UNIVERSITÉ DU QUÉBEC À MONTRÉAL

UTILISATION DES PATRONS DE CONCEPTION DANS LE DÉVELOPPEMENT DES APPLICATIONS MOBILES DÉPENDANTES DU CONTEXTE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

BOUSSOFARA AMINE

JUILLET 2014

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.07-2011). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

REMERCIEMENTS

J'adresse tout d'abord mes remerciements les plus sincères à mon directeur de recherche, Monsieur Obaid Abdellatif, qui m'a donné l'opportunité de travailler sur ce projet de recherche. Je tiens à lui exprimer toute ma gratitude et ma profonde reconnaissance pour sa patience, sa disponibilité et ses précieux conseils. Ce fut un grand plaisir de travailler avec lui.

Je désire également remercier ma co-directrice Madame Ghizlane El Boussaidi professeure à l'ETS et membre du laboratoire *LATECE*, pour sa générosité, sa disponibilité et son suivi rigoureux.

J'adresse aussi ma totale reconnaissance envers les professeurs de l'UQAM qui m'ont permis d'avoir une formation de qualité.

À mes chers parents, qui m'ont donné l'opportunité de vivre cette magnifique expérience et qui m'ont soutenu tout au long de mes études.

Finalement, j'adresse mes remerciements aux membres du jury qui m'ont fait l'honneur de bien vouloir juger ce modeste travail.

TABLE DES MATIÈRES

LISTE DES TABLEAUX vii

LISTE DES FIGURES ix

RÉSUMÉ xiii

CHAPITRE I
INTRODUCTION 1

1.1 Problématique 1

1.2 Objectifs 3

1.3 Organisation du document 5

CHAPITRE II
LES APPLICATIONS SENSIBLES AU CONTEXTE 7

2.1 Introduction 7

2.2 Définition de la notion de contexte 7

2.3 La sensibilité au contexte 10

2.4 Acquisition du contexte 11

2.5 Modélisation du contexte 13

2.6 Conception des applications sensibles au contexte par application de patrons 17

2.6.1 Les patrons de conception 18

2.6.2 Utilisations des patrons dans les systèmes sensibles au contexte 22

2.7 L'ingénierie des lignes de produits logiciels appliquée aux applications dépendantes du contexte 28

2.7.1 Les lignes de produits logiciels 28

2.7.2 Framework pour LDP 29

2.7.2.1 L'ingénierie de domaine 30

2.7.2.2 L'ingénierie d'application 32

2.7.2.3 La variabilité 33

2.7.3 Utilisation des LDP dans les applications sensibles au contexte 34

2.8 Conclusion 38

CHAPITRE III
APPROCHE CONCEPTUELLE PROPOSÉE 39

3.1 Introduction 39

3.2 Vue générale de l'architecture proposée 40

3.3 Étude de cas 44

3.4 Modèle conceptuel de l'application cliente 45

3.5 Adaptation de l'application serveur au contexte 50

3.5.1 Phase initiale 53

3.5.2 Phase de reconfiguration 55

3.6 Conclusion 61

CHAPITRE IV
PLAN DE VALIDATION ET OUTILS 63

4.1 Introduction 63

4.2 Outils 63

4.3 Problématiques rencontrées 64

4.4 La mise en place du serveur d'application 65

4.5 Limites et perspectives 75

4.5.1 Limites 75

4.5.2 Perspectives 76

CONCLUSION 77

LISTE DES TABLEAUX

Tableau Page

2.1 Les patrons de conception de GOF [11]. 19

LISTE DES FIGURES

Figure		Page
2.1	Le contexte Toolkit [20].	12
2.2	Un message ContextML [20].	16
2.3	Patron Command [11].	20
2.4	Patron Observateur [11].	20
2.5	Patron État [11].	21
2.6	Patron Décorateur [11].	21
2.7	Les 45 pré-patrons proposés [7].	23
2.8	Modèles applicatif et spatial [26].	24
2.9	Définition des services [26].	25
2.10	L'approche Reverse architecting proposée par [25].	26
2.11	Les patrons proposés et leurs relations [25].	28
2.12	Ingénierie de domaine et Ingénierie d'application [24].	30
2.13	Diagramme de caractéristiques associé à l'ingénierie de domaine [23].	31
2.14	Exemple d'artefacts construit à partir du diagramme de caractéristiques [23].	32
2.15	Architecture Proposée [21].	35
2.16	Diagramme de caractéristiques associé à la deuxième approche [2].	36
3.1	Architecture d'une application mobile dépendante du contexte.	40
3.2	Les tâches effectuées du côté client.	42
3.3	Les tâches effectuées du côté serveur.	43

3.4 Modélisation des valeurs contextuelles associées à notre approche. 45

3.5 Structure du patron observateur associé à notre approche. 47

3.6 La relation entre le patron Observateur et Commande. 48

3.7 Modèle conceptuel de l'application cliente. 49

3.8 Diagramme de caractéristiques. 50

3.9 Diagramme d'activités correspondant à notre scénario. 52

3.10 Assemblage des composants avec l'architecture SCA. 53

3.11 Un assemblage de composants dans une architecture SCA. 54

3.12 Architecture de l'application serveur à la phase initiale. 55

3.13 Méta-modèle d'adaptation. 57

3.14 Assemblage des composants correspondant à la phase initiale. . . 57

3.15 Fpath et Fscript associés à notre scénario. 59

3.16 L'architecture SCA suite à l'adaptation. 60

3.17 Méta-modèle de l'application serveur. 61

4.1 Un exemple de structure associée à une application SCA. 65

4.2 la structure des composants MedicalServiceApp, EtatDuPatient et Localisation. 67

4.3 Capture d'écran - Exécution du serveur d'application avec Frascati. 68

4.4 Capture d'écran - Visualisation et reconfiguration de l'application avec Frascati Explorer. 68

4.5 Capture d'écran- Frascati Fscript. 69

4.6 Procédure d'adaptation avec Frascati Fscript. 70

4.7 Capture d'écran- L'interface graphique de simulation. 71

4.8 Capture d'écran- Un cas d'utilisation associé à notre scénario. . . 72

4.9 Structure du composant Frascati Fscript. 73

4.10 Un extrait de la trace d'exécution générée. 74

RÉSUMÉ

Le développement rapide des technologies de communication mobile suscite un intérêt de plus en plus croisant. Face à ce phénomène, de nouveaux domaines de recherche ont émergé pour faire face à cette nouvelle tendance de l'informatique qui est devenue mobile.

Dans ce travail, notre intérêt s'est tourné vers cette nouvelle tendance de l'informatique mobile et plus particulièrement vers ces nouvelles applications mobiles dites sensibles au contexte. Ces applications ont la capacité de s'adapter à l'état courant de l'environnement dans lequel se trouve l'usager. Elles bénéficient ainsi de l'information du contexte et des services qui sont présents dans ce contexte pour changer leurs comportements d'une façon dynamique. Pour faire face à ce défi, et compte tenu des limitations des solutions existantes, nous proposons une démarche plus générale pour concevoir des applications mobiles capables d'adapter leurs comportements au contexte de l'usager à travers les services qu'elles offrent.

Notre travail se divise en deux parties, une application cliente pour mobile qui gère et assure la dissémination des données du contexte vers les différents services offerts, et un serveur d'application dédié qui adapte ses services aux données reçues.

Mots-clés : informatique mobile, contexte, application dépendante du contexte, patron de conception, ligne de produit logiciel, SCA, Frascati.

CHAPITRE I

INTRODUCTION

1.1 Problématique

Les progrès rapides des technologies de communication sans fil et des dispositifs mobiles (capteurs, téléphones intelligents, GPS, etc.), ont permis la création d'environnements technologiques informatiques et de communication tout à fait intégrés [29].

L'omniprésence de ces appareils mobiles et des services qu'ils offrent permettent aux utilisateurs d'accéder à l'information et d'effectuer des transactions n'importe où, n'importe quand et à partir de n'importe quel terminal.

Face à ces avancées technologiques, de nouveaux domaines de recherche ont émergé pour faire face à cette nouvelle tendance de l'informatique qui est devenue mobile. En effet, l'informatique mobile a amené de nouvelles problématiques et défis découlant des contraintes liées aux plateformes mobiles. Ces contraintes incluent les ressources limitées en termes de mémoire et de batterie de ces plateformes. Cela inclut aussi les variations dans la qualité du réseau sans fil et certaines variations dans l'environnement d'utilisation de ces plateformes.

Ces nouvelles contraintes ont donné naissance à un nouveau type d'applications, nommé *Applications mobiles dépendantes du contexte*. Ce type d'applications ne se limite plus à exploiter les sources de données standards pour récupérer des informations pertinentes et les retourner sous forme de services à l'utilisateur. Mais elle doit être capable de prendre en compte le contexte de l'utilisateur et d'adapter son comportement (i.e., ses services) à la situation dans laquelle celui-ci se trouve. Le contexte de l'utilisateur peut inclure autant des informations reliées à sa localisation physique que des informations sur son état physiologique et émotionnel [29].

Dans ce domaine, la plupart des solutions existantes font de gros efforts dans la façon de capturer le contexte et le disséminer à l'application (e.x. [9, 13, 20]), mais sans donner de solution structurée et systématique sur la façon d'effectuer cette adaptation au contexte. En fait, aucune attention n'a été accordée à la façon de concevoir ces applications pour permettre cette adaptation. La majorité de ces applications sont souvent mises en œuvre de façon ad-hoc en généralisant le modèle client-serveur. Cette approche est très limitée car elle suscite l'utilisation de middleware auprès du serveur ou des paradigmes de programmation. Ce middleware est considéré comme un recours pour échanger, filtrer et stocker l'information dérivant du contexte.

De plus, ce type d'applications adopte le plus souvent un modèle d'adaptation basé sur l'utilisation des règles métier pour faire interagir l'application avec un changement contextuel. Ce qui rend difficile l'extension et la réutilisation de ces applications quand on désire inclure de nouveaux types de contexte une fois la phase de développement terminée. Par conséquent, chaque solution a tendance à utiliser des caractéristiques spécifiques et aucun mécanisme qui soit à la fois réutilisable et général n'a émergé.

La conception d'une application qui s'adapte au contexte pose plusieurs défis et soulève plusieurs questions de recherche. Parmi ces questions, nous citons les suivantes :

- Comment représenter les données du contexte dans un format qui puisse être analysé et traité par une application ?
- À quelle fréquence faut-il récupérer les données du contexte ?
- Comment combiner les données du contexte avec les données de l'application ?
- Quelle architecture et quels patrons de conception, communément utilisés dans la conception logicielle, sont les plus appropriés pour la conception d'une application dépendante du contexte ?
- Quelles technologies peuvent faciliter la mise en œuvre d'une telle architecture ?
- Quel processus suivre pour développer et mettre en œuvre une application dépendante du contexte ?

Dans le cadre de ce mémoire, nous nous intéressons aux trois dernières questions de cette liste qui sont reliées à l'architecture et le processus de développement d'une application dépendante du contexte.

1.2 Objectifs

L'objectif principal de notre travail est de proposer une démarche plus générale pour concevoir des applications mobiles capables d'adapter leurs comportements au contexte à travers les services qu'elles offrent. Le but de notre démarche est de pallier aux inconvénients des solutions spécifiques et proposer en contrepartie une solution plus commune permettant aux concepteurs et développeurs de réutiliser et partager leurs efforts de conception et de développement.

Les principaux objectifs de notre travail sont :

1. La proposition et la mise en œuvre d'un modèle conceptuel pour le traitement et la dissémination des données du contexte.
2. La conception et l'implémentation d'une architecture pour l'adaptation des services offerts par l'application aux données captées.
3. L'utilisation d'une approche plus structurée au niveau de la conception de l'application.

Comme nous ne disposons pas dans le cadre de ce projet de l'infrastructure nécessaire pour capter les données du contexte, nous supposons que la capture du contexte et l'interprétation des données captées sont déjà fournies. En fait, nous proposons de simuler directement la modélisation des données captées sous forme de fichiers XML conformes aux standards existants.

Pour atteindre nos objectifs, nous utilisons la notion de *patron de conception* pour mettre en place un modèle conceptuel pour la transmission des paramètres contextuels captés vers le fournisseur de services. L'utilisation des patrons de conception nous permet de proposer un modèle conceptuel facile à maintenir et à étendre pour inclure de nouvelles données caractérisant le contexte.

Pour ce qui est de l'adaptation des services du côté du fournisseur, nous appliquons une approche inspirée de l'*ingénierie des lignes de produits* (*Sofwtare Product Line*) pour modéliser les variations des données du contexte et leur associer une architecture qui supporte la reconfiguration dynamique des services. Pour mettre en œuvre notre proposition, nous utilisons l'architecture SCA (*Service Component Architecture*) qui fournit un modèle standard pour la création et la composition de services.

1.3 Organisation du document

Ce document est structuré comme suit.

Le chapitre II présente un état de l'art sur les applications sensibles au contexte. Nous présentons dans ce chapitre la notion de contexte et les différents travaux effectués dans ce domaine. Ensuite, nous exposons les différents modèles conceptuels proposés pour adapter une application au contexte de son utilisateur ainsi qu'une nouvelle pratique dite *Lignes de produits logiciels (Software Product Line)* pour gérer la variabilité induite par le changement continu des paramètres contextuels et de fournir un mécanisme de reconfiguration dynamique lors de l'exécution de l'application. Le chapitre III décrit l'approche que nous proposons pour adapter les services d'une application mobile aux changements contextuels et le chapitre IV présente le plan de validation de cette approche et expose les différents outils à utiliser. Enfin, le chapitre V présente nos conclusions et nos perspectives de travaux futurs.

CHAPITRE II

LES APPLICATIONS SENSIBLES AU CONTEXTE

2.1 Introduction

Dans ce chapitre, nous présentons un état de l'art sur les applications sensibles au contexte et sur les différents travaux proposés dans ce domaine. Nous commençons par présenter les différentes définitions de la notion de contexte en exposant les différentes visions. Ensuite, nous présentons la définition des systèmes dits *Sensibles au contexte* et comment exploiter les interactions entre ces systèmes et leurs utilisateurs. Nous analysons aussi les techniques fournies pour acquérir un contexte et l'interpréter et les différentes approches proposées pour modéliser le contexte. Nous présentons enfin des approches qui ont été utilisées pour la conception et la mise en œuvre d'applications dépendantes du contexte.

2.2 Définition de la notion de contexte

Plusieurs définitions de la notion de *Contexte* sont proposées dans la littérature. Leurs utilisations diffèrent d'un domaine de recherche à un autre. Par exemple, dans le domaine de l'intelligence artificielle, la notion de contexte apparait comme un moyen de partitionnement d'une base de connaissances sous forme d'ensembles gérables pour faciliter les activités de raisonnement [19].

Dans les premiers travaux où la notion de contexte a été introduite, on trouve la définition de Schilit et Theimer [30] :

"La localisation de l'utilisateur, l'identité des personnes et des objets qui l'entourent, et les modifications à apporter à ces objets ".

Dans cette définition, le contexte a été défini comme étant un environnement d'exécution en constante évolution où il faut répondre aux questions suivantes : «Où êtes-vous ?», «Qui êtes-vous ?» et «Quelles sont les ressources à proximité ? » afin d'identifier les aspects importants du contexte. Ainsi l'environnement qui définit le contexte est composé de [30] [1] :

- Un environnement informatique : les processeurs disponibles, les dispositifs accessibles pour l'entrée de l'utilisateur et de l'affichage, la capacité du réseau, la connectivité et les coûts de l'informatique.
- Un environnement utilisateur : la localisation, les personnes à proximité, la situation sociale.
- Un environnement physique : l'éclairage et le niveau du bruit.

Dans une définition similaire Brown et al [4] définissent le contexte comme étant :

"L'identité des personnes qui entourent l'utilisateur, sa localisation, l'heure, la température, la saison... ".

Contrairement à la première définition, celle-ci a limité la notion de contexte à l'environnement de l'utilisateur en introduisant l'heure, la température, l'identité et la localisation [4] [1].

En fait, les premiers travaux ont réduit la notion de contexte à l'utilisateur et à son environnement. Ceci a poussé les chercheurs dans des travaux plus récents à proposer une définition plus générale et plus claire.

Dans ce sens, Pascoe est l'un des premiers à avoir généralisé la notion de contexte en proposant la définition suivante [22] [5] :

"Le contexte est un sous-ensemble des états physiques et conceptuels ayant un intérêt pour une entité particulière".

Par la suite, plusieurs précisions à cette définition ont été apportées comme celle de Dey [1] [5] :

" Toute information pouvant être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un lieu ou un objet qui peut être pertinente pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application elle-même ".

Cette définition est considérée comme la plus complète et la plus adoptée par les chercheurs.

Néanmoins, Chaari et al [5] considèrent que cette dernière ne permet pas de séparer les données appartenant au contexte de celles de l'application. Selon eux, cette séparation est importante dans la conception d'un système sensible au contexte. De ce fait, ils ont apporté plus de précisions à la définition de Dey :

"Le contexte est l'ensemble des paramètres externes à l'application qui peuvent influer sur son comportement en définissant de nouvelles vues sur ses données et ses services".

Nous avons constaté que toutes les définitions existantes sont soit très abstraites soit très spécifiques à un domaine particulier. Cependant, cette dernière définition est la plus appropriée pour notre travail puisqu'elle permet de distinguer les données appartenant au contexte de celles de l'application. Ce qui facilite la formalisation et l'interprétation des données contextuelles.

2.3 La sensibilité au contexte

La sensibilité au contexte vise à exploiter les interactions homme-machine en fournissant des applications qui prennent en compte le contexte de l'utilisateur. De ce fait, un système dit sensible au contexte est un système qui peut réagir aux changements de son environnement [21].

La première définition de la notion de sensibilité au contexte a été introduite comme étant la capacité d'une application mobile à découvrir et à réagir aux changements dans l'environnement qui entoure l'utilisateur [30]. Plusieurs définitions de cette notion ont suivi par la suite. Elles ont divisé la sensibilité au contexte en fonction de deux aspects : l'utilisation du contexte et l'adaptation au contexte.

À titre d'exemple, Pascoe [22] définit la sensibilité au contexte comme étant la capacité des appareils informatiques de détecter, sentir, interpréter et réagir à certains aspects de l'environnement local de l'utilisateur ainsi qu'aux périphériques informatiques eux-mêmes. Tandis que, Abowd et al [1] considèrent les applications sensibles au contexte comme des applications qui adaptent leurs comportements d'une façon dynamique dans le temps en fonction du contexte, de l'application et de l'utilisateur.

Plus généralement, nous pouvons dire : *un système sensible au contexte est un système dont le comportement varie en fonction des informations contextuelles collectées.* En d'autres termes, c'est un système capable de détecter les éléments du contexte qui entoure l'utilisateur, de les interpréter et de fournir comme retour à l'utilisateur un ou plusieurs services adaptés aux éléments de ce contexte. Dans ce sens, nous considérons que la définition de Chaari et al [5] est la plus convenable pour utiliser efficacement le contexte dans un système informatique, et ce dans le cadre de notre travail :

"La sensibilité au contexte est la capacité d'un système à percevoir la situation dans laquelle se trouve l'utilisateur et d'adapter en conséquence le comportement du système en termes de service, de données et d'interface".

2.4 Acquisition du contexte

Dépendamment du domaine de l'application, il existe plusieurs moyens de capturer les informations contextuelles nécessaires au fonctionnement d'un système sensible au contexte. Parmi ces moyens, on distingue [20] :

- Contexte capté ou détecté : Ce type d'information est acquis moyennant des capteurs physiques ou logiciels tels que des capteurs de température, de pression atmosphérique, de lumière ou du niveau de bruit.
- Contexte dérivé : Ce type de contexte est calculé durant l'exécution d'un système, l'exemple le plus illustratif est celui de l'heure et de la date.
- Contexte explicitement fourni : C'est lorsque l'utilisateur communique explicitement ses préférences au système sous forme d'information.

Cependant, l'acquisition du contexte à partir des capteurs n'est pas un processus facile, ceci est dû à plusieurs raisons [20] :

- L'information peut être acquise à partir de différents capteurs et nécessite une étape supplémentaire d'interprétation avant d'être utilisée par l'application. De plus, les informations sont généralement de nature dynamique, ce qui nécessite des outils spécifiques pour capturer les changements dans les données captées.
- Les informations contextuelles proviennent de sources hétérogènes et distribuées. Ainsi, les technologies de détection utilisées sont généralement déterminées par des méthodes bien spécifiques. Ce qui entraine des mauvaises pratiques d'un point de vue génie logiciel car cela empêche la réuti-

lisation du code de l'application lorsque les capteurs changent.

Pour contourner ce problème, un ensemble d'architectures ont été proposées dans la littérature. Parmi les solutions proposées, on distingue celle élaborée par Dey appelée *Contexte Toolkit*. Cette dernière est considérée comme étant l'une des premières architectures et l'une des plus intéressantes dans la conception d'une solution réutilisable pour l'acquisition du contexte [9] [20].

Dans le Contexte Toolkit, une application est exécutée en trois étapes :

1. Le contexte est capturé grâce à des capteurs.
2. Les informations captées sont interprétées afin de les rendre plus exploitables pour l'application.
3. Les informations interprétées sont fournies à l'application.

La figure 2.1 illustre l'architecture proposée.

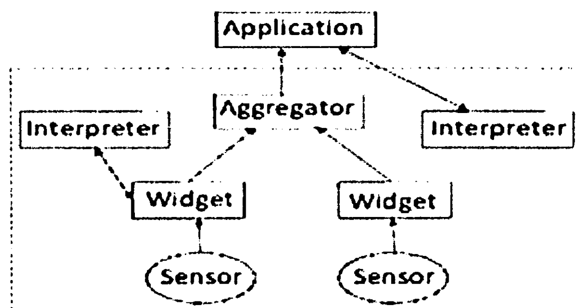

Figure 2.1 Le contexte Toolkit [20].

Cette architecture est composée d'un ensemble de composants abstraits : les widgets, les interpréteurs et un agrégateur. Ils favorisent l'acquisition des données contextuelles provenant des capteurs et les transforment en informations contextuelles de haut niveau.

Ces trois composants sont :

- Widget : C'est un composant logiciel qui permet l'accès aux informations contextuelles à travers une interface uniforme. Il sert aussi à masquer la complexité des capteurs et fournit en retour un bloc d'éléments réutilisable pour la capture du contexte.
- Interpréteur : Il est responsable de traduire le contexte capté sous une forme utile et prête à être utilisée par l'application. Par exemple, un interpréteur peut traduire les coordonnées GPS fournies par les widgets sous forme d'un nom de rue.
- Agrégateur : permet la collecte des informations contextuelles à partir des divers widgets existants et fournit à l'application une vue d'ensemble de toutes les données collectées. Il peut être aussi responsable de la transmission d'un contexte associé à une entité particulière (personne, objet ou lieu).

Cependant, cette architecture présente des limites. Sa faiblesse principale est le non-soutien continu de l'information contextuelle, ce qui a poussé les chercheurs à proposer de nouvelles solutions. Une de ces solutions consiste à stocker les informations contextuelles pour garder une trace de l'historique des valeurs capturées et cela avant de les transmettre à l'application. Ceci a donné naissance à un nouveau besoin, celui de la modélisation du contexte.

Dans la section suivante, nous détaillons les principales approches de modélisation proposées dans la littérature.

2.5 Modélisation du contexte

L'objectif de la modélisation est de faciliter l'utilisation du contexte au niveau de l'application. Un tel processus se traduit par la mise en place d'un modèle

qui fournit un haut niveau d'abstraction des informations contextuelles collectées.

Cependant, certains aspects liés à la nature de l'information contextuelle doivent être pris en considération lors de la définition d'un modèle. Dans [13], on définit un certain nombre d'observations sur la nature de l'information contextuelle qui déterminent les exigences de conception lors de la modélisation :

- Les informations contextuelles sont temporelles : le contexte peut être classé selon deux aspects, statique et dynamique.
  1. Une information contextuelle statique décrit les aspects invariants du contexte dans le temps, tels que par exemple, le nom, prénom et la date de naissance d'une personne.
  2. Un contexte avec un aspect dynamique influe sur la façon de collecter les informations puisque les applications dites sensibles au contexte sont de plus en plus intéressées par l'état actuel du contexte dans le temps.
- Les informations contextuelles présentent des défauts : les environnements informatiques ubiquitaires sont très dynamiques ; ce qui signifie que les informations partagées entre les dispositifs peuvent rapidement devenir obsolètes. En effet, les informations collectées nécessitent un traitement supplémentaire avant de les transmettre aux consommateurs ; ce qui peut entrainer d'importants retards entre la production et l'utilisation de l'information contextuelle. Un autre problème important est que les producteurs de contexte tels que les capteurs peuvent fournir une information erronée.
- Les informations contextuelles sont fortement interdépendantes : plusieurs liens existent entre les informations contextuelles. Certaines informations peuvent être liées entre elles par des règles de dérivation qui décrivent la façon dont l'information est obtenue à partir d'un ou plusieurs éléments

d'informations intermédiaires. En conséquence, le concept de modélisation proposé doit prendre en compte de tels liens.

- L'existence de plusieurs représentations alternatives du contexte : la plupart des informations contextuelles impliquées dans les systèmes ubiquitaires sont dérivées de capteurs. Il existe un écart important entre les informations captées et celles utiles pour les applications. Un tel écart doit être comblé par différents types de traitement spécifiques. En conséquence, un modèle contextuel doit prendre en charge de multiples représentations d'un même contexte sous différentes formes et à différents niveaux d'abstraction et doit être en mesure de saisir les relations qui existent entre les autres représentations.

Compte tenu de ces caractéristiques, il est plus que nécessaire de définir un modèle contextuel qui supporte les différents types d'informations contextuelles collectées.

Dans la littérature, plusieurs modèles sont proposés. Dans [31], les modèles les plus intéressants ont été présentés. Parmi ces modèles, nous distinguons trois approches de modélisation du contexte :

1. Utiliser un ensemble de paires (clé, valeur) pour stocker le contexte. Cette approche est fréquemment utilisée dans le cadre des services distribués. Ces services sont généralement décrits avec une liste d'attributs simples suivant le modèle clé-valeur et la procédure de découverte des services utilisés se basant sur des algorithmes d'appariement.

2. Utiliser un modèle adapté d'XML nommé ContextML [28] : C'est un protocole basé sur XML et utilisé comme format standard pour échanger des informations contextuelles entre un serveur et un client mobile. Les messages ContextML sont regroupés dans une balise nommée *<context>*. Un contexte décrit certains aspects de l'environnement actuel du client, et peut

inclure d'autres variables tels que l'emplacement, la vitesse et les détails de personnes impliquées dans un contexte donné. En plus de ces aspects, on peut définir un ensemble de contraintes que le serveur devrait appliquer dans le choix des informations à envoyer. Pour cela une balise nommée *<require>* spécifie au serveur le contexte requis.

L'exemple ci-dessous illustre un message envoyé par un client pour indiquer son emplacement et qui nécessite une note supplémentaire contenant le nom et la valeur de cette note contenue dans la balise *require*.

```
<context session="123" action="update">
<spatial proj="UTM" zone="33"
    datum="Euro 1950 (mean)">
<point x="281993" y="4686790" z="205" />>
</spatial>
<require>
<note>
<data name="landuse" value="pasture" />
</note>
</require>
</context>
```

Figure 2.2 Un message ContextML [20].

3. La troisième approche utilise les ontologies pour modéliser le contexte. Une ontologie permet de représenter les concepts et leurs interrelations. Elles sont particulièrement adaptées pour projeter une partie de l'information utilisée sous forme d'une structure de données utilisable par les ordinateurs.

Plusieurs autres modèles ont été proposés, parmi lesquels on peut citer les graphes contextuels et les modèles orientés objet [20][31].

2.6 Conception des applications sensibles au contexte par application de patrons

Bien que la représentation et l'acquisition du contexte soient bien étudiées dans la littérature, l'adaptation quant à elle est généralement conçue en utilisant des pratiques ad-hoc. La plupart des travaux effectués sur ce type d'applications adaptent une infrastructure centrée plutôt qu'une vue conceptuelle pour la gestion et la diffusion des informations contextuelles collectées [12] [27].

Une fois les informations contextuelles acquises et transformées, elles prennent la forme des données associées à l'application. La plupart des applications qui adoptent de telles approches utilisent des règles d'adaptation basées sur une structure de : « si-alors-sinon », qui selon les valeurs contextuelles captées décident quelle action doit être effectuée. Toutefois, de telles approches restent limitées bien qu'elles soient faciles à programmer [26].

Plus récemment, on trouve de plus en plus de travaux dans le domaine des *systèmes sensibles au contexte* qui se basent sur la notion de *patron* pour formuler des solutions communes à plusieurs types de problèmes. En effet, pour faire face à la complexité de plus en plus croissante des systèmes informatiques sensibles au contexte, plusieurs architectures ont été proposées pour adapter dynamiquement le comportement de ces systèmes aux changements de leurs environnements. Toutefois, ces architectures restent limitées, et chacune des solutions proposées a tendance à avoir des caractéristiques distinctives et aucun mécanisme réutilisable n'a vu le jour [25].

Dans le reste de cette section, nous présentons d'abord la notion de patron de conception et quelques exemples de patrons avant d'aborder quelques approches qui ont utilisé les patrons dans les applications dépendantes du contexte.

2.6.1 Les patrons de conception

Plusieurs travaux se sont intéressés à l'étude et la représentation des patrons. Parmi les définitions proposées d'un patron, nous citons celle de Johnson dans [14] :

"Un patron décrit un problème à résoudre, une solution, et le contexte dans lequel cette solution fonctionne. Il nomme une technique et décrit ses coûts et avantages. Il permet aux développeurs d'utiliser un vocabulaire commun pour concevoir leurs modèles".

Gamma et al [11] proposent plusieurs patrons de conception classés selon leurs rôles et leurs domaines d'applications. Ces patrons sont classés selon 3 types qui sont : a) les patrons de création, b) les patrons structurels et c) les patrons de comportement. Le tableau 2.1 illustre ces types de patrons.

Une présentation uniforme et structurée a été adoptée pour décrire et documenter de la même façon tous les patrons proposés. Parmi les propriétés utilisées pour structurer chaque patron nous citons :

- Nom : permet de faire référence à une famille de problèmes, une famille de solutions et leur impact.
- Intention : décrit le rôle du patron, son but et quel problème particulier il résout.
- Motivation : donne un problème de conception pouvant être résolu.
- Indication d'utilisation : décrit des situations pratiques où on peut utiliser le patron.
- Conséquence : décrit les conséquences positives et négatives de l'utilisation du patron.

Patrons de création	Patrons structurels	Patrons de comportement
Fabrique Abstraite	Adaptateur	Interpreteur
Moniteur	Pont	Chaine de responsabilités
Fabrication	Composite	Commande
Prototype	Decorateur	Iterateur
Singleton	Facade	Mediateur
	Poids mouche	Memento
	Procuration	Observateur
		Etat
		Stratégie
		Patron de methode
		Visiteur

Tableau 2.1 Les patrons de conception de GOF [11].

Nous décrivons brièvement dans ce qui suit quelques exemples des patrons de conception. Nous avons choisi de présenter ceux que nous pouvons exploiter dans notre approche.

- **Patron Commande**

  Ce patron est de type *Comportemental.* Il sert à encapsuler une requête comme un objet, permettant de découpler l'invocateur et le receveur des requêtes. Un objet *Commande* sert à communiquer une action à effectuer, ainsi que les arguments requis [11].

  Ce patron est représenté par le diagramme de classes suivant :

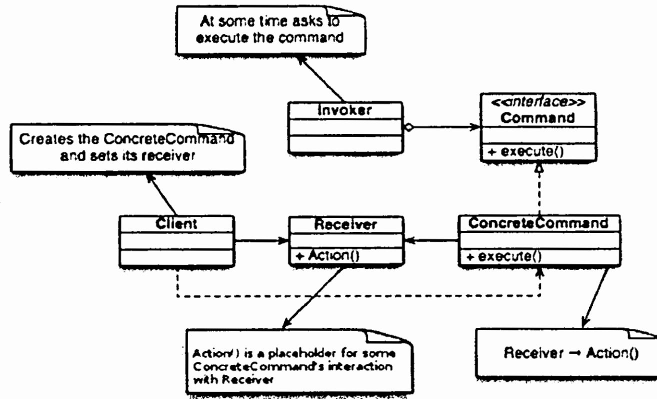

Figure 2.3 Patron Command [11].

- **Patron Observateur**

  Ce patron est de type *Comportemental.* Il a pour objectif de construire une dépendance entre un sujet et des observateurs de sorte que chaque modification associée à un sujet soit notifiée aux observateurs afin qu'ils puissent mettre à jour leurs états [11].

  Le patron Observer est représenté par le diagramme de classe suivant :

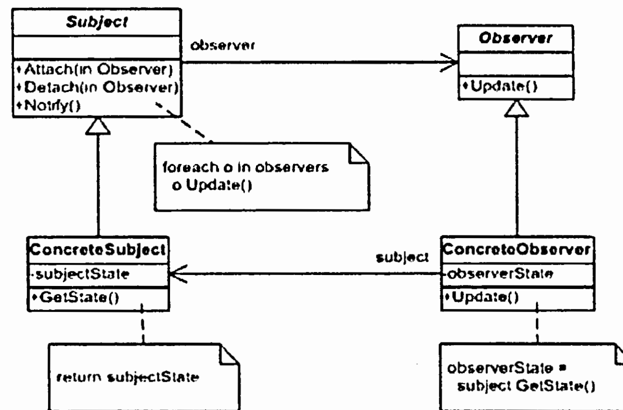

Figure 2.4 Patron Observateur [11].

- **Patron État**

  Ce patron est de type *Comportemental.* Il permet à un objet de changer son comportement en fonction de son état. L'objet se comportera comme s'il avait changé de classe [11]. Il est représenté par le diagramme de classe suivant :

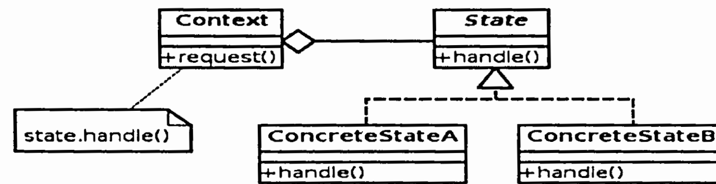

Figure 2.5 Patron État [11].

- **Patron Décorateur**

  Ce patron est de type *structurel.* Le but de ce patron est d'ajouter dynamiquement des fonctionnalités supplémentaires à un objet. Cet ajout ne modifie pas l'interface de l'objet et reste donc transparent vis-à-vis des clients. Il fournit ainsi une alternative flexible à l'héritage pour étendre les fonctionnalités d'un objet [11]. Ce patron est représenté par le diagramme de classe suivant :

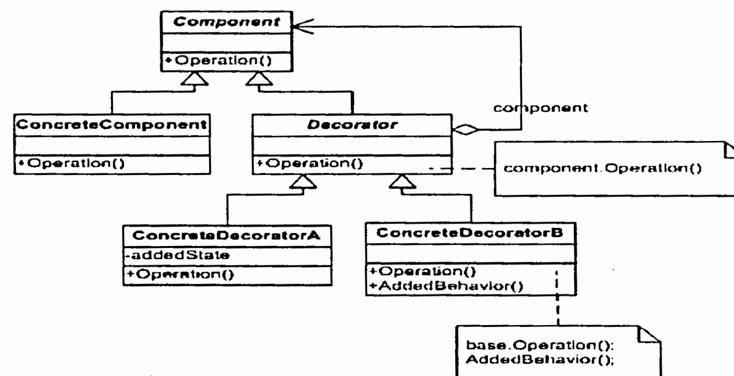

Figure 2.6 Patron Décorateur [11].

2.6.2 Utilisations des patrons dans les systèmes sensibles au contexte

Il existe peu de travaux qui associent les patrons de conception et les applications sensibles au contexte. Landay et Borriello [16] font partie des premiers chercheurs à avoir souligné l'importance des patrons de conception comme moyen efficace pour partager des solutions communes à des problèmes de conception dans les systèmes informatiques ubiquitaires. L'idée consiste à trouver des solutions communes à des problèmes donnés à partir des modèles qui ont été utilisés avec succès dans des systèmes réels.

Cette approche a été développée par la suite dans [7]. Ainsi, 45 pré-patrons ont été définis et classés en fonction de quatre critères :

- Les types d'applications : ce groupe décrit les larges groupes d'utilisations de l'informatique ubiquitaire ;
- Les espaces physiques et virtuels : ce critère est lié à la façon dont les objets et les espaces physiques peuvent être fusionnés ;
- Gestion de la vie privée : ce groupe décrit les politiques et les mécanismes de gestion de la vie privée de l'utilisateur final ;
- Les interactions : ce groupe décrit les techniques d'interaction avec les capteurs et les dispositifs.

La figure 2.7 illustre les 45 pré-patrons de conception destinées aux applications ubiquitaires. Pour valider ces pré-patrons, les auteurs ont mis en place un processus d'évaluation basé sur un ensemble de tests pour évaluer l'efficacité de chaque pré-patron proposé. Pour cela, ils ont fait appel à un groupe de concepteurs composé de professionnels et d'étudiants pour tester chaque pré-patron sur des cas pratiques et noter leurs observations.

A – Ubiquitous Computing Genres	B – Physical-Virtual Spaces	C – Developing Successful Privacy	D – Designing Fluid Interactions
Describes broad classes of emerging applications, providing many examples and ideas	Associating physical objects and spaces with information and meaning; location-based services; helping users navigate such spaces	Policy, systems, and interaction issues in designing privacy-sensitive systems	How to design for interactions involving dozens or even hundreds of sensors and devices while making users feel like they are in control
Upfront Value Proposition (A1) Personal Ubiquitous Computing (A2) Ubiquitous Computing for Groups (A3) Ubiquitous Computing for Places (A4) Guides for Exploration and Navigation (A5) Enhanced Emergency Response (A6) Personal Memory Aids (A7) Smart Homes (A8) Enhanced Educational Experiences (A9) Augmented Reality Games (A10) Streamlining Business Operations (A11) Enabling Mobile Commerce (A12)	Active Map (B1) Topical Information (B2) Successful Experience Capture (B3) User-Created Content (B4) Find a Place (B5) Find a Friend (B6) Notifier (B7)	Fair Information Practices (C1) Respecting Social Organizations (C2) Building Trust and Credibility (C3) Reasonable Level of Control (C4) Appropriate Privacy Feedback (C5) Privacy-Sensitive Architectures (C6) Partial Identification (C7) Physical Privacy Zones (C8) Blurred Personal Data (C9) Limited Access to Personal Data (C10) Invisible Mode (C11) Limited Data Retention (C12) Notification on Access of Personal Data (C13) Privacy Mirrors (C14) Keeping Personal Data on Personal Devices (C15)	Scale of Interaction (D1) Sensemaking of Services and Devices (D2) Streamlining Repetitive Tasks (D3) Keeping Users in Control (D4) Serendipity in Exploration (D5) Context-Sensitive I/O (D6) Active Teaching (D7) Resolving Ambiguity (D8) Ambient Displays (D9) Follow-me Displays (D10) Pick and Drop (D11)

Figure 2.7 Les 45 pré-patrons proposés [7].

Dans [27], on définit une approche de type conception modulaire pour construire des applications dépendantes du contexte. L'approche proposée consiste à identifier un ensemble de micro-architectures qui divisent l'espace de conception en deux modèles : un modèle applicatif et un modèle spatial qui fournissent les mécanismes d'adaptation pour les contextes existants, définis à base d'objets. Pour valider leur approche, les auteurs proposent comme scénario un logiciel existant qui manipule les informations sur un campus d'une université. Comme modifications, ils proposent de rajouter d'autres services qui dépendent du contexte spatial de l'utilisateur. Ainsi, grâce à des dispositifs mobiles, les utilisateurs peuvent récupérer, lors de leurs déplacements dans le campus, des informations relatives aux

programmes offerts, cours, professeurs, horaires, etc.

Le modèle applicatif correspond aux classes de l'application qui représentent les endroits dans le campus et qui constituent des contextes associés à la position géographique de l'utilisateur. La figure 2.8 illustre les modèles applicatif et spatial de l'application. Ce modèle permet d'établir une correspondance entre l'endroit où se trouvent l'utilisateur et l'objet spécifique de l'application. Pour réaliser une telle correspondance, les auteurs proposent d'enrichir leur modèle avec des décorateurs nommés *GeoObjet*. Les différents *GeoObjet* proposés sont reliés entre eux par des relations de contenant/contenu comme illustré dans la figure 2.8.

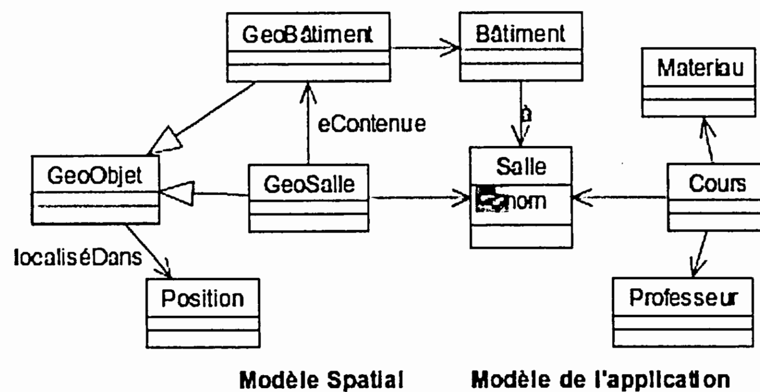

Figure 2.8 Modèles applicatif et spatial [26].

Une fois le modèle applicatif défini, la deuxième phase consiste à fournir un mécanisme d'adaptation de l'application en fonction du contexte existant. Pour cela, les services fournis comme réponse au contexte de localisation sont à leur tour définis comme étant des objets. Le patron de conception utilisé pour définir les services comme des objets connus par les *GeoObjet* qui les fournissent est le patron *Command* (voir figure 2.9).

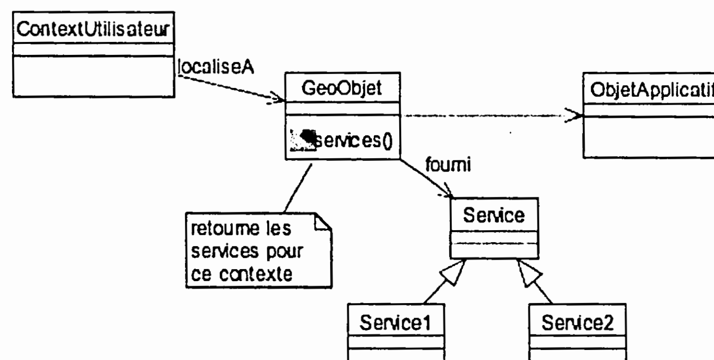

Figure 2.9 Définition des services [26].

Chaque objet dans le modèle d'adaptation répond au message *Service* provenant de l'application en retournant un objet applicatif contenant seulement les services qui correspondent à la position de l'utilisateur.

Dans [25], on propose une approche basée sur la notion *Reverse Architecting* appliquée sur des architectures de systèmes sensibles au contexte existants dont le but est de déterminer des patrons de conception implicitement adoptés pour résoudre des problèmes similaires. La notion de *Reverse Architecting* est le processus d'analyse de plusieurs logiciels pour identifier leurs composants, leurs relations et de créer des représentations du système sous une autre forme ou à un niveau plus élevé d'abstraction [15][6].

L'approche proposée se compose de trois tâches, à savoir la rétro-ingénierie, l'identification des sous-systèmes et la découverte des patrons de conception adéquats pour le système. Durant la phase de rétro-ingénierie, un modèle architectural global du système est recouvert. Ensuite, la phase d'identification des sous-systèmes permet d'identifier un groupe de composants qui répondent à une question spécifique, cette identification inclut aussi les relations existantes entre

ces composants. Le résultat de cette phase est un nouveau modèle architectural réorganisé comprenant les sous-systèmes identifiés.

Une fois le modèle architectural fourni, ce dernier est exploité afin d'identifier de nouveaux patrons de conception ou bien pour utiliser des patrons existants, notamment ceux de *GOF*. La figure 2.10 illustre les différentes tâches proposées avec plus de détails.

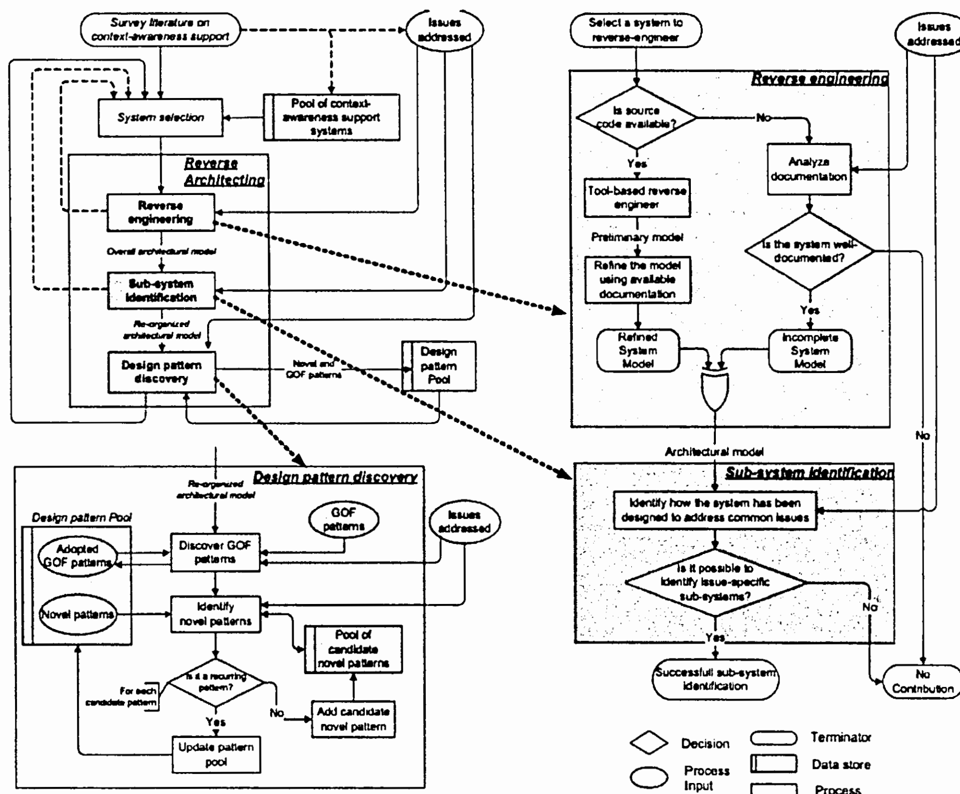

Figure 2.10 L'approche Reverse architecting proposée par [25].

Parmi les patrons de conception identifiés, on trouve le patron *Flyweight*, *Strategy*, *Observer* et *Mediator*. Les deux premiers patrons servent à partager et à gérer les données contextuelles captées à partir des capteurs. Le patron *Flyweight* sert à

stocker les valeurs des paramètres contextuels bruts et de les partager par la suite. Cependant, il ne gère pas l'interprétation des données, c'est plutôt l'application qui détermine les informations associées à ces valeurs (par exemple le numéro de la rue, la ville, etc.). Le patron *Strategy* définit un ensemble de stratégies basées sur des règles pour déterminer quel type d'action il faut déclencher lorsque certaines conditions sur les paramètres contextuels sont vérifiées.

Les deux autres patrons, *Observer* et *Mediator* servent à détecter tout changement dans les paramètres contextuels et assurer la dépendance entre ces paramètres et les composants de l'application. Le patron *Observer* se charge de notifier tous changements dans les paramètres contextuels qu'il observe aux différents composants de l'application qui jouent le rôle d'observateurs. Toutefois, ce type de dépendance peut devenir très complexe car le type et le nombre de sujets peuvent changer au fil du temps, ainsi que le nombre d'observateurs qui peuvent croitre de manière significative. Pour contourner ce problème, le patron *Mediator* a été utilisé pour réduire la prolifération des interconnexions en découplant les sujets et les observateurs.

Néanmoins, les patrons de conception proposés par *GOF* restent insuffisants. Par conséquent, la mise en place de nouveaux patrons de conception était nécessaire pour réussir la dernière phase. Les nouveaux patrons proposés sont *Flexible Context Processing* et *Encator*. Ces derniers adoptent le même formalisme proposé par *GOF* pour définir chaque patron. Ainsi, le premier patron *Flexible Context Processing* identifie les opérateurs de traitement du contexte les plus significatifs et les encapsule par la suite dans un objet. Le deuxième patron *Encator* permet d'encapsuler la logique de l'application dans un composant afin de traiter des actions ou des données contextuelles reçues. La figure 2.11 montre tous les patrons de conception proposés et leurs relations.

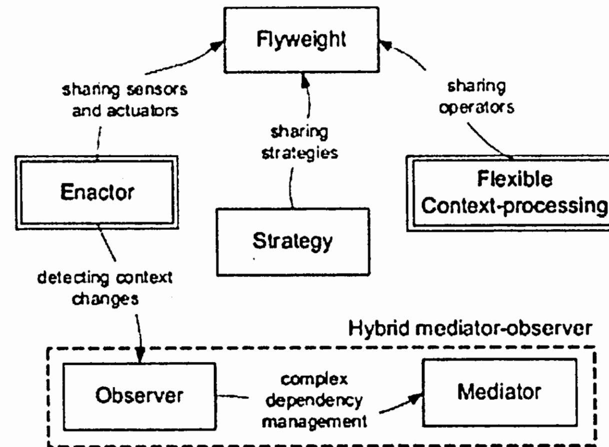

Figure 2.11 Les patrons proposés et leurs relations [25].

2.7 L'ingénierie des lignes de produits logiciels appliquée aux applications dépendantes du contexte

L'*Ingénierie des lignes de produits logiciels* permet de concevoir des logiciels réutilisables dans des environnements de production de masse [24]. Elle offre aujourd'hui de nouvelles perspectives dans le domaine des systèmes sensibles au contexte. Dans cette section, nous présentons la notion *Ingénierie des lignes de produits* et ses caractéristiques ainsi que les travaux récents qui ont associé cette notion avec les systèmes sensibles au contexte.

2.7.1 Les lignes de produits logiciels

Les *Lignes de produits logiciels* [1] (LDP) sont une famille de produits conçus pour partager un ensemble de propriétés communes et satisfaire des besoins spé-

1. En anglais : Software Product Line (SPL)

cifiques avec un gain considérable en termes de cout, temps et qualité [8] [24].

Dans la littérature, on distingue deux définitions des LDP. La première définition proposée dans [8] est :

" Un ensemble de systèmes logiciels qui partagent et gèrent un ensemble de caractéristiques communes satisfaisant les besoins spécifiques d'un segment de marché particulier ou une mission, et qui sont développées à partir d'un ensemble commun d'atouts essentiels décrit d'une manière pertinente".

Comme deuxième définition nous citons celle apparue dans [24] qui décrivent les LDP comme étant :

"Un paradigme pour développer des applications (des systèmes logiciels intensifs et des produits logiciels) en utilisant des plate-formes et la production à grande échelle de produits adaptés aux besoins de chaque client ".

2.7.2 Framework pour LDP

L'*Ingénierie des lignes de produits logiciels* comporte deux processus [24] : l'*Ingénierie de domaine* et l'*Ingénierie d'application.* L'*Ingénierie de domaine* est un processus qui établit une plate-forme réutilisable et par conséquent définit les points communs et la variabilité du produit. Tandis que l'*Ingénierie d'application* est un processus qui dérive des applications de la ligne de produits à partir de la plate-forme créée dans l'ingénierie de domaine. La figure 2.12 illustre ces deux processus.

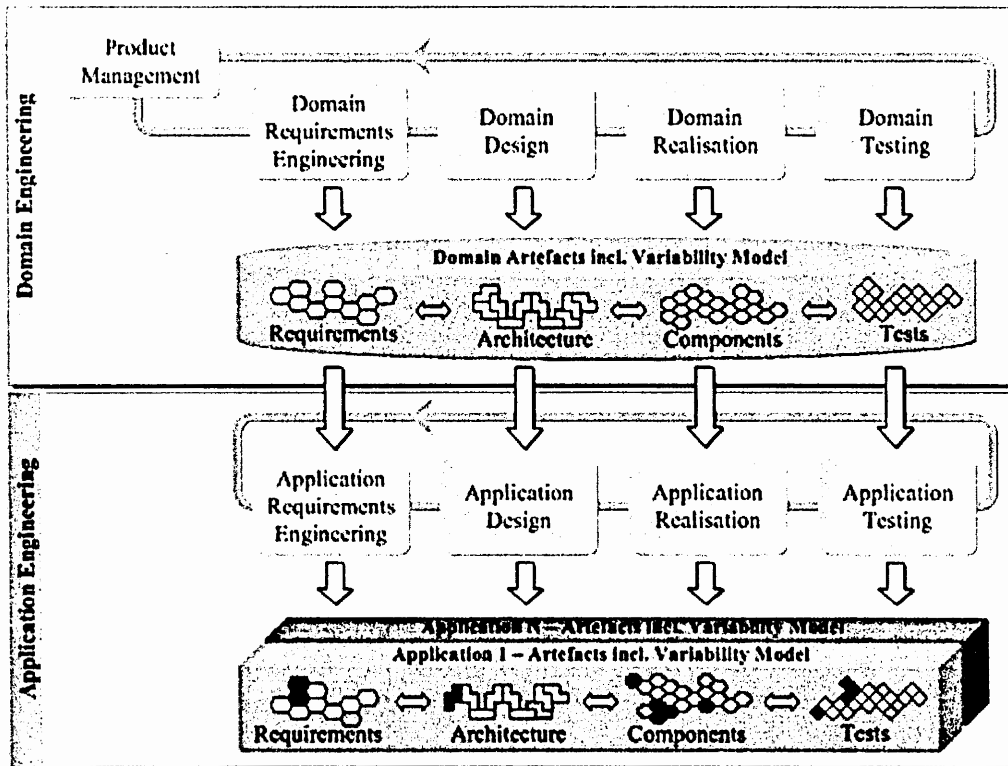

Figure 2.12 Ingénierie de domaine et Ingénierie d'application [24].

2.7.2.1 L'ingénierie de domaine

Les principaux objectifs du processus d'ingénierie de domaine sont [24] :

- Définir les points communs et la variabilité de notre ligne de produits logiciels.
- Définir l'ensemble des applications qui sont prévues par la ligne de produits.
- Définir et construire des artefacts réutilisables qui permettent de réaliser la variabilité souhaitée.

Pour accomplir ces objectifs, le processus d'ingénierie de domaine est composé de quatre activités [24] qui sont : l'ingénierie des exigences de domaine, la modélisation de domaine, la réalisation de domaine et le test de domaine. Ce processus est composé à son tour de quatre sous-activités que sont les spécifications et exigences du produit, la conception, la réalisation et les tests (Voir figure 2.12).

Cette étape nécessite l'intervention des experts du domaine pour identifier les caractéristiques communes et variables pour une famille de produits et se traduit par la mise en place d'un diagramme de caractéristiques de cette famille. Pour mieux comprendre cette étape prenons l'exemple de l'approche de [23]. La figure 2.13 illustre le digramme de caractéristiques associé à cette approche. Il s'agit d'un catalogue pour une application e-commerce.

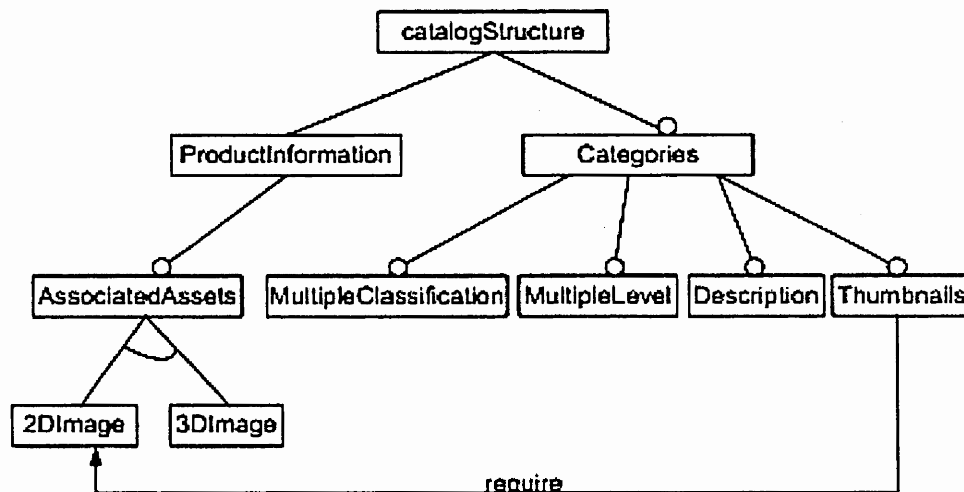

Figure 2.13 Diagramme de caractéristiques associé à l'ingénierie de domaine [23].

Une fois les points communs et la variabilité de la ligne de produits logiciels définis, on procède à la mise en place des artefacts logiciels nécessaires au

développement de produits. Pour cela, il faut associer à chaque caractéristique les artefacts nécessaires pour l'implémenter. La figure 2.14 donne un aperçu de la construction des artefacts de produits à partir du diagramme de caractéristiques précédent.

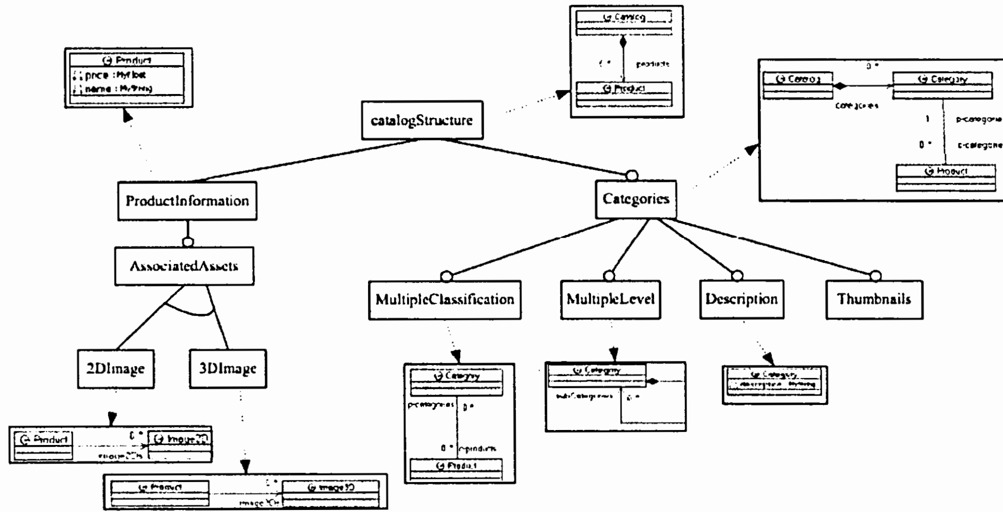

Figure 2.14 Exemple d'artefacts construit à partir du diagramme de caractéristiques [23].

2.7.2.2 L'ingénierie d'application

Le processus de l'ingénierie d'application permet de dériver les produits finaux à partir des artefacts identifiés dans la phase précédente. Les principaux objectifs de cette phase sont :

- Maximiser le taux de réutilisation des artefacts définis lors de la définition et l'élaboration d'une application de la ligne de produits.
- Exploiter les points communs et la variabilité de la ligne de produits logiciel au cours de l'élaboration d'une application.

- Documenter les artefacts de l'application, à savoir les exigences, l'architecture, les composants et les tests. Par la suite, relier chaque activité aux artefacts du domaine.
- Estimer l'impact des différences entre l'application et les exigences du domaine sur l'architecture, les composants et les tests.

Comme l'ingénierie de domaine, l'ingénierie d'application adopte la même structure. Elle est composée de quatre activités et qui sont l'ingénierie des exigences de l'application, la conception de l'application, la réalisation de l'application et le test de l'application [24].

2.7.2.3 La variabilité

La variabilité est un concept clé dans les lignes de produits logiciels. Elle définit un ensemble d'hypothèses montrant comment des produits provenant de la même ligne de produits diffèrent. Cette variabilité est définie en premier lieu durant la phase de l'ingénierie de domaine et exploitée par la suite durant la phase de l'ingénierie de l'application moyennant la construction des variantes appropriées [24]. .

Ces variantes déterminent les caractéristiques de variabilité d'un point de variation dans la ligne de produits logiciel [24]. Par exemple, le moteur d'une voiture peut tourner à l'essence, au gazole ou bien à l'électricité. Dans ce cas le type du moteur est un point de variation et (essence, gazole, électricité) sont les variantes.

2.7.3 Utilisation des LDP dans les applications sensibles au contexte

De plus en plus de chercheurs dans des travaux récents s'intéressent à la notion des lignes de produits logiciels pour résoudre des problèmes liés aux systèmes sensibles au contexte [17] [10] [18], notamment le problème de la variabilité due aux changements continus des données contextuelles collectées et la façon d'adapter ces changements au cours de l'exécution d'une application.

Dans la littérature, on distingue deux approches intéressantes. La première approche [21], propose d'utiliser le processus de LDP pour construire des applications orientées services et adapter leurs exécutions au cours du temps en fonction d'un contexte donné. Le scénario proposé est une application qui fournit des informations sur des films en prenant le problème de connexion avec la base de données comme contexte. L'approche est composée de deux phases : une phase initiale et une phase itérative.

La phase initiale consiste en premier à identifier les caractéristiques de la ligne de produits logiciels et procéder par la suite à une composition des artefacts logiciels nécessaires au développement du produit à partir des caractéristiques sélectionnées dans le but de dériver la première version du produit final. La composition de ces artefacts est assistée par un méta-modèle décrivant la structure et le comportement général de l'application et de ses composants. Pour chaque caractéristique sélectionnée, il existe un modèle partiel (dérivé du méta-modèle) qui correspond au produit lui-même. Une fois les artefacts définis, ces derniers seront transformés en composants moyennant l'architecture SCA [2] [3] (Service Component Architecture) et par la suite en code source.

---

2. http://en.wikipedia.org/wiki/Service_Component_Architecture

3. http://www.davidchappell.com/articles/introducing_sca.pdf

La phase itérative quant à elle gère la variabilité du produit et adapte le comportement de l'application à tous changements contextuels d'une façon dynamique. Dans cette phase, deux activités sont nécessaires. D'abord, introduire une architecture pour la gestion et l'acquisition du contexte. Cette architecture doit être reliée aux artéfacts et aux différentes caractéristiques proposées dans le modèle de caractéristiques de la ligne de produits logiciels définis lors la phase initiale. Par la suite, ils proposent une plate-forme d'exécution pour adapter dynamiquement l'application en fonction de toute modification dans le contexte. La plate-forme utilisée dans cette approche est la plate-forme *FraSCati*[4] . Cette dernière est une implémentation open-source du consortium OW2 destinée aux spécifications SCA. La figure 2.15 représente l'architecture proposée dans la première approche.

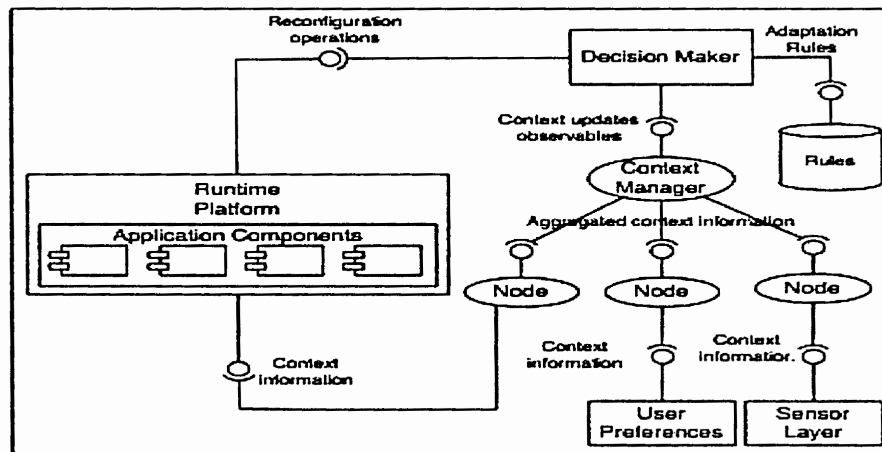

Figure 2.15 Architecture Proposée [21].

La deuxième approche proposée dans [2] adapte aussi le principe de la LDP pour concevoir et implémenter des systèmes sensibles au contexte. Elle se focalise

4. http://wiki.ow2.org/frascati/Wiki.jsp?page=FraSCAti

sur la façon de fournir des services web autonomes et de les adapter dynamiquement au cours de l'exécution dans une application mobile. Le cas d'étude utilisé dans cette approche consiste en une application mobile pour touristes qui liste les différentes attractions touristiques et le plan de visite de ces attractions en fonction de la météo et la position géographique courante de l'utilisateur. La figure 2.16 illustre les différentes caractéristiques de cette application.

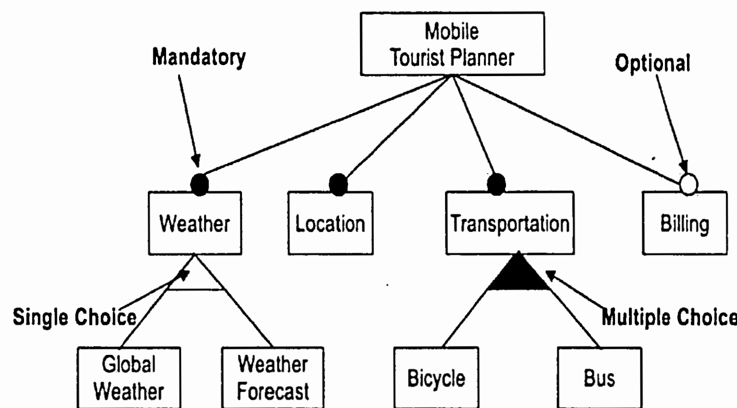

Figure 2.16 Diagramme de caractéristiques associé à la deuxième approche [2].

Pour concevoir ce type d'application, deux phases ont été proposées, une phase initiale et une phase de configuration lors de l'exécution. La phase initiale consiste à promouvoir la réutilisation des services web et de leurs artefacts dans la ligne de produits logiciels. Ces services web sont développés en utilisant le principe de l'informatique autonome ; tandis que la phase de configuration consiste à faciliter la recomposition des services lorsqu'un changement contextuel est détecté. Pour cela, des modèles d'exécution ont été utilisés pour déterminer automatiquement comment la composition de services devrait évoluer.

Pour réaliser la phase de configuration à l'exécution, trois outils sont utilisés :

1. *SALMon* : C'est un outil qui permet de contrôler et détecter des violations dans le contrat de service (*Service level agreement* ou [5] SLA). Il combine des techniques de contrôles et de tests pour obtenir au moment de l'exécution des informations sur la qualité de service dont les attributs sont de nature dynamique (les attributs dynamiques sont des attributs dont la valeur peut changer en cours d'exécution, tels que le temps de réponse et la disponibilité). Sur la base de ces informations, l'outil *SALMon* peut détecter toute violation dans les clauses de SLA [3].

2. *MoRE-WS* : C'est un moteur de reconfiguration à base de modèles pour les services Web. Il utilise les modèles de variabilité fournis lors de la phase initiale et détecte toute transformation dans ces modèles (activation et désactivation des services web représentés sous forme de caractéristiques) lors de l'exécution pour déterminer comment une composition de services devrait être reconfigurée suite à un changement contextuel. Le rôle de *MoRE-WS* est d'interroger régulièrement les nouveaux flux de données pour détecter de nouvelles informations contextuelles. Lorsqu'un nouvel événement contextuel est détecté, *MoRE-WS* évalue le type de changement détecté et procède à la création d'un plan de reconfiguration. Ce plan contient un ensemble d'actions de reconfiguration pour modifier la composition des services.

3. *Swordfish* : Les actions de reconfiguration définis par *MoRE-WS* sont exécutées via le framework *Swordfish*. *Swordfish* est un projet de la fondation Eclipse dans l'objectif est de fournir une plate-forme d'exécution SOA.

---

5. `http://fr.wikipedia.org/wiki/Service_level_agreement`

2.8 Conclusion

Nous avons débuté ce chapitre par exposer la notion de contexte en étudiant ses aspects les plus importants, qui sont les différentes définitions proposées, l'acquisition du contexte et les modèles de représentation existants. Nous avons ensuite présenté des approches qui portent sur l'aspect de l'ingénierie pour concevoir et mettre en œuvre des applications dépendantes du contexte.

À travers les approches proposées, nous avons constaté que la plupart de ces travaux restent limitée et ne traitent que des cas bien spécifiques. Par exemple, les approches qui portent sur l'application des patrons de conception proposent des modèles conceptuels qui simplifient l'évolution et la réutilisation de ces modèles mais sans se préoccuper de l'aspect adaptabilité des applications sensibles au contexte durant leurs exécutions. Ceci a donné naissance à une nouvelle pratique qui utilise les lignes de produits logiciels pour définir la variabilité induite par les applications sensibles au contexte et gérer l'adaptation dynamique de ces applications au contexte. Nous avons constaté aussi que l'évolution des types de données caractérisant le contexte n'est pas prévue dans ces approches.

Dans le chapitre suivant, nous présentons notre approche pour concevoir et mettre en œuvre des applications mobiles dépendantes du contexte mieux structurées en adaptant les services offerts par ces applications au contexte d'une façon dynamique.

CHAPITRE III

APPROCHE CONCEPTUELLE PROPOSÉE

3.1 Introduction

Dans le chapitre I, nous avons présenté les objectifs de notre travail pour concevoir des applications mobiles dépendantes du contexte. Les objectifs définis sont :

1. la mise en œuvre d'un modèle conceptuel pour traiter et disséminer les données du contexte.

2. la conception d'une architecture pour l'adaptation des services offerts par l'application aux données captées.

3. l'utilisation d'une approche plus structurée au niveau de la conception de l'application.

Pour atteindre ces objectifs, nous avons utilisé des pratiques de conception qui supportent l'extension et des pratiques de développement de *lignes de produits* qui supportent la variabilité dans les applications. Dans ce chapitre, nous proposons une vue générale de l'architecture proposée, une étude de cas pour illustrer notre approche et une description détaillée de cette architecture en décrivant les différentes pratiques utilisées.

3.2 Vue générale de l'architecture proposée

L'architecture générale de notre approche est illustrée dans la figure 3.1. Cette architecture est répartie en deux parties, une application cliente pour mobile et un serveur d'application.

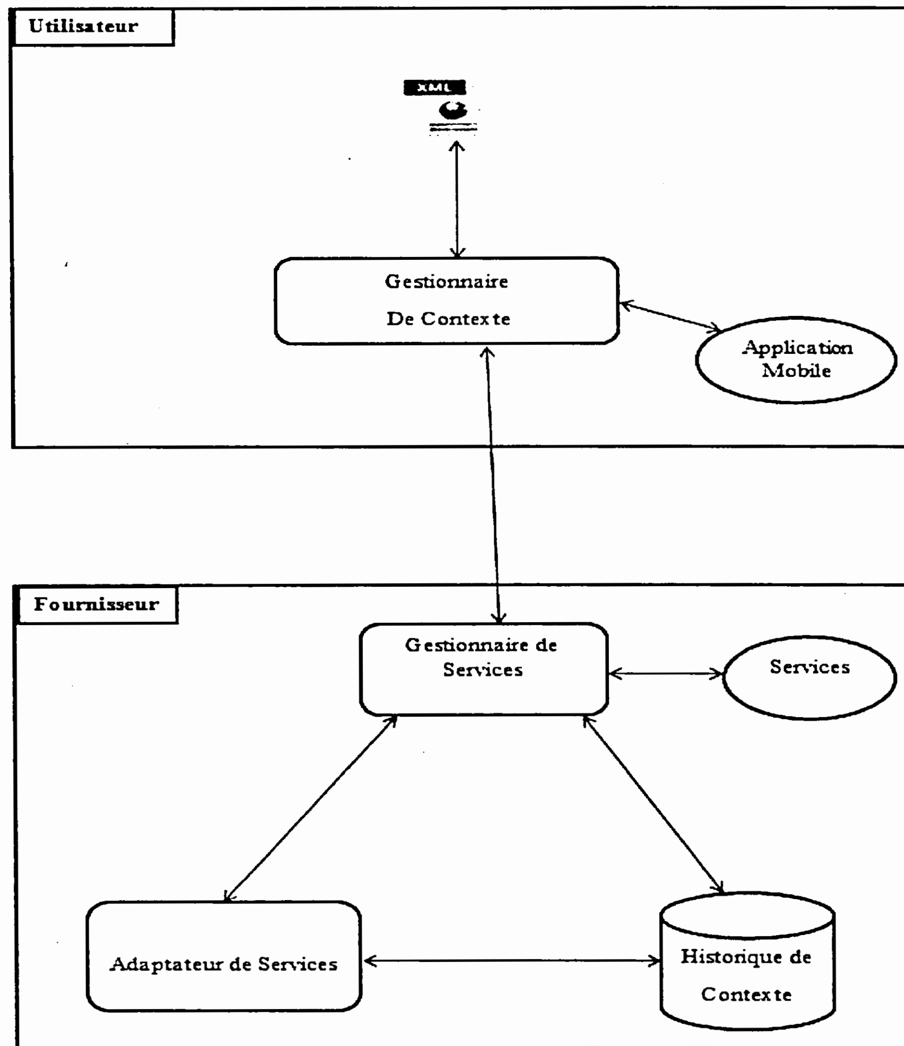

Figure 3.1 Architecture d'une application mobile dépendante du contexte.

Les services proposés dans le serveur d'application forment une combinaison de données et d'informations mises à la disposition des utilisateurs distants.

Dans un modèle classique client-serveur, l'interaction entre un consommateur de service et un fournisseur se fait moyennant des requêtes simples. Toutefois, dans le cas des applications dépendantes du contexte, il est de plus en plus difficile de se limiter à ce type de modèle pour interagir avec le serveur vu la variabilité continue du contexte dans lequel l'utilisateur se trouve.

Pour gérer cette variabilité, nous proposons de répartir les tâches entre les deux parties de notre architecture : la partie *Client* gère le contexte de l'utilisateur avant d'invoquer un service et transmettre une requête au serveur, et la partie *Fournisseur de Service* qui gère les requêtes reçues pour les adapter au contexte de l'utilisateur.

Dans la partie client, la première étape consiste à observer à partir du contenu d'un fichier XML tout changement dans les valeurs contextuelles captées, traiter les nouvelles valeurs observées et de transmettre ces valeurs au fournisseur de service. Ainsi, lorsqu'un utilisateur désire invoquer un service dépendant du contexte, le gestionnaire de contexte notifie tout changement dans les valeurs contextuelles captées au fournisseur de service et assure la transmission des nouveaux paramètres contextuels pour chaque service.

Par conséquent, pour mettre en œuvre notre gestionnaire de contexte et pour avoir une conception flexible qui nous permet de gérer les variations du contexte et l'ajout éventuel d'autres types de données (c-à-d. de nouveaux capteurs) nous avons utilisé les patrons de conception.

La figure 3.2 illustre les tâches du client.

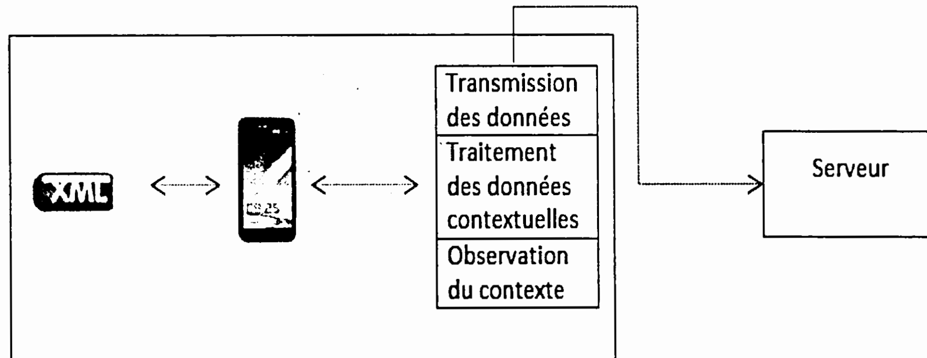

Figure 3.2 Les tâches effectuées du côté client.

Toutefois, l'interaction entre l'utilisateur et les services offerts n'est pas directe. Elle est orchestrée par un gestionnaire de service qui assure toute interaction entre un demandeur de service et son fournisseur.

Lors de la réception d'une requête provenant du gestionnaire de contexte, le gestionnaire de service assure la correspondance entre les nouveaux paramètres envoyés (correspondance entre le type de services demandés et les nouvelles valeurs contextuelles qui leurs sont associées) et les services existants, ainsi que la sauvegarde de ces paramètres dans une base de données pour garder un historique. Dans le cas d'un changement contextuel qui nécessite un ou plusieurs services non-fonctionnels, le gestionnaire de service fait appel à l'adaptateur de service afin de déclencher un processus d'adaptation pour invoquer ces services.

Une fois le processus d'adaptation terminé, le gestionnaire de service retourne à l'utilisateur l'ensemble de services nécessaires comme réponse aux changements contextuels survenus.

Du côté du serveur (fournisseur de service), nous avons appliqué les pratiques dans les lignes de produits logiciels pour caractériser notre architecture par un ensemble de composants/services qui s'activent ou désactivent selon les données du contexte transmises par le client dans ses requêtes vers le serveur.

En particulier, nous avons modélisé l'architecture de l'application serveur en utilisant un diagramme de caractéristiques dans lequel chaque caractéristique est un service. Ce modèle-là spécifie l'ensemble des configurations possibles de notre application. Donc à la réception d'une requête du client, les paramètres de la requête sont analysés par le gestionnaire de service et sont utilisés avec le diagramme de caractéristiques pour décider quelles reconfigurations sont nécessaires (e.x. activer un service).

Pour la mise en œuvre d'une telle architecture nous avons utilisé une architecture qui facilite la création, la composition et la reconfiguration des services. Spécifiquement nous avons utilisé l'architecture SCA (Service Component Architecture).

La figure 3.3 illustre les tâches du serveur.

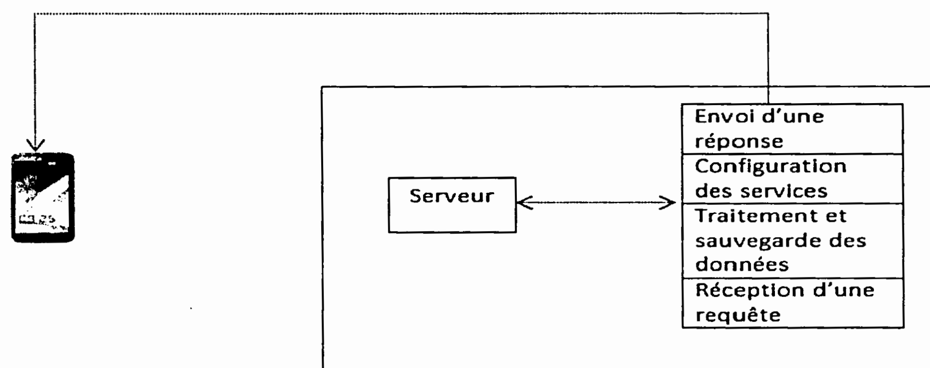

Figure 3.3 Les tâches effectuées du côté serveur.

Dans ce qui suit, nous présentons avec plus de détails cette architecture en décrivant la phase de gestion et de dissémination des valeurs contextuelles captées ainsi que la phase de gestion et de l'adaptation dynamique des services existants comme réponse à ces valeurs.

3.3 Étude de cas

Pour illustrer notre approche, nous optons pour l'utilisation d'un exemple de scénario dans notre démarche de modélisation et de conception.

Nous proposons de concevoir une application mobile qui propose une multitude de services à un patient. Ces services dépendent d'un ensemble de données contextuelles captées telles que la température corporelle, la pression artérielle et le rythme cardiaque de l'utilisateur (voir figure 3.4). En fonction de ces données, l'application fournit à l'utilisateur un ou plusieurs services portant sur chaque contexte.

Comme services, nous proposons d'offrir au patient des informations pertinentes sur son état de santé. Ainsi, l'utilisateur sera en mesure de récupérer différentes informations sur un contexte donné. Par exemple quand l'utilisateur désire mesurer sa pression artérielle, il aura accès à tous les services qui sont disponibles dans le contexte de la pression artérielle. Dans le cas d'une variation inhabituelle, une information complète est envoyée au médecin ou bien à l'infirmière, et ceci en fonction de l'anomalie détectée.

Nous proposons aussi, un service de localisation qui offre à l'utilisateur l'adresse des services médicaux (hôpitaux, cabinet de médecin etc.) les plus proches de sa positon courante.

Nous rappelons que dans le cadre de ce projet, nous supposons que la capture du contexte et l'interprétation des données sont déjà fournies. On utilise un fichier XML qui simule les données du contexte. La figure 3.4 présente un exemple de contexte relié à notre exemple d'application sur l'état d'un patient.

```xml
<?xml version="1.0"?>
<sci_reply version="1.0">
  <send_message>
    <device id="00000000-00000000-00409DFF-FF4395ED">
      <rci_reply version="1.1">
        <do_command target="idigi_dia">
          <channel_dump>
            <device name="sensor0">
              <channel name="Blood_pressure" value="70" units="cmHg" timestamp="Thu Jan  1 12:42:08 2013" type="float"/>
              <channel name="Heart_rate" value="60" units="min" timestamp="Thu Jan  1 01:31:05 2013" type="float"/>
              <channel name="Human_body_temperature" value="37" units="C" timestamp="Thu Jan  1 12:42:08 2013" type="float"/>
            </device>
          </channel_dump>
        </do_command>
      </rci_reply>
    </device>
  </send_message>
</sci_reply>
```

Figure 3.4 Modélisation des valeurs contextuelles associées à notre approche.

3.4 Modèle conceptuel de l'application cliente

Nous nous sommes intéressés à présent au rôle que peut jouer l'application cliente dans la gestion des informations contextuelles captées et le lien qu'elle peut avoir avec l'architecture d'adaptation à concevoir. De plus, avec l'avancement technologique que connaissent les appareils mobiles en termes de mémoire, temps de calcul et capacité de stockage, ces derniers se présentent comme étant une alternative intéressante pour permettre le partage des tâches entre la partie client et partie serveur d'une application.

Par conséquent, le modèle conceptuel proposé aura pour objectif de faciliter le traitement des informations contextuelles captées et la transmission de ces informations vers le fournisseur de service.

Notre approche consiste à enrichir les classes de l'application cliente pour faciliter la correspondance entre les requêtes émises par le client et les services existants du côté du serveur. En effet, il est important de faire la correspondance entre un contexte donné et le service correspondant.

Pour cela, nous proposons d'utiliser deux patrons de conception [11] pour enrichir notre modèle conceptuel.

En premier lieu, nous utilisons le patron *Observateur* pour détecter tout changement dans notre contexte et mettre à jour les différentes valeurs contextuelles associées aux services à invoquer.

Grace à la notion observateur/observable illustrée dans la figure 3.5, nous définissons la classe *Contexte* comme observable et la classe *Observateur_ Contexte* comme observateur. La classe *Contexte* prend comme données observables notre contexte et notre fichier XML contenant les informations contextuelles captées. Si on se réfère à notre scénario, le sujet à observer est le contexte de l'utilisateur incluant sa température corporelle, sa pression artérielle, son rythme cardiaque et sa localisation.

Pour détecter un changement dans le contexte de l'utilisateur, l'idée consiste à analyser tous les changements dans les valeurs contextuelles captées en comparant les anciennes valeurs mémorisées à celles existantes dans le fichier XML puis envoyer une notification à l'observateur l'informant du nouvel état du contexte en cas de changement.

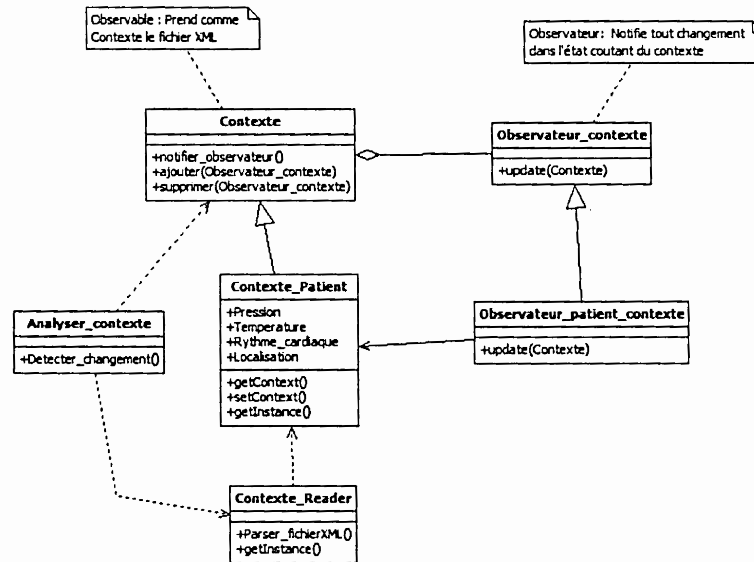

Figure 3.5 Structure du patron observateur associé à notre approche.

Cependant, pour éviter le problème des changements continus dans les valeurs contextuelles captées et par conséquent la transmission successive de ces valeurs sous forme de requêtes au serveur, nous proposons de filtrer toutes les valeurs contextuelles captées pour garder uniquement les valeurs les plus pertinentes.

Comme solution, nous proposons d'utiliser le patron de conception *Commande*. Grâce au patron *Commande* nous définissons un ensemble de services dans lesquels chaque service exécute un ensemble d'actions pour traiter les paramètres contextuels qui lui sont associés avant d'envoyer une requête au serveur. Pour cela, chaque objet d'un service communique une ou plusieurs actions à effectuer, ainsi que les paramètres requis pour le faire.

Le but de cette démarche est d'assurer la correspondance entre les informations contextuelles captées et l'objet spécifique de chaque service du côté client.

La figure 3.6 illustre la relation entre le patron observateur et le patron commande.

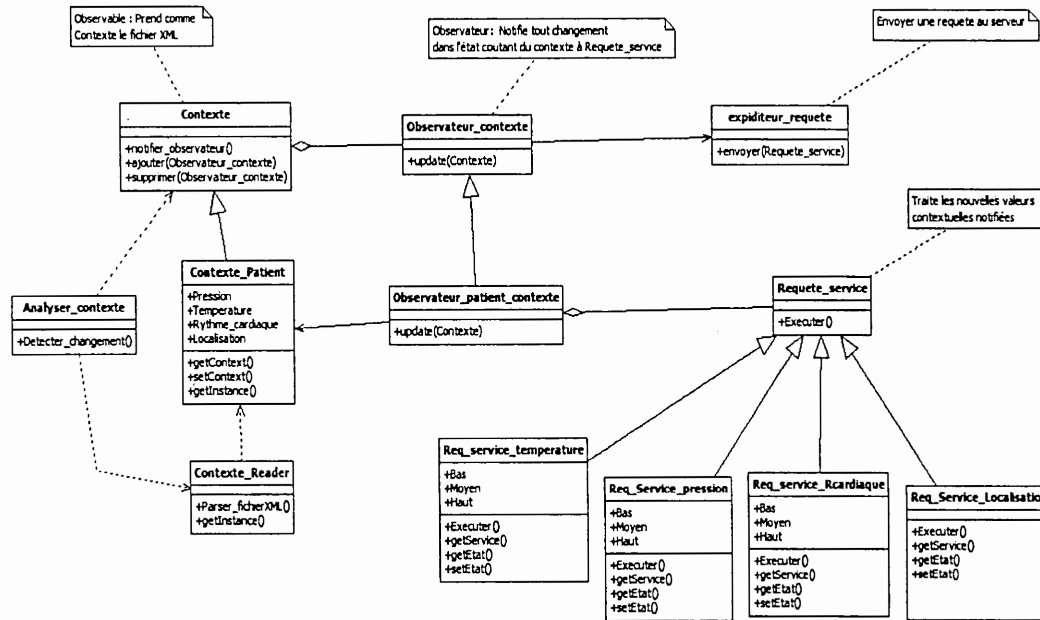

Figure 3.6 La relation entre le patron Observateur et Commande.

Suite à une modification dans les valeurs contextuelles captées, l'observateur (invocateur de la commande) invoque la ou les *Requete_Service* appropriées pour effectuer les opérations de traitements associées à ces nouvelles valeurs contextuelles notifiées avant d'envoyer une requête au serveur.

Les opérations de traitement consistent à détecter les services associés à ces nouvelles valeurs notifiées et leur degré de pertinence. Par conséquent, chaque sous-requête de *Requete_Service* sera en charge d'exécuter les actions adéquates pour filtrer les valeurs qui lui sont associées. Dans notre scénario, nous proposons de procéder à un filtrage basé sur des seuils pour classifier les valeurs contextuelles associées à la température corporelle, pression artérielle et rythme cardiaque en

trois catégories (Bas, Moyen et Haut). Le but de cette classification est de déterminer la pertinence des nouvelles valeurs notifiées et d'identifier le ou les services à invoquer pour faire face à ce changement contextuel.

Par exemple, pour une pression artérielle moyenne l'application cliente n'a pas besoin d'envoyer de requêtes au serveur pour traiter l'information. C'est l'application elle-même qui se chargera de retourner l'information à l'utilisateur. Par contre, dans le cas d'un passage de moyen à haut dans la pression artérielle de l'utilisateur l'application cliente se doit de faire appel au fournisseur de service pour retourner le ou les services les plus pertinents comme réponse à ce changement. Comme par exemple, retourner les services médicaux les plus proches.

D'une façon plus générale, nous définissions notre modèle conceptuel pour l'application cliente comme l'indique la figure 3.7.

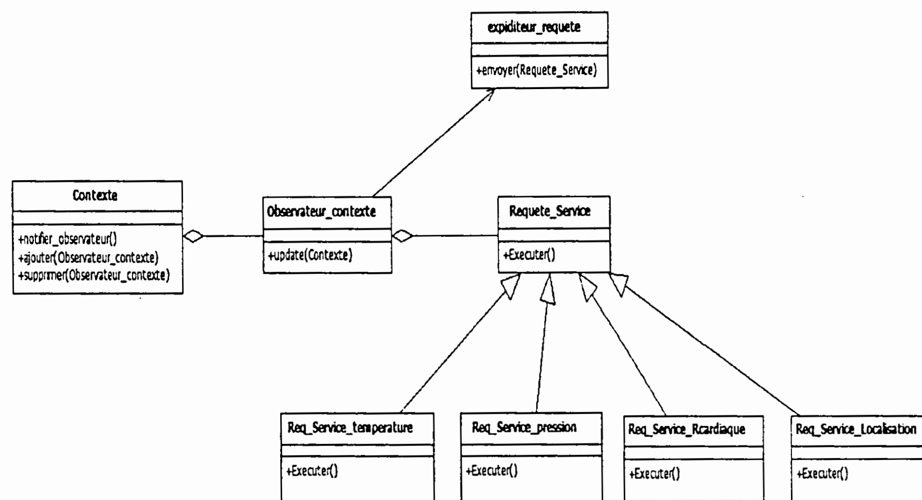

Figure 3.7 Modèle conceptuel de l'application cliente.

3.5 Adaptation de l'application serveur au contexte

Pour adapter l'application serveur au contexte, nous proposons de mettre en place cette nouvelle pratique de *la ligne de produit logiciel LDP* associée aux systèmes sensibles au contexte et qui consiste à diviser la construction d'un produit logiciel en deux phases, la phase initiale et la phase de reconfiguration.

La phase initiale commence par la sélection des caractéristiques prises en charge par l'application et de son déploiement. La phase de reconfiguration commence dès le déploiement de l'application et propose un processus d'adaptation qui gère toute variabilité induite par les changements contextuels survenus lors de l'exécution.

En LDP, les caractéristiques d'un logiciel sont représentées sous forme d'une arborescence appelée diagramme de caractéristiques. Ce diagramme est utilisé pour classer toutes les exigences qui peuvent être satisfaites par l'application. Dans notre cas, nous utilisons le diagramme de caractéristiques pour modéliser les services possibles de notre application. La figure 3.8 illustre le diagramme de caractéristiques correspondant à notre scénario.

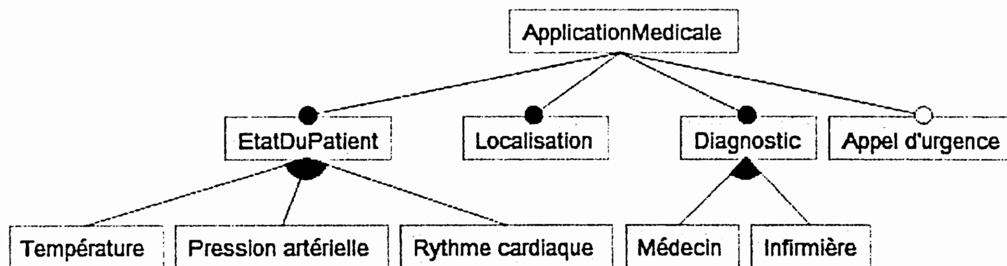

Figure 3.8 Diagramme de caractéristiques.

Le premier niveau de connexion représenté par les caractéristiques (*État du patient*, *Localisation*, *Diagnostic*) et muni d'un cercle noir représentent les caractéristiques obligatoires de l'application tandis que ceux en blanc représentent des caractéristiques optionnelles (Appel d'urgence dans notre cas).

Chaque niveau de connexion peut à son tour définir d'autres caractéristiques alternatives. Ainsi, les caractéristiques tels que *État du patient* et *Diagnostic* sont des points de variations et *température*, *pression*, *rythme cardiaque*, *infirmière* et *médecin* sont leurs variantes. Les arcs noirs inversés définissent ces variantes comme étant des choix multiples pour chaque point de variation.

Ainsi, l'état du patient offre comme alternatives des informations sur la température, la pression et le rythme cardiaque de l'utilisateur, alors que diagnostic relie toutes informations relatives au patient à son médecin ou son infirmière.

La localisation fournit à l'utilisateur les adresses des services médicaux les plus proches ou bien l'adresse de son médecin ou infirmière en se basant sur sa position géographique.

Pour mieux comprendre le comportement de notre scénario, nous proposons de construire un diagramme d'activités UML pour représenter les différentes liaisons possibles entre nos caractéristiques (les liaisons établies entre services à différents moments et selon l'état du contexte). Ce diagramme d'activités est illustré dans la figure 3.9.

Le service *Application médicale* est un service composé qui reçoit des requêtes à partir d'un dispositif mobile et retourne des informations pertinentes sur l'état de santé de l'utilisateur et permet aussi au médecin ou à l'infirmière de faire le suivi de leur patient à distance.

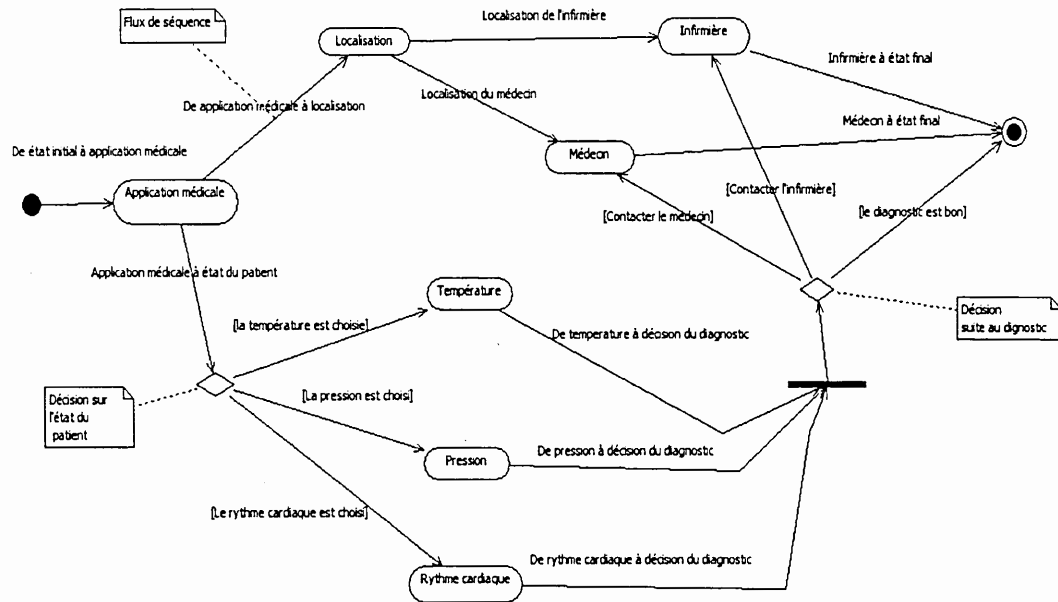

Figure 3.9 Diagramme d'activités correspondant à notre scénario.

Au niveau de la prise de décision sur l'état du patient et en fonction des changements contextuels survenus, l'application peut choisir entre trois services (*température corporelle*, *pression artérielle*, *rythme cardiaque*) pour obtenir des informations relatives à ces nouvelles valeurs contextuelles. Suite à cette première prise de décision, une deuxième décision doit être prise sur le nouvel état du patient. En fonction de cette décision, une information complète sera envoyée au médecin ou à l'infirmière selon la gravité de l'état du patient. Dans le cas d'un diagnostic avec des résultats stables l'application se contente de retourner à l'utilisateur une fiche complète sur son état. Le service Localisation quant à lui récupère l'emplacement actuel de l'utilisateur et retourne les adresses des hôpitaux, médecins et infirmières les plus proches.

Dans ce qui suit, nous allons présenter les deux phases de construction et d'adaptation de notre application serveur au contexte.

3.5.1 Phase initiale

La première phase commence par la sélection, à partir du diagramme de caractéristiques (figure 3.8), des fonctions qui seront prises en charge lors du démarrage de l'application. Cette démarche se fait d'une façon manuelle et consiste à créer et générer les composants représentant la configuration initiale de l'application.

Pour notre scénario, nous proposons de sélectionner les caractéristiques suivantes : *État du patient* (température, pression artérielle, rythme cardiaque), *Localisation* et *Diagnostic* (alerte d'une infirmière). Par la suite, pour générer les composants du produit final nous proposons d'utiliser l'architecture SCA (*Service Component Architecture*) comme outil pour la composition des caractéristiques sélectionnées sous forme de service. La figure 3.10 illustre l'assemblage des composants qui forment l'application serveur à déployer.

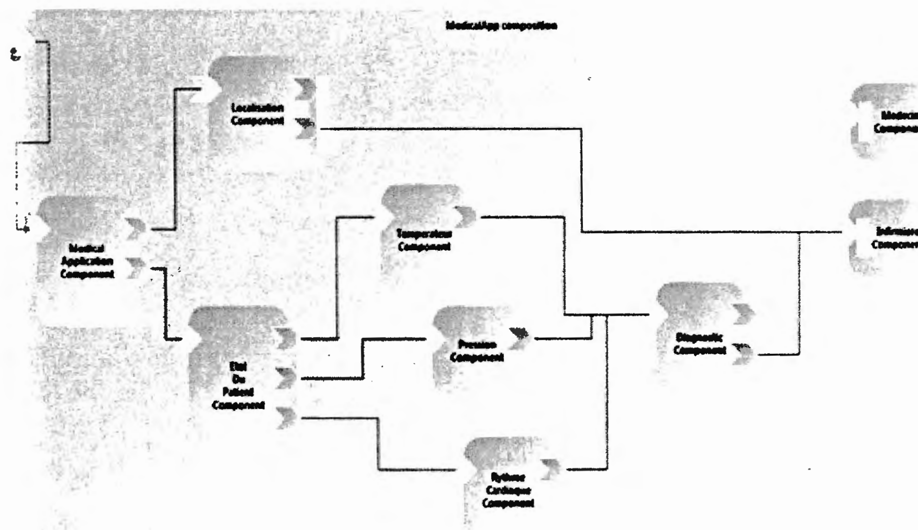

Figure 3.10 Assemblage des composants avec l'architecture SCA.

Une fois les composants créés, ces derniers forment l'architecture de notre application serveur en termes de structure et comportement. La structure est représentée par les services et les références qui constituent l'application. Tandis que le comportement est représenté par l'interaction entre les services et les références des composants.

Les flèches qui se trouvent à gauche de chaque composant représentent le service fourni, alors que les flèches de droite représentent les références requises pour chaque service afin de faire le lien avec d'autres composants.

La figure 3.11 donne un aperçu général de quelques spécifications nécessaires pour l'assemblage de composants dans une architecture SCA.

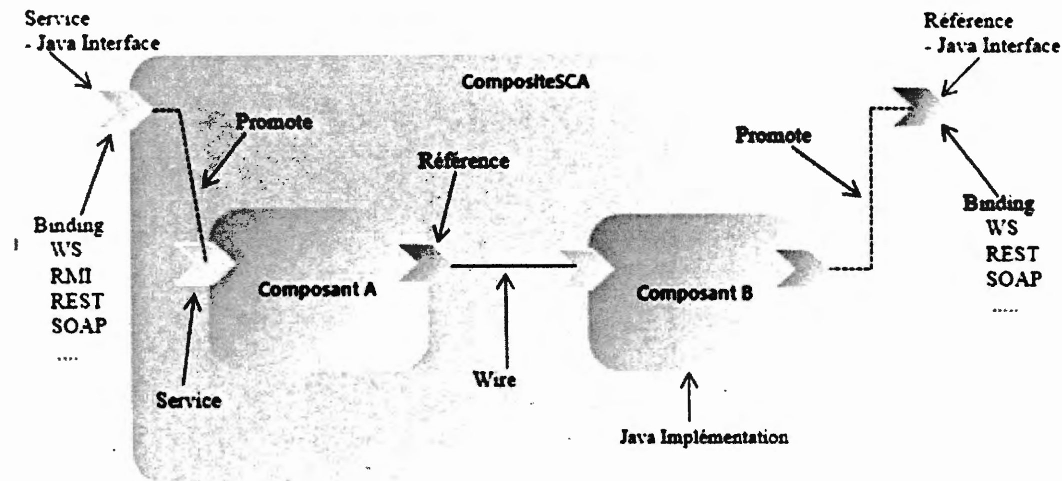

Figure 3.11 Un assemblage de composants dans une architecture SCA.

Chaque composant est une implémentation java qui définit une interface java comme entrée pour les services et comme sortie pour les références. Ces interface sont reliés entre eux par un lien appelé *Wire.* Ils sont exposés aussi à d'autres interfaces externes au travers de (RMI, REST, SOAP, etc.).

Suite à la création des composants, la dernière étape consiste à relier ceux qui sont sélectionnés, et générer par la suite le code source de l'application.

La figure 3.12 illustre le méta-modèle de l'architecture associée à l'application serveur après la phase initiale.

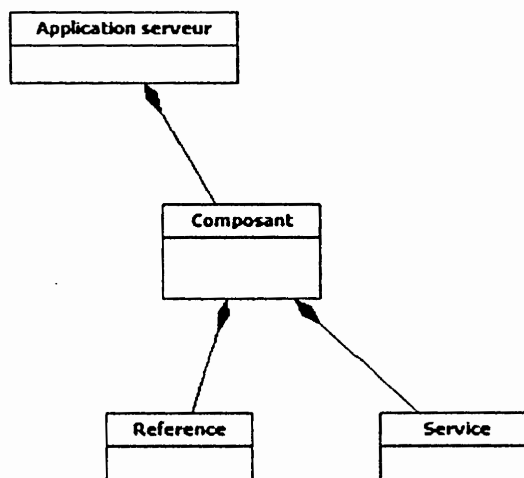

Figure 3.12 Architecture de l'application serveur à la phase initiale.

3.5.2 Phase de reconfiguration

Une fois l'application serveur déployée, la phase de reconfiguration joue le rôle d'un gestionnaire de service entre l'utilisateur et l'application mobile en adaptant dynamiquement les services disponibles en fonction des changements contextuels reçus. Toutefois, lors de l'adaptation, une question essentielle se pose :

- Comment gérer l'adaptation dynamique de l'application au cours de l'exécution ?

Avant d'entamer un processus d'adaptation suite à un changement contextuel, il faut définir certaines règles pour faire le lien entre le type de requête en-

voyé et les services existants pour pouvoir déterminer le ou les services appropriés comme réponse à ce changement.

Dans [21], un méta-modèle qui répond à cette question a été proposé. Leur proposition consiste à composer des artefacts sensibles au contexte en plusieurs clauses. Chaque clause est composée à son tour d'une partie Test qui, moyennant des conditions prédéfinies, observe tout changement dans l'environnement de l'application, et d'une partie Corps qui suite à une notification s'occupe de l'adaptation dynamique des services existants en détectant l'emplacement et le type de modification à faire.

Dans notre approche, nous avons exploité le méta-modèle de [21] et nous avons essayé de l'adapter à notre cas. Ainsi, notre processus d'adaptation consiste à définir certaines règles pour déterminer quel service est demandé, est-il fonctionnel ou non et quel type d'adaptation à effectuer.

Toutefois, l'adaptation dynamique d'un service au cours de l'exécution réside dans la capacité d'une application à reconfigurer son comportement pour faire face à tout changement. Par conséquent, l'application doit être capable de suspendre l'exécution de son système et de modifier sa structure en procédant à des opérations comme : *ajouter*, *supprimer*, *lier* et *délier* les différents composants existants.

Pour réussir l'adaptation dynamique, deux étapes sont nécessaires. La première consiste à identifier l'emplacement du changement, tandis que la deuxième consiste à adapter le changement détecté sur le composant sélectionné.

La figure 3.13 présente le méta-modèle d'adaptation de nos services aux changements contextuels survenus en cours d'exécution.

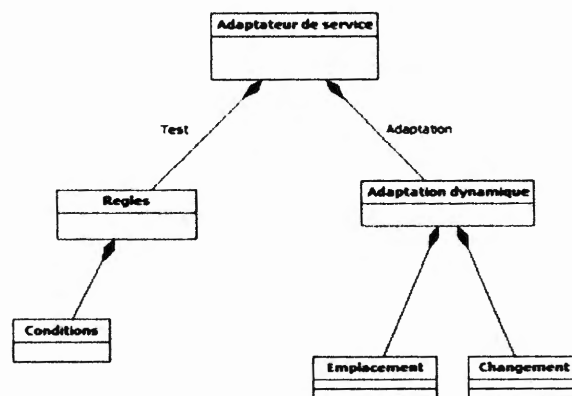

Figure 3.13 Méta-modèle d'adaptation.

Pour mieux comprendre le processus d'adaptation prenons le scénario suivant. Supposons que la configuration initiale de notre application est la suivante :

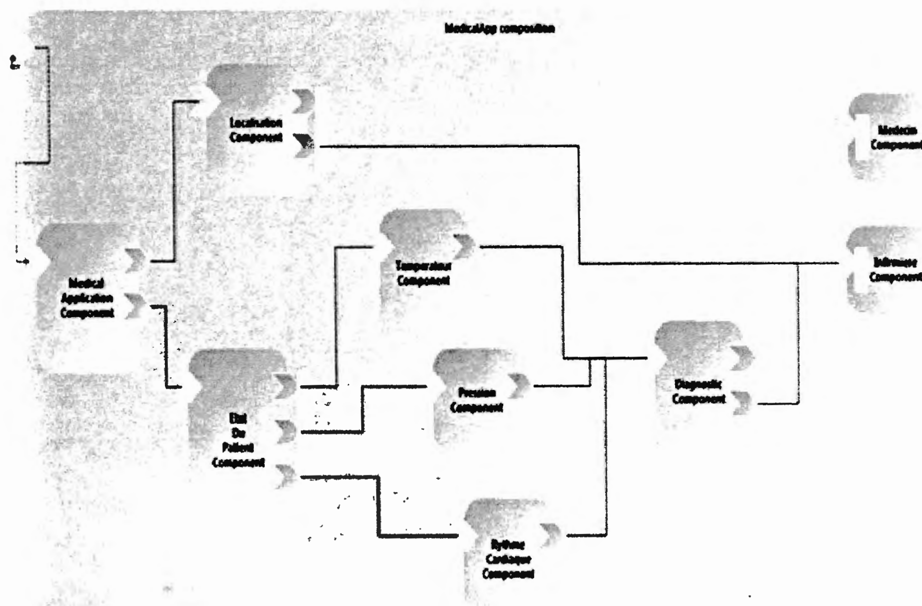

Figure 3.14 Assemblage des composants correspondant à la phase initiale.

Comme scénario initial, nous supposons que l'état du patient est stable est ne nécessite pas l'intervention de son médecin, seule l'infirmière est en mesure de faire le suivi. Suite à un changement dans les données contextuelles, l'état d'un patient nécessite l'intervention de son médecin pour l'informer du changement survenu. Dans ce cas, pour que l'application adapte son comportement à ce nouveau changement, elle doit faire appel au composant *Médecin* déjà existant dans l'architecture proposée par SCA.

Pour réaliser ce type de reconfiguration, nous avons utilisé la plate-forme *Frascati*. Cette plate-forme fait partie de plusieurs implémentations des spécifications SCA tels que *Apache Tuscany* et *IBM WebSphere* mais contrairement à *Frascati* ces derniers n'offrent pas de capacités pour reconfigurer à l'exécution les applications SCA.

Cependant, la plate-forme Frascati propose trois moyens de reconfiguration ; soit par *Frascati explorer* un outil graphique de visualisation, soit par une API java spécifique, soit par *Frascati Fscript*[1] un langage dédié particulièrement adapté aux interrogations d'architecture.

Frascati Fscript utilise deux notations complémentaires *Fscript* et *Fpath* ;

Fpath facilite la navigation à l'intérieur d'une architecture SCA en utilisant des requêtes simples et lisibles permettant ainsi de détecter l'emplacement d'un changement à l'intérieur de l'architecture SCA. Alors que *Fscript*, qui fait usage de *Fpath*, permet de modifier cette architecture pour l'adapter dynamiquement au cours de l'exécution.

Ainsi, pour reconfigurer une architecture SCA, il faut commencer par identifier les chemins des composants à modifier puis définir les opérations à effectuer

1. `http://fractal.ow2.org/fscript/`

pour réaliser cette reconfiguration.

Pour cela, *Frascati Fscript* fournit des variables (telles que la variable *$domain* qui fait référence au domaine SCA) et des actions prédéfinies (telles que *start()*, *stop()*, *scabinding()*, sc*awire()* etc.). Il offre aussi la possibilité de définir des variables et des procédures.

Si on reprend notre scénario de reconfiguration. La première étape consiste à identifier l'emplacement de notre changement. Ainsi, les composants à identifier sont *Médecin*, *Localisation* et *Diagnostic.* Une fois les composants identifiés, il faut identifier les références et les services qui serviront à faire le lien entre ces composants.

Une fois l'emplacement identifié, on procède aux changements souhaités dans notre architecture. Pour cela on fait appel à l'action *add-scawire()* pour lier les composants entre eux.

La figure 3.15 illustre le script qui correspond à notre scénario.

```
---- Emplacement
med=$domain/scadescendant::Medecin
loca=$domain/scadescendant::Localisation
diag=$domain/scadescendant::Diagnostic
local-ref=$loca/scareference::medecin
diag-ref=$diag/scareference::medecin
med-serv=$med/scaservice::medecin

----Changement
add-scawire($local-ref, $med-serv)
add-scawire($diag-ref, $med-serv)
```

Figure 3.15 Fpath et Fscript associés à notre scénario.

Il faut noter que, pour faire une action de type *add-scawire()*, on n'a pas besoin d'arrêter le composant possédant la référence à lier. Tandis que pour suppri-

mer des liaisons entre des composants, il faut procéder à l'arrêt de ces composants en utilisant l'action set-state($Path-composant, "STOPPED") et les redémarrer en utilisant l'action set-state($Path-composant, "STARTED") une fois l'adaptation terminée.

L'adaptation souhaitée terminée, l'architecture ressemblerait à la configuration de la figure 3.16.

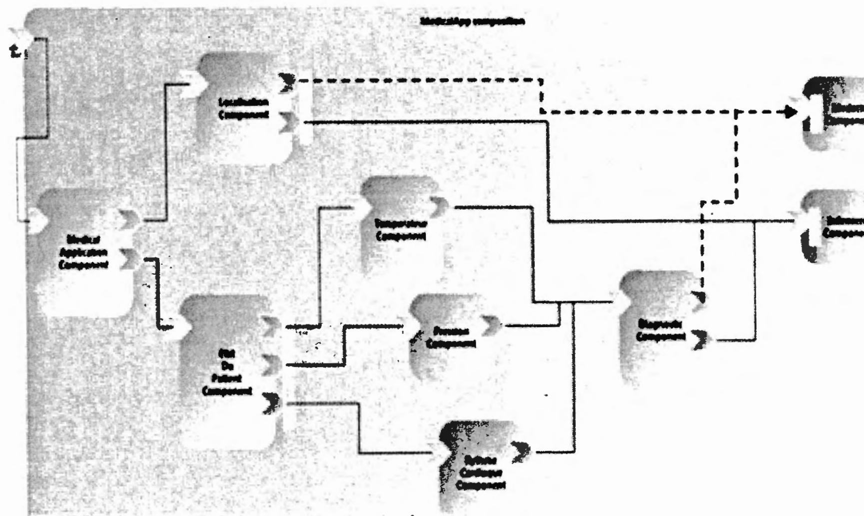

Figure 3.16 L'architecture SCA suite à l'adaptation.

Ainsi, la plate-forme *Frascati* offre un support de composants SCA réflexifs permettant la reconfiguration à chaud des assemblages pendant leur exécution en modifiant le comportement et la structure de l'application.

Une fois les deux phases de construction et d'adaptation de l'application serveur au contexte présentés, nous proposons le méta-modèle suivant comme étant une présentation générale de l'architecture associée à l'application serveur.

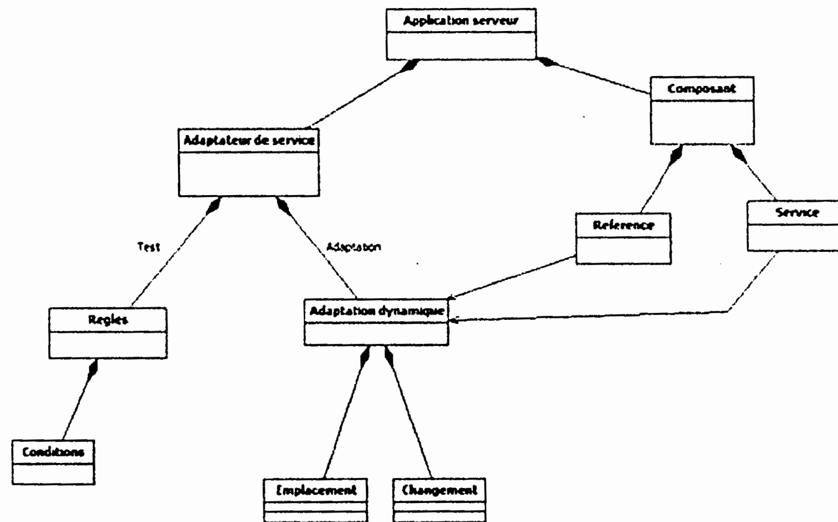

Figure 3.17 Méta-modèle de l'application serveur.

La phase de construction de l'application serveur consiste à définir les différents composants de l'application qui se composent de service et de référence. Tandis que la phase adaptateur de service comporte une partie test qui définit les différentes règles et conditions qui s'appliquent pour chaque changement et une partie adaptation dynamique qui gère la reconfiguration de l'application en détectant en premier lieu l'emplacement des changements à faire et en procédant par la suite à l'activation et la désactivation des liens (les services et les références) qui relient les différents composants.

3.6 Conclusion

Dans ce chapitre, nous avons présenté notre approche pour concevoir des applications mobiles dépendantes du contexte. À travers cette approche, nous avons présenté l'architecture générale de l'application et nous en avons détaillé chaque partie. Dans la partie client nous avons exploité des pratiques bien établies

de conception pour gérer les changements et l'évolution des différents types de données composants le contexte. Pour cela, nous avons mis en place un modèle conceptuel extensible et réutilisable afin de gérer les variations du contexte et faciliter le traitement et la transmission des données contextuelles au serveur. Dans la partie serveur nous avons utilisé les lignes de produits logiciels pour caractériser la variabilité de nos services et construire une architecture qui facilite la création et la réutilisation de ces services. L'architecture fournie, assure l'adaptabilité de l'application serveur en reconfigurant ses services selon les données du contexte transmises par le client dans ses requêtes.

CHAPITRE IV

PLAN DE VALIDATION ET OUTILS

4.1 Introduction

Après avoir exposé notre approche conceptuelle, nous mettons l'accent dans ce chapitre sur la mise en oeuvre de cette approche et sur les problématiques rencontrées.

4.2 Outils

Nous proposons le plan et les outils suivants pour la validation de notre approche :

- La mise en place d'un serveur d'application dédié qui repose sur *Frascati* : Ce serveur fournit les différents services qui seront consommés par l'application cliente. Le développement des services à exposer sur le serveur sont bâtis par l'architecture SCA, une architecture orientée service à base de composants.

  L'avantage de SCA est qu'il fournit un cadre architectural pour les applications orientées services où il suffit d'écrire le code métier et spécifier dans un fichier XML les services web à exposer.

Quant à la plate-forme *Frascati*[1], elle est téléchargeable sous forme d'un fichier zip. Pour installer la plate-forme, il suffit d'extraire les fichiers existants.

- La mise en place de l'application cliente, une application mobile (Android par exemple) : cette application fournit les interfaces nécessaires pour assurer la communication et la consommation des services exposés sur le serveur d'application dédié par *Frascati.*

4.3 Problématiques rencontrées

L'une des problématiques que nous avons rencontrée lors de notre travail, c'est le manque de support et de documentation sur la plate-forme *Frascati.*

En fait, la plate-forme *Frascati* est un projet de recherche qui a été élaboré par l'INRIA [2] (un organisme public de recherche dédié aux sciences et technologies du numérique) et dont le développement est toujours en cours.

Actuellement, la plate-forme destinée aux spécifications SCA la plus utilisée par la communauté est Apache Tuscany où plusieurs supports et documentations sont fournis. L'avantage avec Tuscany est qu'elle offre une large gamme de protocoles de communication et divers types de langages SCA (tels que Java, C++, BPEL, etc.), contrairement à la plate-forme Frascati qui ne supporte pas tous les protocoles de communication et se focalise principalement sur les technologies Java. Par contre, *Frascati* offre des capacités pour reconfigurer des applications SCA à l'exécution, contrairement à *Tuscany.*

---

1. `http://forge.ow2.org/project/showfiles.php?group_id=329`

2. `http://www.inria.fr/`

L'avantage avec la plate-forme Frascati est qu'elle propose plusieurs outils pour reconfigurer et interagir avec des applications SCA en cours d'exécution.

Dans ce qui suit, nous proposons d'introduire la mise en place du serveur d'application dédié par *Frascati.*

4.4 La mise en place du serveur d'application

Nous présentons les démarches à suivre pour mettre en place le serveur d'application et comment utiliser la plate-forme *Frascati* pour interagir et effectuer des opérations de reconfiguration à l'exécution à partir du serveur.

La première étape consiste à définir la structure de notre application serveur. Ainsi, il faut définir, une partie ressources qui va contenir le descripteur de l'architecture SCA (un fichier spécifiant les services à exposer) et une deuxième partie qui va contenir les classes métier de nos services (c-à-d les interfaces et leur implémentation). En fait, le compilateur de *Frascati* s'attend à une certaine structure lors de la compilation d'une application SCA.

La figure 4.1 donne un aperçu de cette structure.

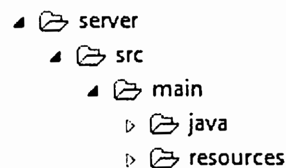

Figure 4.1 Un exemple de structure associée à une application SCA.

La deuxième étape consiste à générer le descripteur de l'architecture SCA (un fichier **.composite*) et construire notre application SCA. Pour construire une application par assemblage de composants, SCA propose un langage de programmation décrit en XML. Chaque composant représente une fonctionnalité implémentée par une classe.

Il existe deux façons pour générer le descripteur d'architecture :

- Soit en utilisant un éditeur STP/SCA d'Eclipse. Cet éditeur offre un outil graphique qui permet d'assembler les composants de l'application et génère automatiquement le descripteur d'architecture.
- Soit en générant manuellement notre fichier.

Pour modéliser le descripteur d'architecture associé à notre scénario, nous devons construire nos neufs composants, définir les relations existantes entre chaque composant, implémenter les classes qui définissent les fonctionnalités de chaque composant et définir le protocole de communication grâce auquel le serveur d'application sera exposé. Comme protocole nous proposons d'utiliser dans cet exemple le style architectural REST et le protocole http. Ainsi, notre serveur fournit des services sous la forme de méthodes via une URI. Pour accéder à ces méthodes nous avons travaillé essentiellement avec les verbes GET et POST de http pour manipuler nos ressources.

La figure 4.2 nous donne un aperçu sur la structure des composants *MedicalServiceApp*, *EtatDuPatient* et *Localisation*.

```
<component name="MedicalServiceApp">
    <implementation.java class="org.ow2.frascati.memoire.medical.lib.MedicalServiceImpl" />
    <service name="MedicalService">
        <frascati:binding.rest uri="http://localhost:8080/MedicalService"/>
    </service>
    <reference name="EtatPatient" target="EtatDuPatient/EtatPatient"/>
    <reference name="localisation" target="Localisation/localisation"/>
</component>

<component name="EtatDuPatient">
    <implementation.java class="org.ow2.frascati.memoire.medical.lib.EtatDuPatientImpl" />
    <service name="EtatPatient"/>
    <reference name="temperature" target="Temperature/temperature"/>
    <reference name="pression" target="Pression/pression"/>
    <reference name="rythmecardiaque" target="RythmeCardiaque/rythmecardiaque"/>
</component>

<component name="Localisation">
    <implementation.java class="org.ow2.frascati.memoire.medical.lib.LocalisationImpl" />
    <service name="localisation"/>
    <reference name="medecin"/>
    <reference name="infirmiere" target="Infirmiere/infirmiere"/>
</component>
```

Figure 4.2 la structure des composants MedicalServiceApp, EtatDuPatient et Localisation.

Une fois la création et l'assemblage des composants terminés, il nous reste plus qu'à compiler et exécuter notre application SCA.

Lors de la compilation, nous appelons Frascati avec la commande *compile* pour créer le fichier jar de l'application avec :

- *<frascati compile src medic-server>*

  Ensuite, l'exécution est réalisée grâce à la commande *run.*

- *<frascati run MedicApp-server -libpath medic-server.jar>*

Suite à l'exécution, le message illustré dans la figure 4.3 s'affiche au niveau du terminal. Ainsi, notre serveur d'application expose un service RESTful avec *Frascati.*

```
Running OW2 FraSCAti ...
OW2 FraSCAti Standalone Runtine
FraSCAti is running in a server mode...
Press Ctrl+C to quit...
```

Figure 4.3 Capture d'écran - Exécution du serveur d'application avec Frascati.

Pour interagir avec une application SCA implémentée avec *Frascati*, on utlise l'outil graphique *Frascati Explorer*. Cet outil, nous permet de modifier des propriétés, ajouter ou supprimer des composants, mettre à jour des liaisons, etc.

Pour invoquer l'outil *Frascati Explorer*, on fait appel à la commande <*frascati explorer*>, par la suite on charge notre fichier jar généré lors de la compilation. La figure 4.4 nous donne un aperçu de la visualisation et la reconfiguration en cours d'exécution d'une application SCA avec *Frascati Explorer*.

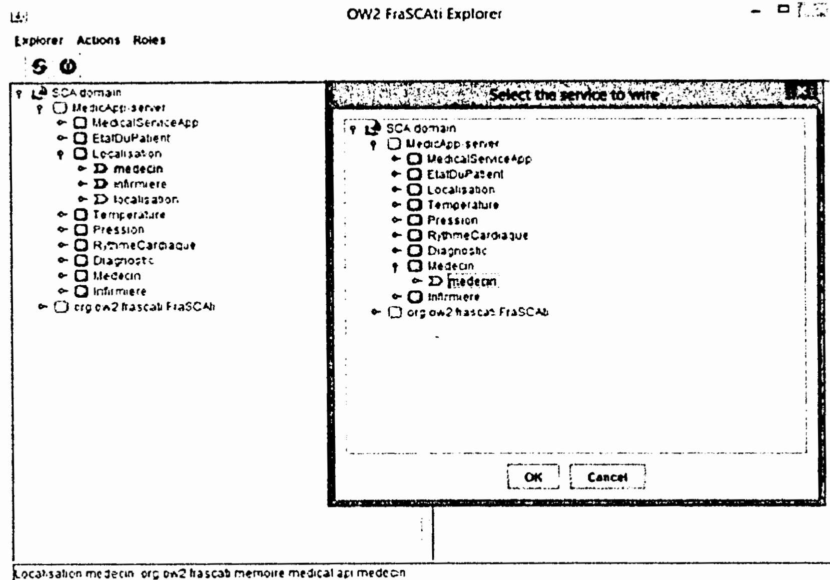

Figure 4.4 Capture d'écran - Visualisation et reconfiguration de l'application avec Frascati Explorer.

En plus, de l'outil de visualisation graphique *Frascati Explorer*, nous pouvons traduire les opérations effectuées avec l'outil graphique par le langage d'interrogation *Frascati Fscript*.

Pour cela *Frascati Explorer propose* une console pour exécuter les différents scripts associés à *Frascati Fscript*. Pour lancer cette commande, il faut ajouter l'argument -s à la commande <frascati explorer -s>. La figure 4.5 donne un aperçu sur la console de *Frascati Fscript* et les scripts associés à notre scénario :

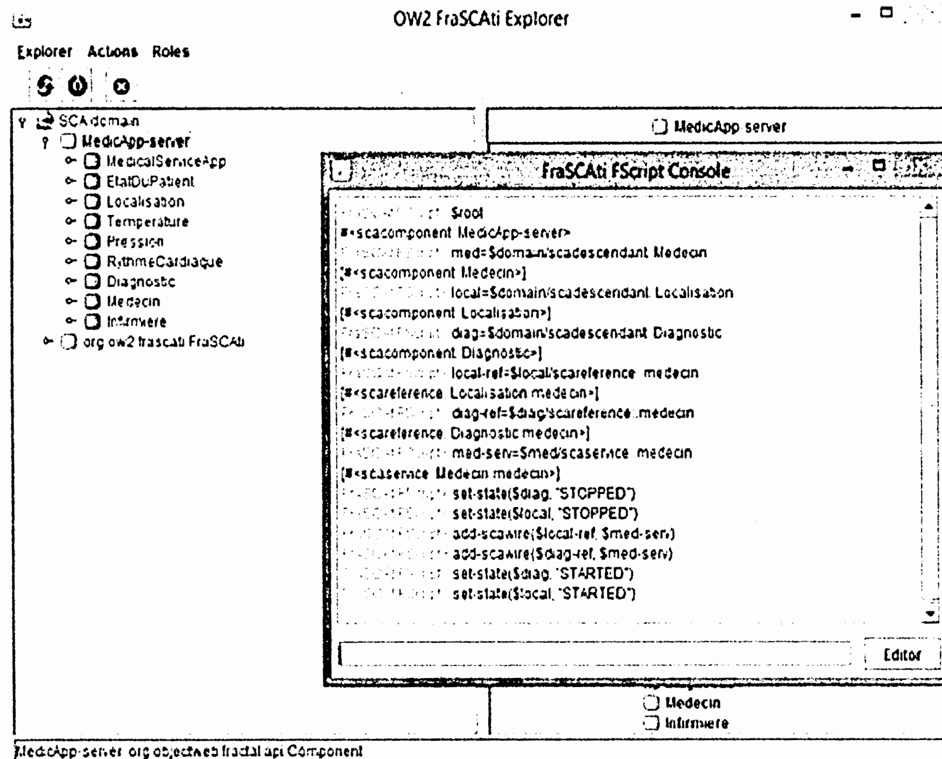

Figure 4.5 Capture d'écran- Frascati Fscript.

Pour s'assurer que la reconfiguration a eu lieu, il faut rafraichir l'affichage de *Frascati Explorer*.

Cependant, il n'est pas envisageable de procéder à ces modifications pour des applications en production. Ainsi, il est possible d'enregistrer la reconfiguration effectuée par Frascati Fscript dans une procédure et l'intégrer dans le descripteur d'assemblage. Cette procédure peut être alors réutilisée par la suite.

La procédure de *Frascati Fscript* associée à notre scénario est la suivante :

```
action adaptation_dynamqiue(){

 med=$domain/scadescendant::Medecin;
 local=$domain/scadescendant::Localisation;
 diag=$domain/scadescendant::Diagnostic;
 local-ref=$local/scareference::medecin;
 diag-ref=$diag/scareference::medecin;
 med-serv=$med/scaservice::medecin;

 set-state($diag, "STOPPED");
 set-state($local, "STOPPED");
 add-scawire($local-ref, $med-serv);
 add-scawire($diag-ref, $med-serv);
 set-state($diag, "STARTED");
 set-state($local,"STARTED");

 }
```

Figure 4.6 Procédure d'adaptation avec Frascati Fscript.

Une fois la plate-forme *Frascati* introduite, nous présentons dans ce qui suit la mise en œuvre de notre scénario. Nous avons simulé notre scénario directement sur le serveur grâce à l'outil *Frascati Explorer*.

Pour cela nous définissons une interface graphique qui simule la réception des valeurs contextuelles reçues à partir du client et affiche les informations nécessaires comme réponse aux données reçus.

Dans la figure 4.2, nous avons défini notre service MedicalService comme étant l'interface qui expose des services RESTfull aux applications distantes. Par conséquent, notre interface graphique sera instanciée à partir de l'interface MedicalService.

La figure 4.7 illustre l'interface graphique que nous avons utilisée pour réaliser notre simulation à partir de *Frascati Explorer.*

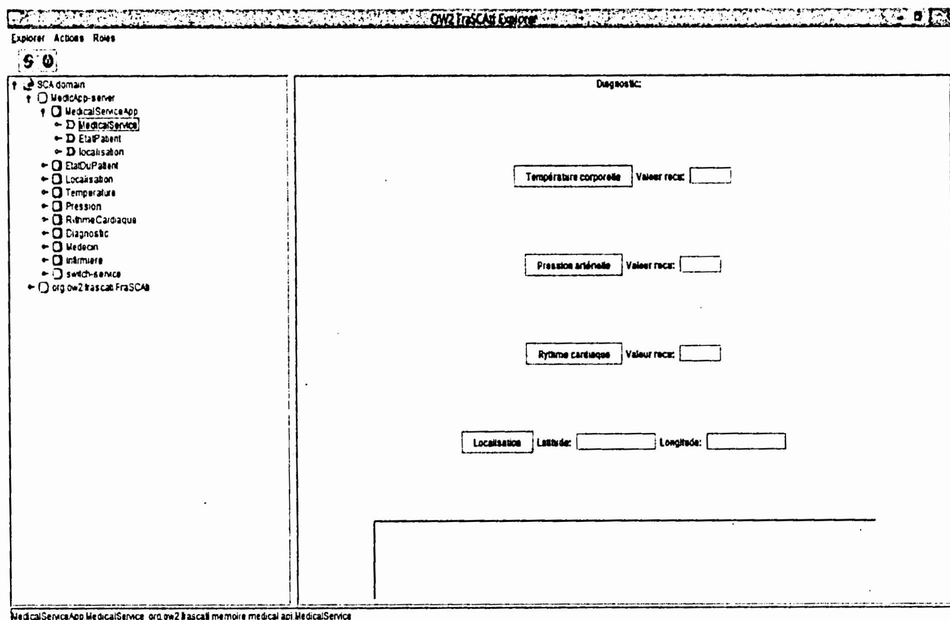

Figure 4.7 Capture d'écran- L'interface graphique de simulation.

Les valeurs récupérées sont transférées à leurs services respectifs. Ainsi, les valeurs associées à la température corporelle, la pression artérielle et le rythme cardiaque sont transférées en premier au service *EtatDuPatient* et par la suite à leurs services respectifs *Temperature*, *Pression* et *RythmeCardiaque.* Quant à la *Localisation*, nous avons travaillé avec les coordonnées GPS (latitude et longitude) pour déterminer les services médicaux les plus proches.

Ainsi, chaque service (Composant) implémente les services associés à ces valeurs contextuelles. Par exemple, le service *Localisation* contient une liste de tous les hôpitaux de Montréal (Nom, adresse et coordonnées géographique) et retourne en fonction des coordonnées reçus la liste des hôpitaux les plus proches. Par contre, le service *Diagnostic* récupère les informations traitées dans les services

Temperature, *Pression* et *RythmeCardiaque* pour mettre à jour les services associés à l'*Infirmiere* ou *Medecin* selon le diagnostic.

La figure 4.8 illustre un cas d'utilisation associé à notre scénario. En fonction des valeurs contextuelles reçues, la pression artérielle et la localisation courante, le serveur d'application traite ces valeurs et retourne le résultat du diagnostic et l'adresse des hôpitaux les plus proches par (nom, adresse et distance).

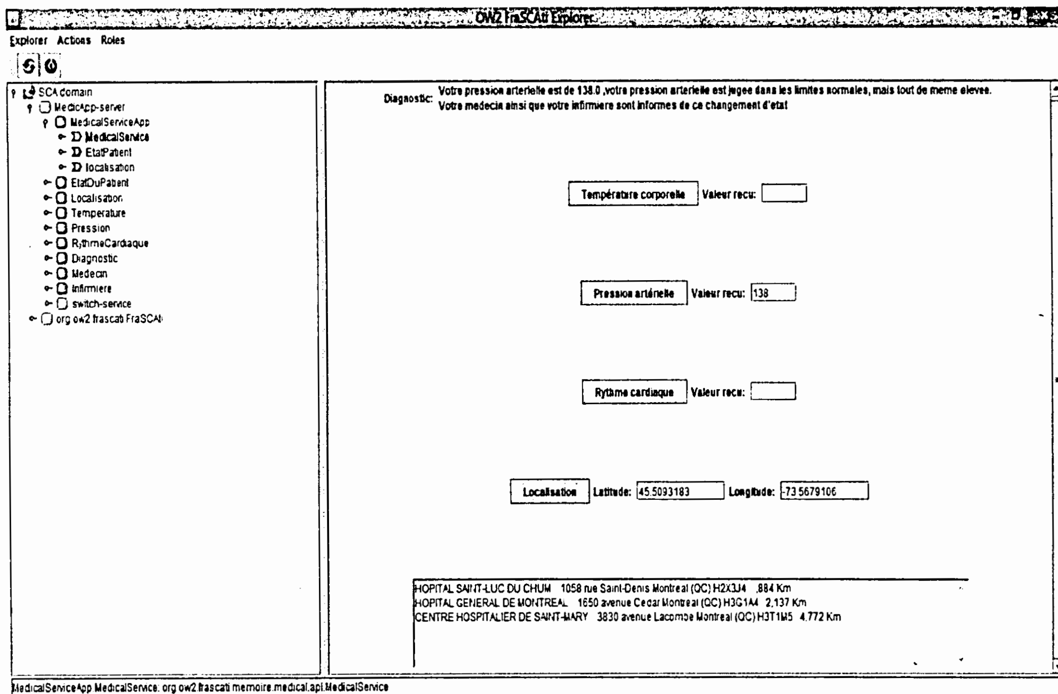

Figure 4.8 Capture d'écran- Un cas d'utilisation associé à notre scénario.

Une fois la phase initiale terminée, nous procédons à la mise en place de la phase de reconfiguration. Pour cela, nous reprenons notre scénario de reconfiguration mentionné dans le chapitre III.

La première étape consiste à définir notre partie *Test* en identifiant le ou

les services demandés et en vérifiant s'ils sont fonctionnels ou non. La deuxième étape consiste à intégrer et déclencher notre adaptation dynamique directement dans notre application. Pour cela, nous définissons un composant nommé *Switch-service* (voir figure 4.8) dans l'implémentation est un script *Frascati Fscript* en ajoutant les lignes suivantes dans notre descripteur d'assemblage :

```
<component name="switch-service">
<frascati:implementation.script script="reconfig.fscript"/>
 <service name="myReconfig">
 <interface.java interface="org.ow2.frascati.memoire.medical.api.Reconfig"/>
 </service>
</component>
```

Figure 4.9 Structure du composant Frascati Fscript.

Le fichier *reconfig.script* contient la procédure *adaptation_dynamique* décrite dans la figure 4.6 tandis que l'interface *Reconfig* reflète la procédure disponible dans le script. Il est aussi possible d'exposer notre procédure de reconfiguration à un appel distant grâce à un protocole d'accès comme RMI, REST, SOAP etc. Cependant, il faut noter que suite à la compilation le script n'est pas généré dans le jar de l'application, par conséquent il faut le rajouter.

Ainsi notre application est capable d'adapter son comportement d'une façon autonome en fonction des changements survenus lors de l'exécution.

Pour conclure notre simulation, nous avons généré une trace d'exécution reflétant le comportement de notre serveur d'application dans un fichier XML. La figure 4.10 traduit un extrait du fichier XML généré.

```
<?xml version="1.0"?>

<ServerApplication>
<Request id="1">
<Value> 138 mm Hg</Value>
<Type>Pression Arterielle</Type>
<Time>2014-03-20 19:26</Time>
<Category>Haut</Category>
<Service_Invoked>
    <Service_1 Port_Input="etatDuPatient" Port_Output="pression">EtatDuPatient Component </Service_1>
    <Service_2 Port_Input="pression" Port_Output="diagnostic">Pression Component</Service_2>
    <Service_3 Port_Input="diagnostic" Port_Output="medecin">Diagnostic Component</Service_3>
    <Service_4 Port_Input="medecin" >Medecin Component</Service_4>
    <Service_5 Port_Input="infirmiere">Infirmiere Component</Service_5>
</Service_Invoked>
<Response>votre pression arterielle est jugee dans les limites normales, mais tout de meme elevee
Votre medecin ainsi que votre infirmiere sont informes de ce changement d'etat</Response>
<Status>Success</Status>
</Request>

<Request id="2">
<Value>
    <Latitude> 45.5093183 </Latitude>
    <Longitude>-73.5679106</Longitude>
</Value>
<Type>Localisation</Type>
<Time>2014-03-20 19:27</Time>
<Service_Invoked>
    <Service_1 Port_Input="localisation">Localisation Component</Service_1>
</Service_Invoked>
<Response>HOPITAL SAINT-LUC DU CHUM  1058 rue Saint-Denis Montreal (QC) H2X3J4   0,884 Km
HOPITAL GENERAL DE MONTREAL   1650 avenue Cedar Montreal (QC) H3G1A4   2,137 Km
CENTRE HOSPITALIER DE SAINT-MARY   3830 avenue Lacombe Montreal (QC) H3T1M5   4,772 Km  </Response>
<Status>Success</Status>
</Request>
</ServerApplication>
```


Figure 4.10 Un extrait de la trace d'exécution générée.

Pour chaque requête traitée, nous avons sauvegardé les informations relatives à celle-ci dans notre fichier XML.

Les informations sauvegardées porte sur :

- Le type d'information contextuelle reçue : correspond à notre contexte (Température corporelle, Pression artérielle, Rythme cardiaque et Localisation).
- La valeur du contexte : correspond à la valeur contextuelle reçue. Ainsi, pour les valeurs contextuelles associées à l'état du patient, c'est une valeur unique. Par contre, pour la localisation, elle comporte deux sous-valeurs (Latitude et Longitude).

- La date à laquelle cette requête a été envoyée : correspond à la date système.
- La catégorie dans laquelle cette valeur a été classée : correspond à la catégorie (bas, moyen, haut) dans laquelle la valeur contextuelle a été classée lors de son traitement.
- Les services invoqués pour traiter une requête : correspond aux différents composants invoqués pour traiter une requête en identifiant les services et les références qui assurent le lien entre les différents composants(représentés par Port_input pour les services et Port_Output pour les références).
- La réponse qui sera envoyée : correspond au résultat qui sera envoyé au client une fois le traitement d'une requête terminé.

4.5 Limites et perspectives

4.5.1 Limites

Notre travail s'est limité à introduire la mise en place d'un serveur d'application dédié avec la plate-forme *Frascati.* En fait, la plate-forme *Frascati* étant relativement récente, peu de documentations et de travaux sont consacrés à cette plate-forme surtout pour les applications mobiles. Les quelques travaux existants sont des projets récents qui ont été élaborés au sein de l'équipe *Frascati* (l'équipe ADAM) soit sous forme de travaux de thèses soit des projets de recherche.

Toutefois, nous avons essayé avec les supports existants de documenter la démarche à suivre pour mettre en place un serveur d'application avec *Frascati* en prenant notre cas d'étude comme scénario.

4.5.2 Perspectives

L'approche que nous avons proposée pour concevoir une application mobile dépendante du contexte nous a présenté de nouveaux défis très intéressants au niveau de la mise en œuvre de cette approche.

Comme perspective future, il faudra exploiter au mieux la plate-forme *Frascati* pour implémenter une application complète. Il faut pour cela que l'outil soit bien documenté et qu'un support adéquat soit offert par ses développeurs.

Ceci permettra de :

- Terminer la mise en œuvre du serveur d'application *Frascati.*
- Implémenter une application mobile qui consomme des services RESTful exposé par le service d'application *Frascati.*
- Intégrer les deux applications.

CONCLUSION

L'objectif de notre travail était d'utiliser une approche plus structurée pour concevoir une application mobile dépendante du contexte. En fait, cette problématique de la conception et l'exécution d'applications mobiles sensibles au contexte est devenu un sujet de recherche très actif ces dernières années étant donné l'évolution et la complexité de plus en plus croissante de ces systèmes.

Pour aborder cette problématique, nous avons d'abord étudié plusieurs travaux relatifs à ce sujet. Notre premier objectif était de comprendre la définition même de la notion de contexte. Par la suite, nous avons considéré la notion de patron logiciel et l'approche des *lignes de produits logiciels* pour garantir une meilleure structuration de notre approche.

Nous avons également proposé une approche plus générique pour concevoir les applications mobiles dépendante du contexte dans le domaine médical.

À travers cette approche, nous avons présenté une architecture répartie en deux parties : une application cliente pour mobile et un serveur d'application dédié.

Dans l'application cliente, notre attention s'est portée sur la gestion et la dissémination des données contextuelles captées. Ceci nous a permis de fournir une structure conceptuelle qui soit à la fois réutilisable et plus générique pour gérer et disséminer les données captées en se basant sur les patrons de conception.

Au niveau du serveur d'application, notre objectif était d'adapter les services offerts par l'application à ces données captées. Pour cela, nous avons proposé de

construire notre application serveur en utilisant l'approche des *ligne de produit logiciel.*

Cette pratique consiste à définir lors d'une phase initiale les services qui seront pris en charge lors du déploiement de l'application. Et à définir lors d'une phase de reconfiguration, le processus d'adaptation qui sera mis en place pour adapter ces services en fonction des changements contextuels survenus. Dans cette phase de reconfiguration, nous avons proposé d'utiliser la plate-forme *Frascati* pour assurer cette adaptation.

Nous avons donc pu atteindre presque tous les objectifs définis dans le cadre de ce mémoire.

Cependant, un autre défi intéressant demeure celui de mettre en œuvre une application complète qui intègre l'application cliente et le serveur d'application de Frascati.

RÉFÉRENCES

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer, 1999.

[2] Germán H. Alférez and Vicente Pelechano. Context-aware autonomous web services in software product lines. In *15th International Software Product Line Conference*, pages 100–109. IEEE Computer Society, 2011.

[3] David Ameller and Xavier Franch. Service level agreement monitor (salmon). In *International Conference on Composition-Based Software Systems*, pages 224–227. IEEE Computer Society, 2008.

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

[5] Tarak Chaari, Frédérique Laforest, and André Flory. Adaptation des applications au contexte en utilisant les services web. In *Proceedings of the 2nd French-speaking Conference on Mobility and Ubiquity Computing*, pages 111–118. ACM, 2005.

[6] Elliot J. Chikofsky and James H. Cross II. Reverse engineering and design recovery : A taxonomy. *IEEE Software*, 7(1) :13–17, 1990.

[7] Eric S. Chung, Jason I. Hong, James Lin, Madhu K. Prabaker, James A. Landay, and Alan L. Liu. Development and evaluation of emerging design

patterns for ubiquitous computing. In *Proceedings of the 5th Conference on Designing Interactive Systems : Processes, Practices, Methods, and Techniques*, pages 233–242. ACM, 2004.

[8] Paul C. Clements and Linda Northrop. *Software Product Lines : Practices and Patterns*. Addison-Wesley, 2001.

[9] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1) :4–7, 2001.

[10] Andrés Fortier, Gustavo Rossi, Silvia E. Gordillo, and Cecilia Challiol. Dealing with variability in context-aware mobile software. *Journal of Systems and Software*, 83(6) :915–936, 2010.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[12] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, pages 77–86. IEEE Computer Society, 2004.

[13] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Generating context management infrastructure from high-level context models. In *4th International Conference on Mobile Data Management (MDM) - Industrial Track*, pages 1–6, 2003.

[14] Ralph E. Johnson. Frameworks = (components + patterns). *Communications of the ACM*, 40(10) :39–42, 1997.

[15] René L. Krikhaar. Reverse architecting approach for complex systems. In *International Conference on Software Maintenance*, pages 4–11. IEEE Computer Society, 1997.

[16] James A. Landay and Gaetano Borriello. Design patterns for ubiquitous computing. *IEEE Computer*, 36(8) :93–95, 2003.

[17] Jaejoon Lee and Gerald Kotonya. Combining service-orientation with product line engineering. *IEEE Software*, 27(3) :35–41, 2010.

[18] Fabiana G. Marinho, Fabrício Lima, João B. Ferreira Filho, Lincoln Rocha, Marcio E. F. Maia, Saulo B. de Aguiar, Valéria L. L. Dantas, Windson Viana, Rossana M. C. Andrade, Eldânae Teixeira, and Cláudia Werner. A software product line for the mobile and context-aware applications domain. In *Proceedings of the 14th International Conference on Software Product Lines : Going Beyond*, pages 346–360. Springer, 2010.

[19] John Mccarthy. Notes on formalizing context. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, pages 555–562. Morgan Kaumann, 1993.

[20] Ghita Kouadri Mostéfaoui, Jacques Pasquier-Rocha, and Patrick Brézillon. Context-aware computing : A guide for the pervasive computing community. In *The IEEE/ACS International Conference on Pervasive Services*, pages 39–48. IEEE Computer Society, 2004.

[21] Carlos Parra, Xavier Blanc, and Laurence Duchien. Context awareness for dynamic service-oriented product lines. In John McGregor and Dirk Muthig, editors, *13th International Software Product Line Conference*, pages 131–140, 2009.

[22] Mr. Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2Nd IEEE International Symposium on Wearable Computers*, pages 92–99. IEEE Computer Society, 1998.

[23] Gilles Perrouin, Jacques Klein, Nicolas Guelfi, and Jean-Marc Jézéquel. Reconciling automation and flexibility in product derivation. In *Proceedings of the 2008 12th International Software Product Line Conference*, pages 339–348. IEEE Computer Society, 2008.

[24] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering : Foundations, Principles, and Techniques*. Springer, 2005.

[25] Oriana Riva, Cristiano di Flora, Stefano Russo, and Kimmo E. E. Raatikainen. Unearthing design patterns to support context-awareness. In *PerCom Workshops*, pages 383–387. IEEE Computer Society, 2006.

[26] Gustavo Rossi, Silvia Gordillo, and Robert Laurini. Génération de services dépendant du contexte pour des applications mobiles. In *Proceedings of the 1st French-speaking Conference on Mobility and Ubiquity Computing*, pages 44–47. ACM, 2004.

[27] Gustavo Rossi, Silvia Gordillo, and Fernando Lyardet. Design patterns for context aware adaptation. In *The 2005 Symposium on Applications and the Internet Workshops*, pages 170–173. IEEE Computer Society, 2005.

[28] Nick Ryan. Contextml : Exchanging contextual information between a mobile client and the fieldnote server. `http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html`, 1999. Online ; accessed 20-Jan-2014.

[29] Mahadev Satyanarayanan. Pervasive computing : vision and challenges. *IEEE Personal Communications*, 8(4) :10–17, 2001.

[30] B. N. Schilit and M. M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5) :22–32, 1994.

[31] Thomas Strang and Claudia L. Popien. A context modeling survey. In *1st International Workshop on Advanced Context Modelling, Reasoning and Management, at The 6th International Conference on Ubiquitous Computing*, pages 31–41, 2004.