UNIVERSITÉ DU QUÉBEC À MONTRÉAL

UTILISATION DES PATRONS DE CONCEPTION POUR L'ADAPTATION DES INTERFACES GRAPHIQUES DES APPLICATIONS MOBILES DÉPENDANTES DU CONTEXTE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

MOHAMED CHEBBI

JUILLET 2014

REMERCIEMENTS

J'aimerais tout d'abord exprimer mes remerciements à mon directeur de recherche, Abdelatif Obaid, pour m'avoir confié ce travail de recherche, pour son appui et sa grande confiance. Ses critiques et ses conseils m'ont été précieux pour structurer le travail et l'améliorer au fil des discussions.

J'adresse aussi mes remerciements à Madame Ghizlane Elboussaidi pour son suivi et sa générosité.

Je tiens à exprimer ma gratitude à tous les professeurs de l'UQAM qui m'ont permis d'avoir une formation de qualité. Aussi, à mon directeur de programme, Monsieur Étienne Gagnon, pour son écoute, son soutien et son aide.

Merci également à tous les membres du laboratoire du recherche LATECE pour le partage de leurs connaissances, particulièrement à Madame Dufour-Landry Claudie et Madame Anne-Marie Amjaainsi qu'à Monsieur Choukri Djellali, Monsieur Alexix Laferrière etMonsieur Anis Boubaker.

J'exprime du fond de mon coeur mon affection à mes parents Hedi et Khemissa ainsi qu'à mes sœurs Hela et Manel. Sans eux, cette extraordinaire expérience professionnelle, culturelle et personnelle n'aurait jamais eu lieu.

Enfin, je remercie profondément tous ceux qui ont contribué de près ou de loin à mon succès.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Avec les progrès des réseaux sans fil, la miniaturisation des capteurs, la prolifération des technologies mobiles associées aux préférences des utilisateurs et à l'environnement exploité, l'adaptation d'interface utilisateur (IU) aux différents contextes d'utilisation estdevenueune nécessité.
Plusieurs approches ont été proposées dans le domaine d'adaptationdes interfaces. Toutefois, la plupart présente quelques limites notamment : la négligence du processus d'adaptation, la prise en considération d'une gamme limitée de contexte, un manque au niveau de la description de composant graphique, la préparation des interfaces avant l'exécution, etc. Par conséquent, il devient indispensable de proposer un nouveau cadre conceptuel.
Dans ce travail, nous combinons une approche basée sur les patrons de conception avec une architecture basée sur les règles métiers et sur une description sémantique de la UI. L'objectif est de satisfaire la génération d'une interface dépendante du contexte en cours d'exécution.
D'une part, les patrons de conception sont considérés comme une solution à un problème spécifique déjà abordé lors de la conception d'autres systèmes. Ils nous seront utiles pour l'identification des différentes variations contextuelles, la représentation de l'interface à l'utilisateur ainsi que pour la satisfaction du processus d'adaptation. D'autre part, l'architecture à base de règlesnous permettera de suggérer une interface de l'utilisateur en cours d'exécution à partir du modèle de spécification de l'interface. Cette dernière représente un service à l'utilisateur selon ses finalités.

Mots-clés : informatique mobile, interface utilisateur (UI), contexte, application dépendante du contexte, patrons de conception, règlemétier, adaptation, plasticité des interfaces.

CHAPITRE I

INTRODUCTION

De nos jours, nous constatons queles progrès des réseaux sans fil, la miniaturisation des capteurs et la prolifération des appareils portatifs, tels que les tablettes et les téléphones intelligents, tendent à révolutionner l'interaction homme-machine (Thomas, Parkinson, Moore, Goodman, Xhafa et Barolli, 2013).

Du point de vue du génie logiciel, les avancements en technologie mobile ont augmenté la capacité de déployer de nouveaux services. Par conséquent, la demande pour le développement efficace d'applications mobiles est en hausse. Cette tendance se répercute sur l'efficacité des prestations de services sur les réseaux mobiles dans une variété de domaines tels que la santé, la géo-localisation, les opérations de secours d'urgence, etc.

Ces services sont d'autant plus pertinents qu'ils peuvent s'adapter au contexte et aux environnements entourant les usagers. C'est ce qu'on appelle *la sensibilité au contexte*. Cette évolution a donné lieu à ce qu'on appelle *l'Informatique pervasive* et *l'informatique ubiquitaire*.

L'évolution courante des technologies de communication mobiles prend alors en considération la variation permanente du contexte. Ce dernier comprend plusieurs aspects (Mottiet Vanderdonckt, 2013):

- La plate-forme que l'usager utilise;
- Son environnement (bruit, lumière, température, etc.)
- Les préférences de l'usager.

Ainsi, les utilisateurs pourront bénéficier de services bien adaptés et exposés en fonction de l'état de leur environnement.

Toutes ces évolutions technologiques, matérialisées par le développement des applications mobiles, proposent des services par le biais d'interfaces ergonomiques adaptées aux utilisateurs. Ceci est d'une grande valeur ajoutée à plusieurs niveaux : diminution du temps d'apprentissage, facilité les tâches des utilisateurs, exposition de tout le besoin et l'augmentation de l'efficacité dans l'exécution des opérations par les usagers.

Partant du fait que cette ère de logiciel omniscient s'est introduite aujourd'hui dans notre vie quotidienne, cette valeur ajoutée ne peut être atteinte que grâce à des applications qui adaptent leurs contenus et leurs interfaces aux différents contextes des usagers.

1.1 Problématique

« Le comportement dépendant du contexte est de plus en plus important pour un large éventail de domaines d'applications mobiles »(Simoninet Carbonell, 2007). Ces dernières sont souvent utilisées pour répondre aux besoins de l'utilisateur moyennant des services spécifiques liés à leurs mobilités.

Les plateformes mobiles sont souvent caractérisées par leurs ressources limitées et par deschangementsde leurs environnements d'exécution.

Les infrastructures des applications traditionnelles présentent quelques limites au niveau de la réaction face au changement, de l'identification du meilleur contexte d'adaptation, del'ajustement des interfaces graphiques en fonction des ressources disponibles des terminaux, de l'ergonomie, etc..

Ce constat est le principal facteur derrière l'infidélité des utilisateurs envers les applications développées. « *Dans une période de 12 mois : Après un mois d'utilisation, seulement 38% des utilisateurs d'appareil IOS ou Android continueront d'utiliser l'application. Après 6 mois, seulement 14% l'utiliseront encore et finalement 2 %* » (Grenier, 2011).

Pour faire face à ces limites et satisfaire les utilisateurs, un système mobile doit être capable d'observer, d'auto-identifier le contexte d'usage le plus pertinent et d'adapter les interfaces graphiques au contenu assimilé.

Toutefois, cela est loin d'être une tâche aisée. Plus spécifiquement, la création d'interfaces a été reconnue comme l'une des tâches les plus fastidieuses dans l'ensemble du cycle de développement d'une application (Yu, 2008).

De ce fait, diverses architectures sont proposées pour affronter ces problèmes et améliorer l'interaction entre l'utilisateur et son appareil mobile.

Sur le plan de la conception, nous pouvons remarquer quelques défaillances telles que : l'absence de formalisme pour garantir l'adaptation du contenu et de l'interface simultanément, l'absence de formalisme descriptif des composants formant l'interface graphique et le manque des points d'observation. Ceci implique la non-couverture de l'ensemble des informations contextuelles. Les contraintes d'intégrités établies entre les différentes entités ne sont pas bien identifiées. Cela pourrait rendre le modèle plus complexe et plus rigide lors du déploiement d'un grand volume de contexte et lors de la mise à jour (Sottet, 2008).

Notre principale question de recherche est formulée de la manière suivante :

"Comment adapter les interfaces graphiques des applications mobiles dépendantes du contexte?"

1.2 Objectifs

Face aux problemes soulevés dans la problèmatique, nous voulons déterminer une approche généraliste basée sur les patrons de conceptions, les règles métiers et une description sémantique de l'interface. L'objectif étant de supporter l'adaptation dynamique des interfaces graphiques des applications mobiles en fonction de la variation d'un grand volume de contexte d'utilisation.L'architecture à proposer doit apporter un gain en temps (par rapport à l'adaptation de l'interface aux changements contextuels) et en qualité (vis-à-vis l'interactivité et la convivalité de l'interface).

1.3 Méthodologie de recherche

Tout d'abord, nous donnons une revue de la littérature des concepts de base tels que la notion de contexte, les différents types d'adaptation, l'interfaçage et les patrons de conception. Cette revue portera également sur les approches et les systèmes existants qui s'inscrivent dans la même lignée que notre vision de recherche.

Par ailleurs, nous allons exploiter la notion des *capteurs d'inférence* pour la détection de la variation du contexte. Aussi, nous allons explorer et dégager les patrons architecturaux que nous jugerons nécessaires afin de supporter le mécanisme d'adaptation d'interface. Cette interface, nous conduira à étudier le formalisme de la plasticité (interface graphique) et la notion des services Web pour la réalisation de l'arrière-plan qui exprime le besoin des utilisateurs.

Également, Nous allons créer un descriptif général des composants d'interface et nous générons des règles pour établir le raisonnement en fonction du contexte.

Aussi, nous allons utiliser une approche basée les patrons de conception et les règles métiers pour l'adaptation ou la création des interfaces des applications mobiles en fonction du changement du contexte.

Pour mettre en pratique notre approche, nous allons mettre en œuvre une application mobile dépendante du contexte de l'utilisateur. À travers cette dérnière, nous effectuons les tests pour évaluer la convivialité des interfaces dans un environnement simulé.

1.4 Organisation du mémoire

Dans le chapitre II, nous mettrons l'accent sur le formalisme des patrons de conception tout en spécifiant leurs cadres d'utilisation, leurs représentations et leurs classifications. Ce formalisme nous sera utile pour l'identification du contexte d'usage, la préparation du contenu, la représentation de l'interfaceainsi que pour garantir le processus d'adaptation.

Dans le chapitre III, nous rappellerons les notions importantes pour l'adaptation des interfaces des systèmes dépendants du contexte. Une revue de la littérature sera présentée. Elle portera sur les concepts de contexte, des capteurs et d'adaptation d'interfaces. Ensuite,

nous présenteronsles approches d'adaptation dépendantes du contexte d'utilisation déjà existantes.

Dans le chapitre VI, nous décrironsnotre approche pour l'adaptation des interfaces d'application mobiles dépendantes du contexte. Par ailleurs, nous détaillerons chaque partie de notre approche basée sur les patrons de conception et les règles métier. Nous présenterons un scénario et des diagrammes synthétisant l'approche.

Le chapitre V est un chapitre de réalisation. Il servira à présenter l'approche d'implantation de notre démarche.

CHAPITRE II

LES PATRONS DE CONCEPTION

Notre problématique d'adaptation des interfaces des applications mobiles en fonction du contexte peut se résoudre de différentes façons, notamment en fonction de l'architecture du système qui supporte le processus d'adaptation.

Les patrons de conception peuvent former une bonne solution à notre problématique.

Les modèles de conception sont des formalismes orientés objet. *« Un patron décrit un problème à résoudre, une solution, et le contexte dans lequel cette solution se situe»* (Johnson, 1997). Ils représentent une solution à un problème spécifique déjà abordé lors de la conception d'autres systèmes tels que la séparation du noyau fonctionnel d'une application de son interface, la détection d'un changement sur un objet, etc.

Dans notre problématique, une interface graphique doit s'adapter à la variation du contexte d'usage. De leurs côtés, les patrons de conception peuvent offrir des solutions à ce type de problème.

Dans la suite de cette section, nous ferons un tour d'horizon du concept de patron de conception.

2.1 Type de patron

Dans son formalisme, « *un patron de conception représente une idée utile, utilisable et utilisée* » (Aubert, 2001). Cette idée prouve le caractère créatif de ces modèles. Il est nécessaire de dégager la meilleure granularité et le niveau d'abstraction le plus fiable pour concevoir un système.

En se basant sur la tâche d'un modèle de conception appelé aussi *Rôle* et sur le domaine dans lequel le modèle s'applique, l'auteur dans (Vlissides, Helm, Johnson et Gamma, 1995) a identifié trois types de patrons :

- *Patrons de création* : ils s'attaquent à la création des objets et offrent des solutions en rapport avec l'instanciation des classes;
- *Patrons de structure* : ils concernent la composition de classes ou d'objets. Par ailleurs, ils offrent des solutions pour la réalisation de nouvelles fonctionnalités;
- *Patrons de comportement* : ils s'intéressent à l'interdépendance entre les classes et les objets ainsi qu'à la délégation des responsabilités.

Généralement, chaque patron de conception possède sa propre représentation. Il est modélisé sous la forme de diagramme orienté objet.

2.2 Représentation des patrons

Les mises en forme de représentation des patrons de conception sont différentes. Dans notre mémoire, nous allons présenter le formalisme utilisé par GOF « *Gang Of Four* »afin de faciliter leurs repérages. Les propriétés utilisées sont les suivantes (El Boussaidiet Mili, 2004 ; Vlissides *et al.*, 1995):

- Nom : il servira par la suite à enrichir le catalogue des patrons;
- Intention : un court descriptif de ce que fait le patron, son but et à quel problème il s'attaque;
- Alias : cite les autres noms communs à ce patron, s'il y a lieu;
- Motivation : indique le contexte dans lequel ce patron s'applique. Il illustre le problème d'une situation donnée moyennant un scénario dans le but de clarifier les descriptions qui suivront;
- Indication d'utilisation : décrit les différentes situations dans lesquelles ce patron peut être employé;
- Structure : description structurelle d'un patron. Elle est faite par le recours aux diagrammes de classe et de collaborations;
- Participant : liste les différents intervenants (classes ou objets) lors de l'application du patron;

- Collaboration : présente sous forme d'une description textuelle les parties prenantes qui interagissent dans le but de dégager leurs responsabilités;
- Conséquences : description des conséquences résultantes de l'utilisation de ce patron. Elles peuvent être positives et/ou négatives;
- Implémentation : c'est la réponse à cette liste de questions (liste non exhaustive) : à quoi s'attendre du programme? Quels sont les pièges auxquels il faut faire attention? Quelles sont les astuces et les techniques à reconnaître lors de l'implémentation?
- Exemple de code : présente les fragments de code élaborés par un langage de programmation orienté objet dans le but d'illustrer la façon avec laquelle ce patron est implémenté;
- Utilisation connue : présente des exemples bien documentés de l'application de ce patron dans des systèmes réels;
- Patrons apparentés : ils listent les patrons en rapport avec celui en cours de traitement et présentent leurs différences.

2.3 Exemple de patron

Suite à la complexité des systèmes logiciels, les informaticiens ont toujours eu recours à la technique de modélisation telle que UML « Unified Modeling Language » (Siauet Halpin, 2001). Elle schématise un patron de conception sous la forme d'un diagramme orienté objet simple. Ce dernier sert à décrire les différents composants ainsi que leurs interactions dans un contexte spécifique. Il est réutilisable par la suite.

Bien que plusieurs patrons de conception existent, il y a une base de modèles à aspect général déjà identifiés dans le catalogue de *GOF*. Cette base sert à supporter la création de nouveaux patrons. Nous en détaillerons les plus utilisés. Parmi ces derniers, quelques-uns seront utilisés dans notre approche que nous présenterons dans le quatrième chapitre. Dans ce qui suit, nous nous attarderons sur les autres patrons.

2.3.1 Le patron de conception « Façade »

Ce patron vise à simplifier la complexité d'un système. Il permet d'encapsuler un sous-système complexe moyennant une interface simple. Habituellement, il fournit une interface

unifiée pour un ensemble d'interfaces disponibles dans un sous-système. Dans ce cadre d'utilisation, *Façade* suggère une vue unique du sous-système afin que les utilisateurs s'adressent directement à leurs fins spécifiques. Par analogie avec notre travail, ce modèle encapsulera la phase de suggestion d'interface en cours d'exécution pour suggérer une unique interface graphique. Elle représentera le premier point d'accès de l'utilisateur à l'application.
Comme l'indique sa structure dans figure A.1 (de l'annexe A), la communication avec le sous-système est garantie moyennant l'envoi des requêtes par les clients à la façade. Ce dernier détermine à quel sous-système la requête est destinée. Il l'achemine sans que le client ne s'en aperçoive et sans qu'il ne se préoccupe de la destination finale.

2.3.2 Le patron de conception « Bridge »

Ce patron est utilisé dès la phase de conception d'un nouveau programme en ayant comme intention le découplage d'une interface (abstraction) et son implémentation. Grâce à cette idée, chaque élément pourra être modifié et invoqué indépendamment de l'autre. Il sert à faciliter la gestion du changement entre les différentes stratégies implémentées.
Comme le montre sa structure décrite dans la figure A.2 (de l'annexe A), les clients envoient leurs requêtes à l'abstraction. Cette dernière s'occupe de la liaison entre les abstractions raffinées et les objets «Implémenteurs ».

2.3.3 Le patron de conception « State »

Ce patron permet de modifier le comportement d'état d'un objet lorsque son état interne change. Étant donné que les valeurs des attributs d'une classe représentent l'état d'un objet, le changement de comportement doit se produire dynamiquement en fonction de la valeur de cet état.
Sa structure représentée dans la figure A.3 (de l'annexe A) indique que ce patron de conception est fondé sur une entité *Contexte* et sur un ensemble d'entités représentant chacun un état possible. Le contexte délègue les requêtes à l'état désiré tout en offrant la possibilité de passer en paramètre sa propre référence.

2.3.4 Le patron de conception « Observer »

Comme son nom et sa structure présentée sur la figure A.4 (de l'annexe A)l'indiquent, il offre la possibilité de l'observation en étant muni d'un processus de notification. Par conséquent, le modèle Observateur/Observable permet à un *sujet* (observable) d'envoyer un signal à un *observateur* suite à un déclenchement d'une certaine action. Ce dernier, étant à l'écoute permanente des événements et du changement d'état, il enregistre les observations et effectue la tâche adéquate en tenant compte des informations enregistrées. L'observateur agit donc d'une manière transparente selon l'état du sujet. Ce dernier, ayant comme tâche principale la notification, il ne se préoccupe en aucun cas du comportement qui vient après.

Ce modèle de conception représente un principe primordial dans notre approche. Avec sa simplicité, il représentera la meilleure solution à notre problème d'acquisition du contexte.

À partir de la grande variété des modèles de conception, nous devrons identifier les plus pertinents pour répondre à notre problématique. Ainsi, les modèles doivent êtrecapables de satisfaire l'adaptation d'interface des applications mobiles suivant le changement contextuel. Dans ce cadre, nous pouvons souligner le grand intérêt de tenir compte du concept d'interface. Cette solution a été choisie pour sa flexibilité, son extensibilité et pour sa capacité à isoler la partie fonctionnelle de l'interface.
Dans le chapitre 4, nous détaillerons davantage les patrons de conception afin de montrer leur intérêt et leur usage pour la modélisation d'une approche supportant l'adaptation des interfaces des applications mobiles dépendant du contexte. Auparavant, dans le troisième chapitre, nous explorerons les différents concepts liés à notre problématique ainsi que les approches existantes.

CHAPITRE III

REVUE DE LA LITTÉRATURE

Dans ce chapitre, nous présenterons les notions de base indispensables pour une meilleure compréhension de notre sujet de recherche. Nous proposerons ensuite un retour sur les approches d'adaptation d'interface dépendantes du contexte d'utilisation déjà existantes.

3.1 Concepts de base

3.1.1 Informatique ambiante

Les dispositifs mobiles permettent d'offrir aux usagers un service continu en tous lieux et tous moments (Weiser, 1991). Un tel environnement a conduit les développeurs à s'orienter vers l'usage des petits terminaux età implanter une grande variétéd'applications. D'où l'apparition d'une nouvelle idéologie dite **informatique ambiante :**

« *L'idée de l'informatique ambiante a d'abord résulté de la contemplation de la place de l'ordinateur dans les activités de la vie quotidienne. En particulier, les études anthropologiques de la vie professionnelle nous enseignent que les gens travaillent principalement dans un monde de situation partagée et de compétence technique non examinée* »(Weiser, 1993).

Pour paraître utile, cette innovation doit faciliter la tâche des utilisateurs et leur offrir la possibilité de s'adapter à leurs environnements.

3.1.2 La notion de contexte

La notion de« *contexte* » a pris beaucoup de place dans la littérature. Elle a attiré l'attention d'un grand nombre de chercheurs dans plusieurs domaines tels que la philosophie, la psychologie, l'informatique, etc.

Dans(Weiser, 1993), cette notion a été introduite en l'associant avec l'informatique ambiante : « *l'ensemble des informations à prendre en compte en vue d'une adaptation* ».

Par la suite, diverses modificationsont été apportées à cette définition comme dans (Schilitet Theimer, 1994). Pour ces auteurs, il s'agit de répondre aux questions suivantes : « *où es-tu?* », « *Avec qui es-tu?* », «*De quelles ressources disposes-tu à proximité?* ». Partant de cette vision, la notion de contexte désigne tout changement qui affecte l'environnement physique, le comportement des utilisateurs et les ressources de la plateforme.

Les auteurs dans (Chenet Kotz, 2000) ont introduit la notion d'information passive et active dans leurs définitions «*Le contexte est un ensemble d'état et de paramètres qui soit détermine le comportement d'une application ou bien dans lequel un évènement d'application se produit et est intéressant pour l'utilisateur* ».

Cependant, avec plus de précision, Deya mis l'accent sur la notion d'entité ainsi que sur la tâche de l'utilisateur en définissant le contexte comme suit: « *le contexte couvre* ***toutes les informations*** *pouvant être utilisées pour caractériser la situation d'une entité.* ***Une entité*** *est une personne, un lieu, ou un objet qui peut être pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application eux-mêmes* »(Dey, Salber, Futakawaet Abowd, 1999). Cette définition étant d'ordre générique, elle permet d'englober tous les travaux antérieurs.

Par la suite, la définition suivante a été proposée : « *Le contexte d'une entité est une collection de mesures et de connaissances déduites qui décrivent l'état et l'environnement dans lequel une entité existe ou a existé* »(Strassner, Samudrala, Cox, Liu, Jiang, Zhang, Meer, Foghlú et Donnelly, 2008).

Partant de cette variété de définitions, le contexte d'usage peut être modélisé par le triplet « Utilisateur, plate-forme, environnement » (Tamine-Lechani, Boughanemet Daoud, 2010):

- *Utilisateur* : désigne une personne de la communauté ciblée par un système informatique. Il est typiquement représenté par sa capacité cognitive, sa perception, ses compétences et ses caractéristiques culturelles;

- *Plate-forme* : désigne une base de travail formée d'une structure matérielle et logicielle qui sert principalement à garantir une interaction. Dans notre étude, nous nous limiterons aux assistants personnels tels que les terminaux mobiles. Ces derniers sont caractérisés par leurs aspects interactifs, la taille réduite de l'écran, etc. ;
- *Environnement* : désigne l'environnement physique dans lequel une interaction devra avoir lieu. Il est représenté par un ensemble d'informations telles que : la température, le niveau de la batterie, la géolocalisation, le bruit, etc.

3.1.3 Les capteurs

L'informatique ambiante vise l'intégration du contexte relative à l'utilisateur au moment où il effectue sa tâche. Les informations du contexte peuvent être déterminées à l'aide de divers moyens physiques tels que les capteurs.Cet appareilpermet de détecterune ou plusieurs grandeurs physiques brutes, la quantifier en une information logique, pour finir par la convertir en une grandeur exploitable.

Les capteurs peuventêtre classés en trois catégories(Baldauf, Dustdaret Rosenberg, 2007):

- *Les capteurs physiques* : ils sont les plus fréquents et les plus utilisés pour fournir une donnée physique telle que la localisation (GPS), la température (Thermomètre), le bruit (détecteur de bruit), etc.
- *Les capteurs virtuels* : ils sont capables d'extraire des informations contextuelles à partir des applications ou des services Web. Nous pouvons citer, par exemple, la détection de l'activité de l'utilisateur moyennant la vérification des mouvements de la souris;
- *Les capteurs logiques* : ils combinent les données extraites à partir des capteurs physiques et virtuels avec des données supplémentaires à partir d'autres sources. L'objectif est de résoudre des tâches plus élevées en se basant sur le raisonnement par la logique.

La tendance des progrès technologiques tourne autour de la miniaturisation des capteurs et leurs intégrations dans des petits dispositifs mobiles permettant à l'utilisateur d'accéder aux informations n'importe où et n'importe quand. Par conséquent, l'idée d'exploiter ces capteurs, dans des applications qui offrent des services aux utilisateurs, rend possible la mise en œuvre d'applications omniprésentes.

De tels systèmes, capables de mesurer divers phénomènes physiques à travers des capteurs englobés dans des unités mobiles telles que les téléphones intelligents, ouvrent des perspectives intéressantes pour la mise en œuvre d'applications mobiles dépendantes du contexte.

Ces systèmes doivent cependant suivre une discipline rigoureuse de construction de logiciels adaptés à de telles réalités.

3.1.4 Les systèmes sensibles au contexte

Dans le domaine de l'informatique ambiante, les systèmes dépendants du contexte ont créé un vaste champ de recherche. Ce champ est basé principalement sur la prise en compte du contexte de l'utilisateur dans des situations différentes. Partant de ce fait, les chercheurs ont commencé à s'attaquer au problème de la mobilité dans le but de rendre le contexte invisible pour cet utilisateur.

Un système est dit sensible au contexte s'il est capable de s'adapter aux changements contextuels en temps réel. D'où l'apparition de la définition la plus utilisée par la plupart des chercheurs dans leurs contributions, celle de Dey« *Un système est sensible au contexte s'il utilise le contexte pour fournir des informations et des services pertinents pour l'utilisateur, ou la pertinence dépend de la tâche demandée par l'utilisateur* » (Abowd, Dey, Brown, Davies, Smithet Steggles, 1999).

Cette définition a mis en évidence trois étapes indispensables pour qu'un système soit qualifié de « sensible au contexte » :

- Capture et stockage du contexte d'une façon transparente pour l'usager ;
- Interprétation du contexte capté afin de changer l'état de représentation des services. Ces derniers seront capables d'être exploités par les utilisateurs finaux de l'application à travers des actions à exécuter. Ces tâches forment implicitement les préférences des usagers ;
- Passage des informations interprétées à l'application afin de garantir l'exécution des services selon le changement contextuel sans l'intervention explicite des utilisateurs.

Pour mettre les étapes précédentes en évidence dans une application sensible au contexte, il est nécessaire de répondre aux deux questions suivantes (Chaari, Laforestet Flory, 2005):

- Comment concevoir une architecture garantissant l'adaptation au contexte dynamiquement au cours de l'exécution?
- Comment concevoir l'application, elle-même, pour qu'elle s'adapte au contexte?

Une architecture sensible au contexte a été proposée à travers ContextToolKit (voir Figure 3.1).

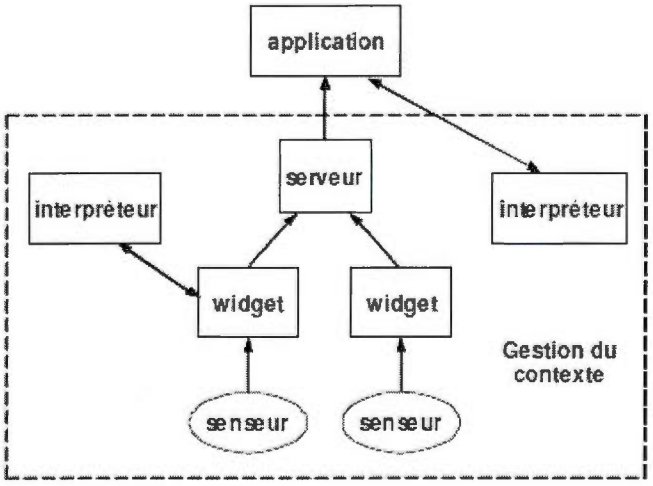

Figure 3.1 Le contexte ToolKit de Dey (Chaari *et al.*, 2005)

Les capteurs se chargent de capter le contexte physique de l'environnement et de le passer au widget. Ce dernier délègue les interprétations d'information aux interpréteurs externes qui s'occupent de les stocker dans des serveurs. L'application sera notifiée à chaque changement contextuel.

Comme nous l'avons évoqué précédemment, plusieurs travaux ont traité la notion de sensibilité au contexte. Néanmoins, ce domaine se trouve toujours face à diverses difficultés telles que : (i) la notion du contexte qui ne cesse d'évoluer; (ii) le repérage du contexte;(iii) l'interprétation et la représentation des informations.

Malgré ces problèmes, nous croyons que la sensibilité au contexte est un facteur primordial pour la création d'applications dans le domaine de l'informatique omniprésente capable de s'adapter au changement.

3.1.5 L'adaptation

Partant de l'hypothèse que n'importe quel système en informatique ambiante est un système adaptable, c'est-à-dire capable de prendre plusieurs formes par le biais d'un choix de l'utilisateur et/ou un changement dans l'environnement, nous évoquons le concept d'adaptabilité.

Dans cette section, nous nous limiterons à laquestion « *À quelles parties de l'environnement le système s'adapte-t-il?* »(Holland, 1992).

L'adaptation avec son aspect dynamique peut affecter trois niveaux de l'application (Chaari *et al.*, 2005):

1. Flux de données entre les services et l'utilisateur : adaptation de contenu,
2. Interface utilisateur : adaptation de présentation,
3. Fonctionnement des services offerts à l'utilisateur : adaptation de comportement.

L'adaptation du contenu consiste à développer des contenus dédiés aux utilisateurs à travers la modification des propriétés des données. Une fois modifiées, elles passent par le processus d'adaptation de la présentation au sein duquel les interfaces sont déjà générées automatiquement par les développeurs. Ces derniers font en sorte que les interfaces soient conformes à divers facteurs tels que le flux de données et les capacités du terminal.

L'adaptation de présentation : La présentation peut être soit statique (déjà prédéfinie par le concepteur de l'application), soit construite en cours d'exécution. À cet instant, elle évolue avec l'exécution. Elle est donc dite dynamique.

L'adaptation du comportementa été divisée en deux. « *Une adaptation statique pour la sélection de l'algorithme adéquat en rapport avec les exigences statiques du système et l'adaptation dynamique qui propose la modification d'un algorithme durant l'exécution à chaque fois où l'opportunité s'offre* »(Fayadet Johnson, 1999).

Dans le domaine de l'informatique omniprésente, l'adaptation rend un système capable de modifier son comportement d'une façon automatique. Partant de cette idée, nous allons proposer un mécanisme d'adaptation permettant d'adapter les interfaces des applications mobiles à leurs contextes d'utilisation.

3.1.6 Interfaces

La technologie a permis la miniaturisation des dispositifs portables. Nous assistons au passage de l'utilisation des PC à l'utilisation des téléphones intelligents et des tablettes. Parallèlement, le volume de données n'a pas cessé de croître. Ainsi, l'utilisateur dispose d'une source importante d'informationsqu'il désire souvent utiliser selon les ressources disponibles telles que la taille de l'écran de l'appareil mobile utilisé. Cette analogie prouve un double constat. D'un côté, la difficulté de remplir l'interface entière lors de la représentation des données sur un PC, et de l'autre, la complexité d'adapter l'interface limitée d'un mobile.

Nous pouvons dégager trois familles d'interfaces pour les dispositifs mobiles : les interfaces préconçues, les interfaces préprogrammées et les interfaces à base de modèles. Dans ce qui suit, nous citerons leurs principales caractéristiques (Joyeux, 2006):

Les interfaces préconçues

Ce type d'interface est fixé à l'avance par le développeur de la plateforme sans exposer tous les composants à l'utilisateur final. La totalité des facettes est stockée dans le terminal. La partie indispensable est exposée tandis que la deuxième partie peut être sélectionnée par le client.

L'inconvénient majeur de ce type d'interfaces réside dans le fait que le développeur doit prévoir les divers besoins des utilisateurs et les inclure dans son application pour satisfaire toute la communauté des usagers potentiels.

Les interfaces programmées

Ce type d'interface est également fixé à l'avance par le développeur. La seule différence réside dans le fait que les facettes sont exposées en totalité,et non partiellement, à l'usager.

Ceci rend ces interfaces souvent inflexibleset non modelables vis-à-vis des finalités réelles des usagers.

Les interfaces à base de modèle

À la lumière de la définition des patrons de conception présentés précédemment « *Un patron représente une solution précieuseà un problème spécifique en rapport avec un contexte d'exécution bien déterminé* ». Ces modèles ont été utilisés pour la conception des interfaces Homme-Machine (IHM).

Cette idée a été accentuée en 2000 par CHI Workshop « *la description d'un modèle doit représenter précisément la solution à un problème dans le but d'être réutilisé dans le cadre de système interactif. Il doit inclure le nom des composants, leur classement, des exemples clairs, le contexte, les justifications, la solution, des croquis, des références à d'autres patrons, une synthèse et les crédits* ».

Afin de satisfaire la création desinterfaces, le concepteur doit absolument sélectionner les patrons de conception à utiliser et les adaptater selon le domaine d'application. Cette démarche est résumée dans le schéma ci-dessous.

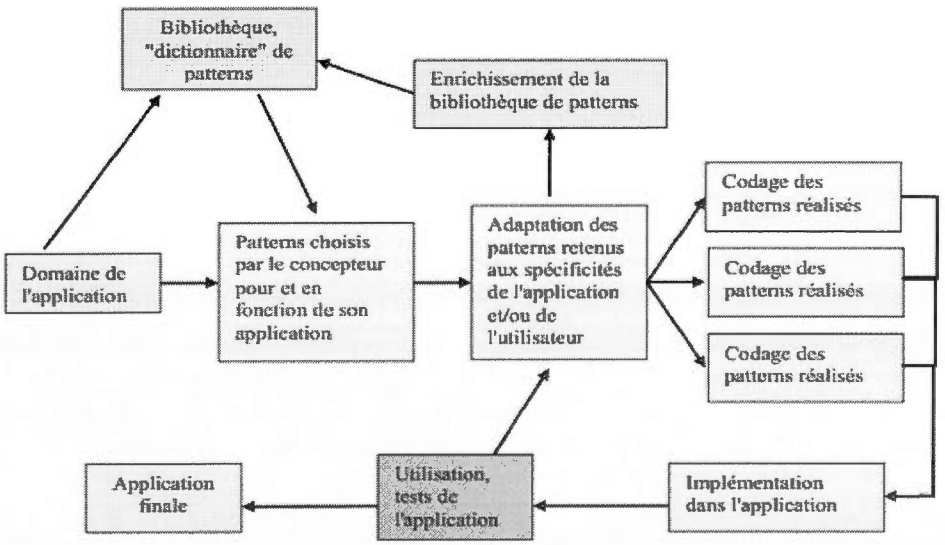

Figure 3.2 Principe d'utilisation de patron (Joyeux, 2006)

3.2 Les approches existantes

Plusieurs travaux effectués ont pour objectif de développer des applications dépendantes de leurs contextes d'interaction.Ce constat a stimulé l'apparition de la propriété de *Plasticité* des IHM, énoncée comme l'aptitude d'adapter les interfaces aux caractéristiques physiques du système et à l'environnementtout en préservant la facilité d'utilisation (Theveninet Coutaz, 1999). De ce fait, une telle adaptation est devenue indispensable.

Pour faire face à ce problème, plusieurs travaux ont été publiés. Ces derniers ont été fondés sur la base de diverses constatations(Ganneau, Calvaryet Demumieux, 2008) :les activités de l'utilisateur changent en fonction de la variation de son contexte.

La majorité des travaux portant sur la plasticité des interfaces utilisateurs a rencontré certaines difficultés lors du traitement du contexte d'usage(Calvary, Coutaz, Thevenin, Limbourg, Bouillonet Vanderdonckt, 2003). Ces recherchesse sont penchéesuniquement sur l'ajustement de l'interface en fonction de la taille du périphériqueen négligeant la partie fonctionnelle du système. Cela est dû à la grande diversification du contexte.

Afin de remédier au problème de la diversification du contexte, la plupart des travaux ont tenté de répondre à la problématique d'adaptation d'interface en se basant surla démarche proposée dans le système *CAMELEON (Bandelloniet Paternò, 2004)*. Cette approche a divisé le processus de développement enquatre étapes, comme le montre la figure 3.3: tâche et concept, interface utilisateur abstraite, interface utilisateur concrète et interface utilisateur final.

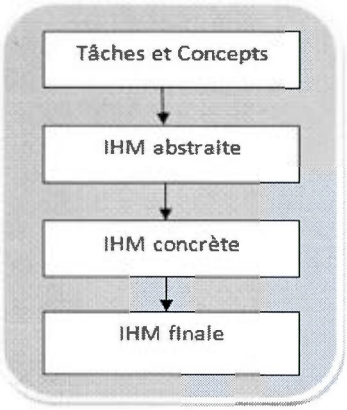

Figure 3.3 Étapes de l'approche CAMELEON

Deux principaux courants peuvent être identifiés. Dans le cadre de cette section, nous les détaillerons en distinguant l'approche s'articulant autour des composants et celle autour des modèles.

Approche basée sur les composants

Généralement, une interface homme-machine est composée d'un ou plusieurs composants. Afin d'adapter les interfaces au contexte d'utilisation, une approche basée sur l'assemblage dynamique de composants a été proposée.

Dans (Hariri, Lepreux, Tabary et Kolski, 2009), les auteurs ont ajouté une base de connaissances à la structure métier. Elle comporte les différents éléments de la présentation (c-à-d. les composants d'interface). Ces derniers sont prédéveloppés et stockés pour satisfaire un besoin bien déterminé. Cette idée est venue supporter la prise de décision du système composé par les éléments suivants :

Tableau 3.1 Composition et rôle de chaque composant de la structure métier

Composants	Rôle
Interface présentation, Interface fonctionnelle, base de connaissance	Déterminer l'interface à utiliser par rapport au changement contextuel
2 Composants fonctionnels	Proposer la tâche (fonctionnalité) en fonction du contexte
2 Composants de présentation	Représentent l'interface choisie
Base de connaissances	Prise de décision de l'interface

Le schéma suivant illustre la démarche de l'approche à base de composants proposée:

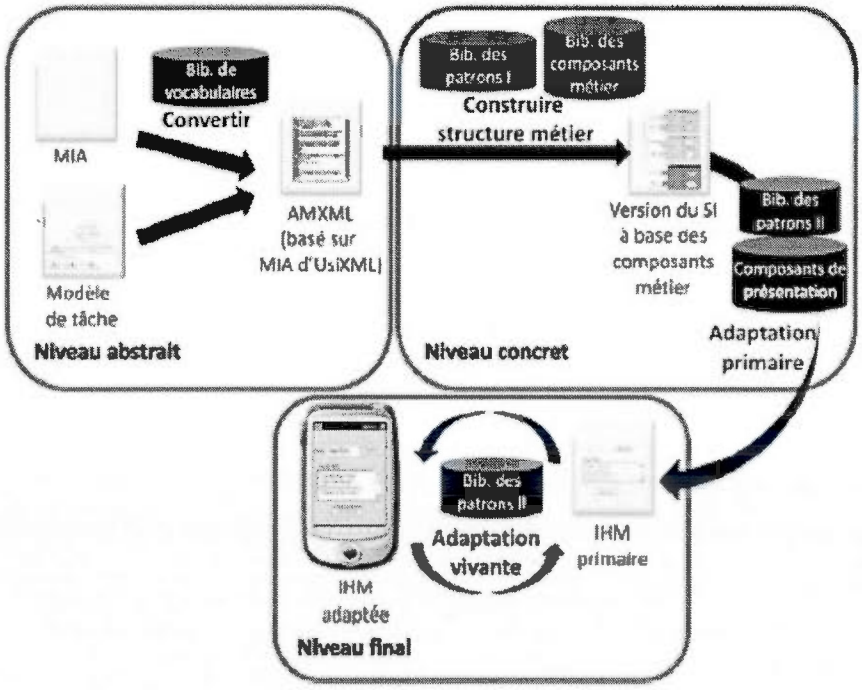

Figure 3.4 Démarche globale pour la génération d'interface plastique(Hariri *et al.*, 2009)

Selon cette approche, le processus d'adaptation suit les étapes suivantes :

1. Dans la partie abstraite : En se référant au type de définition du document (TDD) et à la bibliothèque de vocabulaire, le modèle d'interface Abstrait (MAI) et celui de tâches sont transformés en un modèle abstrait dans XML. Ce dernier comprend la description des tâches du système, les relations entre elles et une partie du contexte de l'utilisateur.
2. Dans la partie concrète :
    - Construction de la structure métier : les patrons de conception choisissent le composant métier en fonction de la tâche spécifiée dans l'AMXML. Ces modèles sont déjà identifiés à l'avance par le concepteur selon le domaine de l'application.
    - Adaptation primaire : Les patrons de conception sélectionnent le composant de présentation le plus adéquat. Ces patrons sont toujours choisis relativement au contexte à partir du catalogue de patron.
3. Dans la partie finale, il s'agit de l'étape de l'adaptation vivante en cas de changement contextuel. La BC *« base de connaissance »*sélectionne des nouveaux composants.

Approche basée sur les modèles

Selon la démarche d'une approche d'Ingénierie Dirigée par les Modèles, une proposition basée sur les métamodèles et les règles a été suggérée dans le but de garantir l'adaptation d'interface au contexte. Une telle proposition possède un double constat. D'une part, elle incite les concepteurs à se pencher sur les règles d'adaptation. D'autre part, elle soutient l'harmonisation systématique du code source à l'instant coïncidant avec l'identification du contexte.

Dans (Ganneau, Calvary et Demumieux, 2007), les auteurs ont divisé le processus d'adaptation en trois modules : la reconnaissance du contexte, le calcul et la réaction. Cette décomposition est illustrée dans la figure ci-dessous (3.5).

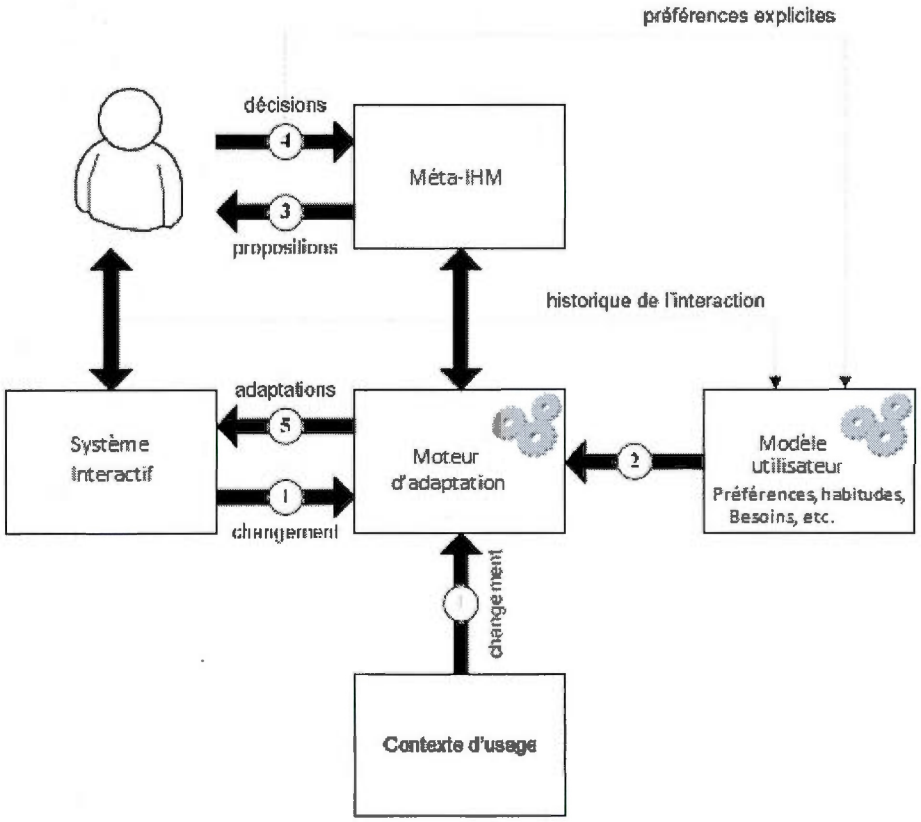

Figure 3.5 Décomposition fonctionnelle de la plasticité (Ganneau *et al.*, 2007)

La réaction, selon cette décomposition, dépend du contexte d'usage et du système interactif qui exprime les préférences de l'utilisateur. Le rôle principal d'un concepteur est de définir un ensemble exclusif de règles au sein du moteur d'adaptation. Ces règles jouent un rôle important pour l'adaptation des interfaces à leurs contextes. Il s'agit d'un module de transformation par paramètre du contexte(Sottet, 2008).

Pour illustrer cette approche, nous pouvons citer l'exemple suivant : « *en fonction de la langue utilisée par l'utilisateur sur son terminal, des interfaces en anglais ou en français seront créées parle module de transformation* ». Toutefois, lors de l'apparaition d'un nouveau contexte (Modèle 2 de contexte comme le montre la figure 3.6), de nouvelles règles doivent être ajoutées.

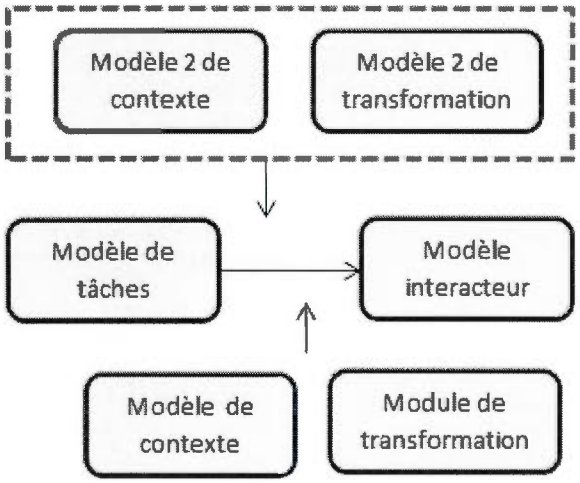

Figure 3.6 Transformation spécifique à une application et un contexte particulier(Sottet, 2008)

Parmi les points forts de cette approche, nous pouvons dégager : la capitalisation et une meilleure manipulation des connaissances. Tandis que la modélisation des transformations semble être non générique, les règles doivent être modifiées avec chaque changement de contexte. Cette modification nécessite l'ajout d'une extension au sein du module de transformation.

De ce fait, l'approche dans (Hachaniet Dupuy-Chessa, 2009) est venue remédier aux problèmes rencontrés dans(Sottet, 2008). En effet, d'une part, elle a ciblé l'adaptation des tâches des utilisateurs. D'autre part, elle a visé à faciliterle travail du concepteur moyennant la généralisation des règles de transformation.

Cet objectif a été atteint en s'inspirant de la propriété de ligne de produits (Babar, Chenet Shull, 2010). Dans ce cadre, une extension basée sur les arbres de tâches avec des points de variation a été ajoutée aux métamodèles de tâches cibles spécifiques. Ce dernier, déjà existant dans la proposition de Sottel, a été enrichi par les auteurs moyennant deux autres métamodèles sources(Hachaniet Dupuy-Chessa, 2009) :

1. Méta-modèle générique de contexte nécessaire pour la description du contexte d'usage;

2. Méta-modèle de tâches adaptables au contexte nécessaire pour faire les liens entre l'interface et les paramètres du contexte.

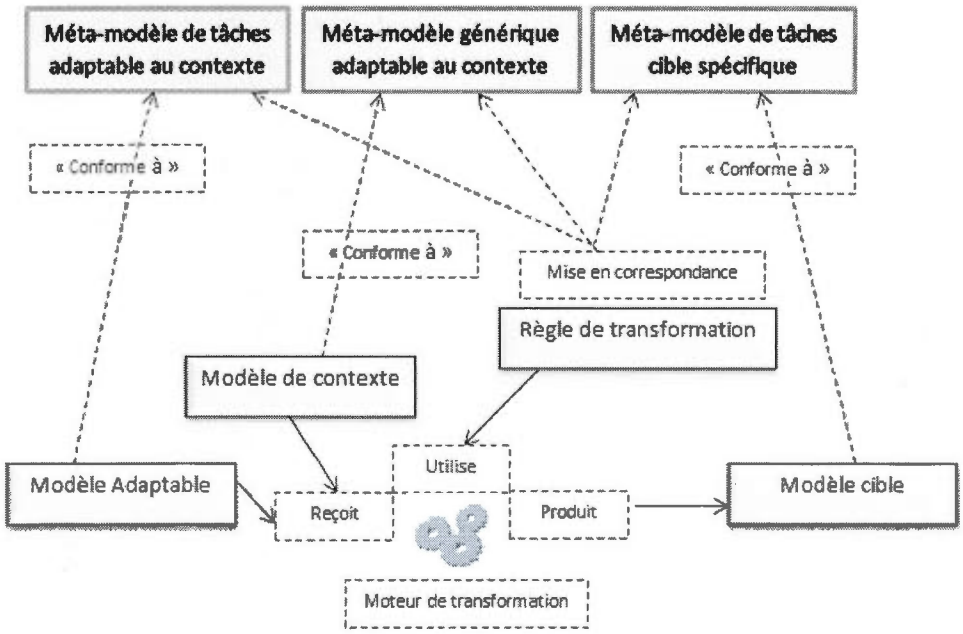

Figure 3.7 Cartographie de l'approche proposée (Hachaniet Dupuy-Chessa, 2009)

La première intention de cette approche était de bien identifier le contexte et de distinguer s'il s'agit d'une entité fixe ou variable. L'accent a été mis sur l'entité variable qui a été modélisée par des points de décision. Ils reflètent l'état du contexte et sa variation. Ainsi, chaque point est associé à un paramètre spécifique du contexte. Une fois ces points identifiés, le modèle de tâches adaptables est créé comme le montre la figure A.5 (de l'annexe A).

Dès la détection d'une variation dans le contexte, l'application s'adapte en choisissant le scénario adéquat et en réduisant la variabilité du modèle. Cette réduction est garantie par le module de transformation générique qui prend en entrée le modèle de tâchesadaptables de la figure A.5 (de l'annexe A) et le modèle de contexte de la figure A.6 (de l'annexe A).

C'est sur cette base que les choix d'adaptation seront fixés. La transformation du modèle cible aux modèles sources est satisfaite. Le résultat obtenu est matérialisé par la naissance d'un modèle de tâche spécifique au contexte tel que le montre la figure ci-dessous :

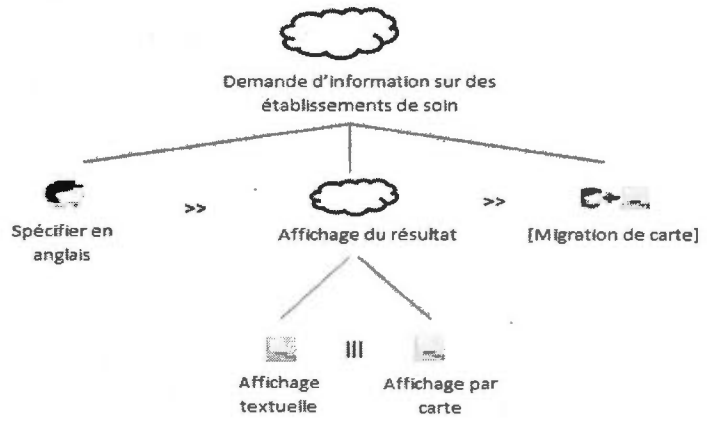

Figure 3.8 Modèle de tâche spécifique au contexte (Hachaniet Dupuy-Chessa, 2009)

L'approche de rapprochement est illustrée dans la figure suivante 3.9

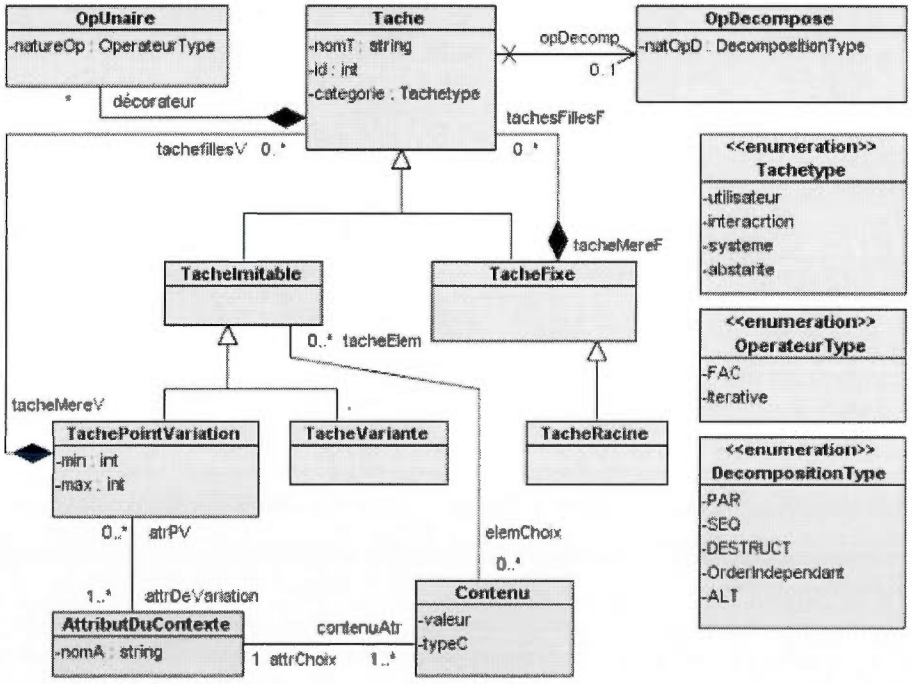

Figure 3.9 Métamodèle de tâche adaptable au contexte(Hachaniet Dupuy-Chessa, 2009)

Les classes clés du «Métamodèle de tâches adaptables au contexte » sont les suivantes :*Attribut du Contexte* (identifie le contexte, cet attribut considère la tâche comme étant un point de variation) et le *Contenu* en relation directe avec la classe *tâche imitable*. Cette dernière facilite la distinction des différentes possibilités d'adaptation.

La contribution principale de cette approche à base de modèle est la suggestion d'un prototype générique et opérationnel pour supporter la variabilité, proposer une meilleure structuration du contexte et garantir l'adaptation d'interface lors d'une situation donnée. Les règles de transformations introduites au sein de ce modèle ont facilité la tâche des concepteurs d'application. Ces derniers sont obligés d'identifier le contexte à utiliser et de définir le modèle de tâche adaptable, uniquement. Par ailleurs, les règles s'occupent de toutes les transformations.

Le métamodèle de tâches adaptables proposées ne manque pas de défaillance. À ce niveau, nous pouvons remarquer que les contraintes d'intégrité établies entre les différentes entités ne sont pas bien identifiées. Cela pourra rendre le modèle plus complexe et plus rigide lors du déploiement d'un grand volume de contexte. Il convient aussi de noter que ce modèle ne satisfait pas l'adaptation au moment de l'exécution.
Ce manque est lié à la non-existence d'un moyen capable de transformer le modèle résultant en un code source. Ainsi, le modèle ne couvre aucune description des composants de l'interface vu que l'auteur a mis l'accent uniquement sur la notion de contexte.

Toutes ces approches traitent le problème de l'adaptation d'interface en fonction de la variation de contexte d'utilisation. Cependant, ces solutions ne traitent pas une large gamme de contextes et manquent d'un processus de raisonement sur ces derniers pour s'adapter aux besoins des utilisateurs. Par conséquent, nous allons présenter dans le chapitre 4 notre méthodologie pour contourner ces limites.

CHAPITRE IV

MÉTHODOLOGIE PROPOSÉE

Un servicedépendant du contexte doit être en mesure de réaliser plusieurs tâches : capturer les différents changements environnementaux d'un utilisateur provenant de diverses sources, établir la meilleure interprétation de ces informations et les exposersur une interface. Ceci doit se faire en respectant les limites des ressources à utiliser (mobile, tablette, etc.) et en optimisant leurs expositions.

Afin de répondre à cette problématique et résoudre les insuffisances des travaux précédents dans le domaine de la plasticité des interfaces telles que:

- la prise en compte d'une partie limitée du contexte ;
- le manque de processus supportant l'adaptation ;
- l'absence d'un formalisme de spécification des composants d'interface.

Nous proposons lamodélisationd'une infrastructure qui supporte ces différentes tâches.

Dans le but de réaliser notre objectif, nous allons utiliser les patrons de conception(Aubert–Olivieret Beugnard–Antoine, 2001).Ceux-ci sont reconnus comme des techniques de conception servant à répondre à des problèmes déjà rencontrés. Ils nous seront utiles pour: a) l'identification de tout contexte en relation directe avec l'utilisateur et b) l'élaboration d'un modèle supportant l'adaptation de l'interface. Aussi, nous utiliserons une architecture basée sur les règles(Klemisch, Weber et Benatallah, 2013). Cette architecture permettra la suggestion des interfaces en cours d'exécution à travers un modèle décrivant la spécification de l'interface.

4.1 Motivation

La variation dans l'environnement d'exécution, qui caractérise les terminaux mobiles, ainsi que leurs ressources limitées représentent un terrain favorable pour la mise en œuvre de notre approche. Cette dernière a pour objectif de garantir une autoadaptation. Elle offre une interface unique à l'utilisateur en fonction de ses préférences et des capacités du dispositif qu'il utilise tout en étant adaptéeà sa situation environnementale instantanée. Notre approche fait partie des travaux établis sur la plasticité des interfaces.

La plasticité des interfaces dénote sa capacité à s'adapter au contexte d'usage « utilisateur, plate-forme, environnement » dans le respect de son utilisabilité(Theveninet Coutaz, 1999). Elle représente une forme d'adaptation du type « une action nécessite une réaction ».

Une action peut être soit l'entrée, soit la sortie d'un contexte(Schmidt, 2000). D'où le passage d'un contexte C1 à un contexte C2 déclenche une action et entraîne une réaction qui a un impact sur les interfaces.

Un exemple d'adaptation consiste à redimensionner l'interface en fonction de la taille de l'écran comme le montre la figure ci-dessous.

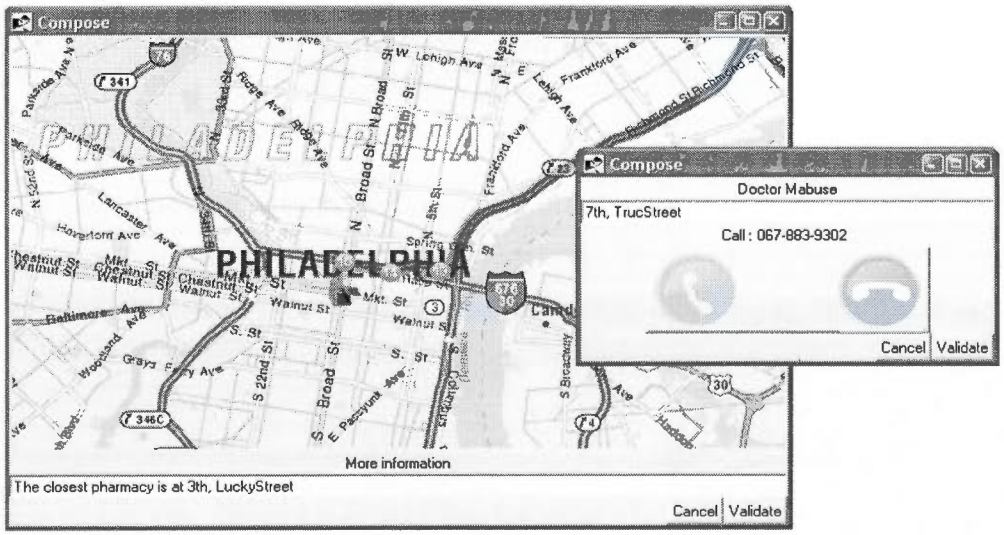

Figure 4.1 Exemple d'adaptation (Gabillon, Calvaryet Fiorino, 2011)

La redistribution de l'interface requiert la génération d'une nouvelle interface ajustée au nouveau contexte en respectant son domaine de plasticité(Calvary et Coutaz, 2002)(voir figure 4.2).

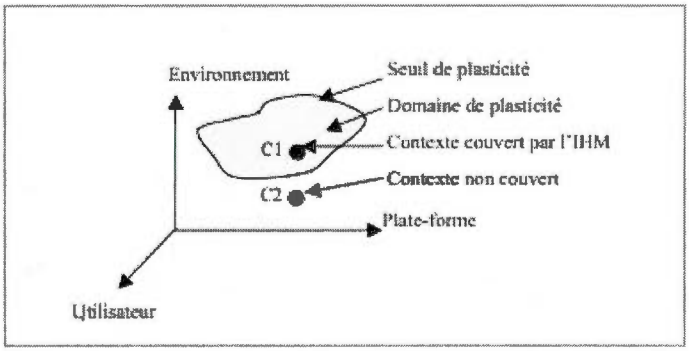

Figure 4.2 Domaine de la plasticité des interfaces(Calvary et Coutaz, 2002)

Dans le cas de l'entrée d'un nouveau contexte C2 pour remplacer le contexte déjà exposé C1. Si les deux contextes couvrent toujours la même interface (Figure 4.3(a)), alors l'interface reste la même. Si le nouveau contexte dépasse le seuil de l'interface déjà exposée (Figure 4.3(b)) alors, la redistribution de l'interface est nécessaire.

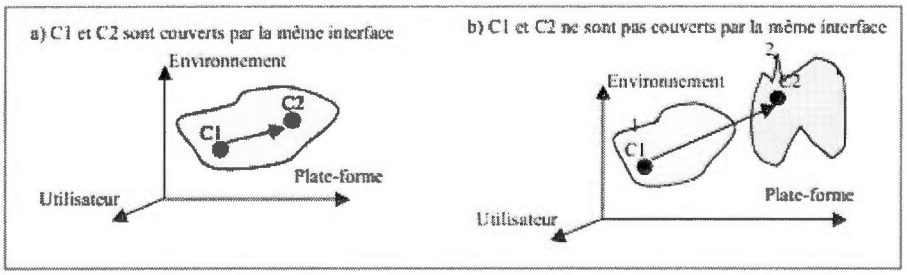

Figure 4.3 Domaine de plasticité et changement du contexte(Calvaryet Coutaz, 2002)

La plupart des organisations de développement mobile se sont orientées vers l'utilisation des architectures baséessur les services Web afin de garantir l'intégration des données et la communication entre le logiciel et les informations situées sur des serveurs distants.
Cette idée,qui a accentué le phénomène d'adaptation, nous a conduits à utiliserun paradigme de services Web dans le cadre de notre travail.

Notre but est de soutenir les concepteurs et les développeurs dans le développement d'applications mobiles et leur permettre d'offrir de la façon la plus simple un service en fonction du contexte formé du triplet (utilisateur, plateforme, environnement). Ces services sont présentés aux utilisateurs sous forme d'interface composée d'un certain nombre de composants (textbox, listview, etc.). Ces interfaces doivent être calculées au cours de l'exécution.

4.2 Structure de l'approche

L'étude du domaine de la plasticité des interfaces nous a permis de connaître les conditions nécessaires afin qu'un système satisfasse l'adaptation des interfaces des applications mobiles au contexte.Àla fois, les informations du service à représenter, la situation environnementale de son invocateur et les propriétés des composants d'interfaces doivent être clairement identifiées et partagées entre les différentes entités du système(Yauet Liu, 2006).

Notre approche doit donc avoir un mode de fonctionnement bien structuré afin d'intervenir au niveau de tout problème de ce genre. Par ailleurs, nous identifions les étapes suivantes :

1. Acquisition du contexte. Cela consiste à:
   - Acquérir la situation de l'utilisateur à partir des capteurs;
   - Acquérir les informations sur les services à partir du service Web(Brown, Johnstonet Kelly, 2002).
2. Raisonnement sur les informations recueillies avec un moteur de gestion de règles métier pour former le *backend* de l'interface;
3. La prise de décision del'unique interface à exposer sur le terminal. Cette interface doit être générée automatiquement. Cette tâche sera établie par une approche à base de règles;
4. La gestion de l'adaptation, nous distinguerons ici deux types d'évènements: OnAction et OnChange.

Nous suggérons une approche basée sur les patrons de conception d'une part et sur un système de suggestion d'interfaces à base de règles d'autre part,pourrésoudre les différentes facettes du problème.

Une fois le domaine de l'application défini par le concepteur, notre patron de conception va gérer *l'acquisition du contexte.* Cette étape peut être faite par un mécanisme supportant l'observation du changement et la notification.

Partant du contexte recueilli, le modèle doit *raisonner* sur ces informations (c-à-d. comparer les données contextuelles du service Web avec ceux de l'utilisateur) afin de former le backend de l'interface.

Ensuite, pour *prendre la décision*de l'interface à exposer, le patron de conception délègue la tâche de la suggestion d'interface à un sous système à base de règles. Cette interface sera créée, suivant un modèle qui spécifie la description des composants de l'interface. Une fois créé, le patron est alors prêt à afficher l'interface à l'utilisateur et *gérer l'adaptation* suivant les nouveaux changements contextuels. La démarche à suivre est illustrée dans la figure 4.4

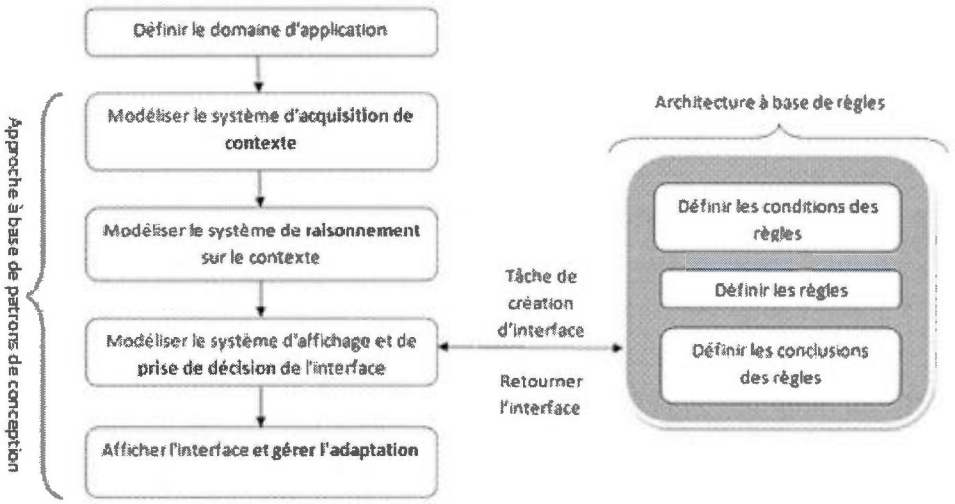

Figure 4.4 Démarche d'adaptation d'interface au contexte d'usage.

Dans ce qui suit, nous exposerons la structure générale de notre approche en spécifiant le rôle de chaque composant du système

4.2.1 Acquisition du contexte

Nous avons identifié deux sources de contexte :

- les informations liées à l'environnement de l'utilisateur
- les données spécifiques au service.

Le patron *OBSERVER* (voir chapitre n° 2 portant sur les patrons de conception) offre une meilleure solution à ce type de problème. En observant un sujet (observable), il détermine la présence d'un nouveau contexte et notifie son observateur qui enregistre ses informations en agissant d'une manière transparente vis-à-vis du client.

Par ailleurs, afin d'acquérir les informations contextuelles de l'utilisateur, nous avons placé un observateur sur le terminal(mobile de l'utilisateur). Ce dernier jouera le même rôle que le *sujet*. Les capteurs intégrés dans l'appareil mobile sont en mesure de détecter les différentes variations du contexte de l'utilisateur telles quela température, le niveau de bruit, la luminosité, l'humidité, la position géographique, etc. Ils détectent également les variationsliés à la plateforme tels que la version du système d'exploitation, la taille de l'écran, la langue. Toutefois, lors de chaque détection de changement,une notification sera envoyée à l'observateur afin qu'il puisse effectuer le comportement adéquat tout en agissant sur l'ensemble de l'information extraite.

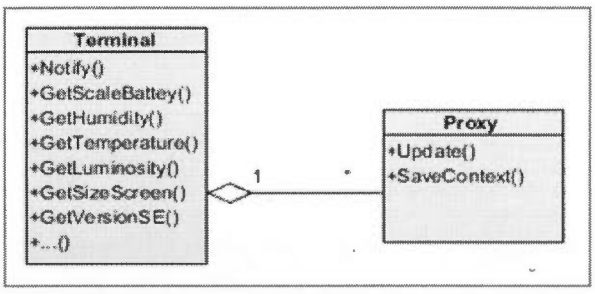

Figure 4.5 Acquisition du contexte de l'utilisateur du terminal avec le patron *Observer*

Nous avons utilisé les capteurs du terminal afin d'acquérir les informations liées à l'utilisateur et à la plateforme qu'il utilise. Ces données ne peuvent en aucun cas représenter le backend de l'interface à exposer à l'utilisateur. Pour faire face à ce genre de problèmes,

nous allons appliquer une architecture orientée service, soit l'architecture REST (Representation State Transfert) que nous présentons dans la section suivante.

4.2.1.1 L'architecture REST (Respresentation State Transfert)

REST a été présentée dans(Fielding, 2000)comme étant une architecture orientée service (AOS) tout comme XML-RPC et SOAP. Alors que ces derniers représentent des standards et respectent une spécification W3C pour la description, REST n'est qu'un style d'architecture qui permet d'accéder facilement à des ressources distantes moyennant une architecture logicielle simple.

Une ressource est un descriptif constitué des opérations et des données spécifiques à l'utilisationpour que cette ressource soit exécuté selon une certaine chronologie. Ces descriptifs sont stockés sur des serveursWeb distants sous forme de fichiers (XML, HTML ou JSON). Pour y accéder, l'organisation distributrice du service doit fournir l'adresse de ce dernier sous la forme d'une URL afin que les utilisateurs d'application l'invoquent à travers l'envoi des requêtes aux serveurs sans se préoccuper de la façon de son utilisation, ni comment il est exploité. L'architecture REST suggère donc d'exploiter ces ressources à travers quatre méthodes simples, soit (voir figure 4.3) :

GET : Pour la lecture d'une ressource;

POST : Pour la création d'une ressource;

PUT : Pour la modification d'une ressource;

DELETE : Pour la suppression d'une ressource du système;

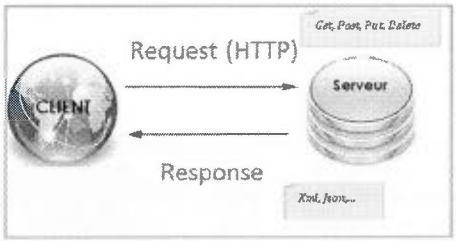

Figure 4.6 Architecture REST

Vu que nous nous retrouvons dans un milieu dynamique où les services ne cessent d'évoluer, un mécanisme d'identification des changements sur le Web semble être indispensable pour

l'acquisition des données spécifiques aux services.Ces informations représentent le backend de l'interface.

L'idéé derrière l'utilisation du patron *Observateur* ne se restreint pas à l'observation d'un seul sujet. Ce dernier, exprimant tout contexte d'utilisation, peut subir divers changements. En effet, l'entité *proxy* du modèle *Obervateur* est capabled'écouter les évenementsappliqués sur tous les sujets en même temps. À cet égard, nous appliquons de nouveau le modèle de conception *Observer* afin d'observer les changements liés au service.

Dans notre cas, le service Web représente notre second sujet d'observation alors que l'observateur sera le même déja appliqué au terminal. La figure suivante rassemble les deux cas d'utilisation du patron *Observateur* pour l'identification à la fois du contexte d'utilisateur et les informations du service. Partant de ces informations, le backend de l'interface sera créé par la suite.

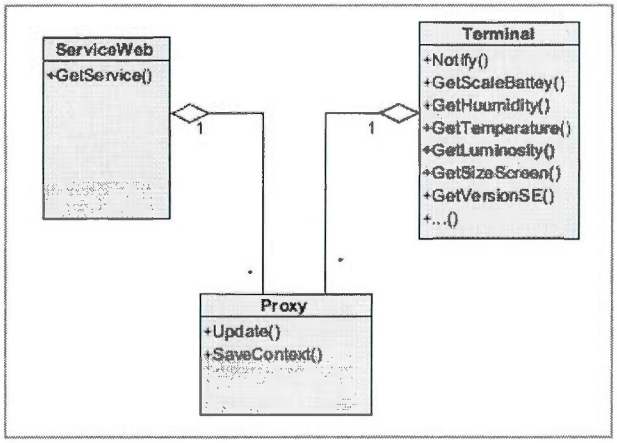

Figure 4.7 Acquisition du contexte par Observateur

En appliquant le patron *Observer* dans notre approche, nous avons réussi à répondre à la problématique d'acquisition du contexte spécifique à l'utilisateur, l'environnement et la plateforme à travers l'observation du terminal et les informations du service à exposer moyennant l'observation des changements sur les services Web.

Dans la partie suivante, nous allons détailler le comportement à avoir lieu en fonction des informationsextraites. Il s'agit d'un raisonnement sur les informations recueillies pour former le backend de l'interface.

4.2.2 Raisonnement

L'idée derrière l'utilisation des patrons de conception dans l'adaptation d'interface au contexte est de séparer le noyau fonctionnel de l'application de l'interface. Toutefois, l'observateur que nous allons nommer *Proxy* dans la suite de notre travail va jouer le rôle d'un intermédiaire entre l'entité responsable de la notification *Sujet* et le reste des entités formant le processus d'adaptation.

Cependant, une fois notifié par un changement contextuel,le *Proxy* récupère les informations acquises du terminal et du service et les transmet au moteur de règles afin de comparer les données contextuelles liées au service, par exemple la température, le bruit, l'humidité, etc. avec celle de l'utilisateur.

Ce raisonnement est géré moyennant les règles métiers pour finir par retourner le backend du service à représenter.Un moteur de règles «*est un système capable de définir des règles et de les appliquer à des faits* »(Lecoz, 2010).

Nous avons choisi d'utiliser le patron de conception *Spécification*(Eric Evans, 1999). Sa structure est présentée sur la figure (A.7 de l'annexe A). Il représente une solution légère, réutilisable pour la gestion des règles métier et pour le traitement des problématiques complexes.

La figure 4.7 modélise la fusion de l'étape d'identification du contexte vue à la section 4.2.1 (patron *Observer*) et de raisonnement avec le patron *Spécification* basé sur les règles.

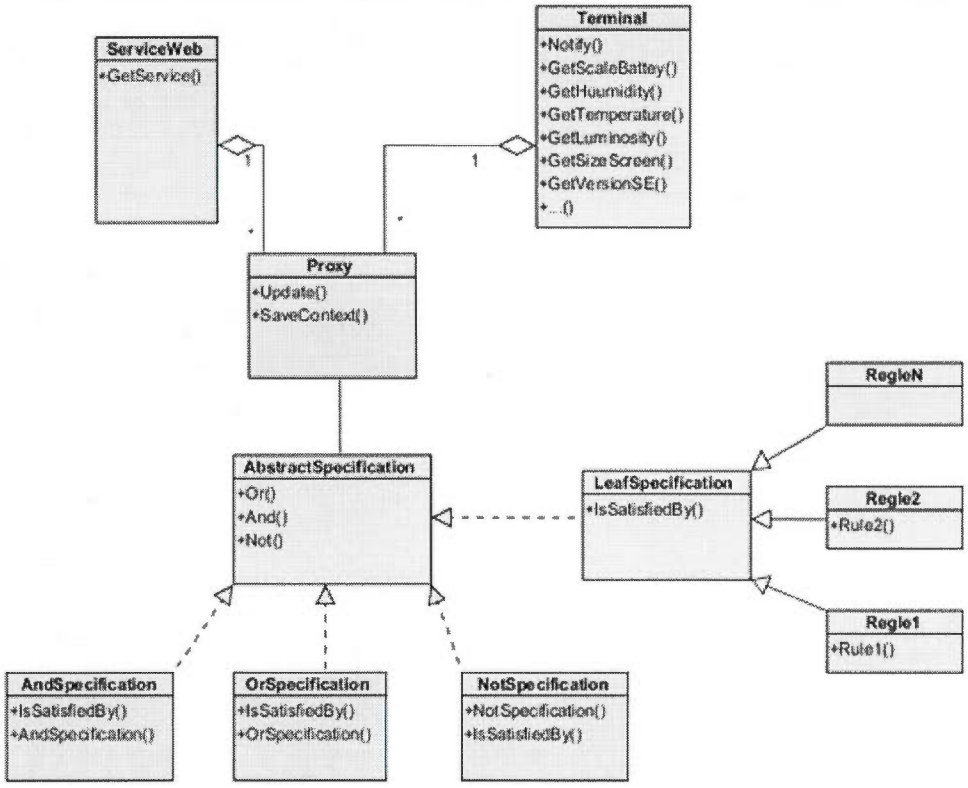

Figure 4.8 Observation du contexte et Spécification des données captées

Le *Proxy* récupère les informations une fois notifiées par l'existence d'un changement et les transmet à *AbstractSpecification*. Cette dernière entité combine moyennant les méthodes «OR, AND et NOT » entre les différentes règles déjà prédéfinies par le concepteur sous la forme "*If (condition) then (conclusion)*". Ces dernièrs expriment un ensemble de déclarations explicites pour décrire comment l'application doit comparer ses faits (données). Toutes les règles ayant une condition valide seront retenues. Un exemple de règle Température peut être :

If (TempératureUser==TempératureService) then (Retourner TempératureService)

Cela permet de retourner à la fin du processus seulement les données formant le backend de l'interface à exposer à l'utilisateur.

Notre modèle supporte jusqu'à présent l'acquisition du contexte avec le patron *Observer* et la spécification du backend de l'interface. Dans la section suivante, nous allons aborder la partie du choix de l'interface à exposer.

4.2.3 La prise de décision

Les patrons de conception *Observer* et *Spécification* sont indispensables pour le bon acheminement du processus d'adaptation. Toutefois, ils sont insuffisants pour tout satisfaire, vu qu'ils ne permettent pas de représenter une interface pour le client.

Le modèle de conception *Façade* a tendance à simplifier la complexité d'un système puisqu'il permet d'encapsuler un sous-système complexe moyennant une interface unique. Celle-ci sera le premier point d'accès pour l'utilisateur de l'application et déléguera la tâche de la proposition d'interfaces à un système de suggestion d'interfaces à base de règles.

La figure 4.9 représente la modélisation complète de notre patron de conception que nous avons intitulée « Adaptation d'interface au contexte » (Voir sa description détaillée dans l'annexe B). Il s'occupe de l'identification du contexte, du raisonnement et de la délégation de la tâche de suggestion d'interfaces de l'application à un autre module encapsulé sous la *Façade* à représenter à l'utilisateur. Cette *Façade* s'occupe, par la suite, de représenter l'interface suggérée à l'utilisateur final de l'application.

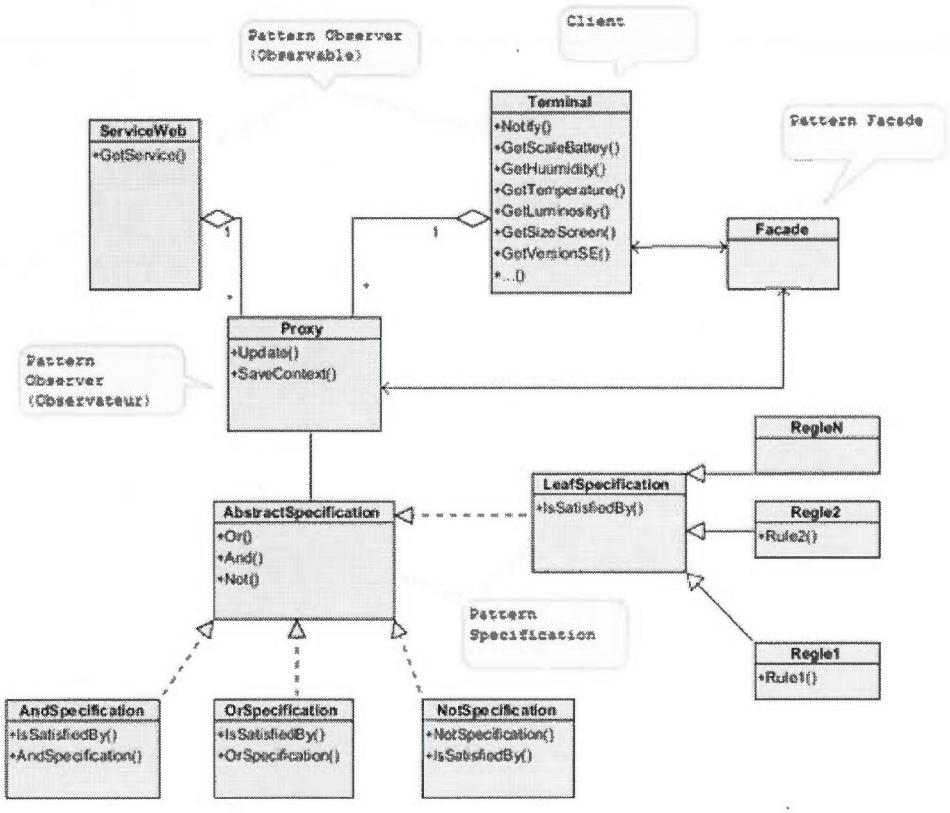

Figure 4.9 Structure de l'approche

Dans la suite de ce travail, nous allons aborder la tâche de remodelage des interfaces déléguéepar l'entité *Façade* à un sous-système de suggestion d'interfaces à base de règles afin de satisfaire l'adaptation de cette dernière à son contexte d'usage.

4.2.3.1 Architecture du système de suggestion d'interfaces à base de règles

Une interface homme-machine permet un échange continu entre l'utilisateur et le système. Bien que différents types d'interfaces existent telle que ceux des applications bureau, Web ou mobile, elles sont toujours composées d'un certain nombre de composants tels que les textbox, les gridview, etc. Ces composants doivent être liés aux données à représenter.

Une adaptation d'interface à son contexte d'usage doit être décrite par un modèle de tâches, un modèle d'utilisateur et un modèle de présentation. Par conséquent, dans notre travail, nous allons proposer une architecture à base de règles pour la suggestion d'interfaces graphiques en cours d'exécution selon un modèle descriptif de la spécification de cette interface.

Cette architecture se base sur la méthode « *Single-Conclusion Ripple-Down Rules* SCRDR (Richards, 2009) ». C'est une approche de construction progressive de connaissancesen cours d'exécution.

Pour un domaine particulier, le concepteur de l'application doit être en mesure d'identifier à l'avance certains aspects tels que : les tâches de l'utilisateur, le contexte à utiliser, les composants d'interfaces ainsi que les relations entre ces déniers et le backend. Tous ces aspects seront raffinés au cours du temps par l'expert du domaine en leur rajoutant des raffinements selon le besoin.

Le cœur de l'approche étant les règles(Yu, 2008), ces dernières sont divisées sous forme d'une condition pour spécifier si une règle doit être appliquée et une conclusion pour déterminer quelle interface doit être créée. De ce fait, la structure est la suivante :

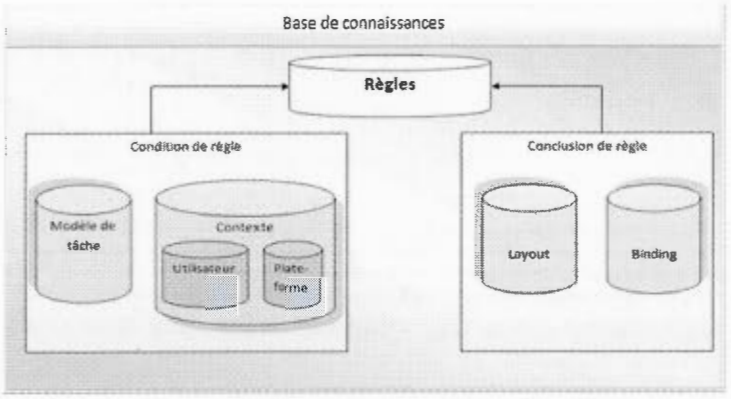

Figure 4.10 Architecture à base de règles

Conditions de règles

Comme le montre la figure 4.10, la partie gauche forme la condition de la règle.

Par *Contexte*, nous désignons tout ce qui est en relation avec l'utilisateur et la plateforme tels que : les systèmes exploitation (Android), la langue du terminal, l'orientation de l'écran et les préférences des utilisateurs (Par exemple, l'utilisateur souhaite avoir un service adapté à sa température, bruit, lumière, localisation, etc.). Ces contextes sont stockés dans une base de connaissances «BC».

En plus du contexte, la condition est formée par un modèle de tâches (Shimada, Gerofi, Horiet Ishikawa, 2013). Celui-ci est une description formelle ou semi-formelle des activités de l'utilisateur qui peuvent être identifiées par le concepteur selon l'activité de l'application. Ce modèle étant organisé d'une façon hiérarchique et stocké dans une BC, il établit les différentes interactions de l'utilisateur avec l'interface.

Un exemple d'une telle condition serait:

if(ContextPlatforme= «Android» AND
TacheUtilisateur= «afficherListeRestaurant»AND Langue= « Français » AND
PreferenceUtilisateur= « Temperature » AND PreferenceUtilisateur= « Bruit ») ...

Pour chaque condition établie, une conclusion formant la suggestion de l'interface doit avoir lieu.

Conclusion de la règle

Selon notre approche, une interface graphique devra être exprimée par un modèle qui décrit ces différents composants et qui établit le lien entre ces derniers et le backend à exposer.

Les éléments qui forment la conclusion de la règle sont d'un côté les *layouts* pour définir d'une façon hiérarchique la composition des éléments, les contraintes et les relations entre les différents composants (la préparation de layout par le concepteur doit être établie suivant une certaine description). D'un autre coté le « *binding* » pour faire le lien entre les composants de l'interface et le backend.

Dans ce qui suit, nous détaillerons ces aspects :

Le layout

Cette partie représente le cœur de l'interface. Elle permet de décrire la façon avec laquelle les composants sont arrangés sur l'écran tels que le positionnement, le style, les contraintes,la couleur, le thème, etc.

Habituellement, un layout est défini d'une façon hiérarchique, de sorte qu'un composant puisse être un layout lui-même et permet d'hériter d'autres. Dans notre approche, ces layouts sont prédéfinis par le concepteur de l'application suivant un langage bien déterminé. Par la suite, ils serontstockés dans la base de connaissances pour servir à des usages divers. La figure 4.11, présente la syntaxe du langage proposé dans (Hossein Sadatet Ghorbani, 2005) pour supporter la création des layouts.

1. <Program>→<layoutDef> {LayoutDef>}
2. <LayoutDef>→ [<Modifier>] Layout [' :' <Type>] '{' <Def> '}'
3. <Modifier>→ main
4. <Def>→{<ItemDef> | <Attribute> }
5. <ItemDef>→ item <Id> [' :' <Type>] ['{' < Attribs> '}']
6. <Type>→ Figure | <Id> | (HorizontalBox | VerticalBox | ListView) '(' <Id> ',' <Num> ')')
7. <Attribute>→<Alignment> | <Margin>
8. <Alignment>→ align (left | right | top | bottom) [':' <Id> {',' <Id> } ';']
9. <Margin>→ (marginx | marginy) '=' <Num>
10. <Attribs>→<Attrib> ';' {<Attrib> ';'}
11. <Attrib>→<Pos> | <Graphics> | <Size>
12. <Size>→ (width | height) '=' <Num>
13. <Graphics>→<Font> | <Color>
14. <Font>→ (font_face '=' <Id>) | (font_size '=' <Num>) | blod | italics | underline
15. <Color>→ (backgroun | foreground) '=' <Num>
16. <Pos>→<Abs> | <Rel>
17. <Abs>→ (x | y) '=' <Num>
18. <Rel>→ rightmost | leftmost | top | bottom | <RelPos><Id>
19. <RelPos>→ below | above | right | left

Figure 4.11 Grammaire de spécifications d'interface (Hossein Sadatet Ghorbani, 2005)

Comme nous l'avons déjà évoqué, une présentation peut être composée d'une ou de plusieurs définitions de layout. Ce dernier aura la possibilité d'en hériter d'autres.

Par ailleurs, un layout doit avoir un *modifier* (la ligne 2 de la grammaire). Ce dernier indique quelques informations sur son rôle et sur le type d'éléments qui seront réutilisés à partir d'autre layout ou élément.Par la suite, vient sa composition réelle.

Chaque élément possède un nom qui l'identifie (id), un type (*Figure*, *HorizontalBox*, *ListView*, etc.) et des attributs tels que le positionnement, la taille et les caractéristiques des éléments.

Tout élément de l'interface doit avoir un alignement et une marge par rapport à un autre composant. L'alignement peut être soit gauche,droit, haut ou bas. L'espace entre les éléments est défini par une distance x et y. Un élément de l'interface peut avoir comme attribut la taille qui est définie par une largeur et une hauteur, la police de caractères, la position, etc.

La position est un aspect important qui a été traité dans la description de l'interface. Il permet d'identifier la position de chaque item par rapport à l'autre. Cette idée résoud la contrainte abstraite qui décrit à haut niveau la description de la relation entre les éléments de l'interface dans le layout. Par exemple, les contraintes spatiales, représentent la relation directe exprimant la structure géométrique de la présentation tel que l'élément X est relatif à Y(Loket Feiner, 2001).

La grammaire offre différents types de relations comme *absolute* pour déterminer la succession des éléments et *relative* pour spécifier la position géographique d'un élément par rapport à un autre.

Afin de garantir un meilleur positionnement des composants sur l'écran du mobile, le patron de conception *Iterator* s'occupera des modifications sur les layouts définis par l'intermédiaire de simples instructions.

La figure 4.12 propose un exemple d'interface pour une simple description de layout. Sur cet exemple, le layout principal *Exemple* est aligné en haut avec une marge de 0. Ce dernier est composé de deux parties : un *header* situé au top de layout avec une largeur égale à la taille

de l'écran et une hauteur égale à la taille de l'écran/3, et un *body* qui se situe en bas du header.

```
Main Layout Exemple{
        Item header{
                top=0;
                widh="largeurscreen";
                Hheight="hauteurscreen/3"
}
Item body{
        below header;
        widh="largeurscreen";
        align= top
        margin=0
}
```

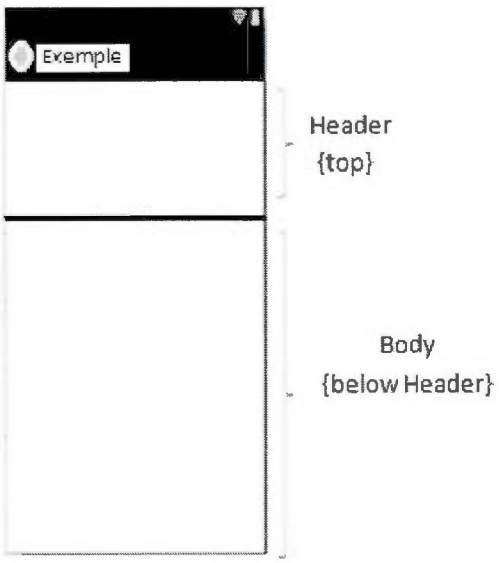

Figure 4.12 Exemple d'interface créée en fonction de la description de Layout

Une fois les layouts définis et avant de passer au stockage dans la base de connaissances, le concepteur doit compiler les layouts générés pour s'assurer qu'ils sont syntaxiquement et lexicalement corrects. Plusieurs compilateurs offrent cette possibilité parmi lesquels nous pouvons citer JavaCC(Sun Microsystems, 2006). Une fois validé, il génère un arbre de syntaxeabstraite commele montre la figureA.8 (de l'annexe A), relative au layout de la figure 4.12.

Binding

Ce composant de l'architecture permet d'établir le lien entre le backend de l'interface déjà identifiée par le patron de conception et le composant de l'interface graphique. Dans le cas d'un changement du backend, et non du layout formant l'interface, le mécanisme de binding indique donc quel backend sera associé à quel composant.

Les règles

Nous avons évoqué dans les deux parties précédentes deux aspects importants pour la création d'interfaces en cours d'exécution. La première étant les conditions et la deuxième concerne la conclusion formée de *layouts* et de *binding*.

Un autre point essentiel dans notre architecture est les règles. En résumé, lorsqu'un expert du domaine doit expliquer comment il est arrivé à sa conclusion, il fournit une justification que celle-ci est correcte. Implicitement, il prouve que sa conclusion est préférable par rapport à d'autres pour une condition précise(Compton, Peters, Edwards et Lavers, 2006).

Nous proposons ci-dessous des règles composées du contexte comme condition et d'une suggestion d'interface comme conclusion.

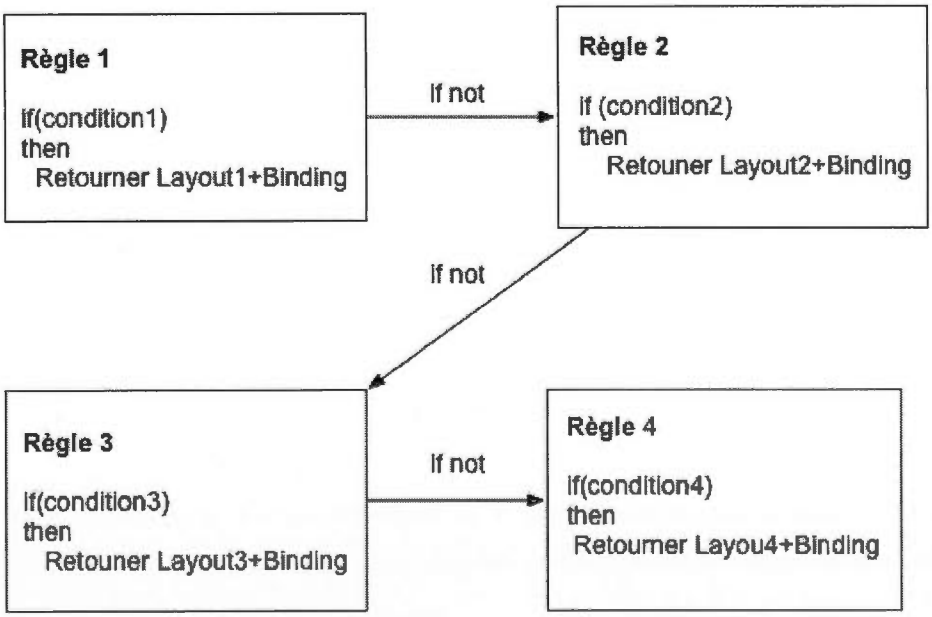

Figure 4.13 Exemple de règles

Par ailleurs, le concepteur définit quelques règles à l'avance. Ces dernières permettront de déterminer quel composant d'interface doit être affiché à l'écran pour une condition donnée.

Ces règles, étant stockées dans une base de connaissances d'une façon hiérarchique, la méthode *SCRDR* utilisée permet de les parcourir et de retourner la conclusion qui satisfait sa condition. Cette méthode offre la possibilité de l'ajout de connaissance en cours d'exécution. Par ailleurs, lorsqu'une règle fournit par le système de base de connaissances est inccorcte, une nouvelle règle de raffininement sera ajoutée et sera reliée à la mauvaise règle de sorte que cette dernière ne sera plus évaluée dans le même contexte. La conclusion de la règle de raffinement est utilisée plutôt que la conclusion de la règle parent.

Notre système permet de prendre la décision de l'interface à présenter en faisant le lien entre la condition (i.e. Contexte) et la conclusion (i.e. Comoposant d'interface) à travers les règles à conclusion unique.

Une fois la suggestion est faite, le patron *Façade* s'occupe de représenter le service sous forme d'interface. Des outils techniques permettront d'atteindre ces fins.

Cette approche étant adaptative, nous exposerons brièvement les deux types d'adaptation*OnAction* et *OnChange*. Ils représentent les principales causes pour lesquelles une adaptation d'interface doit avoir lieu.

4.2.4 La gestion de l'adaptation

L'approche à base de modèle pour l'adaptation des interfaces graphiques des applications mobiles en fonction du contexte (présentée dans la section précédente sur la figure 4.11) réalise une adaptation transparente aux clients finaux. Elle délègue la tâche de suggestion d'interfaces à une architecture basée sur les règles.

Toutefois, selon (Boinot, Marlet, Noye, Mulleret Consel, 2000), nous pouvons distinguerdeux types d'adaptation soit*OnChange*et *OnAction*. Le premier exige un nouveau comportement à l'instant marqué par un changement dans l'environnement, alors que l'adaptation OnActionexige la même chose, mais suite à une action exécutée par le client.

Nous allons détailler le comportement de notre système afin de connaitre la façon avec laquelle il réagit face à ces deux types d'adaptation.

4.2.4.1 Adaptation OnAction

Afin d'aborder cet aspect, nous allons présenter la séquence d'interaction entre les différents acteurs formant notre système d'adaptation d'interfaces. Ces échanges seront schématisés sur la figure 4.14:

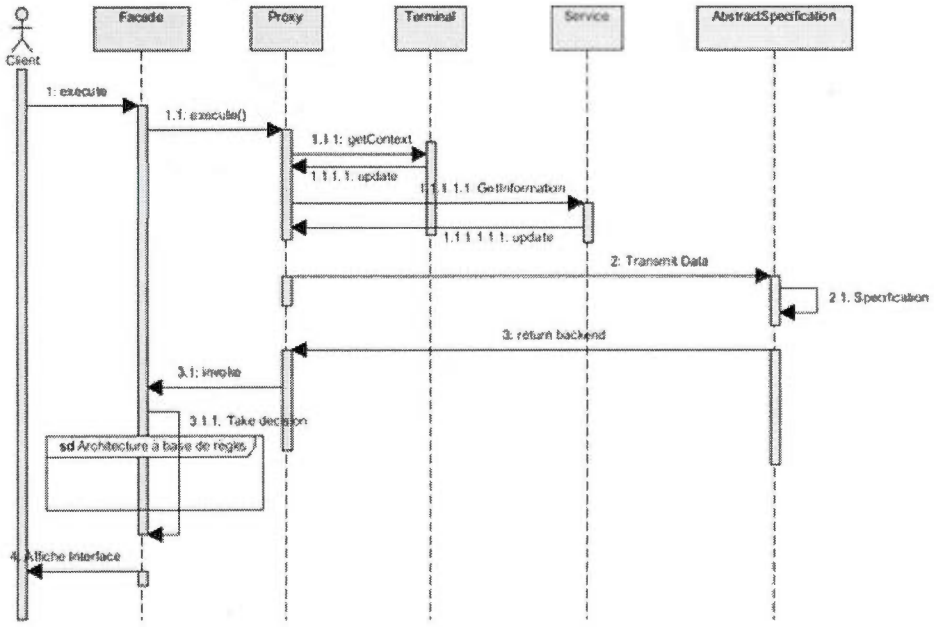

Figure 4.14 Séquence d'interaction des entités à l'adaptation OnAction

Au cours du processus d'adaptation *OnAction*, le client commence par exécuter une opération en invoquant l'unique façade de son service représenté par le composant *Façade* du modèle. Comme exemple de cette opération, nous pouvons citer l'ouverture de l'application mobile. La façade envoit une requête au composant *Observateur* du modèle *Observer* lui demandant d'aller chercher les informations contextuelles, à la fois liées àla plateformede l'utilisateur et au service, à partir du sujet modélisérespectivement par les composants *Terminal* et *Service*. Ces derniers répondent à la requête du Proxy qui,à son tour, les transmet à *AbstarctSpecification* du modèle*Specification*. À ce niveau, le contenu de l'interface sera

déterminé. Une fois cette tâche de raisonnement sur le contexte achevée, *AbstractSpécification* retourne le *backend* de l'interface au proxy qui invoque à son tour la façade. Cette entité délègue la tâche de suggestion d'interface à l'approche à base de règles(section 4.2.3.1) puis se charge de sa représentation sur le terminal.

4.2.4.2 Adaptation Onchange

Le processus d'adpatation *OnChange* s'applique une fois que le traitement de OnAction ait été exécuté au moins une fois. Cela implique que l'application est déjà en cours d'exécution et peut subir des changements. La figure 4.15 illustre la séquence d'interactions entre les différents acteurs formant notre système d'adaptation d'interface. Cette tâche est exécutée suite à la détection d'un changement dans l'environnement qui entoure le système.

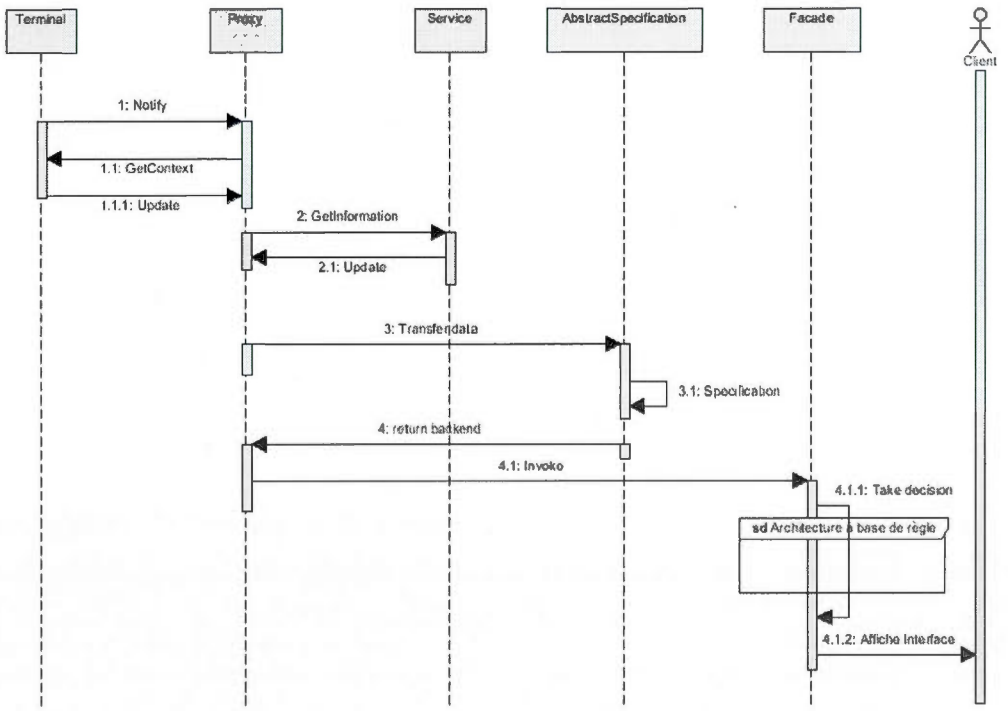

Figure 4.15 Séquence d'interaction des entités à l'adaptation OnChange

Le sujet du patron *Observateur/Observable*, modélisé par le composant *terminal* notifie le *proxy*, par une requête, que le contexte capté a subi un changement. Le *proxy* qui s'occupe du comportement après l'observation requiert le nouveau contexte de l'utilisateur du Terminal et s'assure que les données du service n'ont pas été changées. Dans le cas contraire, il demande les nouvelles informations du service. Pour le reste du traitement, il est semblable à celui décrit dans le processus d'adaptation OnAction.

Notre approche basée d'un côté sur les patrons de conception *Observer*, *Spécification* et *Façade* et d'un autre coté sur une architecture de base règles, forme un mécanisme puissant. L'objectif est de garantir une adaptation dynamique des interfaces graphiques des applications mobiles dépendant de tout changement contextuel pouvant affecter le service, l'utilisateur ou la plateforme.

D'une part, le modèle de conception proposé offre un moyen structuré etextensible afin de supporter l'adaptation *OnAction* et *onChange*. La bonne structuration de notre modèle sépare les différents processus nécessaires à l'adaptation : celui de l'observation, du raisonnement et de la prise de décision de l'interface à exposer.Celle-ci propose une meilleure organisation des composants d'interface. Elle est suggérée par une architecture à base de règles et exposée à l'utilisateur final par l'entité Façade du patron.

Afin de valider notre approche, nous allons proposer dans le chapitre suivant, un prototype d'application mobile.

CHAPITRE V

RÉALISATION

L'adaptation de l'interface des applications mobiles au contexte d'utilisation requiert une bonne démarche pour l'acquisition du contexte, le raisonnement, la prise de décision concernant l'interface à représenter et la gestion de l'adaptation à chaque changement de contexte. L'architecture proposée, étant basée sur les patrons de conception et les règles de logique, couvre ces différents besoins fonctionnels.

Dans ce chapitre, une mise en œuvre d'une étude de cas tirée de notre vie a été réalisée.Cet exemple constitue une preuve de concept de l'architecture proposée pour l'adaptation des interfaces des applications.

5.1 Étude de cas

Nous allons considérer l'étude de cas suivante pour illustrer notre approche :

Un utilisateur décide de choisir un restaurant en adaptant sa situation contextuelle aux conditions environnementales de l'endroit (restaurant). L'interface de l'application doit s'adapter en fonction de la tâche de l'utilisateur, ses préférences et les capacités de son téléphone pour présenter la liste des restaurants. L'utilisateur choisit en premier ses préférences. Ces dernières représentent les informations contextuelles à la fois du client et du restaurant comme : la température, le bruit, l'humidité, la luminosité, la pression atmosphérique, la localisation, etc. Par la suite, il envoie sa requête. À cet instant, l'application doit adapter le contenu et l'interface en fonction des préférences choisies et des ressources du terminal telles que la taille de l'écran, la langue,l'orientation du texte, etc. L'application doit ainsi continuer à gérer l'adaptation pour faire face au changement du contexte d'utilisation.

5.2 Technologie utilisée

Notre prototype a été produit en Java sur la plateforme *Android*. L'appareil mobile utilisé pour les simulations est le « *Samsung-S4* ». Ce dernier, avec son système d'exploitation, offre d'innombrables moyens pour le devleppoment de notre application.

Nous avons opté pour une architecture orientée service « *REST* ». Elle permet de servir un fichier JSON comportant les informations contextuelles de chaque restaurant ainsi que les informations qui lui correspondent. Ces dernières forment le backend de l'interface.

5.3 Mise en oeuvre

Notre exemple, malgré sa simplicité, offre diverses propriétés importantes pour satisfaire une adaptation d'interface au contexte, notamment la mobilité des utilisateurs, la variabilité des préstataires du service sur le web, etc. À cet effet, nous avons adapté notre architecture pour avoir la forme donnée dans la figure 5.16:

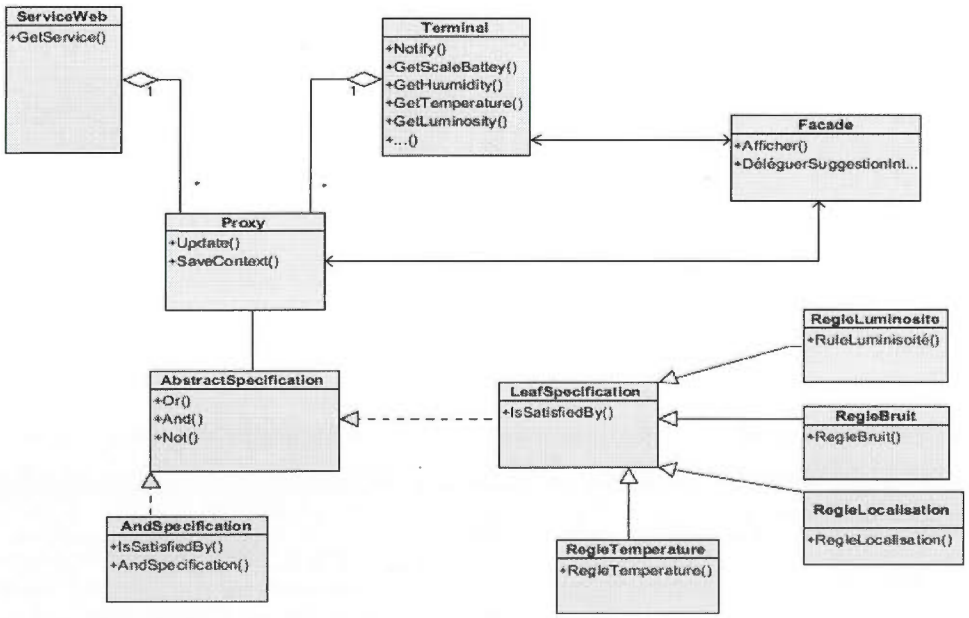

Figure 5.1 Modèle de conception adapté au cas d'étude

5.3.1 Acquisition du contexte du terminal

Le patron de conception Observer permet de détecter tout changement lié à l'environnement ou à la plateforme de l'utilisateur à travers les capteurs inclus dans l'appareil. Ainsi le mobile *Samsung S4* utilisé lors du développement de notre prototype est équipé de plusieurs capteurs intégrés. Le tableau ci-dessous couvre seulement notre besoin :

Tableau 5.1 Liste des capteurs utilisés

Capteur	Rôle
Luminosité	Utilisé pour indiquer le niveau de luminosité de l'endroit
Baromètre	Utilisé pour déterminer la pression atmosphérique
Température	Utilisé pour indiquer la température ambiante
humidité	Utilisé pour déterminer le niveau de l'humidité
GPS	Utilisé pour déterminer la position géographique
Bruit	Utilisé pour déterminer le niveau de bruit de l'endroit
Orientation	Utilisé pour détecter l'orientation de l'écran : paysage ou portrait
Langue	Utilisé pour déterminer la langue utilisée par l'utilisateur

Dans notre cas d'étude, nous avons limité les préférences de l'utilisateur aux variables suivantes : la température, le bruit, la position géographique et l'humidité. Par conséquent, le client aura un service adapté à ces préférences.

5.3.2 Acquisition du contexte du service Web

Le patron *Observer* s'occupe de la détection du changement lié au *backend* du service. Ce dernier présenté sous un fichier Json, comporte les informations contextuelles de chaque restaurant à un moment précis telles que: sa température ambiante, sa position géographique, son humidité, le niveau de son bruit, sa luminosité et son niveau de pression. Il comporte également des données associées à chaque restaurant telles que le nom, l'adresse, la fourchette du prix, etc.

L'extrait suivant du fichier Json, utilisé lors de notre mise en œuvre, présente ces différentes rubriques :


```
{
    "Restaurant": [
    {
      "id": "1",
      "Nom": "L'ASSOMMOIR",
                                    "Image":
"http://lh6.ggpht.com/_ZN5zQnkI67I/TCFFZaJHDnI/AAAAAAAABVk/YoUbDQHJRdo/s144-c/P9250508.JPG",
          "Adresse": "112, avenue Bernard Ouest Mile End, Montréal",
    "Coordonnes": "514 272-0777",
    "Description":"Vous y découvrirez un menu unique et audacieux où le travail et la passion sont palpables. Nos barmans experts André, Jimmy, Marjo, Manu et cie sauront vous proposer le cocktail parfait pour accompagner votre soirée, alors qu'en cuisine, le chef Pascal et son équipe vous feront vivre une expérience haute en délices! Pour vous, ils coupent, cuisent, flambent et servent ces spécialités que sont nos fameux tartares, cévichés, tapas et plateaux à partager. Chaque mois, nous vous proposons en plus de la carte, une ardoise de saison fraîchement concoctée afin de charmer vos papilles.",

    "Site":"http://www.assommoir.com/#!assond/cec3",
    "Specialite": "Amérique du Nord",
    "HeureOuverture": "Heures d'ouverture Lundi : 11:00 à 3:00Mardi : 11:00 à 3:00Mercredi : 11:00 à 3:00Jeudi : 11:00
à 3:00",
    "TypeDePaiement": "Visa, Master Card, American Express",
    "PrixMoyen": "20$-40$",
    "Temperature": "20",
    "Pression": "15",
    "Lumiere": "4",
    "Latitude": "45.525908",
    "Longitude": "-73.603332",
    "Humidity": "50",
    "Noise": "50"
```


Figure 5.2 Extrait du fichier Json

5.3.3 Raisonnement

Lors de la détection des changements contextuels, le *proxy* du patron *Observer* est notifié de cette variation. À cet instant, le *proxy* récupéreses informations et les transmet au patron

spécification qui s'occupe de former le *backend* de l'interface à représenter. Cette étape est effectué moyennant les règles à définir par le concepteur.

Si le client choisit d'avoir la liste des restaurants ayant une température, un niveau de bruit et une localisation proche à son état actuel, le système comparera les informations contextuelles de l'utilisateur avec cellesdu service à travers les règles suivantes : température, bruit et localisation.

À la fin de ce processus, un ensemble de données formant le service à représenter est prêt à être servi. L'étape manquante est la prise de décision de l'interface à exposer en respectant les ressources du mobile utilisé.

5.3.4 Prise de décision

Le patron façade de notre modèle s'occupe de la représentation de l'interface à l'utilisateur puisqu'il représente le seul point d'accès à l'application. Possédant déjà les informations à représenter et n'ayant aucune idée sur l'interface à déployer, il délègue la tâche de suggestion d'interface à son sous-module d'acquisition de connaissances« *Single-Conclusion Ripple-Down Rules* » (*SCRDR).*

Le système expert *SCRDR* est construit au moment où l'application est en cours d'exécution. Il débute par un système de base de connaissances vide qui sera construit au fil du temps selon l'évolution du contexte d'usage. L'expert du domaine ajoute donc des règles, des conditions et des conclusions pour satisfaire la suggestion d'interface à sa condition d'usage.

La condition

Pour cette partie de la règle, nous avons commencé par créer trois modèles spécifiques à notre exemple : Le premier comprend les tâches des utilisateurs telles que : ouvrir application et régler préférence.

```
<taskModel><!--Définition des tâches du système-->
<task id= "task1" name= "Ouvrir application">
<task id= "task1.1" name= "Regler préférence">
        <task id= "task1.1.1" name= "Set préférence température">
        <task id= "task1.1.2" name= "Set préférence bruit">
        <task id= "task1.1.3" name= "Set préférence humidité ">
        <task id= "task1.1.4" name= "Set préférence localisation">
        <task id= "task1.1.5" name= "Set préférence lumière">
</task>
<task id= "task1.2" name= "Afficher restaurant">
</task>
</taskmodel>
```

Le deuxième est spécifique aux préférence de l'utilisateur, soit la température, bruit, humidité, localisation où lumière. L'interface finale sera exposée en fonction de ces préférences.

```
<contextPréférenceModel><!--Définition des préférences prévus-->

        <context id= "température"></context>
        <context id= "bruit"></context>
        <context id= "humidité"></context>
        <context id= "localisation"></context>
        <context id= "lumière"></context>
</ contextPréférenceModel>
```

Le dernier modèle est en relation avec la plateforme. Il indique quel mobile est utilisé par client. Par conséquent, la taille de l'écran sera facilement détéctée et l'interface sera spécifique au système d'exploitation du téléphone.

```
<contextPlateformeModel><!--Définition des préférences prévus-->
        <context id= "Mobile" name= "SamsungS4"></context>
        <context id= "SE" name= "Android "></context>
        <context id= "Langue">
                <context id "Langue1" name = "Francais">
                <context id "Langue2" name = "Anglais">
```

```
</context>
<context id= "OrintationText">
        <context id" OrintationText 1" name = "Paysage">
        <context id " OrintationText 2" name = "Portrait">
</context></ contextPlateformeModel>
```

Ces trois éléments sont stockés dans une base de connaissances et ils seront enrichis suivant notre besoin afin de satisfaire la création de l'interface dépendament de la variation du contexte.

La conclusion

Cette partie spécifie une description de layout formant l'interface à représenter. À titre d'exemple, nous avons créé un squelette d'interface sur la figure 4.18 spécifique à la description suivante de layout :

```
Main layout afficheList {
  Item header : ImageContainer { top;}
  Item body : listviewContainer {below header}n
  Align top;
  Marginx = 0;
}
LayoutimageContainer : {
  Item imageView{width = "largeurscreen";height = "hauteurscreen/4";}
}
LayoutlistviewContainer : VerticalBox (itemdescription , 1) {Marginx = 5;}
Layoutitemdescription : HorizontalBox( newItem , "nombredebackend"){
Marginx = 0;
}
LayoutnewItem {
        Item Imagedescription { Width = "largeurscreen/3";heigth = 50 ; Left; }
        Item Description { Backgroung = x00ffffff; font-size=12; font-face=TimeNewRoman ;
rightmost ImageDescription ;}
}
```

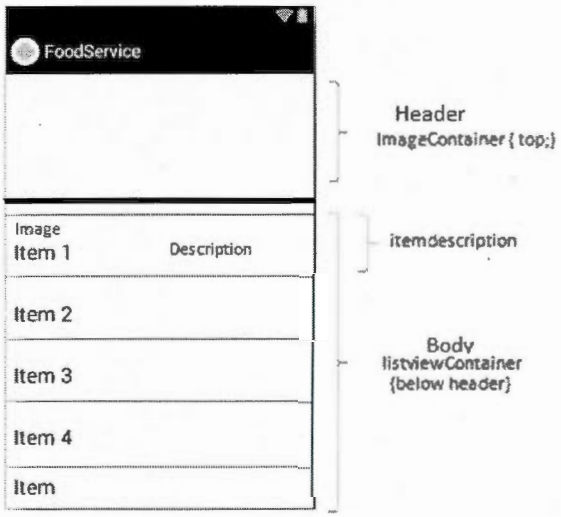

Figure 5.3 Exemple d'interface définie en fonction de la définition de Layout

Le layout principal «afficheList » contient une en-tête (*header)* en haut et un corps (*body)* qui est positionné juste en dessous du *header.* Ce dernier est défini par un layout *imageContainer* qui comprend un élément du type *imageView* dont la largeur est égale à la largeur de l'écran et sa hauteur est égale à la hauteur de l'écran divisé par 4. Le *body* contient un layout*listviewContainer* qui se situe à une marge de 5mm du *header.* Ce *listView* est du type *Verticalbox* qui, de son côté, est formé d'un layoutqui contient des layouts *newItem.* Ces éléments sont organisés horizontalement sur l'écran et ils sont divisés en deux parties : une *imagedescripton* située sur la gauche avec une largeur égale à la taille de l'écran divisé par 3 et une hauteur de 50mm, alors que la deuxième partie est une description.

Les règles

Comme notre système d'acquisition de connaissances évolue progressivement, la conclusion peut avoir trois formes : correcte, incorrecte ou manquante. Dans le cas où cette conclusion est incorrecte ou manquante, l'expert ajoute la condition, la conclusion et la règle qui établissent le lien entre ces deux dérniers.

Nous proposons par la suite, sur la figure ci-dessous, un ensemble de règles pour notre contexte utilisé:

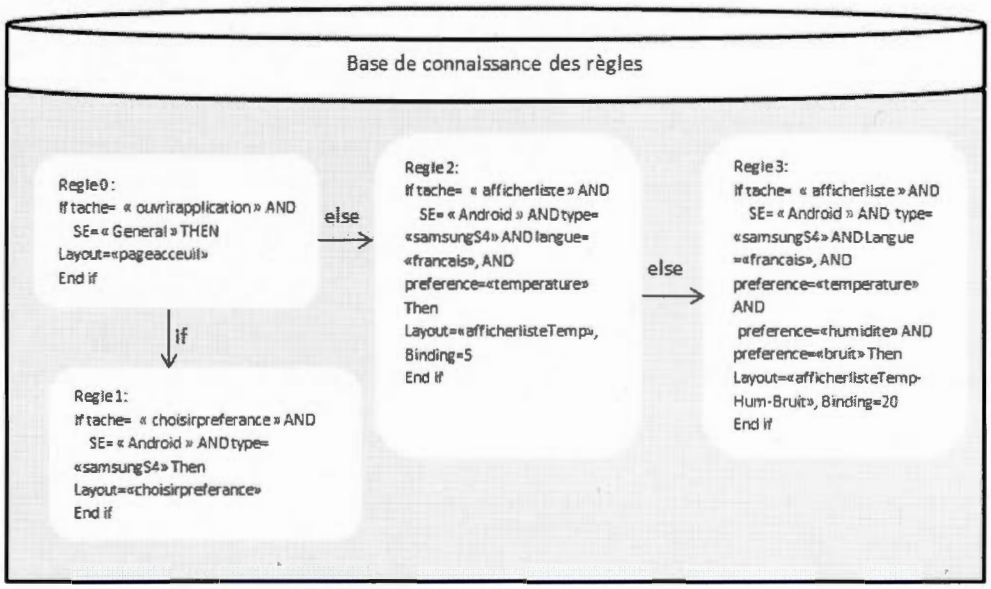

Figure 5.4 Exemple de règles

Selon cette base de connaissances, nous avons prévu des interfaces pour chaque condition.

La règle 0 suggère une interface de page d'accueil pour un contexte formé uniquement de la tâche de l'utilisateur *ouvrirApplication*. Par la suite, pour chaque tâche de l'utilisateur, le système commence à vérifier les règles. Si l'utilisateur veut choisir ses préférences et selon son type de mobile (ex. SamsungS4 avec un système d'exploitation *Android*), le système trouve la règle qui correspond à son besoin, soit la Règle1. Sinon comme deuxième scénario, si la tâche correspond à *afficherListe*, (le système d'exploitation est *Android*, le type de mobile est *SamsungS4*, la langue utilisée est *Français* et l'utilisateur a déjà choisi *Température* comme préférence), le système suggère l'interface basée sur *afficherlistTemp* comme layout avec le binding = 5 pour lier le backend à l'interface.

Notre système permet de prendre la décision de l'interface à présenter en faisant le lien entre la condition (c-à-d Contexte) et la conclusion (c-à-d Composant d'interface) à travers les règles à conclusion unique.

Afin de traduire la description de l'interface (*layout*) à une syntaxe supportée par le système d'exploitation *Android et obtenir* une interface graphique liée à nos conditions d'utilisation, nous envisageons d'intégrer un module de transformation. Ce dernier, une fois implémenté, devra aboutir à l'imprime-écran ci-dessous représentant l'interface que notre approche devrait satisfaire. Cet exemple présente une liste de réstaurants adaptés aux préférences du client (localisation, humidité, température et lumière).

Figure 5.5 Imprime-écrand'interface

La simulation précédente démontre l'efficacité de notre modèle à base de règles proposé pour la suggestion d'interface selon le contexte d'utilisation. Les autres conditions,

conclusions et règles doivent être rajoutées au fur à mesure de l'identification d'un nouveau contexte. Ainsi, le modèle de conception que nous avons proposé se sert de cette suggestion pour la traduire en interface concrète et s'occupe de son adaptation suite à l'interaction du client avec le système ou lors d'un changement dans le contexte.

CHAPITRE VI

CONCLUSION

Le domaine de la plasticité des interfaces a récemment attiré l'attention d'ungrand nombre de chercheurs.

Malgré l'existence de diverses approches relatives à l'adaptation d'interfaces au contexte d'usage, notamment pour les applications mobiles, elles présentent plusieurs limites. Partant de ce constat, nous nous sommes basés sur ces insuffisances pour en faire notre objet de recherche.

Nous avons centré nos efforts sur les problématiques de ces systèmes telles que la prise en compte d'une gamme limitée du contexte, le manque d'un processus d'adaptation et l'absence d'un formalisme descriptif des composants de l'interface.

Des tels types d'adaptation nécessitent (a) l'identification de tout contexte en relation avec l'utilisateur et b) l'élaboration d'un modèle supportant l'adaptation de l'interface. Pour atteindre notre objectif d'adaptation d'interface au trois contextes (qui sont: utilisateur, plateforme, environnement) , nous avons proposé un modèle de conception supportant à la fois l'acquisition du contexte à partir de plusieurs sources, la spécification du backend du service, l'adaptation aux évènemenets *OnAction* et *OnChange* ainsi que la représentation de l'interface.

Cette interface est suggérée par un système expert d'acquisition de connaissances à base de règles. Ce dernier est formé de trois parties.Premièrement, une condition composée d'un modèle de tâches et du contexte de l'utilisateur ainsi que de la plateforme. Deuxièment, une conclusion qui décrit les composants d'interfaces selon un modèle de présentation et un *binding* pour lier les composants et le *backend* de l'interface. Finalement, les règles

pourétablirle lien entrela conclusion etla condition unique. Cette règle permet donc de retourner la suggestion finale de l'interface.

Par ailleurs, les objectifs visés dans le cadre de notre travail de recherche ont été réalisés.

Cependant, pour mettre en œuvre notre approche, nous avons développé notre modèle de conception en l'appliquant à un cas d'étude tiré de la vie réelle. Nous avons également identifié les différents éléments formant notre système expert à base de règles. C'est ce dernier que nous nous proposons d'enrichir afin de couvrir une large gamme de contextes d'utilisation.

Bien que notre approche prenne en considération une large gamme de contextes au moment de l'adaptation de l'interface, elle ne manque pas de limites. Parmi ces dérnières, nous pouvons mentionner le manque d'un moyen capable de traduire la description d'interface en code concrêt et le non support de plusieurs plateformes de developpement simultanément.

Afin de contourner ces limites, nous envisageons, dans le futur, de proposer un module de transformation. Ce module consistera àtraduire la description de la représentation suggérée par l'architecture à base de règlesà une interface concrête multiplateforme.

Pour finir, à travers ce travail, nous espérons avoir des systèmes interactifs adaptables aux différents contextesen conservant la réutilisablitée de notre approche dans le cadre des problématiques futures.

APPENDICE A

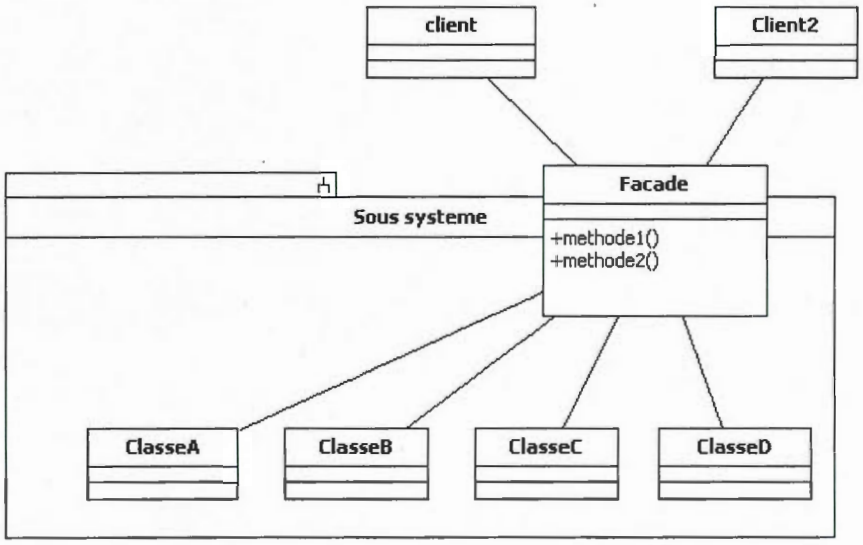

Figure A.1 Structure du patron de conception Façade(Gamma, Helm, Johnsonet Vlissides, 1994)

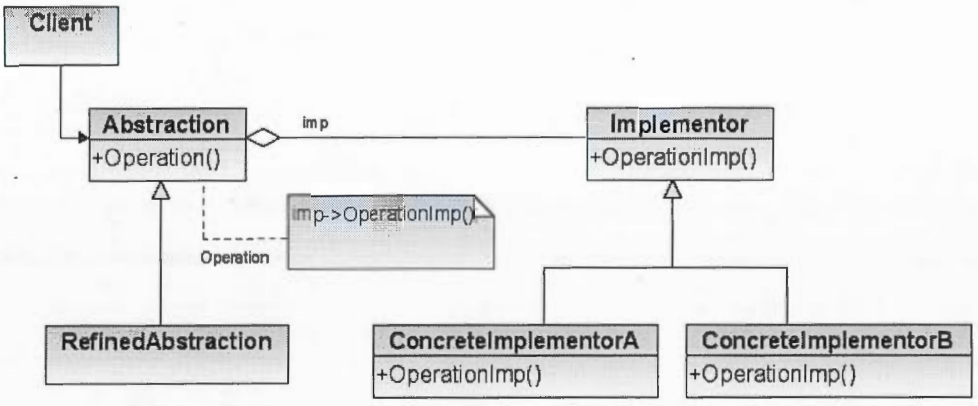

Figure A.2 Structure du patron de conception Bidge(Gamma *et al.*, 1994)

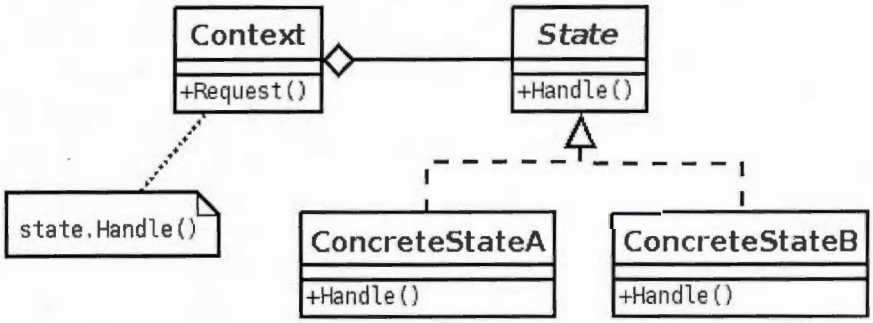

Figure A.3 Structure du patron de conception State(Gamma *et al.*, 1994)

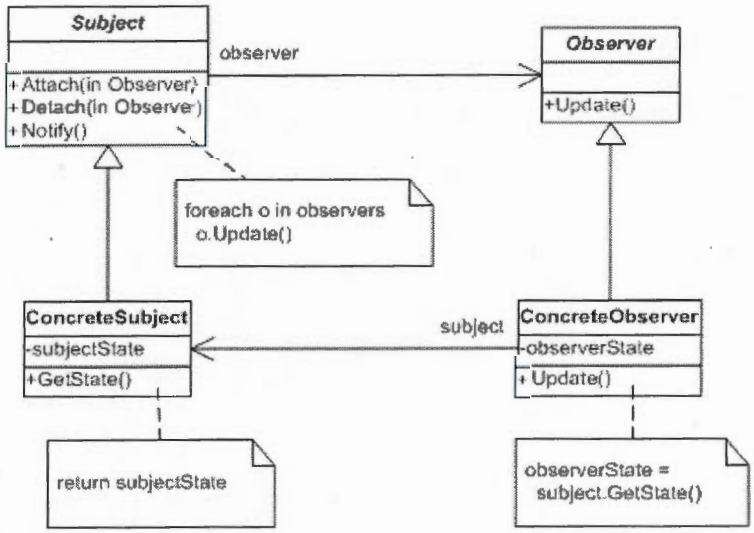

Figure A.4 Structure du patron Observer(Gamma *et al.*, 1994)

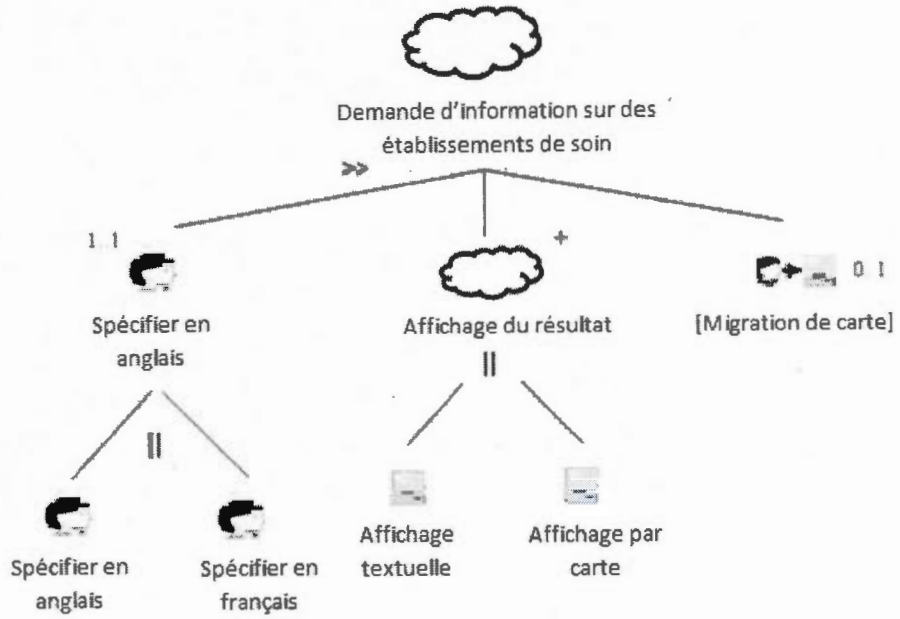

Figure A.5 Modèle de tâche adaptable (Hachaniet Dupuy-Chessa, 2009)

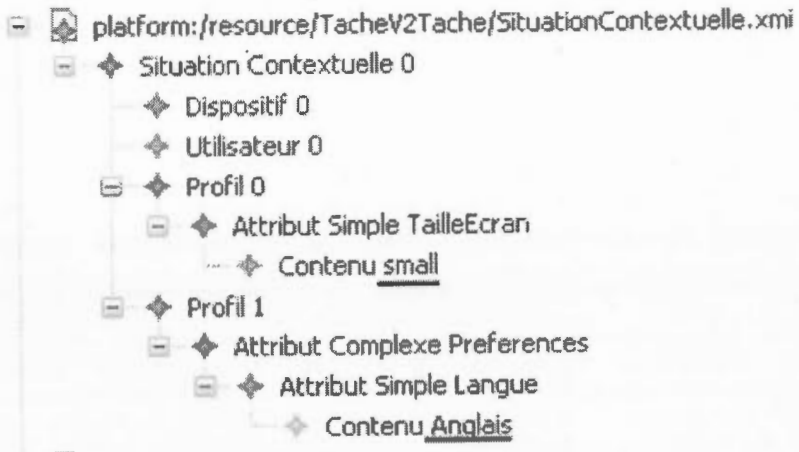

Figure A.6 Modèle de contexte (Hachaniet Dupuy-Chessa, 2009)

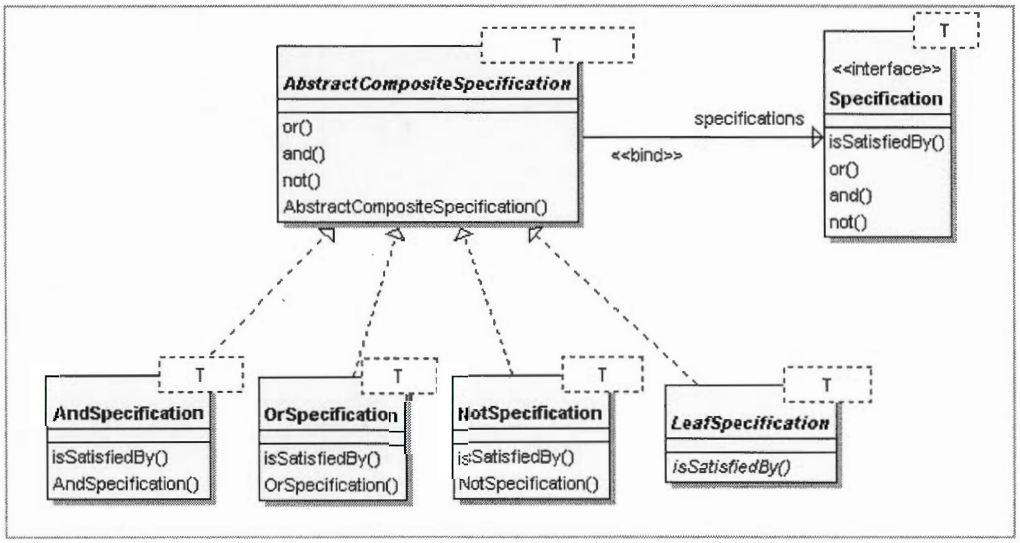

Figure A.7 Structure du patron Spécification(Eric Evans, 1999)

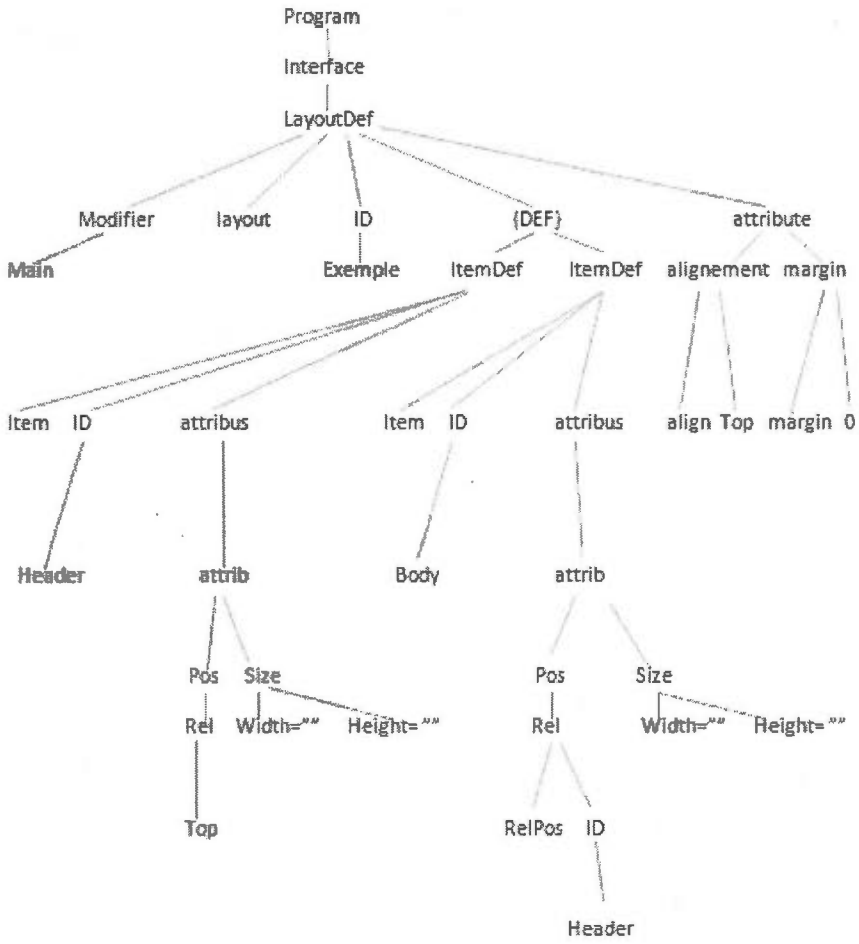

Figure A.8 Syntaxe d'arbre abstraite pour la description de la figure 4.12

APPENDICE B

Patron de conception Adaptation d'Interface AU contexte

B.1 Nom

Nous allons présenter ici le modèle de conception « Adaptation d'interface au contexte » suivant le formalisme de *GOF*annoncé à la section 2.2.

B.2 Intention

Le patron de conception « Adaptation d'interface au contexte » permet à un système sensible au contexte d'être autoadaptatif. Ainsi, il supporte l'adaptation d'interface tout en n'en exposant qu'une seule à l'utilisateur en fonction des ces changements contextuels « utilisateur, plateforme, environnement ». Ce modèle acquiert les informations du service à exposer, forme le backend, et dèlègue la tâche de suggestion d'interface à un système expert à base de règles encapsulé sous la façade. C'est cette dernière qui s'occupe de la repésentation.

B.3 Motivation

La variation dans l'environnement d'exécution, qui caractérise les terminaux mobiles ainsi que leurs ressources limitées,représentent un terrain favorable pour la mise en œuvre de notre patron. Cederniera pour objectif de garantir une autoadaptation en offrant une interface unique à l'utilisateur en fonction de ses préférences, des capaçités du dispositif qu'il utilise tout en étant adapté à sa situation environnementale instantanée. Par conséquent, le passage d'un contexte C1 à un contexte C2 déclenche une action et entraîne une réaction qui a un impact sur les interfaces (la redistribution de l'interface).

Nous pouvons considérer l'exemple suivant comme illustration de notre modèle :

Un utilisateur décide de choisir un restaurant en adaptant sa situation contextuelle aux conditions environnementales de l'endroit (restaurant). L'interface de l'application doit s'adapter en fonction de la tâche de l'utilisateur, ses préférences et les capacités de son mobileàprésenterla liste des restaurants. L'utilisateur choisit en premier ses préférences. Ces dernières représentent les informations contextuelles à la fois du client et du restaurant. Il peut s'agir de la température, du bruit, de l'humidité, de la luminosité, de la pression atmosphérique, de la localisation, etc. Par la suite, il envoie sa requête. À cet instant, l'application doit adapter le contenu et l'interface en fonction des préférences choisies et des ressources du terminal telles que la taille de l'écran, la langue, l'orientation du texte, etc. L'application doit ainsi continuer à gérer l'adaptation pour faire face au changement du contexte d'utilisation.

Une fois le domaine de l'application défini par le concepteur, notre patron de conception va gérer *l'acquisition du contexte*. Cette étape peut être faite par un mécanisme supportant l'observation du changement et la notification.

Partant du contexte recueilli, le modèle doit *raisonner* sur ces informations (c-à-d. comparer les données contextuelles du service Web avec ceux de l'utilisateur) afin de former le backend de l'interface.

Ensuite, pour *prendre la décision*de l'interface à exposer, le patron de conception délègue la tâche de la suggestion d'interface à un système à base de règles. Cette interface sera créée, suivant un modèle qui spécifie la description des composants de l'interface. Une fois créé, le patron est alors prêt à afficher l'interface à l'utilisateur et *gérer l'adaptation* suivant les nouveaux changements contextuels.

B.4 Indication d'utilisation

Le patron de conception « Adaptation d'interface au contexte » peut être exploité lorsque :

- Le système désire exposer une seule interface;
- Le système doit acquérir diverse informations contextuelles;

- Le système doit raisonner par la logique pour former le *backend* d'interface;
- Le système doit gérer un sous-système complexe qui représente une seule interface à l'utilisateur;
- Le système doit être auto adaptatif.

B.5 Structure

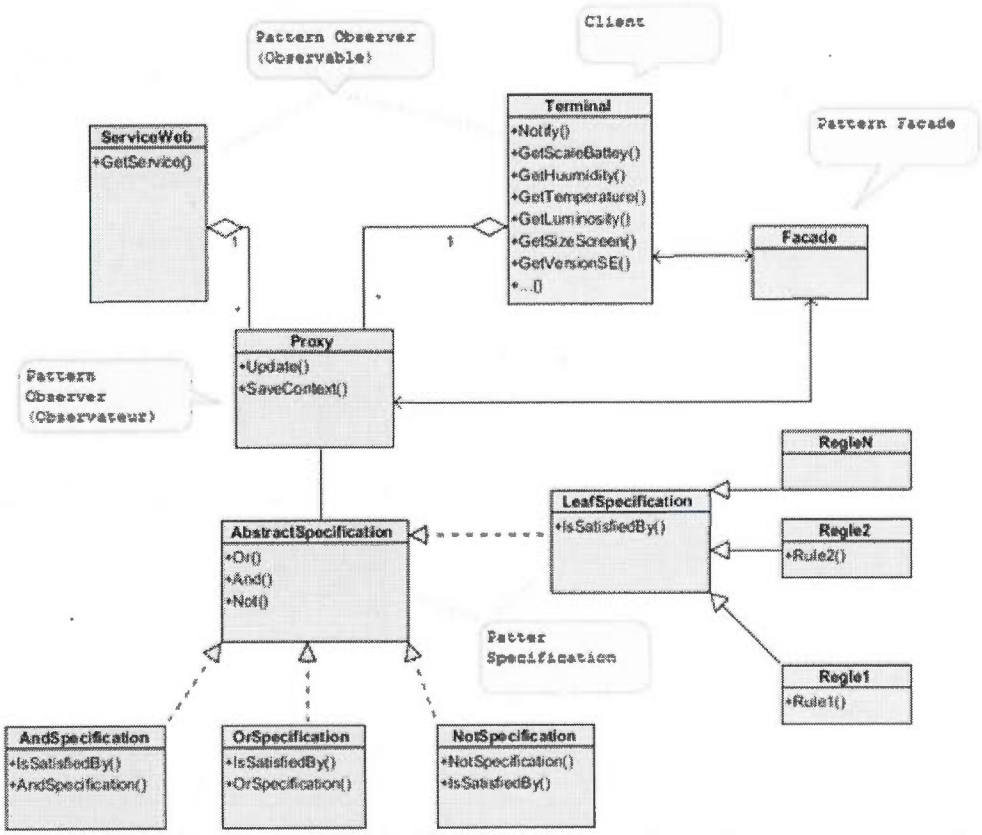

Figure B.1 Structure du patron « **Adaptation d'interface au contexte** »

B.6 Participant

Tableau B.1 Composition du patron « Adaptation d'interface au contexte »

Patron de conception	Entité	Rôle
Observer	Service- Web	• Sujet du patron Observer • Représente le descriptif du service à représenter. Autrement dit, Il contient les données de l'application et il est situé sur un serveur distant sous forme d'un fichier « XML, Json, etc.. » • Il notifie le proxy « Observateur » suite à chaque changement observé.
	Terminal	• Sujet du patron Observer • À travers les données extraites par les capteurs qu'il contient, il représente la source du contexte de l'utilisateur. • Il notifie le proxy « Observateur » suite à chaque changement observé
	Proxy	• L'observateur commun des deux sujets précédents. • Récupère les données du terminal et du service Web une fois notifié. • Transmet l'information récupérée au patron « Spécification »
Specification	AbstractSpecification	• Gestion des règles métiers • Composer entre plusieurs règles
	« And/Or/Not »SpeS pecification	• Utilisé pour assurer la combinaison des règles métier
	LeafSpecification	• Représente les règles métiers qui déterminent l'orientation de l'application

Façade	Terminal	• Représente implicitement le client. • Sur son écran sera représentée l'interface « le service » • Encapsule le sous-système de suggestion d'interface établi par la base de connaissance à base de règle (Section 4.2.3)

B.7 Collaboration

Selon(Boinot *et al.*, 2000), nous avons distingué deux types d'adaptation, soit *OnChange* et *OnAction*. Le premier exige un nouveau comportement à l'instant marqué par un changement dans le contexte de l'utilisteur, la plateforme ou l'environnement. L'Adaptation OnAction, quant à elle, exige la même chose mais suite à une action executée par le client.

Au cours du processus d'adaptation *OnAction*, le client exécute une opération en invoquant l'unique façade de son interface représentée par le composant *Façade*du modèle. Cette dérnière envoie une requête au composant *Observateur*pour lui notifier la détection d'un contexe. Le Proxy,à son tour, récupére les informations du Terminal et du Service et les transmet à *AbstarctSpecification*. À ce niveau, le contenu de l'interface sera dégagé. Une fois cette tâche de raisonnement sur le contexte est achevée, *AbstractSpécification* retourne le backend de l'interface au proxy qui invoque à son tour la façade. Cette entité délègue la tâche de suggestion d'interface à l'approche à base de règle puis se charge de l'exposer sur le terminal.

Au cours du processus d'adaptation *OnChange*, le sujet du patron *Observateur/Observable*, modélisé par le composant *Terminal*ou *Service,*notifie le proxy par une requête que le contexte capté a subi un changement. Le proxy récupére le nouveau contexte Service et le transmet à *AbstarctSpecification*. À ce niveau, le contenu de l'interface sera dégagé. Une fois cette tâche de raisonnement sur le contexte est achevée, *AbstractSpécification* retourne le backend de l'interface au proxy qui invoque à son tour la façade. Cette entité délègue la tâche de suggestion d'interface à l'approche à base de règles puis se charge de l'exposer sur le terminal.

B.8 Conséquence

Nous dégageons les propriétées suivantes après l'utilisation du patron de conception « Adaptation d'interface au contexte » :

- Séparation de l'interface du noyau fonctionnel du système;
- La transparence du contexte envers le client;
- Une bonne gestion des règles;
- Il permet au système d'être extensible et facile à maintenir;
- La suggestion d'interface peut se faire d'une façon indépendante du système;

B.9 Patrons apparenté

Le patron proposé est composé des modèles suivants :

- Façade
- Spécification
- Observer

RÉFÉRENCES BIBLIOGRAPHIQUES

Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M. et Steggles, P. (1999). *Towards a better understanding of context and context-awareness. Handheld and ubiquitous computing (27 au 29 septembre 1999)*, Actes du colloque, 1999, Germany : Springer.

Aubert–Olivier, O. et Beugnard–Antoine, A. (2001). Adaptive Strategy Design Pattern. Récupéré de http://olivieraubert.net/download/koalaplop01.pdf

Aubert, O. (2001). *Patron de conception pour l'analyse et la construction de systèmes à comportements autoadaptatifs.* Rennes 1. Récupéré de *theses.fr*.http://www.theses.fr/2001REN10140.

Babar, M.A., Chen, L. et Shull, F. (2010). Managing variability in software product lines. *Software, IEEE, 27*(3), 89-91, 94.

Baldauf, M., Dustdar, S. et Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing, 2*(4), 263-277.

Bandelloni, R. et Paternò, F. (2004). Migratory user interfaces able to adapt to various interaction platforms. *International journal of human-computer studies, 60*(5), 621-639.

Boinot, P., Marlet, R., Noye, J., Muller, G. et Consel, C. (2000). *A declarative approach for designing and developing adaptive components. The Fifteenth IEEE International Conference on Automated Software Engineering.*, Actes du colloque, 2000, France : IEEE.

Brown, A., Johnston, S. et Kelly, K. (2002). Using service-oriented architecture and component-based development to build web service applications. *Rational Software Corporation*

Calvary, G. et Coutaz, J. (2002). Plasticité des Interfaces: une nécessité. *Actes des deuxièmes Assises nationales du GDR I, 3*, 247-226.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. et Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers, 15*(3), 289-308.

Chaari, T., Laforest, F. et Flory, A.(2005) Adaptation des applications au contexte en utilisant les services web. Dans Communication présentée à /au Proceedings of the 2nd French-speaking conference on Mobility and ubiquity computing p. 111-118) : ACM.

Chen, G. et Kotz, D. (2000). *A survey of context-aware mobile computing research.* : Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.

Compton, P., Peters, L., Edwards, G. et Lavers, T.G. (2006). Experience with ripple-down rules. *Knowledge-Based Systems, 19*(5), 356-362.

Dey, A.K., Salber, D., Futakawa, M. et Abowd, G.D. (1999). *An architecture to support context-aware applications.* (93-23). Georgia : Institute of Technology.

El Boussaidi, G. et Mili, H. (2004). *Les patrons de conception: représentation et mise en oeuvre.* (rapport de recherche). Montreal : LATECE.

Eric Evans, M.F. (1999). *Specifications.* Récupéré le 01-08-2013 2013 de http://www.martinfowler.com/apsupp/spec.pdf

Fayad, M.E. et Johnson, R.E. (1999). *Domain-specific application frameworks: framework experience by industry.* : John Wiley & Sons, Inc.

Fielding, R. (2000). Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture*, 76-85.

Gabillon, Y., Calvary, G. et Fiorino, H. (2011). Composition d'Interfaces Homme-Machine en contexte: approche par planification automatique. *Technique et Science Informatiques (TSI), 30*(10), 1143-1166.

Gamma, E., Helm, R., Johnson, R. et Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. : Pearson Education.

Ganneau, V., Calvary, G. et Demumieux, R. (2007). *Métamodèle de règles d'adaptation pour la plasticité des interfaces homme-machine. Proceedings of the 19th International Conference of the Association Francophone d'Interaction Homme-Machine*, Actes du colloque, 2007, New York : ACM.

Ganneau, V., Calvary, G. et Demumieux, R. (2008). *EMMA: modèle utilisateur pour la plasticité des interfaces homme-machine en mobilité. Proceedings of the 4th French-speaking conference on Mobility and ubiquity computing*, Actes du colloque, 2008, New York : ACM.

Grenier, P.L. (2011). *Mobilité et application mobile : quelques statistiques*. Récupéré le 05-10-2013 2013 de http://www.pierrelucgrenier.com/2011/11/22/mobilite-quelques-statistiques/

Hachani, S. et Dupuy-Chessa, S. (2009). *Une approche générique pour l'adaptation dynamique des IHM au contexte. Proceedings of the 21st International Conference on Association Francophone d'Interaction Homme-Machine*, Actes du colloque, 2009, New York : ACM.

Hariri, A., Lepreux, S., Tabary, D. et Kolski, C. (2009). Principes et étude de cas d'adaptation d'IHM dans les SI en fonction du contexte d'interaction de l'utilisateur. *Ingénierie des systèmes d'information, 14*(3), 141-162.

Holland, J.H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artific*. : A Bradford Book.

Hossein Sadat, S. et Ghorbani, A.A. (2005). *A presentation specification language for adaptive web systems. Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual*, Actes du colloque, 2005, Canada : IEEE.

Johnson, R.E. (1997). Frameworks=(components+ patterns). *Communications of the ACM, 40*(10), 39-42.

Joyeux, P. (2006, juin 2006). *IHM pour des produits industriels :*

conception automatisée, principes, avantages, inconvénients: Rapport probatoire. : Conservatoire National des Arts et Métiers Récupéré de http://pascal.joyeux.free.fr/PROBATOIRE/Rapport/Probatoire_corige.pdf

Klemisch, K., Weber, I. et Benatallah, B. (2013). *Context-Aware UI Component Reuse. Advanced Information Systems Engineering*, Actes du colloque, 2013, Spain : Springer.

Lecoz, N. (2010). *Drools et les moteurs de règles*. Récupéré le 09-09-2013 2013 de http://blog.xebia.fr/2010/01/08/drools-et-les-moteurs-de-regles/

Lok, S. et Feiner, S. (2001). A survey of automated layout techniques for information presentations. *Proceedings of SmartGraphics, 2*

Motti, V.G. et Vanderdonckt, J. (2013). A Unified Model for Context-aware Adaptation of User Interfaces. *Revista Română de Interacţiune Om-Calculator, 6*(3), 211-248.

Richards, D. (2009). Two decades of Ripple Down Rules research. *Knowledge Eng. Review, 24*(2), 159-184.

Schilit, B.N. et Theimer, M.M. (1994). Disseminating active map information to mobile hosts. *Network, IEEE, 8*(5), 22-32.

Schmidt, A. (2000). Implicit human computer interaction through context. *Personal Technologies, 4*(2-3), 191-199.

Shimada, A., Gerofi, B., Hori, A. et Ishikawa, Y. (2013). *Proposing a new task model towards many-core architecture. Proceedings of the First International Workshop on Many-core Embedded Systems*, Actes du colloque, 2013, USA : ACM.

Siau, K. et Halpin, T. (2001). *Unified Modeling Language: Systems Analysis, Design and Development Issues*. Hershey PA : IGI Global. Récupéré de Google books http://books.google.fr/

Simonin, J. et Carbonell, N. (2007). Interfaces adaptatives Adaptation dynamique\a l'utilisateur courant. *arXiv preprint arXiv:0708.3742*

Sottet, J.-S. (2008). *Mega-IHM: Malléabilité des Interfaces Homme-Machine Dirigée par les Modèles.* PhD thesis, JOSEPH FOURIER-GRENOBLE 1.

Strassner, J., Samudrala, S., Cox, G., Liu, Y., Jiang, M., Zhang, J., Meer, S.v.d., Foghlú, M.Ó. et Donnelly, W. (2008). *The design of a new context-aware policy model for autonomic networking. International Conference on Autonomic Computing (2 au 6 juin 2008)*, Actes du colloque, 2008, Chicago : IEEE.

Sun Microsystems. (2006). *Java Compiler Compiler tm (JavaCC tm) - The Java Parser Generator*. Récupéré le 01 novembre 2013 de https://javacc.java.net/

Tamine-Lechani, L., Boughanem, M. et Daoud, M. (2010). Evaluation of contextual information retrieval effectiveness: overview of issues and research. *Knowledge and Information Systems, 24*(1), 1-34.

Thevenin, D. et Coutaz, J. (1999). *Plasticity of user interfaces: Framework and research agenda. Proceedings of INTERACT*, Actes du colloque, 1999, Amesterdam : IOS Press

Thomas, A., Parkinson, J., Moore, P., Goodman, A., Xhafa, F. et Barolli, L. (2013). *Nudging through Technology: Choice Architectures and the Mobile Information Revolution. Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)* Actes du colloque, 2013, UK : IEEE.

Vlissides, J., Helm, R., Johnson, R. et Gamma, E. (1995). Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley, 49*, 120.

Weiser, M. (1991). The computer for the 21st century. *Scientific american, 265*(3), 94-104.

Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM, 36*(7), 75-84.

Yau, S.S. et Liu, J. (2006). *Incorporating situation awareness in service specifications. Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing.*, Actes du colloque, 2006, China : IEEE.

Yu, J. (2008). *A UI-driven Approach to Facilitating Effective Development of Rich and Composite Web Applications.* University of New South Wales Sydney, NSW 2052, Australia.