Faculté des Sciences de Tunis

TELECOM et Management SudParis

Rapport de Stage
Élève Ingénieur en Informatique

INGÉNIERIE DIRIGÉE PAR LES MODÈLES ET COMPOSANTS SENSIBLES AU CONTEXTE

Mehdi ZAIER

Responsable de stage : Chantal TACONET

Juin 2008

Ce stage d'ingénieur a été réalisé au sein du laboratoire CNRS Samovar, Équipe MARGE du département **Informatique** de **TELECOM et Management SudParis**

Remerciements

C'est une tâche très agréable, mais bien délicate, de présenter mes remerciements à tous ceux qui m'ont aidé dans la réalisation de ce travail.

Qu'il me soit permis d'exprimer en premier lieu ma gratitude à Madame Chantal TACONET Maître de Conférences au TELECOM et Management SudParis (ex INT), qui a proposé le sujet, a accepté de m'encadrer et m'a fait profiter de son savoir et de sa grande expérience. Qu'elle trouve, ici, le témoignage de ma sincère reconnaissance et mes vifs remerciements.

Monsieur Samir MOALLA est à l'origine de cette « aventure ». Comment doser, non seulement dans son enseignement, mais aussi dans les conversations que nous avons eues, l'apport considérable dont j'ai bénéficié. Je le remercie aussi pour l'intérêt qu'il m'a toujours manifesté.

Ce travail sera examiné et évalué par mes chers professeurs, qu'ils soient vivement remercier pour avoir accepté de faire partie du Jury. J'exprime ma très haute considération et mes vifs remerciements à tous mes enseignants du Département des Sciences de l'Informatique pour les enseignements qu'ils m'ont prodigués tout au long de mes années de formation dans la filière Ingénieur en Sciences de l'Informatique et en première année de Mastère de Sciences de l'Informatique.

Ce travail a été réalisé au sein de l'équipe MARGE au Département Informatique du TELECOM et Management SudParis, pendant une durée d'environ 4 mois et demi. Je tiens donc à témoigner toute ma reconnaissance à tous les

Enseignants, Personnels administratifs et techniques de cette Institution pour l'assistance et l'aide qu'ils m'ont prodigué pour réaliser ce travail. Je remercie particulièrement Messieurs Samir TATA, Denis CONAN, Bruno DEFUDE, Mesdames Sophie CHABRIDON, Brigitte HOUASSINE et mes collègues Zied, Mahmoud, Léon, Mounis, Salah, Mohamed, Soumaya, Wided, Ines, Marie, etc.

Durant mon séjour à Evry, j'ai pu bénéficier d'une allocation de la part du TELECOM et Management SudParis. À cette occasion, je veux remercier tous les Responsables de cette Institution pour l'ensemble des facilités qui m'ont été offertes.

Je me dois de remercier vivement Messieurs Hassen AMRI et Khaled BSAIESS respectivement Doyen de la Faculté des Sciences de Tunis et Directeur du Département des Sciences de l'Informatique d'avoir facilité ma mission. Il serait injuste de ne pas y associer Madame Dorra AMMAR GARGOURI Sous Directrice des Relations avec l'Environnement, Insertion Professionnelle et Formation Continue à l'Université de Tunis El Manar.

De très nombreuses personnes dont les noms ne sont pas cités m'ont aidé, encouragé, conseillé. J'espère pouvoir un jour leur témoigner ma reconnaissance.

Enfin, je tiens à exprimer mes affectueuses reconnaissances à ma sœur Sonia et à mes parents pour l'aide et les encouragements qu'ils m'ont prodigués particulièrement tout au long de cette période.

Table des matières

Table des figures

Introduction générale

Le présent travail s'inscrit dans le cadre de la réalisation d'un projet de fin d'étude des élèves ingénieurs en Sciences de l'Informatique de la Faculté des Sciences de Tunis. Il est réalisé au sein de l'équipe MARGE (Middleware pour Applications Réparties avec Gestion de l'Environnement) au Département Informatique de l'Institut TELECOM SudParis ; suite à une convention tripartite signée entre le Département Informatique de la Faculté des Sciences de Tunis, le Département Informatique TELECOM et Management SudParis et moi même pour une durée d'environ quatre mois et demi. Le Stage proposé porte sur l'Ingénierie Dirigée par les Modèles et composants sensibles au contexte. Il entre dans le cadre d'un projet de l'Institut TELECOM intitulé CASAC (Composants Auto-adaptables Sensibles Au Contexte) dans lequel est aussi impliquée l'équipe CAMA de TELECOM Bretagne. L'encadrement au département informatique de l'INT a été assuré par Madame Chantal TACONET, Maître de conférences dans cette Institution.

Depuis quelques années, nous assistons à l'émergence de nouvelles plates-formes distribuées, qualifiées d'ubiquitaires, qui ne sont plus limitées à une interconnexion de stations de travail définissant un réseau stable. Ces plates-formes intègrent éventuellement des machines puissantes et robustes mais aussi, et de plus en plus, des équipements mobiles et à faibles ressources (ordinateurs portables, PDA, téléphones mobiles, capteurs ...). Bien que ce type de réseaux soit de plus en plus répandu, leur exploitation effective constitue encore un défi. Il reste en effet difficile de construire, de déployer et de maintenir des applications distribuées dans les environnements pervasifs en tenant compte de l'hétérogénéité, de la mobilité ou de la volatilité des

équipements. Dans un contexte général où la mobilité des utilisateurs et l'ubiquité des applications se généralisent, les fournisseurs de logiciels souhaitent offrir des applications adaptables, appelées aussi applications sensibles au contexte ; c'est à dire dont la structure ou le comportement change en fonction de situations de contextes. Les modèles et plates-formes de composants logiciels actuels ne permettent l'adaptation au contexte que par un contrôleur extérieur. Cette approche ne permet pas aux composants adaptables d'être auto-suffisants. Cela rend plus complexe l'écriture et la maintenance des applications adaptables et diminue grandement la possibilité de réutiliser un composant (adaptable) d'une application dans une autre.

Suite à ce constat, le projet CASAC s'est intéressé à proposer une approche générique pour permettre à un composant de devenir sensible au contexte. Pour cela, le composant fournit une description des informations de contexte dont il a besoin. La plate-forme d'exécution peut alors synthétiser les sondes nécessaires à la collecte de ces informations. Ainsi, le contrôle de l'adaptation du composant peut être embarqué dans le composant lui-même qui devient alors un composant auto-adaptable. Les solutions proposées et développées durant ce stage s'intègrent dans un cadre logiciel générique pour le développement et l'exécution des applications sensibles au contexte. L'approche explorée dans ce stage consiste à intégrer la sensibilité au contexte dès la modélisation de l'application. Il s'agit de définir dans la phase de modélisation des applications, les observables à collecter, les situations d'adaptation à identifier et les réactions d'adaptation.

Un méta-modèle de sensibilité au contexte a été ainsi proposé pour cela par l'équipe MARGE du Département Informatique de l'Institut Télécom SudParis.

Ce méta-modèle sera utilisé pour explorer les modèles d'applications pour générer une partie du code nécessaire à la gestion de la sensibilité au contexte de l'application. L'objectif de notre travail durant cette période de stage est d'étendre le méta-modèle de sensibilité au contexte d'une part et d'écrire des générateurs de code de gestion de sensibilité au contexte d'autre part.

Dans un premier temps, ce méta-modèle sera revu et complété au vu des besoins exprimés pour l'expression de compositions de contextes et pour modéliser des réactions d'adaptation qui seront mises en œuvre dans la membrane d'un composant pour qu'il devienne sensible au contexte. Dans un deuxième temps, le modèle de sensibilité au contexte propre à un composant applicatif servira à la génération de code extra-fonctionnel de la membrane du composant auto-adaptable pour la liaison avec des sondes d'observation et pour la réalisation des adaptations.

Nous commençons dans ce rapport par détailler, dans un premier chapitre, la problématique et les objectifs du projet. Dans le deuxième chapitre, nous définissons la sensibilité au contexte ainsi que la modélisation du contexte qui sont les éléments clés de notre travail. Dans le troisième chapitre, nous décrivons le composant COSMOS et la manière avec laquelle il gère les information de contexte. Il s'agit d'un type de collecteur de contexte qui utilise les éléments observables afin de collecter l'information de contexte et permettre à l'application de s'adapter aux différents changements. Nous détaillons dans le quatrième chapitre notre démarche et le processus de développement que nous avons adopté pour introduire la sensibilité au contexte au niveau d'un composant. Enfin, nous terminons par la conclusion et les perspectives de nos travaux.

Chapitre 1

Problématique et Objectifs du Stage

Pour commencer, nous présentons dans la première section, le projet CASAC ainsi que les motivations pour lesquelles ce projet a été mis en œuvre. Nous détaillons, dans la deuxième section, les différents lots qu'il comporte et nous situons notre travail par rapport à ces différents lots. La troisième section cite les différents impacts attendus par notre projet.

1.1 Motivation

Dans un contexte général où la mobilité et l'ubiquité deviennent des exigences des utilisateurs, les fournisseurs de logiciels souhaitent pouvoir offrir des applications adaptables à leurs clients. Or, les technologies et méthodes actuelles étant assez rudimentaires, le coût de réalisation d'une application adaptable est relativement élevé (par rapport aux autres types d'applications). Pire encore, les composants adaptables sont difficiles à réutiliser dans le cadre d'autres applications ou d'autres plates-formes.

Globalement, l'objectif du projet est de faciliter la réalisation d'applications adaptables et donc, d'en diminuer le coût de réalisation. Pour cela, il sera question de :

- Rendre plus simple la conception de composants (et donc d'applications) auto-adaptables ;
- Séparer les préoccupations entre l'observation des données de l'environnement et leur utilisation pour décider des adaptations éventuellement nécessaires ;
- Rendre déclarative la définition de l'observation par le composant. Le langage employé doit être simple, donc dédié au domaine, afin d'envisager par la suite d'ajouter une analyse de la cohérence des informations de contexte obtenues ;
- Rendre explicite les exigences d'observation du contexte au sein de l'architecture.

1.2 Présentation du projet CASAC

Le contenu du projet CASAC couvre trois lots. Le premier lot concerne la définition d'un langage de description d'observation du contexte. Il s'agit donc de définir un langage dédié qui facilite la composition d'informations de contexte. Le deuxième Lot consiste à introduire la notion de qualité dans les observations de contexte afin de les rendre plus précises et ainsi améliorer l'adaptation. Le troisième lot, qui relie les deux autres études, applique une approche d'ingénierie des modèles pour modéliser la sensibilité au contexte dans des composants applicatifs. Le présent travail s'intéresse au troisième lot. Nous décrivons dans

la première sous-section ce troisième lot. la deuxième sous-séction détaille les différentes tâches qu'il contient. À la fin, nous consacrons une section pour présenter les impacts attendus par ce projet.

1.2.1 Description du travail à mettre en œuvre

L'ingénierie logicielle dirigée par les modèles permet de séparer les préoccupations métier, des préoccupations liées aux plates-formes d'exécution. La logique métier de l'application est décrite à un niveau d'abstraction élevé. Grâce aux techniques de transformation de modèles, le modèle de l'application permet de produire une partie du code de l'application. Certaines des adaptations au contexte de l'application doivent être définies au niveau du modèle de l'application. L'approche adoptée consiste à intégrer dans le processus de modélisation de l'application une phase de définition de sa sensibilité au contexte. Le méta-modèle de contexte proposé dans CASAC définit les méta-classes de définition de contexte, ainsi que leurs associations avec les éléments du méta-modèle de l'application qui définissent leurs politiques d'adaptation.

Le méta-modèle de contexte élaboré par l'équipe MARGE durant un autre projet intitulé ITEA S4ALL (Services-for-All) sera donc repris et étendu pour les besoins du projet CASAC. Le modèle de sensibilité au contexte de chacun des composants applicatifs sera utilisé par la chaîne de production de la membrane du composant Fractal auto-adaptable : pour la liaison avec des sondes d'observation des collecteurs de type COSMOS et pour la réalisation des adaptations.

1.2.2 Description des différentes tâches à réaliser

La première tâche étant de bien étudier le méta-modèle de contexte fourni, de le revoir et le compléter au vu des différents besoins pour l'expression de politiques de contexte COSMOS ainsi que pour modéliser des réactions d'adaptation qui seront mises en œuvre dans la membrane Fractal pour qu'il devienne sensible au contexte.

À partir de ce méta-modèle, il s'agit de construire pour une application donnée, un modèle conforme à ce méta-modèle. Le modèle construit nous permet de connaître la liste des nœuds de contexte et leur mode d'interaction avec l'application. Le modèle dirige aussi les différentes interactions avec les collecteurs COSMOS.

Parmi les principales interactions entre l'application et les collecteurs, nous citons à titre d'exemple :

- La détermination des différents éléments observables associés au modèle de l'application.
- L'enregistrement auprès des nœuds de contexte COSMOS afin de recevoir les notifications concernant les variations.
- Éventuellement, l'observation directe de ces nœuds de contexte, si l'application a besoin de savoir l'état de son environnement.
- La gestion des différents types de messages COSMOS reçus et la réalisation des tâches prévues pour chacun de ces types de messages.

Nous distinguons deux modes d'utilisation du modèle :

- Utilisation dynamique pendant l'initialisation du composant. Il s'agit dans cette méthode de parcourir d'une manière générique le modèle écrit avec

les APIs générées par le méta-modèle. Cette méthode offre la possibilité de modifier le modèle pendant l'exécution pour une meilleur adaptation.
– Utilisation statique lors de la compilation pour la génération du code d'interaction. Le code généré avec cette méthode est spécifique à l'application. Il s'agit d'un code réduit transportable sur petit matériel.

Il s'agit de réaliser les deux méthodes afin de comparer les résultats trouvés en terme de temps d'exécution et d'empreintes mémoires.

1.3 Impact

Les résultats attendus du projet sont de nature à enrichir l'offre existante en termes de composants logiciels avec sensibilité au contexte. Ceci vise à faciliter le développement d'applications dans de nombreux domaines tels que l'informatique ubiquitaire, les réseaux ambiants, la télésurveillance, et ouvre la voie à de multiples applications tirant parti de la sensibilité au contexte. Des exemples de telles applications sont les guides touristiques avec navigation contextuelle, les applications d'annotations contextuelles ou encore les applications avec enrichissement contextuel et réalité augmentée telles que les jeux multi-joueurs. Depuis quelques années, de nombreux projets internationaux se sont intéressés à la sensibilité au contexte des applications.

La particularité du projet est d'intégrer la modélisation de la sensibilité au contexte dans la phase de conception des applications et d'utiliser ces descriptions pendant l'exécution pour diriger la sensibilité au contexte.

Chapitre 2

Modélisation de contexte

Il existe de nombreux travaux liés à la modélisation de contexte. C'est pourquoi cette étude commence par une présentation des domaines de la modélisation. L'étude de recherche présentée ici s'appuie sur l'état de l'art d'un projet intitulé CAPUCCINO (**C**onstruction et **A**daptation d'a**PP**lications **U**biquitaires et de **C**omposants d'**IN**tergiciels en environnement **O**uvert pour l'industrie du commerce) [11].

Dans ce chapitre, nous détaillons les travaux de la modélisation de contexte. La première section introduit la notion d'applications ubiquitaires ainsi que l'importance de l'adaptation au contexte pour ce genre d'application. La deuxième section s'intéresse aux applications sensibles au contexte et aux mécanismes qu'elles emploient pour assurer leurs adaptations. Nous détaillons dans la troisième section la notion du modèle du contexte et nous présentons différentes définitions afin de clarifier cette notion. Dans la dernière section, nous faisons le lien entre les « modèles de contexte » et les « systèmes sensibles au contexte » .

2.1 Introduction aux Applications ubiquitaires

La multiplication des terminaux mobiles et la généralisation des réseaux sans fil nécessitent des changements dans la manière dont les applications logicielles sont conçues et exploitées. Ce nouveau défi, appelé informatique ubiquitaire, soulève de nombreux problèmes à savoir l'hétérogénéité des équipements, la limite des ressources, la distribution des applications, la mobilité des terminaux, la sécurité, la découverte des services, le déploiement automatique du logiciel sur le terminal, etc. Il n'existe actuellement pas de solution générale qui couvre le cycle complet du processus de construction des applications ubiquitaires, de la conception à l'exploitation.

Afin d'adapter ces applications aux différents contextes, il faut leur fournir de nouvelles informations en plus de celles qu'elles traitent. L'application doit donc avoir conscience de l'environnement dans lequel elle s'exécute, sur quel type de terminal, de quel type de réseau elle dispose pour la communication, à quel endroit se situe le terminal mobile. Toutes ces informations constituent le contexte d'exécution de l'application et forment un nouveau type d'information à traiter.

2.2 Applications sensibles au contexte

Les applications ubiquitaires sont caractérisées par le fait qu'elles s'exécutent dans des environnements variables. La modélisation de contexte doit donc permettre de représenter de manière abstraite les différents environnements d'exécution de ces applications, afin d'identifier les contextes d'exécution des applications, et aussi de définir la sensibilité des applications

à leur contexte d'exécution. Dans cette section nous présenterons la notion d'application sensible au contexte, la terminologie et les exigences en ce qui concerne la modélisation de contexte.

Les applications sensibles au contexte sont, d'après la définition de Brown [2], des applications dont le comportement peut varier en fonction du contexte. Dans le cadre des applications sensibles au contexte, Dey [7] a fourni cette définition du contexte : « Le contexte est toute information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un lieu, ou un objet qui est considéré comme pertinent pour l'interaction entre un utilisateur et une application, incluant l'utilisateur et l'application elle-même ». Suivant la définition de Dey, le contexte peut caractériser l'application elle-même, mais aussi des entités extérieures à l'application. Les informations de contexte peuvent ainsi être de nature diverse. Dans les applications traditionnelles, seuls les utilisateurs interagissent avec l'application. Dans les applications sensibles au contexte, l'application peut évoluer aussi avec le contexte dans lequel elle s'exécute.

De manière à ce qu'une application soit sensible au contexte, il faut la doter de mécanismes qui lui permettent d'obtenir et d'analyser les informations de contexte en premier lieu et puis de réaliser les adaptations aux situations de contexte dans un second lieu. Lors du développement des premières applications sensibles au contexte, le développeur de l'application gérait entièrement la sensibilité au contexte depuis l'interaction avec les capteurs d'acquisition des données de contexte jusqu'aux opérations d'adaptation. L'inconvénient de cette approche est que le fort couplage entre les capteurs et l'application rend difficile la modification des applications lors de changement de technologies de

capteurs par exemple.

Les intergiciels (en anglais, Middleware) de nouvelles générations doivent faciliter le développement d'applications sensibles au contexte et prendre en charge une partie de la gestion de la sensibilité au contexte. Cette prise en charge ne peut s'envisager que si l'intergiciel peut s'appuyer sur des services de gestion de contexte qui eux-mêmes peuvent s'appuyer sur la modélisation des informations de contexte. L'intergiciel, lui-même, peut s'appuyer sur ces techniques pour être sensible au contexte.

S'appuyer sur un modèle de contexte doit permettre d'ajouter plus facilement de nouveaux types de contexte, de nouveaux types de capteurs, de nouvelles techniques d'analyse de situations de contexte, et de nouveaux modes d'adaptation et de sensibilité au contexte dans les applications.

2.3 Modèles de contexte : Objectifs et terminologie

2.3.1 Objectifs des modèles de contexte

La modélisation de contexte permet de définir, pour chaque application, les contextes qui ont une influence sur l'application et les interactions entre le contexte et l'application. Cette modélisation dépend bien sûr du contexte même où l'application est utilisée et de la finalité de la modélisation. La question qui se pose est de savoir si il est préférable d'avoir une modélisation de contexte dédiée à une application ou à une classe d'applications, ou plutôt une modélisation générique applicable à tout type d'application. Le modèle générique est probablement une quête impossible, cependant il semble intéressant d'avoir

une base ou un support de modélisation commun.

Le cœur de modélisation doit être suffisamment ouvert pour que des extensions puissent être ajoutées pour différents domaines d'applications. Il est souhaitable que le modèle de contexte puisse évoluer pendant le cycle de vie de l'application. L'application doit pouvoir puiser les informations nécessaires à son adaptation sans pour autant qu'il soit nécessaire de toucher au code métier de l'application.

2.3.2 Définitions

Les définitions suivantes seront utilisées pour parler des éléments décrits dans notre modèle de contexte.

Système sensible au contexte « Un système sensible au contexte est un système dont le comportement ou la structure peut varier en fonction de l'état de l'espace des informations de contexte » .

Le terme « espace des informations de contexte » est défini ci-dessous. Le terme système sensible au contexte est plus général que applications sensibles au contexte. Le système peut être une application, mais aussi l'intergiciel ou le système d'exploitation. Les définitions qui suivent sont relatives à un système sensible au contexte.

Entité (observable) « Élément représentant un phénomène physique ou logique (personne, concept, etc.) qui peut être traité comme une unité indépendante ou un membre d'une catégorie particulière, et auquel des « éléments observables » (terme défini ci-dessous) peuvent être associés. »

(Élément) Observable « Un élément observable est une abstraction qui définit un type d'informations à observer. Pour un système donné, un élément observable est rattaché à une entité observable. Un élément observable donne lieu à des observations (défini ci-dessous). »

Par exemple, à partir de l'entité observable « terminal » , sont associés les éléments observables suivants : mémoire vive disponible (type entier), liste des dispositifs d'interaction avec l'utilisateur (type : ensemble de chaînes de caractères), liste des connexions réseaux (type : ensemble de chaînes de caractères), connexion réseau disponible (type booléen).

(Élément) Observable Composite « Un élément observable composite est un élément observable dont les observations sont obtenues par une fonction prenant en entrée les observations d'un ou plusieurs éléments observables. »

Observation « Chaque élément observable est associé à une ou plusieurs observations, chaque observation définit un état de l'élément observable. »

Élément observable immuable « Un élément observable immuable est un élément observable dont les observations auront la même valeur pendant toute la durée de vie du système. »

Espace des informations de contexte « L'espace des informations de contexte d'un système est caractérisé par l'état des différents éléments le caractérisant : les entités observables, les éléments observables et l'état des observations attachées aux éléments observables. »

Situation d'adaptation « Une situation d'adaptation est un élément obser-

vable qui permet de repérer un changement d'état dans l'espace des informations de contexte. Ce changement d'état significatif pour le système nécessite une réaction dans le système. Cette réaction est appelée adaptation. »

Une situation d'adaptation est un élément observable dont les changements d'état génèrent une modification dans le système sensible au contexte.

2.4 Modèles de contexte et systèmes sensibles au contexte

Nous parlons de méta-modèles de contexte pour la définition de la structure des informations de sensibilité au contexte (quels sont les éléments qui seront présents dans le modèle et quelles sont les relations entre ces différents éléments).

Nous parlons de modèles de contexte pour décrire les informations propres à une application ou à un domaine d'application. Un modèle décrira par exemple les entités observables et les éléments observables d'un système donné. Les instances de modèles correspondront quant à eux à une réalité pendant les différentes exécutions de systèmes sensibles au contexte (une instance par exécution). Pendant une exécution, nous observons non pas un terminal mais le terminal utilisé par l'utilisateur X.

Un modèle de contexte utilisé par un intergiciel, doit non seulement identifier les entités observables, les éléments observables, mais aussi, être complété pour décrire les méthodes de collecte du contexte, les éléments permettant l'analyse de situations d'adaptation et les informations permettant de réaliser

les adaptations des applications pour qu'il puisse réaliser la prise en compte de la sensibilité au contexte des applications.

Notre méta-modèle se différencie par les entités du modèle mais aussi par les relations exprimées entre ces différents éléments. Les relations entre les différents éléments du modèle sont essentielles pour parcourir les différents éléments du modèle. Elles sont déterminantes pour l'utilisabilité du méta-modèle.

Certains systèmes sensibles au contexte ont seulement besoin d'identifier un profil d'utilisation. Par exemple, certaines applications ont besoin à leur démarrage de connaître le type du mobile sur lequel elles sont utilisées. Pour ces applications, il s'agit d'identifier les caractéristiques immuables du mobile sur lesquelles elles sont utilisées. Pour ces applications, une interrogation du profil du terminal mobile utilisé leur permet d'adapter leur configuration. D'autres systèmes doivent être utilisés avec un gestionnaire ou collecteur de contexte. Dans ce dernier cas, le modèle de contexte peut définir les interactions entre l'application et le collecteur de contexte : identification des collecteurs de contexte, identification du mode d'interaction synchrone/asynchrone, périodicité des interactions, qualité des informations collectées.

Certains méta-modèles de contexte permettent d'analyser l'espace des informations de contexte afin d'identifier des situations d'adaptation. C'est le cas des ontologies utilisées conjointement avec un moteur d'inférence qui à l'aide de règles permettent d'identifier des situations d'adaptation en parcourant les instances de modèle de contexte.

Enfin, un méta-modèle de contexte peut être complété par un méta-modèle de sensibilité au contexte qui permet de définir les adaptations du système qui pourront être appliquées en fonction de situations d'adaptation.

2.5 Conclusion

Ce chapitre nous a permis de bien assimiler la notion de la modélisation de contexte qui définit pour chaque application, les entités observables, les éléments observables ainsi que les situations d'adaptations permettant à l'application de tenir compte des différents changements et de s'adapter en conséquence. Introduire cette notion d'adaptation depuis la phase de modélisation, permet une meilleure gestion de la sensibilté au contexte des applications surtout dans un environnement ubiquitaire.

Chapitre 3

Gestion des informations de contexte avec COSMOS

Dans notre travail, nous utilisons un ensemble de collecteurs d'informations de contexte afin de déterminer les différentes variations des éléments à observer et ainsi permettre à l'application de prendre conscience de ces différentes variations. Nous travaillons avec des collecteurs de type COSMOS (**CO**ntext entitie**S** co**M**positi**O**n and **S**haring) qui est une proposition de Conan [4]. Il s'agit d'un collecteur qui compose l'information de contexte et permet de construire des gestionnaires de contexte dans des environnements ubiquitaires.

Nous détaillons dans ce chapitre ce collecteur COSMOS. La première section présente ses motivations et ses objectifs. La deuxième section, définit la façon avec laquelle COSMOS gère les informations de contexte. La troisième, détaille le nœud de contexte, qui est le concept clé de COSMOS, elle montre ses propriétés et son architecture. La dernière section décrit le modèle de composant Fractal sur lequel COSMOS se base.

3.1 Motivations et objectifs de COSMOS

Comme nous avons déja mentionné dans le chapitre précédent, les environnments ubiquitaires imposent des contraintes fortes sur la conception et le développement des applications. Ces applications doivent continuellement gérer le contexte dans lequel elles s'exécutent afin de détecter les situations d'adaptation [5]. Nous rappelons que le contexte est constitué de différentes catégories d'entités observables (ressources logicielles telles que les ressources système et les préférences des utilisateurs, et ressources matérielles telles que les capteurs), des rôles de ces entités dans le contexte et des relations entre ces entités.

La prise de décision pour déclancher une adaptation au contexte est un problème complexe pour lequel peu de solutions existent. Cette décision repose sur une collecte, une analyse et une synthèse des nombreux paramètres physiques et logiques fournis par le contexte d'exécution [13, 6, 5]. Pour cela, COSMOS a été proposé pour la gestion des informations de contexte. Il s'agit d'un canevas logiciel orienté composant pour la gestion d'informations dans des applications sensibles au contexte. Il permet de construire des gestionnaires de contexte en environnements ubiquitaires.

Cette gestion de contexte ne peut pas être transparente aux applications, au risque d'être contre-productive pour certaines d'entre elles en ne prenant pas en compte leur sémantique. L'application et l'intergiciel doivent donc s'organiser [10]. Pour organiser cette organisation, le paradigme composant/conteneur est bien adapté. L'intergiciel surveille les ressources et signale au composant de l'application tout changement significatif via le conteneur. Le composant de l'application, quant à lui, fournit les politiques de gestion de contexte par

l'intermédiaire de son conteneur.

COSMOS est développé selon trois principes, à savoir : la séparation entre les activités de collecte et de synthèse des données de contexte, l'organisation des politiques de gestion de contexte en assemblages de composants logiciels et enfin l'utilisation systématique de patrons de conception. L'originalité de COSMOS est l'expression de la composition de contexte dans un langage de définition d'architecture logicielle. Ainsi, les objectifs de COSMOS sont clairement identifiés :

- Composer des informations de contexte de manière déclarative grâce un langage dédié au domaine. Ceci doit faciliter la conception par composition, l'adaptation et la réutilisation des politiques de gestion de contexte ;
- Isoler chaque couche de l'architecture de gestion de contexte des autres couches afin de promouvoir la séparation des préoccupations et des cycles de vie des informations de contexte ;
- Fournir les concepts « système »pour gérer finement les ressources consommées par les différents traitements.

COSMOS est utilisé pour adapter les applications. Il est conçu de façon à être lui-même adaptable selon le besoin. Il facilite la conception, la composition, l'adaptation et la réutilisation des politiques de gestion de contexte. La puissance de COSMOS lui permet d'être appliqué même pour la composition d'informations de contexte de ressources système (processeur, mémoire, réseau, etc.).

3.2 Gestion de contexte

Dans cette section nous présentons l'architecture de gestion de contexte et nous introduisant par la suite le concept de Composition d'informations de contexte avec COSMOS.

3.2.1 Architecture du gestionnaire de contexte

L'architecture globale du gestionnaire de contexte inspirée de [5, 6, 13] présente 3 niveaux (comme le montre la figure 3.1). Les briques de base les plus proches du système d'exploitation sont la réification des ressources système, des capteurs à proximité du terminal et des préférences de l'utilisateur, et la distribution des informations de contexte dans le réseau de terminaux mobiles. Ces canevas logiciels fournissent les données de contexte à la base du traitement par ce qui est appelé processeur de contexte. La sensibilité au contexte de l'application, quant à elle, est gérée par les conteneurs des composants applicatifs. Dans la boite la plus haute.

En termes de fonctionnalités, la réification des ressources système permet de collecter des données brutes, souvent numériques, comme la qualité du lien réseau. Le processeur de contexte en déduit des informations de contexte de plus haut niveau, souvent des données symboliques comme le mode de connectivité (connecté, partiellement connecté ou déconnecté) et permet d'identifier des situations comme la perte prochaine de la connectivité réseau. Les politiques d'adaptation, quant à elles, sont généralement gérées en collaboration avec l'application au niveau du conteneur : par exemple, l'exploitation consiste à avertir l'utilisateur du mode de connectivité à l'aide d'un icône dédié.

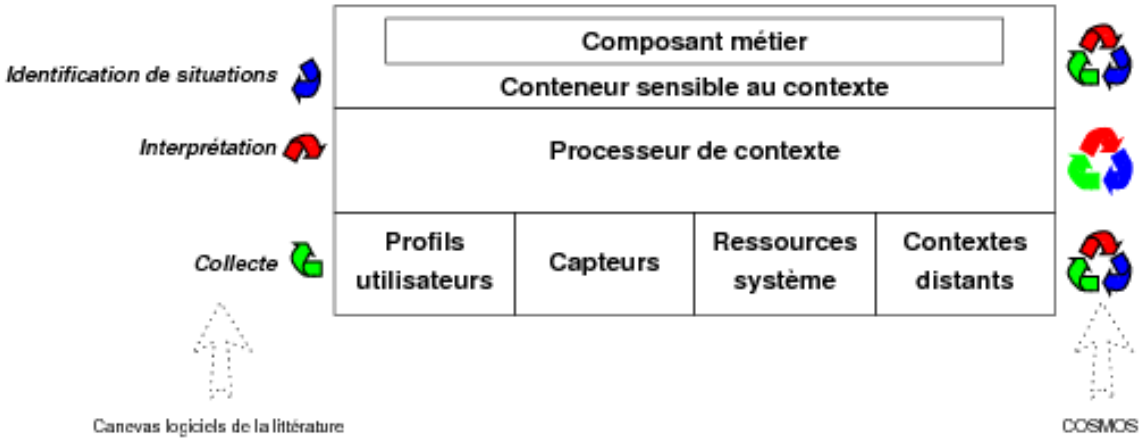

FIG. 3.1 – Architecture de gestion de contexte

COSMOS introduit des cycles indépendants « collecte / interprétation / identification de situations » dans chacune des couches de l'architecture (cf. partie droite de la figure 3.1). Donc, de tels cycles complets sont présents dans les collecteurs de contexte, plusieurs autres dans le processeur de contexte, et de même pour le conteneur sensible au contexte. En d'autres termes, cela revient à ajouter dans chaque couche où boite une notion de temps dans l'identification de situations.

Dans COSMOS, les différentes couches sont indépendantes quant à la gestion des ressources système (mémoire, activités) qu'elles consomment pour leurs traitements. Ainsi le système obtenu est faiblement couplé et facilement reconfigurable par les concepteurs de gestionnaires de contexte et d'applications sensibles au contexte.

3.2.2 Composition d'informations de contexte avec COSMOS

Cosmos s'appuie sur les principes de base de construction d'intergiciels : le canevas logiciel est construit à partir d'éléments génériques, spécialisables et modulaires afin de composer plutôt que de programmer. Il s'agit donc d'une approche à base de composants.

Cette approche apporte une vision unifiée dans laquelle les mêmes concepts (composant, liaison, interface) sont utilisés pour développer les applications et les différentes couches intergicielles et système sous-jacentes. Cette vision unifiée en facilite également la conception et l'évolution. Elle autorise également une vision hiérarchique dans laquelle l'ensemble « canevas et application » peut être vu à différents niveaux de granularité.

Par ailleurs, la notion d'architecture logicielle associée à l'approche orientée composant permet d'exprimer la composition des entités logicielles indépendamment de leurs implantations, rendant ainsi plus aisée la compréhension de l'ensemble. La notion d'architecture logicielle favorise aussi la dynamicité en autorisant la redéfinition des liaisons de tout ou partie du canevas, voire de l'application à l'exécution. La reconfiguration et l'adaptation à des contextes nouveaux non prévus au départ en sont facilitées.

3.3 Nœud de contexte

Dans cette section, nous détaillons le nœud de contexte qui est le concept de base de COSMOS.

3.3.1 Le concept nœud de contexte

Un nœud de contexte est une information de contexte modélisée par un composant. Les nœuds de contexte sont organisés en une hiérarchie avec possibilité de partage. Tous les composants de la hiérarchie sont potentiellement accessibles par les clients de COSMOS. Ce sont tous des composants composites. Le concept de nœud de contexte est le concept structurant du canevas logiciel COSMOS :

Un nœud de contexte est représent' par un composant et la composition d'informations de contexte par un graphe (hiérarchie avec partage) de composants. Les relations entre les nœuds sont donc des relations d'encapsulation. Le graphe représente l'ensemble des politiques de gestion de contexte utilisées par les applications clientes du gestionnaire de contexte.

Le partage de nœuds de contexte correspond à la possibilité de partage ou d'utilisation d'une partie d'une politique de gestion de contexte par plusieurs politiques. Les nœuds de contexte feuilles de la hiérarchie encapsulent les informations de contexte élémentaires, par exemple les ressources système du terminal (mémoire vive, qualité du lien WiFi, etc.). Leur rôle est d'isoler les inférences de contexte de plus haut niveau, qui deviennent donc indépendantes du canevas logiciel utilise pour la collecte des données brutes.

3.3.2 Propriétés d'un nœud de contexte

Passif ou actif Chaque nœud peut être passif ou actif avec exécution périodique de tâches dans des activités. Un nœud passif est un nœud de traitement utilisé par des activités extérieures au nœud qui l'interrogent pour obtenir

une information. Un nœud actif peut à contrario initier un parcours du graphe. Le cas d'utilisation le plus fréquent des nœuds actifs est la centralisation de plusieurs types d'informations et la mise à disposition de ces informations afin d'isoler une partie du graphe d'accès multiples trop fréquents.

Observation ou notification Les rapports d'observation contenant les informations de contexte circulent du bas vers le haut de la hiérarchie dans des messages (dont les constituants élémentaires sont typés). Lorsque la circulation s'effectue à la demande d'un nœud parent ou d'un client, c'est une observation (en anglais pull) ; dans le cas contraire, c'est une notification (en anglais push).

Passant ou bloquant Lors d'une observation ou d'une notification, le composant qui traite la requête peut être passant ou bloquant. Lors d'une observation, un nœud de contexte passant demande d'abord un nouveau rapport d'observation à ses enfants, puis calcule un rapport d'observation pour le transmettre (en retour) vers le haut de la hiérarchie.

Lors d'une notification, un nœud de contexte passant calcule un nouveau rapport d'observation avec la nouvelle notification, puis passe vers le haut ce rapport en notifiant ses parents. Dans le cas bloquant, le nœud observé fournit l'information de contexte qu'il détient sans observer les nœuds enfants, et le nœud notifié modifie son état interne sans notifier les nœuds parents.

Enfin, lorsque les informations de contexte des nœuds enfants ne peuvent pas être collectées, par exemple parce que la ressource système « interface réseau WiFi »n'existe pas, une exception est remontée vers les nœuds parents.

La remontée de cette exception peut bien sur être bloquée par un nœud de contexte donné pour masquer l'indisponibilité de l'information de contexte aux nœuds parents.

Opérateur Un nœud de contexte récupère des informations de contexte de nœuds enfants de la hiérarchie et infère une information de plus haut niveau d'abstraction en appliquant un opérateur. COSMOS propose des opérateurs génériques classés selon la typologie de [9] : opérateurs élémentaires pour la collecte, opérateurs à mémoire comme le calcul de la moyenne, de traduction de format, de fusion de données avec différentes qualités, d'abstraction ou d'inférence comme « l'additionneur » , et opérateurs a seuil comme un détecteur de connectivité [12] ou un évaluateur de profil « énergie, durée prévisible de connexion, espace mémoire » [1] .

Il est à noter que, dans une architecture de gestion de contexte classique, les premiers opérateurs élémentaires pour la collecte feraient partie de la couche « collecte » et la plupart des autres opérateurs de la couche « interprétation » tandis que les derniers opérateurs à seuil seraient dans la couche « identification de situations » . Dans COSMOS, ils peuvent être utilisés dans toutes les couches.

Cycle de vie et gestion des ressources Tous les nœuds de contexte de la hiérarchie sont gérés finement, tant au niveau de leur cycle de vie qu'au niveau de la gestion des ressources qu'ils consomment. Le cycle de vie des nœuds de contexte enfants est contrôlé par les nœuds de contexte parents. Pour la gestion des taches, les nœuds de contexte actifs enregistrent leurs taches auprès d'un gestionnaire d'activités. Ainsi, le gestionnaire d'activités, lui-même

paramétrable, peut créer une activité par tache (observation ou notification) ou bien une activité par nœud de contexte actif ou encore une activité pour tout ou partie de la hiérarchie. Pour la consommation d'espace mémoire, un gestionnaire de messages gère des réserves (en anglais, pools) de messages et autorise aussi bien les duplications « par référence » que « par valeur » .

Nommage Pour faciliter les parcours dans le graphe et les reconfigurations, les nœuds de contexte possèdent un nom. Puisque un gestionnaire de contexte construit avec COSMOS est local à un terminal, les noms sont locaux. Ces noms doivent aussi être uniques.

3.3.3 Architecture d'un nœud de contexte

Avant de présenter l'architecture d'un nœud de contexte, nous introduisons quelques concepts généraux et les notations graphiques utilisées pour l'approche composant.

Comme présenté dans la figure 3.2, un composant est une entité logicielle qui fournit et requiert des services. Ces services sont regroupés au sein d'interfaces. Il y a deux types d'interfaces, celle dites « serveur »qui fournissent des services, des celle dites « client »qui sont la spécification des services requis. Un composant possède un contenu. Celui-ci peut être composé d'un ensemble de sous-composants. Dans ce cas, le composant est dit composite. Au dernier niveau, les composants sont dits primitifs. Il est ainsi possible de construire des hiérarchies de composants offrant une vision avec différents

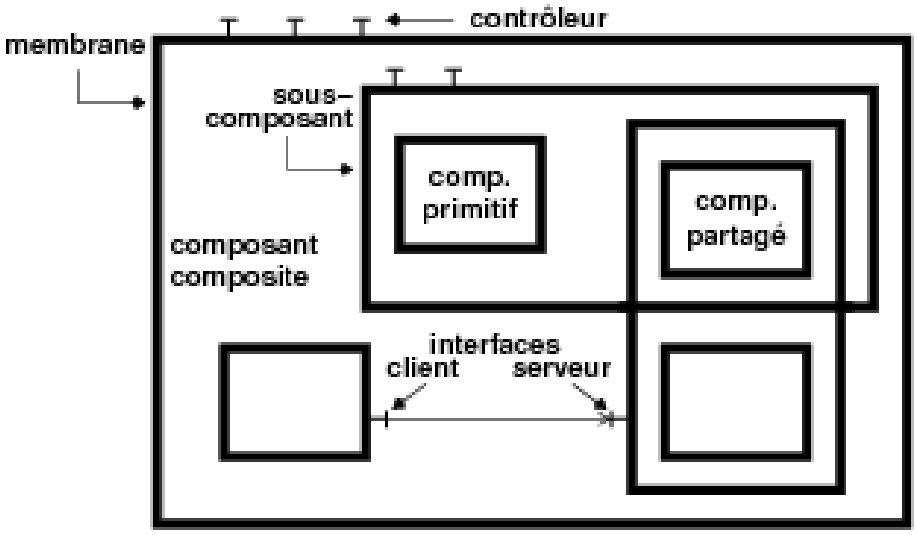

FIG. 3.2 – Modèle de composant Fractal

niveaux de granularité.

Les composants inclus dans plusieurs Composites sont dits partagés. Cette notion représente de façon naturelle le partage de ressources système (segments de mémoire, activités [en anglais, threads], etc.). Les composants sont assemblés à l'aide de liaisons. Une liaison représente un chemin de communication entre deux composants, plus précisément entre une interface requise (client) et une interface fournie (serveur) compatible. Enfin, les compositions de composants sont décrites avec un langage de description d'architecture (ADL pour Architecture Description Language).

Architecture d'un nœud de contexte Toutes les informations de contexte sont des composants étendant le composite abstrait ContextNode (cf. figure 3.3). Les interfaces Pull et Push sont respectivement, les interfaces pour l'observation et la notification. Les rapports d'observation sont des messages, constitués de sous-messages et de blocs (en anglais, chunks) types. Dans COSMOS,

toutes les informations élémentaires des rapports d'observation des entités observables liées aux collecteurs donnent lieu à un bloc type : par exemple, la qualité du lien réseau WiFi est mémorisée dans un bloc de type LinkQualityChunk.

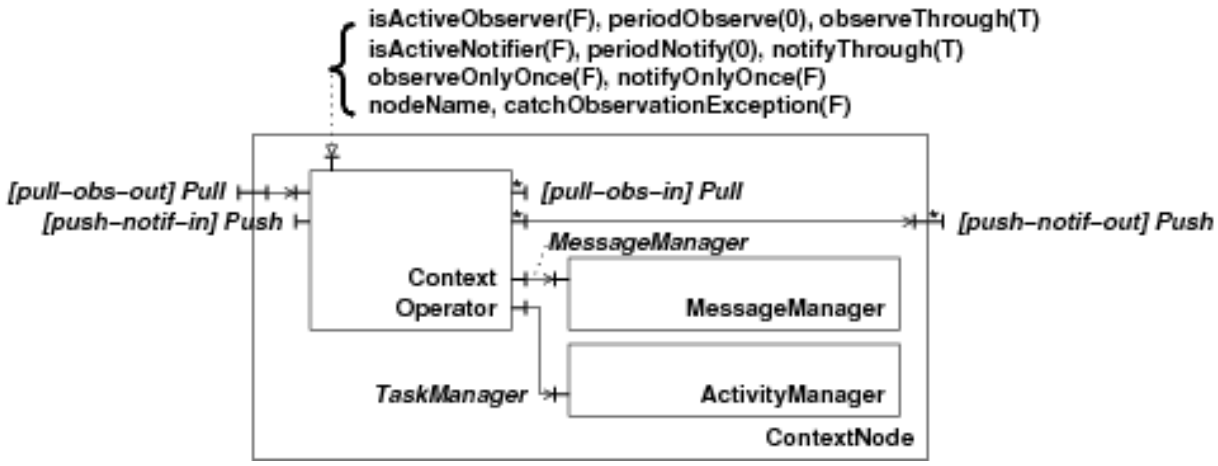

FIG. 3.3 – Modèle de composant Fractal

Les possibilités de configuration des nœuds de contexte sont exprimées par des attributs au niveau du composant primitif ContextOperator. Ainsi, le composant ContextNode est spécialisables via des attributs pour la gestion des taches d'observation et de notification, et pour la propriété passant/bloquant. Par défaut, les nœuds sont passifs et passants pour l'observation et la notification.

Les nœuds de la hiérarchie se classent ensuite en deux catégories : les feuilles et les autres. Pour les feuilles, le ContextNode est étendu pour contenir, en plus de l'opérateur et des gestionnaires d'activités et de messages, un composant primitif qui encapsule l'accès à la couche du dessous dans l'architecture en couches du gestionnaire de contexte. Cette couche peut être le

système d'exploitation ou un autre canevas logiciel construit ou non à l'aide de COSMOS.

3.4 Modèle de composant Fractal

Fractal [3]est un modèle de composant du consortium ObjectWeb pour le domaine des intergiciels en logiciel libre. C'est un modèle de composant hiérarchique, indépendant des langages de programmation et extensible. Contrairement aux modèles de composant EJB, CCM ou COM+/.NET, le modèle de composant Fractal ne suppose pas que les services techniques fournis aux composants soient figés. Tout composant Fractal est sous la responsabilité d'une membrane. Celle-ci est composée d'un ensemble de contrôleurs. Chaque contrôleur fournit un service technique élémentaire. Il existe ainsi de façon standard des contrôleurs pour gérer les liaisons, le cycle de vie ou les attributs des composants. De nouveaux contrôleurs et de nouvelles membranes peuvent être développés. Les composants sont alors adaptables à des domaines ou des environnements dans lesquels les services techniques varient. Plusieurs implantations du modèle de composant Fractal existent dans différents langages de programmation : Java, C, C++... Le modèle de composant Fractal est associé au langage de description d'architecture appelé Fractal ADL. Basé sur une syntaxe XML, ce langage permet d'exprimer des assemblages de composants Fractal. La définition du type du document (DTD pour Document Type Definition) et la chaîne de traitement de ce langage peuvent être étendues.

Plusieurs bibliothèques de composants Fractal existent. COSMOS utilise particulièrement la bibliothèque Dream [8]. Celle-ci permet de construire des systèmes orientés message et de gérer de façon fine des activités concur-

rentes organisées en réserves (pool). Dream est un système faiblement couplé, dynamique et reconfigurable. Les composants de contexte ContextNode actifs enregistrent leurs taches auprès d'un gestionnaire d'activités Dream (ActivityManager) via un contrôleur, un second contrôleur servant alors à appeler périodiquement les taches (observation ou notification). Le gestionnaire d'activités est chargé de faire la correspondance entre les taches définies par les composants et les unités concrètes d'exécution fournies par le système (typiquement des activités). Cette correspondance peut être mise en œuvre de différentes façons, par exemple en attribuant des priorités différentes aux tâches ou en exécutant les tâches à l'aide d'une réserve d'activités. En outre, le gestionnaire de messages Dream (MessageManager) est aussi utilisé.

3.5 Conclusion

Ce chapitre nous a permis de comprendre le composant COSMOS. Ce composant représente le collecteur d'informations de contexte qui permet à l'application d'être sensible au contexte. Ce chapitre nous a aussi permis d'assimiler la façon avec laquelle COSMOS compose l'information de contexte. Cette information de contexte est parmi les éléménts clés de notre projet.

Chapitre 4

Mise en œuvre

Nous rappelons que l'objectif principal de notre travail, est de rendre un composant donné sensible au contexte. Cette sensibilité au contexte sera introduite depuis la phase de modélisation. Dès le début, nous définissons les éléments observables à collecter, les situations d'adaptations à identifier et enfin les réactions d'adaptations à mettre en œuvre. Il s'agit d'écrire le code permettant de réaliser cette sensibilité au contexte et de l'intégrer dans la membrane d'un composant pour qu'il devienne sensible au contexte.

Dans ce chapitre, nous détaillons le processus de développement que nous avons adopté afin de mettre en œuvre ce travail. Nous commençons dans la première section, par décrire les deux méthodes proposées afin d'atteindre les objectifs déja fixés par notre projet. La deuxième section cite les principaux outils permettant la réalisation de notre travail. La troisième section présente le méta-modèle sur lequel nous nous sommes basés. La quatrième section détaille les étapes par lesquelles nous sommes passés pour implémenter la première méthode.

4.1 Description des méthodes proposées

Pour répondre aux besoins de notre projet, nous proposons principalement deux modes de fonctionnement. Le point de départ de notre travail, est d'éditer un modèle en utilisant les APIs générées par le méta-modèle. Ce modèle représente toutes les entités observables, les éléments observables, les collecteurs d'informations de contexte ainsi que les situations d'adaptation (cf. section suivante).

Le premier mode de fonctionnement proposé, utilise le modèle déja édité d'une façon dynamique pendant l'initialisation du composant. Il s'agit de parcourir le modèle d'une manière générique et d'observer tous les changements qui peuvent avoir lieu. Cette méthode offre la possibilité de modifier le modèle pendant l'exécution pour une meilleur adaptation. Elle peut être utilisée dans des environnements complexes et très variants.

Le deuxième mode de fonctionnement, utilise le modèle d'une façon statique lors de la compilation pour la génération du code d'interaction. Le code généré avec cette méthode est spécifique à l'application. Il s'agit donc d'un code réduit transportable sur petit matériel.

4.2 Présentation des outils de développement utilisés

Dans cette section, nous citons les différents outils utilisés lors de l'élaboration de ce travail. Les trois premiers outils (EMF, MAVEN, SVN) concernent

simultanément les deux modes de fonctionnements décrits dans la section précédente. Le dernier outil (JET) concerne seulement le deuxième mode de fonctionnement.

4.2.1 Eclipse Modeling Framework (EMF)

EMF est un outils logiciel sous Eclipse qui traite des modèles : cela peut s'entendre ici sous le sens que EMF offre à ses utilisateurs un cadre de travail pour la manipulation des modèles. Il permet de stocker les modèles sous forme de fichier pour en assurer la persistance. EMF permet aussi de traiter différents types de fichiers : conformes à des standards reconnus (XML, XMI) et aussi sous des formes spécifiques (code Java) ou avec des diagrammes UML.

Comme le montre la figure 4.1, l'objectif général de EMF est de proposer un outillage qui permet de passer du modèle au code Java automatiquement. Pour cela, EMF s'articule autour d'un modèle ECORE (Core Model).

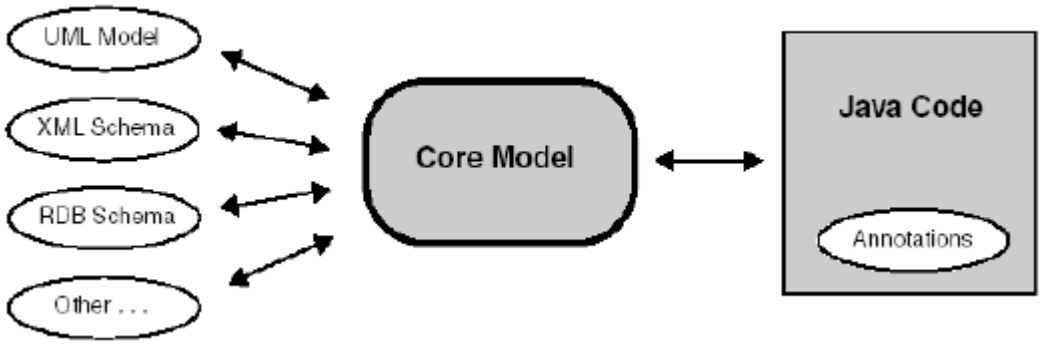

FIG. 4.1 – Organisation générale d'EMF

Cet outil propose plusieurs services :

- La transformation des modèles d'entrée (à gauche sur la figure 1), présentés sous diverses formes, en Core Model,
- La gestion de la persistance du Core Model,
- La transformation du Core Model en code Java.

EMF peut de base gérer en entrée des modèles présentés sous trois formats : UML, XMI et code Java Annoté. Nous avons utilisé dans notre travail directement le Core Model. Il s'agit là, d'éditer un méta-modèle Ecore, et par la suite, générer les APIs JAVA. Ces APIs nous permettent d'éditer des modèles conformes au méta-modèle d'une part, et de manipuler les modèles construits d'autres part.

4.2.2 Maven

Maven est un outil « open-source »pour la gestion et l'automatisation de production des projets logiciels Java. Il nous permet de produire un logiciel à partir de ses sources, en optimisant les taches réalisées à cette fin et en garantissant le bon ordre de fabrication. Maven utilise une approche déclarative, où la structure et le contenu du projet sont décrits, plutôt qu'une approche par tâche utilisée par exemple par les fichiers « make »traditionnels. Cela aide à mettre en place des standards de développements et réduit le temps nécessaire pour écrire et maintenir les scripts de « build ».

Cet outil utilise un paradigme connu sous le nom de POM (**P**roject **O**bject **M**odel) afin de décrire un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de taches prédéfinies, comme la compilation du code Java ou encore

sa modularisation.

Maven nous permet de gérer les dépendances entre les modules constituant notre projet ainsi que les bibliothèques dont il dépend. Il nous offre aussi la possibilité d'automatiser les différentes tâches et ainsi nous aurons la possibilité de générer automatiquement des tests unitaires. C'est donc un outil très riche qui facilite le développement des projets.

4.2.3 Subversion (SVN)

SVN est un logiciel de gestion de sources et de contrôle de versions. Il permet à des utilisateurs distincts et souvent distants de travailler ensemble sur les mêmes fichiers. Il s'appuie sur le principe d'un dépôt centralisé et unique dans lequel les utilisateurs peuvent déposer des données, les récupérer ou publier leurs modifications. SVN garde un historique des différentes versions des fichiers d'un projet, il permet aussi le retour à une version antérieure quelconque. Cet outil nous offre une meilleure gestion du travail en collaboration.

4.2.4 JAVA Emitter Template (JET)

JET est un outil pour la génération de code JAVA au sein d'EMF à partir de fichier modèle en utilisant des templates. Ces templates sont constitués par du texte et des commandes JET permettant d'extraire des informations à partir du modèle d'entrée.

JET contient principalement quatre librairies de commandes à savoir :

– Commandes de Contrôle : Utilisées pour accéder au modèle d'entrée et pour contrôler l'exécution du template.

– Commandes de format : Utilisées pour modifier le format du texte dans les modèles selon certaines règles.
– Commandes Java : Ce sont des balises utiles pour générer du code Java.
– Commandes de Workspace : Utilisée pour la création de ressources dans l'espace de travail, telles que des fichiers, des dossiers et des projets.

Comme le montre la figure 4.2,la génération du code se fait à partir du fichier modèle, JET Builder permet d'obtenir les paquetages et les classes java correspondantes. Cette génération utilise deux classes :

– JETCompiler : C'est la classe de base pour la traduction des modèles
– JETEmitter : La méthode generate() de cette classe combine la traduction des modèles et la génération dans une seule étape.

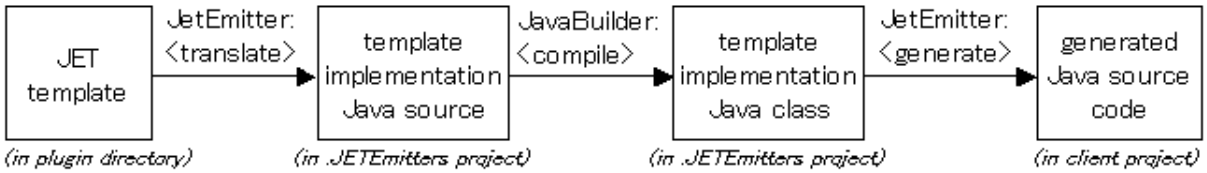

FIG. 4.2 – Les étapes de la génération avec JET

Donc cet outil nous donne la possibilité de générer automatiquement le code spécifique à un composant sensible au contexte

4.3 Présentation du Méta-modèle de sensibilité au contexte

Dans cette section, nous présentons le méta modèle sur lequel nous nous sommes basés pour élaborer ce travail. Ce méta-modèle a comme principal objectif d'automatiser la sensibilité au contexte des applications et la rendre déclarative plutôt que programmative. De plus, l'aspect sensibilité au contexte est pris en charge par l'intergiciel qui fait la liaison avec les collecteurs de contexte d'une part et interagit avec l'application d'autre part. Parmis les adaptations de l'application prises en charge par l'intergiciel, nous pouvons citer à titre d'exemple : la mise à jour de différentes variables, une modification du comportement de l'application, ou encore une modification de l'assemblage. Le méta-modèle définit la structure des modèles de sensibilité au contexte des applications ubiquitaires qui sont conformes au méta-modèle.

La figure 4.3, montre une vue d'ensemble de notre méta-modèle. Certaines parties du méta-modèle sont partagées par toutes les applications ubiquitaires, à savoir les deux paquetages ContextView et CollectorView. D'autres parties sont à définir par l'application sensible au contexte, à savoir le paquetage ContextAwarenessView.

D'autre part, l'intergiciel interagit avec le modèle afin de déterminer les informations de contexte nécessaires à l'adaptation. Parmi les principales interactions, nous pouvons citer par exemple, la détermination des différents éléments observables associés au modèle. Ces observables nous permettent de récupérer les informations de contexte. Ajoutons à cela, l'enregistrement du

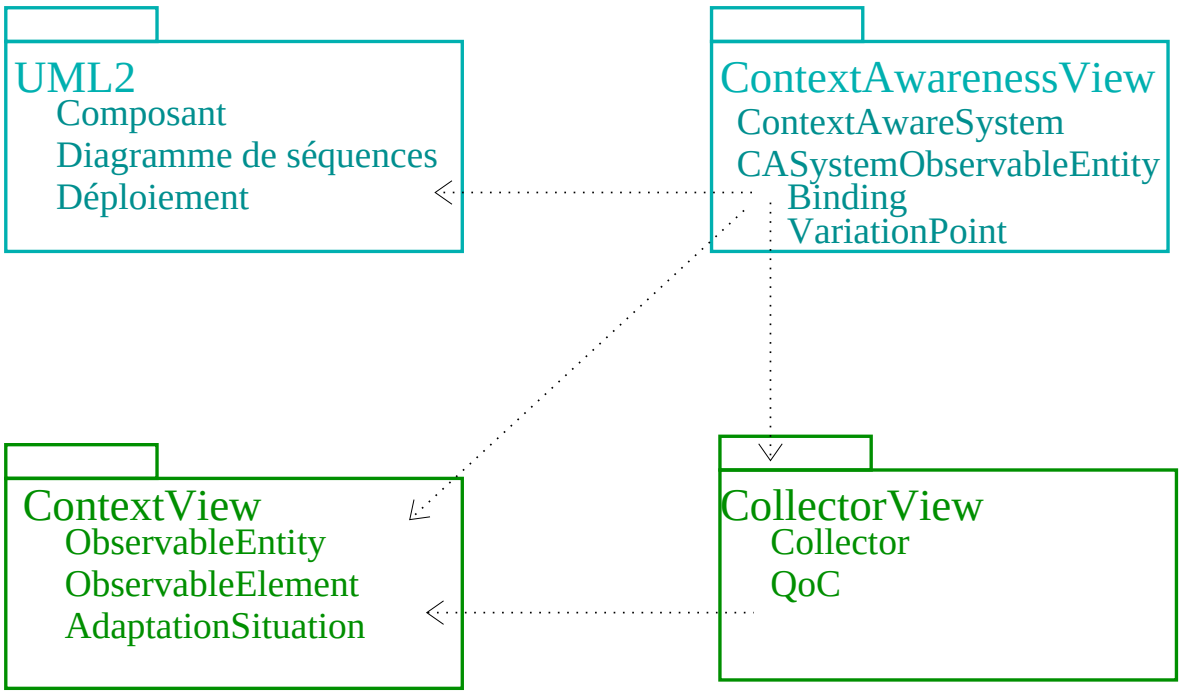

FIG. 4.3 – Les paquetages du méta-modèle

composant à un nœud de contexte (il s'agit du collecteur COSMOS) pour recevoir les notifications concernant les différentes variations qui peuvent arriver dans le système. L'application peut aussi observer les nœuds de contexte, si elle a besoin de connaitre une information de contexte à un moment donné. La figure 4.4 illustre les interactions entre le modèle et l'intergiciel.

Maintenant, nous détaillons les différents paquetages construisant notre méta-modèle. Le paquetage ContextView comprend les modèles conformes au méta-modèle des domaines indépendant de l'application. Nous pouvons citer comme principales classes :

- ObservableEntity : des exemples de cette classe sont computer, user, networkinterface, etc.
- ObservableElement : nous citons par exemple computerIdentification,

username, bandwidth

– Observation : cette classe manipule des valeurs datées
– AdaptationSituation : connecté au réseau

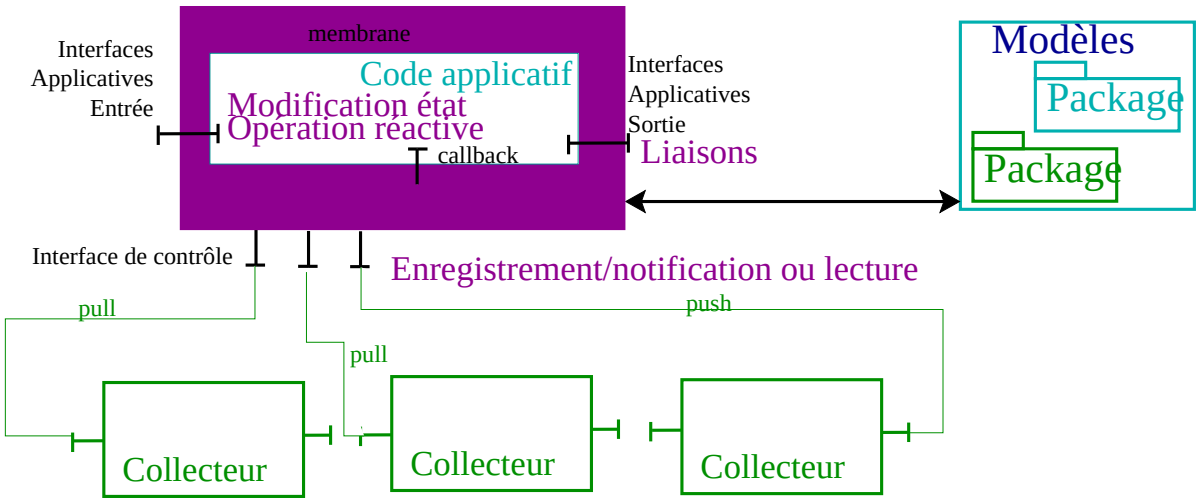

FIG. 4.4 – Interactions modèle-intergiciel

Notre méta-modèle se différencie par les entités du modèle mais aussi par les relations exprimées entre ces différents éléments. Les relations entre les différents éléments du modèle sont essentielles pour parcourir les différents éléments du modèle. Elles sont déterminantes pour l'utilisabilité du méta-modèle.

Le paquetage CollectorView définit des possibilités d'interactions entre les applications et les observables via les collecteurs. Il définit aussi les collecteurs disponibles pour un même type d'observable, chaque collecteur présente des unités différentes, des interfaces différentes, des qualités de contextes QoC

différentes. Pendant l'exécution de l'application, ce paquetage définit comment retrouver ou découvrir un collecteur déjà existant, il s'agit donc d'un premier mode de fonctionnement. Le deuxième mode consiste à instancier un nouveau collecteur pour collecter les informations de contexte. Ce paquetage définit aussi le type d'interaction entre l'application et le collecteur. Les deux types d'interaction mises en œuvre sont les pushs et les pulls qui ont été définis précédemment. Il est aussi à noter, que nous envisageons différents types de collecteurs. Nous distinguons donc les collecteurs COSMOS des autres collecteurs.

Le paquetage ContextAwarenessView définit des entités observables concrètes associées à l'application. Ce paquetage s'occupe de la sélection des observables parmi ceux définis pour ce type d'entité observable et pour chaque observable, il sélectionne le collecteur correspondant. Ce paquetage a aussi comme rôle de définir les interactions entre l'application et les observables. Il définit aussi les réactions à mettre en œuvre lors d'une détection d'une situation d'adaptation. Parmi les différentes réactions, nous pouvons citer à titre d'exemple un basculement vers le réseau wifi lorsque celui-ci devient disponible.

4.4 Description de la solution élaborée

Cette section s'intéresse essentiellement à l'élaboration de la première méthode du parcours dynamique du modèle de l'application, déjà présentée au début de ce chapitre. Nous détaillons dans ce qui suit les étapes par lesquelles nous sommes passés pour réaliser cette solution de l'édition du modèle de sensibilité au contexte jusqu'à l'écriture du code JAVA qui sera intégré dans la

membrane du composant Fractal.

4.4.1 Modèle conforme au métamodèle

Le modèle à éditer est une représentation de l'ensemble du système. Il ficilite la manipulation ainsi que l'observation de ce système. Pour éditer notre modèle, nous utilisons les classes générées à partir du méta-modèle. EMF nous permet de créer facilement un modèle conforme au méta-modèle.

Nous travaillons avec un ensemble d'entités observables. Chacune de ces entités dispose d'un ou plusieurs éléments observables et d'une situation d'adaptation. Il est à noter que chaque élément observable dispose de son propre collecteur de contexte. Pour chaque Collecteur, nous spécifions toutes les informations qui le caractérisent et particulièrement la famille à laquelle il appartient. Dans notre travail, nous utilisons des collecteurs de type COSMOS, mais il est possible d'ajouter d'autres types de collecteurs et de spécifier le traitement correspondant.

La figure 4.5 montre le modèle sur lequel nous nous sommes basés. Nous détaillons dans cette partie les différents éléments composant notre modèle afin de mieux le clarifier. Les entités observables que nous utilisons dans le modèle sont : l'entité « Computer » , qui dispose d'une entité « Printer » , il est utilisé par une entité « User » et appartient à une entité « NetworkInterface » . L'entité « ComputerPrinter » permet d'avoir une relation entre le « Computer » et le « Printer » .

Nous passons maintenant aux éléments observables : l'élémént « Color » est associé à l'entité « Printer » , il permet d'obtenir les couleurs utilisés par

cette imprimante. L'élément « Proximity » est lié à l'entité « ComputerPrinter » , il permet de trouver les imprimantes proches d'un ordinateur donné. L'élémént « UserAgent-Id » est associé à l'entité « Computer » , il permet d'avoir l'identifiant de l'utilisateur du PC. L'élémént « GeographicalLocation » est lié à la fois à « Computer » et « Printer » , il permet d'avoir la localisation géographique de l'ordinateur, et de l'imprimante. Cet élément est dérivé à partir d'un autre élémént « City » . Ce dernier est lié à l'entité « Computer » et permet de savoir dans quelle ville elle se trouve. Pour l'entité « NetworkInterface » , nous disposons d'un élément observable « Bandwidth » permettant d'avoir la bande passante du réseau. et enfin l'entité « User » dispose de 4 observables, à savoir : « Identity » , « Activities » , « Health » , « IdentityPapers » pour détérminer respectivement, l'identité de l'utilisateur, ses activités, sa santé et finalement ses papiers d'identité.

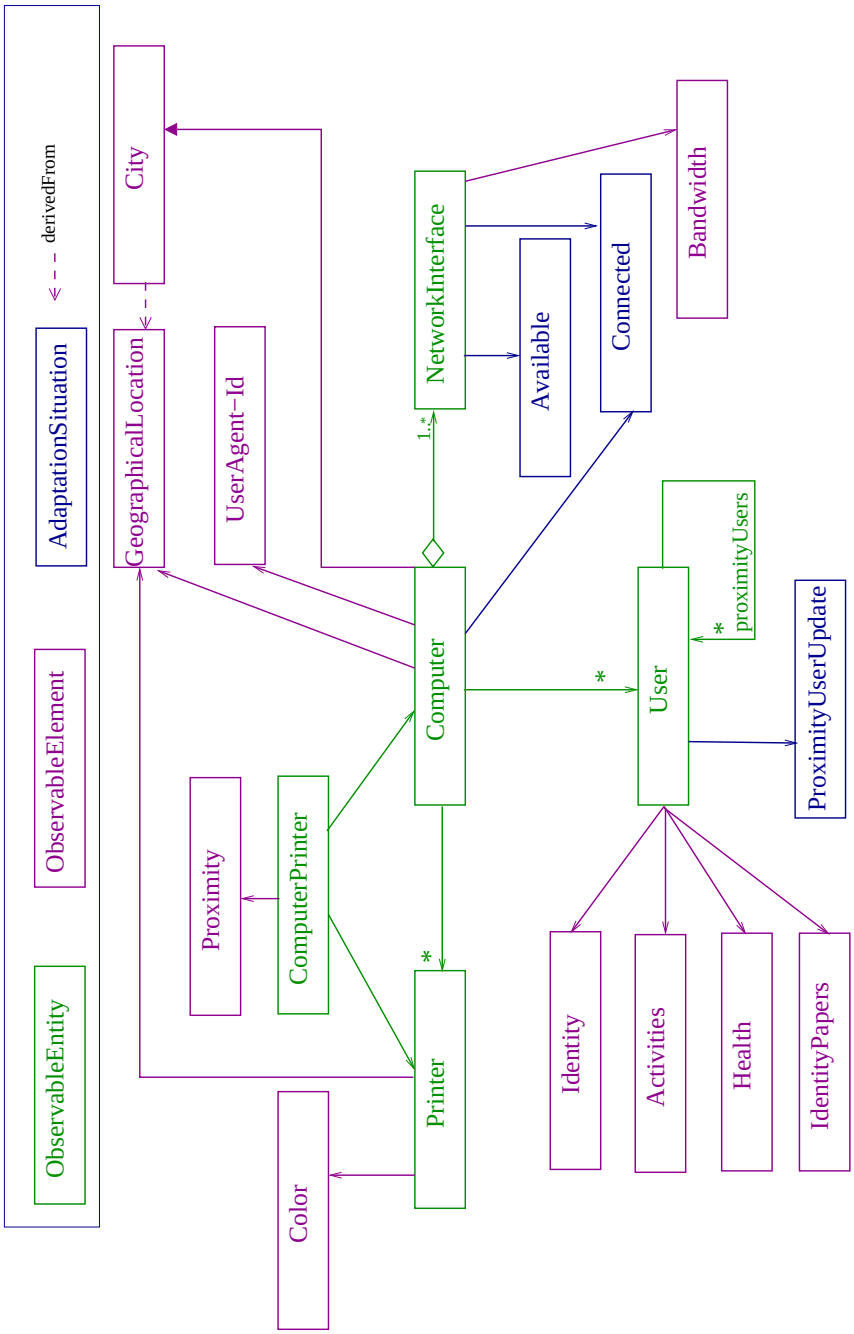

FIG. 4.5 – Entités et éléments observables de notre modèle

Il nous reste maintenant à détailler les situations d'adaptation. Notre modèle

en contient 3, qui sont : La situation « Available » relative à l'entité « NetworkInterface » , elle est utile pour savoir si un réseau est disponible ou non. La situation « Connected » relative à « Computer » , ainsi que « NetworkInterface » , pour savoir si ces entités sont connectés ou non. La dernière situation d'adaptation est « ProximityUserUpdate » , elle est relative à un utilisateur et consiste à mettre à jour la liste de ses utilisateurs voisins.

4.4.2 Diagramme de classes et détail des fonctionnalités

Cette sous section présente le diagramme de classes de la solution à implémenter. Nous avons prévu l'utilisation des deux méthodes « statique » et « dynamique » . Notre diagramme de classe est composé de 3 paquetages :

- eu.it-sudparis.cacomp.cacontroler
- eu.it-sudparis.cacomp.cacontroler.cosmos
- eu.it-sudparis.cacomp.cacontroler.tests

La figure 4.6 montre essentiellement les 2 premiers paquetages. Le premier paquetage contient l'ensemble des classes utilisées pour assurer les différentes fonctionnalités :

Dans le cas de la première méthode, nous commençons tout d'abord par charger le modèle. La classe ModelAccesor s'occupe du chargement du modèle et crée une ressource représentant le modèle à parcourir. Pour une meilleure souplesse, nous sauvegardons dans un fichier de propriétés, les principaux paramètres nécessaires à l'exécution de notre code, comme par exemple le chemin d'accès au modèle à parcourir, tout en prévoyant des valeurs par défaut si jamais le fichier n'existe pas. Ainsi ces paramètres seront chargés lors de l'exécution.

La classe CAControlerDynamicModel s'occupe du parcours du modèle proprement dit et crée les bridges collecteurs qui font l'interaction avec l'application à travers leurs interfaces. Le parcours consiste à retrouver les différentes entités observables, pour chaque entité, parcourir la liste des éléments observables. Comme nous avons déjà précisé, chaque élément observable dispose d'un collecteur. Lorsque nous arrivons au niveau du collecteur, un COSMOSBrdigeCollector sera créé si le collecteur trouvé est de type COSMOS. Dans le cas contraire, une exception est lancée. Pour le moment nous travaillons seulement avec les collecteurs de type COSMOS. Le COSMOSBridgeCollector créé aura comme but de collecter l'information de contexte. L'application peut utiliser ces collecteurs pour faire des observations (PULL) par exemple, ou encore recevoir des notifications(PUSH) des variations des éléments observables. Nous sauvegardons l'interface de ces collecteurs dans une « map » tout en leur attribuant une clé afin de les retrouver en cas de besoin.

Nous mettons l'accent sur le fait qu'il y a deux modes de création. Le premier mode est par découverte, les observables peuvent découvrir des collecteurs déja existant et les utiliser pour collecter les informations de contexte. Nous utilisons la classe DiscoveryInformation dans ce premier mode. Le deuxième mode est par instanciation, les observables peuvent instancier leurs propres collecteurs et les utiliser aussi pour la collecte d'informations. Dans ce deuxième cas, nous utilisons la classe InstantiationInformation.

Le deuxième paquetage est utilisé pour gérer les collecteur de type COSMOS. Nous faisons appel à la classe COSMOSBridgeCollector de ce paque-

tage lors de le création d'un collecteur COSMOS. Enfin, nous utilisons le troisième paquetage pour tester l'ensemble de notre application. Ce paquetage fait appel aux différentes classes pour charger un modèle, le parcourir et enfin créer les BridgeCollector qui collectent l'information de contextes.

4.4.3 Implémentation

Dans notre travail, nous écrivons du code en utilisant le langage JAVA. Nous manipulons aussi, des composants COSMOS lors de l'instanciation d'un collecteur donnée. Dans ce qui suit, nous montrons une partie de notre code JAVA. Il s'agit de la partie relative à la première méthode seulement (méthode dynamique). Cette partie s'occupe du parcourt un modèle après son chargement et crée les CosmosBridgeColecteurs pour collecter l'information de contexte.

Comme le montre la figure 4.7, dans le constructeur de la classe « CAControlerDynamicModel », nous faisons appel à la classe « ModelAccessor » (ligne 52) qui s'occupe du chargement du modèle dont le chemin d'acces est passé en paramètre. L'étape suivante consiste à appeler la méthode « createAllBridges » (ligne 53) afin de créer un « BridgeCollector » pour chaque élément observable.

Concernant la méthode « createAllBridges » , son rôle est de parcourir le modèle chargé. Elle détermine, ensuite, les éntités observables et pour chacune d'elle, elle parcourt les éléments observables correspondants. Pour chaque observable, elle crée le Bridge Collecteur en faisant appel à la méthode

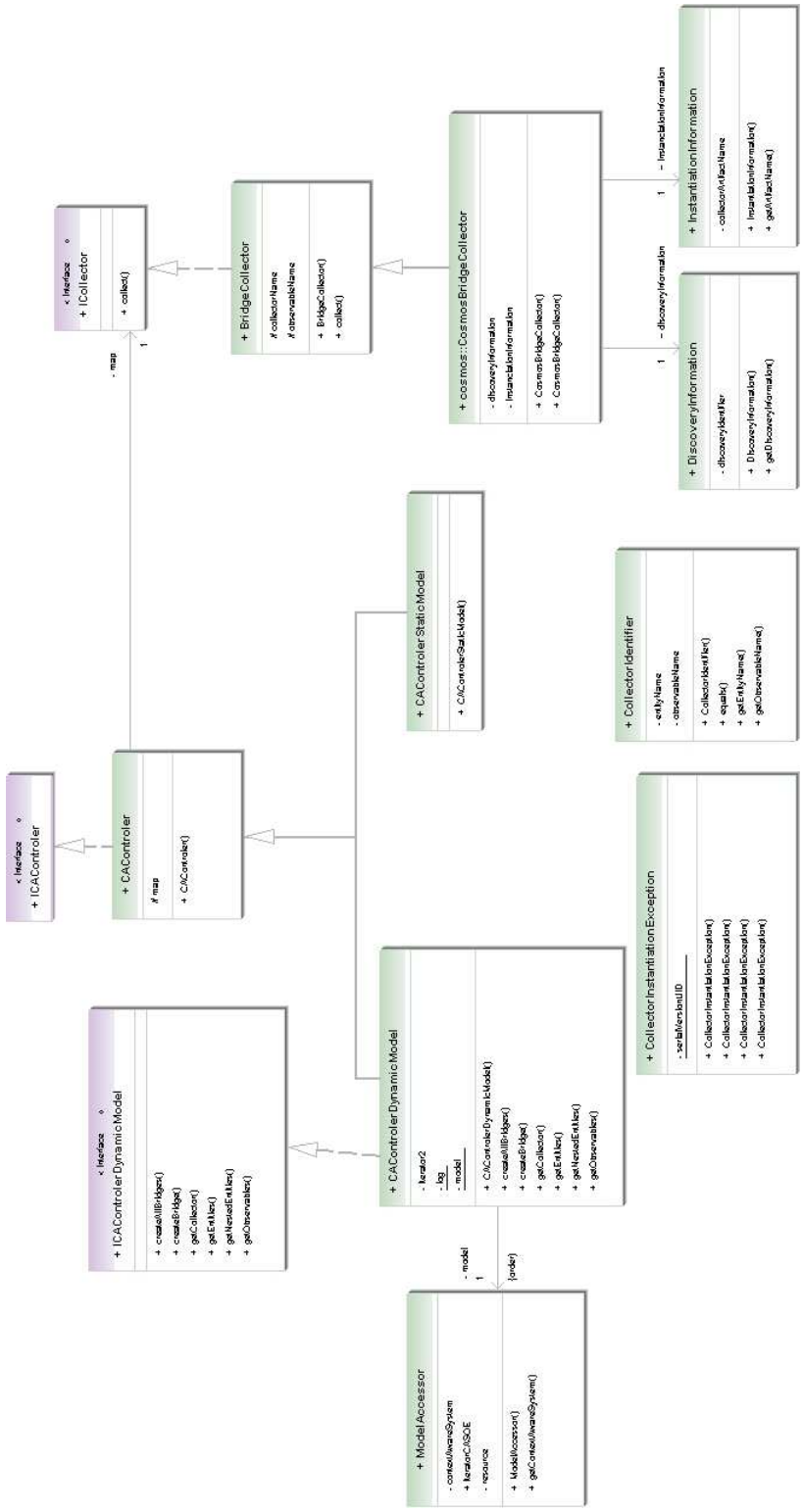

FIG. 4.6 – Diagramme de classe de la solution envisagée

« createBridge », puis l'insère dans une map de « Icollector » en lui associant une clé pour pouvoir le retrouver. La figure 4.8 détaille le code de cette méthode.

La méthode « createBridge » détermine le collecteur correspondant à l'élément observable passé en paramètre. Ensuite, elle teste le type de ce collecteur. Pour le moment, nous gérons des collecteurs COSMOS, mais il est possible d'étendre l'application et d'ajouter d'autres types de collecteurs. Pour ces collecteurs COSMOS, cette méthode vérifie si le mode utilisé est par découverte ou par instanciation pour savoir comment les créer. Le code de cette méthode est détaillé dans la figure 4.9.

4.4.4 Discussion

Dans ce chapitre, nous avons détaillé la démarche que nous avons suivie afin de réaliser ce travail. Nous nous sommes concentrés, en premier lieu, sur l'étude du méta-modèle de la sensibilité au contexte. Ensuite, nous avons édité un modèle conforme à ce méta-modèle. Ce modèle représente les éléments observables, les collecteurs des informations de contexte et les réactions d'adaptations de l'application à rendre sensible au contexte. Après l'édition du modèle, nous avons distingué deux modes d'utilisations ; le premier mode parcours le modèle lors de l'initialisation du composant pour obtenir les divers changements qui peuvent avoir lieu, il est utilisé dans des environnements ubiquitaires. Le deuxième mode utilise le modèle d'une manière statique pour générer un code JAVA propre à l'application, ce code est transportable sur petits matériels. Dans le cadre de notre travail, nous avons implémenté le premier mode seulement, mais nous avons prévu dans notre conception l'utilisation des deux modes, ainsi que les outils nécessaires à la réalisation. Le développement du deuxième mode nous permettra d'obtenir des résultats intéressant dans ce domaine.

```
1  package eu.it_sudparis.cacomp.cacontroler;
2  import java.util.Collection;
3  import java.util.Collections;
4  import java.util.Iterator;
5  import java.util.Properties;
6  import java.util.Set;
7  import org.eclipse.emf.common.util.EList;
8  import collectorView.Collector;
9  import collectorView.CollectorDiscoveryFactory;
10 import collectorView.CollectorFamilyValues;
11 import collectorView.CollectorInstantiationFactory;
12 import collectorView.impl.CollectorInstantiationFactoryImpl;
13 import contextAwareness.CAObservableElement;
14 import contextAwareness.CASystemObservableEntity;
15 import contextView.EntityRelation;
16 import contextView.ObservableElement;
17 import contextView.ObservableEntity;
18 import eu.it_sudparis.cacomp.cacontroler.cosmos.CosmosBridgeCollector;
19 import java.io.*;
20 import java.util.logging.*;
21 /**
22  * This class browse dynamically the given model.For each CAObservableEntity
23  * it get the CAObservableElements and create the correspondent collector
24  * and save it into a map
25  * @author Zaier Mehdi and Chantal Taconet
26  */
27 public class CAControlerDynamicModel extends
28                 eu.it_sudparis.cacomp.cacontroler.CAControler implements
29                 ICAControlerDynamicModel {
30         /**
31          * Java logger.
32         */
33        private static Logger log=Logger.getLogger("CAControlerDynamicModel");
34         private static ModelAccessor model;
35         Iterator<CASystemObservableEntity> iterator2;
36         /**
37          * Constructor of the CAControlerDynamicModel class
38          *
39          * @param path
40          */
41         public CAControlerDynamicModel(String path) {
42                 String newLevel = System.getProperty("level", "ALL");
43                 Level level = Level.parse(newLevel);
44                 try {
45                        Handler hand=new FileHandler("CAControler.log",false);
46                         log.addHandler(hand);
47                 } catch (IOException e) {
48                         // Impossible to open the log file
49                         log.warning("new FileHandler IOException: " + e);
50                 }
51                 log.setLevel(level);
52                 model = new ModelAccessor(path);
53                 createAllBridges();
54         }
```

FIG. 4.7 – Constructeur de la classe CAControlerDynamicModel

```
/**
* Browse all observable entities and get the correspondent observable
* elements
*/
public void createAllBridges() {
        ICollector iCollector = null;
        Iterator<CASystemObservableEntity> iterator;
        CASystemObservableEntity cASystemObservableEntity = null;
        iterator = model.iteratorCASOE;
        // browse all CASystemObservableEntities
        while (iterator.hasNext()) {
            cASystemObservableEntity = iterator.next();
            // Get all observable elements
            EList<CAObservableElement> cAObservableElements =
                        cASystemObservableEntity
                         .getObservableElementSubset();
            Iterator<CAObservableElement> iteratorOBE =
                        cAObservableElements.iterator();
            CAObservableElement cAObservableElement = null;
            // browse all cAObservableElements
            while (iteratorOBE.hasNext()) {
                 cAObservableElement = iteratorOBE.next();
                 System.out.println("");
                 log.info("The observable element is :"
                  + cAObservableElement.getObservableElement());
                 try {
                   // Create the iCollector
                   iCollector = createBridge(cAObservableElement);
                  } catch (CollectorInstantiationException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                  }
                  // creating a key (collectorIdentifier) for the collector
                  CollectorIdentifier collectorIdentifier =
                        new CollectorIdentifier(cASystemObservableEntity
                        .getName(),cAObservableElement.getObservableElement()
                        .getName());
                  // add an iCollector in the map
                  map.put(collectorIdentifier, iCollector);
                  log.info("The number of collectors in the map is "
                    + map.size());
                }
        }
}
```

FIG. 4.8 – Méthode createAllBridges()

```
/**
* this method create a collector and form the corresponding key and save
* the pairs (key,collector) into the map
* @param cAObservableElement
* @return ICollector
*/
public ICollector createBridge(CAObservableElement cAObservableElement)
                throws CollectorInstantiationException {
        // Get the collector
        Collector collector = cAObservableElement.getCollector();
        log.info("The collector of the observable element"
                        + cAObservableElement.getObservableElement().getName()
                        + " is :" + collector);
        switch (collector.getCollectorFamily().getValue()) {
        // Cosmos collector
        case CollectorFamilyValues.COSMOS_VALUE:
                if (collector.getCollectorFactory() instanceof
                CollectorInstantiationFactory) {
                log.info("Collector is on instanciation mode ");
                CollectorInstantiationFactory collectorInstantiationFactory=
                (CollectorInstantiationFactory) collector
                                .getCollectorFactory();
                String instantiationArtifact = collectorInstantiationFactory
                                .getInstantiationArtifact();
                InstantiationInformation instanciationInformation = new
                                 InstantiationInformation(instantiationArtifact);
                return new CosmosBridgeCollector(instanciationInformation);
                }
                else {
                if (collector.getCollectorFactory() instanceof
                CollectorDiscoveryFactory) {
                        log.info("Collector is on discovery mode ");
                        CollectorDiscoveryFactory collectorDiscoveryFactory=
                        (CollectorDiscoveryFactory) collector
                                        .getCollectorFactory();
                String discoveryIdentifier = collectorDiscoveryFactory
                                        .getDiscoveryIdentifier();
                DiscoveryInformation discoveryInformation = new
                        DiscoveryInformation(discoveryIdentifier);
                return new CosmosBridgeCollector(discoveryInformation);
                }
        }
        // not a cosmos collector
        default:
        throw new CollectorInstantiationException();
        }
}
```

FIG. 4.9 – Méthode createBridge()

Conclusion et Perspectives

La réalisation de notre travail concernant l'Ingénierie Dirigée par les Modèles et Composants Sensibles au Contexte, s'est étalée sur une période d'environ quatre mois et demi au Département Informatique de TELECOM et Management SudParis .

Au cours de cette période j'ai intégré un milieu de recherche, mais aussi de développement au sens vrai du terme. Cette intégration m'a été très bénéfique, dans la mesure où elle m'a permis d'acquérir une expérience tant sur le plan théorique que sur le plan pratique ; suite à l'acquisition et la maîtrise de nouvelles technologies et des nouveaux aspects dans le domaine de la sensibilité au contexte.

Notre travail s'est intéressé à étudier la sensibilité au contexte d'exécution et à proposer une approche générique pour rendre un composant donné auto-adaptable. L'intégration de cette sensibilité au contexte est réalisée depuis la modélisation de l'application. Dans cette phase de modélisation, nous commençons par définir tous les éléments à observer, les collecteurs permettant de les observer, ainsi que les réactions d'adaptations.

Nous avons bien précisé au début de notre rapport que nous envisageons d'implémenter deux modes de fonctionnement. Le premier est un mode dynamique qui parcourt d'une manière générique le modèle de l'application lors de l'initialisation du composant. Le parcours permet de collecter l'information de contexte permettant de réaliser l'adaptation. Le travail auquel nous avons abouti, permet de réaliser ces différentes tâches. Pour tester notre code,

nous avons utilisé un scénario d'un modèle. Toutefois, l'application à laquelle nous avons abouti reste ouverte à toute évolution dans la mesure où l'apport des nouvelles informations contribuera à son amélioration. Une telle tâche nécessite beaucoup plus de temps que celui qui nous a été accordé.

Revenons au deuxième mode de fonctionnement, nous envisageons d'utiliser statiquement le modèle de l'application pour générer le code de l'interaction. Nous rappelons que le code à générer est spécifique à l'application. Dans notre travail, nous n'avons pas encore développé ce deuxième mode, mais nous avons prévu de l'intégrer dans notre application. De plus, nous avons aussi trouvé l'outil qui permet de générer le code nécessaire à ce mode (Il s'agit de l'outil JET déjà détaillé dans le dernier chapitre). La mise en œuvre de ce deuxième mode, nous permettra de faire des comparaisons ainsi que des évaluations très intéressante sur ces deux modes. La poursuite de ce travail reste donc à entreprendre très prochainement à partir du mois prochain.

Enfin, nous sommes convaincus que la réalisation de ce travail est une expérience très fructueuse pour notre formation pratique et académique. Les connaissances théoriques que nous avons acquises tout au long de notre formation à la Faculté des Sciences de Tunis (FST) ont été parfaitement consolidées. Par la même occasion, ce stage constitue une très bonne expérience humaine et sociale à travers le contact avec les enseignants, les doctorants ainsi que les stagiaires de différentes nations.

Bibliographie

[1] M. Boulkenafed and V. Issarny. A middleware service for mobile ad hoc data sharing, enhancing data availability. in m. endler and d. schmidt, editors, proc. ifip/acm/usenix international middleware conference, volume 2672 of lecture notes in computer science. June 2003.

[2] P.J. Brown, J.D. Bovey, and X. Chen. Context-aware Applications : from the Laboratory to the Marketplace. *IEEE Personal Communications*, 4(5) :58–64, October 1997.

[3] É. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The Fractal Component Model and Its Support in Java. *special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11) :1257–1284, sep 2006.

[4] D. Conan, R. Rouvoy, and L. Seinturier. COSMOS : composition de nœuds de contexte, GNU Lesser General Public License. July 2007.

[5] Coutaz, J. and Crowley, J.L. and Dobson, S. and Garlan, D. The disappearing computer : Context is Key. *Communications of the ACM*, 48(3) :49–53, 2005.

[6] A. Dey, D. Salber, and G. Abowd. A conceptual framework and a TOOLKIT for supporting the rapid prototyping of context-aware applications, 2001.

[7] A.K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[8] V. Quéma M. Leclercq and J.-B. Stefani. Dream : a component framework for the construction of resource-aware, configurable moms. sep 2005.

[9] G. Rey and J. Coutaz. Le Contexteur : Capture et distribution dynamique d'information contextuelle. In Actes de la 1ère Conférence ACM Francophone Mobilité et Ubiquité, volume 64 of ACM International Conference Proceeding Series. pages 131–138, June 2004.

[10] M. Satyanarayanan. The Many Faces of Adaptation. *IEEE Pervasive Computing*, pages 4–5, July 2004.

[11] C. Taconet. État de l'art cappucino(construction et adaptation d'applications ubiquitaires et de composants d'intergiciels en environnement ouvert pour l'industrie du commerce), March 2007.

[12] L. Temal and D. Conan. Détections de défaillances, de connectivité et de déconnexions. in actes de la 1ère conférence acm francophone mobilité et ubiquité, volume 64 of acm international conference proceeding series. pages 90–97, June 2004.

[13] Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K. S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3) :33–40, 2002.