèle contexte et, enfin
4. L'indépendance avec toute plateforme technologique particulière.

Sur la base de ces exigences, les auteurs ont décidé d'inclure une vue de service (présentée dans la Figure 4.4) dans le métamodèle de contexte. Le principal avantage de l'approche basée sur les services est de tirer parti de l'interopérabilité entre les plateformes, les applications et les fournisseurs de contexte, chaque fois que l'information de contexte est nécessaire, ce qui rend beaucoup plus facile d'ajouter des fonctionnalités de façon standard. L'architecture orientée services assure généralement le découplage des applications par, seulement, exposer les interfaces de services et en offrant des schémas flexibles et dynamiques de composition de services.

Les principales métaclasses dans ce métamodèle sont les suivantes : `Service` représente un service fourni par un spécifique `Provider` et ses instances peuvent être composées avec d'autres services. Un service de source de contexte (métaclasse `ContextSourceService`)

offre les informations contextuelles primaires pour toute plateforme grâce à la méta-association `provides` et peut être fourni par un fournisseur de contexte. A « `ContextSourceService` » peut être soit un service de contexte atomique ou une collection de services équivalents ou alternatifs de différents fournisseurs. La métaclasse `CollectionContextService` représentant ce dernier est responsable de placer les paramètres de QoS utilisés pour choisir entre différents, quoique équivalents, services de contexte qui peuvent varier dans le coût, la disponibilité, la précision, et ainsi de suite. La métaclasse `ContextAwareService` spécialise la métaclasse `Service` et se réfère à tout service métier qui demande ou dépend de l'information contextuelle pour fonctionner. Une application sensible au contexte englobe un ensemble de services sensibles au contexte. Ces services s'appuient sur d'autres `Services` pour effectuer leurs actions (méta-association `activates`). Enfin, un service se compose d'un certain nombre d'opérations, représentée par la métaclasse `ServiceOperation`. Les opérations sont composées de messages (métaclasse `Message`), comme entrées, sorties, ou fautes. Un message est construit de parties de message (métaclasse `MessagePart`). Le rôle joué par chaque partie du message lors de l'interaction peut être définie par l'énumération `PartKind`. Toute partie du message peut se référer à une `CWClassifier`, `CWEntity` ou `CWAttribute`.

4.2.3.2 Développement Dirigé par les Modèles des Applications Web Composites Sensibles au Contexte

L'approche citée dans [73] propose une architecture d'adaptation au contexte des applications Web comprenant les services Web et une méthodologie dirigée par les modèles pour le développement de telles applications composites sensibles au contexte. Cette méthodologie adopte l'approche de l'Ingénierie Dirigée par les Modèles qui utilise des diagrammes UML [90] au cours de la phase de conception. Il est clair que le modèle UML concret est suffisant pour générer automatiquement un service Web fonctionnel d'une application Web à travers un processus de transformation adéquat [19].

Au niveau de la modélisation, la conception est, dans une large mesure, maintenue indépendante de la plateforme spécifique d'implémentation et suffisamment flexible pour permettre l'introduction des différents codes des correspondances spécifiques. Durant les phases de développement et d'exécution de l'application, la logique de service et celle de l'adaptation au contexte sont maintenues indépendantes, offrant ainsi la flexibilité en facilitant la maintenance de l'application.

Dans cette approche, la modélisation exploite un certain nombre de profils prédéfinis (un profil de service Web, un métamodèle de contexte et un profil de présentation), alors que l'implémentation cible est basée sur une architecture qui réalise l'adaptation au contexte des services Web basée sur l'interception des messages SOAP (Simple Object Access Protocol). En outre, il est démontré que le processus d'adaptation au contexte est totalement transparent du noyau de l'application Web. Cette approche a plusieurs avantages [72], notamment :

- L'adaptation au contexte est effectuée au niveau de l'interface de service, et le client est maintenu indépendant et ne nécessite aucun logiciel spécifique sur son côté.
- La modélisation utilise une notation UML plutôt que d'un Langage Spécifique de Domaine (DSL) propriétaire, offrant ainsi la compatibilité avec les outils de modélisation UML.

– L'adaptation au contexte et la modélisation d'application sont abordés simultanément entrainant la génération d'applications Web sensibles au contexte, sans introduire des liaisons indésirables entre l'application et la fonctionnalité d'adaptation au contexte.

4.2.4 Approches ad-hoc basées sur un DSL

Un certain nombre d'œuvres, parmi lesquels [6] et [127], ont également proposé des Langages Spécifiques de Domaine (DSL) pour la modélisation des services sensibles au contexte. Un DSL est un langage de programmation ou langage de spécification exécutable qui offre, par des notations appropriées et des abstractions, la capacité expressive axée sur un domaine, généralement limité, de problème particulier. Par rapport aux approches présentées dans les sections précédentes, les solutions présentées ici ne sont donc pas basées sur UML, mais plutôt sur un langage dédié spécifiquement et conçu pour s'adresser au problème de sensibilité au contexte.

4.2.4.1 Development Dirigé par les Modèles des Services Sensibles au Contexte

Dans [6], les auteurs proposent une trajectoire de conception dirigée par les modèles pour les services mobiles et sensibles au contexte. Dans ce travail, un certain nombre de concepts tels que l'indépendance de plateforme, plateformes abstraites, sensibilité au contexte et l'orientation de service jouent un rôle important. Les auteurs présentent cette trajectoire de conception en examinant les niveaux nécessaires de modèles, le choix des langages de modélisation et la définition des plateformes et des transformations.

Leur processus de conception est composé de deux phases principales : la phase de préparation et la phase de création de service. Dans la phase de préparation, les experts identifient les niveaux requis de modèles, de leurs plateformes abstraites et les langages de modélisation à utiliser. En outre, cette phase définit également les transformations entre les niveaux liés de modèles. Les résultats de la phase de préparation sont utilisées dans la phase de création de service. Cette dernière implique la création de modèles d'un service spécifique à l'aide des langages de modélisation spécifiques et des plateformes abstraites et l'application (manuelle et automatique) des transformations de modèles. Par la suite, la phase de création de service conduit à une réalisation de service sensible au contexte qui satisfait les besoins des utilisateurs.

Basé sur [41] qui définit le contexte de l'utilisateur comme une «*collection de conditions en corrélation dans lesquelles quelque chose existe ou se produit*», Almeida et al. décomposent le service sensible au contexte, en fonction de l'architecture représentée sur la Figure 4.5. Cette architecture se compose de sources de contexte (`context sources`), qui sont en mesure de sentir le contexte et le représenter comme information de contexte dans le domaine du système. Le service fourni par des sources de contexte est utilisé par un composant de coordination (`Coordination component`), qui demande des actions exécutables par les fournisseurs d'actions (`actions providers`) en fonction des situations déduites de l'information de contexte.

Les composants d'utilisateur (`User components`) et le composant de coordination exhibent le comportement spécifique du service, et sont appelés composants de service. En revanche, les sources de contexte et les fournisseurs d'action sont d'usage général et, par

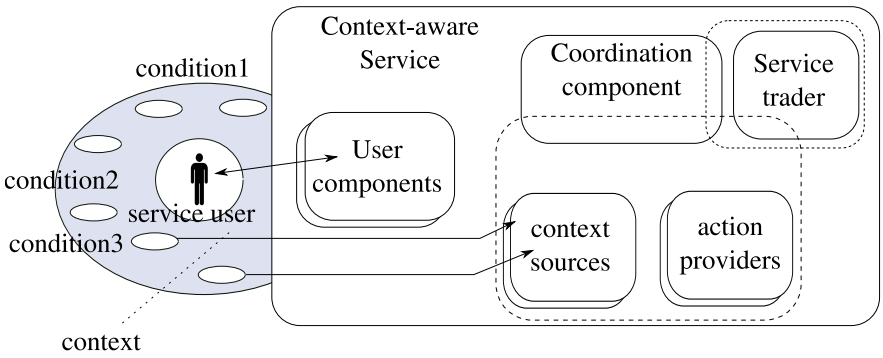


FIGURE 4.5 – Décomposition du service sensible au contexte

conséquent, peuvent être réutilisés dans différents services sensibles au contexte. Le service fourni par ces derniers pour le composant de coordination est enregistré dans un opérateur service (`Service trader`). Cela permet au composant de coordination de sélectionner les sources de contexte et les fournisseurs d'action dynamiquement en fonction de offres de service qui sont enregistrés dans l'opérateur de service.

Afin de faciliter la transformation du niveau de spécification de services au niveau de réalisation de service, les auteurs introduisent un niveau intermédiaire de modèles. Et donc, les trois niveaux de modèles pour la conception d'un service sensible au contexte sont :

Niveau de spécification : Modélise le comportement d'un service sensible au contexte à un haut niveau d'abstraction et de son environnement dans une perspective intégrée, en utilisant la notion d'action pour modéliser, à la fois, l'occurrence d'évènements provenant de sources de contexte et de l'exécution des actions.

Niveau de conception indépendante de la platforme : Décrit le comportement d'un service sensible au contexte dans une perspective interne, révélant un composant de coordination spécifique du service et la plateforme de service A-MUSE. Fondée sur SOA, la plateforme A-MUSE fournit une abstraction des plateformes médiatrice et de recherche des services et comprend les services de contexte et d'action. En outre, cette plateforme abstraite prend en charge la découverte des services avec des propriétés de service dynamique, qui permet de découvrir des services basés sur des informations de contexte.

Niveau de conception spécifique à la platforme : Décrit la réalisation d'un service sensible au contexte dans le niveau spécifique à la plateforme. Il est décrit en termes de technologies cibles spécifiques, tels que technologies des services Web et BPEL. Des modèles de transformation peuvent être utilisés dans cette réalisation.

Ces niveaux d'abstraction présentent l'avantage principal de cette approche de développement pilotée par les modèles. A l'opposition, l'inconvénient enregistré dans ce travail est l'inexistence des niveaux métamodèle pour le contexte et la sensibilité au contexte.

4.2.4.2 Modèle de Rôle Dépendant du Contexte

Une autre approche de cette catégorie est présentée par Vallejos et al. dans [127]. Ils proposent un modèle de programmation orienté objet basé sur les rôles, appelé modèle de rôle dépendant du contexte (CDR), afin de faciliter le développement des adaptations dépendantes au contexte dans les systèmes distribués mobiles. Ce modèle consiste à ce que :

1. Les rôles représentent les différentes adaptations de comportement d'une application logicielle qui peuvent être adoptées dynamiquement selon le contexte,
2. Une application décide, de manière autonome, le rôle approprié en fonction du contexte de l'ensemble des participants, et
3. Une adaptation est strictement délimitée par la portée d'une interaction.

Ainsi, dans le modèle CDR, un acteur encapsule une hiérarchie de délégation composée d'un comportement par défaut et de ses différentes adaptations dépendantes du contexte, tous sont représentés comme des rôles, les acteurs répondent aux messages en sélectionnant d'abord le rôle approprié et ensuite l'exécution de la méthode correspondante dans l'objet d'adaptation de ce rôle. Le rôle requis pour l'exécution d'un message est sélectionné de façon autonome par l'acteur qui reçoit le message en utilisant le sélecteur de rôle dépendant du contexte et en fonction du contexte de l'expéditeur et le destinataire du message. Les adaptations ont une portée d'action délimitée qui est définie par l'exécution d'un message, et une référence de contexte permet à l'expéditeur du message d'être sensible du contexte exposé au récepteur du message.

Le modèle CDR est validé par une implémentation d'une extension de AmbientTalk [45], un langage de programmation spécialement conçu pour les applications informatiques pervasives.

4.3 Évaluation et leçons apprises

D'après l'étude précédente sur la conception et le développement des applications sensibles au contexte à base de services et les différentes approches abordées dans la littérature, et en s'inspirant sur le travail de [72], nous sommes arrivé à déterminer les critères qui peuvent constituer la base de notre comparaison entre ces différentes approches, à savoir :

- Le langage de représentation ou modélisation du contexte.
- Le degré de découplage entre la logique métier du service et la gestion du contexte.
- La conformité avec le standard MDA (PIM, PSM, processus de transformation, ...).
- Le temps d'adaptation à l'information de contexte (au moment de la conception ou de l'exécution)
- Le côté où l'adaptation au contexte se produit (Client et/ou Serveur).
- Le traitement des aspects de la sécurité et de la vie privée.
- La plateforme d'implémentation utilisée.

En fonction de ces points, le tableau 4.1 résume les approches discutées et qui sont les plus représentatives dans le développement dirigé par les modèles des applications sensibles au contexte et basés sur les services.

	Modélisation de Contexte	Découplage entre contexte et logique métier	Conformité avec MDA	Temps d'adaptation au contexte	Coté d'adaptation au contexte	Sécurité et vie privé	Implémentation (plateforme / framework)
***ContextUML* [113]**	Basé-UML	Oui	Non	@design-time	Client	Non	Services Web
***AOP* [103]**	Basé-UML	Oui	Non	@run-time	Client	Non	Services Web
***SMS Project* [29]**	Basé-UML	Oui	Partiel	@design-time	Serveur	Non	Divers
***MOF metamodel* [44]**	Basé-UML	Oui	Partiel	@design-time	Serveur	Non	Infraware platform
***Composite context-aware* [73]**	Basé-UML	Oui	Partiel	@run-time	Client/Serveur	Non	Services Web
***A-MUSE Service Platform* [6]**	ECA-DSL	Oui	Partiel	@design-time	Serveur	Non	A-MUSE Service Platform(services Web, BPEL)
***CDR model* [127]**	Basé-Impl.	Partiel	Non	@run-time	Client/Serveur	Oui	AmbientTalk framework

TABLE 4.1 – Synthèse des approches de développement des applications sensibles au contexte à base de services

Nous observons que toutes les approches ont réalisé le découplage entre la gestion de contexte et la logique métier à un certain degré pour chacune. Pour la la représentation du contexte, certaines approches utilisent les modèles que ce soient génériques (par exemple, UML) ou spécifiques (en cas de DSL). Autres approches citées dans [71] utilisent des techniques à base de code et d'interception de messages pour la représentation du contexte qui sont hors de la portée de cette thèse. Malheureusement, même que les issues de sécurité et de la vie privée (confidentialité) sont des aspects essentiels pour les services sensibles au contexte, ils ne sont considérées que par un très peu nombre d'approches comme il est montré dans [72].

Différents langages d'implémentation, plateformes sous-jacentes et médiateurs sont adoptés par chaque approche dont la plupart ont maintenu un certain degré d'indépendance pour le développement de ce type de applications (c.-à-d., l'adaptation se déroule sans affecter la technologie médiatrice sous-jacente). L'adaptation peut prendre place à un côté (client ou serveur), alors qu'il y a aussi des cas où les deux sont considérés comme les acteurs équivalents (cela arrive, par exemple, dans le modèle de rôle dépendant du contexte).

Aussi, et bien que la conformité avec le standard MDA et l'adaptation contextuelle dynamique pendant l'exécution sont des aspects essentiels, ils sont ignorés partiellement ou complètement par ces approches.

Toutefois, ces limitations représentent des sujets de recherche intéressants dans l'avenir pour le développement des applications sensibles au contexte, à savoir :

– **Conformité avec MDA** : la technologie MDA est l'approche de génie logiciel la

plus récente proposée par l'OMG en tant que norme d'ingénierie dirigée par les modèles. L'adoption de la modélisation et le développement en suivant la norme MDA pour les applications sensibles au contexte répond à de nombreuses exigences en matière de génie logiciel tels que l'interopérabilité, la généricité et la réutilisabilité.

- **Adaptation contextuelle dynamique** : le contexte, de sa nature, se change fréquemment, ce qui nécessite une adaptation rapide et dynamique des applications sensibles au contexte pendant le temps d'exécution sans interrompre la disponibilité du service. Une telle adaptation dynamique est seulement traitée par quelques travaux et souvent considérée au niveau du code, par exemple, en utilisant la programmation orientée aspect. Nous voyons que l'adaptation dynamique des applications basées sur les services doit être considérée à un niveau beaucoup plus abstrait en utilisant des modèles, et nous croyons que la modélisation orientée aspect et/ou les modèles@Runtime, développées récemment, par la communauté de l'ingénierie dirigé par les modèles, sont des domaines très prometteurs et peuvent servir cette considération.
- **Sécurité et confidentialité :** la vie privée est la demande des individus, des groupes et des institutions afin de déterminer eux-mêmes, quand, comment et dans quelle mesure les informations de leur sujet est communiqué aux autres [130]. Tous les développements de applications sensibles au contexte, visant à atteindre la vie privée, doivent mettre en œuvre des lois et des règlements pour protéger la vie privée de l'individu, par exemple, garder certaines données secrètes.

Pour notre travail, nous allons prendre en considération seulement les deux premiers points pour proposer notre approche.

4.4 Conclusion

A travers ce chapitre, nous avons présenté une étude sur les approches et les méthodologies proposées pour le développement dirigé par les modèles des applications/services sensibles au contexte. Nous avons décrit et classé les différentes solutions présentées dans la littérature. L'avantage commun entre ces approches est la séparation claire entre la gestion de la logique métier de l'application et le celle du contexte d'adaptation. La conception des applications sensibles au contexte est une tâche très compliquée, mais avec l'exploitation des principales contributions offertes par le standard MDA (comme, les différentes transformations automatiques ou semi-automatiques entre les modèles y compris la génération du code), la mission des développeurs de tels services devient claire et son accomplissement nécessite moins d'efforts.

Dans la deuxième partie de cette thèse, nous allons proposer notre approche basée sur MDA et la modélisation orientée aspect (AOM) pour développer les applications à base de services sensibles au contexte, tout en suivant une méthodologie axée essentiellement sur l'indépendance complète entre la logique métier et les mécanismes de gestion de contexte, et prenant en compte l'adaptation au changement de contexte de la conception à l'exécution. Mais avant détailler ceci, nous devons d'abord présenter notre modèle pour représenter les informations de contexte dans le chapitre qui suit.

Deuxième partie

Contributions et Étude de Cas

Chapitre 5

Modélisation de Contexte avec les Ontologies

5.1 Introduction

Comme nous avons vu précédemment (Sous-section 2.3.1), plusieurs définitions de l'information de contexte ont été fournis dans la littérature et résumées dans [13], la plus populaire est celle de Dey et al. [46] qui définit le contexte comme : « Toute information qui peut être utilisée pour caractériser la situation des entités (c'est à dire, si une personne, un lieu ou un objet) qui sont considérés comme pertinents pour l'interaction entre un utilisateur et une application, y compris l'utilisateur et l'application eux-mêmes. Le contexte est typiquement l'emplacement, l'identité et l'état des personnes, des groupes et des objets informatiques et physiques » . Dans ce travail , nous nous sommes concentrés sur cette définition pour présenter notre approche.

La première étape dans le développement des applications sensibles au contexte consiste à modéliser correctement toutes les informations de contexte [113] avec une précision très élevée afin de les stocker dans un référentiel en utilisant le meilleur formalisme possible pour faciliter, par la suite, leur exploitation. Une modélisation précise de contexte conduit à fournir un comportement correct dans une forme correcte à l'utilisateur correct au moment et dans l'endroit corrects [122].

D'après la section 2.3.1.4, les ontologies offrent un formalisme fiable pour construire des terminologies consensuelles du domaine de contexte de façon formelle [120, 71, 16, 5] afin qu'ils puissent être plus facilement partagés et réutilisés par les différents partenaires dans des environnements mobiles et ubiquitaires. Les modèles ontologiques ont des avantages évidents en matière de soutien de l'interopérabilité et de l'hétérogénéité. En outre, puisqu'elles soutiennent la représentation des relations complexes et des dépendances entre les données de contexte, elles sont particulièrement bien adaptées par leur connaissance des abstractions de contexte de haut niveau [16]. Dans la littérature, un ensemble d'ontologies ont été développés spécifiquement pour une utilisation dans l'informatique pervasive dont la plupart sont résumés dans [120, 133, 16]. Mais, aucune d'entre eux ne semble couvrir adéquatement l'espace des préoccupations applicables dans la conceptions des applications. A cet effet, le but de ce chapitre est de présenter un modèle de contexte général basée sur les ontologies et qui remplit bien les exigences suivantes :

1. Suffisamment général pour être utilisé par les différentes applications sensibles au

contexte centrées sur l'utilisateur,

2. Suffisamment spécifique pour couvrir les principales entités contextuelles proposées dans la littérature des applications mobiles et sensibles au contexte, et
3. Suffisamment flexible pour permettre son extension par la prise en compte des nouvelles entités spécifiques à un domaine d'application donné.

Ce modèle sera construit suite à la combinaison du développement dirigé par les modèles (MDD) et les ontologies pour bénéficier de plus d'avantages des deux paradigmes. Le métamodèle de définitions des ontologie ODM (Ontology Definition Metamodel) proposé par l'OMG [93] permet d'impliquer facilement ce modèle ontologique dans un processus de développement piloté par les modèles.

5.2 Un métamodèle de contexte

Dans l'ingénierie dirigée par les modèles, l'utilisation des métamodèles permet d'assurer, non seulement, une syntaxe précise à une représentation commune à différents modèles, mais aussi une sémantique propre à un domaine d'application. Comme nous nous sommes intéressés aux applications sensibles au contexte centrées sur l'utilisateur, différents métamodèles de contexte ont été proposés dans ce sens. Certains d'entre-eux sont d'un niveau d'abstraction ne permettant élevé pas de connaitre les entités contextuelles de base [113, 44]. D'autres sont spécifiques à un type d'application particulier tel que l'environnement d'une maison intelligente [60] ou à un domaine d'application spécifique tel que celui du tourisme [32], ou encore plus spécifiquement à la recherche intelligente d'un restaurant [29]. Nous proposons, ici, d'utiliser un metamodèle plus large dans le cadre MDA.

5.3 Les ontologies et le contexte

Dans cette section nous allons discuter les avantages d'exprimer le contexte avec les ontologies.

5.3.1 Définition des ontologies

Le terme « ontologie » a été emprunté de la philosophie et introduit dans le domaine de l'ingénierie des connaissances comme un moyen pour faire l'abstraction et la représentation des connaissances[58]. L'ontologie est un moyen prometteur pour le partage des connaissances et la réutilisation. Il permet la capture et la spécification des connaissances d'un domaine avec sa sémantique intrinsèque à travers une terminologie consensuelle et des axiomes formels. Les ontologies s'appuient sur un ensemble de primitives de modélisation pour définir les classes, les individus, leurs attributs et leurs relations. Les axiomes formels précisent le sens voulu de la terminologie avec les primitives de modélisation et de leurs relations apriori d'une manière explicite [61]. La terminologie consensuelle permet de partager et de réutiliser des données et des connaissances entre différents composants d'un système et aussi, entre les différents systèmes pervasifs.

5.3.2 Avantages des ontologies dans la modélisation du contexte

Différents modèles basés sur l'ontologie de l'information de contexte sont proposées dans la littérature (voir la Section 5.4, ci-après) où chacun d'eux argumente les avantages de l'utilisation des ontologies dans la modélisation du contexte. Dans ce qui suit, nous synthétisons le plus important de ces avantages communes sur lesquels nous sommes basés pour proposer notre modèle ontologique contextuel.

5.3.2.1 Formalisme extensible à grande échelle

Les ontologies sont représentés par plusieurs langages Web, tel que OWL [1] [84], qui mettent l'accent sur la prémisse de l'hypothèse du monde ouvert OWA (Open World Assumption) [48]. Cela signifie que nous ne pouvons pas supposer que nous savons toutes les informations de contexte sur un domaine d'application. Et par conséquent, le manque d'information spécifique au contexte ne signifie pas le déni. Par exemple, « Enseignant » et « Etudiant » sont des classes d'une ontologie, nous ne pouvons pas dire que l'enseignant ne peut pas être un étudiant parce que nous n'avons pas déclaré que ces deux concepts sont disjoints avec « owl : disjointWith ». En revanche, les bases de données et la programmation orientée objet (POO) se concentrent sur l'hypothèse du monde clos CWA (Closed World Assumption) liée à l'échec ou la négation (Négation comme Echec). Cela revient à croire faux chaque prédicat qui ne peut pas être prouvé pour être vrai [2]. Le CWA s'applique typiquement quand un système a un contrôle complet sur l'information ; ce qui signifie que nous supposons que nous savons toutes les informations sur le domaine. Par conséquent, l'absence d'information spécifique au contexte signifie qu'il est dénié. Par exemple, en POO, si une base de données d'une compagnie aérienne typique ne contient pas une attribution de siège pour un voyageur, cela signifie que le voyageur n'a pas fait son enregistrement.

En conclusion, quand nous voulons décrire une information de contexte dans un environnement ouvert et à grande échelle tels que dans les environnement omniprésents ou pervasifs, nous devrions être basé sur un formalisme OWA comme celui offert par les ontologies (OWL) pour décrire les connaissances en permettant de créer, ajouter ou modifier d'autres objets (concepts, relations, axiomes, ...) des ontologies de contexte déjà existantes de manière flexible et dynamique.

5.3.2.2 L'expressivité et la richesse sémantique

Les systèmes informatiques pervasifs, par nature, sont complexes et hétérogènes et qui doivent intégrer les informations de contexte à partir des sources diverses et hétérogènes. Les ontologies, caractérisées par le niveau d'expressivité avec des langages du Web sémantique comme OWL, fournissent un mécanisme bien fondé de la représentation et l'échange des informations structurées comme les informations de contexte pour être partagées et réutilisées par les différents systèmes informatiques pervasifs [133]. Dans ce sens, le langage OWL [84] permet, avec plus de capacité, de :

– Décrire les données de contexte complexes avec une conceptualisation consensuelle, y compris les concepts, les taxonomies et les relations entre les concepts et leurs

1. une deuxième édition est disponible à `http://www.w3.org/TR/owl2-overview/`
2. `http://en.wikipedia.org/wiki/Closed_world_assumption`

propriétés, et
- Fournir une sémantique formelle à des données de contexte en utilisant les axiomes et les contraintes. Les axiomes et les contraintes sont les principaux blocs pour la construction de l'interprétation sémantique des concepts et des relations des ontologies, ainsi, de rendre la possibilité de partager et/ou de réutiliser le contexte entre les différentes sources d'information[16].

5.3.2.3 Capacités de raisonnement et d'inférence

Le domaine d'application des ontologies dépasse la modélisation et la description des données au raisonnement sur ces données. Dans la littérature, de nombreux projets de l'informatique ubiquitaire ont appliqués les ontologies pour leur puissance de raisonnement. Les langages de raisonnement (tels que, OWL-DL [3], SWRL [65]) permettent :
- De vérifier la consistance des concepts, et leurs relations permettent de décrire un modèle de contexte.
- Et, plus important, l'obtention d'autres informations de contexte à partir ceux de base en élaborant des règles et en utilisant des des moteurs d'inférence à base de règles.

L'ontologie est une conceptualisation prometteuse qui peut être utilisée pour décrire et capture le maximum d'informations de contexte entourant une situation. Et par un mécanisme de raisonnement, elle permet de déduire d'autres informations contextuelles qui pourraient compléter le contexte d'une situation décrite.

5.4 Modèles ontologiques proposés

Il existe peu de travaux utilisant une approche hybride combinant les ontologies et le développement dirigé par les modèles (MDD) pour représenter les informations de contexte dans un environnement pervasif. A notre connaissance, il n'y a que deux œuvres. Dans [98], Ou et al., ont examiné la façon dont l'architecture dirigée par les modèles d'OMG (MDA) [91] peut être appliquée afin de résoudre le problèmes de la modélisation de contexte, et de la modélisation et le développement de l'application sensible au contexte (CAA). Un Modèle ontologique contextuel (COM) a été présenté pour modéliser les informations de contexte à deux niveaux : le niveau supérieur (ULCOM) et le niveau spécifique étendu (ESCOM). En utilisant de méta-modèles RDFS/OWL, ULCOM capture l'ontologie de concepts qui sont essentiels pour une caractérisation générique du contexte dans le domaine des services pervasifs sous trois entités principales, « Entity », « EntityProperty », et « EntitySpecification ». ESCOM définit les concepts spécifiques et leurs propriétés pour le contexte comme des extensions des entités ULCOM. Par exemple, les concepts spécifiques PDA, ordinateur portable, PC, téléphone mobile existent normalement dans tout environnement informatique pervasif. Egalement dans ce travail, une Architecture d'Intégration dirigée par les Modèles (MDIA) est proposée pour intégrer les spécifications du modèle rigoureux et générer des implémentations de la CAA soit d'une façon automatique ou semi-automatique. Le principal avantage de cette approche est d'étudier la faisabilité de la fusion de l'UML, MDA et les langages d'ontologie (comme RDFS et OWL) pour la

3. `http://www.w3.org/TR/owl-ref/#OWLDL`

modélisation du contexte à base d'ontologie, et de l'architecture d'intégration basée sur MDA pour le développement automatique des applications sensibles au contexte visant à améliorer la précision et réduire le temps et les coûts. Dans la pratique, une étude de cas a été présentée dans [55] montrant en détail le développement d'une application sensible au contexte de m-commerce de tourisme en suivant l'approche proposée.

En outre, Vale et Hammoudi, dans [126], ont présenté une architecture pour le développement d'applications sensibles au contexte mobiles basée sur le métamodèle de définitions des ontologies (ODM) dans lequel des métamodèles de contexte et de sensibilité de contexte sont exposés pour leur utilisation dans un processus de développement guidé par les modèles.

Autrement, nous trouvons plusieurs propositions de modèles de contexte basé sur les ontologies dans la littérature [133]. Chen et al., ont développé « CoBra-Ont » [38], une ontologie OWL qu'est une collection d'ontologies pour décrire les lieux, les agents, les évènements et les propriétés associées dans les environnements intelligents de la zone d'accueil. Ce travail est inscrit dans leur architecture courtière de contexte appelée « CoBrA » qui fournit le partage des connaissances, le raisonnement sur contexte, et les supports de protection de la vie privée pour les systèmes pervasifs sensibles au contexte.

Sur la base de l'ontologie précédente, les mêmes auteurs ont présenté un autre modèle de contexte [39]. C'est une ontologie standard pour les applications ubiquitaires et pervasives « SOUPA » qui permet le partage des connaissances, le raisonnement sur le contexte et l'interopérabilité dans un environnement ubiquitaire. SOUPA traite plusieurs domaines de l'informatique omniprésente que Cobra-Ont et de mapper plusieurs de ses concepts à des concepts d'ontologies communes existantes comme l'ontologie friend-of-a-friend (FOAF) [4] pour permettre l'interopérabilité en utilisant le mappage ontologique des constructions standards d'OWL(owl :equivallentClass et owl :equivallentProperty). Il est constitué de deux ensembles d'ontologies dites « SOUPA-Core » et « SOUPA-Extensions ». SOUPA-Core définit un vocabulaire générique qui est universel pour les différentes applications de l'informatique pervasive (par exemple, personne, agent, politique, temps, espace), tandis que SOUPA-Extensions définit des ontologies supplémentaires pour soutenir le type spécifique d'applications et de fournir des exemples pour étendre les ontologies futures (par exemple, maison, bureau, attraction).

L'ontologie de contexte CONtext ONtology [128] est une ontologie OWL contextuelle pour la modélisation du contexte dans l'environnement de l'informatique omniprésente et le support du raisonnement de contexte basé sur la logique. Wang et al., ont proposé une ontologie supérieure de contexte qui capture les concepts généraux sur le contexte de base tels que l'emplacement, l'activité, la personne ou l'entité de calcul, et des ontologies spécifiques de domaine pour la détention de ces concepts généraux et leurs caractéristiques dans chaque sous-domaine couvert de manière hiérarchique. En outre et en fonction de cette ontologie de contexte, les auteurs ont étudié l'utilisation de raisonnement logique pour vérifier la consistance des informations de contexte, et de raisonner sur le bas niveau du contexte afin de tirer le haut niveau implicite du contexte. D'une manière similaire, le travail de [60] a examiné les notions d'ontologie supérieur et de l'ontologie spécifique à un domaine pour présenter le médiateur orienté service et sensible au contexte (SOCAM).

Dans leur projet « CoDAMoS » [5] et en respectant certaines exigences de l'informatique

4. `http://www.foaf-project.org/`
5. `http://distrinet.cs.kuleuven.be/projects/CoDAMoS/`

mobile, Prenveneers et al. ont présenté une ontologie de contexte adaptable et extensible pour la création d'infrastructures informatiques sensibles au contexte, variant de petits dispositifs intégrés à des plateformes de services de haute de gamme [102]. Elle est définie autour de quatre entités de modélisation : Utilisateurs, Environnement, Plateformes et services. Cette ontologie a été conçu dont le but de résoudre de nombreux défis, tels que l'adaptation de l'application, la génération automatique du code, la mobilité du code, et la génération d'interfaces utilisateur spécifiques au dispositif mobile pour permettre l'interopérabilité dans un environnement d'intelligence ambiante.

Strang et al. [121] décrivent une approche de modélisation de contexte en utilisant des ontologies comme fondation formelle. Ils présentent leur « ASC » modèle et montrent comment il est lié à d'autres modèles. Un langage ontologique de contexte « CoOL » est dérivé de ce modèle, qui est utilisé pour activer la sensibilité au contexte et l'interopérabilité contextuelle lors de la découverte et l'exécution des services dans une architecture distribuée.

5.5 La spécification ODM

Dans les systèmes d'information, une ontologie conceptualise formellement les connaissances dans un domaine comme un ensemble de concepts et de relations entre ces concepts à différents niveaux d'abstraction afin de décrire ce domaine, et agir pour rendre cette description partagée et réutilisée. L'ontologie est une «formelle spécification explicite d'une conceptualisation partagée» [59].

La contribution des ontologies pour comprendre, partager et intégrer l'information est bien établi. En effet, la recherche et la pratique dans ce domaine commencent à porter leurs fruits, notamment pour le Web sémantique. Récemment, les ontologies sont utilisées pour modéliser des informations de contexte dans divers travaux pour le développement d'applications mobiles sensibles au contexte (comme nous avons vu, précédemment, dans la section 5.4). Et pour sa large utilisation par l'ingénierie dirigée par les modèles (MDE), l'organisme OMG a proposé le métamodèle de définitions des ontologies (ODM) [93], un métamodèle standard qui vise à permettre la modélisation d'une ontologie selon le formalisme MOF [94](voir la Figure 5.1). En outre, nous pouvons traiter tous les ontologies par MDE avec tous ses avantages, principalement, le processus de transformation entre les modèles. L'objectif principal de l'ODM est de combler les lacunes existantes entre les ontologies et les espaces de modélisation MDA [54].

Pour cet objectif, ODM définit cinq méta-modèles (RDFS, OWL, Topic Maps, Logique Commune et Logiques de Description), deux profils UML (le profil RDFS/OWL, Topic Maps sur profil) et un ensemble de correspondance QVT de l'UML vers OWL, de Topic Maps vers OWL et de RDFS/OWL à la logique commune [93]. Nous sommes intéressés par la langage ontologique OWL [84] pour sa grande expressivité et surtout par sa version OWL-DL (pour la logique de description) qui permet de faire des raisonnements pour déduire d'autres informations de contexte de haut niveau à partir des informations de contexte de base de bas niveau. Dans ce travail, nous allons basé sur OWL et utiliser le profil UML pour OWL défini dans ODM pour décrire notre modélisation de contexte basée sur l'utilisation des ontologies.

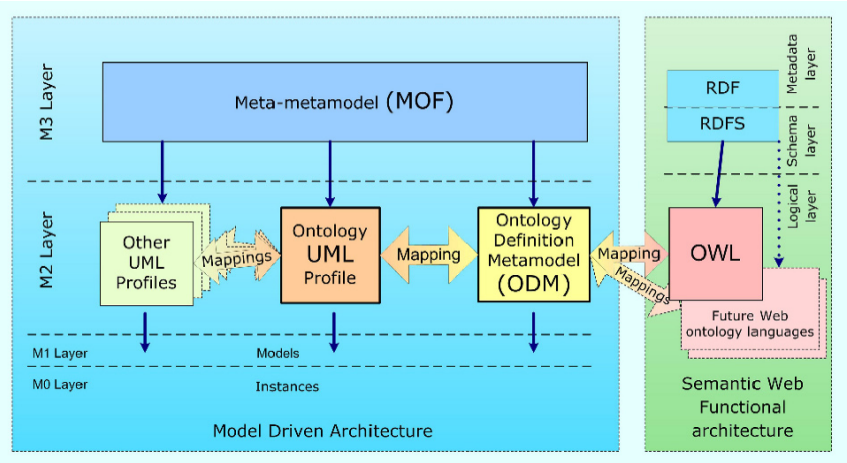


FIGURE 5.1 – Métamodèle de Définition des Ontologies (ODM) dans l'espace MDA

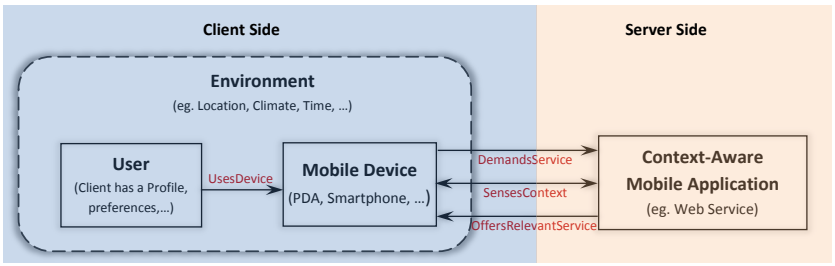


FIGURE 5.2 – Vue d'ensemble de l'application mobile sensible au contexte

5.6 Modélisation de contexte basée sur les ontologies

Dans le cadre de notre recherche guidée par les modèles, le but de cette section est de proposer une modélisation de contexte basée sur les ontologies et respectant le cadre de l'architecture MDA. Nous allons présenter ci-dessous notre modèle ontologique de contexte structuré avec la notation ODM pour capturer toutes les informations pertinentes de contexte. Mais avant de présenter ce dernier, il est important de donner d'abord, dans les deux sous-sections suivantes, une définition explicite des applications qui vont exploiter ce modèle dans leur processus de développement, et de présenter notre étude de cas sur lequel ce modèle va être instancié.

5.6.1 Applications mobiles sensibles au contexte

Dans la portée de l'informatique pervasive, une classe d'applications qui a suscité l'intérêt de plus en plus dans le milieu de la recherche est celui des applications sensibles au contexte invoqués par un utilisateur mobile (centrée sur l'utilisateur) utilisant des dispositifs mobiles. Ces applications peuvent dynamiquement, capturer des informations contextuelles afférentes à leur utilisation (comme l'emplacement de l'utilisateur, le temps et la météo, le dispositif et les activités des utilisateurs, ...) et d'en profiter pour adapter leur comportement en conséquence. D'un point de vue de mise en œuvre, les nouvelles technologies basées sur l'architecture orientée services (voir la Section 2.4.1) semblent prometteuses pour ce type d'applications mobiles.

La Figure 5.2 montre une vision simple sur l'invocation d'une application mobile sen-

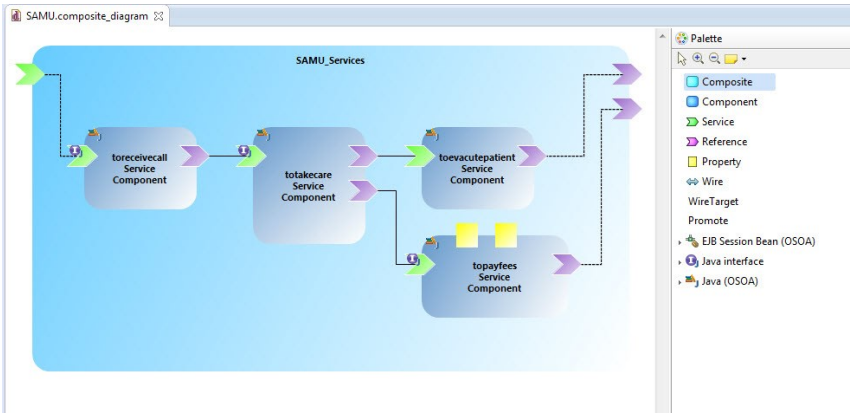


FIGURE 5.3 – Application SCA de SAMU

sible au contexte et centrée sur l'utilisateur. L'utilisateur caché derrière son dispositif mobile (côté client) demande un service ou des informations à l'application (coté serveur). Cette dernière, et avant de fournir une réponse pertinente en fonction du contexte, devrait détecter et prendre en compte toutes les informations contextuelles entourant l'utilisateur et son interaction avec l'application tels que les préférences de l'utilisateur et les informations de l'environnement dans lequel l'interaction évolue (lieu, temps, climat, ...).

5.6.2 Etude de cas : Le service SAMU

L'exemple suivant illustre les principaux services médicaux d'urgence pour les patients dans le besoin comme il fait le SAMU en France[6]. Ce dernier répond aux demandes d'aide médicale d'urgence, où les patients sont (dans la rue, à la maison, ou au travail ...), et quelle que soit la gravité de leur état (malaise, maladie, grossesse ou autre), le système évacue également des patients aux services hospitaliers les plus appropriés selon leurs cas (diabétique, hypertension, ...). Pour l'aide médicale d'urgence sur place, une ambulance est envoyée pour une éventuelle évacuation du patient à l'hôpital désigné par le médecin SAMU après avoir examiné son dossier médical. Notre application SAMU sera composée de quatre services principaux, à savoir « recevoir l'appel du patient » ; « fournir un soin pré-hospitalier » ; « évacuer le patient (si nécessaire et selon son état de santé) de sa localisation à l'hôpital spécialisé le plus proche (dans le temps le plus court possible) » ; et enfin « sélectionner le mode de paiement pour prendre en charge les frais de l'intervention médicale et de l'évacuation ». La Figure 5.3 présente une application avec ces quatre services de SAMU respectivement "toreceivecall", "totakecare", "toevacuatepatient" et "topayfees" comme des services composants avec une architecture SCA [14] qui sera détaillée par la suite.

Cette application contient les services requis pour accomplir les missions importantes de SAMU. Cependant, les informations de contexte peuvent être fournies pour contrôler et faciliter ses activités par rapport à la pratique actuelle des comportements de SAMU (par exemple, pour localiser le lieu de l'appelant au lieu de poser de nombreuses questions, qui peuvent être une problématique pour les cas graves ou pour les personnes âgées/enfants qui ne peuvent pas être en mesure de donner leurs locations exactes. Ou, pour évacuer les

6. pschitt ://fr.wikipedia.org/wiki/Service_d'aide_m%C3%A9dicale_urgente

patients dans des conditions graves dans les plus brefs délais par le chemin le plus court en vue de sauver leur vie). Pour cet objectif, notre application SAMU peut être considérée comme une application sensible au contexte pour fournir les services décrits précédemment. Les sections suivantes permettront de clarifier notre approche pour construire ce type d'applications en utilisant des mécanismes de sensibilité au contexte .

5.6.3 Modèle de contexte avec la notation ODM

La spécification ODM fournit un métamodèle OWL conforme à la spécification MOF, composé d'un ensemble de diagrammes pour représenter les différents éléments de l'ontologie (diagramme des classes OWL, diagramme des propriétés OWL, diagramme des instances OWL, ...). Aussi, et compte tenu de l'importance de UML, les développeurs ODM ont défini des correspondances entre le métamodèle UML standard et le métamodèle OWL en proposant un Profil UML d'Ontologie (OUP) focalisé sur les mécanismes d'extension UML. Le but de l'OUP est de permettre l'utilisation de la notation graphique UML standard pour développer des ontologies [54]. OUP réutilise les constructions UML quand ils ont la même sémantique que OWL, et où lorsque cela n'est pas possible, il utilise des constructions UML stéréotypes qui sont consistantes et aussi près que possible de la sémantique d'OWL [93].

Dans [26], nous avons présente notre modèle de contexte basé sur les ontologies et construit à l'aide de l'ODM. La Figure 5.4 dépeint l'instanciation de ce modèle pour notre exemple SAMU cité dans la section 5.6.2. Ce modèle est structuré en trois sous-ontologies à trois niveaux d'abstraction et de généralisation [43] : ontologie générique, ontologie de domaine et ontologie d'application qui sont de poids lourds contenant des axiomes et des contraintes pour plus d'expressivité sémantique.

1. **L'ontologie générique** : est une ontologie de haut niveau [134] qui décrit, indépendamment de n'importe quel domaine particulier, les trois principales dimensions de l'information de contexte dans un environnement mobile et pervasive [126], en l'occurrence, l'Utilisateur (`User`), le dispositif mobile (`MobileDevice`) et de l'environnement (`Environment`) et leurs relations (`usesDevice` et `evolvedIn`) :
   - Concept `User` : est une personne (ou une application, parfois) qui est l'entité centrale dans les systèmes informatiques pervasives ayant des informations personnelles comme le profil, les préférences. Il évolue dans un environnement et utilise des dispositifs mobiles pour demander des services,
   - Concept `MobileDevice` : il est utilisée par l'utilisateur pour l'accès aux différents services disponibles et aussi, il permet de recueillir les informations contextuelles sur l'environnement qui entoure l'exécution ou l'interaction utilisateur/application y compris les informations matérielles du dispositif mobile utilisé (type du dispositif, la largeur de la bande passante du réseau, ...), et
   - Concept `Environment` : c'est l'espace dans lequel l'utilisateur évolue. Il représente l'ensemble des informations entourant l'utilisateur et son dispositif mobile qui peuvent être pertinentes pour une application sensible au contexte. Il peut inclure différentes catégories d'informations de contexte telles que l'information spatiale (ville, construction, adresse), des informations temporelles (heure, date, saison) et des données climatiques (prévisions météorologiques, température, ...)

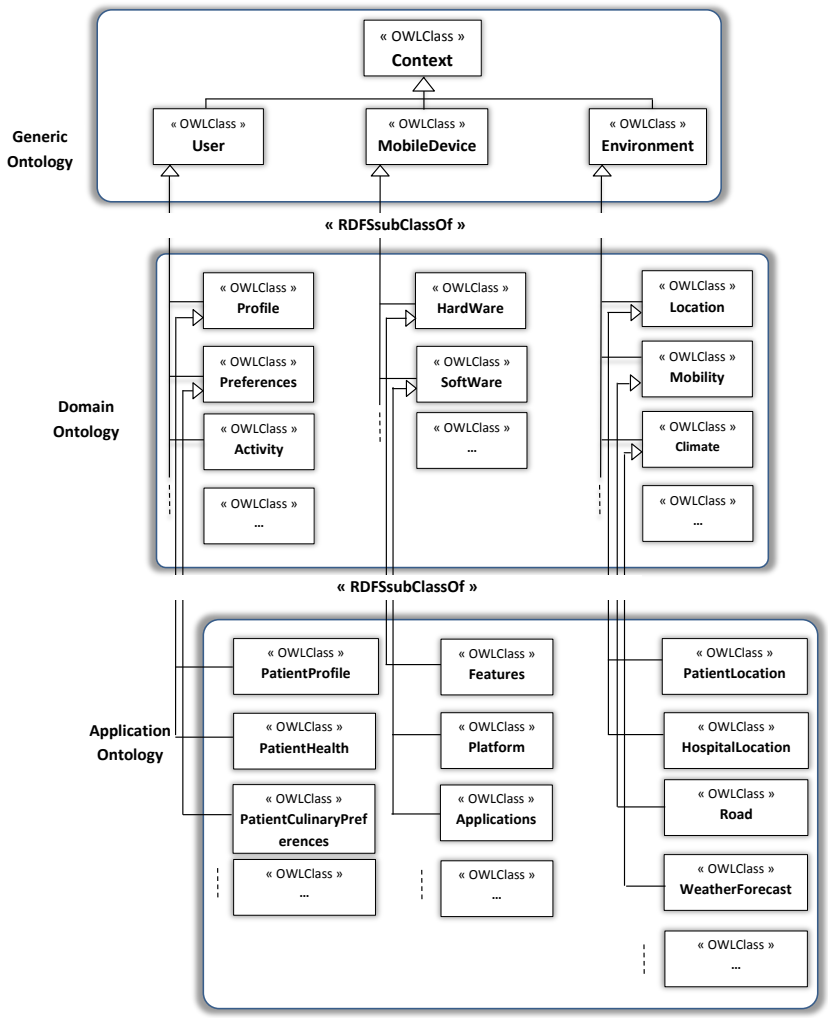


FIGURE 5.4 – Modèle de contexte basé sur les ontologies avec la notation ODM

2. **L'ontologie de domaine** : capture les éléments de contexte primaire les plus communs dans l'informatique pervasive définis dans la littérature (telles que, localisation, profil, temps, climat, activité, ...) comme il est résumé dans [133]. Chaque concept de l'ontologie de domaine doit être attaché à un modèle extensible d'ontologie spécifique de de contexte nommé ontologie d'application.
3. **L'ontologie d'application** : est une ontologie spécifique qui décrit les contextes spécifiques à la tâche pour une application spécifique (comme la météo, hôtel, et patients dans notre cas d'étude). L'ontologie d'application est conforme dans sa construction aux ontologies générique et de domaine [35].

Les informations de contexte permettant l'individualisation de ces ontologies sont fournies par deux manières différentes : par la détection (capture) ou par l'inférence. La méthode de détection est utilisée directement pour fournir de contexte basique de bas niveau, alors que, l'inférence est utilisée pour en déduire un contexte de haut niveau en utilisant des langages d'inférence à base de règles [16] pour exprimer les règles de raisonnement comme le langage SWRL [65](voir quelques exemples dans la Section suivante).

Il convient de noter, que les informations contextuelles détectées (à bas niveau) sont fournies par deux types de fournisseurs de contexte : de source locale et/ou de source à

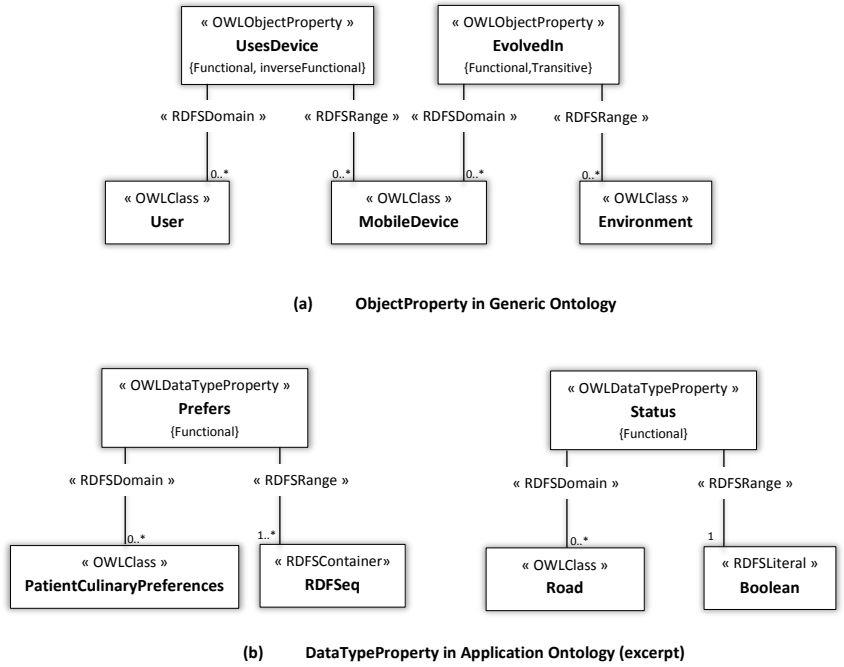


FIGURE 5.5 – Modélisation des propriétés des ontologies de contexte par ODM

distance. Le premier est pour une fourniture locale des informations de contexte et qui est disponible au coté client (mobileDevice) ou au coté serveur (application). Par exemple, le centre d'intérêt d'un participant à une conférence existe dans son profil et peut être utilisé comme information de contexte pour l'informer sur toutes les communications en fonction de son domaine de recherche. Le source de contexte à distance est utilisé pour apporter d'autres informations de contexte de l'extérieur pouvant être fournies par une tierce partie (par exemple, les services Web renvoyant la température, la probabilité de pluie, vitesse du vent , les programmes de télévision, ...).

Dans ce modèle, la Figure 5.5 (a et b) montre, respectivement, l'utilisation des relations et des attributs (les stéréotypes « ObjectProperty » et « DataTypeProperty ») à l'aide du Profil UML d'Ontologie (OUP) selon le métamodèle OWL [54].

5.6.3.1 Les règles SWRL pour la déduction du contexte de haut niveau

Le langage SWRL (Semantic Web Rule Language) [65] est basé sur la combinaison du langage OWL avec le langage RuleML (Rule Markup Language) en utilisant la logique de description [11]. SWRL comprend une syntaxe abstraite de haut niveau pour élaborer les règles. Une sémantique du modèle théorique est donnée pour fournir la signification formelle des ontologies OWL y compris les règles écrites dans cette syntaxe abstraite.

Par exemple, si la langue du participant à une conférence est le français (à partir les informations de contexte de bas niveau détectées de son profil), nous pouvons en déduire que les programmes de télévision parlant la langue française ou traduites vers elle, sont des informations de contexte de haut niveau et sont parmi ses préférences. Ceci peut être exprimé par la règle SWRL suivante :

Participant(?p) and hasLanguage(?p, "french") implies hasPreferencesTV(?p, "FrenchEmissions").

Un autre exemple, le contexte de haut niveau « ForecastWeather = isHarsh » est dérivé

de la température, la vitesse du vent ou de la probabilité de pluie en tant que contexte de bas niveau qui peut être exprimé avec la règle de SWRL. Par exemple, pour le cas de la température, la règle est comme suit :

Weather(?w) and hasTemperature(?w, ?temp) and swrlb :greaterThan(?temp, 35) and swrlb :lessThanOrEqual(?temp, 10) implies ForecastWeather(?w, "isHarsh").

Pour notre cas d'étude de SAMU, nous pouvons déduire que le pain sans sel rentre dans les préférences culinaires des hypertendus comme suit : Patient(?p) and hasChronicDisease*(?p, "BloodPressure") implies hasCulinaryPreferences**(?p, "BreadWithoutSalt").

Les règles SWRL s'exécutent par des moteurs d'inférence à base de règles tel que JESS (Java Expert System Shell) [7] [53] qui peut être facilement intégré dans un outil de développement des ontologies tel que Protégé [8] afin d'enrichir et compléter les propriétés OWL dans l'ontologie de contexte.

5.6.3.2 Un modèle de contexte satisfait les exigences tracées

Malgré la présence de plusieurs approches de modélisation de contexte, on constate qu'il n'y a toujours pas de consensus sur leurs modèles de contexte qui sont soit incomplets, ou soit pour un domaine spécifique. Le modèle de contexte devrait être général pour être utilisé par les différentes applications sensibles au contexte, spécifiques pour couvrir les principales entités contextuelles proposées dans la littérature, souple pour permettre une extension et à prendre en compte les nouvelles entités spécifiques à un domaine d'application donné. Pour répondre à ces exigences fixées aussi dans l'introduction de ce chapitre, nous avons proposé un modèle de contexte à base d'ontologie. Or, l'ontologie de niveau générique donne une abstraction de haut niveau en proposant des entités générales pour être utilisées par différentes applications mobiles centrées sur l'utilisateur sensibles au contexte. Par contre, l'ontologie de domaine permet de couvrir toutes les principales entités contextuelles proposées dans la littérature de la modélisation de contexte, et l'ontologie d'application précise le contexte entourant l'espace de l'application.

Les ontologies de domaine et d'application sont suffisamment flexibles en permettant de couvrir toute entité contextuelle d'une situation pervasive autour de l'utilisateur. Comme elles permettent de prendre en compte toute nouvelle entité liée au contexte d'une manière générale ou à l'application sensible au contexte de manière spécifique. En outre, par des capacités de raisonnement sur ces ontologies avec l'utilisation des règles SWRL, le modèle peut déduire d'autres informations contextuelles qui pourraient compléter le contexte décrivant une situation d'interaction utilisateur/application.

5.7 Conclusion

Dans un environnement mobile et ubiquitaire, les ontologies sont connues dans la littérature comme un formalisme très puissant pour modéliser le contexte avec une expressivité sémantique très riche et extensible.

Dans ce chapitre, nous avons présenté un modèle extensible de contexte basé sur les ontologies et structuré avec la spécification ODM. Ce modèle est composé de trois niveaux d'ontologies, une ontologie générique, une ontologie de domaine pour les concepts

7. www.jessrules.com
8. http://protege.stanford.edu/

de contexte populaires et communs, et une ontologie d'application pour une description précise du contexte dans un espace clos.

Cependant, l'ODM offre un profil UML pour représenter les ontologies (OUP) sous forme d'un modèle UML facile à intégrer dans un processus de développement dirigée par les modèles des applications mobiles sensibles au contexte. L'ODM permet d'allier les deux paradigmes de l'ingénierie des logiciels, à savoir, l'ingénierie pilotée par les modèles (MDE) avec son standard MDA et les ontologies, et de combler l'écart entre eux.

En outre, les capacités de raisonnement offertes par les ontologies permettent à notre modèle d'utiliser les règles SWRL (Semantic Web Rule Language) [65], et cela pour déduire d'autres informations contextuelles de haut niveau qui pourraient compléter le contexte de bas niveau décrivant une situation d'interaction utilisateur/application.

Dans le chapitre suivant, nous allons voir l'exploitation de ce modèle de contexte par les mécanismes d'adaptation pour développer les applications sensibles au contexte à base de services.

Chapitre 6

MDD et AOM pour le Développement des applications Sensibles au Contexte

6.1 Introduction

La deuxième étape dans le développement des applications sensibles au contexte consiste à réaliser l'adaptation ou la sensibilité au contexte qui est déjà capturé et modélisé dans le modèle ontologique proposé dans le chapitre précédent. Le terme de sensibilité au contexte introduit initialement par Schilit et al. [108] a connu, aussi, de nombreuse définitions dans la littérature. Dans le présent travail, nous basons sur la définition de Pascoe dans [101] qui définit la sensibilité au contexte comme « *la capacité d'un programme ou d'un dispositif pour détecter ou capturer différents états de son environnement et de lui-même* ».

Se référant à cette définition, une application sensible au contexte doit avoir la capacité de collecter des informations de contexte et d'adapter son comportement en fonction des changements de contexte au moment de l'exécution. Dey et al. estiment qu' « *un système est sensible au contexte s'il utilise le contexte pour fournir des informations et/ou services pertinents à l'utilisateur, où la pertinence dépend de la tâche de l'utilisateur* » [46].

Le présent chapitre vise à proposer une approche de développement dirigée par les modèles (MDD) en se basant sur l'architecture MDA[27], dans laquelle, nous allons exploiter les avantages de développement dirigé par les modèles (MDD) et la modélisation orientée par aspect (AOM) [8].

6.2 Mécanismes de sensibilité au contexte

Les applications sensibles au contexte doivent être en mesure de recueillir des informations de contexte et rapidement adapter leur comportement aux changements de contexte. L'adaptation au contexte requiert l'identification de l'ensemble des informations de contexte qui influe sur l'application, et la spécification de ses différents comportements en fonction de cette identification de contexte. Dans la littérature, nous trouvons trois importantes relations stéréotypées de l'adaptation contextuelle [116, 73, 56] représentant les différents mécanismes pour que les applications mettent en œuvre la sensibilité au contexte, à savoir :

ContextBinding : associe une liaison entre les éléments de contexte et les éléments

d'application capables d'être sensibles au contexte et à prendre leurs valeurs directement à partir de ces éléments de contexte. Il permet la recherche d'information pour les utilisateurs en fonction de l'information disponible sur le contexte. Dans notre exemple SAMU (voir la Section 5.6.2), les services composants «totakecare» et «toevacuatepatient» nécessitent de connaitre l'emplacement du patient comme paramètre d'entrée pour l'envoi d'une ambulance afin de lui présenter les premiers secours et l'évacuation à l'hôpital en cas de nécessité. Grâce à l'association ContextBinding, au lieu d'obtenir du patient l'information de son emplacement, la valeur de ce paramètre peut être directement déduit de l'appel reçu du Smartphone de patient en utilisant le Système de Positionnement Global (GPS).

BehaviourAdaptation : ce mécanisme permet de modifier alternativement le comportement de l'application en sélectionnant un parmi plusieurs comportements disponibles (ou opérations de la même fonctionnalité) conformément à la situation contextuelle actuelle. Certaines contraintes sur le contexte doivent être respectés afin de sélectionner le comportement approprié. Formellement, ils sont modélisés comme prédicats (à savoir, les expressions booléennes) qui peuvent être exprimées en OCL (Object Constraint Language) [92] de UML. Le comportement du service «topayfees» peut prendre, selon le mode de paiement, les opérations "payWithCash", "payWithCreditCard" ou "payWithSocialSecurityCard" pour les personnes démunies/assurés en fonction de profil du patient.

ContextTriggering : Elle permet l'adaptation des résultats d'exécution d'une application en fonction de contexte. Le mécanisme ContextTriggering repose sur deux parties ; un ensemble de contraintes contextuelles et un ensemble d'actions. La partie de l'action sera effectuée que si toutes les contraintes sont satisfaites. Le tri, le filtrage (par exemple, dans de le cas de nombreux résultats) ou la conversion (par exemple, pour les cas de métrique et de devise) sont des exemples d'actions. Pour le service «toevacuatepatient», une action de tri peut être appliquée pour afficher uniquement l'itinéraire le plus court à l'hôpital général le plus proche, ou même plus, à l'hôpital spécialisé le plus proche pour un patient en fonction de sa maladie chronique (comme une contrainte) si elle est fournie dans son profil.

Dans ce travail, nous considérons que les mécanismes `ContextBinding`, `BehaviourAdaptation` et `ContextTriggering` peuvent être spécifiées et appliquées pour les paramètres d'entrée, le comportement de l'application et les paramètres de sortie respectivement. De plus, leur combinaison peut concevoir la logique d'adaptation au contexte pour toute situation contextuelle d'une application. Le mécanisme `ContextBinding` peut être réalisé par des opérations « `Setter` » afin de fournir des valeurs d'entrée comme dans les méthodes Setter pour les classes Java, et par «`adaptation`» pour spécifier les comportements ou les actions à appliquer sur les résultats des services pour les deux autres mécanismes.

De plus, ces mécanismes peuvent proposer une solution à l'adaptation structurelle et architecturale sensible au contexte soulevée par Ayed et Berbères dans [10]. Par exemple, le changement du comportement de l'application (composée d'une ou plusieurs opérations) par un autre (une ou plusieurs opérations) induit l'adaptation architecturale.

Maintenant à propos de la conception et le développement d'applications (ou services) sensibles au contexte, il y a deux issues importantes[113]. La première concerne la fourniture des informations de contexte. D'autre part, la deuxième issue est sur la méthodologie

de développement à l'aide des mécanismes de sensibilité au contexte pour construire des applications capables d'adapter leurs comportements en fonction des informations fournies de contexte, et cela sans intervention explicite de l'utilisateur. Étant donnée que le premier point a été détaillé dans le chapitre précédent, dans ce qui suit nous allons présenter notre approche pour donner une solution au deuxième issue.

6.3 Architecture conforme à MDA pour développer des applications sensibles au contexte

En utilisant les informations de contexte formalisé dans le modèle ontologique présenté précédemment (voir Section 5.6.3), cette section présente l'architecture de notre approche pour concevoir et développer des applications sensibles au contexte à base de services.

6.3.1 L'architecture de l'approche proposée

L'approche qui sera détaillée ci-après se concentre sur la considération que la sensibilité au contexte peut être gérée à travers la notion de variabilité introduite initialement dans les approches des logiciels des lignes de produits (SPL : Software Product Line) [62, 85, 77, 57]. En utilisant cette notion de variabilité, l'application sensible au contexte peut être considérée comme une application avec plusieurs points de variation qui refluent leurs éléments sensibles au contexte. Pour chaque point de variation, des multiples variantes sont associées afin de sélectionner l'une d'eux en fonction du contexte actuel.

Dans la pratique, le principe de la variabilité est pris en charge par la modélisation orientée aspects (AOM) au niveau modèle [87] et par la programmation orientée aspect (AOP) au niveau code [49]. Dans l'espace MDD [109, 28, 5], nous sommes intéressés par l'utilisation d'AOM dans lequel les concepts de point de variation et la variante sont mappés aux point d'insertion ou point de jonction (*joinpoint*) et greffon (*advice*) respectivement. Les différents joinpoints (points de variation) d'une application sont sélectionnés par une expression des points d'action ou points de coupe (pointcut) sans être dénombrées.

L'AOM est employée dans notre travail pour permettre :

1. La modélisation des différentes variantes (ou adaptations sensibles au contexte) dans des modèles d'aspect appelés ContextAspect (à détailler dans la section 6.4) en utilisant les mécanismes de sensibilité au contexte cités auparavant. Ces modèles seront considérés comme des préoccupations transversales fonctionnels [15, 2], et
2. Le tissage de ces modèles d'aspect dans le modèle de la logique métier d'une application à la conception et pendant l'exécution.

En outre, cette approche est basée sur le standard MDA [91] avec ses quatre couches de modélisation. MDA est construit sur l'idée de la séparation de la conception de la logique métier de l'application des détails techniques de la plateforme d'exécution. La logique métier, indépendamment de tout détail technique, est modélisé dans le modèle indépendant de la plateforme (PIM). Alors que cette logique métier y compris les détails techniques nécessaires d'une plateforme technologique sont modélisés dans le modèle spécifique à la plateforme (PSM). Ce dernier est généré à partir du modèle PIM en utilisant un processus de transformation sur la base de la mise en correspondance trouvée entre les éléments des

métamodèles de PIM et PSM. De cette façon et pour le même modèle PIM, nombreux PSM peuvent être produits pour diverses plateformes technologiques.

Pour cet objectif, MDA utilise un ensemble de normes technologiques, à savoir, le langage de modélisation unifié (UML) [90], MOF [94], ODM [93], XMI (XML Meta - Data Interchange) [97], QVT (Query/View/Transformation) [95] et OCL (Object Constraint Language) [92]. Ensemble pour unifier et simplifier la modélisation, la conception, la mise en œuvre et l'intégration du système [19]. Dans notre cas, l'architecture MDA est utilisée pour rendre l'intégration simple de la logique de l'adaptation sensible au contexte dans la logique métier au niveau modèle et de développer les applications sensibles au contexte sur une plateforme d'architecture orientée services.

Notre approche est composée d'une architecture fondée sur MDA et son processus de développement comme une méthodologie pour concevoir les applications sensibles au contexte à base de services. L'architecture est représentée dans la Figure 6.1 dont le métamétamodèle MOF constitue sa couche M3.

La couche M2 contient cinq métamodèles. Le métamodèle PIM (UML ou Ecore, par exemple) capture la conception de la logique métier de l'application (le modèle de base). Le métamodèle ODM capture les informations du contexte dans des modèles basés sur l'ontologie selon le profil UML Ontology [54]. Le métamodèle « ContextAspect » (à définir dans la section suivante) sert à modéliser les différents modèles d'aspects sensibles au contexte (ou les adaptations sensibles au contexte) qui peuvent être tissés dans l'application pour ajouter la logique de sensibilité au contexte. Le métamodèle « Weaving » contient les principales constructions qui permettent de tisser des modèles ContextAspect dans le PIM. Enfin, le métamodèle PSM spécifie la plateforme cible de l'application résultant de ce tissage. Dans cette thèse, la « Service Component Architecture (SCA) » est choisie comme une plateforme pour développer les applications sensibles au contexte en tant que composants de services distribués (autres architectures de services peuvent être utilisés aussi bien).

La couche M1 est composée d'un ensemble de modèles (conformes à leurs métamodèles correspondants décrits) et des modèles de transformation, le modèle PIM, le modèle de contexte basé sur des ontologies et un ou plusieurs modèles de ContextAspect englobant différentes adaptations sensibles au contexte pour les différentes situations contextuelles d'une application. Pour effectuer la sensibilité au contexte, les modèles de ContextAspect sont injectés dans le modèle PIM pour fournir le CPIM (Contextual PIM), le modèle qui constitue la première application sensible au contexte au moment de la conception, indépendamment de tout cadre/plateforme d'exécution. Le modèle CPIM est transformé en un modèle contextuel PSM (CPSM) afin de prendre en compte les détails techniques de SCA dans l'application résultante. Ensuite, le CPSM est converti en codes d'application nécessaires pour être exécuté sur la plateforme FraSCAti (qui sera détaillée ci-dessous).

La couche M0 comprend les implémentations des modèles précédents afin d'obtenir l'exécution et l'adaptation dynamique des applications sensibles au contexte à base de services sur la plateforme FraSCAti.

Ci-après, nous présentons le métamodèle d'aspect sensible au contexte (« ContextAspect ») et le processus de développement de l'approche proposée sur la base de l'architecture représentée dans la Figure 6.1.

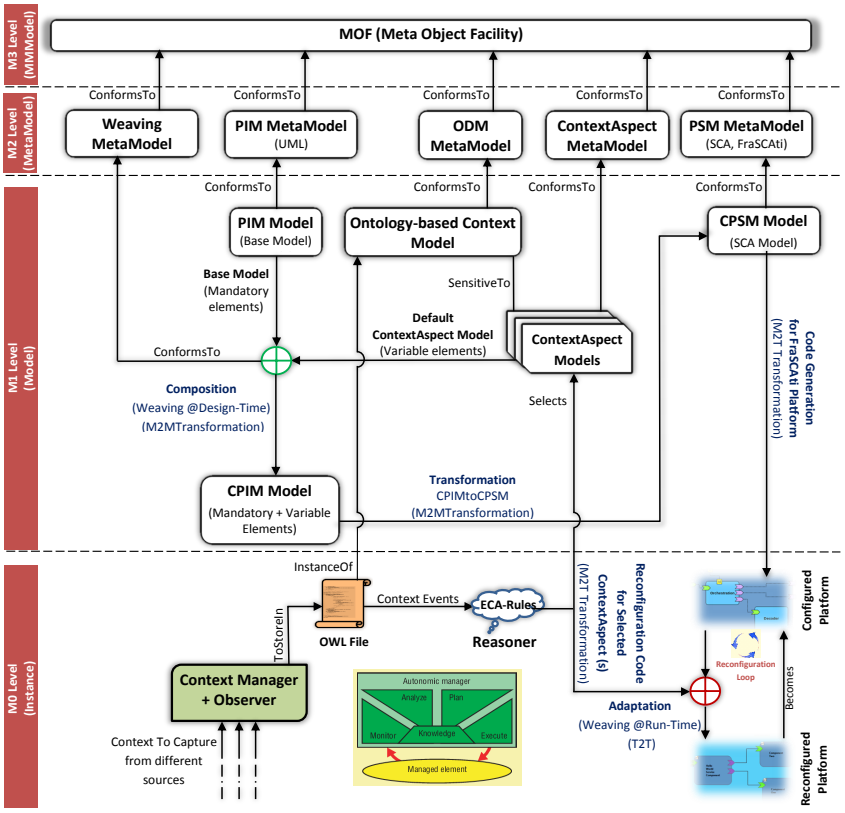


FIGURE 6.1 – Architecture MDA pour le Développement des Applications Sensibles au Contexte

6.4 Le Metamodèle ContextAspect

« ContextAspect » représente le métamodèle le plus important utilisé dans l'architecture proposée (voir la Figure 6.1). Il vise à définir et préciser où et comment l'adaptation sensible au contexte aura lieu. La Figure 6.2 montre la structure de ce métamodèle dont ContextAspect (la classe « `ContextAspect` » classe) doit contenir les éléments de la modélisation des trois parties : d'aspect, de contexte et de sensibilité au contexte, détaillées comme suit :

6.4.1 Les éléments du modèle Aspect

Comme dans AspectJ [76], un aspect comprend des greffons (*advice*) et des points de coupe (*pointcut*). Le greffon décrit le comportement transversal (le code) à tisser dans les points de jonction (*join points*), tandis que le point de coupe définit les expressions pour détecter les *join points*. Un point de jonction est le lieu où le comportement transversal émerge dans le programme de base (ou l'application noyau). Les greffons d'un aspect contient trois parties logiques : Avant(*Before*), Autour(*Arround*) et Après (*After*). La partie *Before* du greffon (respectivement la partie *After*) permet l'introduction du comportement transversal avant (respectivement après) le point de joinpoint. Cependant, la partie *Arround* définit, quant à lui, un bloc d'instructions qui tourne autour d'un *join point*, comme il peut éventuellement remplacer complètement une méthode dans ce *join point*.

Dans la littérature, comme il y a des extensions d'aspect définies pour les langages

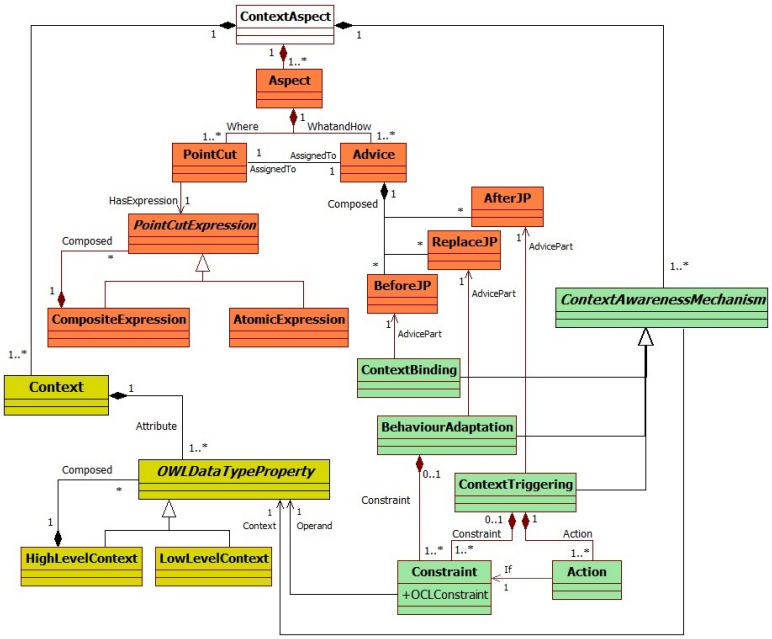


FIGURE 6.2 – Le Métamodèle ContextAspect

de programmation, il existe des extensions d'aspect définies pour les notations de modélisation dans laquelle plusieurs métamodèles sont proposées afin de formaliser les aspects [131]. Cependant, aucun d'entre eux est considéré comme un modèle de référence pour les aspects. En outre, les propositions existantes ne sont pas le plus proche de ce que nous allons faire au sujet de la manipulation de la sensibilité au contexte afin d'être réutilisés, aucun d'entre eux ne précise les différentes parties de greffon (avant, autour, et après) dans leurs modèles, et qui représentent le point clé le plus important dans la construction de notre approche. Dans la Figure 6.2, nous proposons le nôtre en mettant l'accent sur les concepts de paradigme orientée aspect.

La classe « `Aspect`» est composée d'un ou plusieurs classes « `PointCut` » et « `Advice` » qui sont liés par une association d'affectation entre eux. Chaque « `PointCut` » a une classe d'expression « `PointCutExpression` » qui peut être primitive « `AtomicExpression` » ou composite « `CompositeExpression` » à l'aide des opérateurs logiques. Les deux types d'expression peuvent être exprimées avec OCL [92]. « `PointCutExpression` » permet le « `pointcut` » de sélectionner (par correspondance) les différents *join points* (classe « `JoinPoint` ») qui définissent les points de jonction dans le modèle de base, où les aspects peuvent introduire des éléments de modèle supplémentaires et qui sont définis dans « `Advice` ». Cette dernière classe est composée des classes « `BeforeJP` », « `ReplaceJP` » et « `AfterJP` » qui représentent les parties de l'« `Advice` » à tisser dans les trois endroits possibles dans le modèle de base : avant, après ou remplacer le point de jonction (*Join Point*). l'« `Advice` » peut contenir un ou plusieurs éléments de greffon comme, il ne peut contenir aucun élément (en cas de détissage).

6.4.2 Les éléments du modèle Contexte

Le fonctionnement des applications sensibles au contexte nécessite la collecte de toutes les informations de contexte d'exécution formalisé dans un modèle communément compris.

Nous sommes convaincus que le formalisme des ontologies représenté avec la spécification ODM répond bien à cette exigence. Le modèle de contexte basé sur les ontologies, proposée précédemment dans le chapitre 5, stocke les informations de contexte (la classe «Context») dans les attributs ou `DatatypeProperty` qui sont reconnus par «`OWLDataTypeProperty`» selon la spécification ODM [54], dans lequel les informations de contexte, capturées depuis des sources matérielles et/ou logicielles, sont modélisés en tant que contexte de bas niveau dans la classe «`LowLevelContext`» , et celles déduites, en utilisant des règles SWRL, sont modélisés comme un contexte de haut niveau dans la classe «`HighLevelContext`».

6.4.3 Les éléments du modèle Sensibilité au Contexte

Les mécanismes de sensibilité au contexte (la classe abstraite «`ContextAwarenessMechanism`») mentionnées dans la section 6.2 sont représentés par les relations «`ContextBinding`», «`BehaviourAdaptation`» et «`ContextTriggering`» entre la classe de contexte «`OWLDataTypePropoer` et les classes «`BeforeJP`», «`ReplaceJP`» et «`AfterJP`» du greffon de l'aspect respectivement.

La classe « `ContextBinding` » permet de lier les éléments du greffon « `BeforeJP` » aux éléments ontologiques correspondants dans le modèle contexte avant l'élément sensible au contexte (c.-à-d, point de variation ou *join point*). Par exemple, toutes les valeurs d'entrée d'un service capturées à partir du contexte sont liées aux greffons «`BeforeJP`» avec une relation «`ContextBinding`», comme dans le cas de la localisation des patients pour des services d'intervention de SAMU (Section 5.6.2) .

«`ContextTriggering`» permet de déclencher, après l'élément sensible au contexte, une ou plusieurs actions (la classe «`Action`») si une ou plusieurs contraintes de contexte (la classe « `Constraint`») sont évaluées à vrai. La classe «`Action`» dans le greffon «`AfterJP`» est pour spécifier le traitement à appliquer sur le résultat de l'application telles que les actions de filtrage ou de la conversion. Les contraintes sont établies comme des prédicats booléennes sur les valeurs de contexte dans lequel les attributs de contexte («OWLDataTypeProperty» classe) ont lieu des opérandes. Par exemple, comme une action, c'est la sélection du chemin le plus court à l'hôpital afin d'évacuer le patient si son état de santé le nécessite (comme une contrainte).

Cependant, «`BehaviourAdaptation`» implique de remplacer complètement la logique métier de l'élément sensible au contexte par un autre encapsulé dans le greffon «`ReplaceJP`». Ceci est permis si une ou plusieurs contraintes sur le contexte (classe «`Constraint`») sont vérifiées. A titre d'exemple, considérons la commutation entre le paiement en espèces ou par carte de crédit afin de couvrir les couts des soins de santé.

6.5 Processus de développement

La présente approche commence par la conception de l'application initiale sensible au contexte au niveau du modèle (M1). Une fois créée et déployée sur une plateforme orientée services, l'application peut alors être soumis à plusieurs adaptations dynamiques. Cette approche s'articule sur un même métamodèle d'aspect à la conception et à l'exécution utilisé pour construire les différentes adaptations sensibles au contexte en utilisant des techniques de tissage. Il effectue la conception et l'adaptation des applications sensibles au

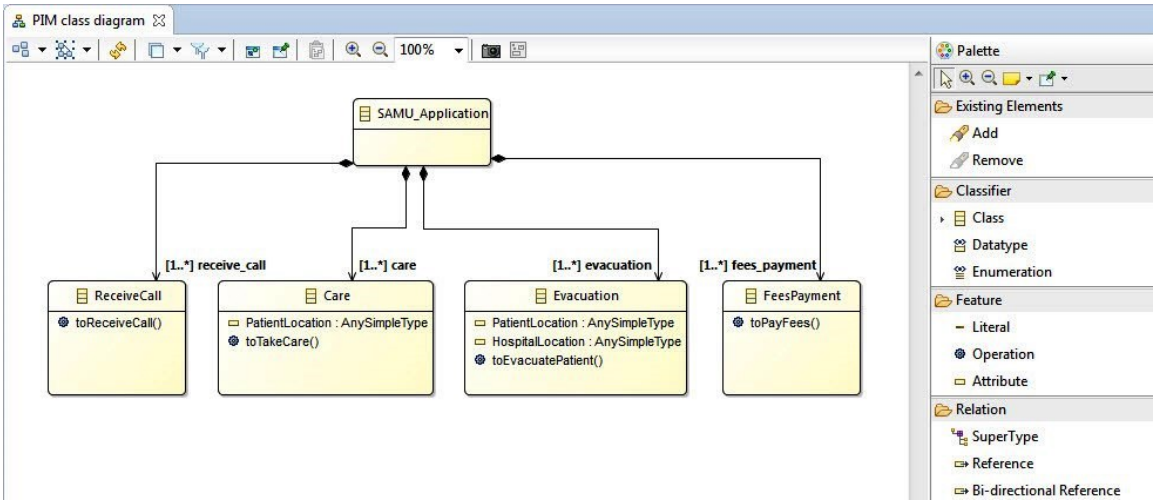


Figure 6.3: Le Platform Independent Model (PIM) de l'Application SAMU

contexte à base de services en quatre phases de développement, à savoir, la modélisation, la composition, la transformation et l'adaptation.

6.5.1 Phase de modélisation

Après la modélisation du contexte par le modèle basé sur l'ontologie (Section 5.6), le processus de développement passe à modéliser le modèle PIM contenant la logique métier de l'application et un ou plusieurs modèles de « ContextAspect » qui encapsulent les différents comportements que l'application peut prendre dans les différentes situations de contexte. Notons que PIM et « ContextAspects » sont des modèles basés sur UML qui peuvent être modélisé en utilisant Eclipse Modeling Framework (EMF) [1]. Actuellement, EMF [119] est le standard *de facto* pour la modélisation et la génération de code afin de construire des outils et des applications basées sur des modèles.

Considérant que le modèle d'application sensible au contexte est composé à partir des éléments obligatoires et d'autres variables, cette phase se concentre à modéliser ces éléments à l'aide des classes de diagrammes UML. Les éléments obligatoires sont modélisés dans une unique logique métier appelée modèle de base qui représente les points communs de l'application et qui feront partie de tous les modèles d'application sensibles au contexte construites après. Respectant la terminologie MDA, le modèle logique métier représente le modèle indépendant de la plateforme (PIM) dans notre architecture (Figure 6.1).

En ce qui concerne les éléments variables, leurs lieux et leurs variantes correspondantes (ou comportements) sont définis dans des modèles de ContextAspect conformes au métamodèle « ContextAspect » (Figure 6.2) pour leurs éventuels tissages dans le modèle de base. La variante appropriée est sélectionnée en fonction de la situation de contexte courante, et une nouvelle application sensible au contexte peut être construite automatiquement par le tissage du modèle ContextAspect sélectionné dans le modèle de la logique métier.

Cependant, l'un des modèles ContextAspect sera désigné comme ContextAspect par défaut à tisser dans le PIM. De préférence, la désignation de ContextAspect par défaut est donnée en fonction du comportement le plus utilisé par l'application sensible au contexte

1. `http://www.eclipse.org/modeling/emf/`

dans la plupart des situations contextuelles.

Nous soutenons cette désignation pour deux raisons :

1. Le choix d'un modèle ContextAspect particulier comme modèle par défaut rend facile les transformations CPIM vers CPSM vers code (à définir, ci-dessous, dans la Section 6.5.3) qui nécessitent d'être achevées par l'intervention humaine en l'absence de technique automatique et complète de transformation.
2. En outre, le ContextAspect par défaut représente l'état de mode sans échec d'une application sensible au contexte en cas de multiples et rapides fluctuations de contexte. Le contexte dans lequel ces applications évoluent est très dynamique et peut potentiellement changer plus rapidement que l'application adaptative [85]. Par exemple, les fluctuations de la bande passante (comme information de contexte) sont plus rapides que le processus d'adaptation, ce qui pourrait rendre l'application toujours sensible au contexte dans une configuration n'est pas adaptable au contexte courant. Dans ce cas, il est préférable que l'application passe à l'état d'équilibre représentée ici par le modèle ContextAspect par défaut avant de passer à nouveau à la première situation contextuelle stable avec le processus d'adaptation (à détailler dans la Sous-section 6.5.4)

La Figure 5.3 représente le PIM de notre exemple d'application SAMU (voir la Section 5.6.2), y compris certains services avec leurs opérations. Les opérations toTakeCare, toEvacuatePatient et toPayFees sont des éléments variables et peuvent être modifiés en fonction du contexte. En revanche, l'opération toReceiveCall est un élément obligatoire, qui apparait dans toute application sensible au contexte produite sans être influencée par le changement de contexte. Les Figures 6.4 et 6.5 sont deux modèles ContextAspect qui présentent les adaptations sensibles au contexte pour deux cas possibles de l'intervention de SAMU à savoir et respectivement : fournir les premiers soins médicaux et l'évacuation du patient à l'hôpital le plus proche pour le premier cas, et pour le second, de ne fournir que les premiers soins médicaux sans la nécessité d'évacuer le patient.

Dans la Figure 6.4, le paquetage "FirstMedicalCarewithPatientEvacuation" contient trois sous-modèles d'aspects sensibles au contexte (Care, Evacuation and FeesPayment). L'aspect «Care» a un pointcut (CarePointcut) exprimé pour trouver le service "toTakeCare" dans l'application comme un point de jonction afin d'appliquer ce qui est décrit dans le greffon (CareAdvice). Dans ce cas, il s'agit de tisser une autre opération appelée "SetPatientLocation" avant le point de jonction sélectionné («BeforeJP»). Le rôle de cette opération est de capturer de manière automatique et directe l'emplacement du patient à partir des informations de contexte disponibles dans l'*OWL DatatypeProperty* "PatientLocation" dans la classe OWL "Location". L'association établie entre l'aspect «Care» et l'information de contexte «Location» est autorisée par une relation ContextBinding.

L'aspect d'évacuation est pour le tissage des trois opérations de greffon dans le service toEvacuatePatient à savoir : SetDeparture et setDestination pour prendre des valeurs à partie les informations de contexte PatientLocation et HospitalLocation respectivement, par deux relations ContextBinding. En outre, si le docteur veut évacuer le patient (PatientHealth.DoctorOpinion="EvacuationNeeded"), la troisième opération "toSelectItinerary" sera appliquée sur les itinéraires d'évacuation («AfterJP») afin de sélectionner le plus court à l'hôpital le plus proche de l'emplacement du patient en utilisant la relation ContextTriggering.

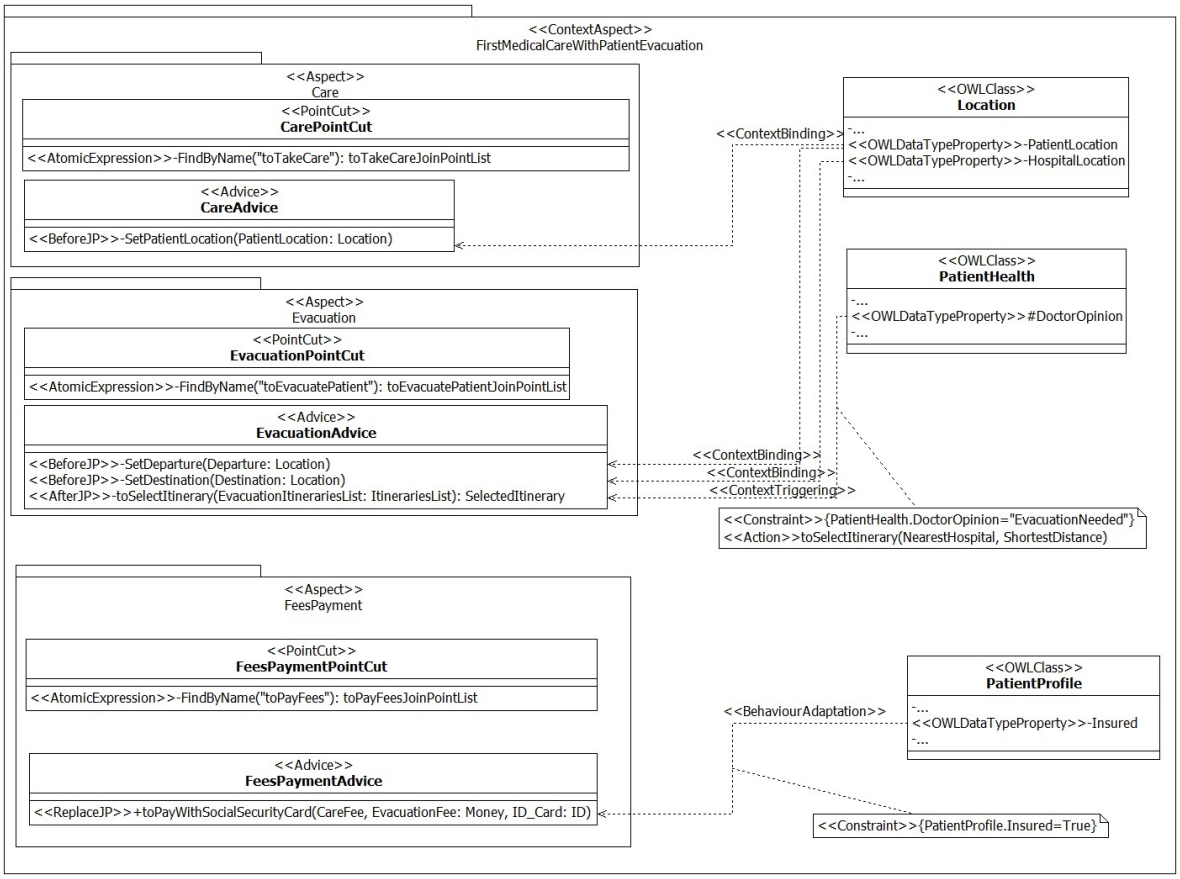


Figure 6.4: Le modèle ContextAspect «First Medical Care with Patient Evacuation»

Considérant que, l'aspect FeesPayment illustre comment choisir le mode de paiement («ReplaceJP») de l'intervention du SAMU par la sécurité sociale (toPayWithSocialSecurityCard) si le patient est assuré (PatientProfile.Insured=True) en utilisant la relation BehaviourAdaptation.

Le modèle ContextAspect FirstMedicalCarewithoutPatientEvacuation illustré dans la Figure 6.5 se compose des deux services Care et Evacuation. Ici, le service d'évacuation n'est pas pour le patient, mais juste pour un retour à vide de l'ambulance à son centre SAMU (SAMUCentreLocation) sans qu'il soit nécessaire de sélectionner un chemin plus court.

Le premier modèle FirstMedicalCarewithPatientEvacuation est considéré comme un modèle ContextAspect par défaut et qui sera composé dans un premier temps avec le modèle de PIM pour fournir l'application initiale sensible au contexte (comme il sera détaillé dans la phase suivante).

6.5.2 Phase de composition

Dans la modélisation orientée aspect et en suivant une approche asymétrique, la composition commence toujours par le premier modèle étant le modèle initial qui peut être fusionné avec le modèle d'aspect à un moment donné [104]. Cette phase est pour composer les deux modèles de PIM et un ContextAspect choisi en fonction du contexte (premièrement ou pour l'état de mode sans échec, c'est le modèle de ContextAspect par défaut). La composition peut être réalisée par une technique de tissage d'aspects [66, 100]. Elle consiste à tisser le modèle ContextAspect dans le PIM et retourner un seul modèle appelé

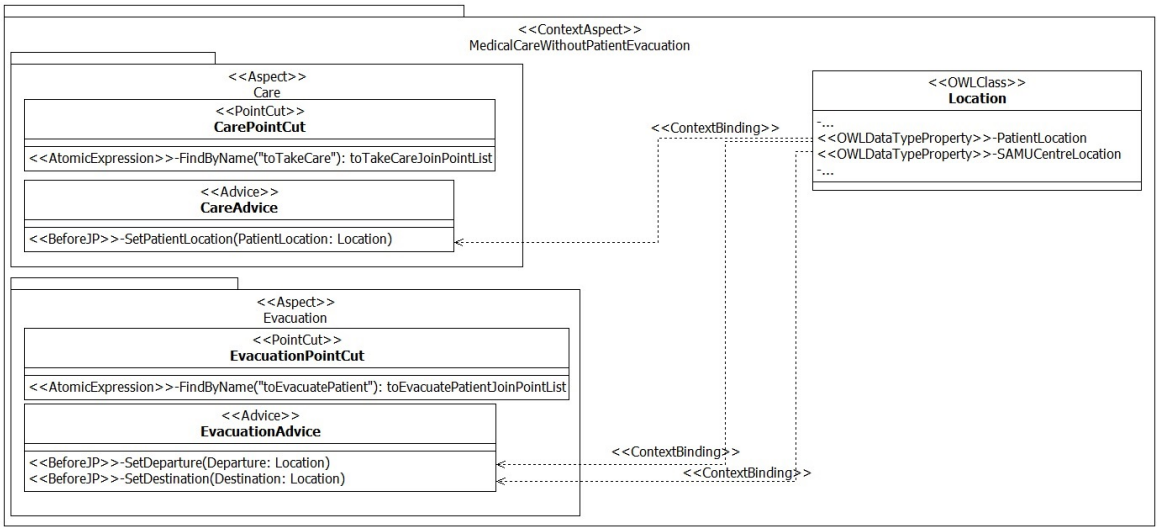


Figure 6.5: Le modèle ContextAspect «First Medical Care without Patient Evacuation»

le modèle contextuel indépendant de la plateforme (CPIM) qui contient les éléments du modèle de la logique métier et ceux des greffons d'aspect du modèle ContextAspect.

Le tissage dans l'AOM est réalisé par un modèle de transformation de modèles basé sur l'expression de point de coupe (aussi appelé, tisserand AOM) [85]. Dans notre cas, nous utilisons un tisserand AOM qui est un modèle de transformation M2M basé sur *pointcut*, et qui repose sur le métamodèle de tissage (voir la Figure 6.6) et applique l'algorithme de tissage suivant :

- **Instruction 1** : Les tisserand itère pour chaque point de coupe dans chaque aspect sensible au contexte appartient au modèle ContextAspect, choisi pour tisser (ou détisser) ses modifications dans le modèle PIM en 2-passages :
- **Instruction 2 (opération de correspondance ou d'appariement)** : En premier passage, le tisserand procède à l'appariement basé sur le nom (la classe « FindByName ») en tant que mode d'appariement (la classe « MatchingOperations ») pour sélectionner tous les points de jonction ou les éléments PIM (la classe « PIMElement ») définies par l'expression de point de coupe associée (la méthode « getMatchingElements »). Suite à cette opération de correspondance, tous les endroits touchés par l'aspect sensible au contexte sont identifiés et une liste de points de jonction est retournée.
- **Instruction 3 (opération de tissage)** : Le tisserand, en seconde passe, procède à tisser (ou détisser) les modifications insérées dans le greffon (la classe « WeavableElement ») dans les points de jonction correspondants. Selon le type d'élément de greffon, ces modifications seront introduites avant, après ou remplacer les endroits (opérations) impactés afin de produire des éléments de CPIM (la classe « CPIMElement »).

Le processus de tissage se termine lorsque toutes les modifications de tous les aspects sensibles au contexte sont effectuées. A ce stade, toutes les variantes de modèle ContextAspect ont été composées avec le modèle PIM. Le modèle qui en résulte (CPIM) est une représentation complète d'une application sensible au contexte en prenant les constructions de comportement d'une situation de contexte défini par le modèle ContextAspect. Toutefois, cette opération de tissage (Instruction 3) doit respecter certaines règles pour

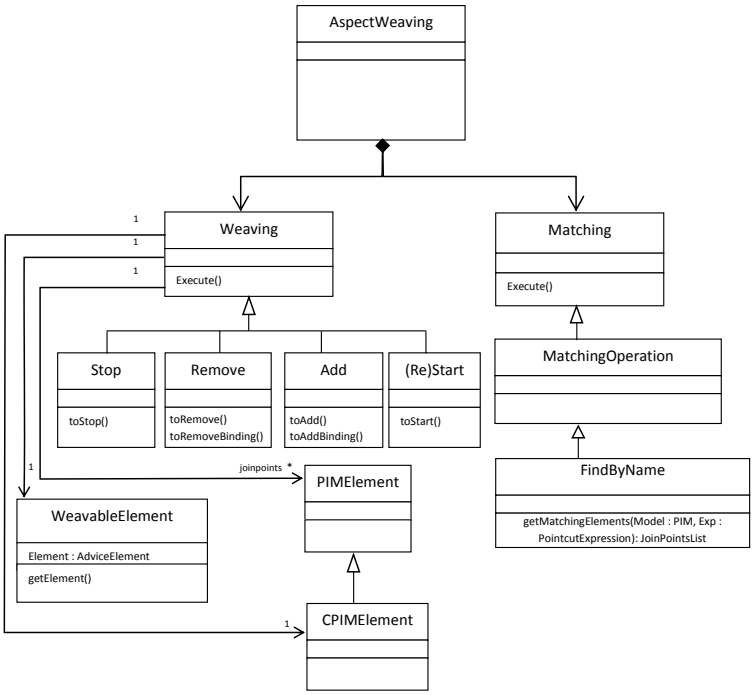


FIGURE 6.6 – Le Metamodèle de Tissage

éviter tout conflit pourrait se produire au cours de la composition de PIM avec les modèles ContextAspect, à savoir :

- **Règle 1** : Pour tisser un greffon d'aspect dans un point de jonction, le tisserand passe par trois ou quatre étapes ; **l'arrêt de l'opération impactée** (avec la classe «Stop» dans métamodèle de tissage), **détissage** (opération optionnelle) du greffon déjà tissé (avec la classe « Remove »), le **tissage** du nouveau élément de greffon (la classe «Add») et le **redémarrage** de l'élément composé (la classe « Restart »). En particulier, l'étape de détissage sera ignorée pour construire la première application sensible au contexte en composant le PIM avec le modèle ContextAspect ou bien, dans le cas où plusieurs greffons devraient être tissés dans le même point de jonction (voir règle 3)
- **Règle 2** : Si la partie greffon d'un modèle aspect ne contient aucun élément à tisser, alors il s'agit d'une opération de détissage seulement (détissage = tissage sans éléments de greffon). Également dans la cas « ReplaceJP », le détissage consiste à retourner à l'élément PIM initial afin d'effectuer la mise en correspondance pour d'autres modèles de ContextAspect. Par exemple, l'élément PIM topayfees sera retourné pour chaque commutation entre les différents ContextAspects de modes de paiement (payWithCash, payWithCreditCard ou payWithSocialSecurityCard).
- **Règle 3** : Le tisserand peut tisser plus d'un greffon de nombreux aspects sensibles au contexte dans des différents points de jonction. Comme plusieurs greffons d'un ou plusieurs aspects peuvent être intervenus (tissés) à un seul point de jonction (lorsque des situations contextuelles diverses sont impliquées, par exemple). Ceci exige le respect de la priorité d'exécution (règles 4 et 5) et de garantir la coordination de fonctionnement (règles 6 et 7) qui sont pour but d'assurer le bon fonctionnement des aspects et éviter tout comportement incohérent de l'application.
- **Règle 4** : Pour les parties de greffon définis dans le même aspect sensible au

contexte, la priorité est établie par leur ordre de définition. Par exemple, dans la Figure 6.4 (EvacuationAdvice), la trace d'exécution serait : «BeforeJP» SetDeparture, «BeforeJP» setDestination et «AfterJP» toSelectItinerary. «BeforeJP» SetDeparture a la priorité sur «BeforeJP» setDestination parce qu'il a été défini avant, et ainsi de suite ...

- **Règle 5** : Lorsque deux parties de greffon ou plus (d'un aspect ou plus) sont tissés au même point de jonction, la priorité d'exécution est spécifié en suivant l'ordre de tissage des parties de greffon ; le premier tissé dans le point de jonction sera d'abord exécuté. Bien que cet ordre d'exécution est partiellement différent des technologies typiques d'aspect comme AspectJ, il est plus approprié pour développer les applications sensibles au contexte avec notre approche qui est basée sur le tissage de modèles d'aspect pour chaque changement de contexte, notamment pour permettre l'adaptation dynamique de ces applications. Dans certains cas, l'adaptation dynamique nécessite le tissage d'un aspect supplémentaire (contient un greffon après («AfterJP»)) sans le détissage du premier (qui a contenu aussi un greffon après («AfterJP»)) avec lequel l'application est en cours d'exécution. Par exemple, une application peut d'abord indiquer, pour un visiteur, les attractions intérieures et extérieures dans une ville (sélectionnés en fonction de la localisation des visiteurs), puis, si le temps devient dure (augmentation des précipitations, par exemple), cette application doit s'adapter pour montrer seulement les attractions à l'intérieur par une une action de filtrage (comme un greffon après («AfterJP»)) qui sera exécuté après le premier greffon. Dans tels cas, l'ordre d'exécution proposé facilite l'adaptation dynamique.
- **Règle 6** : La règle par défaut pour la coordination de fonctionnement entre les parties des aspects est de suivre l'ordre général dans l'exécution de parties de greffons. Cette exécution doit commencer par la partie «BeforeJP» du greffon et se termine par la partie «AfterJP» en passant par la partie «ReplaceJP».
- **Règle 7**: La coordination entre plusieurs greffons tissés dans le même point de jonction est réalisée en considérant les trois cas de fonctionnement suivants :

  1. Partie «BeforeJP» : en utilisant la conjonction logique avec l'opérateur "AND" afin d'obtenir des informations précises par exemple, ou en utilisant la disjonction logique avec l'opérateur "OR" pour la consolidation de l'information. Selon notre exemple SAMU, l'emplacement du patient peut être capturé par la consolidation des informations de contexte à partir les informations GPS, les dossiers des clients de la compagnie de téléphone, ainsi que de la surveillance vidéo donnant un emplacement initial.
  2. Partie «ReplaceJP» : en utilisant la conjonction logique avec l'opérateur "AND" afin d'appliquer plusieurs comportements, ou disjonction logique avec l'opérateur "OR" pour les comportements alternatifs. Un patient peut diviser le paiement de ses frais de soins médicaux en espèce et un paiement par carte de crédit en appliquant les deux méthodes de comportements correspondants.
  3. Partie «AfterJP» : en utilisant la disjonction logique avec l'opérateur "OR" afin d'appliquer de nombreuses actions sur les résultats de manière séparée (filtrage ou de sélection, par exemple), ou en utilisant la conjonction logique par l'opérateur "AND" avec la sémantique d'appliquer la dernière action tissée

sur les résultats issus de l'application de la précédente. Par exemple, un résultat peut d'abord être filtrée et ensuite converti de métrique aux mesures impériales.

Les opérateurs "AND" et "OR" peuvent être spécifiés par le concepteur à travers les règles des ECA (voir ci-dessous dans la Sous-section 6.5.4) pour déterminer les aspects à tisser avec la possibilité de coordonner plusieurs aspects en utilisant les opérateurs logiques booléens.

Dans la pratique, la phase de composition (correspondance et tissage) peut être mise en œuvre par un modèle de langage de transformation dédié tels que ATL (dans le style déclaratif) [69] ou Kermeta (dans le style impératif) [67, 88]. Cependant, dans un champ d'application mobile et omniprésent, nous préconisons l'utilisation des langages de programmation portables sur les données dans un format ouvert afin d'assurer l'interopérabilité entre les deux côtés de l'application (serveur et client). Dans notre cas, nous avons utilisé le langage Java avec le parseur DOM XML [2] afin de traiter les modèles PIM et ContextAspect dans leur format sérialisé XMI (XML Metadata Interchange) [97] fourni par EMF. Java pratiquement est pris en charge par la plupart des systèmes d'exploitation, plateformes logicielles et les navigateurs Web populaires, et il est même construit, ou en cours de construction, dans les appareils électroniques de grand public (tels que les boitiers décodeurs de télévision, Smartphone et Tablet PC). Le support Java devient omniprésent. Cependant, XMI est actuellement la norme ouverte de données la plus fréquemment prise en charge.

La Figure 6.7 montre (en haut) les fichiers téléchargés XMI de PIM (Figure 6.3) et de ContextAspect par défaut (Figure 6.4) , et le CPIM (fichier XMI en bas) découlant de la composition de PIM et des trois aspects sensibles au contexte (Care, Evacuation and FeesPayment) du modèle ContextAspect FirstMedicalCarewithPatientEvacuation.

Pour l'aspect Care, cette composition est réalisée par le tissage de l'opération SetPatientLocation avant le point de jonction toTakeCare. Pour l'opération toEvacuatePatient de l'aspect Evacuation, les opérations de SetDeparture et SetDestination sont tissés avant, et toSelectItinerary après. D'autre part, l'opération de toPayWithSocialSecurityCard, est tissée comme un comportement de toPayFees.

6.5.3 Phase de transformation

Le modèle CPIM, résultant de la phase précédente, est toujours indépendant de toute technologie ou plateforme. Suite à un processus de transformation classique de MDD, le même modèle composé peut être transformé en nombreuses plateformes orientées services et qui supportent le paradigme orienté aspect. Dans ce travail, nous avons opté pour l'architecture SCA (*Service Component Architecture)* [14] et sa mise en œuvre par FraSCAti [111] en tant que plateforme cible pour nos applications. SCA est un ensemble de spécifications pour développer et déployer des applications orientées service réparties en utilisant les principes d'architecture orientée services (SOA) et de l'ingénierie logicielle à base de composants (CBSE). SCA est pris en charge par la plateforme FraSCAti pour permettre la reconfiguration dynamique des applications en cours d'exécution. FraSCAti fournit une API d'exécution permettant l'introspection dynamique et la modification d'une application SCA. Cette caractéristique est d'une importance particulière pour la conception et

2. `http://www.jdom.org/`

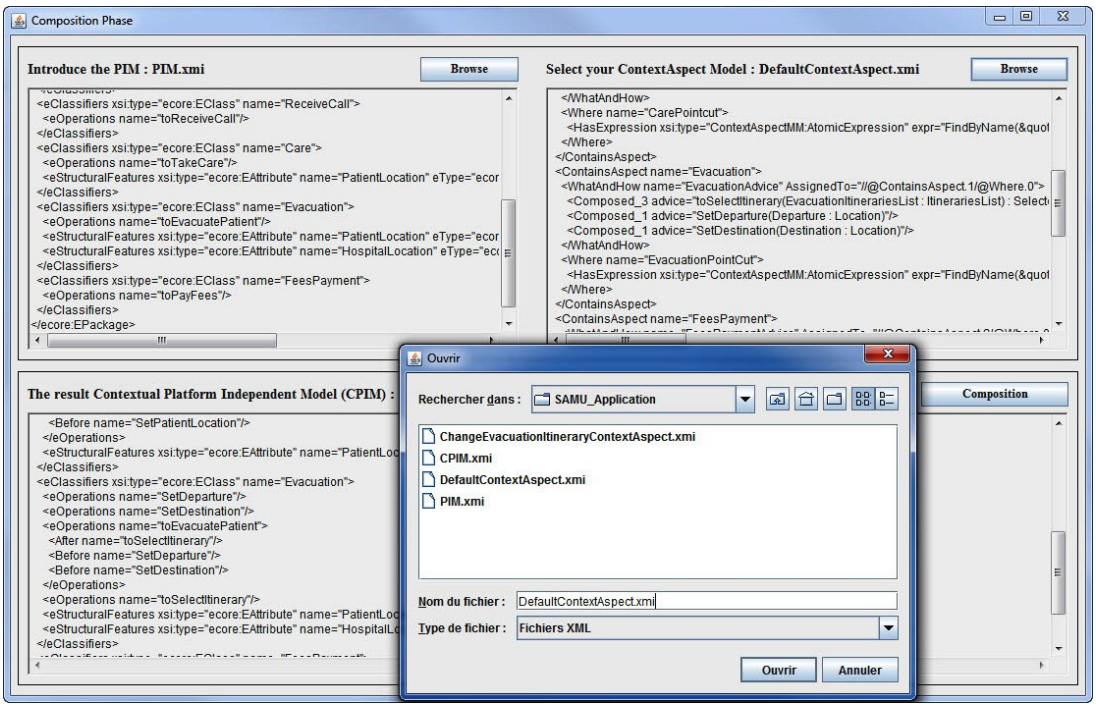


Figure 6.7: La phase de Composition des modèles PIM et ContextAspect

la mise en œuvre des applications SCA agiles, telles que les applications autonomes [75] et les applications sensibles au contexte [112] comme dans notre cas.

Premièrement et par une transformation M2M mise en œuvre par un langage de transformation tel que ATL [69] ou en utilisant un langage de programmation d'ordinateur à usage général (tel que, Java), le CPIM est converti en un modèle contextuel spécifique à la plateforme (CPSM) lié à un modèle SCA. La conversion est autorisée par l'utilisation des règles de transformation sur la base des correspondances trouvées entre les éléments du métamodèle source de CPIM (métamodèle UML) [96]) et le métamodèle cible de CPSM (métamodèle SCA [3]).

Le tableau 6.1 montre quelques correspondances utilisés dans cette transformation. Nous supposons que chaque point de variation du modèle CPIM (ou opération sensible au contexte) sera transformée en une composite SCA contenant les différents éléments de greffon tissés en tant que composants. De sorte que les greffons «BeforeJP» et «AfterJP» sont représentées par l'ajout d'un `subComponent` et `superComponent` [42], respectivement, attachés au composant de l'opération impactée. L'exécution de cette dernière est effectuée après l'exécution du `subComponent` et avant l'exécution de `superComponent`. Alors que, le greffon « ReplaceJP» consiste à remplacer ce composant d'opération par un autre.

Deuxièmement, le modèle de CPSM obtenu sera soumis à autre une transformation de modèle vers texte (M2T) pour générer le code adapté à la plateforme cible d'exécution. Pour atteindre une telle transformation M2T, de nombreux outils disponibles pourraient être utilisés, tels que Kermeta Modèle Emitter [4] ou MOFScript [5].

Pour ce travail, la phase de transformation, comme la précédente, est gérée par le traitement des fichiers XMI sérialisés avec Java et DOM API pour générer du code FraSCAti

3. `http://wiki.eclipse.org/SCA/Components/SCA_Meta_Model`
4. `http://www.kermeta.org/mdk/ket`
5. `http://www.eclipse.org/gmt/mofscript/`

Elements UML ou ContextAspect	Element SCA Equivalent
Class	Composite
Operation	Component
Attribute	Property
Input Parameter	Reference
Output Parameter	Service
Relationship	Wire
«ContextBinding» Relationship	Wire
«ContextTriggering» Relationship	Wire
Context-Aware Operation (expressed in «PointCut» Expression)	Component as a SubComposite
«BeforeJP» Advice	Component as a SubComponent
«AfterJP» Advice	Component as a SuperComponent
«ReplaceJP» Advice	Component

Table 6.1: Quelques Mappages entre les métamodèles UML, ContextAspect et SCA

(fichier composite pour l'architecture des composants et des fichiers Java pour chaque implémentation composante). L'omniprésence de Java et la fiabilité du DOM font de la combinaison Java/DOM un outil garanti pour manipuler les fichiers XMI (quelle que soit la taille et le contenu) et pour assurer l'évolutivité et l'efficacité dans les applications à grande échelle. D'autres outils dédiés tels que Eclipse ATL pourraient être utilisés pour les transformations de modèles en utilisant les correspondances trouvées entre les métamodèles source et cible. Ces outils ont une syntaxe limitée et simple qui pourrait diminuer considérablement leur utilisation pour le développement d'applications dans un environnement ubiquitaire flexible tel que le cas pour des applications sensibles au contexte.

Heureusement, tous les éléments SCA (certains d'entre eux sont dans le tableau 6.1) trouvent leurs artéfacts dans la plateforme FraSCAti [112] avec les mêmes noms, ce qui facilite la tâche de cette phase (pour ce travail) et permet de considérer le modèle CPSM comme un fichier composite de d'une architecture des composants. Toutefois, certains éléments spécifiques à la technologie FraSCAti devraient être intégrées manuellement, si nécessaire, dans ce fichier composite telles que les spécifications de liaison [112]. En outre, les mises en œuvre de composants sont à la charge des développeurs pour la raison d'absence des techniques de transformation complètes et automatiques assurant la conversion intégrale du modèle en code.

Notre application basée sur Java permet ces transformations dans lesquelles le CPIM d'exemple SAMU (Sous-section 6.5.2) est transformée à un modèle d'application composite SCA (ou FraSCAti) contenant quatre composants services sous forme des sous-composites, à savoir : ReceiveCall, Care, Evacuation et FeesPayment. Chaque sous-composite contient un ou plusieurs composants liés entre eux par des interfaces de référence/service. La Figure 6.8 montre ces sous-composites générés et affiche le contenu de composite Evacuation.

Les composants et les comportements d'interfaces invoqués dans chaque sous-composite doivent être mises en œuvre (ou réutilisées, si elles sont disponibles) et ajoutés dans le code composite généré en FraSCAti, qui est similaire à un langage de description d'as-

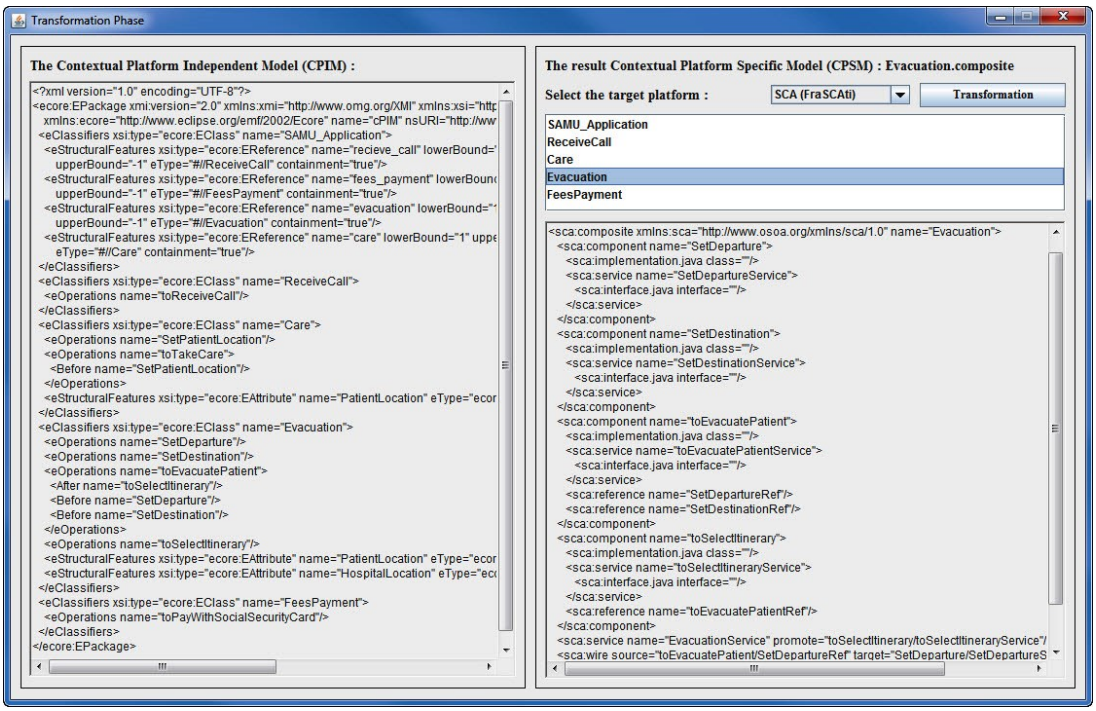


Figure 6.8: Phase de Transformation pour générer un modèle d'application SCA

semblage d'une composite de SCA prêt à être exécuté sur la plateforme FraSCAti comme il est montré dans la Figure 6.9 pour le fichier Evacuation.composite.

Par souci de simplicité, nous nous concentrons sur l'une des sous-composites avec ses composants services dans le reste du présent document. La Figure 6.10 représente le diagramme d'architecture SCA de l'application « Evacuation », initialisée selon le langage de description d'assemblage ci-dessus (Figure 6.9). Cette architecture est composée de composant toEvacuatePatient avec trois d'autres composants autour de lui ; les sous-composants SetDeparture et SetDestination (un résultat de la correspondances «BeforeJP»), et le supercomposant toSelectItinerary (suite à la correspondance «AfterJP»).

Maintenant, l'application d'évacuation sensible au contexte peut fonctionner et agir sans l'intervention humaine en déduisant le départ de l'itinéraire à partir de l'emplacement de l'appel du patient en utilisant la technologie GPS, et la destination à l'hôpital sera également déduit d'après les informations sur la maladie chronique trouvée dans le profil de santé du patient (en utilisant un service tiers, par exemple, un service Web pour fournir la liste des hôpitaux spécialisés) ou donné par un médecin (si nécessaire). Ces deux paramètres contextuels aideraient le composant service pour sélectionner l'itinéraire le plus approprié avec la distance la plus courte (et le temps, parfois) parmi plusieurs itinéraires existants.

6.5.4 Phase d'adaptation

Le code généré FraSCAti, prêt à être exécuté, représente la configuration de l'application sensible au contexte à base de services avec le modèle ContextAspect par défaut

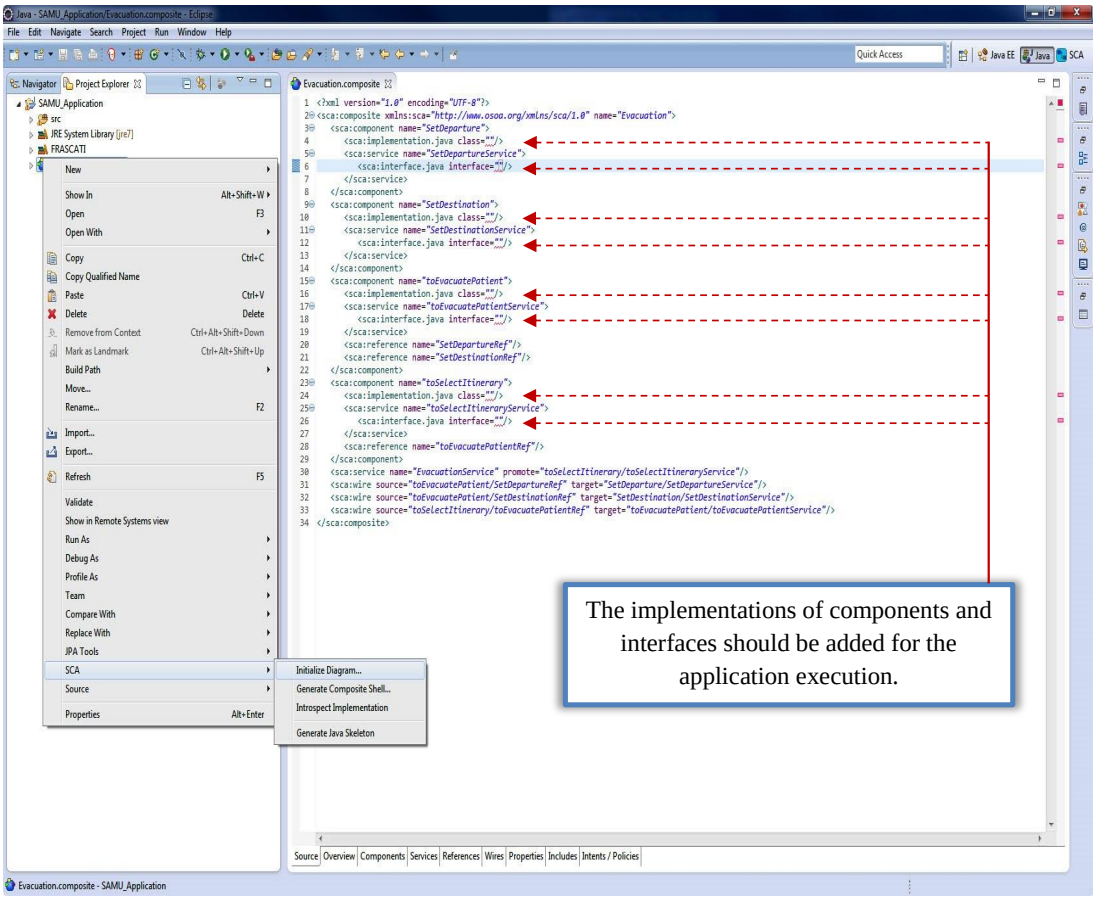


Figure 6.9: Le Langage de Description de l'Assemblage généré de Evacuation.composite dans Eclipse

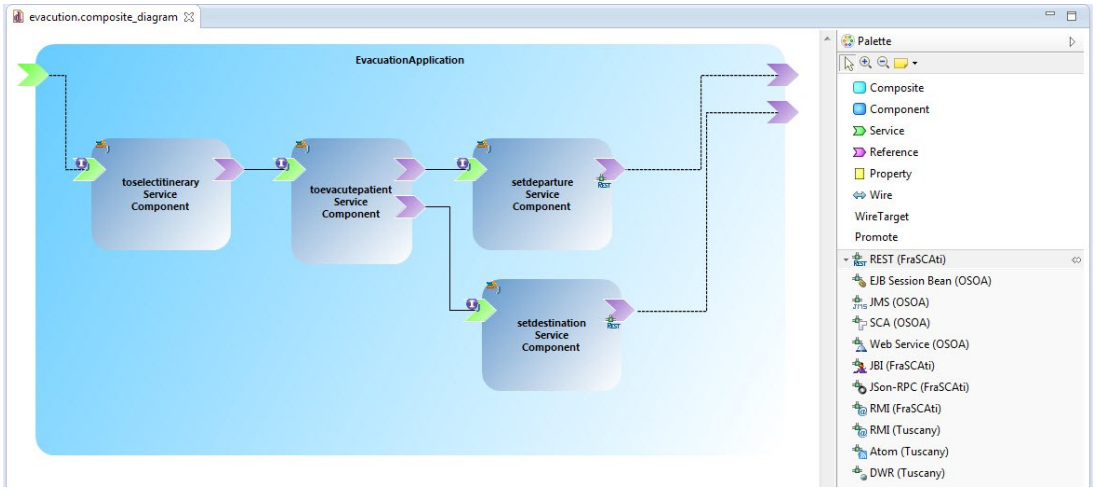


Figure 6.10: Architecture à base de composants de l'application SCA «Evacuation»

de la situation de contexte par défaut. Pendant l'exécution, les informations de contexte pourraient être changées en permanence à d'autres situations. Par conséquent, l'application en train d'exécution doit être adaptée à toute nouvelle situation de contexte par une reconfiguration au moment de l'exécution.

Dans la plateforme FraSCAti, les composants SCA basés sur Java sont en même temps conforme au SCA et conforme à Fractal [31]. Grâce à cette propriété, tous les composants de FraSCAti bénéficient des propriétés du couplage faible et de la plateforme agnostique de SCA, aussi des propriétés hiérarchiques et de réflexion de Fractal qui permettent l'application d'être reconfigurée dynamiquement [111]. Bobrow et al. [22] ont défini la réflexion comme : « *La capacité d'un programme à manipuler comme données quelque chose représentant l'état du programme lui-même pendant sa propre exécution. Il y a deux aspects d'une telle manipulation :* ***introspection*** *et* ***intercession****. L'introspection est la capacité d'un programme d'observer et donc raisonner au sujet de son propre état. L'intercession est la capacité d'un programme à modifier son propre état d'exécution ou de changer son interprétation* ». Fractal utilise la réflexion pour contrôler le cycle de vie des composants, et fournie un ensemble d'opérations permettant le démarrage, l'arrêt et la modification de l'architecture au moment de l'exécution.

En exploitant ce principe de réflexion pour réaliser l'adaptation dynamique dans notre sujet, l'application sensible au contexte peut être considérée comme un système autonome avec une boucle de contrôle MAPE-K (surveillance ou observation, analyse, planification, exécution) [75, 79], comme il indiqué dans l'architecture proposée (Figure 6.1, niveau M0) :

1. **Surveillance** : Le gestionnaire de contexte équipé d'un module observateur peut détecter tout changement dans le contexte acquis par des capteurs logiciels (services Web, ...) ou matériels (appareil photo, Camera, GPS, ...). L'observateur peut être réalisé par le système complexe de traitement des évènements (CEP) [81]. Nous considérons que la technologie CEP pourrait en permanence contrôler et traiter la confluence et la fluctuation du contexte afin de le raffiner et de le stocker dans un fichier OWL individualisée selon le modèle ontologique de contexte proposé dans la section 5.6. Le raffinement du contexte peut être effectuée en suivant les phases de traitement du contexte de Baldauf et al. [12].
2. **Analyse** : Les nouvelles valeurs de contexte seront analysés par un raisonneur qui est un moteur d'inférence basée sur les règles ECA (Evènement-Condition-Action) [99]. La syntaxe générale de la règle ECA est « sur l'**Evènement** si la **Condition** faire **Action** ».
   - La partie **Evènement** spécifie le signal qui déclenche l'invocation de la règle. Dans notre cas, l'évènement est le changement de contexte détectée par l'observateur.
   - La partie **Condition** est un test logique que, si satisfait, elle provoque une action à effectuer. Elle est une ou plusieurs contraintes sur les valeurs de contexte.
   - La partie **Action** se compose des mises à jour ou des invocations sur les données locales. Ici, elle implique la sélection des adaptations nécessaires encapsulées dans les modèles ContextAspect. Les règles de l'ECA sont utilisées pour sélectionner le modèle de ContextAspect (un ou plusieurs) qui convient avec la situation de contexte spécifique pour être tissé dans le modèle d'application.

   Toutefois, un contexte particulier peut déclencher plusieurs règles ayant rempli les

conditions avec les mêmes valeurs de contexte. Cela pourrait provoquer des actions contradictoires. Pour éviter ce conflit, nous suggérons d'ajouter une contrainte supplémentaire comme un nombre de priorité d'exécution et la syntaxe de la règle ECA devient « sur l'**Evénement** si la **Condition** avec haute **Priorité** faire **Action**». En outre, ces règles peuvent être établies à partir les contraintes de la classe « Constraint » conçues dans les modèles de ContextAspects.

3. **Planification** : Le modèle ContextAspect sélectionné sera converti par une transformation M2T pour générer les codes FPATH et FScript [42] nécessaires pour reconfigurer l'application en cours d'exécution sur la plateforme FraSCAti. Les expressions de point de coupe de ce modèle sont transformés en code FPath. FPath est un langage de requête pour naviguer dans les architectures à base de Fractal . Il facilite la navigation dans les applications de composants et permet aux développeurs de définir des requêtes qui recherchent des éléments de l'architecture de l'application en cours d'exécution et qui répondent à certains critères. Cependant, le greffon est transformé en code FScript qui est un langage de script dédié à la reconfiguration architecturale d'applications sensibles au contexte à base de Fractal [100]. Généralement, une reconfiguration (ou adaptation dynamique) se compose de deux étapes principales :
   - Trouver les endroits sensibles au contexte appariés par le point de coupe à travers le code FPath, et
   - Tisser les modifications exprimées dans le greffon et codés par FScript.

   Le tissage des modifications sera planifié dans des actions en conformité avec le métamodèle de la Figure 6.6 et sera réalisé par le même algorithme en respectant les règles de la sous section 6.5.2. Dans l'ensemble, pour chaque endroit touché par le code FPath, ce tissage doit passer par les actions suivantes :
   - Arrêter des composants de FraSCAti en cours d'exécution (ou composite),
   - Enlever tous les composants et les relations (ou câbles) du modèle ContexAspect déjà tissé (optionnel, s'il y en a),
   - Ajouter des composants et des câbles pour tisser le script du nouveau élément de greffon, et
   - Redémarrer les composants arrêtés .

   En revanche et dans le même temps, une adaptation au niveau du modèle doit être effectué. Encore une fois, la phase de composition (Sous-section 6.5.2) est appelée considérant que le modèle de fonctionnement CPIM devient le nouveau modèle PIM pour avoir accepté d'être composé avec le modèle ContextAspect sélectionné dans cette phase d'adaptation. Cette adaptation dynamique au niveau modèle peut être activé à l'aide de la notion des modèles à l'exécution (modèles@run-time) [20, 86, 67] pour établir un lien de causalité entre le modèle et l'application en cours, et à combler l'écart (dans les deux sens) qui pourrait existe entre l'abstraction et la réalité [85]. En outre, cette adaptation (au niveau modèle) peut servir à valider l'application reconfigurable sensible au contexte s'il y aura des techniques de transformation et des outils permettant une dynamique (au moment de l'exécution) et complète transformation du CPIM au code en passant par le CPSM sans aucune intervention du développeur.

4. **Exécution** : Avec les actions planifiées, l'application peut, maintenant, subir une

auto-reconfiguration au moment de l'exécution. FraSCAti soutient le tissage à l'exécution et permet donc la réalisation de ces actions de reconfiguration [112].

6.6 Conclusion

La puissance du développement dirigé par les modèles (MDD) réside dans l'utilisation des modèles et des transformations pour concevoir les applications à haut niveau, indépendamment, de tout détail technique de plateforme. Tandis que, la modélisation orientée aspect (AOM) est utile pour séparer les différentes adaptations sensibles au contexte dans des modèles d'aspects transversaux prêts à être tissés dans le modèle d'application à la conception et pendant l'exécution en fonction de l'évolution du contexte au fil du temps.

Dans ce chapitre, nous avons exploité les avantages de la combinaison de MDD et AOM pour l'ingénierie des applications sensibles au contexte à base de services. Nous avons proposé une architecture logicielle pour construire ces applications et une méthodologie de développement sur la base de cette architecture. La méthodologie proposée est dirigée par les modèles et comprend quatre phases (modélisation, composition, transformation et adaptation) qui agissent en conformité avec la technologie MDA. Les trois premières phases ont été bien détaillées par le déroulement de notre exemple de SAMU. Cependant, l'importance de la dernière phase nous exige de l'expliciter dans un chapitre entier.

Chapitre 7

Scénario d'Adaptation Dynamique

7.1 Introduction

L'adaptation dynamique représente la principale caractéristique des applications sensibles au contexte. Ces dernières devraient prendre en considération le changement de contexte au fil du temps ; pendant la conception et avant d'être lancées, et même, après, à l'exécution par l'adaptation de leurs comportements. La plupart des travaux de la littérature ont ignoré l'aspect de cette adaptation en temps d'exécution. Dans notre travail, nous avons introduit cette caractéristique dans une phase entière du cycle de développement des applications sensibles au contexte à base de services comme nous avons vu dans le chapitre précédent.

Le but de ce chapitre est de présenter en détail cette phase d'adaptation dynamique en expliquant le processus de sa boucle de contrôle MAPE-K [79] sur un scénario de notre exemple de SAMU.

7.2 Processus d'Adaptation Dynamique

Pour ce travail, nous avons mis en œuvre toutes les phases de la méthodologie proposée sur notre étude de cas de SAMU (Sous-Section 5.6.2) [1]. Et pour bien illustrer le processus dynamique de la dernière phase d'adaptation du chapitre précédent, nous allons dérouler un scénario simple sur notre exemple [25, 27] (un autre exemple est discuté dans [24]). Nous considérons que le modèle ContextAspect FirstMedicalCarewithPatientEvacuation (Figure 6.4) est tissé dans l'application, ce qui permet au service d'évacuation d'être invoqué, et que l'itinéraire est sélectionné et affiché pour évacuer un patient, le reste de cette section montrera comment ce service va se comporter lorsqu'un évènement contextuel indique que l'itinéraire choisi est impossible (en raison d'un accident de voiture, des travaux imprévus, ou le trafic lourd).

7.2.1 Étape de surveillance du changement de contexte

Le gestionnaire du contexte (Figure 6.1, Niveau M0) doit être équipé de la description du fichier OWL du contexte initialement créé selon le modèle de contexte basée les ontologies (voir la Section 5.6). Son module observateur est mis en œuvre en utilisant une

1. voir `http://contextaspectmethodology.leadtech.dz/`

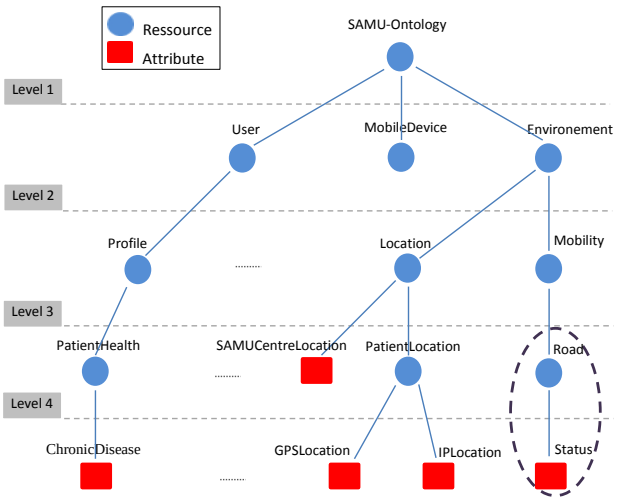


FIGURE 7.1 – Hiérarchie d'exécution de WildCAT de l'ontologie SAMU (un extrait)

combinaison d'outils ; système de WildCAT [2] et Esper [3] qui sont équipés de capteurs et un moteur CEP. La mission de ce module est, en continu et en temps réel, le contrôle et le traitement de la fluctuation des évènements contextuels simples ou complexes afin d'affiner le contexte et de le stocker dans le fichier OWL, en utilisant l'API Jena [4]. Le fonctionnement de l'étape de surveillance est résumée dans une classe Java avec quatre méthodes dans l'ordre suivant :

1. Chargement du modèle de contexte : le modèle de contexte basé sur l'ontologie et enregistrée dans le fichier OWL est chargé en mémoire à l'aide de l'API Jena.
2. Construire une hiérarchie de contexte WildCAT en utilisant l'API WildCAT en mode push : à partir du modèle d'ontologie chargée, les Concepts et les Datatype-properties seront transformés en ressources et attributs dans WildCAT respectivement. La Figure 7.1 donne une vue de la hiérarchie de WildCAT construit avec un accent sur l'attribut Status de la ressource Road selon notre cas.
3. Définition et attachement des capteurs : un capteur de WildCAT peut être défini par l'extension de la classe POJOAttribute Java et associé à une sonde liée à l'environnement sous forme de capteurs matériels (caméra, thermomètre, etc.) ou des capteurs logiciels (service Web, par exemple). Ensuite, les attributs de la hiérarchie de WildCAT que nous voulons surveiller sont attachés aux capteurs de WildCAT définis (par exemple, l'attribut Status qui a eu d'abord la valeur "IsNotBlocked").
4. Stockage de l'évolution des valeurs de contexte et le déclenchement du Raisonneur : le capteur en charge de la surveillance des attributs doit stocker les valeurs modifiées dans le fichier OWL et déclencher le raisonneur. Pour notre exemple, une requête Esper relative au Status de la route qui a pris "IsBlocked" comme une nouvelle valeur peut faire cette action. Ainsi, la valeur de Status DatatypeProperty du fichier OWL passe de "IsNotBlocked" à "IsBlocked" pour l'itinéraire choisi. Ce changement de valeurs est considéré comme un évènement déclencheur pour lancer le processus

2. http://wildcat.ow2.org/
3. http://esper.codehaus.org/
4. https://jena.apache.org/documentation/ontology/

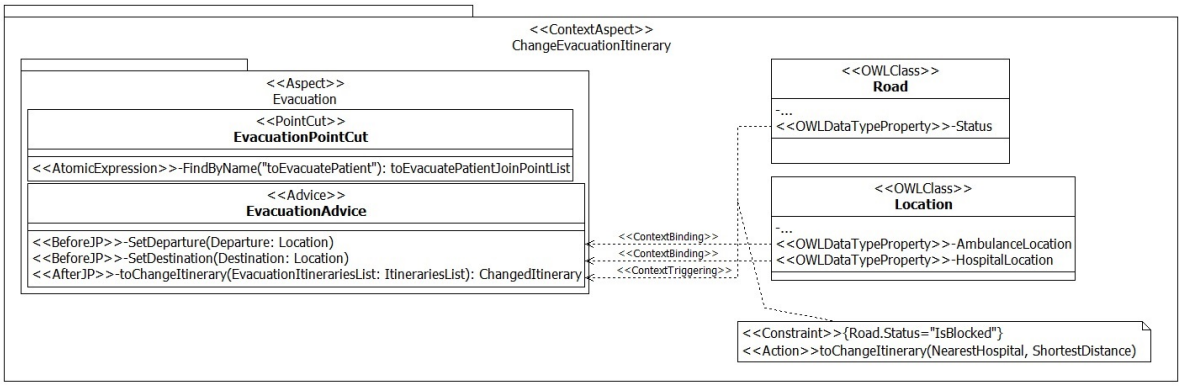


Figure 7.2: Le modèle ContextAspect ChangeEvacuationItinerary

d'adaptation dynamique afin de changer l'itinéraire choisi pour l'évacuation (voir les sections suivantes).

7.2.2 Étape d'analyse

Le raisonneur basé sur des règles ECA, qui sont déjà établies et stockées dans une base de règles au moment de la conception, analyse la nouvelle valeur de contexte afin de vérifier et déclencher les règles correspondantes. Les règles ECA peuvent être mises en œuvre et exécutées par Drools Engine [5] ou par des déclarations simples "IF-THEN" comme il est employé dans notre cas. Dans ce scénario, la déclaration (If Road.Status="IsBlocked" Then Weave(ChangeEvacuationItinerary)) décide de tisser le modèle ContextAspect ChangeEvacuationItinerary (Figure 7.2) pour appliquer une adaptation dynamique afin de changer l'itinéraire d'évacuation précédent qui est devenu maintenant bloqué (par exemple, en raison d'un accident de voiture, des travaux imprévus ou la congestion routière). Ce modèle ContextAspect propose à l'ambulance un nouvel itinéraire GPS en conséquence à cette situation de contexte. Pour d'autres situations contextuelles, d'autres modèles ContextAspect peuvent être tissés en conséquence. Par exemple, aucun changement d'itinéraire, ni le choix du plus court chemin sont nécessaires si le moyen d'évacuation est un hélicoptère.

7.2.3 Étape de planification

Le modèle ContextAspect ChangeEvacuationItinerary sélectionné sera converti en codes FPath et FScript pour générer les scripts de reconfiguration nécessaires pour les appliqués sur le service toEvacuatePatient. L'expression de pointcut est transformé en code FPath (Figure 7.3) qui sélectionne le composant de service toEvacuatePatient sujet d'adaptation sans affecter les autres composants de l'architecture de l'application SCA qui continuent leur exécution.

En utilisant des actions de tissage, le greffon est ainsi transformé en code FScript comme un script de reconfiguration pour le tissage du modèle ContextAspect ChangeEvacuationItinerary. Dans la Figure 7.4, le listing de script met en œuvre une opération de tissage, comme décrit dans la section 6.5.2 (Instruction 3, Règle 1). Tout d'abord, il apporte l'application dans un état de repos (ligne 2), puis crée une instance de trois composants de

5. `http://www.jboss.org/drools`

```
// Equivalent FPath to PointCutExpression: FindByName('toEvacuatePatient')

        toDisplayJP=$root/descendant-or-self::*[name(.)=='toEvacuatePatient']
```

Figure 7.3: Le code FPath équivalent pour le ContextAspect ChangeEvacuationItinerary

```
1  action WeaveEvacuation(EvacuationApplication, EvacuationAdvice){
2   stop($EvacuationApplication);
3   // subcomponent of SetDepartureServiceComponent
4   SetDepartureServiceComponent = adl-new($SetDepartureAdvice);
5   addFcSubComponent($EvacuationApplication, $SetDepartureServiceComponent);
6   bindFc($toEvacuatePatientServiceComponent/interface:: Departure,
7   $SetDepartureServiceComponent/interface:: Departure);
8   // subcomponent of SetDestinationServiceComponent
9   SetDestinationServiceComponent = adl-new($SetDestinationAdvice);
10  addFcSubComponent($EvacuationApplication, $SetDestinationServiceComponent);
11  bindFc($toEvacuatePatientServiceComponent/interface:: Destination,
12  $SetDestinationServiceComponent/interface:: Destination);
13  // supercomponent of toChangeItineraryServiceComponent
14  toChangeItineraryServiceComponent = adl-new($toChangeItineraryAdvice);
15  addFcSuperComponent($EvacuationApplication, $toChangeItineraryServiceComponent);
16  bindFc($toChangeItineraryServiceComponent/interface:: EvacuationItinerariesList,
17  $toEvacuatePatientServiceComponent/interface:: EvacuationItinerariesList);
18  promote($toChangeItineraryServiceComponent/interface:: ItineraryChanged, $EvacuationApplication);
19  bind($toChangeItineraryServiceComponent/interface:: ItineraryChanged, "http");
20  start($EvacuationApplication);
21 return $EvacuationApplication;}
```

Figure 7.4: Le code FScript de Reconfiguration pour le tissage de ChangeEvacuationItinerary

service, à savoir : SetDeparture (ligne 4), SetDestination (ligne 9) et toChangeItinerary (ligne 14). Les deux premiers composants sont décrits par des greffon «BeforeJP» (voir la Figure 7.2), où ils sont inclus dans l'architecture de l'application en tant que sous-composants par la primitive addFcSubComponent (lignes 5 et 10) et liées à toEvacuatePatientServiceComponent (lignes 6-7 et 11-12). Cependant, le dernier nouveau composant est inclus comme supercomposant par la primitive addFcSuperComponent (ligne 15) pour un greffon «AfterJP» (Figure 7.2) et lié à toEvacuatePatientServiceComponent (lignes 16-17). Enfin, le service ItineraryChanged est promu au composite englobante (ligne 18) et exposée comme une liaison HTTP (ligne 19). Lorsque ces actions sont terminées, l'exécution de l'application peut être repris (ligne 20).

Pour une action de détissage, nous pouvons utiliser les primitives RemoveFcSuperComponent, RemoveFcSubComponent et UnbindFC [42, 112].

7.2.4 Étape d'exécution

Au niveau de l'exécution, l'adaptation se produit comme suit : tout d'abord et avant l'évacuation du patient de sa localisation à l'hôpital le plus proche, le service "toEvacuatePatient" affiche sur l'écran du GPS de l'ambulance une liste avec une carte d'itinéraires possibles qui ne sont pas bloqués (haut de la carte carte de la Figure 7.5) afin de sélectionner le plus court à prendre par le conducteur de l'ambulance (bas de la carte de la Figure 7.5). Ensuite, et après un certain temps de conduite sur cet itinéraire sélectionné, un évènement contextuelle est observée, et qui indique un changement dans l'état de la route de l'itinéraire choisi (de "IsNotBlocked" à "IsBlocked"). Ceci va déclencher une adaptation du service toEvacuatePatient (comme détaillé dans les étapes précédentes) afin de modi-

fier l'itinéraire en tenant compte de l'emplacement actuel de l'ambulance (voir la Figure 7.2) comme un nouveau départ (i.e, où il a reçu le nouvelle information de contexte sur le blocage de l'itinéraire précédent). Parmi plusieurs nouveaux itinéraires avec nouvelles distances vers la même destination (en haut de la carte de la Figure 7.6), ce service va changer l'itinéraire en choisissant également le plus court (en distance et en temps) d'entre eux. Enfin, une nouvelle carte pour l'itinéraire nouvellement changé est affiché afin de la suivre par le conducteur de l'ambulance (bas de la carte de la Figure 7.6), et ceci en vue d'une évacuation rapide du patient en état grave pour sauver sa vie.

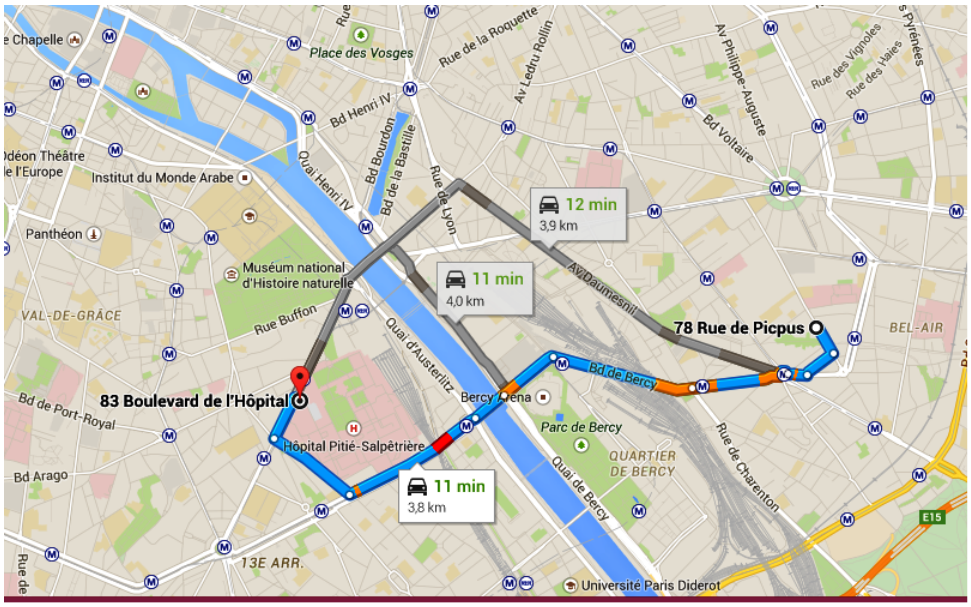
Il faut noter que ces cartes sont affichées en utilisant l'API Google Maps (version 3) [6].

7.3 Conclusion

Dans ce chapitre, nous avons détaillé le fonctionnement du processus de l'adaptation dynamique des applications sensibles au contexte à base de services. Ce processus applique la boucle de contrôle MAPE-K pour reconfigurer ce genre d'applications selon le contexte qui est fréquemment changeable. Cette boucle consiste à passer par quatre étapes pour réaliser l'adaptation dynamique, à savoir, la surveillance, l'analyse, la planification et l'exécution. Ce processus d'adaptation a été validé par un scénario d'évacuation d'un patient en utilisant une combinaison de technologies et outils, notamment la plateforme FraSCAti.

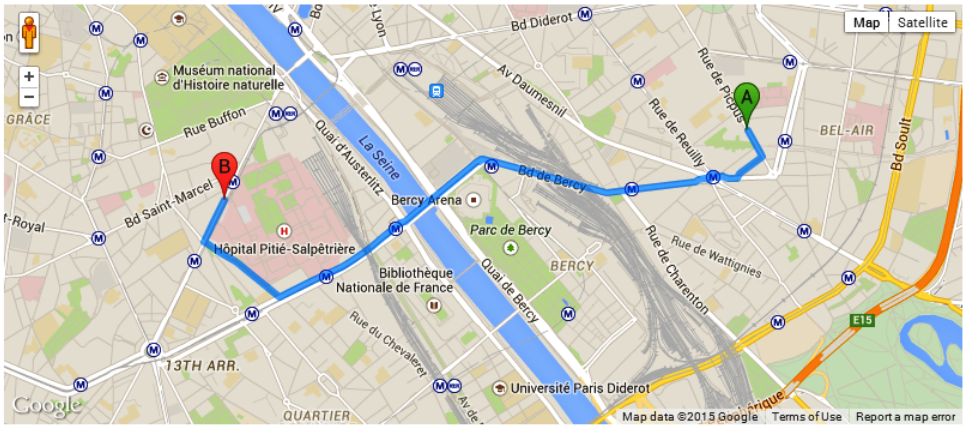
Par l'achèvement de cette phase d'adaptation, nous arrivons à terminer la concrétisation de toutes les phases de l'approche de développement dirigé par les modèles proposée dans cette thèse. Le chapitre suivant fera l'objet sur l'évaluation de cette approche par rapport aux autres travaux proposés dans la littérature.

6. `https://developers.google.com/maps/web/`



EVACUATION SERVICE WITH GOOGLE MAPS API V3

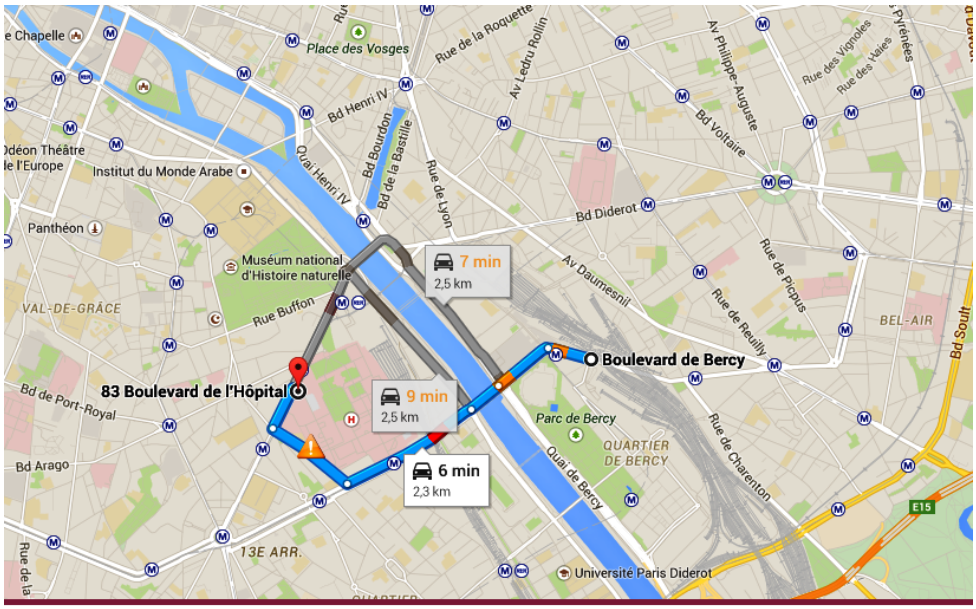
Departure: 78 Rue de Picpus, 75012 Destination: 83 Boulevard de l'Hôpital, Calculate the route



Suggested routes:

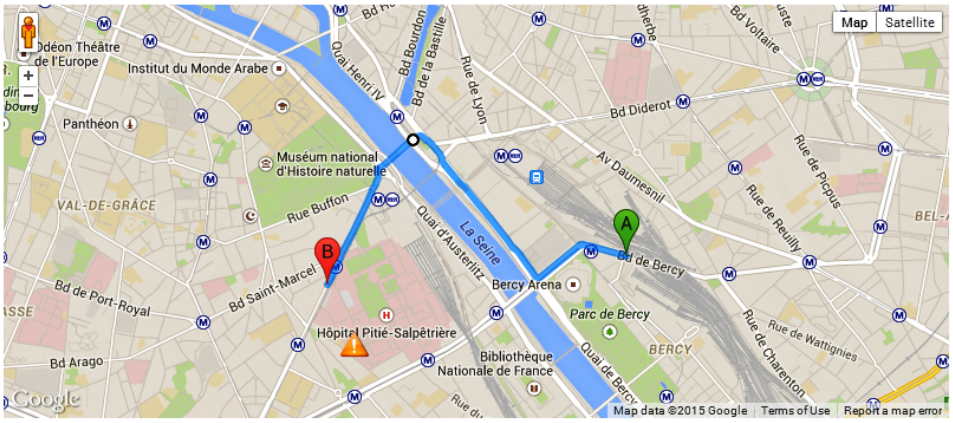
Boulevard de Bercy 3.8 km - about 11 mins
Boulevard de Bercy 4.0 km - about 12 mins
Avenue Daumesnil 3.9 km - about 13 mins

Figure 7.5: Itinéraire d'Évacuation Avant l'Adaptation



EVACUATION SERVICE WITH GOOGLE MAPS API V3

Starting point : Boulevard de Bercy, 7501 Destination : 83 Boulevard de l'Hôpital, Calculate the route



Suggested routes:

Boulevard Vincent Auriol 2.3 km - about 7 mins

Quai de la Rapée and Boulevard de l'Hôpital 2.5 km - about 7 mins

Quai d'Austerlitz and Boulevard de l'Hôpital 2.5 km - about 9 mins

Figure 7.6: Itinéraire d'Évacuation Après l'Adaptation

Chapitre 8

Comparaison, Avantages et Limites

8.1 Introduction

Après la présentation de notre approche dans les chapitres 5, 6 et 7, nous voulons discuter maintenant son évaluation en la comparant avec les autres travaux connexes de développement dirigé par les modèles des applications sensibles au contexte à base de services.

Dans la section 4.3 du chapitre 4, nous avons donné une première comparaison avec quelques critères selon l'objectif de ce chapitre [23]. Dans le present chapitre, nous allons étendre cette comparaison par d'autres issues afin de tirer mieux les avantages et les limites de l'approche proposée [27].

8.2 Comparaison avec les travaux connexes

Le traitement de l'adaptation au contexte comme une préoccupation transversale dans l'application de base n'est pas un nouveau point de vue. Il était l'idée principale de plusieurs travaux proposés et à des niveaux différents que nous pouvons les classer en trois catégories, à savoir :

8.2.1 Travaux combinant MDD avec AOM

Au meilleur de notre connaissance, [33] et [56] sont les deux seuls travaux qui ont combiné le Développement Dirigé par les Modèles (MDD) avec la Modélisation Orientée Aspect (AOM) pour développer des applications sensibles au contexte. Dans le premier, les auteurs exigent que le concepteur commence par la modélisation de l'application pervasive dans des modèles « Theme/UML » dans lequel les préoccupations des domaines spécifiques et sensibles au contexte, qui représentent le comportement transversal, sont séparées de l'application de base. Le concepteur précise ensuite où et comment ces préoccupations seront composées avec les autres. En utilisant les transformations pilotées par les modèles, le concepteur gagne des avantages supplémentaires de l'indépendance avec la plateforme et de la technologie utilisée. Le deuxième travail adresse la conception d'applications sensibles au contexte à base de services en identifiant les préoccupations transversales sensibles au contexte au niveau modèle. Il propose un modèle conceptuel de contexte qui inclut l'historique d'évènements avec l'attribution des valeurs, et un modèle conceptuel

pour l'adaptation de la sensibilité au contexte en utilisant des mécanismes basés sur les concepts de la conception orientée aspect (AOD) [52]. En outre, ils présentent un cadriciel (framework) UML pour concevoir des modèles d'adaptation sensibles au contexte qui peuvent être mappés vers le code AspectJ [1] par un moyen approprié de processus de transformation/raffinement.

8.2.2 Travaux combinant MDD avec AOP

Au niveau du code, nous trouvons quelques travaux qui combinent la Programmation Orientée Aspect (AOP) et MDD pour aborder la complexité du développement des applications sensibles au contexte. Dans [103], Prezerakos et al. ont proposé une logique de service de base et une adaptation sensible au contexte en des préoccupations séparées en utilisant une version modifiée de ContextUML [113] pour la modélisation et les aspects pour encapsuler le comportement dépendant du contexte dans des modules discrètes en code aspectJ. Tanter et al . dans [123] ont présenté, aussi, l'adaptation au contexte dans ce qu'ils appellent les aspects sensibles au contexte. Cela signifie que l'utilisation des aspects est entrainé par le contexte ; un certain aspect peut ou non être exécuté en fonction de son contexte d'utilisation. Les hypothèses supplémentaires faites dans cette approche est que les contextes, guidant l'invocation d'aspect, peuvent avoir des paramètres et qu'ils peuvent être combinés avec d'autres contextes. Cependant, ce qui est plus important dans ce travail, est que le comportement de service peut être affectée non seulement par des valeurs de contexte actuel, mais aussi par celles du passé [72].

8.2.3 Travaux basés sur MDD

D'autres travaux ont uniquement basé sur l'approche MDD pour gérer le développement des applications/services sensibles au contexte tel que discutés dans [72, 23] et dans le chapitre 5. Ci-après, nous citons ceux fortement associés à notre travail.

Sheng et al . [113] ont proposé le métamodèle ContextUML qui étend la syntaxe UML pour introduire des artéfacts appropriés permettant la création de modèles de services sensibles au contexte. ContextUML prend en charge la modélisation du contexte atomique, le contexte composite et la structure des services Web. Cependant, il ne supporte pas l'adaptation de comportement «Behaviour Adaptation » car il est impossible de changer la logique interne d'un service Web encapsulée selon ses auteurs dans[116]. En outre, l'existence de la structure de service Web avec la description WSDL dans le métamodèle ContextUML réduit l'utilisation de ce dernier pour d'autres descriptions de service telle que la structure REST [51].

Ayed et Berbères [10] ont présenté un métamodèle UML qui soutient l'adaptation sensible au contexte dans la conception des services de point de vue structurel, architectural, et de comportement. L'approche de Kapitsaki et al. [73] a proposé une architecture d'adaptation au contexte des compositions de services Web et une méthodologie dirigée par les modèles pour le développement de ces applications composites sensibles au contexte. Cette méthodologie utilise les diagrammes UML au cours de la phase de conception afin de générer des services Web fonctionnels à travers un processus de transformation adéquate. Lors de la phase de modélisation, la conception est maintenue indépendante de la mise en œuvre sur des plateformes spécifiques, est suffisamment flexible pour permettre

l'introduction des différentes correspondances spécifiques de code. Aussi, et pendant les phases de développement et de l'exécution de l'application, les logiques de service et de l'adaptation au contexte sont maintenus indépendants, fournissant ainsi la flexibilité et la facilité de maintenir l'application. La modélisation exploite un certain nombre de profils prédéfinis (services Web, méta-modèle de contexte et présentation), alors que, la mise en œuvre cible est basée sur une architecture qui effectue l'adaptation au contexte des services Web en utilisant l'interception des messages du protocole d'accès simple aux objets (SOAP). Dans cette approche, l'adaptation de comportement est traitée par l'injection des différentes opérations alternatives dans la structure de service Web ce qui gonfle le code implémenté du service Web.

Dans [125], Vale et Hammoudi ont présenté la méthodologie COMODE avec cinq vues (métier, contexte, composition, adaptation et service) pour le développement d'applications sensibles au contexte en utilisant une approche dirigée par les modèles dans lequel il propose :

- Un métamodèle de contexte pour les applications ubiquitaires centrée sur l'utilisateur afin de modéliser les informations de contexte,
- Et, une technique de transformation paramétrée pour composer des informations de contexte avec la logique métier au niveau modèle. Cette technique de composition est basée sur une utilisation laborieuse de marquage et quantification des paramètres contextuels à composer entre les modèles de métier et de contexte.

En même portée que la nôtre, mais sans adopter l'adaptation sensible au contexte comme une préoccupation transversale, Ou et al. dans [98] ont examiné la façon dont MDA peut être utilisée pour la modélisation du contexte et la modélisation et le développement des applications sensibles au contexte (CAA). Un modèle ontologique de contexte (COM) est présenté pour modéliser le contexte à deux niveaux : niveau supérieur (ULCOM) et niveau spécifique étendu (ESCOM). En utilisant les méta-modèles RDFS/OWL, l'ontologie ULCOM capture les concepts qui sont essentiels dans le contexte générique caractérisant le domaine des services pervasives sous trois entités principales, Entity, EntityProperty, et EntitySpecification. ESCOM définit les concepts spécifiques et leurs propriétés pour le contexte comme des extensions des entités ULCOM. Également dans ce travail, une architecture d'intégration dirigée par les modèles (MDIA) est proposée pour intégrer les spécifications du modèle rigoureux et générer des implémentations de la CAA d'une façon automatique ou semi-automatique. Le principal avantage de cette approche est de démontrer la faisabilité de la collecte des UML, MDA et langages d'ontologie (comme RDFS et OWL) pour la modélisation de l'ontologie de contexte et vers l'architecture d'intégration basée sur MDA pour le développement automatique des applications sensibles au contexte. Cette approche est validée par une étude de cas présentée dans [55]. Cependant, l'inconvénient majeur de ce travail réside dans la gestion de l'adaptation au contexte qui est lié avec le développement de la logique métier.

8.3 Avantages et résultats

Notre travail est inspiré de plusieurs travaux combinant les paradigmes orientée aspect et dirigé par les modèles pour la gestion de la variabilité dynamique dans les lignes de produits logiciels (*DSPL: Dynamic Software Product Line*) [85, 100] et les systèmes adap-

tatifs comme dans la méthodologie DIVA [57] [1]. Il consiste, avec une manière innovante, à gérer la logique de sensibilité au contexte des applications comme des points de variation à remplir par le tissage des différentes variantes définies dans des modèles d'aspect sensibles au contexte.

Grâce aux modèles ContextAspect, nous pouvons concevoir des logiques d'adaptation pour une application sensible au contexte, contrairement aux aspects utilisés dans les travaux antérieurs (comme dans [56, 103]) qui contiennent plusieurs codes d'adaptation dans un seul aspect.

Aussi, l'approche proposée est caractérisée par le tissage de plusieurs greffons et aspects dans le même point de jonction en assurant la priorité/coordination entre ces aspects tissés. Cette fonction permet de prendre en compte les différentes situations contextuelles rencontrées dans la vie et de gérer en conséquence un grand nombre d'applications réelles. De nombreuses approches ont préconisé la tendance orientée aspect, et même certains d'entre eux [56, 103, 123] assurent la priorité d'exécution entre les aspects à travers l'utilisation de la plateforme AspectJ. Cependant, ces approches ne traitent pas ce point de tissage et aucune coordination de fonctionnement est proposé en conséquence dans leurs propositions.

En relation avec les approches discutées ci-dessus, notre proposition couvre les différentes questions de la sensibilité au contexte allant des métamodèles de haut niveau à la mise en œuvre sur une véritable plateforme supportant l'adaptation dynamique. En plus des avantages des approches mentionnées, voici les principaux points forts couverts également par notre approche :

1. La conformité totale de la méthodologie de développement proposée avec le processus de la technologie MDA et le paradigme orienté aspect,
2. Le traitement de la logique de sensibilité au contexte (ContextBinding, ContextTriggering and BehaviourAdaptation) à un stade de développement précoce (au niveau du modèle) en composant les adaptations sensibles au contexte encapsulées dans des modèles d'aspect avec le modèle de base (tissage@design-time),
3. Le traitement de la logique de sensibilité au contexte au niveau du code en reconfigurant les applications sensibles au contexte à base de services pour permettre des adaptations dynamiques successives en fonction du changement de contexte (tissage@ run-time), et
4. L'efficacité de l'utilisation de la plateforme FraSCAti pour soutenir l'adaptation dynamique des applications sensibles au contexte à base de services.

Bien que l'étude de cas présentée n'a pas recueilli des données pour appuyer nos revendications suivantes, notre expérience, en mettant en œuvre ce système, nous a conduit à croire que le MDD et AOM rendent le développement des applications sensibles au contexte à base de services moins complexe, moins de lourdeur et avec moins de temps consommé :

Moins complexe : L'idée de séparer la logique métier, les informations de contexte et la logique d'adaptation sensible au contexte dans des modèles a considérablement contribué à surmonter les difficultés connues dans le développement des applications sensibles au contexte comme il est démontré dans ce travail. Le modèle de contexte

1. `http://www.ict-diva.eu/DiVA`

a base ontologique, qui est extensible et conforme à la spécification ODM, permet de traiter les informations de contexte par une méthodologie pilotée par les modèles. Cependant, les adaptations sensibles au contexte sont encapsulées dans des modèles aspects sensibles au contexte. Grâce au métamodèle ContextAspect, les différentes logiques d'adaptation sensibles au contexte liées à une application sont conçues dans des modèles séparés permettant de tenir en compte des situations contextuelles différentes indépendamment les uns des autres, contrairement à d'autres types d'aspect dans certains travaux antérieurs [56, 103] où un seul aspect est utilisé pour contenir plusieurs codes d'adaptation pour différentes situations contextuelles. Et pour une nouvelle situation contextuelle, il suffit de décrire la logique d'adaptation sensible au contexte, qui devrait être suivie, dans un modèle ContextAspect conjointement avec une ou plusieurs règles ECA pour définir quand, où, quoi et comment mener à bien le tissage de cette adaptation au contexte.

Moins de lourdeur : L'adaptation au contexte courant est assurée par une composition de modèles entre la logique métier (PIM) et la logique de sensibilité au contexte (modèles ContextAspect) en un seul modèle contextuel indépendant de la plateforme (CPIM). CPIM ne comprend pas dès le début tous les détails techniques de la plateforme d'exécution qui peut rendre le processus de développement de ce type d'applications compliqué. En outre, les phases de la composition et l'adaptation sont obtenues en utilisant une approche basée sur le point de coupe au lieu de la tâche ardue de la quantification des éléments sensibles au contexte utilisée dans les approches fondées sur la fusion [89] et l'approche paramétrée [125].

Moins de temps consommé : En plus de temps gagné par une approche de développement, qui est moins complexe et moins encombrant, la composition des modèles par des techniques de tissage et la production de code par des transformations automatiques successives (M2M ou M2T) permettent ainsi d'automatiser la plupart des activités de développement des applications sensibles au contexte et, en conséquence, réduire le temps. La boucle de reconfiguration MAPE-K est un autre point clé disponible uniquement dans notre proposition et ignorée dans les autres. Cette boucle est adoptée comme une stratégie dynamique d'adaptation. Faire introduire cette dernière lors de l'exécution dans le développement de ces applications par une méthodologie pilotée par les modèles contribue de manière significative à améliorer le délai de reconfiguration selon le changement de contexte. Par conséquent, tout le temps gagné rend notre approche rapide par rapport à d'autres citées dans cette section.

8.4 Limites de l'approche proposée

Bien que notre approche apporte des avantages comme nous l'avons présenté dans la section précédente, certaines limitations sont également enregistrées dans la présente proposition que ce soit au niveau de la conception ou de l'implémentation de la méthodologie proposée. Parmi ces limitations, nous pouvons citer :

Le présent travail ignore l'exploitation des données historiques des évènements de contexte passés [72] et ne met pas en disposition des politiques pour la confidentialité et la protection des informations de contexte en vue d'assurer la vie privée des utilisateurs

de ce genre d'applications.

Aussi, tout au long de cette approche, nous avons adopté l'appariement basé sur le nom pour la sélection des points de jonction dans une application sensible au contexte à base de services. Ce type d'appariement pourrait conduire à des conflits dans le tissage des aspects sensibles au contexte (les modèles ContextAspect) et au dysfonctionnement des applications en conséquence. Nous pensons que l'utilisation de l'appariement par signature peut faire face au maximum des conflits de noms pourraient y avoir entre les éléments modèles dans une application. Dans ce but, nous pouvons bénéficier de l'utilisation de la plateforme AspectJ qui offre l'appariement par signature comme il était le cas dans les travaux [56, 103, 123].

Également, pour la validation de notre approche, nous avons utilisé un combinaison d'outils et de plateformes/frameworks. Cette diversité d'outils ne permet pas d'évaluer mieux les performances de notre approche et d'en donner des statistiques notamment en matière de temps d'exécution. Nous croyons que la conception et le développement des applications sensibles au contexte nécessitent un cadre de travail (framework) unique pour s'assurer de la qualité des applications produites.

8.5 Tableau comparatif

Nous concluons ce chapitre par un tableau (table 8.1) qui résume les travaux connexes discutés avec leurs avantages et inconvénients, et de les comparer avec notre approche en fonction des différentes issues abordées dans la littérature [72, 116]. De plus, nous avons ajouté des nouveaux critères de comparaison tels que (« Conformance avec la Technologie MDA » et « Adaptation@Run-Time») pour montrer la particularité de notre proposition.

8.6 Conclusion

Dans ce chapitre, nous avons comparé et évalué notre approche par rapport à l'ensemble des autres approches proposées dans la littérature du développement dirigé par les modèles des applications sensibles au contexte à base de services.

Nous avons discuté par un tableau comparatif les points forts et les points faibles de notre travail et aussi bien des autres travaux. Les avantages de l'approche proposée conduit à croire que le MDD et l'AOM rendent le développement des applications sensibles au contexte à base de services moins complexe, moins de lourdeur et moins de temps consommé :

En revanche, certains limites de notre proposition ont été citées et qui pourraient être des sujets de recherche pour le futur proche.

Issues/Approaches		Carton [33]	Grassi [56]	Prezerakos [103]	Tanter [123]	Sheng [113]	Ayed [10]	Kapitsaki [73]	Vale [125]	Ou [98]	Our Approach
Conformance avec la Technologie MDA	PIM, PSM, Transformations.	+	+	-	-	-	-	-	+	+	+
Modélisation du Contexte	Contexte de Bas Niveau	+	+	+	+	+	+	+	+	+	+
	Contexte de Haut Niveau	-	+	+	-	+	-	+	-	+	+
Support des Données Historiques	Support des Données	-	+	-	+	-	-	-	-	-	-
	Historiques										
Modélisation de la Sensibilité au Contexte	Context Binding	+	+	+	+	+	+	+	+	-	+
	Context Triggering	+	+	+	+	+	+	+	-	-	+
	Behaviour Adaptation	-	-	-	-	-	+	-	-	-	+
Découplage Métier/Contexte	Logique Métier vs Logique de Sensibilité au Contexte	+	+	+	+	+	+	+	+	-	+
Modélisation de la Composition	Logique Métier + Logique de Sensibilité au Contexte	+	+	-	-	-	-	-	+	+	+
Adaptation @Run-Time	Niveau Modèle	-	-	-	-	-	-	-	-	-	+
	Niveau Code	-	+	+	-	-	-	+	-	-	+
Vie privée et Sécurité	Vie privée et Sécurité	-	-	-	-	-	-	-	-	-	-
Implément. Plateforme	Basé-SOA	-	-	+	-	+	+	+	+	+	+
Composition des aspects (greffons)	Priorité d'exécution	-	+	+	+	-	-	-	-	-	+
	Coordination d'exécution	-	-	-	-	-	-	-	-	-	+
Type d'appariement	Basé-nom	-	-	-	-	-	-	-	-	-	+
	Basé-signature	-	+	+	+	-	-	-	-	-	-

Table 8.1: Comparaison avec les approches discutées

Chapitre 9

Conclusion Générale

Les applications et services sensibles au contexte sont une technologie prometteuse pour créer des applications personnalisées dans des environnements pervasives riches par les informations de contexte qui sont en constante évolution. Malheureusement, cette technologie est encore difficile à atteindre en raison d'un manque de méthodologie générique qui formalise le processus de développement de ce type d'applications et qui devraient s'adapter selon la logique du changement de contexte en utilisant divers mécanismes de sensibilité au contexte. Aussi, cette adaptation devrait être tenue au niveau de conception comme au niveau d'exécution de l'application et de manière transparente par rapport aux utilisateurs finaux.

Notre travail s'est inscrit dans cet objectif pour faciliter la réalisation des applications sensibles au contexte. Où, nous avons proposé une nouvelle approche qui se concentre sur la considération que la sensibilité au contexte peut être gérée à travers la notion de variabilité introduite initialement dans les approches des logiciels de lignes des produits (SPL). En consequence, l'application sensible au contexte est vue comme une application avec plusieurs points de variation qui reflètent leurs éléments sensibles au contexte. Et pour chaque point de variation, des multiples variantes sont associées afin de sélectionner l'une d'eux en fonction du contexte actuel.

Cette approche prend avantage de combiner le développement dirigé par les modèles (MDD) et la modélisation orientée aspect (AOM) pour soutenir le développement des applications sensibles au contexte à base de services pour les environnements mobiles et pervasifs.

Le MDD pilote la conception et le développement de ces applications par des modèles. Ces derniers peuvent être convertis en d'autres modèles ou codes par des techniques de transformation au lieu de les écrire à la main qui est une tâche ardue et sujette aux erreurs. Cependant, l'AOM gère la logique de sensibilité au contexte dans les modèles « ContextAspect » (comme des variantes) pour remplir des éléments d'application sensibles au contexte (les points de variation) en utilisant des techniques de tissage à la conception et aussi à l'exécution.

A la conception, le tissage permet de produire une large gamme de modèles d'applications sensibles au contexte, sans les concevoir dès le début. D'autre part, le tissage d'exécution se compose de tissage de la reconfiguration nécessaire dans l'application en cours d'exécution et en fonction du changement de contexte, et donc l'accomplissement de son adaptation dynamique.

Grâce à la plateforme FraSCAti, ce genre d'adaptation peut être facilement réalisé ;

cependant, toute autre plateforme basée SOA pourrait être choisie tant qu'elle prend en charge le tissage des aspects à la conception et en temps de fonctionnement tels que les plateformes « AspectJ-like ».

Cette approche a été validée par un exemple illustratif pour montrer comment dérouler la conception, le développement et l'adaptation des applications sensibles au contexte à base de services.

9.1 Synthèse des contributions

Durant la réalisation de ce travail, plusieurs contributions ont été apportées afin de mieux mener notre recherche dans le domaine de développement des applications sensibles au contexte. Dans ce qui suit nous résumons les plus importantes :

9.1.1 Modélisation du Contexte avec les Ontologies

Le développement des applications sensibles au contexte nécessite, d'abord, modéliser correctement toutes les informations de contexte avec une précision très élevée en utilisant le meilleur formalisme possible pour faciliter leur exploitation par la suite.

Pour ce travail, nous avons opté pour l'utilisation des ontologies qui sont connues dans la littérature comme un formalisme très puissant pour modéliser le contexte avec une expressivité sémantique très riche et extensible [120, 71, 16, 5]. Dans le chapitre 5, nous avons présenté un modèle de contexte extensible basé sur les ontologies [23] et structuré avec la spécification ODM (Ontology Definition Metamodel) proposée par l'OMG [93].

L'ODM offre un profil UML pour représenter les ontologies (OUP) sous forme d'un modèle UML facile à intégrer dans un processus de développement dirigée par les modèles des applications mobiles sensibles au contexte. L'ODM permet d'allier les deux paradigmes de l'ingénierie des logiciels, à savoir, l'ingénierie pilotée par les modèles (MDE) avec sa technologie MDA et les ontologies, et de combler l'écart entre eux.

Le modèle de contexte proposé est composé de trois niveaux d'ontologies, une ontologie générique, une ontologie de domaine pour les concepts de contexte populaires et communs, et une ontologie d'application pour une description précise de contexte dans un espace clos. La structuration de ces ontologies de contexte permet de rendre le modèle proposé :

1. Suffisamment général pour être utilisé par les différentes applications sensibles au contexte centrées sur l'utilisateur,
2. Suffisamment spécifique pour couvrir les principales entités contextuelles proposées dans la littérature des applications mobiles et sensibles au contexte, et
3. Suffisamment flexible pour permettre son extension par la prise en compte des nouvelles entités spécifiques à un domaine d'application donné.

En outre, les capacités de raisonnement offertes par les ontologies approuvent à notre modèle d'utiliser les règles SWRL (Semantic Web Rule Language) [65], et cela pour déduire d'autres informations contextuelles de haut niveau qui pourraient compléter le contexte de bas niveau décrivant une situation d'interaction utilisateur/application.

9.1.2 Architecture logicielle conforme à MDA et le métamodèle ContextAspect

Le processus de développement des applications sensibles au contexte consiste à comment réaliser la sensibilité au contexte qui est déjà capturé et modélisé, et l'adaptation du comportement de l'application en fonction des changements de contexte au moment de l'exécution.

En exploitant les avantages de la combinaison de développement dirigé par les modèles (MDD) et la modélisation orientée par aspect (AOM) [8], nous avons proposé dans le chapitre 6 une approche de développement dirigée par les modèles basée sur une architecture logicielle conforme à MDA [27] de quatre niveaux de modélisation. Les développeurs peuvent se servir de cette architecture pour construire des applications sensibles au contexte à base de services.

Parmi les métamodèles définis dans cette architecture on trouve le métamodèle « ContextAspect » qui est le plus essentiel dans l'approche proposée. ContextAspect est pour but d'encapsuler, dans des modèles d'aspect, les différentes comportements qu'une application peut prendre dans les différentes situations contextuelles. La structure du métamodèle ContextAspect contient trois parties d'éléments :

1. Les éléments du modèle « Aspect » : un aspect comprend des greffons (*advice*) et des points de coupe (*pointcut*). Le greffon décrit le comportement transversal à tisser dans les points de jonction (*join points*), tandis que le point de coupe définit les expressions pour détecter ces points de jonction. Les greffons d'un aspect peuvent être tissés dans le modèle de base en trois parties logiques par rapport au point de jonction ; Avant (`BeforeJP`), Autour (`ReplaceJP`) et Après (`AfterJP`).
2. Les éléments du modèle « Contexte » : le modèle de contexte basé sur les ontologies, proposée précédemment, permet de stocker les informations de contexte dans les attributs «`OWLDataTypeProperty`». Les informations de contexte, capturées depuis des sources matérielles et/ou logicielles, sont modélisés en tant que de contexte de bas niveau dans la classe «`LowLevelContext`», et celles déduites, en utilisant des règles SWRL, sont modélisés comme un contexte de haut niveau dans la classe «`HighLevelContext`».
3. Les éléments du modèle « Sensibilité au Contexte » : les mécanismes de sensibilité au contexte, mentionnées dans la section 6.2, sont représentés par les relations «`ContextBinding`», «`BehaviourAdaptation`» et «`ContextTriggering`» entre la classe de contexte «`OWLDataTypePropoerty`» et les classes «`BeforeJP`», «`ReplaceJP`» et «`AfterJP`» du greffon de l'aspect, respectivement.

Les modèles ContextAspect instanciés peuvent être facilement tissés dans la logique métier d'une application sensible au contexte en fonction de son évolution au fil du temps.

9.1.3 Méthodologie Générique de Conception et Développement

La puissance du développement dirigé par les modèles (MDD) réside dans l'utilisation des modèles et des transformations pour concevoir les applications à haut niveau, indépendamment, de tout détail technique de plateforme. Tandis que, la modélisation orientée aspect (AOM) est utile pour séparer les différentes adaptations sensibles au contexte dans

des modèles d'aspects transversaux prêts à être tissés dans le modèle d'application à la conception et pendant l'exécution suivant le contexte qui est fréquemment changeable.

Sur la base de l'architecture logicielle proposée, nous avons présente une méthodologie générique de développement des applications sensibles au contexte à base de services (voir Chapitre 6) [27]. La méthodologie proposée est dirigée par les modèles et comprend quatre phases qui agissent en conformité avec la technologie MDA, à savoir :

1. Phase de modélisation : Après la modélisation du contexte par le modèle basé sur l'ontologie (section 5.6), le processus de développement passe à modéliser le modèle PIM contenant la logique métier de l'application et un ou plusieurs modèles de « ContextAspect » qui encapsulent les différents comportements que l'application doive prendre dans les différentes situations de contexte.
2. Phase de composition : consiste à composer les deux modèles de PIM et un ContextAspect choisi en fonction du contexte (premièrement ou pour l'état de mode sans échec, c'est le modèle de ContextAspect par défaut). La composition peut être réalisée par une technique de tissage d'aspects [66, 100]. Elle procède à tisser le modèle ContextAspect dans le PIM et retourner un seul modèle appelé le modèle contextuel indépendant de la plateforme (CPIM) qui contient les éléments du modèle de la logique métier et ceux des greffons d'aspect du modèle ContextAspect.

   Pour ce tissage, nous utilisons un tisserand AOM qui est un modèle de transformation M2M basé sur *pointcut*, et qui repose sur un métamodèle de tissage et applique un algorithme de tissage (voir la Section 6.5.2).
3. Phase de transformation : Le modèle CPIM, résultant de la phase précédente, est toujours indépendant de toute technologie ou plateforme. Suite à un processus de transformation classique de MDD, le même modèle composé peut être transformé en nombreux modèles de plateformes orientées services supportant le paradigme orienté aspect. Dans ce travail, nous avons opté pour l'architecture SCA (*Service Component Architecture)* [14] et par sa mise en œuvre FraSCAti [111] en tant que plateforme cible pour nos applications.

   Premièrement et par une transformation M2M, le CPIM est converti en un modèle contextuel spécifique à la plateforme (CPSM) lié au modèle SCA.

   Deuxièmement, le modèle de CPSM obtenu sera soumis à une autre transformation de modèle vers texte (M2T) pour générer le code adapté à la plateforme cible d'exécution (Code FraSCAti, dans notre cas).
4. Phase d'adaptation : Le code généré (FraSCAti), prêt à être exécuté, représente la configuration de l'application sensible au contexte à base de services avec le modèle ContextAspect par défaut dans la situation de contexte par défaut. Pendant l'exécution, les informations de contexte pourraient être changées en permanence à d'autres situations, et par conséquent, l'application en train d'exécution doit s'adapter à toute nouvelle situation de contexte par une reconfiguration dynamique au moment de l'exécution.

Les trois premières phases ont été bien détaillées par le déroulement d'un exemple de SAMU. Cependant, l'importance de la dernière phase représente une autre contribution majeure dans notre travail, et qui nous exige de l'expliciter séparément.

9.1.4 Adaptation Dynamique au contexte

L'adaptation dynamique représente la principale caractéristique des applications sensibles au contexte. Ces dernières devraient prendre en considération le changement de contexte au fil du temps ; pendant la conception et avant d'être lancées, et même, après, à l'exécution par l'adaptation de leurs comportements. La plupart des travaux de la littérature ont ignoré cet aspect d'adaptation en temps d'exécution.

Dans notre travail, nous avons introduit cette caractéristique comme une phase entière dans le cycle de développement des applications sensibles au contexte à base de services (voir Chapitres 6, 7) [24, 25]. Nous avons considéré l'application sensible au contexte comme un système autonome que nous avons doté par une boucle de contrôle MAPE-K [75, 79].

Cette boucle consiste à reconfigurer l'application pendant son exécution en passant par quatre étapes pour réaliser l'adaptation dynamique, à savoir, la surveillance de tout changement de contexte, l'analyse de ce changement de contexte pour sélectionner le modèle ContextAspect approprié, la planification de la reconfiguration nécessaire issue de la transformation du modèle ContextAspect sélectionné en codes (FPath et FScript) et enfin, l'exécution de ces codes de reconfiguration pour accomplir l'adaptation dynamique.

9.2 Perspectives envisagées

En vue de renforcer notre approche par d'autres issues, nous planifions à réaliser certains travaux futurs dans le cadre de développement des applications sensibles au contexte à base de services.

Tout d'abord, sur l'enrichissement de l'approche proposée par l'inclusion de « l'appariement par signature » pour la détermination exacte des points de jonction qui sont indispensables pour l'opération de tissage, et cela pour éviter certains conflits qui pourraient découler de l'appariement basé sur le nom.

Aussi, sur la façon d'étendre le modèle de contexte et la méthodologie proposés afin d'intégrer et de traiter les données du contexte social fournies par les différents réseaux sociaux en ligne (sensibilité au contexte social).

Également, sur l'amélioration de notre mise en œuvre pour l'élaboration d'un cadre de travail (Framework) unique, fiable et rapide avec un tisserand AOM générique pour soutenir l'utilisation de l'approche présentée à grande échelle. Ce qui va permettre la construction des applications sensibles au contexte compatibles avec la vie réelle de ses utilisateurs, et de donner une estimation précise sur le temps pris dans leur développement pour des fins de comparaison avec des futurs travaux.

Et enfin, sur la spécification des applications sensibles au contexte selon un processus MDA (CIM, PIM, PSM) par réaliser des modèles de spécification CIM (*Computation Independent Model*) pour les modèles de la logique métier et ceux du ContextAspect. Dans un travail précédent [24], nous avons commencé travailler sur cette perspective que nous souhaitons terminer par d'autres travaux.

Bibliographie

[1] http ://www.eclipse.org/aspectj/.

[2] Dhaminda B. Abeywickrama and Sita Ramakrishnan. Context-aware services engineering :models, transformations, and verification. *ACM Trans. Internet Technol.*, 11(3) :10–28, February 2012.

[3] Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7 :29–58, 2000.

[4] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers : Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[5] Unai Alegre, Juan Carlos Augusto, and Tony Clark. Engineering context-aware systems and applications. *J. Syst. Softw.*, 117(C) :55–83, July 2016.

[6] João Paulo A. Almeida, Maria-Eugenia Iacob, Henk Jonkers, and Dick A. C. Quartel. Model-driven development of context-aware services. In Frank Eliassen and Alberto Montresor, editors, *Proceedings of Distributed Applications and Interoperable Systems, 6th IFIP WG 6.1 International Conference, DAIS 2006*, volume 4025 of *Lecture Notes in Computer Science*, pages 213–227, Bologna, Italy, june 2006. Springer.

[7] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services : Concepts, Architectures and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[8] AOM. Aspect-oriented modelling : www.aspect-modeling.org, 2013.

[9] Colin Atkinson and Thomas Kühne. Model-driven development : A metamodeling foundation. *IEEE Softw.*, 20(5) :36–41, September 2003.

[10] Dhouha Ayed and Yolande Berbers. Uml profile for the design of a platform-independent context-aware applications. In Ian Gorton, Liming Zhu, Yan Liu, and Shiping Chen, editors, *Proceedings of the 1st workshop on MOdel Driven Development for Middleware, MODDM 2006*, volume 183 of *ACM International Conference Proceeding Series*, pages 1–5, Melbourne, Australia, December 2006. ACM.

[11] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The description logic handbook : theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

[12] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4) :263–277, June 2007.

[13] Mary Bazire and Patrick Brézillon. Understanding context before using it. In Anind K. Dey, Boicho N. Kokinov, David B. Leake, and Roy M. Turner, editors, *Modeling and Using Context, 5th International and Interdisciplinary Conference, CONTEXT 2005*, volume 3554 of *Lecture Notes in Computer Science*, pages 29–40. Springer, 2005.

[14] Michael Beisiegel, Dave Booz, Adrian Colyer, Hal Hildebrand, Jim Marino, and Ken Tam. Sca service component architecture : http ://www.osoa.org/display/main/service+component+architecture+specifications, March 2007.

[15] Lodewijk Bergmans, Bedir Tekinerdogan, Maurice Glandrup, and Mehmet Aksit. On composing separated concerns, composability and composition anomalies. In *Proceedings of the 2000 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications, OOPSLA 2000*, 2000.

[16] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6 :161–180, April 2010.

[17] Jean Bézivin, Mireille Blay, Mokrane Bouzhegoub, Jacky Estublier, Jean-Marie Favre, Sébastien Gérard, and Jean-Marc Jézéquel. Rapport de synthèse de l'AS CNRS sur le MDA. CNRS, November 2004.

[18] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. in proceedings of the 16th ieee international conference on automated software engineering, 2001.

[19] Jean Bezivin, Slimane Hammoudi, Denivaldo Lopes, and Frédéric Jouault. Applying mda approach for web service platform. In *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*, pages 58–70. IEEE, 2004.

[20] Gordon Blair, Robert B. France, and Nelly Bencomo. Models@ run.time. *Computer*, 42 :22–27, 2009.

[21] Xavier Blanc, Olivier Collaborateur. Salvatori, and Philippe Desfray. *MDA en action.* Architecte logiciel. Eyrolles, Paris, 2005.

[22] Daniel G. Bobrow, Richard P. Gabriel, and Jon L. White. CLOS in context : the shape of the design space. pages 29–61, 1993.

[23] Boudjemaa Boudaa, Olivier Camp, Slimane Hammoudi, and Mohammed Amine Chikh. Model-Driven development of Context-Aware services : Issues, techniques and review. In *International Conference on Information Technology and e-Services 2012 (ICITeS'2012)*, pages 160–167, Sousse, Tunisia, March 2012.

[24] Boudjemaa Boudaa, Slimane Hammoudi, Abdelkader Bouguessa, and Mohammed Amine Chikh. Supporting runtime adaptation of context-aware services. In *Proceedings of the 3rd International Conference on Context-Aware Systems and Applications, ICCASA 2014, Dubai, United Arab Emirates, October 7-9, 2014*, pages 24–30, 2014.

[25] Boudjemaa Boudaa, Slimane Hammoudi, Abdelkader Bouguessa, Leila A. Mebarki, and Mohammed Amine Chikh. On sustaining dynamic adaptation of context-aware services. *EAI Endorsed Trans. Context-aware Syst. & Appl.*, 2(3) :e4, 2015.

[26] Boudjemaa Boudaa, Slimane Hammoudi, and Mohammed Amine Chikh. ODM-Based modeling for User-Centered Context-Aware mobile applications. In *The 3rd International Conference on Information Technology and e-Services ICITeS'2013(ICITeS'2013)*, Sousse, Tunisia, March 2013.

[27] Boudjemaa Boudaa, Slimane Hammoudi, Leila Amel Mebarki, Abdelkader Bouguessa, and Mohammed Amine Chikh. An aspect-oriented model-driven approach for building adaptable context-aware service-based applications. *Science of Computer Programming*, pages –, 2016.

[28] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, September 2012.

[29] Gregor Broll, Heinrich Hussmann, George N. Prezerakos, Georgia Kapitsaki, and Stefano Salsano. Modeling context information for realizing simple mobile services. In *Proceedings of the 16th IST Mobile & Wireless Communications Summit*, Budapest,Hungary, July 1 - 5 2007.

[30] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications : From the laboratory to the marketplace. *IEEE Personal Communications*, 4(5) :58–64, 1997.

[31] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java : Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36(11-12) :1257–1284, September 2006.

[32] Cinzia Cappiello, Marco Comuzzi, Enrico Mussi, and Barbara Pernici. Context management for adaptive information systems. *Electronic Notes in Theoretical Computer Science*, 146(1) :69 – 84, 2006. Proceedings of the First International Workshop on Context for Web Services (CWS 2005)Context for Web Services 2005.

[33] Andrew Carton, Siobhan Clarke, Aline Senart, and Vinny Cahill. Aspect-oriented model-driven development for mobile context-aware computing. In *SEPCASE '07 : Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[34] Tarak Chaari, Dejene Ejigu, Frédérique Laforest, and Vasile-Marian Scuturici. A comprehensive approach to model and use context for adapting applications in pervasive environments. *Journal of Systems and Software*, 80(12) :1973 – 1992, 2007. Selected papers from the International Conference on Pervasive Services (ICPS 2006).

[35] B. Chandrasekaran, John R. Josephson, and Richard V. Benjamins. What are Ontologies and why do we need them ? *IEEE Intelligent Systems*, pages 20–26, January 1999.

[36] David Chappell. *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[37] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical report, Dept. of Computer Science, Dartmouth College, 2000.

[38] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18 :197–207, 2003.

[39] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. Soupa : Standard ontology for ubiquitous and pervasive applications. In *International Conference on Mobile and Ubiquitous Systems : Networking and Services*, pages 258–267, 2004.

[40] Benoît Combemale. *Metamodeling Approach for Model Simulation and Verification : Application to Process Engineering.* Theses, Institut National Polytechnique de Toulouse - INPT, July 2008.

[41] Patricia Dockhorn Costa, Luis Ferreira Pires, and Marten J. van Sinderen. Designing a configurable services platform for mobile context-aware applications. *International Journal of Pervasive Computing and Communications*, Vol. 1 :13–24, 2005.

[42] Pierre-Charles David, Thomas Ledoux, Thierry Coupaye, and Marc Léger. FPath and FScript : Language support for navigation and reliable reconfiguration of Fractal architectures. *Annales des Télécommunications*, 64(1-2) :45–63, Février 2009.

[43] Jos de Bruijn. Using Ontologies - Enabling Knowledge Sharing and Reuse on the Semantic Web. Technical Report DERI-2003-10-29, DERI, 2003.

[44] Cléver R. G. de Farias, Marcos M. Leite, Camilo Z. Calvi, Rodrigo M. Pessoa, and José G. Pereira Filho. A mof metamodel for the development of context-aware mobile applications. In *SAC '07 : Proceedings of the 2007 ACM symposium on Applied computing*, pages 947–952, New York, NY, USA, 2007. ACM.

[45] Jessie Dedecker, Tom Van Cutsem, Stijn Mostinckx, Theo D'Hondt, and Wolfgang De Meuter. Ambient-oriented programming in ambienttalk. In *ECOOP*, pages 230–254, 2006.

[46] A. K. Dey and G. D Abowd. Towards a better understanding of context and context-awareness. Technical Report git-gvu-99-22,, Institute of Technology, Georgia, June 1999.

[47] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction Journal*, 16(2) :97–166, 2001.

[48] Nick Drummond and Rob Shearer. The open world assumption : http ://www.cs.man.ac.uk/ drummond/presentations/owa.pdf. electronic, 2006.

[49] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming : Introduction. *Commun. ACM*, 44(10) :29–32, October 2001.

[50] Thomas Erl. *Service-Oriented Architecture : Concepts, Technology, and Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[51] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures.* PhD thesis, 2000. AAI9980887.

[52] R. France, I. Ray, G. Georg, and S. Ghosh. Aspect-oriented approach to early design modelling. In *IEE Proceedings - Software*, pages 173–185, 2004.

[53] Ernest Friedman-Hill. Java expert system shell (jess), 2013.

[54] Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Engineering and Ontology Development.* Springer Publishing Company, Incorporated, 2nd edition, 2009.

[55] Nektarios Georgalas and Daphne Economou. Model-driven development of pervasive applications using context and ontologies : An m-commerce case study. *The Electronic Journal for Emarging Tools and Applications, Special Issue Electronic Commerce in Pervasive Environments*, 3(1), 2009.

[56] Vincenzo Grassi and Andrea Sindico. Towards model driven design of service-based context-aware applications. In Alexander L. Wolf, editor, *Proceedings of the 2007 International Workshop on Engineering of Software Services for Pervasive Environments, ESSPE 2007*, pages 69–74, Dubrovnik, Croatia, September 2007. ACM.

[57] Phil Greenwood, Ruzanna Chitchyan, Dhouha Ayed, Vincent Girard-Reydet, Franck Fleurey, Vegard Dehlen, and Arnor Solberg. Modelling service requirements variability : The diva way. In *Service Engineering*, pages 55–84. Springer Vienna, 2011.

[58] Thomas R. Gruber. A translation approach to portable ontology specifications. *KNOWLEDGE ACQUISITION*, 5 :199–220, 1993.

[59] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6) :907–928, December 1995.

[60] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An ontology-based context model in intelligent environments. In *In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 270–275, 2004.

[61] N. Guarino. *Formal Ontology in Information Systems : Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy.* IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st edition, 1998.

[62] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, WICSA '01, pages 45–, Washington, DC, USA, 2001. IEEE Computer Society.

[63] George T. Heineman and William T. Councill, editors. *Component-based Software Engineering : Putting the Pieces Together.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[64] Karen Henricksen, Jadwiga Indulska, Ted McFadden, and Sasitharan Balasubramaniam. Middleware for distributed context-aware systems. In *Proceedings of the 2005 Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part I*, OTM'05, pages 846–863, Berlin, Heidelberg, 2005. Springer-Verlag.

[65] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL : A Semantic Web Rule Language Combining OWL and RuleML. Technical report, World Wide Web Consortium, May 2004.

[66] Jean-Marc Jézéquel. Model Driven Design and Aspect Weaving. *Journal of Software and Systems Modeling (SoSyM)*, 7(2) :209–218, 2008.

[67] Jean-Marc Jézéquel. Ingénierie Dirigé par les Modèles : du design-time au runtime. *Génie Logiciel - Ingéniérie dirigée par les modèles*, 2010.

[68] Jean-Marc Jézéquel, Benoit Combemale, and Didier Vojtisek. *Ingénierie Dirigée par les Modèles : des concepts à la pratique...* Références sciences. Ellipses, February 2012.

[69] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. Atl : a model transformation tool. *Science of computer programming*, 72(1-2) :31–39, June 2008.

[70] Frédéric Jouault and Jean Bézivin. Km3 : A dsl for metamodel specification. In *Proceedings of the 8th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems*, FMOODS'06, pages 171–185, Berlin, Heidelberg, 2006. Springer-Verlag.

[71] Georgia M. Kapitsaki, George N. Prezerakos, and Nikolaos D. Tselikas. *In book "Enabling Context-Aware Web Services : Methods, Architectures, and Technologies"*, chapter Context-aware Web Service Development : Methodologies and Approaches, pages 3–29. Chapman & Hall/CRC Press, 1st edition, 2010.

[72] Georgia M. Kapitsaki, George N. Prezerakos, Nikolaos D. Tselikas, and Iakovos S. Venieris. Context-aware service engineering : A survey. *Journal of Systems and Software*, 82(8) :1285–1297, 2009.

[73] Georgia M. Kapitsaki, George N. Prezerakos, Nikolaos D. Tselikas, and Iakovos S. Venieris. Model-driven development of composite context-aware web applications. *Information & Software Technology*, 51(8) :1244–1260, 2009.

[74] Markus Keidl and Alfons Kemper. Towards context-aware adaptable web services. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters, WWW 2004*, New York, NY, USA, May 17-20 2004. ACM.

[75] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, January 2003.

[76] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. Getting started with aspectj. *Communications of the ACM*, 44 :59–65, 2001.

[77] YOUNG-GAB KIM, SEOK KEE LEE, and SUNG-BONG JANG. Variability management for software product-line architecture development. *International Journal of Software Engineering and Knowledge Engineering*, 21(07) :931–956, 2011.

[78] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained : The Model Driven Architecture : Practice and Promise.* Addison-Wesley, 2003.

[79] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17, Part B :184 – 206, 2015. 10 years of Pervasive Computing' In Honor of Chatschik Bisdikian.

[80] Kewen Liao, Quan Z. Sheng, Jian Yu, and Hoi S. Wong. Smart adelaide guide : a context-aware web application. In *iiWAS '09 : Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, pages 681–687, New York, NY, USA, 2009. ACM.

[81] David C. Luckham. *The Power of Events : An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[82] Zakaria Maamar, Quan Z. Sheng, Samir Tata, Djamal Benslimane, and Mohamed Sellami. Towards an approach to sustain web services high-availability using communities of web services. *International Journal of Web Information Systems*, 5 :32–55, 2009.

[83] Zakaria Maamar, Sattanathan Subramanian, Philippe Thiran, Djamal Benslimane, and Jamal Bentahar. An approach to engineer communities of web services : Concepts, architecture, operation, and deployment. *IJEBR*, 5(4) :1–21, 2009.

[84] D.L. McGuinness and F. van Harmelen. Owl web ontology language overview, w3c recommendation, feb. 2004, <http ://www.w3.org/tr/owl-features/>. 2004.

[85] Brice Morin. *Leveraging Models from Design-time to Runtime to Support Dynamic Variability*. PhD thesis, Université Rennes 1, September 2010.

[86] Brice Morin, Olivier Barais, Jean-Marc Jezequel, Franck Fleurey, and Arnor Solberg. Models@ run.time to support dynamic adaptation. *Computer*, 42 :44–51, October 2009.

[87] Brice Morin, Franck Fleurey, Nelly Bencomo, Jean-Marc Jezequel, Arnor Solberg, Vegard Dehlen, and Gordon Blair. An aspect-oriented and model-driven approach for managing dynamic variability. In *In Proceedings of ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 08)*, Toulouse, France, October 2008.

[88] Pierre-Alain Muller, Franck Fleurey, and Jean marc Jezequel. Weaving executability into object-oriented meta-languages. In *in : International Conference on Model Driven Engineering Languages and Systems (MoDELS), LNCS 3713 (2005)*, pages 264–278. Springer, 2005.

[89] Thi Thanh Tam Nguyen. *Codèle : Une Approche de Composition de Modèles pour la Construction de Systèmes à Grande Échelle*. These, Université Joseph-Fourier - Grenoble I, December 2008.

[90] OMG. Unified modeling language (omg uml) infrastructure, v.2.3, <http ://www.omg.org/spec/uml/2.3 >. 2010.

[91] OMG. Model driven architecture (mda), mda guide revision 2.0, 2014.

[92] OMG. Object constraint language (ocl), version 2.4, 2014.

[93] OMG. Ontology definition metamodel (odm), version 1.1, 2014.

[94] OMG. Meta object facility (mof), version 2.5, 2015.

[95] OMG. Query/view/transformation (qvt), version 1.2, 2015.

[96] OMG. Unified modeling language (uml), version 2.5, 2015.

[97] OMG. Xml metadata interchangel (xmi), version 2.5.1, 2015.

[98] Shumao Ou, Nektarios Georgalas, Manooch Azmoodeh, Kun Yang, and Xiantang Sun. A model driven integration architecture for ontology-based context modelling and context-aware application development. In Arend Rensink and Jos Warmer, editors, *Model Driven Architecture-Foundations and Applications*, volume 4066 of *Lecture Notes in Computer Science*, pages 188–197. Springer Berlin / Heidelberg, 2006.

[99] George Papamarkos, Alexandra Poulovassilis, Ra Poulovassilis, and Peter T. Wood. Event-condition-action rule languages for the semantic web. In *In : Workshop on Semantic Web and Databases*, pages 309–327, 2003.

[100] Carlos Parra, Xavier Blanc, Anthony Cleve, and Laurence Duchien. Unifying design and runtime software adaptation using aspect models. *Science of Computer Programming*, 76(12) :1247–1260, January 2011.

[101] J. Pascoe. Adding generic contextual capabilities to wearable computers. In I. C. Press, editor, *2nd International Symposium on Wearable Computers*, pages 92–99, Los Alamitos, California, 1998.

[102] Davy Preuveneers, Jan Van Den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, E Berbers, Karin Coninx, and Koen De Bosschere. Towards an extensible context ontology for ambient intelligence. In *In : Proceedings of the Second European Symposium on Ambient Intelligence*, pages 148–159. Springer-Verlag, 2004.

[103] George N. Prezerakos, Nikolaos D. Tselikas, and Giovanni Cortese. Model-driven composition of context-aware web services using contextuml and aspects. In *Proceedings of 2007 IEEE International Conference on Web Services (ICWS 2007)*, pages 320–329, Salt Lake City, Utah, USA, July 9-13 2007. IEEE Computer Society.

[104] Raghu Reddy, Robert France, Franck Fleuery, and Benoit Baudry. B. : Model composition - a signature based approach. In *In : Proceedings Aspect Oriented Modeling workshop held with MODELS/UML 2005, Montego*, page 2, 2005.

[105] Gaëtan Rey, Jean-Yves Tigli, Stéphane Lavirotte, Nicolas Ferry, Sana Fathallah, Joëlle Coutaz, Emeric Fontaine, Fabrice Jouanot, Marie-Christine Rousset, Philippe Renevier, Anne-Marie Pinna-Déry, and Vincent Hourdin. Modélisation du contexte et Adaptation. Technical Report D2.1-2.2, ANR Continuum, August 2010.

[106] Enrico Rukzio, George N. Prezerakos, Giovanni Cortese, Eleftherios Koutsoloukas, and Sofia Kapellaki. Context for simplicity : A basis for context-aware systems based on the 3gpp generic user profile, international conference on computational intelligence, icci 2004, december 17-19, 2004, istanbul, turkey, proceedings. In *International Conference on Computational Intelligence*, pages 466–469. International Computational Intelligence Society, 2004.

[107] N. Ryan, Pascoe, J., and D Morse. Enhanced reality fieldwork : the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports. Tempus Reparatum, October 1997.

[108] Bill Schilit and Marvin Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8 :22–32, 1994.

[109] Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2), February 2006.

[110] Ed Seidewitz. What models mean. *IEEE Software*, 20(5) :26–32, 2003.

[111] Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni, and Jean-Bernard Stefani. Reconfigurable sca applications with the frascati platform. In *Proceedings of the 2009 IEEE International Conference on Services*

Computing, SCC '09, pages 268–275, Washington, DC, USA, 2009. IEEE Computer Society.

[112] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures. *Software : Practice and Experience*, 42(5) :559–583, May 2012.

[113] Quan Z. Sheng and Boualem Benatallah. Contextuml : A uml-based modeling language for model-driven development of context-aware web services. In *Proceedings of 2005 International Conference on Mobile Business (ICMB 2005)*, pages 206–212, Sydney, Australia, 11-13 July 2005.

[114] Quan Z. Sheng, Sam Pohlenz, Jian Yu, Hoi S. Wong, Anne Hee Hiong Ngu, and Zakaria Maamar. Contextserv : A platform for rapid and flexible development of context-aware web services. In *Proceedings of 31st International Conference on Software Engineering, ICSE 2009*, pages 619–622, Vancouver, Canada, May 16-24 2009. IEEE.

[115] Quan Z. Sheng, Jian Yu, and Schahram Dustdar. *Enabling Context-Aware Web Services : Methods, Architectures, and Technologies*. Chapman & Hall/CRC, 1st edition, 2010.

[116] Quan Z. Sheng, Jian Yu, Aviv Segev, and Kewen Liao. Techniques on developing context-aware web services. *IJWIS*, 6(3) :185–202, 2010.

[117] Christof Simons. Cmp : A uml context modeling profile for mobile distributed systems. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS '07)*, page 289, Hawaii, 2007. IEEE Computer Society.

[118] Hammoudi Slimane. Contribution à l'étude et à l'application de l'ingénierie dirigée par les modèles, HDR, December 2010.

[119] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF : Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[120] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *In : Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[121] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Cool : A context ontology language to enable contextual interoperability. In *LNCS 2893 : Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). Volume 2893 of Lecture Notes in Computer Science (LNCS)., Paris/France*, pages 236–247. Springer Verlag, 2003.

[122] Norbert Streitz and Paddy Nixon. Introduction. *Commun. ACM*, 48(3) :32–35, March 2005.

[123] Éric Tanter, Kris Gybels, Marcus Denker, and Alexandre Bergel. Context-Aware Aspects. In Welf Löwe and Mario Südholt, editors, *5th International Symposium on Software Composition (SC 2006)*, volume 4089 of *LNCS*, Vienna, Autriche, 2006. Springer.

[124] Hong Linh Truong and Schahram Dustdar. A survey on context-aware web service systems. *International Journal of Web Information Systems*, 5(1) :5–31, 2009.

[125] Samyr Vale and Slimane Hammoudi. Context-aware model driven development by parameterized transformation. In *Proceedings of the 1st International Workshop on the Model Driven Interoperability for Sustainable Information Systems (MDISIS'08) held in cunjunction with CAiSE'08*, Montpellier, France, June 2008.

[126] Samyr Vale and Slimane Hammoudi. Odm-based architecture for the development of mobile context-aware applications. In *Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '09, pages 9 :1–9 :4, New York, NY, USA, 2009. ACM.

[127] Jorge Vallejos, Peter Ebraert, Brecht Desmet, Tom Van Cutsem, Stijn Mostinckx, and Pascal Costanza. The context-dependent role model. In *Proceedings of the 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2007)*, Lecture Notes in Computer Science, Paphos, Cyprus, June 2007. Springer-Verlag.

[128] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0 :18, 2004.

[129] Mark Weiser. The computer for the 21st century. *Human-computer interaction : toward the year 2000*, 0 :933–940, 1991.

[130] Alan F. Westin. *Privacy and freedom*. Atheneum, New York, 1970.

[131] Manuel Wimmer, Andrea Schauerhuber, Gerti Kappel, Werner Retschitzegger, Wieland Schwinger, and Elizabeth Kapsammer. A survey on uml-based aspect-oriented design modeling. *ACM Comput. Surv.*, 43(4) :28 :1–28 :33, October 2011.

[132] Terry Winograd. Architectures for context. *Human-Computer Interactions*, 16(2) :401–419, 2001.

[133] Juan Ye, Lorcan Coyle, Simon Dobson, and Paddy Nixon. Ontology-based models in pervasive computing systems. *Knowl. Eng. Rev.*, 22 :315–347, December 2007.

[134] Juan Ye, Graeme Stevenson, and Simon Dobson. A top-level ontology for smart environments. *Pervasive and Mobile Computing*, 7(3) :359 – 378, 2011.

ملخص:

يعتبر تطوير التطبيقات المستندة إلى الخدمات الحساسة للسياق من بين المجالات البحثية الأكثر دراسة في العقد الماضي. وكان الهدف هو مرافقة التطور التكنولوجي السريع لأجهزة الكمبيوتر المحمول من خلال تقديم خدمات مخصصة قادرة على التفاعل مع الحالات السياقية المختلفة في بيئة واسعة الانتشار. لهذا الغرض، العديد من الأعمال البحثية قد دعت إلى التنمية الموجهة بالنماذج **(MDD)** لبناء التطبيقات المستندة إلى الخدمات الحساسة للسياق، إلا أن ما قدم وفق هذا النهج المقترح لا يتعدى منهجيات محددة خالية من إستخدام معايير التنمية العامة التي يمكن إتباعها من قبل المطورين. وأيضا، فقد تجاهل معظمهم جانب التكيف الحيوي في وقت التشغيل التي يجب أن يميز هذا النوع من التطبيقات حيث لم يتم تقديم أي استراتيجية للتكيف في مقترحاتهم. تهدف الأطروحة الحالية إلى اقتراح نهج عام قائم على النموذج من أجل هندسة التطبيقات الحساسة للسياق المستندة إلى الخدمات مع منهجية تطوير البرمجيات بما في ذلك حلقة إعادة التشكيل لتحقيق التكيف الحيوي لهذه التطبيقات. يركز هذا النهج على الجمع بين **MDD** و نمذجة الجانب المنحى **(AOM)** للاستفادة من فوائدهما. **AOM** يقوم بتغليف مختلف الحالات الحساسة للسياق بشكل منفصل في نماذج مستقلة تدعى **ContextAspect** ، والتي يمكن أن تنسج بسهولة في منطق الأعمال والخدمات وفقا لتغير السياق مع مرور الوقت. وتتضمن منهجية التطوير المقترح أربع مراحل (النمذجة، التركيب، والتحول والتكيف) التي تعمل كلها في توافق مع تكنولوجية **MDA**. النتائج الرئيسية التي يمكن اكتسابها باستخدام النهج المقترح هي إمكانية الجمع بين تكنولوجية **MDA** و مفهوم الجانب المنحى في منهجية تطوير عامة للتطبيقات الحساسة للسياق و القائمة على الخدمات، والتعامل مع تكيفها الديناميكي في وقت التنفيذ وفقا للتغيرات التي قد تطرأ على السياق. تطوير التطبيقات الحساسة للسياق هو مهمة معقدة، ومرهقة، وتستغرق وقتا طويلا. ومع ذلك، فإن التجربة التي توصلنا إليها بتنفيذ المنهجية المقترحة تقودنا إلى الاعتقاد بأن إشراك **MDD** و**AOM** هو مفيد كثيرا للتغلب على بعض أوجه القصور المعترف بها في المناهج الموجودة وجعل هذه المهمة أسهل وأسهل وأسرع.

الكلمات المفتاحية: تطبيق حساس للسياق و مستند إلى الخدمة؛ **ContextAspect**؛ منهجية موجهة بالنموذج؛ نسج الجانب؛ التكيف الحيوي.

Résumé

Le développement des applications sensibles au contexte à base de services a été considéré parmi les domaines de recherche les plus étudiés dans la dernière décennie. L'objectif était d'accompagner l'évolution rapide de la technologie des appareils informatiques mobiles en fournissant des services personnalisés capables d'interagir avec différentes situations contextuelles dans un environnement pervasive. A cet effet, de nombreux travaux de recherche ont préconisé le développement dirigé par les modèles (MDD) pour construire des applications sensibles au contexte à base de services. Cependant, les approches proposées ont présenté des méthodologies spécifiques sans utiliser des normes générales de développement qui peuvent être suivies par les développeurs. En outre, la plupart d'entre eux ont ignoré l'aspect d'adaptation dynamique à l'exécution qui doit caractériser ce type d'applications et aucune stratégie d'adaptation n'a été prise en compte dans leurs propositions. La présente thèse a pour but de proposer une approche générique dirigée par les modèles pour l'ingénierie des applications sensibles au contexte à base de services avec une méthodologie de développement de logiciels incluant une boucle de reconfiguration pour réaliser l'adaptation dynamique de ces applications. Cette approche met l'accent sur la combinaison de MDD et la modélisation orientée aspect (AOM) pour tirer parti de leurs avantages. AOM encapsule les différentes logiques de sensibilité au contexte séparément dans des modèles d'aspect appelé ContextAspect qui peuvent facilement être tissés dans la logique métier du service en fonction de l'évolution du contexte au fil du temps. La méthodologie de développement proposée comprend quatre phases (modélisation, composition, transformation et adaptation) qui agissent en conformité avec la technologie MDA. Les principaux résultats obtenus à l'aide de l'approche actuelle sont la possibilité de combiner la technologie MDA avec le paradigme orienté aspect dans une méthodologie de développement générique pour les applications sensibles au contexte à base de services, et le traitement de leur adaptation dynamique au moment de l'exécution en fonction de l'évolution du contexte. Le développement des applications sensibles au contexte est une tâche complexe, lourde, et qui prend du temps. Cependant, l'expérience atteinte en mettant en œuvre la méthodologie proposée nous amène à croire que la combinaison de MDD et AOM est nettement bénéfique pour surmonter certaines lacunes reconnues de plusieurs approches existantes et de rendre cette tâche plus simple, plus facile et plus rapide.

Mots clés : Application sensible au contexte à base de service; ContextAspect; méthodologie dirigée par les modèles; Tissage d'aspect; Adaptation dynamique.

Abstract

Context-aware service-based applications development has been considered among the most studied research fields in the last decade. The objective was to accompany the rapid technology evolution of mobile computing devices by providing customized services able to interact with different contextual situations of a pervasive environment. For this purpose, many research works have advocated Model-Driven Development (MDD) for building context-aware service-based applications. However, the proposed approaches have presented specific methodologies without using development standards, which may be followed by developers. In addition, most of them have ignored the dynamic adaptation aspect at runtime that should characterize such kind of applications and no adaptation strategy was considered in their proposals. This thesis aims to propose a generic model-driven approach for context-aware service-based applications engineering with a software development methodology including a reconfiguration loop to achieve the dynamic adaptation of these applications. This approach focuses on the combination of MDD and Aspect Oriented Modelling (AOM) to take advantage of their benefits. AOM encapsulates different context-awareness logics separately in aspect models called ContextAspect that can be easily woven into the service's business logic according to the changing context over time. The proposed development methodology includes four phases (modelling, composition, transformation and adaptation) which act in conformance with the MDA technology. The main results gained by using the present approach are the possibility to combine the MDA technology with the aspect-oriented paradigm in a generic development methodology for context-aware service-based applications, and the handling of their dynamic adaptation at execution time according to the changes in the context. The development of context-aware applications is a complex, cumbersome, and time-consuming task. However, the experience reached by implementing the proposed methodology leads us to believe that the involvement of MDD and AOM is significantly beneficial to overcome some recognised shortcomings of several existing approaches and to make this task simpler, easier and faster.

Keywords: Context-aware Service-based Application; ContextAspect; Model-Driven Methodology; Aspect Weaving; Dynamic Adaptation.