Institut National des Télécommunications

Université d'Évry Val d'Essonne

Thèse de doctorat de l'Institut National des Télécommunications dans le cadre de l'école doctorale SITEVRY en co-accréditation avec l'Université d'Évry Val d'Essonne

spécialité informatique

Par :

NABIHA BELHANAFI BEHLOULI

Thèse présentée pour l'obtention du grade de Docteur de l'Institut National des Télécommunications

Ajout de mécanismes de réactivité au contexte dans les intergiciels pour composants dans le cadre d'utilisateurs nomades

Soutenue le 27 Novembre 2006 devant le jury composé de :

Rapporteurs :

Mme. Yolande Berbers	Professeur à l'Université Catholique de Louvain
Mme. Laurence Duchien	Professeur au LIFL, Université de Lille

Examinateurs :

Mme. Hanna Klaudel	Professeur à l'Université d'Evry val d'Essonne
M. Jean-Yves Tigli	Maitre de conférence à l'ESSI & à l'Université de Nice - Sophia Antipolis
M. Bruno Traverson	Ingénieur-chercheur à EDF R&D
M. Guy Bernard	Professeur à l'INT (Directeur de thèse)
Mme. Chantal Taconet	Maître de conférence à l'INT (Encadrant)

Thèse numéro : 06INT010

Remerciements

Les trois années de recherche dont ce mémoire est l'aboutissement ont été riches d'enseignement. Ces travaux n'auraient pu arriver à leur terme sans les encouragements et les conseils de nombreuses personnes qui y ont été impliquées de près ou de loin. Je remercie en particulier :

Mme. Yolande Berbers, Professeur à l'Université Catholique de Louvain et Mme. Laurence Duchien, Professeur au LIFL, Université de Lille, pour l'intérêt qu'elles ont porté à mes travaux et pour l'honneur qu'elles m'ont fait d'être rapporteurs de ce mémoire.

Mme. Hanna Klaudel, Professeur à l'Université d'Evry et M. Jean-Yves Tigli, Maître de conférence à l'ESSI et à l'Université de Nice d'avoir accepté de faire partie du jury.

M. Guy Bernard, mon directeur de thèse de m'avoir accueilli dans le département Informatique et de m'avoir permis d'effectuer cette thèse dans de bonnes conditions.

Mme. Chantal Taconet, pour m'avoir encadré tout le long de cette thèse. Je la remercie pour ses conseils, sa disponibilité et ses encouragements qui m'ont été très profitables et qui m'ont permis d'avancer tout le long de cette thèse.

L'ensemble des membres du département informatique de l'INT et notre secrétaire Brigitte Houassine pour leur aide et leur soutien. Je pense particulièrement à Denis Conan, Samir Tata, Sophie Chabridon et Amel Bouzeghoub pour avoir patiement lu et corrigé ce document. Merci aussi à Djamel Belaid et Alda Lopez pour leurs encouragements ainsi qu'au thésards, les post-doctorants et les stagiaires de notre département pour leur aide amical et permanente.

Nabil Kouici, Dhouha Ayed, Lydialle Chateigner et Meriem Belguidoum pour leur amitié et les moments de distraction nécessaires lors du déroulement d'un tel projet.

Cette thèse n'aurait pu aboutir sans le soutien et les encouragements de mes parents, de mon mari, de mes soeurs et de mon frère. Une seul ligne ne saurait transcrire ma gratitude envers eux. Je les remercie pour leurs aides, leurs conseils et de m'avoir inculqué ce qui est important dans la vie. Sans eux cette thèse n'aurait pas été possible. Ce présent mémoire leurs est dédié.

Résumé

L'évolution technologique des dispositifs mobiles a donné naissance à de nouveaux besoins applicatifs pour assurer l'exécution des applications dans des environnements dynamiques. Ces applications appelées applications sensibles au contexte doivent détecter les variations de l'environnement et s'adapter en conséquence. Le développement de ce type d'applications est difficile à mettre en oeuvre et nécessite de grands efforts de programmation.

La majorité des travaux existants, concernant le contexte et l'adaptation, focalisent leurs efforts soit à proposer des modèles pour décrire le contexte (dans ce cas, les développeurs d'applications doivent implémenter la gestion du contexte et l'adaptation des applications), soit à proposer un intergiciel qui se charge de gérer certains aspects liés à la gestion du contexte sans pour autant offrir un modèle pour décrire le contexte.

Cette thèse s'intéresse à faciliter le développement des applications orientées composant sensibles au contexte. Pour cela, nous proposons l'intergiciel CAMidO (Context-Aware Middleware based on Ontology meta-model). Notre proposition fournie à la fois un méta-modèle pour décrire le contexte et un intergiciel orienté composant auquel nous avons ajouté des entités de gestion du contexte et d'adaptation de l'application. Le méta-modèle de CAMidO englobe la description des informations communes à toutes les applications sensibles au contexte ainsi que des informations spécifiques à chaque application. Cette description est utilisée par un compilateur pour permettre la gestion automatique du contexte et l'adaptation de l'application. Dans CAMidO, nous considérons que la granularité d'adaptation est le composant. Nous prenons en considération deux sortes d'adaptations : l'adaptation comportementale réactive et l'adaptation comportementale proactive. Nous utilisons le paradigme composant/conteneur pour gérer ces adaptations par des propriétés extra-fonctionnelles.

L'intergiciel CAMidO peut être exécuté en deux modes : le mode configuration statique et le mode reconfiguration dynamique. Le mode configuration statique ne prend en compte que les descriptions initiales du modèle de l'application, alors que le mode reconfiguration dynamique offre la possibilité à l'intergiciel de prendre en compte les modifications apportées sur le modèle de l'application après son exécution. Afin de valider ces propositions, nous avons implanté un prototype de CAMidO au dessus de la plate-forme OpenCCM, et nous avons effectuer des évaluations qualitatifs et quantitatifs de ce prototype.

Mots-clés : Sensibilité au contexte, modélisation du contexte, ontologies, adaptation, intergiciel, composant.

Abstract

The rapid evolution of mobile computing induced to new applicative needs to ensure the execution of the applications in dynamic environment. These applications called context-aware applications have to detect the environment changes and adapt their behavior accordingly. The development of such type of application is difficult to implement and requires main programming efforts.

Most of research works dealing with context and adaptation focus their efforts either in proposing context models for context description (application developer has to deal with context and adaptation), or in proposing platforms that interact with context and adapt the application to context changes without providing any meta-model for context description.

The goal of this work is to facilitate the development of component based context-aware applications. For this reason we proposed the CAMidO middleware (Context-aware Midleware based on Ontology meta-model). Our proposition provides both a meta-model for context description and a component based middleware which contains entities for context and adaptation management. The CAMidO meta-model contains description of common informations to all context-aware applications and application specific information. This description is used by a compiler to enable the automatic management of context and adaptation. In CAMidO, the adaptation granularity is the component. We consider two adaptation kinds : the reactive behavioral adaptation and the proactive behavioral adaptation. We used the component/container paradigm in order to manage these adaptation kinds using non functional properties. The CAMidO middleware can be executed in two modes : the static configuration mode and the dynamic reconfiguration mode. The static configuration mode considers only the initial descriptions of the application model, while the dynamic reconfiguration mode allows the middleware to consider all the application model changes carried out after its execution. In order to validate these propositions, we implemented a prototype of CAMidO on top of OpenCCM, and we have done qualitative and quantitative evaluations of this prototype.

Keywords : Context awareness, context modeling, ontology, adaptation, middleware, component.

Table des matières

Table des figures

Liste des tableaux

Introduction

Les progrès technologiques réalisés ces dernières années dans le domaine des réseaux sans fil, de l'informatique mobile, et de la miniaturisation ont rendu les terminaux tels que les assistants personnels numériques (PDAs) et les téléphones portables suffisamment puissants pour y installer des applications grand public ; ceci a permis un développement rapide et croissant de l'utilisation de ces terminaux dans la vie de tous les jours. L'environnement de ces terminaux est caractérisé par une variabilité des capacités matérielles (mémoire, bande passante, batterie...), de plus la mobilité de l'utilisateur introduit une variabilité de l'environnement qui entoure l'application. Par conséquent, les applications traditionnelles ne peuvent s'exécuter dans ce type d'environnement. Donc, il est nécessaire de considérer un nouveau type d'applications qui détectent les changements dans l'environnement et qui tirent partie de ces changements pour adapter leurs comportements en conséquence. Ce type d'applications est appelé `applications sensibles au contexte`.

Le développement d'une application sensible au contexte consiste à programmer toutes les étapes de gestion du contexte en commençant par la description des contextes qu'il faut observer, les contextes pertinents auxquels l'application est sensible et les situations pertinentes qui nécessitent l'adaptation de l'application. La collecte des informations de contexte nécessite de programmer l'interaction de l'application avec différents types de capteurs, alors que l'interprétation et l'analyse du contexte nécessitent de programmer un mécanisme qui étudie constamment les informations observées pour interpréter de nouvelles informations de contexte et détecter leurs variations pertinentes. De plus, un mécanisme d'adaptation doit être programmé pour permettre l'adaptation de l'application dès la détection des situations pertinentes auxquelles l'application est sensible.

Le développement des applications sensibles au contexte est une tâche complexe qu'il est nécessaire de simplifier. En effet, programmer les différentes étapes de gestion du contexte à partir de zéro et d'une manière ad-hoc est difficile à mettre en oeuvre. Cette simplification peut se traduire par l'utilisation d'un ensemble de techniques qui prennent en charge certaines tâches relatives à la sensibilité au contexte et aident ainsi les développeurs dans le processus de création de ce type d'applications. Faciliter le développement des applications sensibles au contexte à pour but de répandre le développement de ce type d'applications en le mettant à la portée du plus grand nombre de développeurs.

Dans le cadre de cette thèse, nous nous intéressons aux applications distribuées sensibles au contexte. Pour simplifier le développement de ce type d'applications il est nécessaire d'utiliser des

intergiciels qui se chargent de gérer la distribution des applications et de gérer certains aspects liées à la sensibilité au contexte.

Motivations

La majorité des travaux existants, concernant le contexte et l'adaptation focalisent leurs efforts soit à proposer des modèles pour décrire le contexte (dans ce cas, les développeurs d'applications doivent implémenter la gestion du contexte et l'adaptation des applications), soit à proposer un intergiciel qui se charge de gérer le contexte et l'adaptation de l'application sans pour autant offrir un modèle pour décrire le contexte.

Les approches de modélisation de contexte existantes (approches orientées applications, approches basées sur la logique, approches orientées modèles, approches orientées ontologie) se focalisent sur la description du contexte sans pour autant considérer que ces informations sont utilisées par des applications sensibles au contexte. De ce fait, aucune description de l'interprétation du contexte ni de l'adaptation de l'application n'est considérée. De leur côté, les intergiciels sensibles au contexte prennent en compte une partie des éléments fonctionnels des applications sensibles au contexte en laissant le reste à la charge du développeur d'application.

Afin de faciliter le développement des applications sensibles au contexte depuis la collecte jusqu'à l'adaptation, notre proposition (l'intergiciel CAMidO), fournit à la fois un méta-modèle qui permet aux concepteurs des applications sensibles au contexte la description du contexte et un intergiciel orienté composant auquel nous avons ajouté des entités de gestion du contexte et d'adaptation de l'application. Ces entités prennent en compte tous les éléments fonctionnels d'une application sensible au contexte.

Contributions

Les contributions apportées par ce travail de recherche ont pour objectif de faciliter pour les développeurs la création des applications orientées composant sensibles au contexte. L'intergiciel CAMidO (Context Aware Middleware based on Ontology meta-model) que nous avons proposé offre cette possibilité. Cet intergiciel prend en compte tous les éléments fonctionnels d'une application sensible au contexte, la seule tâche à la charge du développeur est celle de la description du contexte auquel son application est sensible ainsi que les adaptations. Nos contributions s'étendent de la modélisation du contexte à la mise en oeuvre d'un intergiciel pour la sensibilité au contexte, et se résument comme suit :

– le **méta-modèle de CAMidO** permet de définir des concepts généraux aux applications sensibles au contexte, ainsi que des concepts spécifiques à chaque application. Ce méta-modèle est structuré en trois ontologies qui peuvent être enrichies et réutilisées par les concepteurs d'application. Le but du méta-modèle de CAMidO n'est pas de proposer une nouvelle ontologie de contexte mais de s'appuyer sur des ontologies standards et de les

étendre et ceci, afin de décrire les concepts liés à l'utilisation du contexte par des applications orientées composants. Si nécessaire pour les besoins d'une application, le méta-modèle peut être étendu pour définir de nouveaux contextes observables, des nouvelles règles d'adaptation, de nouveaux capteurs, de nouveaux contextes pertinents, de nouvelles situations pertinentes et de nouvelles politiques d'adaptation.

- le **compilateur de CAMidO** se charge de compiler les informations fournies par le concepteur de l'application et de générer d'une part des fichiers de règles de logiques de premier ordre et d'autre part le code source d'adaptation. Les fichiers de règles sont utilisés pour automatiser l'interprétation du contexte et la détection des situations pertinentes. Le code source d'adaptation est intégré dans le conteneur de composant sous la forme de contrôleurs pour automatiser le processus d'adaptation.
- le **gestionnaire de contexte** libère le développeur de l'application de la programmation des tâches relatives à la gestion du contexte, telles que l'interaction avec les capteurs, l'interprétation du contexte de haut niveau et la détection des situations pertinentes. De ce fait, la tâche des développeurs se résume uniquement à fournir le modèle de sensibilité au contexte associé à l'application en se basant sur le méta-modèle de CAMidO. La valeur ajoutée du gestionnaire de contexte réside dans le fait qu'il sépare l'adaptation de l'application de la gestion du contexte auquel elle est sensible. Cette séparation étend l'utilisation de ce gestionnaire non seulement par les applications orientées composant mais aussi par des services ou des applications ayant leurs propres mécanismes d'adaptation.
- l'**intégration du gestionnaire d'adaptation dans les conteneurs des composants**, dans le but de séparer les préoccupations fonctionnelles des préoccupations extra-fonctionnelles de l'application. Le conteneur que nous proposons travaille en collaboration avec des intercepteurs portables et un mandataire (*proxy*) qui se chargent d'intercepter les invocations des composants et d'appliquer les adaptations appropriées.
- CAMidO fournit **deux modes d'exécution** : le mode reconfiguration dynamique et le mode configuration statique. Le mode reconfiguration dynamique permet à l'intergiciel CAMidO de prendre en compte les modifications apportées sur le modèle de l'application après son exécution, tandis que le mode configuration statique ne prend en compte que les descriptions initiales du contexte, tout changement du modèle associé à l'application nécessite l'arrêt de son exécution et son redéploiement.

Afin de valider nos contributions, nous avons implanté un prototype de CAMidO au dessus de la plate-forme OpenCCM [80] et nous avons effectué un ensemble d'évaluations qualitatives et quantitatives de ce prototype.

Plan du document

Ce document est organisé en deux parties et cinq chapitres. Dans la première partie nous présentons la problématique de développement d'applications sensibles au contexte et les solutions existantes.

- Le chapitre 1 fournit une étude sur les définitions du contexte et l'importance de son utilisation dans le domaine informatique. Puis, il décrit différentes approches de modélisation

du contexte. Cette étude permet de décrire les caractéristiques de chaque approche de modélisation du contexte, ses avantages et ses limites.

– Le chapitre 2 effectue une étude sur la sensibilité au contexte proposé par les intergiciels. Tout d'abord, il définit les objectifs des intergiciels sensibles au contexte et énumère les éléments fonctionnels d'une application sensible au contexte. Puis, il décrit les caractéristiques des intergiciels et les techniques d'adaptation qu'ils peuvent mettre en oeuvre pour gérer l'adaptation des applications réparties. Enfin, il étudie un ensemble d'intergiciels sensibles au contexte, leurs caractéristiques et leurs limites.

La deuxième partie de cette thèse présente CAMidO.

– Le chapitre 3 présente le méta-modèle de CAMidO qui permet de décrire d'une part des concepts généraux à toutes les applications sensibles au contexte et d'autre part des informations plus spécifiques à chaque application.
– Le chapitre 4 est consacré à la description de l'architecture de l'intergiciel CAMidO, à savoir le gestionnaire de contexte et le gestionnaire d'adaptation ainsi que ses modes d'exécution.
– Le chapitre 5 décrit l'intégration de CAMidO à l'intergiciel OpenCCM et valide notre proposition en présentant les évaluations qualitatives et quantitatives de CAMidO.
  Des perspectives d'évolution de notre travail sont proposées en conclusion.

Première partie

État de l’art

Chapitre 1

Définition et modélisation du contexte

1.1 Introduction

L'étude de la sensibilité au contexte nous amène tout d'abord à étudier les travaux de recherche effectués pour définir et modéliser le contexte. La modélisation permet d'offrir les fondations pour une représentation expressive du contexte et de simplifier son utilisation.

Les modèles de description du contexte varient selon leur expressivité et le type d'informations de contexte qu'ils permettent de décrire. Cette variété de modélisation provient de l'utilisation du contexte dans les domaines variés de l'ingénierie comme le traitement du langage, les systèmes d'aide à la prise de décision ou les systèmes sensibles au contexte.

Dans le cadre de cette thèse, nous nous intéressons aux applications sensibles au contexte dans un environnement mobile. Ces applications sont caractérisées par le fait qu'elles s'exécutent dans un environnement hétérogène et dynamique. Cet environnement est soumis aux caractéristiques variables des ressources des dispositifs utilisés et à la mobilité des utilisateurs, ce qui rend nécessaire l'utilisation des applications sensibles au contexte qui détectent les variations de l'environnement et adaptent leurs comportements en conséquence.

Dans ce chapitre, nous commençons notre étude par la définition du contexte (cf. section 1.2), puis nous mettons en évidence l'importance de son utilisation pour augmenter les possibilités des applications, tout en passant en revue différents domaines où le contexte est utilisé (cf. section 1.3). Ensuite, nous décrivons les caractéristiques des informations de contexte (cf. section 1.4) et les différentes méthodes d'acquisition de ces informations (cf. section 1.5). Puisque la modélisation du contexte représente l'une des premières phases de prise en compte du contexte dans le processus de création d'applications sensibles au contexte, nous dressons un état de l'art des travaux de modélisation du contexte (cf. section 1.6), puis nous analysons ces différentes approches pour choisir l'approche la mieux adaptée aux environnements mobiles (cf. section 1.7). En conclusion, nous soulignons l'importance des intergiciels dans la simplification du développement des applications sensibles au contexte (cf. section 1.8).

1.2 Définitions

1.2.1 Contexte

Il existe dans les articles de recherche plusieurs définitions du mot contexte suivant les domaines de l'ingénierie. De manière à situer notre définition d'un contexte, nous commençons par donner une définition générale du terme selon un dictionnaire [76], puis nous présentons plusieurs autres définitions utilisées dans le domaine de l'informatique.

La définition littérale du mot contexte considère le contexte comme ≪l'ensemble des unités d'un niveau d'analyse déterminé (phénomène, unité lexicale, phrase...) constituant l'entourage temporel (parole) ou spatiale (écriture) d'une unité≫ [76]. Selon la même source, le terme contexte représente aussi ≪un ensemble de circonstances liées à un événement qui se produit≫.

Dans le domaine informatique, il existe deux types de définition du contexte : le premier type définit le contexte en énumérant des entités spécifiques qui représentent le contexte associé à une application, comme la localisation par exemple, alors que le second type définit le contexte d'une manière conceptuelle et se focalise sur la structure des informations de contexte (une approche formelle). Nous présentons ci-après une synthèse de ces deux approches.

La liste des travaux qui définissent le contexte à travers des énumérations est assez longue ; [25] en fait une synthèse intéressante. Parmi ces travaux, la définition donnée par Schilit et Theimer [93] qui considère le contexte comme un ensemble d'informations de localisation et l'identité des personnes et des objets à proximité ainsi que les modifications pouvant intervenir sur ces objets, est l'une des premières définitions du contexte. Sensiblement la même définition a été reprise par Brown et al [14], ces derniers considèrent le contexte comme étant un ensemble d'informations de localisation, de profil de personnes auxquels il ajoute des informations temporelles. Ryan et al [89] ajoutent à cette énumération des informations sur l'environnement telles que la température.

Dans le second type de définitions plus générales du contexte, on trouve les définitions suivantes. Schmidt et al [95] définissent le contexte comme étant les informations relatives à un utilisateur et à l'état des dispositifs qu'il utilise. Brézillon et Pomerol [11] définissent le contexte comme étant toutes les connaissances qui participent à la résolution d'un problème. La définition la plus générale et la plus utilisée est celle donnée par Dey [34] qui décrit le contexte comme étant ≪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≫[1]. La définition de Dey inclut les informations explicitement données par l'utilisateur à travers l'interface d'une application par exemple, ainsi que les informations qui nécessitent une acquisition ou un calcul avant leur utilisation.

En donnant ces définitions, les auteurs se basent fortement sur leurs domaines de recherche. En ce qui nous concerne, nous utilisons la définition donnée par Dey pour le reste de ce document . Un contexte observable associé à une entité peut être capturée par le système ou calculée

[1] any information that can be used to characterise the situation of an entity. An entity is a peron, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.

à partir d'autres données (interprété), comme la localisation ou la bande passante. Un contexte observable est utilisé par une entité sensible au contexte pour un but déterminé. Dans le cadre de cette thèse, le contexte est utilisée par des applications dans le but de s'adapter aux changements pertinent du contexte. En effet, le contexte observable peut changer au fil du temps en prenant des valeurs différentes. Nous appelons la valeur observée d'un contexte observable à un instant donné : `contexte collecté` s'il est capturé à partir de capteurs, par exemple la valeur observée '10MBps' du contexte bande passante, et `contexte interprété` s'il est calculé à partir d'autres informations observées.

Tous le long de ce document, nous utilisons le terme de contexte pour désigner un contexte observable, le terme contexte collecté pour désigner la valeur du contexte observé à un instant donné et le terme contexte interprété pour désigner le contexte calculé à partir d'un ensemble d'informations observées à un instant donné.

1.2.2 Contexte pertinent

un contexte pertinent pour une application est un sous ensemble du contexte observé dont les changements de valeur affectent cette application. En d'autre termes, du point de vue de l'application tous les contextes observables ne sont pas pertinents pour l'application. L'ensemble des contextes observables sélectionnés pour une application donnée du fait que cette dernière est sensible à leurs changements de valeur est appelé contexte pertinent.

1.2.3 Adaptations et situation pertinente

Le contexte n'est pas une fin en soi, il est défini pour une finalité précise. Dans le cadre de cette thèse, la finalité visée est l'utilisation du contexte pour créer des applications sensibles au contexte. Ces applications surveillent le contexte qui les entoure, et adaptent leur comportement en fonction de ses changements.

L'adaptation est une modification d'une application en réponse à un changement dans son contexte, cette modification consiste soit à changer la structure de l'application, ou son comportement. Mais, tout changement de contexte n'engendre pas forcément l'adaptation de l'application. Nous appelons situation pertinente, toute observation du contexte pertinent ou composition d'observations à un instant donné qui nécessitent l'adaptation de l'application. En résumé, toute adaptation est conditionnée par la détection d'une ou de plusieurs situations pertinentes.

1.3 Importance du contexte dans le domaine informatique

Le contexte a été introduit dans différentes spécialités du domaine informatique, comme le traitement du langage, les systèmes de prise de décisions et l'informatique sensible au contexte. Par analogie au mode de fonctionnement du raisonnement humain, l'utilisation du contexte dans ces domaines consiste à ajouter une adaptabilité et une aide à la prise de décision à différents

types d'applications. Dans ce qui suit, nous présentons certains travaux de recherche qui s'intéressent à la notion de contexte dans les spécialités citées ci-dessus.

1.3.1 Contexte dans le traitement du langage

Le rôle du contexte est très important dans l'interprétation correcte du langage humain, qu'il s'agisse du texte ou de la parole. Par exemple, si une personne manque le début d'une discussion, il lui est difficile de comprendre le but de cette discussion à moins que l'interlocuteur ne lui donne un résumé de ce qui a été dit.

La traduction automatique est l'une des premières spécialités de ce domaine qui a utilisé le contexte. Ce dernier est représenté par les phrases précédant un mot ou une autre phrase qui permettent de sélectionner la traduction correcte parmi les différentes traductions possibles [8][106].

Le contexte utilisé dans le traitement du langage ne représente pas tout à fait le même contexte que celui utilisé dans le cadre de cette thèse, nous le mentionnons néanmoins pour montrer la variété des informations de contexte utilisées dans différents domaines de l'ingénierie.

1.3.2 Contexte pour l'aide à la prise de décision

Un processus d'aide à la prise de décision est lié à la situation dans laquelle il est utilisé. Des supports de prise de décisions basés sur le contexte ont été utilisés dans différents domaines. SIREN [58], un système sensible au contexte dans des situations d'urgences vise à aider les pompiers pour prendre des décisions rapides dans un environnement de grande tension. Le système est relié à une infrastructure pair à pair basée sur des capteurs pour collecter et distribuer des informations sur l'environnement. De cette façon, les pompiers peuvent avoir une vision globale de l'environnement et de la situation dans laquelle ils se trouvent. Leur tâche de prise de décision en est facilitée.

SART [91] est un système intelligent d'aide à la décision pour les responsables de la régulation du trafic sur une ligne de métro en situation d'incident. La notion de contexte joue un rôle important dans le projet SART. En effet, la stratégie appliquée pour résoudre un incident dépend essentiellement du contexte dans lequel il survient. Par exemple, un problème de traction sur une rame est résolu différemment selon que l'incident survient en heure de pointe ou en heure creuse.

1.3.3 Contexte dans l'informatique sensible au contexte

L'informatique sensible au contexte regroupe les applications ou les systèmes qui utilisent le contexte pour adapter leurs comportements. En se basant sur les travaux de Dey et Abowd [15], ceux de Schilit et al. [92], et ceux de Brown et al. [14], Chalmers a identifié six utilisations possibles du contexte dans un environnement sensible au contexte [24] :

– présentation et affichage des informations de contexte comme par exemple la localisation,

- association d'informations de contexte à une donnée de l'application, comme exemple l'association des personnes présentes à une conférence et leur localisation au programme de la conférence,
- la configuration sensible au contexte, comme par exemple le fait d'effectuer une impression sur l'imprimante la plus proche, sans intervention de l'utilisateur,
- lancement d'actions de réaction selon la valeur du contexte, comme le chargement d'une carte géographique d'une zone quand l'utilisateur se déplace vers cette zone,
- médiation contextuelle, qui consiste par exemple à utiliser le contexte pour modifier un service fourni,
- présentation sensible au contexte qui consiste à modifier l'interface d'une application selon le contexte.

Selon [35], le système *Active Badge* d'Olivetti [9] est la première application créée dans le cadre de l'informatique sensible au contexte. Cette application a été conçue et développée entre 1986 et 1992. Son but est de fournir des informations de localisation d'une personne dans un immeuble afin de rediriger les appels téléphoniques qu'elle peut recevoir vers le téléphone fixe le plus proche de sa localisation. Mais, le terme sensibilité au contexte a été introduit pour la première fois par Schilit et al. en 1994 [93]. Ils définissent un logiciel sensible au contexte comme étant un logiciel qui s'adapte en fonction de sa localisation d'utilisation et de l'ensemble des objets et des personnes qui l'entourent ainsi que les variations observées de leurs états. Une autre définition a été donnée par Dey [35] qui définit un système sensible au contexte comme étant un système qui utilise le contexte pour offrir des informations ou des services pertinents à l'utilisateur. Dans le cadre de cette thèse, nous considérons un logiciel sensible au contexte comme étant un logiciel qui adapte son comportement en fonction du changement de contexte. Cette adaptation consiste à modifier la structure du logiciel ou son comportement.

1.4 Caractéristiques des informations de contexte

Les informations de contexte sont des informations collectées à partir de plusieurs sources hétérogènes. De ce fait, elles ont des caractéristiques variables.

Chaque contexte observable peut être statique ou dynamique. Les observables statiques, représentent des informations qui ne changent pas au cours du temps. Il suffit de les collecter au lancement de l'application. Parmi les observables statiques, nous avons la taille de l'écran ou le profil d'un utilisateur. Les observables dynamiques représentent des informations dont les valeurs changent fréquemment, une observation de leur état peut devenir très rapidement obsolète.

Les observations de contexte effectuées à partir de capteurs physiques peuvent être sujets à des bruitages ou des erreurs de capture. Donc, ces observations sont incorrectes lorsqu'elles ne reflètent pas l'état réel de l'environnement, incohérentes lorsqu'elles contiennent des informations contradictoires et incomplètes lorsque certains aspects du contexte ne sont pas renseignés.

Afin d'utiliser les informations de contexte dans des applications informatiques, il est nécessaire de connaître pour chaque observable les caractéristiques citées ci-dessus. De plus, il est très important d'avoir une description abstraite de ces informations afin de pouvoir les utiliser.

Cette abstraction peut être fournie par des modèles de description du contexte. Nous décrivons différentes approches de modélisation du contexte dans la section 1.6.

1.5 Acquisition du contexte

L'acquisition des informations de contexte peut se faire selon plusieurs méthodes, indépendamment de l'application. Mostefaoui [75] a classé ces méthodes en trois catégories :

- **acquisition par profil** : cette méthode consiste à récupérer des informations soit à travers une interface graphique soit par l'intermédiaire d'un fichier de profil ;
- **acquisition par sonde** : cette méthode consiste à utiliser des sondes (capteurs) pour récupérer les informations de contexte. Schmidt et al. [95] font la distinction entre deux types de sondes : les sondes physiques et les sondes logiques. Les sondes physiques sont des composants électroniques qui permettent de mesurer des paramètres physiques dans l'environnement comme la température et la localisation, alors que les sondes logiques utilisent des logiciels pour récupérer des informations associées à l'hôte où s'exécute l'application comme la bande passante et la charge de la batterie. Le contexte collecté à l'aide de sondes est dynamique, il est donc nécessaire de mettre à jour les observations fréquemment ;
- **acquisition par dérivation** : la dérivation ou l'interprétation du contexte consiste à utiliser un ou plusieurs observables pour déduire ou calculer à la volée un contexte de plus haut niveau en utilisant des méthodes d'interprétation. Par exemple, la rue où se trouve une personne est un contexte de haut niveau calculé à partir des données de localisation en latitude et longitude collectées par un capteur GPS. La méthode d'acquisition d'un observable dépend en grande partie de l'observable.

L'acquisition du contexte à partir de capteurs n'est pas une tâche facile pour un développeur d'applications, cela est dû au fait que chaque capteur a une API propre et qu'un même observable peut être collecté à partir de capteurs différents et peut nécessiter un traitement supplémentaire (interprétation) avant qu'il ne puisse être utilisé par une application.

La collecte ou l'observation du contexte consiste à surveiller et à acquérir l'état observé du contexte. Pour une application sensible au contexte, la collecte du contexte peut être menée de plusieurs façons :

- par un accès direct de l'application à la source du contexte,
- en utilisant une infrastructure intergicielle qui interagit avec la source du contexte, et qui prend en compte certains aspects liés à la gestion du contexte comme son interprétation par exemple.

La première alternative nécessite que l'application s'interface directement à la source du contexte pour le collecter. Le problème de cette méthode réside dans le fait que le code d'interaction de l'application avec cette source du contexte est noyé dans le code fonctionnel de l'application, ce qui ne permet pas de réutiliser ce code pour d'autres applications qui interagissent avec la même source de contexte. De plus, le changement de la source du contexte nécessite la réécriture de l'application. L'application CyberGuide [66] est un exemple type d'application qui interagit directement avec des sources de contexte. Cette application interagit avec des capteurs

de localisation pour proposer à des touristes des endroits à visiter en utilisant une carte géographique interactive.

L'utilisation d'une infrastructure intergicielle pour interagir avec les sources du contexte vise à séparer la collecte du contexte de l'application [37]. Cette séparation a pour but de cacher à l'application les détails se rapportant aux sources du contexte. Certaines infrastructures se chargent de stocker les informations collectées, ce qui permet d'avoir un historique des variations de l'environnement. Cet historique peut être utilisé pour prédire les nouvelles variations du contexte par exemple.

Les infrastructures intergicielles permettent à l'application d'acquérir les observations du contexte soit par interrogation soit par notification. Le mode d'acquisition par interrogation consiste à envoyer une requête vers l'intergiciel pour récupérer la valeur du contexte, alors que le mode d'acquisition par notification consiste à s'inscrire auprès de l'intergiciel pour être notifié d'un changement de valeur du contexte. Le mode requête suppose que l'application sait quand demander l'information de contexte. Le mécanisme de notification est approprié pour un besoin répétitif du contexte. Dans ce cas, l'application peut définir des conditions sur les valeurs du contexte qui précisent quand elle doit être notifiée.

Les infrastructures intergicielles permettent de faciliter le développement des applications sensibles au contexte en se chargeant de collecter le contexte et de le fournir à l'application, ce qui décharge le développeur de programmer l'interaction de son application avec les capteurs.

Pour faciliter l'acquisition, le partage, l'utilisation et la réutilisation des informations de contexte, il est nécessaire de fournir un niveau d'abstraction de ces informations. Cette abstraction peut être obtenue par des modèles de description du contexte. La section 1.6 décrit les différents types de modélisation qu'une application ou qu'un intergiciel peuvent utiliser afin de décrire le contexte auquel ils sont sensibles.

1.6 Modélisation des informations de contexte

Pour décrire la sensibilité d'une application à son contexte d'exécution, il faut déterminer les contextes auxquels cette application est sensible et les décrire dans un modèle. Par conséquent, la modélisation du contexte est la première étape dans le processus de création d'applications sensibles au contexte. Cette modélisation permet à l'application et/ou aux intergiciels de faciliter l'interaction avec le contexte en fournissant une description abstraite des observables. La diversité des informations de contexte et leur utilisation dans divers domaines engendrent différentes façons de les modéliser.

Dans cette section, nous classifions la modélisation du contexte en quatre types d'approches : l'approche paires/triplets, l'approche orientée modèles, l'approche basée sur la logique et l'approche orientée ontologie. Cette classification se base sur la façon dont le contexte est modélisé, les outils utilisés à cet effet, l'expressivité du modèle et la possibilité de le réutiliser.

1.6.1 Approches paires/triplets

Les approches paires/triplets permettent de modéliser les valeurs observées du contexte ; elles utilisent à cet effet des structures de données très simples pour les décrire. Les premiers travaux visant à décrire le contexte sont ceux de Schilit, Theimer et Welsh [94]. Ils proposent d'utiliser un serveur d'environnement dynamique qui se charge d'observer un ensemble de contextes pour le compte d'une application. Chaque observation du contexte est sauvegardée dans une variable d'environnement constituée d'une paire clés/valeurs. La figure 1.1 illustre une variable d'environnement qui permet de décrire les imprimantes les plus proches.

```
Nearest_Printer=HP:HP4:HP6
```

FIG. 1.1 – Acquisition par une variable d'environnement

Schmidt [95] modélise le contexte comme un triplet constitué de l'observable, de la valeur observée du contexte, et un degré de certitude sur la cohérence des données observées. La figure 1.2, illustre un exemple de contexte décrit à l'aide de ce triplet.

```
C=(Location,MeetingRoom,95)
```

FIG. 1.2 – Acquisition du contexte avec des triplets

L'approche paires/triplets, et plus particulièrement celle qui utilise la modélisation clés/valeur, est très fréquemment utilisée dans les *frameworks* des services distribuées, comme le *framework Capeus* [90] par exemple, ou bien dans des travaux qui se focalisent sur l'aspect d'adaptation au changement du contexte, comme dans [85], [68], et [36] qui utilisent ce type de modélisation pour décrire des informations de localisation.

Discussion

La modélisation paires/triplets utilise des structures de données simples à gérer. Par contre, cela ne permet pas une description complète du contexte, ni l'expression des relations qui peuvent exister entre les informations de contexte ou la manière de déduire un contexte de haut niveau. De plus, cela décrit une observation et non un observable. Ce type de modélisation ne prend pas en compte la description des actions d'adaptation ni les conditions qui permettent de les déclencher.

1.6.2 Approches orientées modèle

L'approche orientée modèle utilise des modèles formels pour modéliser les informations de contexte. Son objectif principal est d'offrir la possibilité d'encapsuler le contexte et de permettre sa réutilisation. Dans cette section, nous décrivons un ensemble d'approches appartenant à cette catégorie de modélisation, les informations qui peuvent être décrites avec ces modèles, et les outils utilisés par chaque modèle pour décrire les informations de contexte.

Unified Modeling Language (UML)

La généricité du langage UML en fait un langage approprié pour modéliser le contexte. Bauer [7] a utilisé le langage UML afin de modéliser le contexte auquel une application de gestion du trafic aérien est sensible. Mais, le modèle proposé est spécifique à l'application et ne peut être utilisé dans d'autres applications sans modification.

Sheng et Benatallah [96] ont proposé un méta-modèle basé sur une extension d'UML qui permet de modéliser le contexte auquel des services web sont sensibles. Ce langage est appelé *ContextUML*. Comme l'illustre la figure 1.3, ce méta-modèle est composé de plusieurs classes qui permettent de créer des services sensibles au contexte. La classe *Context* permet de décrire un contexte observable. Ce dernier peut être un contexte de bas niveau représenté par la classe *AtomicContext*, ou bien un contexte interprété représenté par la classe *CompositeContext*. Pour chaque contexte de bas niveau, le modèle permet de spécifier la source à partir de laquelle il a été collecté. Le méta-modèle *ContextUML* permet aussi la description des actions d'adaptation et les situations pertinentes qui permettent de les déclencher. Le méta-modèle *ContextUML* est caractérisé par sa généricité. Mais, la description des relations de dérivation et de dépendance entre les informations de contexte n'a pas été considérée. De plus, ce méta-modèle n'offre pas le moyen de décrire la qualité des informations de contexte ni leur validité temporelle.

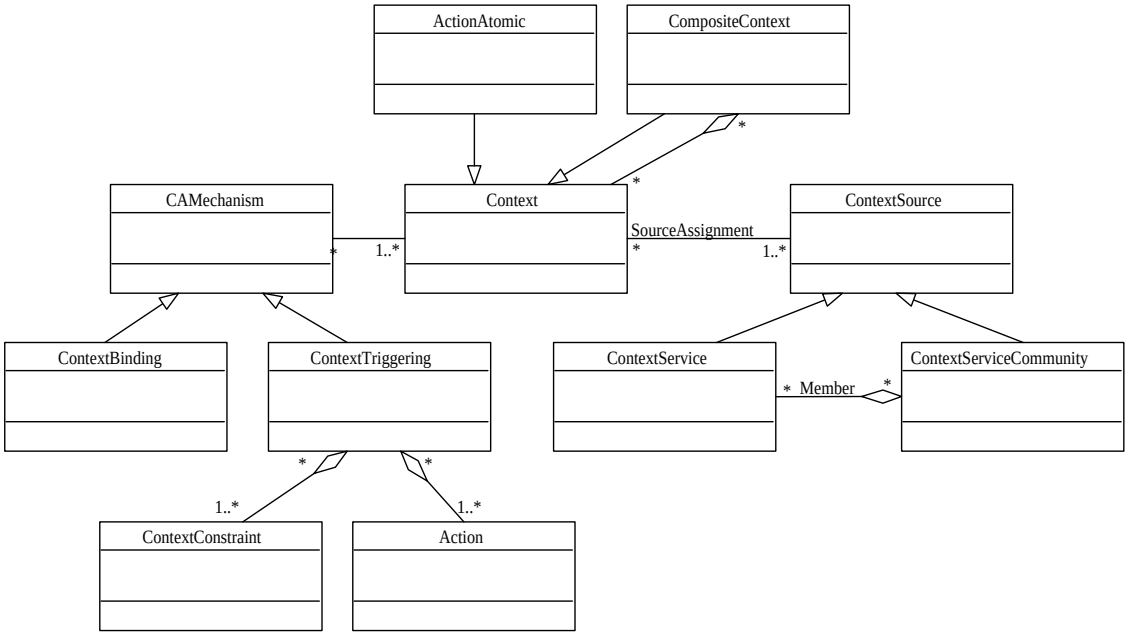

FIG. 1.3 – Méta-modèle de ContextUML

Context Modeling Langage (CML)

Afin de pouvoir modéliser les caractéristiques des informations de contexte et leurs propriétés, Henricksen et al. [51][50] ont proposé un langage orienté objet dérivé de l'ORM (Object Role Modeling) [60]. Ce langage appelé CML (*Context Modeling Language*) [49] permet de modéliser le contexte auquel une application est sensible d'une manière formelle. Un outil graphique assiste le concepteur d'applications dans la tâche de description du contexte auquel son application est sensible. Il lui offre un moyen de décrire les caractéristiques des informations de contexte (capturé, statique,dérivé ou information de profil) et les dépendances entre ces informations. CML permet aussi de spécifier la qualité de chaque information observée et sa validité temporelle.

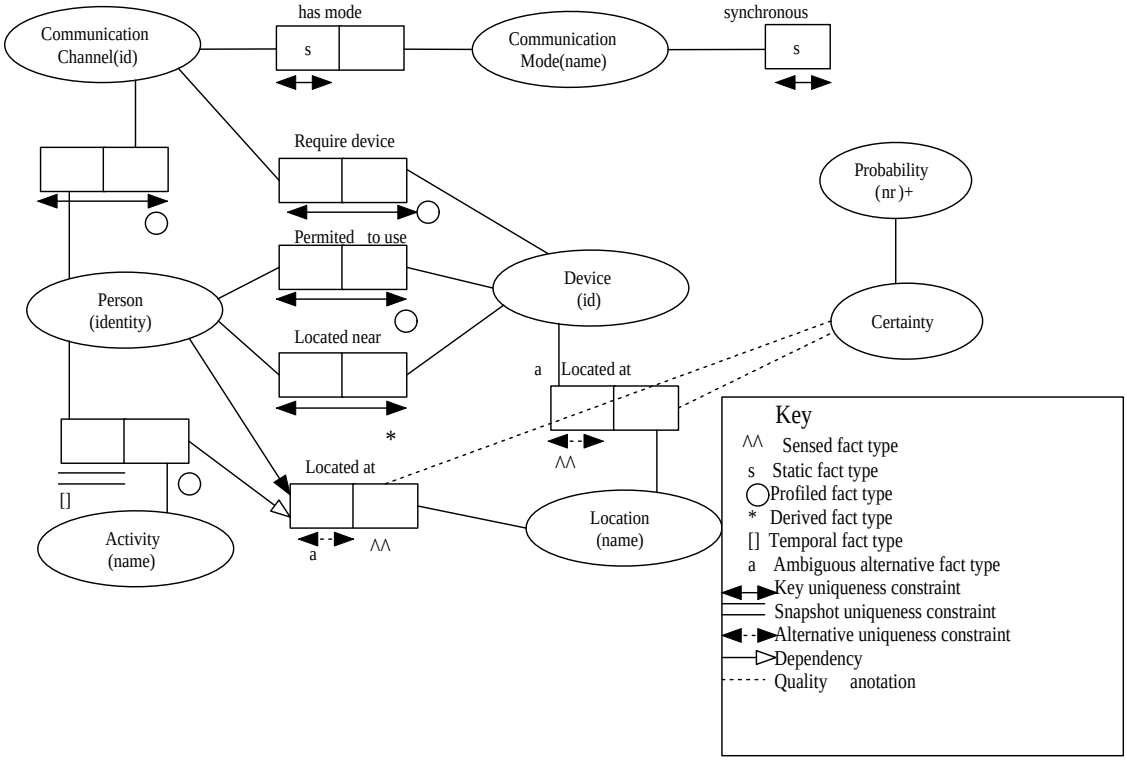

FIG. 1.4 – Exemple de modélisation avec CML

Dans cette modélisation, les informations de contexte sont groupées en un ensemble d'entités. Chacune d'elle décrit un objet conceptuel ou physique tel qu'une personne, un dispositif ou un réseau. Les propriétés des entités telles que le nom de la personne, le nom du dispositif ou l'identificateur du réseau sont représentées par des attributs. Les entités sont liées à leurs attributs ou à d'autres entités à travers des relations. À l'aide de ces relations, le concepteur peut classifier les informations de contexte associées à chaque entité. S'il utilise une association de type *sensed* entre une entité et son attribut, cela veut dire que cet attribut est un contexte obtenu à partir de capteurs matériels ou logiciels, comme la localisation d'une personne. La figure 1.4 illustre un

exemple de modélisation de contexte avec l'outil CML en utilisant différents types d'associations entre les informations modélisées.

Utilisation d'un langage de balises pour profil

Le point commun entre toutes les approches qui utilisent un langage de balises est la structure hiérarchique des données modélisées. Cette modélisation consiste en un ensemble de balises avec des attributs. Dans XML (*eXtensible Markup Language*), les balises sont définies dans une DTD (*Document Type Definition*). La description des profils est l'utilisation typique de ce type de modélisation.

Pascoe [83] a utilisé un protocole d'échange de données contextuelles au format XML entre un serveur et un utilisateur mobile dans le cadre d'une application appelée *Stick-e Note*. Ce protocole baptisé ConteXtML [88] est utilisé d'une part pour décrire le contexte observé associé à l'utilisateur, d'autre part, pour décrire le profil de l'application. Ce profil représente le contexte nécessaire à cette application pour qu'elle puisse s'exécuter. Le message illustré par la figure 1.5 est un exemple qui représente la position observée de l'utilisateur. Cette position est décrite à l'aide des coordonnées x, y et z. Cette modélisation, bien qu'elle soit efficace pour certain types d'applications, ne définit qu'un format d'échange de données. Donc, elle ne permet pas de décrire les contextes interprétés ni le profil de l'utilisateur. De plus, aucune description des relations entre les informations de contexte n'est prévue. Du fait de sa grammaire DTD figée, elle est spécifique à un ensemble limité d'applications.

```
1 <context session="123" action="update">
2   <spatial proj="UTM" zone="3" DATAUM="Euro 1950 (mean)"/>
3     <point x="281993"  y="468790"  z="205"/>
4   </spatial>
5  ...
6 </context>
```

FIG. 1.5 – Exemple de description de contexte avec conteXtML

CC/PP (*Composit Capability/Prefrence Profile*) [63] est la proposition du W3C pour la représentation de profils. C'est un cadriciel basé sur RDF [69]. CC/PP permet de décrire les capacités d'un dispositif ainsi que les préférences de l'utilisateur en utilisant une structure de profils. Cette structure est composée d'une hiérarchie à deux niveaux. Chaque profil a un certain nombre de composants et chaque composant a des caractéristiques, comme l'illustre la figure 1.6, qui représente le profil d'un client. Le vocabulaire offert par CC/PP n'est pas riche et a besoin d'être étendu car il est restreint à la description de profil. De plus, il ne permet pas la description de relations et de contraintes complexes entre les informations de contexte. Certaines tentatives d'extension de CC/PP ont été effectuées par Held et al. [48] et Indulska et al. [54]. Held et al. ont proposé le *Comprehensive Structured Context Profile* (CSCP) qui permet d'avoir une description du contexte non limitée à deux niveaux hiérarchiques. Cette proposition ne permet pas elle non plus de décrire des relations, des contraintes ou des dépendances entre les informations de contexte. Les travaux de Indulska et al. ont étendu le vocabulaire de CC/PP pour pouvoir décrire la localisation, les caractéristiques du réseau et les dépendances d'une application. Ainsi, cette modélisation

peut être utilisée pour décrire les contextes observables associés à une application sensible au contexte, mais Indulska et al. ont conclu que malgré cette extension, cette modélisation reste non intuitive et difficile à utiliser pour décrire des informations complexes [54].

```
<rdf:Description rdf:about="http://www.example.com/profile#MyProfile">
  <ccpp:component>
    <rdf:Description rdf:about="http://www.example.com/profile#TerminalHardware">
      <rdf:type rdf:resource="http://www.example.com/schema#HardwarePlatform" />
      <ex:displayWidth>320</ex:displayWidth>
      <ex:displayHeight>200</ex:displayHeight>
    </rdf:Description>
  </ccpp:component>
</rdf:Description>
```

FIG. 1.6 – Exemple de description de contexte avec CC/PP

Les modélisations orientées balises sont réutilisables grâce à leur DTD et à leur simplicité d'utilisation, mais les modèles existants ne permettent pas de décrire les relations entre les informations de contexte, ni le type d'acquisition de ces informations (collectées à travers des capteurs, interprétées ou informations de profil).

Utilisation d'un langage de balises pour l'adaptation

Capra et al. [20] utilisent le langage XML afin de modéliser le profil de l'utilisateur ainsi que le profil d'une application. Cette modélisation est utilisée par un intergiciel réflexif appelé CARISMA [22] afin de reconfigurer une application ou de l'adapter aux changements du contexte. Comme le montre la figure 1.7, le modèle proposé par Capra permet de décrire le contexte auquel l'application est sensible ainsi que les actions d'adaptation. Les balises *RESOURCE* et *STATUS* permettent de décrire la situation pertinente à laquelle l'application est sensible. L'action d'adaptation est décrite soit à l'aide de la balise *BEHAVIOR* s'il s'agit d'une adaptation proactive (cf. Section 2.3.4) ou à l'aide de la balise *CALLBACK* s'il s'agit d'une adaptation réactive (cf. Section 2.3.4). Cette modélisation du contexte ne permet ni de décrire les relations existantes entre les informations de contexte comme la dépendance et la dérivation, ni de choisir les sources de ces informations.

```
<RESOURCE name="battery">
  <STATUS operator="lessEqual" value=x/>   %configuration du contexte
    <BEHAVIOR  policy="disconnect"/>         %politique
  </RESOURCE>
  <RESOURCE name="location">
    <STATUS operator="lessEqual" value=x/>   %configuration du contexte
      <CALLBACK>
         <APPLICATION_REF>obj_ref_O</APPLICATION_REF> %politique
         <ENTRY_POINT>method_m</ENTRYPOINT>
         <PARAMETER>/CONTEXT/RESOURCE[@name="location"]/@value</PARAMETER>
      </CALLBACK>
     </RESOURCE>
```

FIG. 1.7 – Exemple de modélisation de contexte dans CARISMA

L'originalité de la modélisation utilisée dans CARISMA réside dans la description des règles d'adaptation constituées des situations pertinentes et des opérations d'adaptation à invoquer lorsque ces situations sont détectées.

Utilisation d'une grammaire

Stephen et al. [113] ont proposé une grammaire pour décrire un langage de description d'interfaces sensibles au contexte nommé CA-IDL. Ce langage permet d'ajouter la sensibilité au contexte à des applications réparties orientées objet. Le langage CA-IDL permet de décrire les contextes auxquels l'application est sensible, les situations pertinentes et les actions d'adaptation que l'intergiciel doit invoquer lors de leur détection. Comme l'illustre la figure 1.8, la plate-forme qui fournit le langage CA-IDL [101], définit trois catégories de contexte observable. Chaque catégorie est constituée d'un nombre limité de types de contexte. La catégorie *DeviceSpecificContext* permet de décrire des informations de contexte spécifiques à une machine. La catégorie *EnvironmentSpecificContext* permet de décrire des informations de contexte spécifiques à l'environnement qui entoure l'application. Enfin, la catégorie *UserSpecificContext* permet de décrire des informations de contexte spécifiques à l'utilisateur.

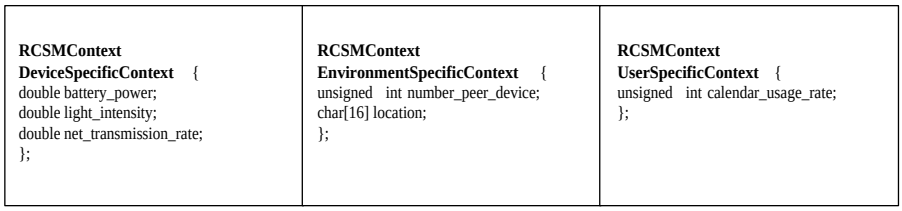

FIG. 1.8 – Catégories de contexte utilisées par CA-IDL

La figure 1.9 illustre la description de l'interface sensible au contexte *instructor-object* à l'aide du language CA-IDL. Cette interface est sensible aux situations pertinentes (location="GWC320") et ((number_peer_device>1) and (location="GWC320")). Une action d'adaptation est associée à chaque situation pertinente.

```
1  //context-sensitive interface
2  interface instructor_object
3  {
4    //context variable
5    RCSMContext_var DeviceSpecificContext C1 where (location="GWC320");
6    RCSMContext_var EnvironmentSpecificContext C2 where (number_peer_device>1);
7    //Context-sensitive method
8    [outgoing][activate at C1] void sendDocumentToPrinter();
9    [outgoing][activate at C1 and C2] void notifyUser();
10 }
```

FIG. 1.9 – Exemple de description CA-IDL

Discussion

Les approches orientées modèle utilisent un modèle formel pour décrire le contexte. Ce type d'approche permet de décrire le contexte de manière plus riche que les approches paires/triplets. De plus, cela offre aux concepteurs d'applications la possibilité de réutiliser le modèle pour d'autres applications. Les modélisations basées sur un langage de balises permettent de décrire des profils ou des contextes simples, sans offrir la possibilité de décrire des relations de dérivation et de dépendance entre ces informations. Les modélisations existantes basées sur UML permettent de décrire des relations entre les informations de contexte mais ne prennent pas en compte la description des dépendances entre ces informations, ni la qualité ou la validité temporelle des données décrites. CML est venu remédier à certains de ces manques en proposant un modèle avec visualisation graphique qui permet de typer le contexte, et de décrire un ensemble de relations entre plusieurs contextes observables. Le langage CA-IDL est un langage qui permet de décrire non seulement les situations pertinentes de l'application mais aussi les actions d'adaptation et les conditions de leur déclenchement. Ces conditions représentent des expressions régulières entre les situations pertinentes. Pour cela, CA-IDL offre une grammaire qui permet de décrire la sensibilité au contexte, mais cette grammaire reste limitée dans la mesure où elle n'offre pas la possibilité d'ajouter de nouvelles sources de contexte et de nouveau observables sans modifier la grammaire de CA-IDL, ni le moyen de décrire l'interprétation d'un contexte de haut niveau.

Le tableau 1.1 résume les caractéristiques des approches orientées modèles en fonction des informations qu'elles permettent d'exprimer et le langage utilisé à cet effet. D'après ce tableau, nous constatons que la plupart des modèles ne permet pas de spécifier à partir de quels capteurs sont effectuées les observations, ni de décrire les états du contexte (situations pertinentes) qui nécessitent une adaptation de l'application. Ces modèles n'offrent pas non plus le moyen de décrire la politique d'adaptation à utiliser quand des situations pertinentes sont détectées. La modélisation ContextUML, CA-IDL et celle proposée par Capra [19] se distinguent par le fait qu'elles permettent la description des situations pertinentes ainsi que les méthodes d'adaptation nécessaires si ces situations sont détectées. Mais, elles ne permettent pas de décrire la manière d'interpréter un contexte de haut niveau. De plus, les modélisations CA-IDL et celle proposée par Capra [19] n'offrent pas la possibilité de décrire les sources de contexte.

La modélisation des méthodes d'adaptation, des situations pertinentes, des règles d'interprétation, et des sources du contexte nous semble être très intéressante pour automatiser le processus d'adaptation. La solution que nous proposons modélise toutes ces informations dans le but de faciliter la tâche de programmation des applications sensibles au contexte, nous présentons ce modèle dans le chapitre 3.

Modèle	Modélisation basée sur	Type de contexte	Modélisation des relations de dépendance	Modélisation des règles d'adaptation	Modélisation de l'interprétation	Modélisation des sources du contexte	Modélisation des situations pertinentes
ContextUML	UML	Tout type de contexte	Oui	Oui	Oui	Oui	Oui
CML	ORM	Tout type de contexte	Oui	Non	Non	Non	Non
ConteXtML	XML	Profil	Non	Non	Non	Non	Non
CC/PP	RDF	Profil	Non	Non	Non	Non	Non
CSCP	RDF	Profil	Non	Non	Non	Non	Non
CARISMA	XML	Profil de l'application	Non	Oui	Non	Non	Oui
CA-IDL	Grammaire	environnement, profil utilisateur, ressources du dispositif	Non	Oui	Non	Non	Oui

TAB. 1.1 – Synthèse sur les approches orientées modèle

1.6.3 Approches basées sur la logique

Les modèles basés sur la logique sont caractérisés par un très grand degré de formalité. Ils utilisent l'algèbre booléenne et la logique du premier ordre pour modéliser le contexte. La logique permet de définir des conditions qui nécessitent de déduire des faits ou des expressions à partir d'un autre ensemble d'expressions ou de faits. Par conséquent, dans les modèles basés sur la logique, le contexte est défini comme des faits, des expressions ou des règles.

La première approche de modélisation du contexte en utilisant la logique a été publiée en 1993 par McCarthy et son équipe [70], [71]. McCarthy définit le contexte comme une entité mathématique abstraite ayant des propriétés. Cette formalisation logique est fondée sur une réification du contexte et un méta-prédicat *ist* ; *ist(p,c)* signifie que l'assertion *p* est vraie dans le contexte *c*. Par exemple la formalisation c : *(ist(contextof("Histoire de Sherlock Holmes"),"Sherlock Holmes est un détective")* considère que le personnage Sherlock Holmes est un détective dans l'histoire de Sherlock Holmes. Ce type de modélisation est utilisé dans le domaine de l'intelligence artificielle où les connaissances sont regroupées en micro-théories [46] ; selon les valeurs du contexte, on se place dans ou hors d'une micro-théorie.

Discussion

Les approches basées sur la logique utilisent l'algèbre booléenne ou la logique du premier ordre pour modéliser le contexte dans le but de raisonner sur les informations collectées. Cette modélisation ne permet pas de décrire la validité temporelle des informations ni les relations qui peuvent exister entre les informations de contexte, mais elle est très efficace pour raisonner sur le contexte et déduire des actions de réaction si une situation pertinente est détectée. L'approche basée sur la logique peut être utilisée dans l'informatique sensible au contexte afin d'intégrer et d'interpréter les données collectées.

1.6.4 Approches orientées ontologie

Une ontologie est une description sémantique, structurée et formelle des concepts d'un domaine et de leurs inter-relations [103]. Plusieurs modèles d'ontologies ont été proposés pour décrire le contexte. Ces approches permettent non seulement de modéliser le contexte, mais aussi de raisonner sur les données décrites. Les approches orientées ontologies sont caractérisées par une possibilité d'extension et de partage des données. Dans cette section, nous donnons tout d'abord une définition des ontologies et nous décrivons leurs caractéristiques, puis nous étudions trois ontologies de description du contexte. Cette étude se focalise sur les éléments du contexte représentés, le type de liens entre ces informations de contexte et le domaine d'utilisation de chaque ontologie.

```
1 [regle1: (?P hasLocation ?x) Equal(?x,'Salle de Reunion')-> Alerte('Salle de Reunion') ]
```

FIG. 1.10 – Exemple de règle de logique du premier ordre

Définition et caractéristiques des ontologies

Le mot ontologie est employé dans des domaines très différents touchant la philosophie, la linguistique et l'informatique. En informatique, une ontologie est définie comme un ensemble structuré de savoirs dans un domaine particulier de la connaissance ou un ensemble de concepts organisés en graphe dont les relations peuvent être sémantiques, de composition ou d'héritage.

Une ontologie est un ensemble de classes, de relations existant entre ces classes, de propriétés attachées aux classes et d'axiomes [107]. La création d'une ontologie se fait avec un langage logique, de façon à ce que l'on puisse faire des distinctions détaillées, précises, cohérentes et logiques entre les classes, les propriétés et les relations. Certains outils ontologiques peuvent effectuer des raisonnements automatisés et ainsi apporter des services évolués aux applications tels que l'exploration et la recherche conceptuelle/sémantique, le support de décision, la gestion des connaissances et les bases de données intelligentes. Les ontologies sont caractérisées par la possibilité de partager des connaissances entre plusieurs systèmes. De plus, elles sont ouvertes et extensibles, ce qui permet à chaque système de les enrichir et d'exploiter les notions qui y sont déjà définies. En effet, les langages d'ontologies offrent le moyen de publier, d'étendre des ontologies existantes et d'employer diverses ontologies existantes pour compléter une nouvelle ontologie.

De nombreux langages informatiques sont apparus pour construire et manipuler des ontologies. Dans le but de mettre au point un langage standardisé, le W3C a créé en novembre 2001 un groupe de travail qui a abouti à la recommandation OWL (*Ontology Web Language*) en février 2004 [107]. OWL définit une syntaxe pour décrire et construire des vocabulaires afin de créer des ontologies. OWL est un langage basé sur RDF [108], il est défini en trois sous-langages de plus en plus expressifs, chacun étant une extension du précédent. OWL-Lite est le sous-langage le plus simple. Il est destiné aux utilisateurs qui ont besoin d'une hiérarchie de concepts simples. OWL-DL permet une expressivité bien plus importante. Il est fondé sur la logique descriptive qui est un domaine de recherche étudiant la logique, ce qui confère à OWL-DL son adaptation au raisonnement automatisé. OWL-Full est la version la plus complexe de OWL, mais également celle qui permet le plus haut niveau d'expressivité. Toutefois, les problèmes liés à la complexité de calcul en un temps acceptable le rendent inutilisable.

À chaque ontologie, on peut associer un moteur d'inférence [56] qui permet de raisonner sur les informations de contexte en exécutant des règles d'inférence. Comme l'illustre la figure 1.10, chaque règle est structurée en deux parties séparées par le symbole $\rightarrow$. La partie gauche de la règle représente des conditions, alors que la partie droite représente une ou des actions à exécuter. Les actions ne sont exécutées que dans le cas où les conditions sont vérifiées. La règle de la figure 1.10 exprime le fait que si une personne se trouve dans la salle de réunion, une alerte lui est envoyée.

```
1  public class Alerte  extends BaseBuiltin
2  {
3    public Alerte()
4    {
5      super();
6    }
7    public String getName()
8    {
9      return "Alerte";
10   }
11   public int getArgLength()
12   {
13     return 1;
14   }
15   public void headAction(Node[] args, int length, RuleContext context)
16   {
17      Node inst=getArg(0,args,context);
18      System.out.println("vous vous trouvez en"+ inst.toString());
19   }
20   public boolean bodyCall(Node[] args,int length,RuleContext context)
21   {
22     return true;
23   }
24 }
```

FIG. 1.11 – Code associé à l'opération d'inférence Alerte avec Jena

Les moteurs d'inférence fournissent un ensemble d'opérations basiques prédéfinies (opérations de comparaison et d'ajout d'instances dans l'ontologie) et offrent aux développeurs la possibilité de définir leurs propres opérations (l'action Alerte dans l'exemple de la figure 1.10). Il est important de souligner que ces opérations peuvent être invoquées dans la partie gauche de la règle si elles ont un booléen comme paramètre de retour. Dans le cadre de Jena [56], le développeur peut définir ses propres opérations en créant une classe qui porte le nom de l'opération et qui implémente l'interface *Basebuiltin*. La figure 1.11 illustre l'implémentation de l'opération Alerte utilisée dans l'exemple de la figure 1.10. La méthode *headAction* de la figure représente le code que le moteur d'inférence doit exécuter si cette opération est invoquée dans la partie droite de la règle, alors que la méthode *bodyCall* représente le corps que le moteur d'inférence doit invoquer si l'opération est placée dans la partie gauche de la règle. La méthode *getName* doit retourner le nom de l'opération, alors que la méthode *getArgLength* doit retourner le nombre de ses paramètres d'entrée.

Nous présentons ci-après trois ontologies de contexte : COBRA-ONT (*Context Broker Architecture ontology*), CONON *(CONtextONtology)*, et CoOL (ContextOntology Language) dans le but d'étudier les contextes représentées dans chaque ontologie, le type de liens entre ces informations de contexte et le domaine d'utilisation de chaque ontologie.

CoBrA-ONT

CoBrA-ONT (*Context Broker Architecture ONTology*) est une ontologie écrite en utilisant le langage OWL-DL. Elle permet la création d'applications multi-agents sensibles au contexte [27]. Cette ontologie a été créée pour décrire le contexte que les applications de maison intelligente multi-agents utilisent [28]. CoBrA-ONT est constituée de quatre sous-ontologies qui permettent

de décrire des informations de localisation en général, des informations d'agents (des personnes ou des agents logiciels), la localisation de ces agents et leurs activités.

Dans les environnements ubiquitaires, les capteurs sont très souvent utilisés pour détecter la présence de personnes ou de dispositifs mobiles dans une pièce ou à un emplacement spécifique. En utilisant CoBrA-ONT chaque capteur (considéré comme un agent) décrit les données qu'il a collectées. Par conséquent, ces données sont partagées entre les agents de chaque application. Ce partage de données permet au système multi-agent CoBrA [26] de raisonner sur les informations de contexte et d'enrichir l'ontologie avec les données qu'il a déduites. Par exemple, si un capteur détecte la présence d'un badge électronique dans une pièce, le système déduit que le propriétaire du badge se trouve dans la même pièce. Ce raisonnement permet au système non seulement de détecter de nouvelles informations de contexte, mais il lui offre aussi le moyen de résoudre les incohérences des données qui peuvent apparaître.

CoBrA-ONT se focalise sur la modélisation des contextes observables et observés utilisés par une application dans le but de les partager entre les agents de l'application. Ce partage offre au système la possibilité de raisonner sur les informations collectées. Cette modélisation ne prend pas en compte la description des situations pertinentes ni les actions d'adaptation. Par contre, CoBrA-ONT permet de décrire les dépendances entre les contextes observables à travers des propriétés ou en utilisant la notion d'héritage. Elle permet aussi de déduire de nouvelles valeurs du contexte en raisonnant sur les données de l'ontologie en utilisant TRIPLE [97]. TRIPLE est un langage de règles qui permet de raisonner sur des ontologies. L'avantage majeur de CoBrA-ONT est l'utilisation de l'ontologie qui par définition permet le partage des données, et le raisonnement sur son contenu.

CONON

CONON (*CONtext ONtology*) est une ontologie de contexte extensible écrite en utilisant le langage OWL-DL. Elle permet de décrire le contexte auquel des services peuvent être sensibles dans un environnement ubiquitaire [109]. Cette ontologie est structurée en deux niveaux. Le premier niveau permet de décrire les concepts généraux des informations de contexte, alors que le second niveau permet de décrire des contextes pertinents spécifiques à un domaine [110] comme l'illustre la figure 1.12.

Le premier niveau de l'ontologie CONON permet de décrire des concepts généraux communs à toutes les applications sensibles au contexte. Ces concepts sont constitués d'informations de localisation, d'informations sur l'utilisateur, d'informations sur des activités et des entités de calcul. Le second niveau de l'ontologie est extensible et permet d'ajouter à CONON des ontologies pour des domaines d'applications spécifiques.

L'ontologie CONON ne prend pas en compte la description des règles d'interprétation ni des politiques d'adaptation. Elle ne modélise pas non plus les capteurs à partir desquels les informations de contexte sont collectées. CONON ne décrit que le contexte et certaines relations existantes entre les informations de contexte en ajoutant la notion de qualité des données [109]. L'idée d'avoir un niveau d'ontologie pour décrire les concepts communs et un niveau plus spéci-

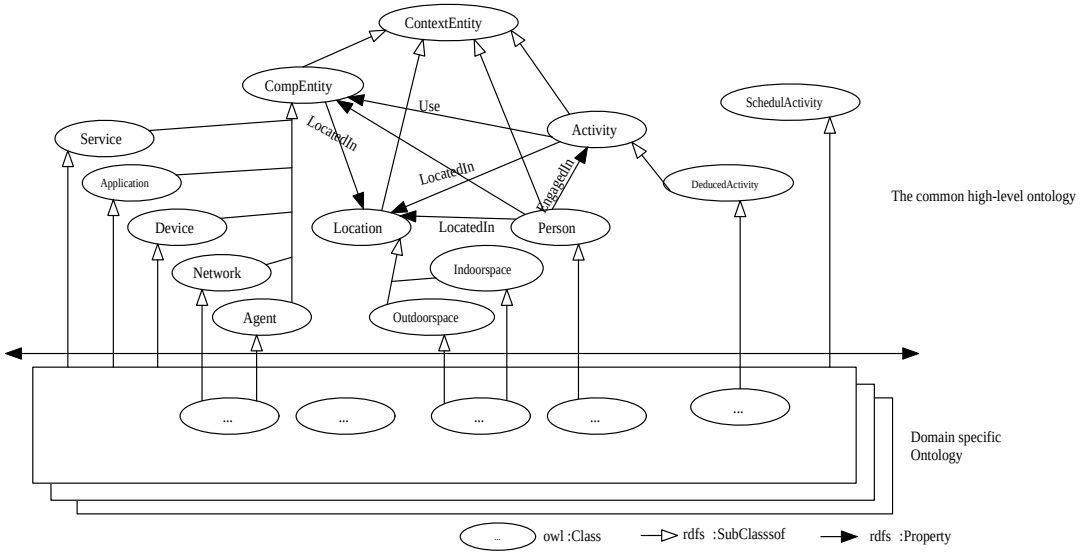

FIG. 1.12 – Ontologie de contexte CONON

fique à l'application nous semble être une idée intéressante à exploiter et à mettre en oeuvre pour la description des applications orientées composants sensibles au contexte.

CoOL

CoOL (Context Ontology Language) est un langage d'ontologie basé sur le modèle ASC (Aspect-Scale-Context) [100]. Ce modèle est composé de trois ensembles : un ensemble d'aspects qui représente le contexte observable, un ensemble d'échelles et un ensemble d'observations. Chaque aspect est composé d'un ensemble d'échelles, chaque échelle est alors composée d'un ensemble d'observations. Par exemple, l'aspect *CoordonnéeGeographique* peut avoir deux échelles, l'échelle *WGS84* ou l'échelle *GaussKrueger*. Une observation du contexte peut appartenir à l'une de ces deux échelles.

Comme l'illustre la figure 1.13, l'ontologie CoOL est constituée de deux parties, la première partie considérée comme le cœur de CoOL représente une projection du modèle ASC, ce qui permet de classifier les observations du contexte selon l'échelle des données qu'elles représentent. La deuxième partie de l'ontologie représente des schémas d'intégration de cette ontologie.

Le cœur de l'ontologie est écrit en plusieurs langages d'ontologie : OWL [107], DAML+OIL [81] et F-Logic [62] pour faciliter son utilisation par plusieurs types de plates-formes. Le schéma d'intégration représente un ensemble de protocoles qui permettent à des systèmes ou des services web d'utiliser le cœur de l'ontologie.

Dans CoOL, les relations décrites entre les informations de contexte peuvent exprimer la qualité de ces informations, mais il n'y a aucune notion de dépendance, ni d'interprétation. Pour

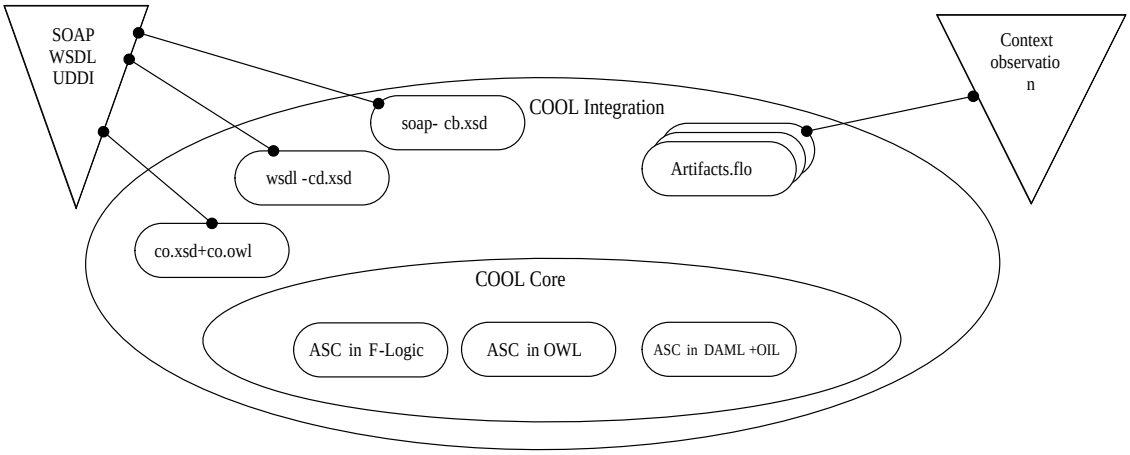

FIG. 1.13 – Le language d'ontologie CoOL

chaque contexte, aucune information ne permet de spécifier la source de l'information qui lui est associée.

Discussion

Les approches présentées dans cette section utilisent une ontologie pour décrire les informations de contexte. L'avantage de ces approches réside dans les caractéristiques même de l'ontologie, qui offrent non seulement le moyen de faire des descriptions sémantiques, mais aussi de publier les données décrites à travers le réseau.

Chacune des approches décrites ci-dessus se focalise sur la création d'un méta-modèle pour classifier les informations de contexte. Cette classification est liée aux applications qui utilisent ces ontologies. La séparation des informations de contexte communes aux applications et des informations spécifiques à un domaine donné est très intéressante car elle permet de réutiliser et d'enrichir plus facilement l'ontologie.

Les ontologies étudiées dans cette section ne prennent pas en compte la description des sources du contexte ni des situations pertinentes auxquelles l'application est sensible, elles ne permettent pas non plus de décrire les règles d'adaptation de ces applications. Cette constatation est justifiée par le fait que le but principal de ces ontologies consiste à décrire les caractéristiques des informations de contexte sans s'occuper de la collecte des informations de contexte, de leur interprétation, ni de l'adaptation de l'application. Elles considèrent que ces tâches sont soit à la charge du développeur de l'application soit à la charge d'un intergiciel sensible au contexte.

Vu le nombre d'ontologies de contexte existantes, il nous semble important de les standardiser. Cette standardisation peut être utilisée et étendue pour faciliter le développement des applications orientées composants sensibles au contexte.

1.7 Synthèse et évaluations

Les informations de contexte jouent un rôle important dans différents domaines de l'ingénierie. Ces informations sont collectées à partir de plusieurs sources. Les informations collectées peuvent être incohérentes, obsolètes, incorrectes ou incomplètes. La complexité des informations de contexte nécessite des moyens de modélisation pour simplifier leur utilisation. Dans la section 1.6, nous avons décrit quatre types d'approches de modélisation du contexte. Dans le but d'étudier la possibilité d'utiliser ces approches dans un environnement mobile sensible au contexte, nous évaluons ces approches selon la richesse et la qualité des informations qu'elles permettent de décrire, leur degré de formalisme, et la possibilité de leur utilisation dans différents types de systèmes. Le tableau 1.2 résume les caractéristiques de ces approches.

Les approches paires/triplets sont caractérisées par une pauvreté d'expressivité et la simplicité des données qu'elles représentent. Elles sont basées sur une description par un tuple (clé/valeur) ou par un triplet (contexte, valeur, degré de certitude). De ce fait, elles ne permettent pas de décrire les observables, ni les relations entre les informations de contexte, ni les règles d'interprétation.

Les approches orientées modèle sont des approches prometteuses car elles utilisent non seulement un modèle formel pour décrire le contexte, mais offrent aussi un méta-modèle de description qui peut être réutilisé par plusieurs applications ou intergiciels. Les approches orientées modèle existantes ne prennent pas en compte la description des règles d'interprétation de contextes de haut niveau, ni la modélisation des capteurs. Cependant, ces approches offrent une possibilité d'extension pour permettre la description de nouveaux types de relations entre les informations de contexte et ainsi la possibilité d'intégrer les notions qui manquent.

Les approches basées sur la logique sont des approches formelles dont l'utilisation a été longtemps restreinte au domaine de l'intelligence artificielle. Ces approches permettent de raisonner sur les informations de contexte pour déduire de nouvelles valeurs du contexte ou pour lancer des réactions au niveau de l'application ou du système. Les approches appartenant à cette catégorie permettent de décrire des relations entre les informations de contexte, mais leurs applications aux systèmes existants restent limitées ; cela est dû à la difficulté de description qu'engendre ce type d'approche.

Les approches orientées ontologie sont des approches formelles qui tirent parti des caractéristiques des ontologies pour modéliser le contexte. En effet, les caractéristiques de partage et de distribution des données ont été exploitées afin de définir des méta-modèles de description du contexte. De plus, les moteurs d'inférence fournis par les ontologies ont été utilisés pour déduire des contextes de haut niveau à partir des données collectées.

L'objectif de l'étude que nous avons effectuée consiste à montrer l'apport de chaque approche de modélisation de contexte dans le but d'utiliser l'une d'elles pour modéliser les informations de contexte associées aux applications orientées composant sensibles au contexte. Notre choix d'une approche de modélisation comporte plusieurs critères : l'expressivité du modèle, la possibilité de sa réutilisation et de son extension et la possibilité de publier les informations qu'elle permet de décrire. Selon le tableau 1.2, l'approche orientée ontologie répond bien à ces cinq cri-

tères, ainsi nous rejoignons la conclusion de l'étude effectuée par Stang et Linnhoff-Popien [99], basée sur six autres critères, qui stipule que l'approche orientée ontologie est l'approche la plus expressive et la plus prometteuse pour la description du contexte dans un environnement sensible au contexte.

1.8 Conclusion

Au début de ce chapitre, nous avons donné une définition du terme contexte puis nous avons montré l'importance de sa prise en compte dans le domaine informatique. Cette importance croît de plus en plus avec l'évolution technologique des équipements mobiles. En effet, les ressources limitées de ces dispositifs et les caractéristiques hétérogènes et dynamiques des systèmes qu'ils utilisent ont engendré la nécessité d'utiliser un nouveau type d'applications qui détectent le changement de l'environnement qui les entoure et qui s'adaptent en conséquence.

Les contextes pertinents utilisés par ces applications sont collectés à partir de sources hétérogènes ; ils sont soit explicitement fournis par l'utilisateur, soit capturés à partir de sondes ou encore dérivés (interprétés) à partir d'autres informations de contexte. De ce fait, ils peuvent être sujets à des ambiguïtés, des incohérences ou des imperfections.

Les caractéristiques variées des informations de contexte nécessitent des modèles de description abstraits. Dans la section 1.6, nous avons décrit plusieurs approches de modélisation : les approches orientées application, les approches basées sur la logique, les approches orientées modèle et les approches orientées ontologie. Après avoir analysé ces approches dans la section 1.7, nous avons conclu que les approches orientées ontologies sont les mieux adaptées pour décrire le contexte dans les systèmes mobiles sensibles au contexte.

La modélisation du contexte est la première démarche dans le processus de création d'applications sensibles au contexte. La modélisation permet de simplifier le développement de ce type d'applications, mais cela reste insuffisant, car la création de ce type d'applications nécessite plusieurs étapes de programmations supplémentaires, entre autres l'interaction de l'application avec les capteurs pour collecter les données, l'analyse de ces données, et le lancement des adaptations nécessaires quand une situation pertinente est détectée. Pour cacher à l'application tous les détails se rapportant à la collecte, à l'analyse du contexte et à l'adaptation de l'application, et pour pouvoir réutiliser ces mécanismes, il faut utiliser les services d'un intergiciel qui se charge d'automatiser toutes ces tâches. Dans le chapitre 2, nous étudions différents types d'intergiciels qui permettent de créer des applications sensibles au contexte et les mécanismes qu'ils utilisent à cet effet.

Approche de modélisation	Caractéristiques	Description de relations entre informations de contexte	Raisonnement sur le contexte	Réutilisation et extension du modèle	Publication des données décrites
Paires/triplets	Simplicité d'utilisation, pauvreté d'expression	Non	Non	Non	Non
Basée sur la logique	Très formelle	Non	Oui	Non	Non
Orientée modèle	Utilisation d'un méta-modèle	Oui	Non	Oui	Non
Orientée ontologie	Utilisation d'un moteur d'inférence	Oui	Oui	Oui	Oui

TAB. 1.2 – Caractéristiques des approches de modélisation du contexte

Chapitre 2

Intergiciels et sensibilité au contexte

2.1 Introduction

Ces dernières années ont été marquées par un très fort taux d'utilisation des équipements mobiles. Ces derniers, étant caractérisés par un environnement d'exécution qui change en permanence, ont engendré la nécessité d'utiliser des applications distribuées sensibles au contexte.

Le développement d'une application sensible au contexte est une tâche fastidieuse qui nécessite plusieurs étapes de programmation supplémentaires. Lors de la conception de l'application, il faut décrire les contextes pertinents pour l'application et les situations pertinentes auxquelles elle est sensible. Pendant l'exécution, l'application doit être interfacée avec les capteurs. Enfin, elle doit être adaptée lors de la détection des situations pertinentes. L'utilisation des services d'un intergiciel sensible au contexte afin de faciliter cette tâche s'avère nécessaire. Le terme intergiciel sensible au contexte englobe deux types d'intergiciels : les intergiciels qui s'auto-adaptent lors de la détection d'une situation pertinente et ceux qui changent le comportement ou la structure d'une application lors de la détection de ces situations. Dans le cadre de cette thèse, nous nous intéressons au deuxième type d'intergiciels qui se chargent d'adapter le comportement ou l'architecture des applications lors de la détection d'une situation pertinente.

Ce chapitre est consacré à l'étude de la sensibilité des applications aux variations du contexte et des intergiciels sensibles au contexte. Après avoir donné des définitions sur la sensibilité au contexte et les objectifs des intergiciels sensibles au contexte (cf. section 2.2), nous décrivons les éléments fonctionnels d'un système sensible au contexte (cf. section 2.3). Ensuite nous présentons dans la section 2.4 les caractéristiques et le rôle des intergiciels. Puisque notre thèse porte sur les intergiciels orientés composants, nous décrivons dans la même section les caractéristiques de ces intergiciels. Nous présentons aussi les techniques d'adaptations que les intergiciels utilisent afin de gérer l'adaptation des applications. Par la suite, dans la section 2.5, nous présentons les travaux de recherche effectués autour des intergiciels sensibles au contexte. Une

synthèse est présentée dans la section 2.6 où nous identifions notamment les limitations des intergiciels à faciliter le développement des applications sensibles au contexte. Enfin, la section 2.7 conclut ce chapitre.

2.2 Définitions et objectifs des systèmes sensibles au contexte

Il existe plusieurs définitions d'un système sensible au contexte, la première a été donnée par Schilit et Theimer [93]. Ces derniers définissent un système sensible au contexte comme tout système qui ≪s'adapte selon son endroit d'utilisation, la collection de personnes et objets voisins aussi bien que le changement de ces objets≫[1]. Hull et al. [53] ainsi que Pascoe et al. [84] considèrent la sensibilité au contexte comme la capacité d'un système à détecter, capturer, interpréter et réagir à un aspect changeant de l'environnement d'un utilisateur et des dispositifs qu'il utilise. D'autres définitions données dans [92], [14], [52], [111], [33] et [64] semblables à celle donnée par Schilit et Theimer, mettent l'accent sur l'adaptation des applications. Ils considèrent un système sensible au contexte comme tout système qui change et adapte son comportement dynamiquement selon son contexte, en incluant dans le terme système les applications et les intergiciels.

Selon Dey [34], les définitions de la sensibilité au contexte données ci-dessus n'incluent pas tous les types de systèmes sensibles au contexte. En effet, un système qui ne fait que collecter le contexte pour le fournir à une application ne rentre pas dans le cadre de ces définitions. Pourtant il est considéré par Dey comme un système sensible au contexte. Pour cela, Dey donne une définition plus générale des systèmes sensibles au contexte qui permet d'inclure ce type de système. Dans sa définition, Dey stipule qu' ≪ un système est sensible au contexte s'il utilise le contexte pour fournir des informations pertinentes et/ou des services à l'utilisateur, cette pertinence des données dépend des tâches de l'utilisateur ≫[2].

La définition donnée par Dey décrit le concept de sensibilité au contexte du point de vue consommateur, c'est à dire que le système est vu comme un fournisseur de contexte. Mais un système sensible au contexte a d'autres fonctionnalités en plus de celles décrites dans les différentes définitions citées jusqu'à présent. Ces fonctionnalités englobent non seulement l'acquisition du contexte, mais aussi son interprétation, et son analyse. Xiaohang [112] a constaté que les définitions des systèmes sensibles au contexte ignorent la description des procédures de gestion du contexte dans un environnement sensible au contexte. De ce fait, il a donné une nouvelle définition qui considère que ≪ la sensibilité au contexte d'un système logiciel est sa capacité à acquérir, gérer, interpréter et répondre aux changements du contexte afin de fournir les services appropriés ≫[3].

[1] Adapts according to its location of use, the collection of nearby people and objects, as well as the changes to those objects over time.

[2] A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

[3] Context-awareness is the ability for a software computing to acquire, manage, interpret, and response to context to provide appropriate services to the changing situation.

Dans le cadre de cette thèse, nous faisons la distinction entre les systèmes qui s'adaptent en fonction du contexte et ceux qui fournissent le contexte. Nous appelons `système pour la sensibilité au contexte` tout système qui se charge de fournir des informations de contexte, alors que nous appelons `système sensible au contexte` tout système qui s'adapte aux changements du contexte. Nous utilisons aussi le terme système sensible au contexte pour désigner les systèmes qui fournissent le contexte et s'adaptent ou adaptent d'autres systèmes aux changements pertinents du contexte.

2.3 Éléments fonctionnels d'un système sensible au contexte

Un système sensible au contexte est caractérisé par des fonctionnalités qui lui permettent d'interagir avec des capteurs, d'analyser les données collectées et d'agir en conséquence. La définition de la sensibilité au contexte donnée par Xiaohang [112] énumère les fonctionnalités que doit fournir ce type de système. Dans le cadre des intergiciels sensibles au contexte, ces fonctionnalités lui permettent d'acquérir des informations de contexte en interagissant avec des capteurs, d'interpréter ces données et de les analyser pour détecter les changements pertinents qu'ils subissent, et enfin, de s'auto-adapter ou d'adapter des applications sensibles au contexte lors de la détection des situations pertinentes.

Un système sensible au contexte peut utiliser un modèle de description du contexte qui lui offre un haut niveau d'abstraction du contexte. Le modèle lui décrit les contextes observables pertinents que le système doit collecter, les situations pertinentes qu'il doit détecter et les actions qu'il doit effectuer lors de la détection de ces situations pertinentes.

Dans cette section, nous décrivons les fonctionnalités des systèmes sensibles au contexte. Nous montrons aussi la difficulté qu'engendre le développement d'applications sensibles au contexte pour les développeurs sans l'utilisation de système pour la sensibilité au contexte. Nous souhaitons ainsi mettre en évidence la nécessite de déléguer certaines tâches de gestion du contexte et de l'adaptation de l'application à un intergiciel.

2.3.1 Acquisition du contexte

L'acquisition du contexte représente le processus de collecte des données à partir de l'environnement qui entoure le système. Comme nous l'avons décrit dans la section 1.5, les méthodes d'acquisition du contexte sont classées en trois catégories : l'acquisition par profil, l'acquisition par sonde et l'acquisition par dérivation. Un système sensible au contexte doit pouvoir interagir avec les sources du contexte quels que soient leurs types. Dans cette section, nous nous intéressons aux méthodes d'acquisition par profil et par sonde, car nous estimons que l'acquisition et la dérivation du contexte sont deux étapes différentes dans le processus de collecte du contexte.

La collecte des informations de contexte dépend de la source du contexte et du type des informations collectées. Les informations de profil sont des informations explicitement fournies par l'utilisateur, elles sont caractérisées par un changement peu fréquent. Le contexte collecté à partir

de sonde, que nous appelons contexte capturé, est sujet à de fréquents changements. Sa collecte nécessite une interaction avec des capteurs logiciels ou matériels distribués et hétérogènes. La fréquence du changement du contexte dépend du type du contexte.

La gestion de l'acquisition du contexte est difficile à mettre en œuvre directement dans une application. En effet, le développeur doit implémenter le code qui permet à son application d'interagir avec les capteurs et de récupérer le profil de l'utilisateur, ce code est noyé dans le code source de l'application, ce qui rend difficile sa réutilisation pour d'autres applications. Cette difficulté peut être réduite en utilisant les services d'un intergiciel pour la sensibilité au contexte. Cet intergiciel se charge de collecter le contexte et de le fournir à l'application. L'interaction entre l'intergiciel et l'application peut se faire soit en mode requête soit en mode notification (cf. section 1.5).

Les données collectées sont généralement sauvegardées dans un dépositaire de contexte pour être utilisées ultérieurement par le système.

2.3.2 Interprétation du contexte

L'interprétation du contexte représente le mécanisme qui déduit un contexte de haut niveau en utilisant d'autres informations de contexte. Les informations utilisées pour déduire un contexte de haut niveau peuvent être des informations interprétées, des informations capturées ou des informations de profil. La fréquence de mise à jour des informations interprétées dépend de la fréquence de mise à jour des informations utilisées lors de l'interprétation. À chaque fois qu'une nouvelle donnée est collectée, le mécanisme d'interprétation doit être relancé pour déduire les nouvelles valeurs du contexte.

2.3.3 Analyse du contexte

L'analyse du contexte est un processus qui contrôle continuellement les données collectées. Ce contrôle permet de filtrer les observations et les interprétations du contexte pour détecter les situations pertinentes dans le but d'adapter le système en conséquence ou de notifier les applications sensibles à ces situations pertinentes.

Le développement du processus d'analyse nécessite une interaction avec le processus de collecte du contexte via le dépositaire de contexte pour vérifier les valeurs du contexte et de détecter leurs changements pertinents. Grâce aux résultats obtenus par cet analyse, des décisions d'adaptation sont prises par le processus d'adaptation afin d'adapter le comportement de l'application aux nouvelles situations pertinentes.

2.3.4 Adaptation aux changements du contexte

Selon [32] : ≪Une adaptation est une modification d'un système, en réponse à un changement dans son contexte, avec l'objectif que le système résultant soit mieux à même pour réaliser sa fonction dans le nouveau contexte≫.

Concernant les applications orientées composant, nous distinguons deux types d'adaptation : l'adaptation structurelle et l'adaptation comportementale. L'adaptation structurelle consiste à modifier l'architecture de l'application, alors que l'adaptation comportementale consiste à modifier le comportement de l'application.

En général, on peut décomposer le processus d'adaptation quelque-soit son type en deux étapes successives :

– prise de décision concernant les opérations d'adaptation à effectuer,
– application des actions d'adaptation en utilisant un mécanisme d'adaptation fourni par le système.

Les conditions qui engendrent l'exécution de chaque étape nous permettent de distinguer deux natures d'adaptation : l'adaptation de nature réactive et l'adaptation de nature proactive. Si la prise de décision d'adaptation et son exécution sont conditionnés par la détection d 'une situation pertinente, on dit que c'est une adaptation réactive. Alors que si la décision d'adaptation est prise au moment de la détection de la situation pertinente alors que son exécution est conditionnée par un autre évènement, on dit que c'est une adaptation proactive. D'autre types d'adaptations proactives, que nous ne traitons pas dans le cadre de cette thèse, consistent à prendre la décision d'adaptation avant la détection d'une situation pertinente, simplement en utilisant l'historique des variations du contexte et un mécanisme d'apprentissage.

Le développement du mécanisme d'adaptation est difficile à mettre en œuvre. Mais, plusieurs techniques, que nous décrivons dans la section suivante, peuvent être utilisées pour faciliter cette tâche comme la réflexivité, la programmation par aspect, le paradigme composant conteneur et le moteur d'inférence.

2.3.5 Discussion

Les systèmes sensibles au contexte sont constitués d'éléments fonctionnels qui leur permettent d'interagir avec l'environnement et de changer de comportement selon les variations que subit cet environnement. Sans l'utilisation d'intergiciels pour gérer la sensibilité au contexte, la création d'applications réparties sensibles au contexte est une tâche fastidieuse. En effet, le développeur doit non seulement gérer la distribution de l'application et la programmation de son code métier, mais il doit aussi programmer l'interaction avec les capteurs afin de collecter les informations de contexte, l'interprétation des données collectées, leur analyse et enfin l'adaptation de l'application lorsqu'une situation pertinente est détectée.

Dans la section suivante, nous décrivons les caractéristiques des intergiciels. Nous présentons par la suite les techniques d'adaptation qu'ils peuvent utiliser pour gérer l'adaptation de l'application.

2.4 Intergiciels et techniques d'adaptation

Un intergiciel fournit des solutions réutilisables aux problèmes récurrents des applications tels que la répartition, la persistance des données, la sécurité, l'hétérogénéité et la portabilité. Son but

est de faciliter le développement des applications en séparant les préoccupations fonctionnelles et les préoccupations extra-fonctionnelles. Les préoccupations fonctionnelles sont la réalisation de la logique métier d'un système alors que les préoccupations extra-fonctionnelles sont indépendantes de la logique métier. Elles peuvent théoriquement être ajoutées ou supprimées sans modifier la logique métier de l'application.

Puisque notre thèse porte sur l'ajout de mécanismes de sensibilité au contexte à des intergiciels orientés composant, nous décrivons dans la section 2.4.1 les rôle des intergiciels. Puis nous décrivons dans la section 2.4 les caractéristiques des intergiciels orientés composant. Par la suite, nous présentons dans la section 2.4.3 les techniques d'adaptation qui peuvent être utilisées par ces intergiciels. Cela nous permettra d'évaluer leur utilisation pour le développement des applications sensibles au contexte.

2.4.1 Caractéristiques et rôle des intergiciels

Avec l'émergence des réseaux informatiques à large échelle, la construction des applications repose de plus en plus sur les systèmes répartis. Les intergiciels contribuent à faciliter le développement des applications distribuées en cachant l'hétérogénéité des réseaux et en offrant aux développeurs d'applications réparties des langages pour décrire le code métier de l'application.

Un intergiciel peut être considéré comme un noyau offrant un ensemble de services à l'application. Le noyau est caractérisé par un modèle de programmation qui offre une abstraction à l'hétérogénéité et à la distribution des machines sur le réseau. La figure 2.1 illustre l'organisation générale d'une application répartie au dessus d'un intergiciel : les entités de l'application utilisent les fonctions de l'intergiciel pour communiquer entre elles de façon transparente, et l'intergiciel utilise les services de base du système d'exploitation pour fournir les fonctions de communication de haut niveau.

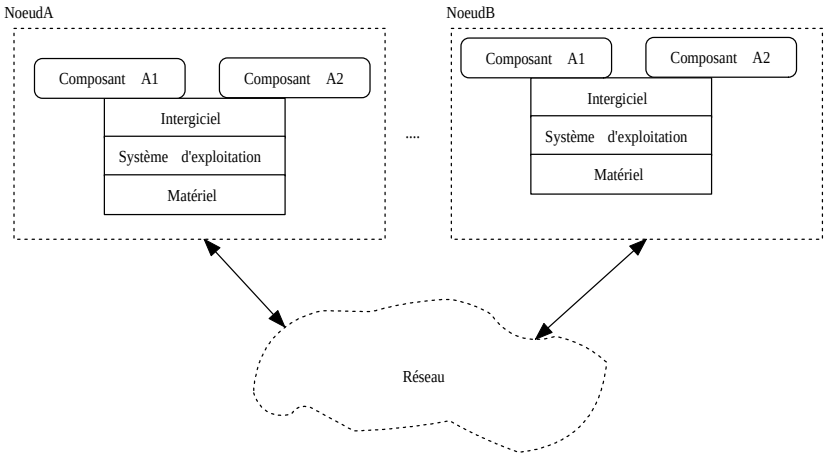

FIG. 2.1 – Architecture d'une application répartie utilisant un intergiciel

Les intergiciels peuvent être classés suivant : la charge requise, le paradigme de communication ou la représentation du contexte. La charge requise définit les ressources requises par

l'intergiciel pour permettre aux applications de s'exécuter. L'intergicicel peut donc être lourd (*heavyweight*) ou léger (*lightweight*). Le paradigme de communication inclut la communication synchrone et la communication asynchrone. Enfin, le contexte d'exécution peut être soit exposé aux application (*awareness*), soit caché par l'intergiciel (*transparency*).

2.4.2 Intergiciels orientés composant

Les intergiciels orientés composant sont une extension des intergiciels orientés objet [78] [74]. Ces derniers ont permis de masquer les problèmes de répartition et d'hétérogénéité des plates-formes, ce qui a facilité la création d'applications réparties. En effet, grâce à la notion d'interface et de langage IDL (*Interface Description Language*) qui permettent de décrire les fonctionnalités d'un objet, il est possible de projeter les interfaces de chaque objet vers plusieurs langages de programmation. Cependant les intergiciels orientés objet comportent certaines limitations. Tout d'abord, les connexions entre objets sont enfouies au coeur des objets et ne peuvent pas être facilement configurées par des architectes externes. Ensuite, l'utilisation des services, comme les services CORBA par exemple, doivent être implantés dans le corps des objets applicatifs et ainsi sont mélangés au code fonctionnel. Enfin, la phase de déploiement des objets n'est pas prise en compte par l'intergiciel. Pour corriger ces défauts, les intergiciels orientés objet ont progressivement évolués d'une approche orientée objet vers une approche orientée composant.

Les intergiciels orientés composant voient une application comme une collection de composants logiciels indépendants et interconnectés entre eux à l'aide de ports ou de canevas de conception [43]. Selon Skyperski [102], un composant peut être défini comme $\ll$ une unité de composition avec des interfaces contractuellement spécifiées et des dépendances explicites sur son contexte. Un composant peut être déployé indépendamment des autres composants et peut être sujet à une composition par des parties tiers $\gg$. L'objectif principal de l'utilisation des composants consiste à faciliter la réutilisation de code sous la forme de composants logiciels et de réduire le développement des applications, en prônant le développement par assemblage plutôt que le développement à partir de zéro. Grâce à une description explicite des services offerts et requis, les composants peuvent facilement être distribués et intégrés dans d'autres applications.

Il existe plusieurs solutions d'intergiciels orientés composant, on peut citer les EJB (Enterprise Java Bean) [55] de Sun, CCM (Corba Component Model) [79] de l'OMG (Object Management Group) et plus récemment Fractal [42] d'ObjectWeb.

EJB [55] représente un canevas de composants serveurs pour concevoir des applications réparties trois tiers. En effet le modèle de composant EJB distingue trois types de composants : session, entité et à message. Au sein des applications, les EJB de type session (composant à état temporaire) et entité (composant à état persistant) sont accessibles à travers une communication synchrone basée sur java RMI [74]. Les EJB à messages (composant sans état) permettent d'intégrer un service d'événements dans les EJB en utilisant l'API JMS [59].

CCM [79] est le modèle de composant de l'OMG qui fait partie de la norme CORBA3. CCM représente un modèle de référence fortement inspiré des EJBs. Contrairement à EJB qui est purement un canevas de composants serveurs, un composant CCM peut être un client ou un serveurs. La spécification CCM définit quatre modèles :

– le modèle abstrait définit les caractéristiques d'un composant CCM. Un composant est caractérisé par des ports classés en synchrones (facettes et réceptacles) et asynchrones (sources et puits d'évènements), et des attributs pour la configuration (cf. figure 2.2). Ces caractéristiques sont décrites dans le langage IDL3, une extension du langage OMG IDL spécifié dans CORBA2. La spécification CCM définit quatre catégories de composants : service, session, processus et entité. Les composants service sont des composants sans état et sans identité. Leur durée de vie correspond à la durée de traitement de l'invocation d'une opération. Les composants session ont un état volatile et une identité non persistante. La durée de vie d'un tel composant est liée à la session d'utilisation de ce dernier. Enfin, l'état des composants processus et entité est persistant. La différence entre les composants les composants processus et entités réside dans l'utilisation de clés primaire pour l'identification des instances entités (sans clés pour les composants processus). En outre, la gestion de la persistance est visible pour le client dans le cas des composants entité et transparente dans le cas des composants processus,
– le modèle de programmation offre aux développeurs de composants le moyen décrire l'implantation d'un composant. Ainsi, CCM définit le langage CIDL (*Component Implementation Definition Language*) utilisé pour décrire la structure d'implantation d'un composant CCM,
– le modèle de déploiement définit un processus qui permet d'installer une application CCM de manière simple et automatique sur un réseau de machines hétérogènes,
– le modèle d'exécution qui définit l'environnement d'exécution des instances de composant représentées par le conteneur (cf. section 2.4.3).

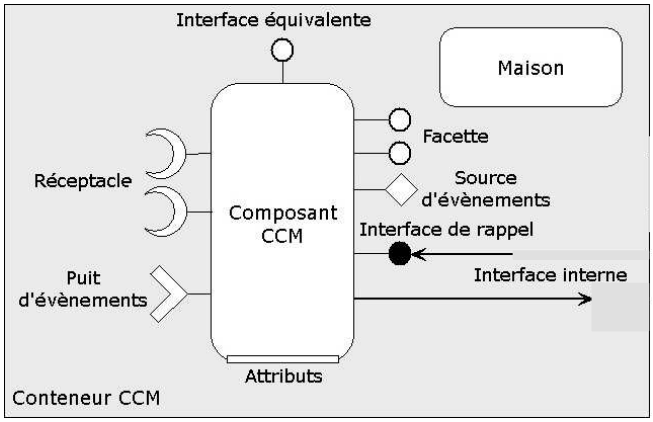

FIG. 2.2 – Modèle abstrait du composant CCM

La spécification CCM est réalisée par plusieurs implantations telles que OpenCCM [80], MicoCCM [73] et Qedo [86].

Le modèle de composant Fractal [42] [16] permet la définition, la configuration et la reconfiguration dynamique des composants. L'objectif de Fractal consiste à fournir une grande modularité et des possibilités d'extension. Contrairement à CCM et EJB, le modèle de composants Fractal est récursif : un composant est de type primitif ou composite. Dans ce dernier cas, le composant correspond à un assemblage d'autres composants primitifs ou composites. Comme CCM, Fractal

offre des interfaces client et serveur. Ainsi, une liaison Fractal représente une connexion entre deux composants.

La plupart des intergiciels orientés composants permettent de séparer le code fonctionnel du code extra-fonctionnel de l'application en se basant sur le principe de séparation des préoccupations [67] [82]. Ce principe consiste à dissocier la programmation des différentes préoccupations composant un programme de manière à faciliter son développement et à rendre son code plus lisible et plus compréhensible. Le principe de séparation des préoccupations peut être utilisé pour séparer la gestion de l'adaptation du code métier de l'application dans le but de faciliter le développement des applications sensibles au contexte.

2.4.3 Techniques d'adaptation

Il existe plusieurs techniques basées sur le principe de séparation des préoccupations qui permettent à une application de s'adapter ou à être adaptée aux variations pertinentes du contexte. Nous décrivons dans cette section les techniques les plus connues.

La réflexivité

Le concept de réflexivité a été introduit par Smith dans sa thèse de doctorat en 1982 [98]. Selon l'auteur, un programme peut posséder une représentation de lui-même, et peut s'en servir pour raisonner sur lui-même. Nous présentons dans cette section comment la réflexivité peut être mise à profit pour la gestion de l'adaptation.

La réflexion permet de distinguer ce que fait un objet (écriture dans une base de données par exemple) de comment il le fait (en utilisant des transactions par exemple). La séparation des préoccupations est alors définie entre le niveau de base, qui représente l'algorithme du domaine de l'application, et le méta-niveau, qui contrôle les aspects associés au niveau de base. Les appels de méthodes dans les langages à méta-objets ne sont pas exécutés directement sur les objets cibles. En effet, comme l'illustre la figure 2.3 tirée de [65], cette invocation met en collaboration l'objet et son méta-objet. Tout d'abord, l'appel est réifié, c'est-à-dire transformé en un objet manipulable par le méta-objet. Puis, le méta-objet exécute le processus de réflexion en invoquant une méthode qui effectue l'appel demandé sur l'objet de base. Enfin, le résultat est transmis au client. Cette technique permet à un système de changer son comportement ou son architecture [40] pour s'adapter en décrivant les actions d'adaptation dans le méta-objet.

La programmation par aspect

La Programmation Orienté Aspect (POA) [61] est un nouveau paradigme de programmation défini par la société Xerox en 1996, c'est une nouvelle manière de structurer et de programmer les applications. Cette technique est un représentant connu pour la mise en œuvre de la séparation des préoccupations.

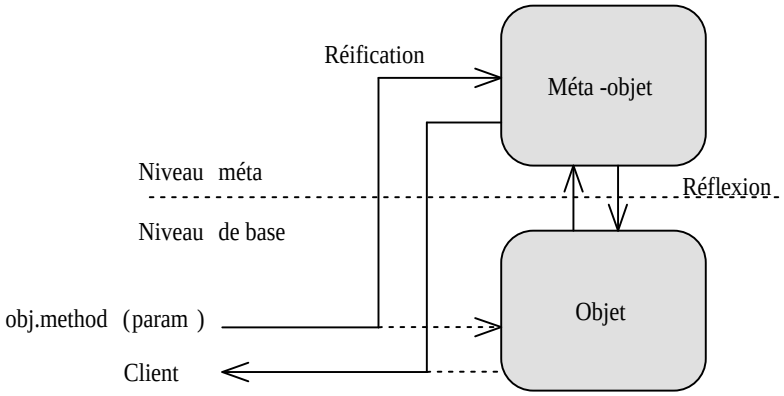

FIG. 2.3 – Principe de fonctionnement des langages à méta-objets

Le principe de la POA consiste à structurer chaque applications en deux parties : un programme de base qui définit les préoccupations fonctionnelles de l'application (sous forme de classes), et un ou plusieurs programmes séparés, écrits dans des langages éventuellement spécialisés, qui définissent les préoccupations extra-fonctionnelles à associer à l'application. La coordination entre les deux parties se fait par des tisseurs d'aspects (*aspect weaver*) dont le rôle consiste à greffer l'ensemble des aspects sur l'ensemble des préoccupations fonctionnelles de l'application à des points de jonctions choisis par le développeur de l'application. L'opération de tissage peut être faite à la compilation ou à l'exécution du programme. Les tisseurs statiques s'appliquent à la compilation du programme, ils prennent en entrée un ensemble de classes et un ensemble d'aspects pour fournir un programme compilé augmenté d'aspects. Les tisseurs dynamiques sont, quant à eux, capables d'appliquer les aspects dynamiquement, pendant l'exécution du programme. Leur principal atout est leur capacité à ajouter, supprimer ou modifier les aspects à chaud pendant l'exécution du programme.

La programmation par aspect peut être utilisée comme technique d'adaptation en considérant l'adaptation comme un aspect.

Le paradigme composant/conteneur

Les intergiciels orientés composant permettent de séparer le code fonctionnel et le code extra-fonctionnel. Cette séparation est faite au moment de la conception des composants qui constituent l'application. L'architecture logique d'un composant est structurée en deux parties, comme l'illustre la Figure 2.4. La première partie, implémente les parties fonctionnelles (≪ contenu ≫ sur la figure 2.4), représente le code métier du composant, alors que la seconde partie, appelée partie extra-fonctionnelle (≪ conteneur ≫ sur la figure 2.4), gère les propriétés extra-fonctionnelles du composant.

L'objectif de chaque conteneur consiste à fournir un environnement d'intégration aux instances de composants, ce qui leur permet de gérer les services (extra-fonctionnels) que le composant utilise de manière transparente. Le conteneur communique avec la partie fonctionnelle du com-

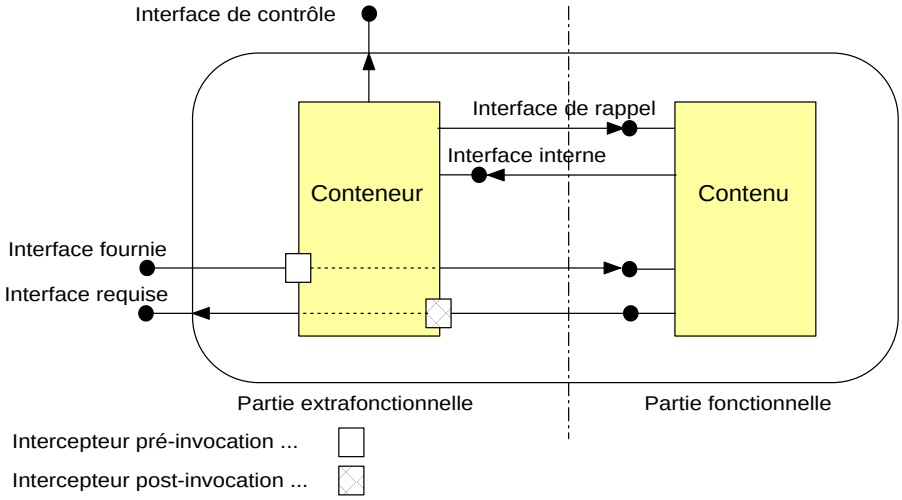

FIG. 2.4 – Structure logique d'un composant

posant à travers des interfaces. Les interfaces internes, offertes par le conteneur et utilisées par le développeur du composant aident à l'implantation du comportement du composant. Les interfaces de rappel, offertes par le composant et utilisées par le conteneur sont disponibles pour la personnalisation des préoccupations extra-fonctionnelles.

Le conteneur contrôle toutes les interactions du composant avec le monde extérieur, ce contrôle est effectué à l'aide de contrôleurs. Dans la plupart des modèles de composants, le conteneur offre au minimum deux objets de contrôle, un intercepteur pré-invocation et un intercepteur post-invocation. L'intercepteur pré-invocation est un objet de contrôle qui offre les mêmes interfaces que le composant. L'intercepteur post-invocation est un objet de contrôle interne. Le conteneur peut contenir d'autres objets de contrôle, chaque objet de contrôle peut représenter un accès vers un service extra-fonctionnel de l'intergiciel ou bien réalise un service que le conteneur utilise dans la gestion des composants. Ces services peuvent être interrogés lorsque le composant reçoit ou émet des invocations.

Dans les applications orientés composants, l'adaptation peut être considérés comme un service extra-fonctionnel. Les conteneurs des composants CCM et EJB fournissent un accès aux services intergiciels suivants : transaction, sécurité, persistance, distribution et messagerie. Mais, ils n'offrent pas le moyen d'intégrer de nouveaux services extra-fonctionnels à l'intergiciel, comme l'adaptation par exemple. Fractal offre peu de services extra-fonctionnels de base, mais laisse la totale liberté quant à l'ajout de nouveaux services extra-fonctionnels. En revanche, Fractal ne définit aucune règle ni d'outil de déploiement particulier pour le faire. Dans le cadre de CCM, des travaux de recherche tournent autour de la définition d'un conteneur qui offre la possibilité d'intégrer facilement des services extra-fonctionnels grâce à une architecture en trois couches : une couche d'interception, une couche de coordination et une couche de contrôle. Un premier prototype dit ouvert a été proposé dans le cadre de la plate-forme OpenCCM [80], ce prototype a été raffiné dans le cadre du projet IST COACH [31], pour aboutir à un modèle de conteneur extensible (ECM, pour Extensible Container Model) [104].

moteur d'inférence

Un moteur d'inférence est un programme qui effectue des déductions logiques d'un système à partir d'une base de connaissance et d'une base de règles. Les règles sont utilisées pour manipuler les connaissances et aboutir à des conclusions.

Dans les systèmes sensibles au contexte, la base de connaissance peut être représentée par les informations de contexte observées ou interprétées. Par conséquent, les règles manipulent les informations de contexte pour détecter des situations pertinentes. Lors de la détection d'une situation pertinente, le moteur d'inférence exécute les actions d'adaptation appropriées. Parmi les implémentations existantes de moteurs d'inférences, nous pouvons citer Flora 2 [3] et Jena 2 [56].

2.4.4 Synthèse

Dans cette section nous avons décrit les caractéristiques des intergiciels et le rôle qu'ils ont joué dans la gestion de l'hétérogénéité des réseaux et la distribution des applications. Par la suite, nous avons étudié les intergiciels orientés composant qui offrent le moyen de créer des applications par composition (en introduisant la notion de composant). Enfin, nous avons décrit un ensemble de techniques d'adaptation qui utilisent le principe de séparations des préoccupations. Ces techniques permettent de faciliter la gestion de l'adaptation en séparant le code fonctionnel de l'application de son code extra-fonctionnel nécessité par l'adaptation.

2.5 Étude d'intergiciels sensibles au contexte

Le développement et l'exécution des applications distribuées sensibles au contexte peuvent être effectués à l'aide des intergiciels sensibles au contexte. Le rôle de ces intergiciels est de faciliter la sensibilisation au contexte, et cela en intégrant dans leurs architectures des entités qui se chargent de gérer le contexte et l'adaptation de l'application.

L'objectif de cette section est de présenter et d'évaluer des propositions existantes pour le développement d'applications sensibles au contexte. Chacune des propositions étudiées est décrite puis évaluée. Cette évaluation est faite en fonction des éléments fonctionnels de la sensibilité au contexte gérés par l'intergiciel, le mode d'interaction de l'intergiciel avec l'application (mode requête ou mode notification), les types d'adaptation gérés par l'intergiciel, les techniques d'adaptation utilisées à cet effet, la modélisation du contexte fournie aux concepteurs et la possibilité d'étendre l'intergiciel pour interagir avec de nouveaux capteurs.

Nous commençons cette étude par la description des intergiciels pour la sensibilité au contexte à savoir l'intergiciel SOCAM (cf. section 2.5.1) et l'intergiciel CASS(cf. section 2.5.2). Puis, nous décrivons des intergiciels sensibles au contexte à savoir l'intergiciel CARISMA (cf. section 2.5.3), l'intergiciel CORTEX (cf. section 2.5.4), l'intergiciel RCSM (cf. section 2.5.5), l'intergiciel K-component (cf. section 2.5.6) et l'intergiciel SAFRAN (cf. section 2.5.7).

2.5.1 SOCAM

Description

SOCAM (*Service Oriented Context-Aware Middleware*) [45] est un intergiciel pour la sensibilité au contexte qui fournit une infrastructure pour la création de services sensibles au contexte. Il offre aux concepteurs de services le modèle d'ontologie CONON pour la description du contexte (cf. section 1.6.4).

Comme l'illustre la figure 2.5, l'architecture de l'intergiciel SOCAM est constituée d'un ensemble de composants qui lui permettent de gérer automatiquement la collecte et l'interprétation du contexte. Le *Context provider* se charge de collecter le contexte auquel chaque service est sensible et de le fournir au *Context interpreter*. Il existe deux types de *Context provider* : les *Context providers externes* et les *Context providers internes*. Les *Context providers externes* se chargent de collecter le contexte à partir de sources de données externes à la machine comme la température de la pièce par exemple. Les *Context providers internes* se chargent de collecter des données sur l'état interne de la machine comme la valeur de la bande passante par exemple. Le *Context interpreter* est un composant qui fournit un service de raisonnement sur les informations de contexte dans le but de les interpréter. Le composant *Service Locating Service* est équivalent aux pages blanches. Il est utilisé par le *Context Interpreter* et le *Context Provider* pour s'enregister et par les services sensibles au contexte pour trouver des services d'interprétation du contexte et de collecte des données.

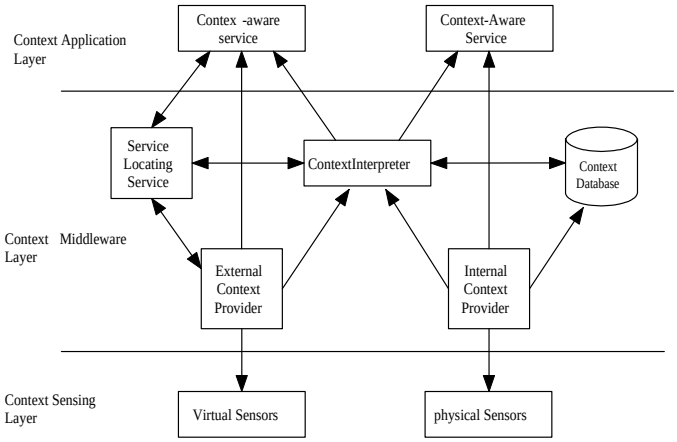

FIG. 2.5 – Architecture de l'intergiciel SOCAM

Les services sensibles au contexte sont soit des applications soit des agents. Ils utilisent deux catégories de contexte observable : le contexte de bas niveau et/ou le contexte interprété. Ils adaptent leurs comportements selon les valeurs du contexte observées. Pour cela, chaque service s'enregistre auprès du *Context provider* pour acquérir le contexte auquel il est sensible (contexte pertinent) et fournit au *Context interpreter* des règles d'interprétation qui permettent à l'intergiciel d'interpréter le contexte. A la réception d'une observation du contexte le service se charge d'analyser les données reçues et le cas échéant d'appliquer l'adaptation nécessaire.

Évaluation

L'intergiciel SOCAM décharge les développeurs de services sensibles au contexte de la tâche de collecte et d'interprétation des données, il leur fournit un modèle pour décrire le contexte à observer. Mais les développeurs doivent implémenter l'analyse et l'adaptation de chaque service. Cet intergiciel n'offre aucun moyen aux développeurs pour choisir les capteurs que l'intergiciel doit utiliser afin de collecter le contexte.

2.5.2 CASS

Description

CASS (*Context-Awareness Sub-Structures*) [39] est un intergiciel pour la sensibilité au contexte qui permet aux développeurs de gérer les applications sensibles au contexte. Cette gestion concerne l'interaction avec les capteurs pour collecter les informations de contexte, la sauvegarde des données collectées dans une base de données et leur interprétation.

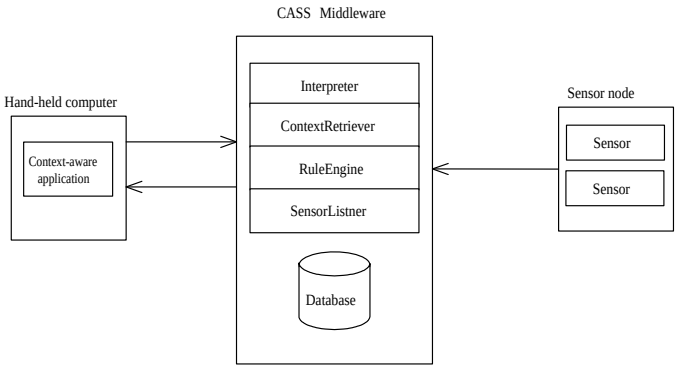

FIG. 2.6 – Architecture de l'intergiciel CASS

Comme l'illustre le figure 2.6, l'intergiciel CASS est structuré en un ensemble de composants pour gérer le contexte. Le *SensorLister* se charge d'interagir avec des noeuds de capteurs pour collecter le contexte et de le sauvegarder dans la base de données. Le composant *Interpreter* se charge de calculer des informations de contexte de haut niveau en utilisant le *RuleEngine* et le *ContextRetriever* se charge de retrouver les observations du contexte dans la base de données. L'intergiciel se charge de détecter le changemement du contexte observé et de notifier les applications sensibles à ces contextes. Chaque application se charge d'analyser ce nouveau contexte et d'appliquer l'adaptation nécessaire.

Évaluation

L'intergiciel CASS facilite l'interaction avec les capteurs et l'interprétation des données en intégrant dans son architecture des composants qui se chargent d'effectuer ces tâches. Les applications sensibles au contexte doivent s'enregistrer auprès de cet intergiciel pour être notifiées

à chaque fois qu'une variation de contexte a été détectée. Le développeur de l'application se charge de programmer l'analyse de ces données ainsi que l'adaptation de l'application. Ce dernier ne peut pas choisir les capteurs que l'intergiciel doit utiliser pour collecter les informations de contexte.

2.5.3 CARISMA

Description

CARISMA (Context Aware Reflexive Middleware for Mobile Applications) [17] est un intergiciel réflexif qui permet aux développeurs de créer des applications sensibles au contexte. Cet intergiciel offre une grammaire basée sur XML (cf. section 1.6.2) pour modéliser les situations pertinentes et les actions d'adaptation associées. Cela permet à l'intergiciel d'adapter les services de l'application lors des variations pertinentes de l'environnement. Des API réflexives permettent de changer le profil de l'application au cours de son exécution.

L'intergiciel est vu par l'application comme un fournisseur de services dynamiquement configurable. Cette configuration est effectuée grâce au profil de l'application décrit dans le modèle et récupérable par le mécanisme de réification [23]. Un profil d'application est constitué de deux parties distinctes [18], la partie réactive et la partie proactive. La partie réactive permet au développeur de modéliser les situations pertinentes devant être notifiées à l'application pour qu'elle réagisse. La partie proactive permet au développeur de modéliser les adaptations proactives qui représentent l'adaptation des services de l'application en fonction du contexte. Chaque service que l'application veut adapter peut être fourni avec différentes politiques, chaque politique étant associée à une situation pertinente.

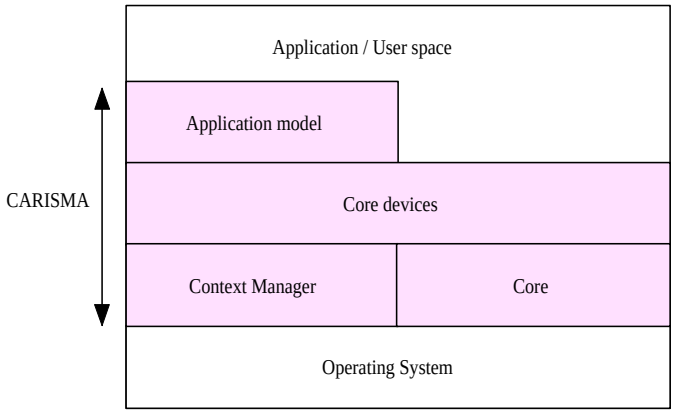

FIG. 2.7 – Architecture de l'intergiciel CARISMA

Comme l'illustre la figure 2.7, l'intergiciel CARISMA est constitué de cinq entités qui se chargent de gérer les applications sensibles au contexte. L'entité *Core* fournit des fonctionnalités de base comme la gestion de la communication dans un environnement distribué et la découverte de service. L'entité *Context Manager* se charge d'interagir avec les capteurs pour collecter

les informations de contexte auxquels chaque application est sensible (contexte pertinent), et de surveiller le changement de leurs valeurs. L'entité *Core Services* se charge de gérer les informations de contexte en activant l'entité *Context Manager*. L'entité *Context Manager* se charge de collecter le contexte, d'intercepter les invocations de l'application, et de les adapter. L'entité *Application Model* offre un support pour créer et exécuter des applications sensibles au contexte.

Lorsque plusieurs applications s'exécutent dans une instance de l'intergiciel, les profils de ces applications peuvent entrer en conflit. La détection et la résolution de ces conflits ne peuvent pas se faire statiquement, puisque les profils de l'application peuvent être changés par l'utilisateur en cours d'exécution de l'application. CARISMA propose une approche dynamique de résolution de conflit à chaque fois qu'ils se produisent. Cette approche est calquée sur un modèle économique de vente aux enchères. Les différentes applications sont considérées comme des consommateurs qui sont en concurrence pour obtenir les services fournis par le producteur (l'intergiciel) [21].

Évaluation

L'intergiciel CARISMA utilise la réflexivité comme technique d'adaptation pour faciliter le développement des applications sensibles au contexte. Cet intergiciel prend en compte la collecte du contexte et l'analyse des données collectées. Il se charge de notifier les applications lors de la détection des situations pertinentes associées aux adaptations réactives, et d'adapter les appels de services lors de la détection des situations pertinentes associées aux adaptations proactives. CARISMA utilise le profil de l'application pour prendre les décisions de notification et d'adaptation appropriées. Cet intergiciel offre la possibilité aux utilisateurs de l'application de changer son profil en cours d'exécution grâce à des APIs réflexives

L'intergiciel CARISMA facilite le développement des applications sensibles au contexte. Cependant, il n'offre aucun moyen aux développeurs de décrire ni de choisir les capteurs à utiliser pour collecter le contexte. L'intergiciel CARISMA interagit avec des capteurs qui ne peuvent collecter que des informations locales à la machine où est installée l'application, ce qui limite considérablement le type d'applications à développer à l'aide de cet intergiciel. CARISMA utilise des informations de contexte de bas niveau. Aucun mécanisme d'interprétation n'est intégré à l'intergiciel.

2.5.4 CORTEX

Description

CORTEX (CO-operating Real-time senTient objects) [105] est un intergiciel qui facilite le développement des applications sensibles au contexte. L'implémentation de cet intergiciel est basée sur OpenCOM [29], qui est un intergiciel réflexif basé sur la technologie COM [10]. Comme le montre la figure 2.8 tirée de [4], l'architecture de CORTEX est structurée en quatre cadres de composants :

– le cadre de composants *Context* permet de collecter et de gérer les informations de contexte. Il est basé sur le modèle des objets sensibles [41]. Les objets sensibles (cf. figure 2.9) sont définis comme des entités capables de consommer des événements depuis des sondes (*sensors*) et de produire des événements réfléchis vers des déclencheurs (*actuators*). L'intelligence du cadre de composants *Context* est due à sa capacité de stockage de ces informations dans une mini base de données et à son moteur d'inférence,
– le cadre de composants *Publish/Subscribe* basé sur un modèle d'événements, appelé STEAM [72] permet la dissémination des événements dans un réseau ad-hoc,
– le cadre de composants *Resource Management* permet de répartir les tâches par rapport aux ressources disponibles,
– le cadre de composants *Service discovery* qui donne une solution aux problèmes posés par l'hétérogénéité du réseau, puisqu'il permet de découvrir des services énoncés par différents protocoles de découverte de service.

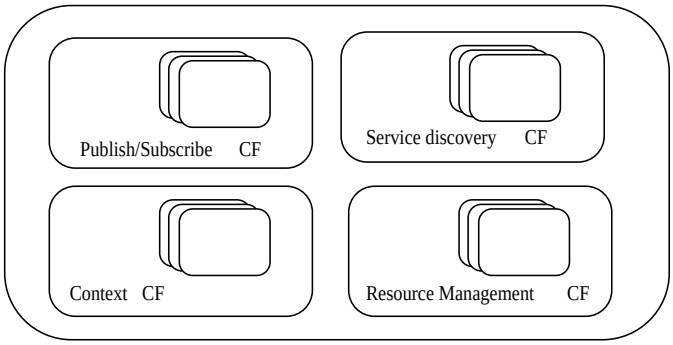

FIG. 2.8 – Architecture de l'intergiciel CORTEX

Le cadre de composants *Context* est basé sur le modèle des objets sensibles. Un objet sensible est composé de trois entités (cf. figure 2.9) qui fournissent respectivement des fonctions de collecte de contexte (*Sensory Capture*), de représentation de contexte (*Context Hierarchy*) et de raisonnement sur les informations de contexte (*Inference Engine*). La fonction de collecte de contexte interagit avec des capteurs à travers des évènements. La fonction de représentation de contexte permet de modéliser le contexte d'une manière hiérarchique en décrivant les attributs de chaque contexte. Dans le cadre de CORTEX, les attributs les plus importants d'un contexte sont : les situations pertinentes auxquelles le composant est sensible et l'ensemble des règles actives dans ces situations. La fonction de raisonnement contient des règles qui dictent les réactions de l'objet sensible dans des situations particulières, ces règles sont décrites à l'aide du langage CLIPS [30].

La création d'une application sensible au contexte avec CORTEX consiste à utiliser le cadre de composant *Context* pour décrire les objets sensibles qui constituent l'application. Les capteurs sont représentés par des objets ayant une interface spécifique. Ils fournissent des évènements aux objets sensibles. Les *actuators* représentent des objets qui se chargent d'adapter l'application. Ils sont des consommateurs d'évènements fournies par les objets sensibles.

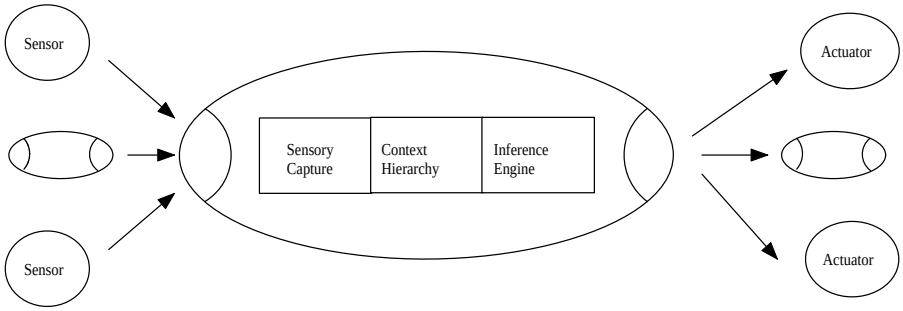

FIG. 2.9 – Structure de l'objet sensible

Évaluation

CORTEX facilite la création d'applications sensibles au contexte en fournissant des cadres de composants et une interface graphique qui aident le développeur dans cette tâche. L'interface graphique fournit un ensemble de *sensors* et d'*actuators* prédéfinis qui peuvent être utilisés par les développeurs d'applications.

L'intergiciel CORTEX offre aux développeurs d'applications un modèle hiérarchique pour décrire le contexte et le moyen de choisir les capteurs avec lesquels l'application doit interagir. Il se charge aussi d'adapter le comportement de l'application aux situations pertinentes du contexte à l'aide du moteur d'inférence. Néanmoins, l'interprétation du contexte de haut niveau n'est pas considérée par cet intergiciel.

2.5.5 RCSM

Description

RCSM (*Reconfigurable Context Sensitive Middleware*) [114] est un intergiciel orienté objet qui permet la création et la gestion d'applications sensibles au contexte. Pour faciliter le développement de ce type d'applications, RCSM demande aux développeurs d'écrire une interface sensible au contexte qui décrit les situations pertinentes auxquelles l'application est sensible et les règles d'adaptation de cette application. Cette interface est décrite à l'aide du langage CA-IDL (Context Aware-Interface Description Language) (cf. section 1.6.1). RCSM utilise cette interface pour générer le code d'adaptation de l'application.

Comme nous l'avons décrit dans le chapitre 1, le langage CA-IDL fige les types de contextes observables que l'intergiciel peut surveiller. Comme l'illustre la figure 2.10, ces types sont répartis dans trois catégories de contexte : la catégorie *DeviceSpecificContext* pour les contextes spécifiques à la machine sur laquelle s'exécute l'application, la catégorie *EnvironmentSpecificContext* pour les contextes associés à l'environnement, et la catégorie *UserSpecificContext* pour le profil de l'utilisateur. RCSM considère une application comme un ensemble d'objets sensibles au

contexte. Chaque objet est structuré en deux parties : une interface sensible au contexte (cf. section 1.6.2) et une implémentation indépendante du contexte. L'interface encapsule la description de la sensibilité au contexte, alors que l'implémentation reste indépendante du contexte.

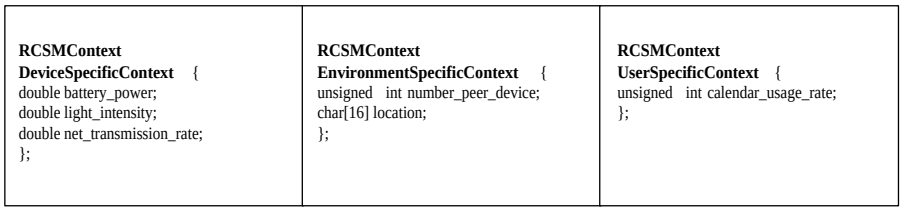

FIG. 2.10 – Liste des contextes considérés par RCSM

La compilation d'une interface sensible au contexte permet à l'intergiciel de générer le code d'adaptation de l'objet auquel l'interface est associée sous la forme de conteneur d'objet. Le conteneur d'objet se charge d'enregistrer l'objet auprès du R-CAP (*Context Aquisition and Processing framework*) pour le contexte auquel il est sensible. Le R-CAP est une entité appartenant à l'intergiciel RCSM qui se charge de collecter le contexte, de l'analyser et de lancer les actions d'adaptation nécessaires quand une situation pertinente a été détectée [101]. À chaque catégorie de contexte est associé un ensemble de capteurs qui se chargent de collecter le contexte auquel l'application est sensible.

Evaluation

L'intergiciel RCSM facilite le développement des applications orientées objet sensibles au contexte en offrant un langage de description d'interfaces sensibles au contexte et en déchargeant le développeur d'application de la tâche de collecte, d'analyse du contexte et d'adaptation de l'application. Cet objectif est atteint grâce à la génération de code d'adaptation à partir des interfaces sensibles au contexte décrites par le développeur d'application. Ces adaptations sont des adaptations comportementales de nature réactives. Cependant, RCSM ne prend pas en compte l'interprétation du contexte et n'offre aucun moyen de choisir, pour chaque contexte, le capteur à partir duquel les données seront collectées. L'ajout ou la suppression d'un capteur nécessite la modification la grammaire du CA-IDL et de la plate-forme RCSM.

2.5.6 K-component

Description

K-Component [38] est un projet développé par l'équipe Distributed System Group au Trinity College de Dublin. C'est un intergiciel orienté composant destiné spécialement au développement des applications sensibles au contexte.

L'intergiciel K-Component offre un langage de description d'interface (K-IDL) qui est un sous-ensemble du langage de définition d'interfaces (IDL-3) développé par l'OMG, pour la description

du code métier du composant, ainsi qu'un langage de description des contrats d'adaptations (*Adaptation Contracts Declarative Language*) pour décrire les règles d'adaptation de l'application.

L'adaptation d'une application au-dessus de K-Component consiste à changer la structure de cette application. Pour cela, l'intergiciel construit un graphe typé où les noeuds représentent les composants de l'application et les arcs représentent les connexions entre ces composants. K-Component utilise la réflexivité architecturale afin d'effectuer des transformations sur ce graphe. Ces transformations consistent à ajouter des composants, à supprimer des composants ou à changer les connexions entre les composants. Les règles d'adaptations sont décrites dans le contrat d'adaptation et déclenchées par l'intergiciel à la réception de certains événements CORBA.

Évaluation

L'intergiciel K-Component adapte l'application en utilisant la technique de réflexivité. Cette adaptation constitue le seul élément fonctionnel de la gestion du contexte considéré par cet intergiciel. Par conséquent, le développeur doit écrire le code d'interaction avec les capteurs pour observer le contexte, le code d'interprétation et d'analyse des données observées ainsi que l'invocation des évènements qui permettent à l'intergiciel de réagir aux situations pertinentes en adaptant la structure de l'application.

2.5.7 SAFRAN

Description

SAFRAN (Self-Adaptive Fractal CompoNents [32] est un système qui étend le modèle de composant Fractal pour faciliter le développement d'applications adaptatives. SAFRAN considère l'adaptation d'une application à son contexte d'exécution comme un aspect. Cet aspect doit être développé séparément du reste de l'application en utilisant un formalisme spécifique. Il peut ainsi être intégré (ou retiré) dynamiquement du reste de l'application.

SAFRAN propose aux programmeurs d'applications un modèle de développement et des outils qui facilitent la création d'applications adaptatives. L'adaptation considérée par SAFRAN est la reconfiguration structurelle de l'application.

La figure 2.11 tirée de [32], illustre l'architecture de SAFRAN. SAFRAN est constituée de deux parties distinctes : la première partie est une extension du modèle Fractal pour permettre le tissage dynamique des politiques d'adaptation dans des composants Fractal et la programmation de ces politiques à l'aide d'un language dédié. Cette extension est consituée de l'entité *Méta-composants* et de l'entité *Contraintes métier*. La deuxième partie est le système SAFRAN qui premet la prise en compte de la collecte du contexte, de son analyse et de l'adaptation de l'application. Ce système est constitué de l'entité *WILDCAT*, de l'entité *Adaptation Controller* et de l'entité *Fscript*.

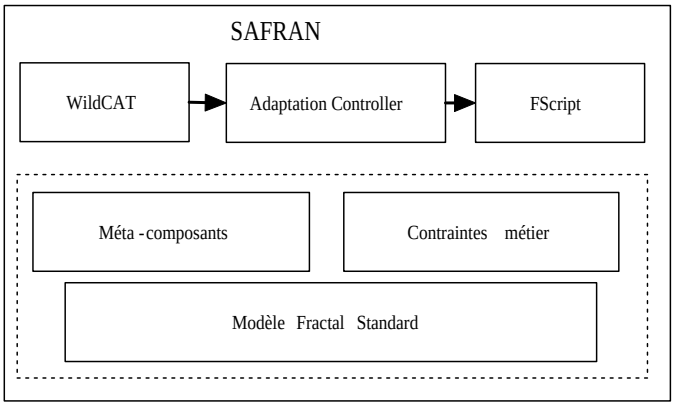

FIG. 2.11 – Architecture du système SAFRAN

L'entité *Méta-Composants* réalise des adaptations d'un composant Fractal de façon non-anticipée. L'entité *Contraintes métier* permet de définir des contraintes architecturales spécifiques à une application, au-delà de ce que peut exprimer Fractal. L'entité *WildCAT* est un *framework* qui permet de modéliser le contexte et de l'observer. Ce *framework* offre deux modes d'interaction avec l'application : le mode requête et le mode notification. Le contexte d'exécution est organisé en un ensemble de domaines contextuels qui représente chacun un aspect spécifique du contexte. Chaque domaine contextuel est modélisé sous la forme d'une arborescence de ressources, chacune d'elle étant décrite par un ensemble d'attributs. *WildCat* fournit une syntaxe inspirée de celle des URI pour désigner les ressources et les attributs. L'entité *Adaptation Controller* permet de gérer l'adaptation de l'application et fournit un langage dédié pour programmer les politiques d'adaptation sous la forme de règles réactives. L'entité *FScript* est un langage qui permet de spécifier les politiques de reconfigurations. Le framework *WildCat* fournit des bibliothèques de sondes génériques et réutilisables, ce qui permet d'ajouter des capteurs à l'intergiciel sans grandes difficultés.

Évaluation

Le système SAFRAN facilite le développement des applications sensibles au contexte en utilisant les aspects comme technique d'adaptation. Ce système offre une modélisation hiérarchique du contexte et donne la possibilité aux développeurs de choisir les capteurs de collecte du contexte. SAFRAN décharge le développeur d'application de la tâche de collecte du contexte, de son analyse et de l'adaptation de l'application. Mais, laisse l'interprétation du contexte à la charge du développeur d'application. L'une des limites de SAFRAN réside dans le fait qu'il ne permet pas l'adaptation des applications distribuées.

2.6 Synthèse et Évaluation

Dans ce chapitre, nous avons étudié la sensibilité au contexte et les intergiciels sensibles au contexte. Pour cela, nous avons tout d'abord donné une définition de la sensibilité au contexte dans la section 2.2, puis nous avons décrit les éléments fonctionnels d'un système sensible au contexte. Ces éléments permettent au système de modéliser, collecter, interpréter et analyser le contexte afin d'appliquer l'adaptation nécessaire à ses variations pertinentes.

Le développement d'applications sensibles au contexte est une tâche difficile à mettre en œuvre surtout si le concepteur doit implémenter tous les éléments fonctionnels décrits dans la section 2.3. L'utilisation des services d'un intergiciel permet au concepteur de l'application de déléguer certaines tâches à l'intergiciel et de faciliter ainsi le développement de ce type d'applications.

Un intergiciel est une couche logicielle qui se situe entre le système d'exploitation et l'application. Il fournit des solutions réutilisables aux problèmes récurrents des applications réparties. Dans la section 2.4, nous avons défini le rôle des intergiciels. Puis, nous avons décrit les caractéristiques des intergiciels orientés composant. Ensuite, nous avons étudié les techniques d'adaptations qui peuvent être utilisées par les intergiciels pour gérer l'adaptation.

Dans la section 2.5, nous avons présenté des intergiciels dont l'objectif consiste à faciliter le développement des applications sensibles au contexte. Certains de ces intergiciels utilisent les techniques d'adaptation étudiées dans la section 2.4.3 pour atteindre cet objectif. Néanmoins, tous les éléments fonctionnels relatifs à la gestion de la sensibilité au contexte, que nous avons décrit dans la section 2.3, ne sont pas pris en compte par ces intergiciels. De ce fait, les tâches restantes sont laissées à la charge du développeur.

Le tableau 2.1 résume l'étude effectuée sur les intergiciels SOCAM, CASS, CARISMA, CORTEX, RCSM, K-Component et SAFRAN concernant les éléments fonctionnels de la sensibilité au contexte gérés automatiquement par ces intergiciels, et les tâches laissées à la charge du développeur.

La modélisation des informations de contexte est la première étape dans le processus de création d'applications sensibles au contexte. La plupart des intergiciels étudiés offrent des modèles pour décrire le contexte. Mais, Ces modèles ne permettent pas tous de décrire une large panoplie d'informations associées à la gestion du contexte. Le modèle CA-IDL offert par RCSM par exemple, ne permet de décrire que les situations pertinentes auxquelles l'application est sensible ainsi que les politiques d'adaptation, mais, n'offre aucun moyen de décrire les capteurs ni les règles d'interprétation.

La description des capteurs offre au concepteur de l'application la possibilité d'associer à chaque contexte le capteur à partir duquel les données sont collectées, alors que les règles d'interprétations lui permettent de spécifier comment les contextes de haut niveau peuvent être déduits à partir des autres informations de contexte. Il nous semble nécessaire de proposer un modèle de description du contexte qui permet de décrire non seulement le contexte auquel l'application est sensible, mais aussi les règles d'interprétation et les politiques d'adaptation de l'appli-

cation. Ce qui permettrait d'automatiser complètement le processus de gestion des applications sensibles au contexte.

Selon le tableau 2.1, l'analyse du contexte n'est pas toujours offerte par les intergiciels sensibles au contexte. Les intergiciels CORTEX, K-Component et CASS laissent cette tâche à la charge du développeur de l'application. L'analyse du contexte consiste à détecter les situations pertinentes qui nécessitent l'adaptation de l'application.

L'adaptation d'une application est laissée à la charge du développeur si elle utilise les services d'un intergiciels pour la sensibilité au contexte comme l'intergiciel SOCAM et l'intergiciel CASS. S'il s'agit d'un intergiciel sensible au contexte comme CARISMA, CORTEX, RCSM, K-Component et SAFRAN, cette tâche est gérée par l'intergiciel. Les adaptations gérées par les intergiciels étudiés dans ce chapitre consistent à changer le comportement de l'application de manière réactive comme c'est la cas pour les intergiciels CARISMA, CORTEX, et RCSM, à changer le comportement de l'application de manière proactive comme c'est le cas pour l'intergiciel CARISMA ou bien à changer l'architecture de l'application comme c'est le cas pour les intergiciels K-Component et SAFRAN.

Les travaux décrits dans ce tableau facilitent le développement des applications sensibles au contexte. Mais, laissent certaines tâches de gestion du contexte à la charge du développeur. Pour faciliter encore plus la création de ce type d'applications, nous estimons qu'il est nécessaire que l'intergiciel fournisse aux développeurs un modèle pour décrire le contexte et qu'il utilise ces descriptions pour gérer tous les éléments fonctionnels d'une application sensible au contexte. En ne laissant au développeur que la tâche de décrire le contexte. Dans la partie suivante, nous proposons un tel intergiciel que nous avons appelé CAMidO (Context-Aware Middleware based on Ontology meta-model).

2.7 Conclusion

Les applications mobiles s'exécutent dans un environnement dynamique. Par conséquent, ces applications doivent détecter les changements de l'environnement et adapter leurs comportements en fonction de ces changements. En outre, développer ce type d'applications est une tâche difficile pour l'analyseur, le concepteur et le développeur de l'application. En effet, sans intergiciel, les développeurs doivent non seulement programmer la collecte du contexte et son analyse afin de détecter les variations pertinentes du contexte, mais aussi implémenter l'adaptation de l'application lors de la détection de ces changements.

Les services d'un intergiciel sensible au contexte permettent de faciliter la tâche de développement d'applications distribuées sensibles au contexte. Les travaux existants sur la sensibilité au contexte ont abouti à la création d'intergiciels sensibles au contexte. Ces intergiciels, que nous avons décrits dans la section 2.5, facilitent la création d'applications sensibles au contexte en prenant en compte une partie des éléments fonctionnels associées à la gestion du contexte.

Afin d'aller plus loin dans les possibilités offertes par les intergiciels sensibles au contexte, il nous semble primordial de proposer un intergiciel qui offre d'une part un cadriciel pour le déve-

loppement d'applications sensibles au contexte, et qui offre d'autre part un méta-modèle de description du contexte qui inclut toutes les informations nécessaires à l'automatisation du processus d'adaptation. Cet intergiciel pourra ainsi fournir tous les éléments fonctionnels liés à la sensibilité au contexte en s'appuyant sur les informations fournies par les modèles de contexte. Notre travail consiste à proposer un intergiciel qui s'appuie sur un modèle de contexte pour construire les fonctions de collecte du contexte, de son interprétation et analyse et d'adaptation de l'application. Notre intergiciel exploite le paradigme composant/conteneur pour gérer l'adaptation de l'application par des propriétés non fonctionnelles.

Intergiciel	Type	Modélisation du contexte	Interprétation du contexte	Analyse du contexte	Adaptation des applications	Type d'adaptation	Techniques d'adaptation	Choix des capteurs
CARISMA	Sensible au contexte	XML	Non prise en compte	Prise en compte par l'intergiciel	Prise en compte par l'intergiciel	Adaptation comportementale de nature réactive et proactive	Réflexivité	Non
CORTEX	Sensible au contexte	Modèle hiérarchique	À la charge du développeur	Prise en compte par l'intergiciel	Prise en compte par l'intergiciel	Adaptation comportementale de nature réactive	Moteur d'inférence	Non
RCSM	Sensible au contexte	CA-IDL	Non prise en compte	Prise en compte par l'intergiciel	Prise en compte par l'intergiciel (conteneur d'objets)	Adaptation comportementale de nature réactive	Génération de code d'adaptation	Non
K-Component	Sensible au contexte	ACDL	À la charge du développeur	À la charge du développeur	Prise en compte par l'intergiciel (réflexivité architecturale)	Adaptation structurelle réactive	Réflexivité	Non
SAFRAN	Sensible au contexte	WILDCAT	Non prise en compte	Prise en compte par l'intergiciel	Prise en compte par l'intergiciel (aspects)	Adaptation structurelle réactive	La programmation par aspect	Oui
SOCAM	Pour la sensibilité au contexte	Ontologie (CONON)	Prise en compte par l'intergiciel à l'aide d'un moteur d'inférence	À la charge du développeur	À la charge du développeur	À la charge du développeur		Non
CASS	Pour la sensibilité au contexte	Non	Prise en compte par l'intergiciel	À la charge du développeur	À la charge du développeur	À la charge du développeur		Non

TAB. 2.1 – Tableau comparatif des intergiciels sensibles au contexte

Deuxième partie

CAMidO, une solution pour faciliter le développement d’applications orientées composants sensibles au contexte

Chapitre 3

Types d'adaptation et méta-modèle de description du contexte de CAMidO

3.1 Introduction

La description du contexte auquel l'application est sensible représente la première étape dans le processus de création d'applications sensibles au contexte. Cette description couvre les aspects suivants : caractéristiques du contexte, informations d'interprétations, contextes pertinents pour l'application, situations pertinentes et adaptation.

Dans ce chapitre, nous présentons le méta-modèle de CAMidO (Context-Aware Middleware based on Ontology model) qui permet d'une part de décrire des informations de contexte génériques communes à toutes les applications sensibles au contexte et d'autre part des informations propres à chaque application. Les caractéristiques des intergiciels orientés composant et les types d'adaptation que nous considérons (adaptation comportementale de nature réactive et proactive) ont dirigé la conception de ce méta-modèle.

Dans un premier temps, nous présentons les motivations qui nous ont amené à proposer le méta-modèle de description de contexte de CAMidO ainsi que les objectifs qu'il permet d'atteindre (cf. section 3.2). Puis, nous présentons les informations que ce méta-modèle permet de décrire (cf. section 3.3). Par la suite, nous définissons les types d'adaptation considérés par CAMidO que nous illustrons à travers des applications sensibles au contexte (cf. section 3.4). Enfin, nous décrivons le méta-modèle de CAMidO dans la section 3.5 et nous résumons ce chapitre tout en mettant l'accent sur l'apport du méta-modèle proposé dans la section 3.6.

3.2 Motivations et objectifs

Le travail présenté dans cette thèse a pour but de proposer un méta-modèle et un intergiciel qui facilitent la tâche de création d'applications orientées composant sensibles au contexte. Cet outil se base sur deux principes : le premier principe est la séparation de la description du contexte de la programmation de son utilisation. Le deuxième principe est l'utilisation des services d'un intergiciel dédié à la gestion du contexte et l'adaptation des applications. Dans ce chapitre, nous nous intéressons au premier principe. Pour cela, nous proposons un méta-modèle qui permet de décrire le contexte, sa collecte et son utilisation.

Selon l'étude effectuée dans le chapitre 1 sur la modélisation du contexte, les modèles de description du contexte existants se focalisent plus sur la description du contexte et ses caractéristiques que sur la description de son utilisation. Par conséquent, les intergiciels qui utilisent ces modèles ne peuvent pas automatiser le processus d'adaptation au contexte et laissent cette tâche à la charge du développeur d'applications. De ce fait, malgré les modèles proposés, le processus de développement d'applications sensibles au contexte reste difficile à mettre en œuvre.

L'objectif principal de notre travail consiste à automatiser le processus de gestion des applications sensibles au contexte par un intergiciel depuis la collecte du contexte jusqu'à l'adaptation de l'application. Cette automatisation n'est possible que si l'intergiciel dispose des informations de gestion du contexte propres à chaque application. Pour atteindre cet objectif, nous proposons le méta-modèle de CAMidO qui fournit à l'intergiciel des informations sur le contexte et des informations supplémentaires concernant sa gestion. Comme nous l'avons décrit dans la section 2.3, le processus de gestion d'une application sensible au contexte regroupe la collecte du contexte, l'interprétation et l'analyse des informations collectées ainsi que l'adaptation de l'application lors de la détection d'une situation pertinente. Ces informations permettent l'automatisation du processus de gestion du contexte et d'adaptation de l'application.

Le méta-modèle de CAMidO est basé sur trois ontologies qui permettent de décrire le contexte et sa gestion d'une manière formelle. Le choix d'utiliser des ontologies pour décrire ce méta-modèle réside dans le fait que les ontologies offrent le moyen de décrire sémantiquement des informations, de partager les données décrites pour qu'elles puissent être utilisées par d'autres applications et d'étendre la description initiale lorsque de nouveaux besoins apparaissent. L'objectif du méta-modèle de CAMidO n'est pas de définir une nouvelle ontologie pour décrire le contexte. Nous pensons qu'un jour ou l'autre des ontologies standards de description de contexte vont exister. Lorsqu'elles existeront, les intergiciels devront s'appuyer sur ces ontologies standards et les étendre pour décrire les informations nécessaires à l'automatisation des processus de gestion du contexte. Notre proposition s'inscrit dans cette démarche. Les ontologies standards n'existant pas encore, nous proposons également une ontologie de contexte, mais ce n'est pas le cœur de notre travail.

3.3 Contenu du méta-modèle de CAMidO

Automatiser le processus de gestion du contexte et d'adaptation des applications permet de faciliter la création d'applications sensibles au contexte. En effet, la tâche de création de ce type d'applications pourra être partagée entre plusieurs acteurs. Les développeurs se chargent de programmer le code fonctionnel des applications alors que les concepteurs se chargent de décrire les informations de sensibilité au contexte associées à ces applications. Lors du déploiement de ces applications, l'intergiciel peut ainsi générer le code de gestion du contexte et d'adaptation de chaque application à l'aide des descriptions fournies par les concepteurs.

En prenant en considération les éléments fonctionnels de gestion du contexte, l'automatisation du processus y affairant nécessite de fournir à l'intergiciel les informations nécessaires à l'exécution de ces éléments fonctionnels, à savoir les logiciels associés aux capteurs qui collectent les contextes, les contextes pertinents pour l'application, les règles d'interprétation, les situations pertinentes et les politiques d'adaptation de l'application associées.

Le méta-modèle de CAMidO offre aux concepteurs d'applications le moyen de décrire les informations nécessaires à l'exécution automatique des processus de gestion de la sensibilité au contexte en utilisant des ontologies. Ce méta-modèle d'ontologie permet la description du contexte, des règles d'interprétation de contexte de haut niveau et des politiques d'adaptation de l'application.

Le choix d'utiliser une ontologie pour atteindre cet objectif est motivé par les caractéristiques des langages d'ontologies qui permettent de créer des modèles expressifs, extensibles, réutilisables, partageables et sur lesquels on peut raisonner à l'aide de moteur d'inférence. Le langage OWL [107], par exemple, est un langage recommandé par le W3C pour décrire des ontologies. Il offre un moyen simple et efficace basé sur un modèle de description XML pour faire des descriptions orientées objets, partager les données décrites, raisonner sur ces données et ajouter des axiomes pour décrire des relations spécifiques entre ces informations.

En utilisant le méta-modèle de CAMidO, ajouter la sensibilité au contexte à une application revient à décrire les informations associées au contexte et à sa gestion. De plus, grâce à ce méta-modèle, le déployeur d'application peut choisir les capteurs que l'intergiciel doit utiliser pour collecter le contexte.

L'intégration de la description des capteurs dans le méta-modèle permet non seulement d'automatiser le processus de collecte du contexte et d'activation des capteurs, mais offre aussi au déployeur d'applications la possibilité de choisir les capteurs à utiliser pour collecter les informations de contexte. Il peut enrichir l'ontologie en ajoutant de nouveaux types de capteurs. L'ontologie de capteurs est partagée et peut être utilisée par d'autres applications.

La description des règles d'interprétation, des situations pertinentes, des composants sensibles au contexte et des politiques d'adaptation sont faites à l'aide du même langage de description. Ces descriptions permettent d'automatiser le processus d'analyse du contexte et d'adaptation de l'application.

Le niveau méta de CAMidO permet de décrire des concepts généraux concernant les capteurs, les contextes et l'application. Le niveau modèle permet de spécifier ces descriptions pour

une application donnée alors que le niveau instances représente l'instantiation du modèle lors de l'exécution de l'application.

Avant d'aborder la description détaillée du méta-modèle offert par CAMidO, il est nécessaire de définir les types d'adaptations pris en compte par cet intergiciel, et de les illustrer à travers des exemples d'applications sensibles au contexte.

3.4 Définition des types d'adaptation gérées par CAMidO

Dans cette section, nous décrivons les types d'adaptations pris en compte par CAMidO, puis nous illustrons leurs utilités à travers des scénarios d'applications sensibles au contexte.

3.4.1 Définition

L'adaptation représente l'élément central d'une application sensible au contexte. Elle consiste à adapter le comportement d'une application lors de la détection d'une situation pertinente. La description de l'adaptation doit être présente dans le modèle pour spécifier à l'intergiciel les situations pertinentes auxquelles l'application est sensible et comment réagir lors de la détection de telles situations. Nous rappelons que dans le cadre de cette thèse, nous considérons les intergiciels orientés composant où les applications sont vues comme un ensemble de composants logiciels indépendants et interconnectés entre eux à l'aide de ports ou de canevas de conception. De ce fait, nous considérons que la granularité d'adaptation est le composant et nous prenons en compte deux sortes d'adaptation : l'adaptation comportementale réactive et l'adaptation comportementale proactive. Chaque type d'adaptation est décrit grâce à une politique d'adaptation dans le méta-modèle.

La politique d'adaptation réactive consiste à spécifier l'opération qui doit être invoquée lors de la détection d'une situation pertinente. L'intergiciel à l'écoute du changement du contexte lance l'une des politiques réactives définie par le modèle si une situation pertinente est détectée, indépendamment de ce qui est en cours d'exécution.

La politique d'adaptation proactive consiste à décrire la situation pertinente qui nécessite la prise de décision d'adaptation et l'évènement qui nécessite l'exécution de l'adaptation. Cet évènement est représenté par l'invocation d'une opération ou par la réception d'une invocation. L'adaptation proactive consiste à détecter les situations pertinentes, à prendre la décision d'adaptation, puis à adapter les invocations de chaque composant lors de l'interception de ces invocations. Cette adaptation consiste à rediriger l'invocation vers un nouveau composant ou à invoquer une nouvelle opération.

L'adaptation proactive peut être exécutée du côté client et/ou du côté serveur de chaque composant. En effet, un composant peut modifier les invocations qu'il émet ou reçoit en fonction du contexte. L'adaptation proactive côté client consiste à intercepter les appels d'invocations d'une opération, par un composant, et à les ré-aiguiller vers une autre opération, ayant les mêmes

paramètres. Cette opération peut être fournie par le même composant ou bien par un autre composant, alors que l'adaptation proactive côté serveur consiste à intercepter les appels entrants dans le but de fournir si nécessaire une opération ayant un comportement différent de celle qui a été invoquée.

Les informations relatives à chaque type d'adaptation doivent être décrites en créant des instances des classes appropriées du méta-modèle (cf. section 3.5). La section suivante présente des exemples fil rouge d'applications qui illustrent les types d'adaptation gérées par CAMidO.

3.4.2 Scénarios d'illustration

Afin de mieux comprendre l'utilité des types d'adaptation pris en compte par CAMidO, nous les illustrons par trois applications sensibles au contexte : une application d'achat en ligne, une application de gestion de conférences et une application de gestion de crises. Ces applications se rapportent à des domaines différents.

Application d'achat en ligne pour utilisateur mobile ≪*SensitivePurchase*≫

Utiliser une application d'achat en ligne peut se révéler intéressant pour des utilisateurs mobiles qui souhaitent diminuer leur temps passé dans les magasins. Grâce à ce type d'application, l'utilisateur gagne du temps en commandant ses achats avant d'arriver au magasin.

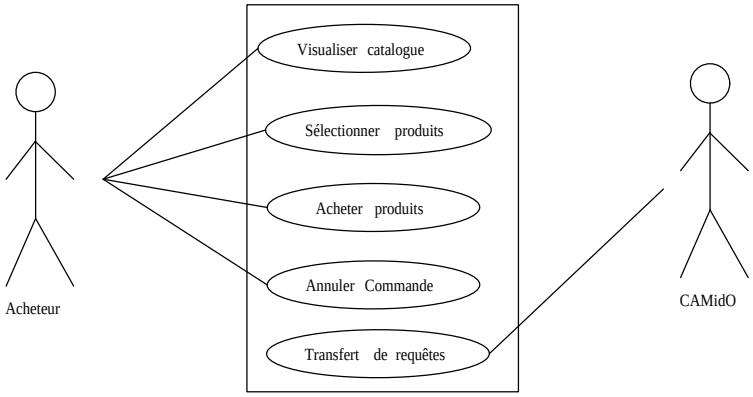

FIG. 3.1 – Diagramme des cas d'utilisation de l'application SensitivePurchase

La figure 3.1 schématise les cas d'utilisation de cette application que nous appelons *SensitivePurchase*. L'application *SensitivePurchase* est accessible via une interface graphique à partir de laquelle un utilisateur peut visualiser des catalogues, sélectionner des produits, acheter des produits ou annuler des commandes. Toutes ces fonctionnalités sont accessibles suivant le paradigme client/serveur.

Les contextes pertinents pour l'application *SensitivePurchase* sont :

1. l'état de la connexion réseau. Les situations pertinentes sont : connecté ou déconnecté,

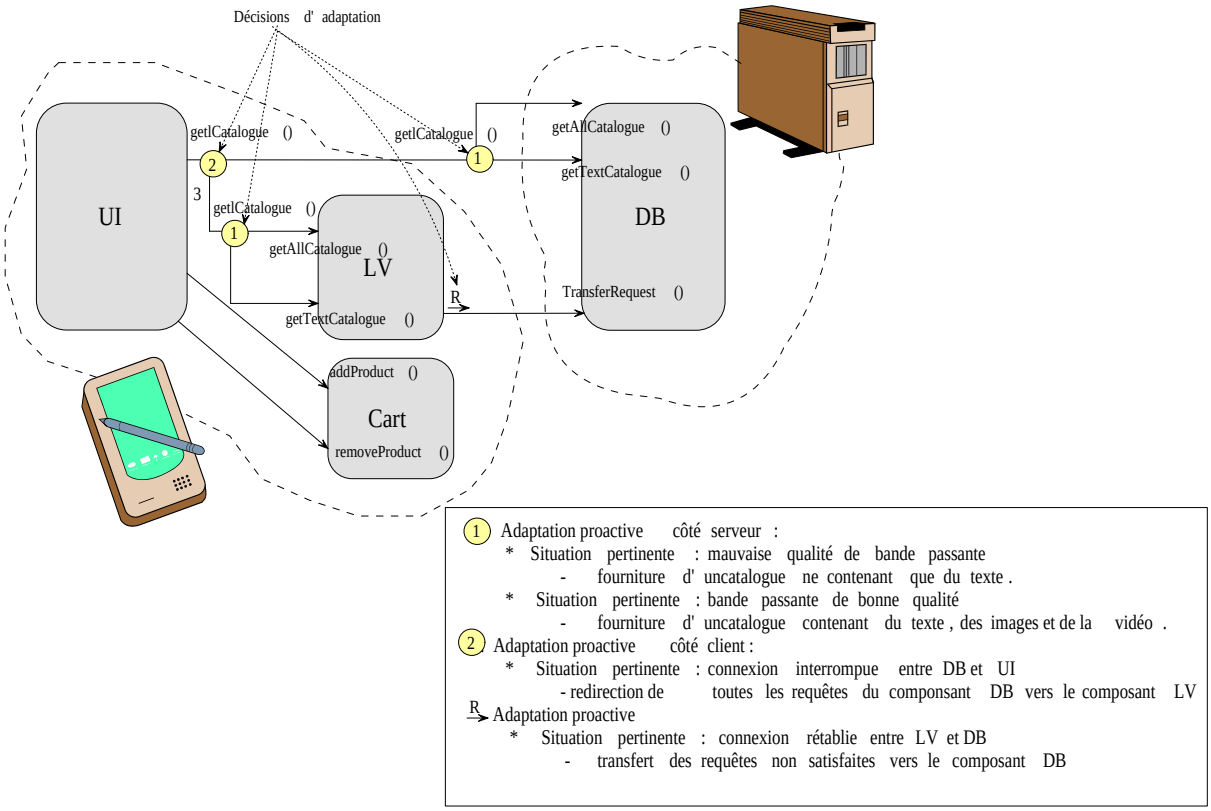

FIG. 3.2 – Structure de l'application *SensitivePurchase*

2. la variation de la bande passante. Les situations pertinentes sont : bande passante de bonne qualité ou de mauvaise qualité.

Comme l'illustre la figure 3.2, l'application d'achat en ligne est constituée de quatre composants :

- le composant Interface Utilisateur (UI) permet à l'utilisateur de se connecter à un magasin, de visualiser les catalogues des produits en vente dans ce magasin, de sélectionner les produits souhaités, de commander et d'annuler des commandes. Ce composant est installé sur le dispositif mobile. Il est sensible à l'état de la connexion réseau (connecté ou déconnecté),
- le composant Base de Données (DB) fournit les catalogues des produits disponibles dans le magasin. Ces catalogues peuvent contenir du texte, des images ou de la vidéo. Le composant DB est déployé sur le serveur de chaque magasin. Il est sensible à la variation de la bande passante,
- le composant Vue Locale (LV) propose une vue partielle des catalogues offerts par le composant DB. Le composant LV sert à se substituer au composant DB quand celui-ci n'est plus accessible dans le cas d'une déconnexion du réseau. Ce composant est installé sur le dispositif mobile. Il est sensible à l'état de la connexion réseau (connecté),
- le composant Panier (Cart) contient les produits sélectionnés par l'utilisateur. Il est installé sur le dispositif mobile.

Quand un utilisateur se retrouve dans une zone couverte par le réseau, il peut sélectionner un magasin et visualiser les catalogues des produits disponibles dans ce magasin à travers le composant DB. En fonction de la qualité de la bande passante, le composant DB adapte le service offert au composant UI en lui fournissant un catalogue ne contenant que du texte si la bande passante est de mauvaise qualité, ou bien un catalogue contenant du texte, des images et de la vidéo si la bande passante est de bonne qualité (adaptation proactive numéro 1 dans la figure 3.2). Ces deux adaptations sont appelées adaptations proactives côté serveur, car la décision d'adaptation est prise côté serveur. En cours d'exécution de l'application, la connexion entre le composant UI et le composant DB peut être interrompue à cause de la mobilité de l'utilisateur, dans ce cas, pour adapter l'application à cette nouvelle situation, une adaptation proactive côté client est prise en compte. Cette adaptation consiste à rediriger toutes les requêtes du composant UI destinées au composant DB vers le composant LV (adaptation proactive numéro 2 dans la figure 3.2). Le composant LV peut satisfaire certaines requêtes du composant UI en lui fournissant les catalogues disponibles. Les requêtes non satisfaites sont sauvegardées par le composant LV. Dès que la connexion est rétablie entre l'utilisateur et le magasin, l'adaptation réactive associée à cette situation pertinente est lancée. Cette adaptation réactive, conduite par le composant LV, consiste à transférer automatiquement toutes les requêtes non satisfaites vers le composant DB.

Application de gestion de conférences

La gestion d'une conférence à l'aide d'une application sensible au contexte permet le déroulement de la conférence dans de bonnes conditions. Cette application offre aux participants le moyen de visualiser et de télécharger les articles et les présentations des intervenants dans leurs dispositifs mobiles d'une manière automatique. La figure 3.3 illustre les cas d'utilisation de cette application.

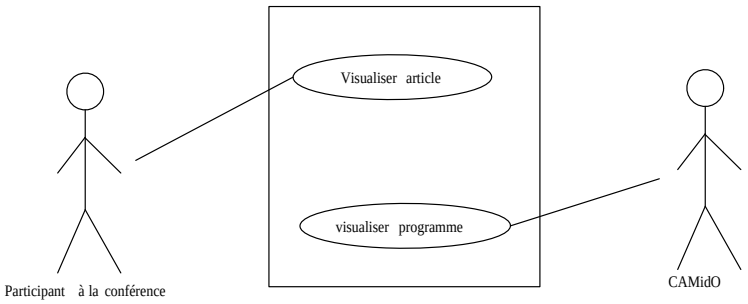

FIG. 3.3 – Diagramme des cas d'utilisation de l'application de gestion de conférences

Chaque participant porte un capteur de localisation *indoor* comme l'ActiveBadge [9] par exemple, qui fournit la localisation de l'individu dans un espace fermé. Les informations de localisation sont à la disposition de l'intergiciel.

Le scénario de cette application est le suivant. À chaque fois qu'une personne rentre dans la salle de conférence, sa présence est détectée et le programme du jour est transféré automatiquement sur son PDA. Au cours des présentations, chaque participant a la possibilité de demander la visualisation du papier associé à la présentation en cours sur son PDA.

Les contextes pertinents pour l'application de gestion de conférences sont :

1. l'espace mémoire disponible sur le PDA. Situations pertinentes : espace disponible important ou pas assez important,
2. la localisation *indoor* de chaque participant. Situation pertinente : le participant est dans la salle de conférence,
3. la qualité de la bande passante. Situations pertinentes : bande passante de bonne qualité ou de mauvaise qualité.

Comme l'illustre la figure 3.4, cette application est constituée de trois composants : le *DisplayerAgent*, le *PaperDownloader* et le *PaperRepository*.

Le *PaperDownloader* et le *DisplayerAgent* sont installés sur le PDA de chaque participant. Le *PaperDownloader* se connecte au *PaperRepository* pour télécharger les différents documents pour qu'ils soient affichés par le *DisplayerAgent*. Le *PaperDownloader* est sensible à l'espace mémoire disponible sur le PDA, ainsi que sa localisation *indoor* (la salle dans laquelle il se trouve).

Quand une personne rentre dans la salle de conférence avec son PDA, sa nouvelle localisation est détectée, et l'action réactive associée à cette situation pertinente est invoquée. Cette adaptation réactive consiste à invoquer l'opération de téléchargement du programme de la conférence par le composant *PaperDownloader*.

Au cours des présentations, chaque participant peut demander le téléchargement et la visualisation de l'article associé à la présentation. L'adaptation proactive, côté client, consiste soit à copier le fichier sur le PDA du participant si l'espace mémoire disponible sur le PDA est important, ou bien à faire une lecture distante de ce fichier si l'espace mémoire disponible n'est pas assez important (adaptation proactive numéro 1 sur la figure 3.4).

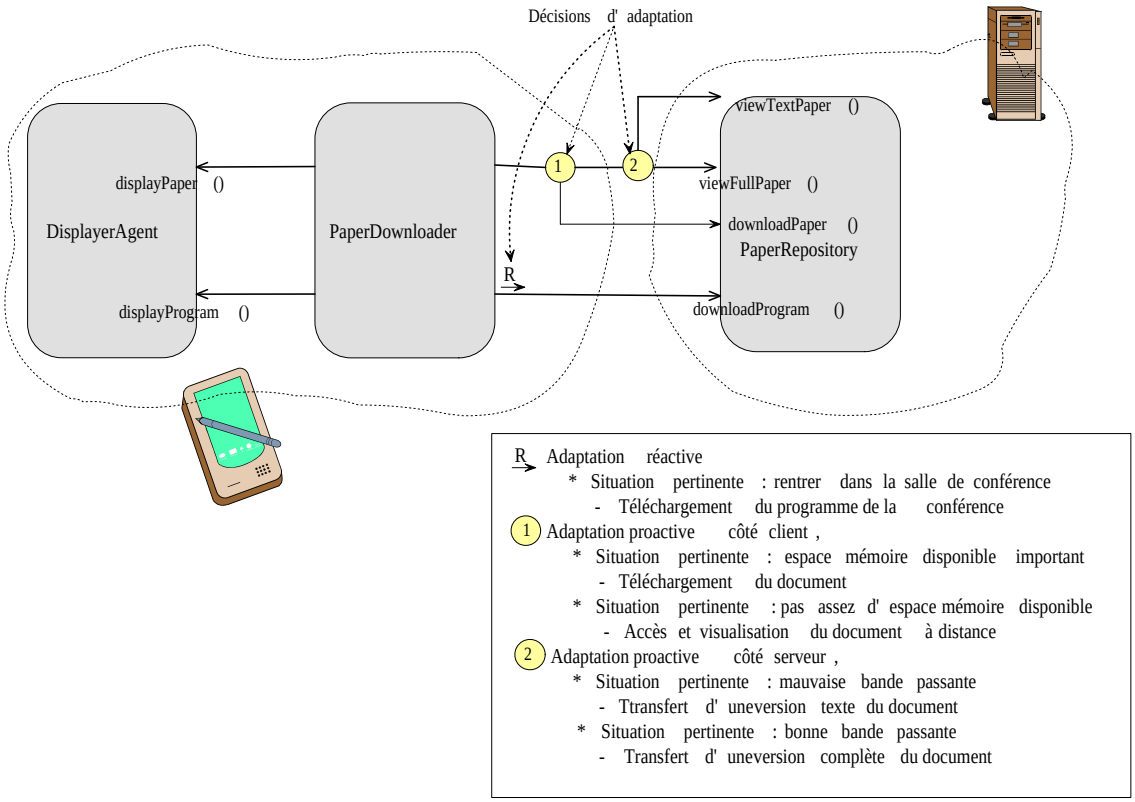

FIG. 3.4 – Structure de l'application de gestion de conférences

Le *PaperRepository* est installé au niveau du serveur. Il contient les différents papiers à présenter ainsi que le programme de la conférence. Ce composant est sensible à la bande passante du réseau. Quand un participant demande à visualiser une présentation, une version texte de celle ci est envoyée si la bande passante est de mauvaise qualité, alors qu'une version contenant du texte, des images et des séquences vidéo est envoyée si la bande passante est de bonne qualité (adaptation proactive numéro 2 sur la figure 3.4) .

Application de gestion de crises

Le scénario de l'application de gestion de crises est issu du projet AMPROS [5]. Cette application permet aux équipes médicales et aux secouristes présents sur le lieu d'une catastrophe de s'organiser et de se synchroniser afin de mener leur travail de sauvetage d'une manière efficace. Les équipes présentes sur le lieu de la catastrophe utilisent une technologie mobile et distribuée pour gérer des tâches telles que le partage d'informations et la prise de décisions. Les équipes de sauvetage arrivent équipés d'une antenne réseau, de serveurs installés dans une voiture (quartier général) et des PDA. Les cas d'utilisation de cette application sont illustrés dans la figure 3.5.

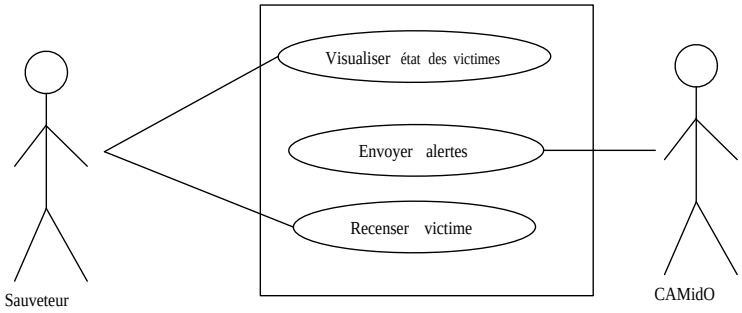

FIG. 3.5 – Diagramme des cas d'utilisation de l'application de gestion de crise

La première équipe arrivée sur le lieu de la catastrophe se charge de mettre les victimes dans des sarcophages. Chaque sarcophage est muni d'un PC, d'un capteur de tension artérielle et d'un capteur de rythme cardiaque pour surveiller l'état de la victime. Comme l'illustre la figure 3.6, dans le sarcophage, sont déployés le composant *Sensor* et le composant *Alert*. Le composant *Sensor* se charge d'envoyer des informations sur l'état de la victime au composant *GlobalView*, alors que le composant *Alert* se charge d'envoyer des messages d'alerte quand l'état de la victime est critique. Le composant *GlobalView* est installé dans le quartier général qui se charge de gérer les secours. Il permet de stocker l'état de toutes les victimes. Lors du recensement des victimes, chaque secouriste (médecin ou pompier) se charge de créer un profil de la victime qu'il a recensé en utilisant le composant *Rescuer* installé sur son PDA. En plus du composant *Rescuer*, le composant *GlobalViewLog* est déployé sur le PDA de chaque secouriste. Le *GlobalViewLog* permet au secouriste de créer des profils pour des victimes même si le PDA qu'il utilise n'est pas connecté au réseau.

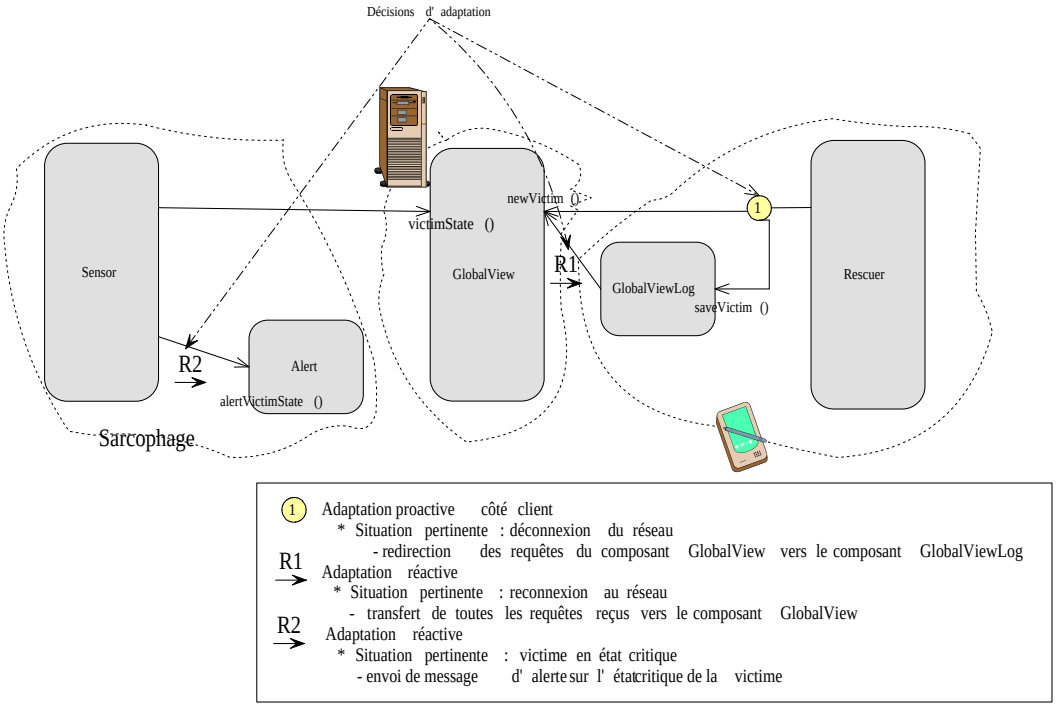

FIG. 3.6 – Structure de l'application de gestion de crises

Les contextes pertinents pour l'application gestion de crises sont :

1. l'état de la connexion réseau. Situations pertinentes : connecté ou déconnecté,
2. l'état de la victime. Situation pertinente : état critique.

Comme l'illustre la figure 3.6, le composant *Rescuer* crée un profil pour chaque victime en invoquant l'opération *newVictim()* offerte par le composant *GlobalView*. Ce composant est sensible à l'état de la connexion réseau. Si le secouriste se trouve dans une zone non couverte par le réseau et qu'il n'a pas accès au composant *GlobalView*, les appels d'opérations destinées au composant *GlobalView* sont redirigées vers le composant *GlobalViewLog* qui se charge d'enregistrer cette requête (adaptation proactive numéro 1 sur la figure 3.6). Le composant *GlobalViewLog* est lui aussi sensible à l'état de la connexion réseau. Dès qu'il détecte une connexion, il transmet toutes les requêtes qu'il a reçues du composant *Rescuer* vers le composant *GlobalView* (adaptation réactive numéro 1 sur la figure 3.6). Le composant *Sensor* installé dans le PC du sarcophage envoi des informations sur l'état de la victime au *GlobalView* à une fréquence prédéterminée. Si la victime se retrouve dans un état critique, le composant appelle automatiquement l'opération *AlertVictimState()* (adaptation réactive numéro 2 sur la figure 3.6). Cette opération se charge d'envoyer un message d'alerte aux secouristes présents dans le quartier général pour les informer sur l'état critique de la victime et sur la nécessité d'une présence médicale auprès d'elle ou d'une évacuation urgente vers l'hôpital.

Synthèse

Dans cette section, nous avons illustré les types d'adaptation considérés par l'intergiciel CAMidO à travers des exemples d'applications sensibles au contexte. Les adaptations réactives permettent aux applications sensibles au contexte de réagir instantanément lors de la détection d'une situation pertinente. Ce type de réaction est nécessaire quelque soit le scénario de l'application sensible au contexte. Les adaptations proactives permettent aux applications d'anticiper l'adaptation des invocations d'opérations en préparant la demande de redirection dès la détection d'une situation pertinente, ce qui évite à l'intergiciel de vérifier l'état de l'environnement à chaque fois qu'une opération est invoquée.

3.5 Description du méta-modèle de CAMidO

L'objectif de cette section consiste à décrire le méta-modèle offert par CAMidO. Ce méta-modèle permet aux concepteurs de créer des applications sensibles au contexte et de réutiliser les vocabulaires et les contextes définis par d'autres applications. Il permet aussi à l'intergiciel CAMidO d'automatiser le processus de gestion du contexte et de l'adaptation de l'application.

L'étude effectuée dans le chapitre 1 sur les différentes approches de modélisation du contexte, nous permet de considérer la modélisation par ontologie comme l'une des approches les mieux adaptées pour décrire le contexte. C'est pourquoi nous proposons un méta-modèle basé sur des ontologies.

Il nous semble nécessaire de préciser que notre objectif n'est pas de proposer de nouvelles ontologies de modélisation du contexte, mais d'utiliser des standards d'ontologies de contexte, dès qu'ils seront définis, et de les étendre afin qu'ils permettent de décrire la gestion de la sensibilité au contexte dans un environnement distribué orienté composant sensible au contexte.

Le méta-modèle de CAMidO permet de décrire des concepts généraux communs à toutes les applications sensibles au contexte ainsi que des données spécifiques aux applications orientées composant. La figure 3.7 schématise ce méta-modèle en décrivant les classes et les propriétés qui le constituent. Le méta-modèle de CAMidO est structuré en trois niveaux, une ontologie est associée à chaque niveau. Les deux premiers niveaux de l'ontologie permettent de décrire des informations sur les contextes que l'intergiciel peut observer, ainsi que les capteurs avec lesquels il doit interagir pour collecter les observations du contexte. Le troisième niveau de l'ontologie permet de décrire des informations spécifiques à chaque application orientée composant. Ce niveau offre aux concepteurs d'applications le moyen de décrire les informations dont l'intergiciel a besoin pour gérer le contexte et l'adaptation des applications d'une manière automatique.

Le premier niveau du méta-modèle, appelé niveau *Capteur*, permet de décrire les capteurs avec lesquels l'intergiciel interagit. La description des capteurs dans le méta-modèle de CAMidO a deux avantages : elle permet d'automatiser le processus d'interaction de CAMidO avec les capteurs, et offre la possibilité à différents intervenants de choisir les capteurs à utiliser pour collecter les informations de contexte. Cela facilite l'interaction de l'intergiciel avec de nouveaux capteurs. Il suffit de les ajouter. Dans CAMidO, un capteur est représenté par un logiciel qui interagit avec des capteurs physiques pour observer le contexte, ou qui collecte des informations de profils à partir d'un fichier ou d'une interface graphique.

Le second niveau du méta-modèle, appelé niveau *Contexte*, permet de décrire les contextes communs aux applications sensibles au contexte que l'intergiciel peut superviser. Les contextes observables considérés par CAMidO sont de deux types, les contextes directs collectés à partir des capteurs et les contextes indirects (ou contextes de haut niveau) interprétés à partir d'autres informations de contexte. Des règles décrivant l'interprétation des contextes indirects sont décrites dans ce niveau du méta-modèle. Néanmoins, le concepteur d'application peut fournir ses propres règles d'interprétation plus spécifiques aux besoins de chaque application lors de la création du modèle de l'application.

Le troisième niveau du méta-modèle est appelé niveau *Application* parce qu'il permet aux concepteurs de décrire des informations propres à chaque application sensible au contexte. Cette description concerne les contextes pertinents pour l'application, les composants sensibles au contexte qui constituent l'application, les règles d'interprétations propres à l'application, les situations pertinentes auxquelles chaque composant est sensible, les politiques d'adaptation réactives et proactives que l'intergiciel doit appliquer lors de la détection d'une situation pertinente et les services sensibles au contexte installés au dessus de CAMidO. Ces services utilisent CAMidO entant que collecteur et analyseur de contexte.

Nous détaillons dans le reste de cette section les ontologies associées à chaque niveaux du méta-modèle que nous venons de citer. Nous présentons pour chaque niveau les classes et les propriétés qui le caractérisent à travers des diagrammes de classes UML. Puis, nous illustrons l'utilisation de chaque ontologie en décrivant certaines informations de contexte associées aux

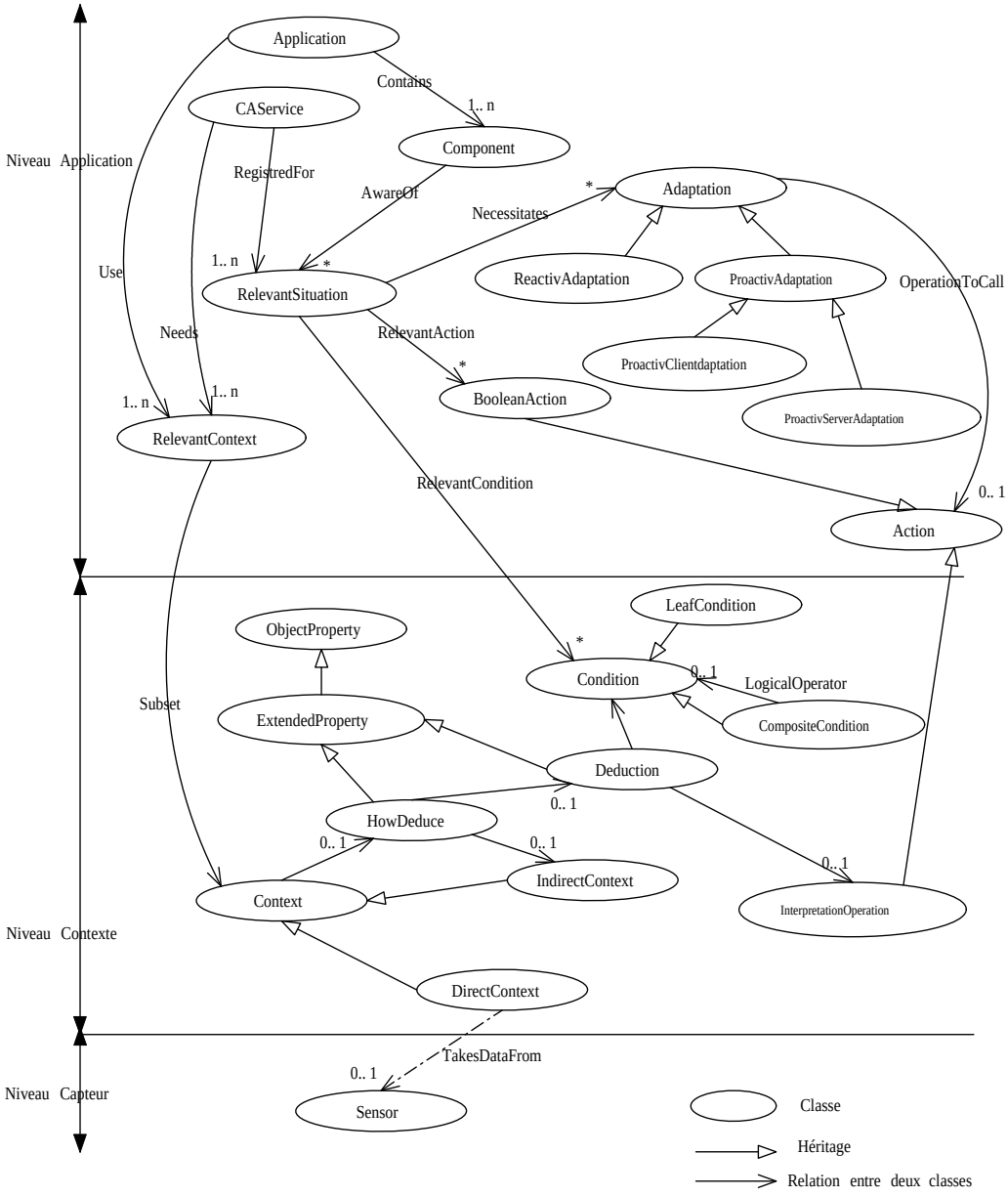

FIG. 3.7 – Vue générale du modèle de contexte de CAMidO

scénarios décrits dans la section 3.4.2 avec le langage OWL. La description OWL détaillée du méta-modèle de CAMidO est donnée dans l'annexe A.

Le choix d'une modélisation UML pour présenter ces ontologies est motivé par le fait qu'il n'existe pas de modélisation graphique standardisée des ontologies décrites en langage OWL [12]. Néanmoins, la modélisation UML ne permet pas de décrire certains aspects liés aux langages d'ontologies, comme la définition d'une classe comme étant le complément d'une autre classe, l'union d'un ensemble de classes ou les équivalences de classes par exemple [47].

3.5.1 Niveau *Capteur* du méta-modèle

L'interaction de l'intergiciel CAMidO avec des capteurs pour collecter des informations de contexte nécessite la présence de leurs descriptions dans le niveau capteur du méta-modèle. En effet, l'intergiciel CAMidO utilise les informations présentes à ce niveau pour activer puis communiquer avec les capteurs de collecte du contexte. L'ajout d'une nouvelle instance de capteur dans l'ontologie permet à l'intergiciel d'interagir avec ce capteur et de collecter les informations de contexte qu'il fournit.

La description des capteurs dans l'ontologie présente deux avantages. D'une part, elle permet à différents intervenants de choisir les capteurs que l'intergiciel doit utiliser. D'autre part, elle offre à l'intergiciel la possibilité d'interagir avec de nouveaux capteurs sans que cela nécessite des efforts de programmation.

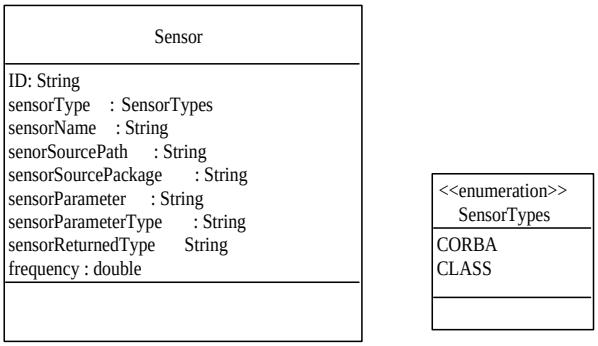

FIG. 3.8 – Diagramme de classes UML du niveau *Capteur*

L'ontologie associée au niveau capteur est constituée d'une classe unique appelée *Sensor*, d'un ensemble de propriétés (attributs dans le diagramme de classe UML) qui permettent de décrire les caractéristiques de chaque capteur et d'un ensemble d'axiomes qui permettent de décrire des contraintes sur ces propriétés. La figure 3.8 représente les attributs de la classe *Sensor*. L'attribut *sensorType* permet de spécifier le type du capteur. En effet, l'intergiciel CAMidO peut interagir avec deux types de capteurs, ceux fournis par des objets CORBA distants, ou ceux fournis par des objets java locaux. Grâce à l'attribut *sensorType*, l'intergiciel connaît le type du capteur avec lequel il interagit et ainsi détermine les techniques à utiliser à cet effet. L'attribut *sensorName* permet d'identifier le nom du capteur. Ce nom est utilisé par l'intergiciel, soit pour

chercher la référence du capteur dans le service de nommage s'il s'agit d'un objet CORBA, soit pour déterminer le nom de la classe java qui offre la méthode d'interaction avec le capteur. L'attribut *sensorSourcePath* et *sensorSourcePackage* permettent de déterminer respectivement l'URI de la classe associée au capteur et le paquetage auquel cette classe appartient. Les attributs *sensorParameter*, *sensorParameterType* et *sensorReturnedType* permettent de décrire l'interface de chaque capteur. L'attribut *frequency* permet de déterminer la fréquence à laquelle les informations de contexte seront collectées. Les informations fournies à travers ces attributs sont utilisées par l'intergiciel afin d'interagir avec le capteur.

```
1  <owl:Class rdf:ID="SensorBandwidth">
2    <rdfs:subClassOf rdf:resource="#Sensor"/>
3  </owl:Class>
4
5  <SensorBandwidth rdf:ID="bandwidthCollector">
6    <sensorType>CLASS</sensorType>
7    <sensorName>bandwidthCollector</sensorName>
8    <sensorSourcePath>C:\\CAMidO\Demo\</sensorSourcePath>
9    <sensorSourcePackage>Sensor.Bandwidth </sensorSourcePackage>
10   <sensorReturnedType>Integer</sensorReturnedType>
11   <frequency>20</frequency>
12 </SensorBandwidth>
```

FIG. 3.9 – Description OWL du capteur *bandwidth*

La figure 3.9 illustre un exemple de capteur de bande passante. Les lignes 1 à 3 représentent la description d'une catégorie de capteurs qui se chargent de collecter des informations de bande passante. Cette catégorie est représentée par la classe *SensorBandwidth*. Les lignes 5 à 12 de cette figure illustrent la description d'une instance de capteur appartenant à cette catégorie. Ce capteur permet de collecter les informations de bande passante toutes les 20 milli-secondes. La classe qui offre la méthode de collecte des ces informations à la même nom que le capteur (*bandwidthCollector*), elle est fournie par le paquetage *Sensor.Bandwidth* qui se trouve dans le chemin suivant : *C :\ \CAMidO\Demo*.

Le capteur *bandwidthCollector* est utilisé par l'application *SensitivePurchase* et l'application de gestion de conférences décrites dans la section 3.4.2. En plus de ce capteur, cette ontologie contient la description des capteurs qui permettent de collecter des informations de localisation, des informations sur le rythme cardiaque et des informations sur la tension artérielle d'une victime. Ajouter de nouveaux capteurs à l'ontologie associée au niveau *Capteur* est facile à mettre en œuvre. Il suffit de créer une instance de ce capteur et de décrire les attributs associés à cette instance (cf. figure 3.9).

3.5.2 Niveau *Contexte* du méta-modèle

L'ontologie *Contexte* permet de décrire les contextes observables que l'intergiciel peut surveiller. Nous considérons deux types de contextes observables : les contextes collectés à partir des capteurs et les contextes interprétés. La description du contexte collecté sera associée à un capteur. L'intergiciel utilisera ce capteur pour collecter les informations associées à ce contexte. Les informations de profil statique peuvent être associées à un capteur dont la fréquence est

égale à 0, ce qui signifie que l'intergiciel collecte des informations de ce capteur qu'une seule fois, lors du démarrage de l'application. La description des contextes de haut niveau nécessite la description des règles d'interprétations.

Description du contexte

Pour permettre la description du contexte, nous avons muni l'ontologie associée au niveau *Contexte* d'un ensemble de classes et d'attributs. Comme le montre la figure 3.10, cette ontologie est constituée de trois classes : la classe *Context*, la classe *DirectContext* et la classe *IndirectContext*. Les classes *DirectContext* et *IndirectContext* dérivent de la classe *Context*. Elles permettent respectivement de décrire les informations de contexte collectées par les capteurs et les informations de haut niveau interprétées en utilisant d'autres informations de contexte. Chaque information de contexte est décrite par son nom et un identificateur grâce aux attributs *contextName* et *ID*. Chaque contexte direct est associé à un capteur grâce à l'association *takesDataFrom*. Cette association permet de choisir le capteur que l'intergiciel doit utiliser pour collecter les données associées à ce contexte.

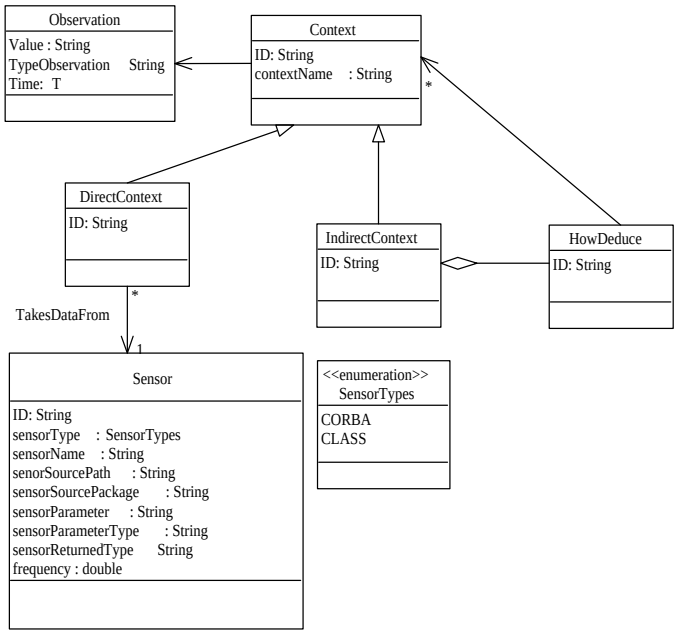

FIG. 3.10 – Extrait du diagramme de classes UML du niveau *Contexte*

La figure 3.11, représente la description des contextes *Location* et *Proximity* utilisés par les applications d'achat en ligne, de gestion de conférences et de gestion de crise. Le contexte *Location* est un contexte direct collecté à partir du capteur de localisation *IndoorSensorLocation*, alors que le contexte *Proximity* est un contexte indirect déduit à partir de règles d'interprétation.

```
<owl:Class rdf:ID="Location">
  <rdfs:subClassOf rdf:resource="#Direct"/>
  <contextName>Location</contextName>
  <TakesDataFrom rdf:resource="IndoorSensorLocation"/>
</owl:Class>

 <owl:DatatypeProperty rdf:ID="X">
    <rdfs:domain rdf:resource="#Location"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
  </owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:ID="Y">
    <rdfs:domain rdf:resource="#Location"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:ID="Proximity">
    ...
    <rdfs:subClassOf rdf:resource="#Indirect"/>
  </owl:Class>
```

FIG. 3.11 – Description OWL des contextes *Location* et *Proximity*

Description des règles d'interprétation

Le deuxième niveau du méta-modèle permet de décrire des règles d'interprétation pour déduire des contextes de haut niveau. Ces règles sont utilisées par l'intergiciel CAMidO pour automatiser le processus d'interprétation (cf. chapitre 4).

```
<owl :Class rdf ID="ExtendedProperty">
  <rdfs:subClassOf OWL:ObjectProperty/>
</owl:Class>
```

FIG. 3.12 – Description OWL de la classe ExtendedProperty

Le langage OWL offre le moyen de créer des associations entre des classes. Cependant ces associations ne peuvent pas être munies d'attributs. Pour remédier à cela, nous avons définit la classe *ExtendedProperty* décrite dans la figure 3.12 qui hérite de *ObjectProperty*. La classe *ExtendedProperty* permet de spécialiser des propriétés en leur ajoutant des informations supplémentaires sous la forme d'attributs.

Comme l'illustre la figure 3.13, La description des règles d'interprétation est faite grâce à la propriété *HowDeduce* qui hérite de la classe *ExtendedProperty*. L'interprétation d'un contexte de haut niveau peut se faire de deux manières : soit en faisant appel à une opération pour calculer la nouvelle valeur du contexte, soit en décrivant les conditions qui nécessitent que le contexte indirect prenne des valeurs spécifiques. La classe *HowDeduce* est décrite grâce à un ensemble d'attributs et d'associations avec des classes. L'association *ExecuteDeduction* permet de spécifier l'opération d'interprétation que l'intergiciel doit invoquer pour calculer la nouvelle valeur du contexte, alors que l'association *IfDeduction* permet de spécifier les conditions sous lesquelles le contexte prend des valeurs spécifiques.

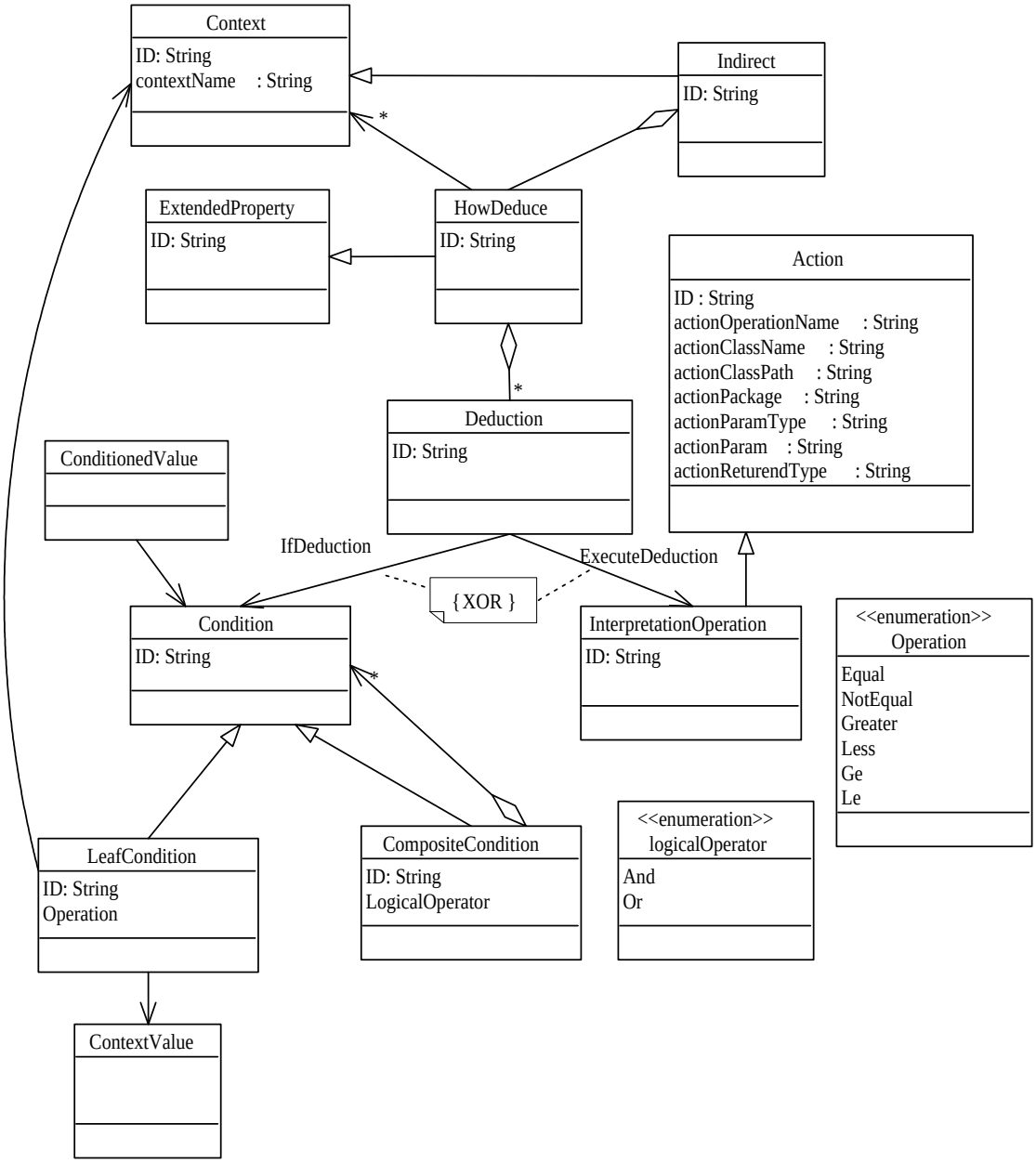

FIG. 3.13 – Diagramme de classes UML des règles d'interprétation

La description des opérations d'interprétation est faite grâce à la classe *InterpretationOperation* qui hérite de la classe *Action*. La classe *InterpretationOperation* permet de décrire les informations associées à l'opération d'interprétation du contexte pour qu'elle puisse être invoquée automatiquement par l'intergiciel. De ce fait, cette classe est décrite par un ensemble d'attributs. L'attribut *actionOperationName* permet de décrire le nom de l'opération d'interprétation qu'il faut invoquer. L'attribut *actionClassName* permet de décrire le nom de la classe java qui fournit cette opération. L'attribut *actionClassPath* décrit le chemin où se trouve cette classe. L'attribut *actionPackage* décrit le paquetage auquel appartient cette classe. Les attributs *actionParam*, *actionParamType* et *actionReturnedType* permettent de décrire respectivement les paramètres de l'opération, leurs types et le type de retour de cette opération.

La description des conditions sous lesquels le contexte prend des valeurs spécifiques est effectuée grâce à la classe *ConditionnedValue*.

La figure 3.14 illustre certaines règles d'interprétation utilisées dans le cadre des applications d'achat en ligne et de gestion de conférences pour déduire des contextes de haut niveau. La règle *DeduceLocation* permet de déduire la localisation *indoor* d'un utilisateur, et donne le nom de la salle dans laquelle il se trouve. La déduction de la localisation d'une personne est faite en invoquant l'opération *RoomLocationName* dont la description est fournie dans les lignes 5 à 13 de la figure. La règle *DeduceConnectionState* permet de déduire l'état de la connexion réseau en fonction de la valeur de la bande passante. Si la bande passante est inférieure à 1kbps, on déduit que la machine est déconnectée (lignes 22 à 25).

```
1  <HowDeduce rdf:ID="DeduceLocation">
2    <howDeduceDomain>Location</howDeduceDomain>
3    <howDeduceRange>LocationName</howDeduceRange>
4    <Deduction rdf:ID="roomLocationNameDeduction>
5      <InterpretationOperation>
6        <actionOperationName>RoomLocationName<actionOperationName>
7        <actionClassName>LocationClass</actionClassName>
8        <actionClassPath>http://www.inf.int.evry.fr/~belhanafi/CAMidO/Demo/</actionClassPath>
9        <actionPackage>org.int.CAMidO.Demo</actionPackage>
10       <actionParamType>double#double</actionParamType>
11       <actionParam>$Location.x#$Location.y</actionParam>
12       <actionReturnedType>String</actionReturnedType>
13     </InterpretationOperation>
14   </Deduction>
15 </HowDeduce>
16
17 <HowDeduce rdf:ID="DeduceConnectionState">
18   <howDeduceDomain>Bandwidth</howDeduceDomain>
19   <howDeduceRange>ConnectionState</howDeduceRange>
20   <ContextTakesValue>Connected<ContextTakesValue>
21   <Deduction  rdf:ID="conditionConnectionstate" >
22     <LeafCondition rdf:ID="condition1">
23       <ContextName>Bandwidth</ContextName>
24       <Operation>Greater</Operation>
25       <ContextValue>1</ContextValue>
26     </LeafCondition>
27   </Deduction>
28 </HowDeduce>
29
```

FIG. 3.14 – Exemple de description des règles d'interprétation

3.5.3 Niveau *Application* du méta-modèle

Le troisième niveau du méta-modèle permet de décrire des informations spécifiques aux applications orientées composant sensibles au contexte. Ce méta-modèle contient des informations sur les contextes pertinents pour l'application, sur les composants sensibles au contexte qui constituent l'application, sur les règles d'interprétation spécifiques à chaque application, sur les situations pertinentes auxquelles chaque composant est sensible, sur les politiques d'adaptation qui décrivent la manière dont l'intergiciel doit adapter l'application lors de la détection d'une situation pertinente, et sur les services sensibles au contexte installés au dessus de l'intergiciel CAMidO.

Description du contexte pertinent

Un contexte pertinent pour une application représente un sous ensemble des contextes que l'intergiciel peut superviser et dont l'application est sensible. La figure 3.15, illustre le diagramme de classes associé au contexte pertinent pour une application. La classe *RelevantContext* permet à une application ou à un service installé au dessus de CAMidO de spécifier les contextes observables qui les interessent et auxquels ils sont sensibles. La figure 3.16, illustre la description des

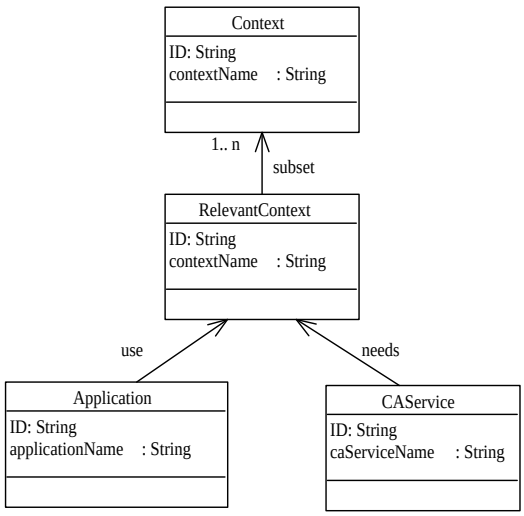

FIG. 3.15 – Diagramme de classes UML du contexte pertinent

contextes pertinents pour l'application *SensitivePurchase*. Les lignes 6 et 7 à de la figure illustrent l'énumération de ces contextes pertinents.

Description des situations pertinentes

Une situation pertinente représente tout état du contexte observé qui nécessite une réaction de l'application. L'intergiciel doit surveiller les variations du contexte pour détecter les situations pertinentes.

```
<Application rdf:ID="SensitivePurchase">
  <rdfs:subClassOf rdf:resource="SensitivePurchase">
</Application>
<SensitivePurchase rdf:ID="Sensitivepurchase">
  <ApplicationName>SensitivePurchase</SensitivePurchase>
  <use: rdf resource="#ConnectionState">
  <use: rdf resource="#Bandwidth">
</SensitivePurchase>
```

FIG. 3.16 – Exemple de description des contextes pertinents pour l'application *SensitivePurchase*

Le niveau *Application* de l'ontologie permet de décrire les situations pertinentes auxquelles chaque application est sensible grâce à la classe *RelevantSituation*. Comme le montre la figure 3.17, on peut exprimer une situation pertinente de deux manières. La première méthode consiste à le calculer grâce à un appel de méthode en utilisant la classe *BooleanAction*. La deuxième méthode utilise une condition à vérifier pour définir si on se trouve dans une situation pertinente. Cette condition est décrite grâce à la classe *Condition*. Dans ce cas, il est possible de définir des combinaisons de conjonctions ou de disjonctions des contextes collectés à partir de sources différentes.

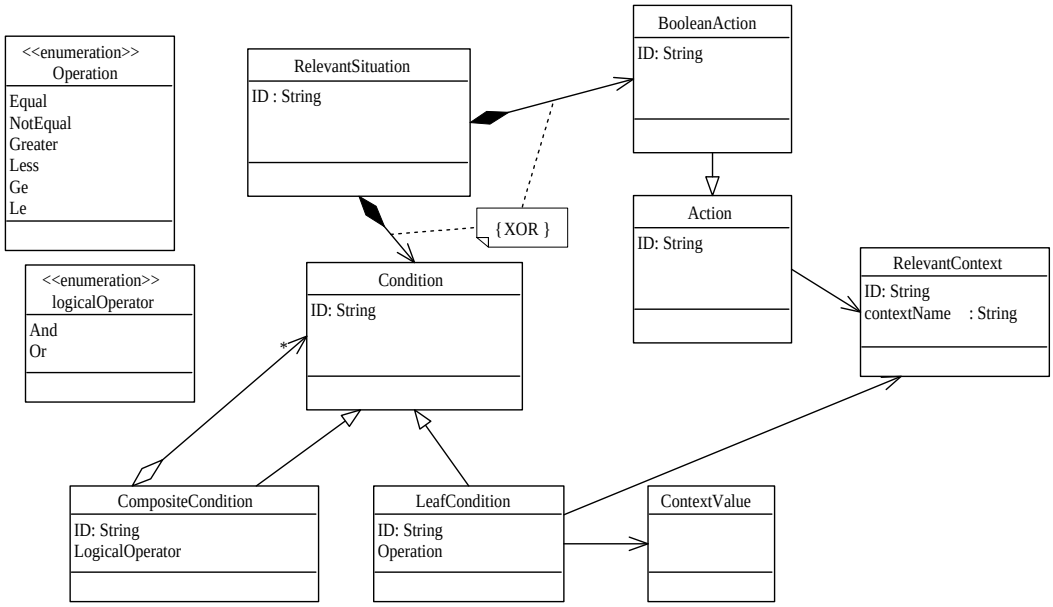

FIG. 3.17 – Diagramme de classes UML des situations pertinentes

La figure 3.18, illustre les situations pertinentes auxquelles l'application d'achat en ligne est sensible.

```
<RelevantSituation rdf:ID="RelevantConnection">
  <Condition rdf:ID="condition2">>
    <LeafCondition rdf:ID="leaf2">>
      <ContextName>ConnectionState</ContextName>
      <Operation>Equal</Operation>
      <ContextValue>Connected</ContextValue>
    <LeafLeafCondition>
  <Condition>
...
</RelevantSituation>

<RelevantSituation rdf:ID="RelevantNoConnection">
  <Condition rdf:ID="condition1">
    <LeafCondition rdf:ID="leaf1">>
      <ContextName>ConnectionState</ContextName>
      <Operation>Equal</Operation>
      <ContextValue>Disconnected</ContextValue>
    <LeafCondition>
  <Condition>
...
</RelevantSituation>
```

FIG. 3.18 – Extrait de description OWL de situations pertinentes de l'application *SensitivePurchase*

Description des politiques d'adaptation

Dans le but d'automatiser le processus d'adaptation, le niveau *Application* du modèle offre aux concepteurs d'applications le moyen de décrire les adaptations que l'intergiciel doit appliquer lors de la détection des situations pertinentes.

Comme nous l'avons décrit dans la section 3.4.1, CAMidO propose deux types d'adaptation : les adaptations réactives et les adaptations proactives. Le lien entre chaque composant et l'adaptation qui lui est associée est réalisé d'une manière transitive grâce à l'association *awareOf* et l'association *Necessitates* (cf. figure 3.19). L'association *awareOf* permet de relier chaque composant à l'ensemble des situations pertinentes auxquelles il est sensible, alors que l'association *Necessitates* relie chaque situation pertinente à l'adaptation qu'il est nécessaire d'appliquer quand cette situation pertinente est détectée.

La figure 3.19 schématise la vue globale des politiques d'adaptation. Comme l'illustre cette figure, la classe *ReactivAdaptation* permet de décrire les politiques d'adaptation réactives, alors que la classe *ProactivAdaptation* permet de décrire les politiques d'adaptation proactives.

Les politiques d'adaptation réactives sont décrites grâce aux classes *Action*, *ReactivAdaptation* et *RelevantContext*. La classe *ReactivAdaptation* qui hérite de la classe *Adaptation* permet de spécifier l'opération qui doit être invoquée par CAMidO lors de la détection de la situation pertinente décrite dans la classe *RelevantSituation*. Cette opération peut être localisée dans un paquetage externe, fournie par le composant qui invoque cette adaptation ou fournie par un autre composant qui appartient à l'application. Si l'opération est fournie par un autre composant appartenant à l'application, le nom de ce composant doit être décrit par l'attribut *OperationComponent*. Si l'opération est fournie par un paquetage externe, le nom de ce paquetage et son chemin doivent être décrits par les attributs *actionClassPath* et *actionPackage* de la classe *Action*.

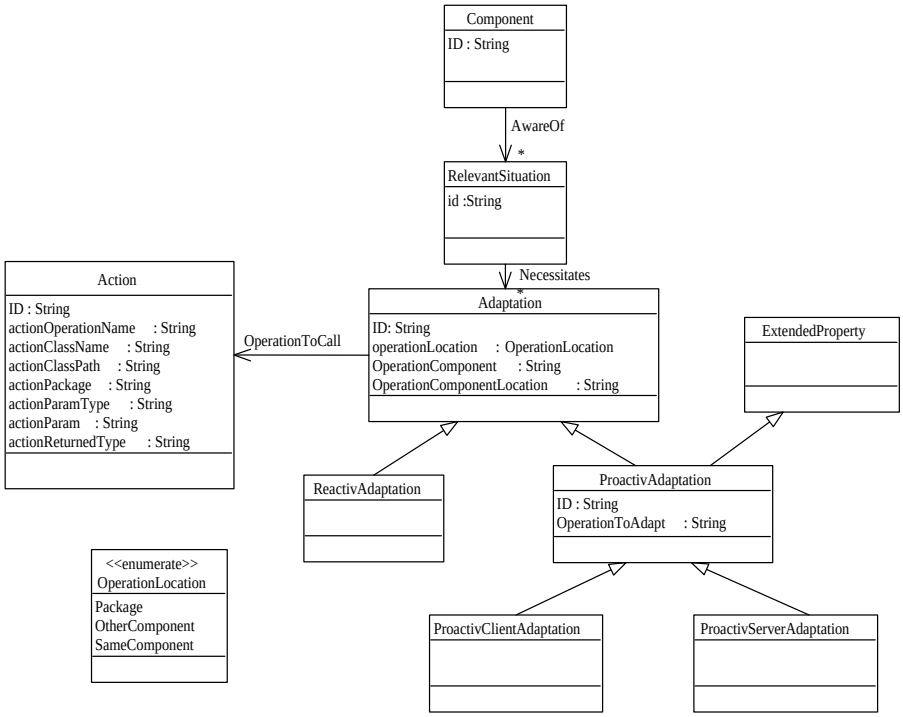

FIG. 3.19 – Extrait du diagramme de classes UML du méta-modèle d'adaptation

La figure 3.20, illustre la description de l'adaptation réactive associée à l'application *Sensitive-Purchase* (cf. section 3.4.2). Cette adaptation qui doit être lancée par le composant *LV* lors de la détection d'une reconnexion au réseau, consiste à invoquer l'opération *TransferRequest* fournie par le composant *LV*[1]. Cette opération se charge de transférer les requêtes non satisfaites vers le composant *DB*.

```
<ReactivAdaptation rdf:ID="AdaptationAction1">
  <OperationLocation>SameComponent</OperationLocation>
  <OperationComponent>LV</OperationComponent>
  <OperationToCall rdf:ID="operation1">
    <Action rdf:ID="Action1">
      <actionOperationName>TransferRequest</actionOperationName>
      <actionClassName>LVImpl</actionClassName>
      <actionClassPath>http://www.inf.int.evry.fr/~belhanafi/CAMidO/Demo/</actionClassPath>
      <actionPackage>org.int.CAMidO.Demo</actionPackage>
      <actionReturnedType>void</actionReturnedType>
    </Action>
  </OperationToCall>
</ReactivAdaptation>
```

FIG. 3.20 – Description OWL de l'adaptation réactive de l'application *SensitivePurchase*

L'adaptation proactive consiste à rediriger les invocations d'un composant. Cette redirection consiste à :

[1]L'opération *TransferRequest* fournie par le composant *LV* se charge d'invoquer l'opération *TransfertRequest* fournie par le composant *DB* en lui passant en paramètres les invocations non satisfaites.

– changer le nom de l'opération qui devait être invoquée,
– changer les paramètres de cette opération,
– changer le composant cible où était dirigée cette invocation.

Cette décision d'adaptation peut être effectuée soit côté client soit côté serveur.

La description des politiques d'adaptation proactive est effectuée grâce aux classes *ProactivAdaptation*, *ProactivClientAdaptation*, *ProactivServerAdaptation*, *Adaptation* et *Action*. Les classes *ProactivClientAdaptation* et *ProactivServerAdaptation* qui héritent de la classe *ProactivAdaptation* permettent de décrire respectivement les adaptations proactives côté client et les adaptations proactives côté serveur. L'attribut *OperationToAdapt* permet de spécifier l'opération qui doit être redirigée si la situation pertinente décrite par la classe *RelevantSituation* est détectée. Les informations concernant cette redirection sont décrites par la classe *Adaptation* et la classe *Action*. La classe *Adaptation* permet de définir le composant vers lequel cette invocation doit être redirigée grâce aux attributs *OperationLocationComponent* et *OperationComponent*. La classe *Action* permet de décrire la nouvelle opération qui doit être invoquée (son nom, ses paramètres ...).

La figure 3.21, illustre une politique d'adaptation proactive associée à l'application *SensitivePurchase* (cf. section 3.4.2). Cette politique d'adaptation associe le composant *UI* à la situation pertinente *DisConnection* et à l'invocation de l'opération *GetCatalogue* (lignes 1 à 9 de la figure). Lors de la détection de la situation pertinente *Disconnection*, toutes les invocations du composant *UI* à l'opération *GetCatalogue* sont redirigées vers le composant *LV* qui offre la même opération.

```
1  <Component rdf:ID="UI">
2    <ComponentName>UI</ComponentName>
3    <Awareof rdf:ID="DisConnect"/>
4      <RelevantSituation rdf:ID="DisConnection">
5        ...
6        <Necessitates rdf:resource="#ProactivAdaptation1">
7      </RelevantSituation>
8    <Awareof
9  </Component>
10
11 <ProactivClientAdaptation rdf:ID="ProactivAdaptation1">
12   <OperationToAdapt>GetCatalogue</OperationToAdapt>
13   <OperationLocation>OtherComponent</OperationLocation>
14   <OperationComponent>LV</OperationComponent>
15   <OperationToCall  rdf:ID="CAOperation1">
16     <Action rdf:ID= ACtionProactAdapt1>
17       <actionOperationName>GetCatalogue</actionOperationName>
18       <actionClassName>LVImpl</actionClassName>
19       <actionClassPath>http://www.inf.int.evry.fr/~belhanafi/CAMidO/Demo/</actionClassPath>
20       <actionPackage>org.int.CAMidO.Demo</actionPackage>
21       <actionReturnedType>File</actionReturnedType>
22     </Action>
23   </OperationToCall>
24 </ProactivClientAdaptation>
```

FIG. 3.21 – Exemple de description d'une adaptation proactive de l'application *SensitivePurchase*

Description des relations entre les informations de contexte

Le niveau *Application* du méta-modèle offre un moyen pour créer des relations entre les données décrites dans l'ontologie associée à ce niveau. Ces relations permettent à l'intergiciel de

déduire de nouvelles situations pertinentes et de les lier avec des politiques d'adaptation et des règles d'interprétation.

```
<owl:ObjectProperty rdf:ID="EquivalentContext">
   <rdfs:range rdf:resource="#Context"/>
   <rdfs:domain rdf:resource="#Context"/>
 </owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="EquivalentRelevantSituation">
   <rdfs:range rdf:resource="#RelevantSituation"/>
   <rdfs:domain rdf:resource="#RelevantSituation"/>
 </owl:ObjectProperty>
```

FIG. 3.22 – Description OWL des relations entre les informations de contexte

Dans le cadre de CAMidO, nous considérons les relations d'équivalence entre les différents types de contexte grâce à la propriété *EquivalentContext* et entre les situations pertinentes grâce à la propriété *EquivalentRelevantSituation*. En utilisant ces relations, l'intergiciel peut relier des situations pertinentes à des règles d'adaptation ainsi que des contextes à des politiques d'interprétation sans que l'utilisateur n'ait besoin de les respécifier. Par exemple si la situation pertinente S1 nécessite de lancer l'adaptation P1 et que l la situation pertinente S2 est équivalent à la situation pertinente S1, alors l'intergiciel va déduire que la situation pertinente S2 nécessite lui aussi le lancement de l'adaptation P1.

Description des services sensibles au contexte

Des services ou des applications sensibles au contexte peuvent posséder leurs propres mécanismes d'adaptation. Ils ont besoin d'interagir avec un intergiciel pour la sensibilité au contexte qui leurs fournissent le contexte dont ils ont besoin pour s'adapter.

Le méta-modèle de CAMidO offre aux concepteurs la possibilité de décrire ces services pour qu'ils puissent utiliser CAMidO entant qu'intergiciel pour la sensibilité au contexte. Cette description permet à ces services de s'enregistrer auprès de CAMidO pour les situations pertinentes auxquelles ils sont sensibles.

La figure 3.23 illustre le diagramme de classes des services sensibles au contexte. La classe *CAService* permet de décrire le nom et l'identificateur de ces services alors que l'association *RegistredFor* permet de spécifier les situations pertinentes pour lesquelles le service s'enregistre auprès de l'intergiciel.

3.6 Synthèse et discussion

Dans ce chapitre, nous avons présenté un méta-modèle d'ontologie pour décrire les informations liées à la sensibilité au contexte pour des applications orientées composant. Les travaux existants sur la modélisation du contexte décrits dans le chapitre 1 ne permettent pas de décrire

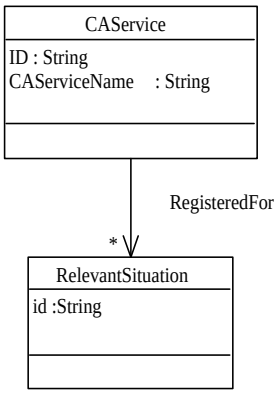

FIG. 3.23 – Diagramme de classes UML des services sensibles au contexte

tous les aspects liés à la gestion du contexte et à l'adaptation des applications orientées composant. De ce fait, notre objectif en proposant le méta-modèle de CAMidO consiste à compléter les modèles existants et à les étendre pour prendre en compte une large partie de ces aspects. Le méta-modèle proposé est structuré en trois niveaux. Les deux premiers niveaux permettent de décrire des concepts généraux des applications sensibles au contexte, alors que le dernier niveau permet de décrire des informations spécifiques aux applications orientées composant sensibles au contexte.

L'ontologie associée au niveau capteur permet de décrire les capteurs avec lesquels l'intergiciel interagit, alors que l'ontologie associée au second niveau permet de décrire les contextes observables. Les contextes observables sont de deux types : observées à travers des capteurs ou interprétées en utilisant d'autres informations de contexte observées. L'ontologie associée au troisième niveau du méta-modèle permet aux concepteurs d'application de décrire des informations spécifiques à l'application. Ces descriptions concernent les contextes pertinents pour l'application, les situations pertinentes de l'application, les politiques d'adaptation réactives et proactives à considérer lors de la détection de ces situations pertinentes, les règles d'interprétation d'un contexte de haut niveau spécifiques à l'application ainsi que le choix des capteurs à utiliser pour collecter les informations de contexte auxquelles l'application est sensible.

Par rapport aux travaux existants sur la modélisation du contexte (cf. chapitre 1), l'originalité de CAMidO réside dans le fait que le méta-modèle proposé permet non seulement la description des caractéristiques du contexte, mais considère aussi le fait que ce contexte est capturé puis utilisé par un système qui se charge d'adapter l'application à ses variations pertinentes. De ce fait, le méta-modèle de CAMidO permet de décrire toutes les informations qui nous semblent nécessaires à l'automatisation du processus de gestion du contexte et de l'adaptation des applications sensibles au contexte.

CAMidO ne prend en compte que les adaptations comportementales qui peuvent être de nature réactives ou proactives. L'intégration d'un nouveau type d'adaptation dans le méta-modèle de CAMidO, comme l'adaptation architecturale par exemple, nécessite l'extension de ce méta-modèle. Dans le cas des adaptations architecturales, cette extension consiste à ajouter des

classes qui permettent de décrire l'architecture de l'application ainsi que des classes pour décrire l'adaptation architecturale de l'application pour chaque situation pertinente. Nous donnons un exemple d'extension du méta-modèle offert par CAMidO dans le chapitre 5.

Le chapitre suivant décrit l'architecture de l'intergiciel CAMidO et les mécanismes qu'il utilise afin de gérer la sensibilité au contexte et l'adaptation comportementale des applications sensibles au contexte en utilisant le méta-modèle de CAMidO.

Chapitre 4

CAMidO, un intergiciel orienté composant sensible au contexte

4.1 Introduction

Dans le chapitre précédent, nous avons décrit le méta-modèle de CAMidO qui permet de modéliser le contexte auquel les applications peuvent être sensibles. Dans ce chapitre, nous présentons l'intergiciel CAMidO dont l'objectif est de faciliter le développement des applications sensibles au contexte. Cet intergiciel utilise le méta-modèle de CAMidO pour automatiser le processus de gestion du contexte et d'adaptation des applications aux situations pertinentes. Pour atteindre cet objectif, nous proposons en premier lieu un gestionnaire de contexte constitué d'un ensemble d'entités qui permettent de gérer le contexte. Par la suite, nous proposons d'intégrer le mécanisme d'adaptation dans les conteneurs des composants.

Ce chapitre présente dans le détail l'architecture de l'intergiciel CAMidO composée du gestionnaire de contexte et du gestionnaire d'adaptation (architecture du conteneur sensible au contexte). Nous expliquons dans la section 4.2 les motivations et les objectifs de l'intergiciel CAMidO, puis nous décrivons l'architecture de cet intergiciel dans la section 4.3. Nous décrivons les modes d'exécution de cet intergiciel dans la section 4.4. Dans la section 4.5, nous présentons le compilateur CAMidO et ses fonctionnalités. Enfin, nous concluons ce chapitre par une discussion (cf. section 4.6) et une conclusion (cf. section 4.7).

4.2 Motivations et objectifs

Sans intergiciel dédié à la sensibilité au contexte, la création d'une application sensible au contexte demeure une tâche fastidieuse qui nécessite plusieurs étapes supplémentaires de programmation par rapport à une application classique. La première étape est la conception qui consiste à définir la structure de l'application et à décrire d'une manière exhaustive le contexte auquel elle est sensible. La seconde étape consiste à programmer l'application. Le programme

développé englobe le code métier de l'application et la gestion de la sensibilité au contexte. Le code relatif à la gestion de la sensibilité au contexte doit permettre à l'application, lors de son exécution, de s'interfacer avec les capteurs pour collecter des informations de contexte, interpréter et analyser les données collectées, et adapter l'application lors de la détection d'une situation pertinente. Le recours aux services d'un intergiciel permet de déléguer certaines tâches de gestion de la sensibilité au contexte et ainsi faciliter le développement des applications sensibles au contexte.

Dans le chapitre 2, nous avons décrit un ensemble d'intergiciels ayant pour objectif de faciliter pour les développeurs la tâche de création d'applications sensibles au contexte. Ces intergiciels prennent en charge la gestion de certains aspects liés à la sensibilité au contexte. Cependant, les aspects non gérés par ces intergiciels restent à la charge du développeur d'applications. C'est le cas pour les intergiciels SOCAM [44] et K-component [38] par exemple. L'objectif des travaux présentés dans cette thèse consiste à faciliter davantage le développement des applications sensibles au contexte en intégrant tous les aspects liés à la sensibilité au contexte au niveau de l'intergiciel CAMidO.

En utilisant l'intergiciel CAMidO, plusieurs acteurs peuvent contribuer à la création d'une application sensible au contexte de façon indépendante. Les concepteurs se chargent de décrire les informations de sensibilité au contexte associées à chaque composant. Cela consiste à instancier le niveau application du méta-modèle de CAMidO. Les développeurs auront deux tâches à développer : la programmation des interfaces fonctionnelles de chaque composant comme pour toute application, et la programmation des méthodes qui concernent l'adaptation réactive et l'interprétation du contexte. Ces méthodes sont définies dans le méta-modèle de CAMidO. En s'appuyant sur les informations décrites par les concepteurs d'application, l'intergiciel CAMidO peut automatiser le processus de gestion du contexte et d'adaptation des applications.

L'intergiciel CAMidO sépare la gestion du contexte de la gestion de l'adaptation pour permettre son utilisation entant qu'intergiciel pour la sensibilité au contexte. L'architecture de CAMidO est décomposée en deux modules distincts : le gestionnaire de contexte et le gestionnaire d'adaptation. Le gestionnaire de contexte regroupe les entités de collecte, d'interprétation et d'analyse du contexte, alors que le gestionnaire d'adaptation regroupe des contrôleurs qui se chargent d'adapter les composants aux situations pertinentes.

4.3 Architecture de CAMidO

Dans le but de faciliter le développement des applications sensibles au contexte, nous avons intégré à l'intergiciel CAMidO un ensemble d'entités qui se chargent de gérer le contexte et l'adaptation des applications. Notre objectif est que les développeurs d'application n'aient plus à programmer l'interaction avec les capteurs, ni l'analyse, ni l'interprétation des contextes collectées, ni l'adaptation de l'application. En effet, le code associé à ces tâches est géré par l'intergiciel CAMidO. L'architecture de CAMidO est structurée en deux parties distinctes : le gestionnaire de contexte que nous décrivons dans la section 4.3.1 et le gestionnaire d'adaptation que nous décrivons dans la section 4.3.2.

4.3.1 Architecture du gestionnaire de contexte

Le gestionnaire de contexte se charge d'interagir avec les capteurs pour observer le contexte, interpréter les données observées et les analyser afin de détecter des situations pertinentes pour les applications gérées par CAMidO.

Comme l'illustre le figure 4.1, la gestion du contexte dans CAMidO est assurée par un ensemble d'entités : le *CollectionManager*, le *ContextAnalyser*, le *ContextInterpreter*, le *Repository* et le *InferenceComponent*. Ces entités travaillent en collaboration pour gérer le contexte selon le modèle fourni par chaque application.

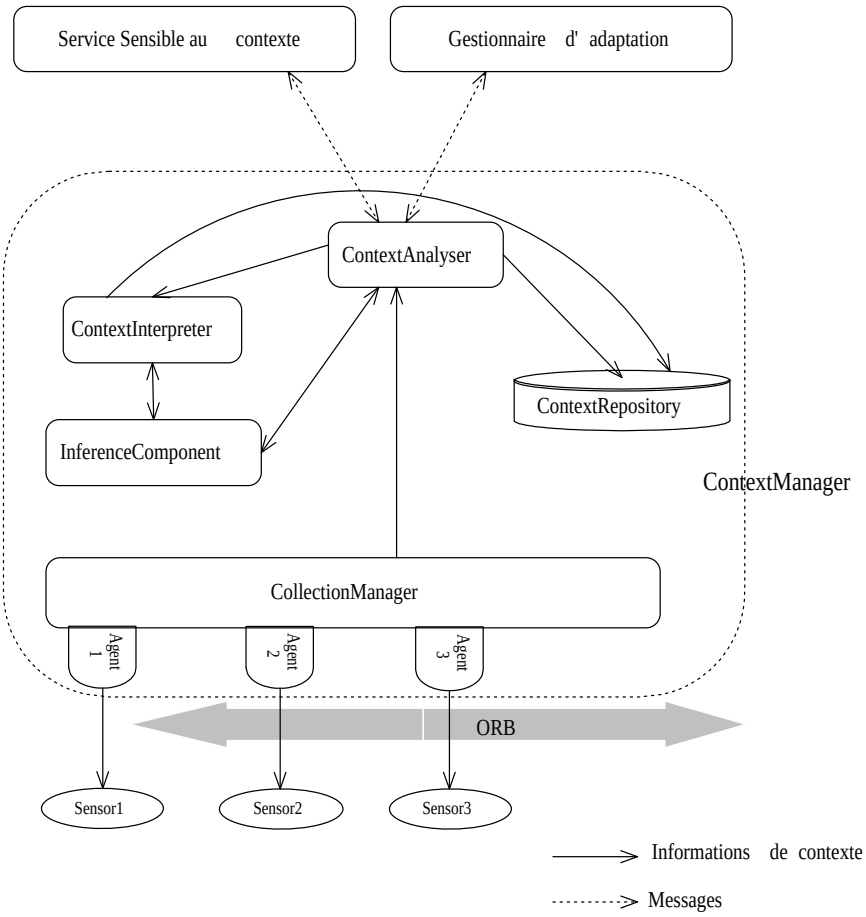

FIG. 4.1 – Architecture du gestionnaire de contexte de l'intergiciel CAMidO

L'entité *CollectionManager* se charge d'activer les capteurs logiciels définis dans l'ontologie *Capteur* du méta-modèle de CAMidO. Ces capteurs logiciels communiquent avec des capteurs physiques ou logiques pour collecter les observations du contexte.

L'entité *ContextInterpreter* se charge d'interpréter les contextes de haut niveau en utilisant les règles d'interprétation décrites dans le méta-modèle de CAMidO.

Le *ContextAnalyser* se charge de déceler les changements de situations, de détecter les situations pertinentes, et de notifier les composants sensibles à ces situations.

L'entité *InferenceComponent* est utilisée par le *ContextInterpreter* pour raisonner sur les informations de contexte et par le *ContextAnalyser* pour détecter les variations de situations pertinentes du contexte.

Le *Repository* se charge de sauvegarder les observations du contexte.

L'interaction du gestionnaire d'adaptation vers le gestionnaire de contexte est assurée grâce à l'interface *ContextManager* décrite dans la figure 4.2. Cette interface permet au gestionnaire de contexte de s'enregistrer auprès du *ContextAnalyser* pour les situations pertinentes auxquelles chaque composant de l'application est sensible.

```
1 interface ContextManager
2 {
3   void subscribe(in string Component_Id, in string listRelevant_Situation);
4 };
```

FIG. 4.2 – Interface IDL *ContextManager*

Le diagramme de séquences associé aux entités qui se chargent de la gestion du contexte et de l'adaptation de l'application est décrit dans la figure 4.3. Lors du déploiement d'une application sensible au contexte, le gestionnaire d'adaptation s'enregistre auprès du *ContextAnalyser* pour les situations pertinentes auxquelles les composants qui constituent l'application sont sensibles. Le *CollectionManager* transmet systématiquement les données collectées au *ContextAnalyser* qui se charge dans un premier temps de vérifier si les valeurs du contexte ont changé. Si c'est le cas, il transmet ces données collectées au *ContextInterpreter*. Le *ContextInterpreter* utilise l'entité *InferenceComponent* pour déduire les contextes de haut niveau. Ce dernier se charge de raisonner sur les informations de contexte en utilisant les règles d'interprétation fournies par le méta-modèle de CAMidO. L'entité *ContextInterpreter* retourne le contexte de haut niveau à l'analyseur qui se charge de détecter les variations de situations pertinentes en utilisant l'entité *InferenceComponent*. Pour chaque situation pertinente détectée, le *ContextAnalyser* notifie le gestionnaire d'adaptation, et plus précisément tous les composants sensibles à ces situations.

Dans la suite de cette section, nous décrivons plus en détail les mécanismes de gestion du contexte assurés par CAMidO, ainsi que les outils et les techniques utilisés à cet effet.

Mécanisme de collecte du contexte

Le mécanisme de collecte du contexte permet à CAMidO d'interagir avec les capteurs pour collecter tous les contextes pertinents pour l'application (les informations de l'environnement, les ressources des machines et les profils de l'utilisateur). Cette tâche est assurée par l'entité *CollectionManager* qui utilise les différents niveaux du méta-modèle pour activer les agents appropriés. Un agent est une entité logiciel qui se charge de communiquer avec un capteur local ou distant (cf.

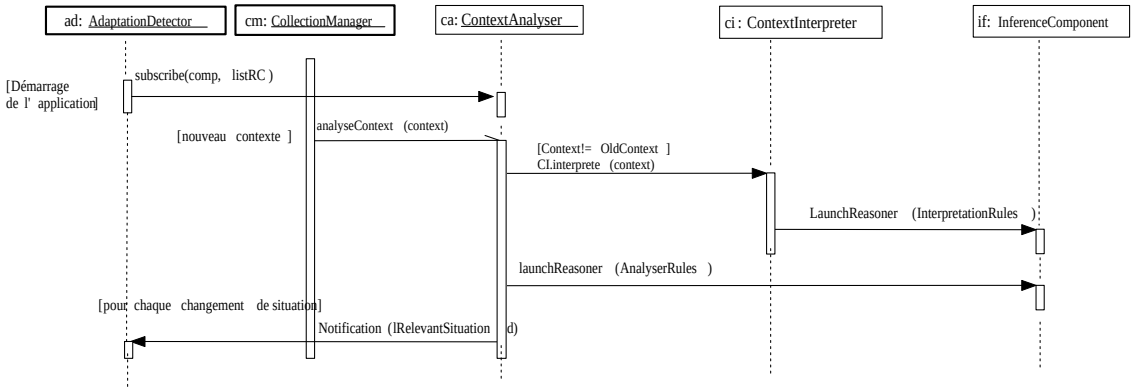

FIG. 4.3 – Diagramme de séquences des entités de gestion du contexte

figure 3.8) pour collecter un contexte. L'ontologie associée au niveau *Capteur*, offre à l'intergiciel la description des capteurs et la structure des informations qu'ils fournissent.

Certains contextes comme le profil de l'utilisateur par exemple sont statiques, ils ne seront collectés qu'une seule fois lors du lancement de l'application. D'autres contextes sont dynamiques, les capteurs doivent les collecter à une fréquence déterminée lors de la description du contexte (cf. figure 3.10).

Lors du lancement de l'intergiciel CAMidO, le mécanisme de collecte du contexte parcourt l'ontologie associée au niveau *Capteur* du méta-modèle pour instancier des agents qui communiquent avec les capteurs. Chaque agent est représenté par une entité logiciel qui fait appel à la méthode de communication avec le capteur décrite dans l'ontologie. Comme précisé dans la section 3.5.1, deux types de capteurs peuvent être décrits dans l'ontologie, les capteurs qui fournissent les informations de contexte à travers un serveur CORBA, et ceux qui les fournissent en instanciant une classe Java. La communication entre les agents et les capteurs est réalisée de différentes manières, selon le type des capteurs.

La figure 4.4 illustre le diagramme de classes des entités impliquées dans la collecte de contexte. La classe *ORBSensorAgent* correspond aux agents clients qui communiquent avec des serveurs de contexte à travers le bus ORB. La communication entre ces deux objets est réalisée à l'aide du mécanisme d'Interface d'Invocation Dynamique (DII) [13] qui permet de construire dynamiquement des requêtes en utilisant la classe *GenericDII*. La classe *ClassSensorAgent* représente un agent instancié par le *CollectionManager*. Cette instanciation est effectuée en utilisant le mécanisme de réflexivité fourni par la classe *ReflexivInvocation*. Chacune des classes *ClassSensorAgent* et *ORBSensorAgent* utilise la classe *OntologyAccess* pour accéder au méta-modèle afin de récupérer des informations sur les capteurs et les contextes.

Le collecteur de contexte peut interagir avec des nouveaux capteurs sans que cela nécessite l'arrêt de CAMidO. La prise en compte des nouveaux capteurs est assurée grâce au *SensorUpdateDaemon*, qui contient un thread démon que le collecteur de contexte active lors du lancement de l'intergiciel. Ce démon se charge de détecter les changements subis par l'ontologie associée au

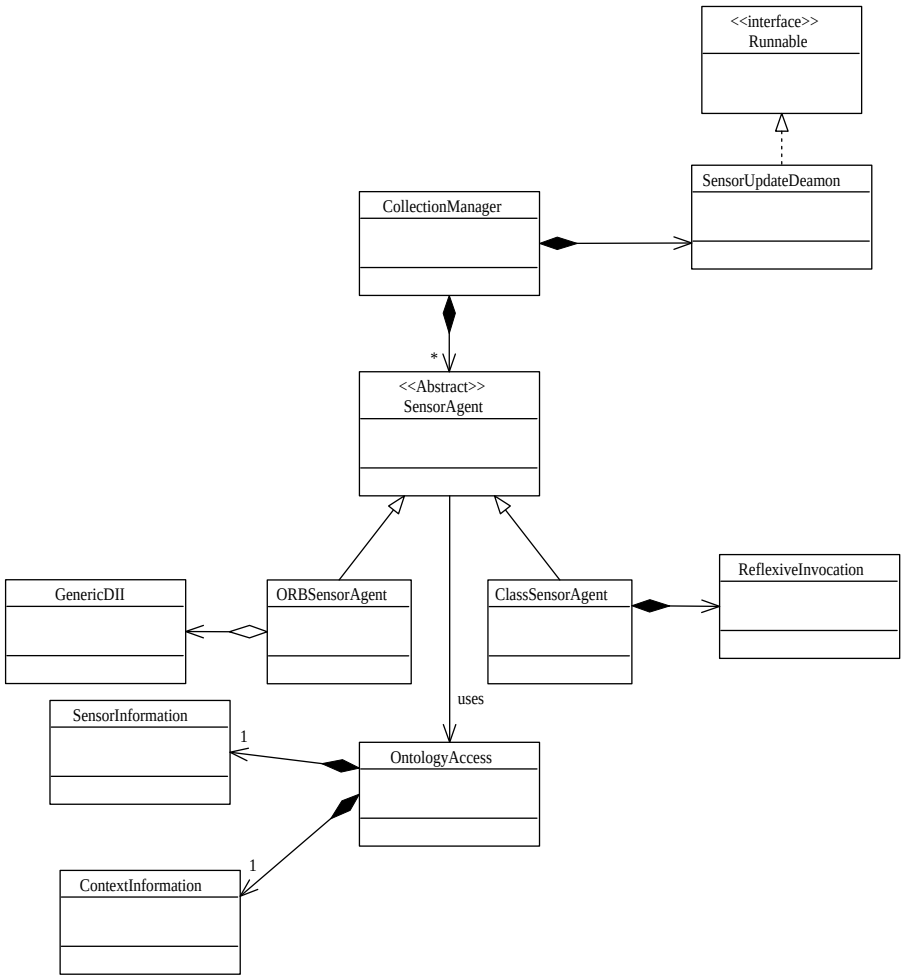

FIG. 4.4 – Diagramme de classes des entités impliquées dans la collecte de contexte

troisième niveau du méta-modèle. Il peut demander le cas échéant au *CollectionManager* d'activer un agent associé à un capteur si le changement consiste en l'ajout d'un nouveau capteur, ou à désactiver un agent s'il s'agit de la suppression d'un capteur.

Mécanisme d'interprétation du contexte

Lors du changement de valeur d'un des contextes observés, le contexte interprété peut changer à son tour. De ce fait, la détection d'une nouvelle valeur du contexte nécessite le lancement du mécanisme d'interprétation (cf. figure 4.3). Ce mécanisme est exécuté grâce aux entités *ContextInterpreter* et *InferenceComponent* qui se chargent de lancer un moteur d'inférence qui permet de raisonner sur les informations de contexte.

Comme nous l'avons décrit dans la figure 4.3, le *ContextAnalyser* se charge de détecter un changement dans les valeurs des contextes collectées. Lors de la détection d'une nouvelle valeur d'un contexte, le *ContextAnalyser* transmet cette valeur au *ContextInterpreter*. Le *ContextInterpreter* fait appel à l'entité *InferenceComponent* qui utilise les règles d'interprétation décrites dans le méta-modèle de CAMidO pour déduire des contextes de haut niveau. Les valeurs de contexte interprétées sont stockées dans le *Repository* puis transmises à l'entité *ContextAnalyser*. Le *ContextAnalyser* se charge de détecter les variations de situations pertinentes du contexte.

Mécanisme d'analyse du contexte

Le mécanisme d'analyse du contexte dans CAMidO est effectué par le *ContextAnalyser* en collaboration avec l'entité *InferenceComponent*.

Le *ContextAnalyser* se charge tout d'abord de détecter les changements des valeurs des contextes collectées, puis, après le retour de l'interpréteur, utilise l'ensemble des nouvelles valeurs de contexte (collectées et interprétées) pour raisonner en utilisant l'entité *InferenceComponent*. Le moteur d'inférence détecte les nouvelles situations pertinentes et celles qui ne sont plus valides en utilisant des règles de filtrages générées par CAMidO à partir du modèle de l'application (cf. section 4.5). Le *ContextAnalyser* se charge de notifier à travers le gestionnaire de contexte les composants sensibles aux situations pertinentes détectées et les composants qui ont été notifiés pour des situations pertinentes alors que ces dernières ne sont plus valides.

4.3.2 Architecture du gestionnaire d'adaptation

Comme nous l'avons mentionné ultérieurement, nous considérons que la granularité de l'adaptation est le composant. Nous exploitons le paradigme composant/conteneur pour gérer l'adaptation de chaque composant à l'aide de propriétés extra-fonctionnelles. De ce fait, nous proposons une nouvelle architecture du conteneur pour lui permettre la gestion de l'adaptation en lui intégrant des contrôleurs. Comme l'illustre la figure 4.5, ces contrôleurs travaillent en collaboration avec des intercepteurs de requêtes et un mandataire pour gérer les types d'adaptation considérés par CAMidO. Dans cette section, nous décrivons progressivement cette architecture.

Nous commençons tout d'abord par décrire l'architecture du conteneur sensible au contexte. Par la suite, nous décrivons le mécanisme d'adaptation et le rôle des intercepteurs de requêtes et du mandataire dans la gestion de l'adaptation.

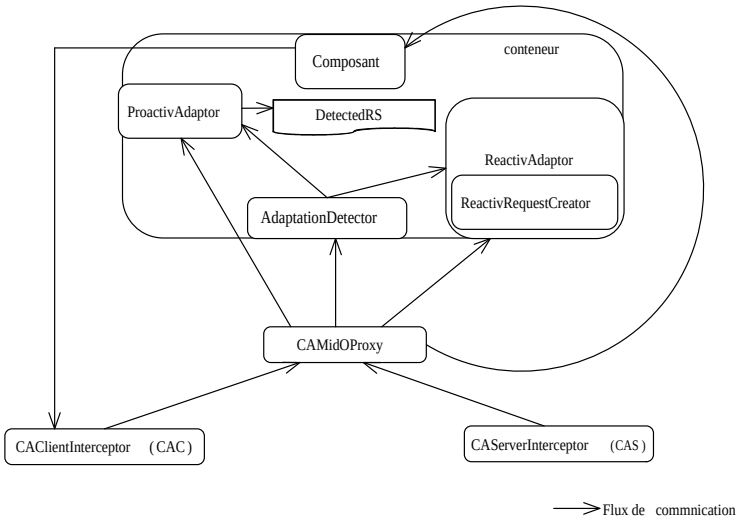

FIG. 4.5 – Architecture du gestionnaire d'adaptation

Architecture du conteneur sensible au contexte

Nous avons proposé une nouvelle architecture du conteneur [1] en lui intégrant un ensemble de contrôleurs comme l'illustre la figure 4.5. Ces contrôleurs se chargent de gérer l'adaptation comportementale de l'application aux changements pertinents du contexte.

Le contrôleur *AdaptationDetector* (AD) se charge d'enregistrer le composant auprès du *ContextAnalyser* pour les situations pertinentes auxquelles il est sensible. Il permet aussi de filtrer les notifications reçues dans le but d'invoquer le contrôleur concerné (le *ProactiveAdaptor* ou le *ReactivAdaptor*). Le contrôleur *ReactivAdaptor* permet d'appliquer l'adaptation réactive associée à la situation pertinente détectée. Le contrôleur *ProactivAdaptor* permet d'enregistrer les nouvelles valeurs des situations pertinentes ou de supprimer les valeurs qui ne sont plus valides. Il permet aussi d'appliquer l'adaptation proactive appropriée lors de l'interception d'une invocation d'opération. Nous décrivons un peu plus loin le mécanisme d'interception des requêtes utilisé dans CAMidO.

Lors de son démarrage, une application doit souscrire tous les composants qui la constituent auprès du gestionnaire de contexte de CAMidO (cf. section 4.3) pour avoir connaissance des situations pertinentes détectées. Cette souscription est réalisée par le contrôleur AD de chaque composant une fois l'installation et la configuration du composant terminées. Lors de la détection d'une situation pertinente, le gestionnaire de contexte notifie les composants qui se sont enregistrés pour cette situation et ceux dont la situation pertinente n'est plus valide. L'entité AD filtre

[1] conteneur de CCM.

les messages reçus et les transmet au *ProactivAdaptor* ou au *ReactivAdaptor* qui se chargent d'appliquer les réactions nécessaires.

```
interface AdaptationDetector
{
    oneway void Notification (in string Relevant_situation_detected, in string id);
};

```

FIG. 4.6 – Spécifi cation en OMG IDL des interfaces de gestion des contrôleurs

La spécification OMG IDL associée au contrôleur *AdaptationDetector* est décrite par la figure 4.6. L'opération *Notification* permet au *ContextAnalyser* de notifier le composant lors de la détection d'une situation pertinente.

La spécification OMG IDL associée au contrôleur *ReactivAdaptor* est illustrée par la figure 4.7. Cette interface est constituée de la méthode *launchAdaptation* qui se charge d'invoquer l'opération d'adaptation associée à la situation pertinente décrite à l'aide de son nom (RelevantSituation) et de son identificateur (idRelevantSituation). Les opérations d'adaptation à invoquer sont de trois types : des opérations fournies par un paquetage externe, des opérations fournies par le composant à adapter et des opérations fournies par un autre composant. Selon le type de l'opération d'adaptation, la méthode *launchAdaptation* fait appel au mécanisme approprié pour invoquer cette opération. La méthode *NotifyReactivAdaptor* permet au contrôleur *AdaptationDetector* de notifier le *ReactivAdaptor* pour qu'il applique l'adaptation associée à la situation pertinente détectée.

```
public interface ReactivAdaptor
{
    public void  launchAdaptation(String RelevantSituation, String idRelevantSituation);
    void NotifyReactivAdaptor(String RelevantSituation);
};
```

FIG. 4.7 – Interface *ReactivAdaptor*

La spécification OMG IDL associée au contrôleur *ProactivAdaptor* est décrite par la figure 4.8. La méthode *add_detected_RS* se charge de sauvegarder la référence et le nom de la situation pertinente détectée dans une structure de données. La méthode *remove_RS* se charge de supprimer la situation pertinente de la structure de donnée lorsque celle-ci n'est plus valide. L'opération *ProactivAdaptationToApply* permet de fournir des informations sur l'adaptation proactive associée à l'invocation de l'opération *OperationName*. La méthode *NotifyProactivAdaptor* permet au contrôleur *AdaptationDetector* de notifier le *ProactivAdaptor* de la détection d'une situation pertinente.

Mécanismes d'adaptation

L'adaptation représente le cœur de toute application sensible au contexte. Dans le cadre de CAMidO, nous avons considéré dans cette thèse l'adaptation comportementale de nature réactive et proactive (cf. section 3.4.1).

```
1 public interface ProactivAdaptor
2 {
3     void  add_detected_RS(in String RelevantSituation, in String id);
4     void  remove_RS(in String RelevantSituation, in String id);
5     String ProactivAdaptationToApply(String OperationName, boolean IncomingRequest);
6     void NotifyProactivAdaptor(String listRelevantSituation);
7 };
```

FIG. 4.8 – Interface *ProactivAdaptor*

Le mécanisme d'adaptation est géré par les contrôleurs d'adaptation installés dans le conteneur de composant, à savoir l'*AdaptationDetector*, le *ProactivAdaptor* et le *ReactivAdaptor*. Chaque contrôleur joue un rôle bien déterminé dans le mécanisme d'adaptation comme le montre le diagramme de séquence de la figure 4.9. Lors de la notification de l'*AdaptationDetector* par le *ContextAnalyser*, il envoi à son tour un message de notification soit vers le *ReactivAdaptor* si une adaptation réactive est nécessaire soit vers le *ProactivAdaptor* si une adaptation proactive est nécessaire ou qu'elle doit être annulée. Le cas échéant, le *ReactivAdaptor* se charge dès lors d'invoquer l'opération d'adaptation associée à la situation pertinente qui a été détectée, cette invocation consiste soit à créer une requête dynamiquement en utilisant le mécanisme DII, ou bien à utiliser la réflexivité pour invoquer une opération fournie par un composant externe. Le *ProactivAdaptor* se charge de mettre à jour la structure *DetectedRS* soit en enregistrant de nouvelles situations pertinentes ou en supprimant celles qui ne sont plus valides. Ce ne sera que lorsqu'un composant émet ou reçoit une invocation d'une opération sensible à l'une des situations pertinentes détectées par l'intergiciel et enregistré par le *ProactivAdaptor* que l'opération adéquate sera invoquée.

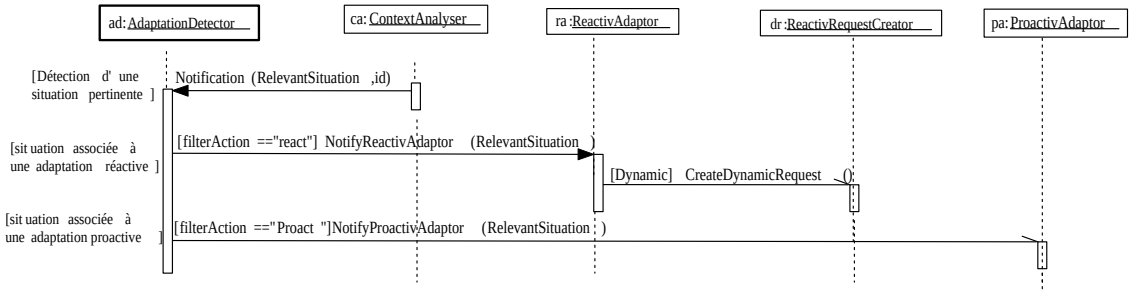

FIG. 4.9 – Diagramme de séquences du mécanisme d'adaptation

L'adaptation proactive consiste à rediriger lors de leurs appels des opérations vers d'autres opérations. Plus précisément, ce mécanisme consiste à intercepter les invocations vers chaque composant et les ré-aiguiller vers les opérations appropriées. Les opérations vers lesquelles les invocations du composant sont ré-aiguillées sont définies dans le modèle de l'application. L'adaptation proactive peut être exécutée du côté client et/ou du côté serveur de chaque composant. En effet, un composant peut altérer les invocations qu'il émet ou reçoit en fonction du contexte.

L'adaptation proactive côté client (cf. figure 4.10) consiste à intercepter les appels d'invocations d'une opération vers un composant par le CRA (*Client Request Adaptor*) et à les ré-aiguiller vers une autre opération ayant les mêmes paramètres. Cette opération peut être fournie par le même composant (de O1 vers O2 dans l'exemple de la Figure 4.10), ou bien par un autre composant (de O1 vers O3 dans la figure 4.11). L'adaptation proactive côté serveur (cf. figure 4.11) à pour rôle d'intercepter les appels entrants par le SRA (*Server Request Adaptor*), afin de fournir si nécessaire une opération ayant un comportement différent de celle qui a été invoquée par le composant.

Par rapport aux entités décrites dans la figure 4.5, le CRA regroupe le *CAClientInterceptor*, le *CAMidOProxy* et le *ProactivAdaptor*. Le SRA est composé du *CAServerInterceptor*, du *CAMidO-Proxy* et du *ProactivAdaptor*

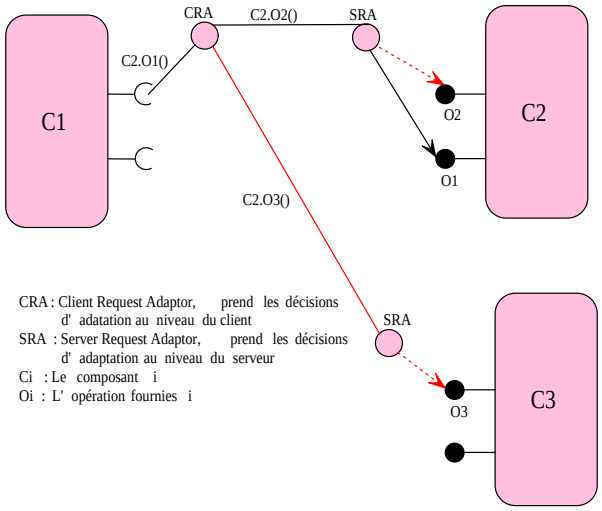

FIG. 4.10 – Adaptation proactive côté client

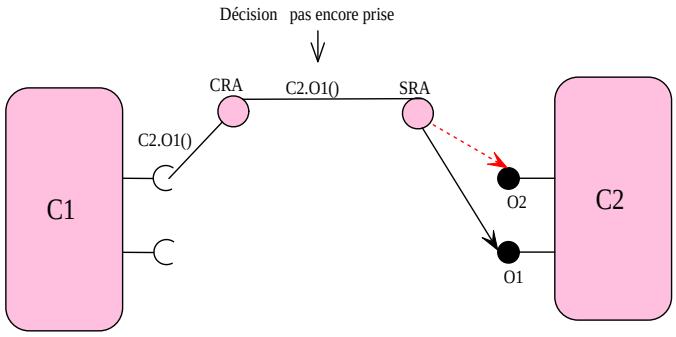

FIG. 4.11 – Adaptation proactive côté serveur

Pour intercepter les invocations d'un composant, nous utilisons les intercepteurs portables CORBA [77]. Ce choix est motivé par le fait que les intercepteurs portables permettent de modifier

le comportement d'un composant pour les invocations reçues ou émises sans changer le code du composant.

Les invocations d'opérations sont transformées par l'intergiciel en requêtes. Les intercepteurs portables sont des objets CORBA qui, comme le montre la figure 4.12, peuvent être appelés à des points particuliers du traitement de la requête. Dans le cadre de CAMidO, nous utilisons deux points d'interception, le *send_request* et le *receive_request* qui permettent d'intercepter respectivement les émissions et les réceptions des requêtes.

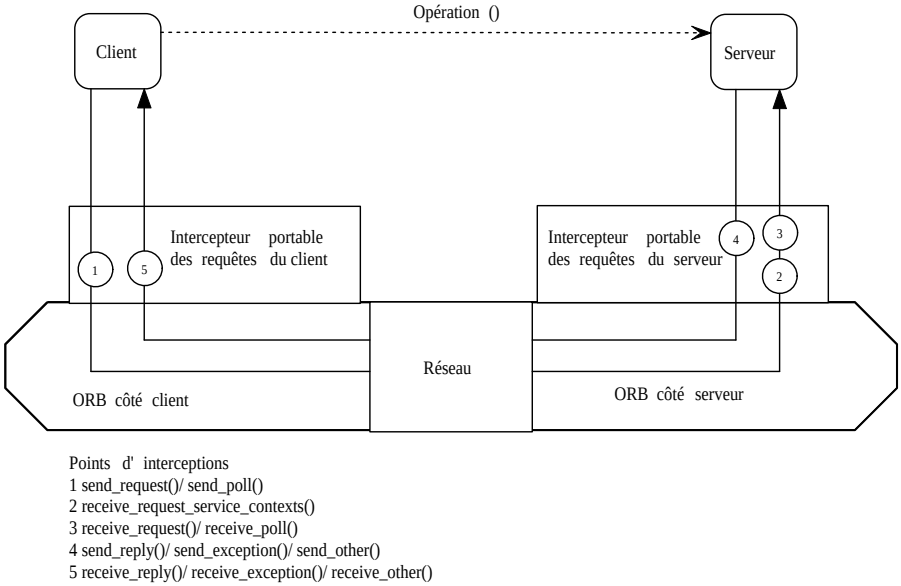

FIG. 4.12 – Points d'interception d'une requête

La figure 4.13 illustre l'architecture des entités qui participent au mécanisme d'adaptation proactive côté client. L'intercepteur *CAClientRequestInterceptor* intercepte toutes les requêtes invoquées par les composants qui s'exécutent au-dessus du bus ORB (O1 sur la figure). Cet intercepteur filtre ces requêtes et redirige celles qui nécessitent un traitement vers le proxy *CAMidOProxy* (O2 sur la figure). Pour chaque requête, le *CAMidOProxy* consulte le *ProactivAdaptor* associé au composant qui a émis la requête pour déterminer si cette invocation nécessite une adaptation (O3 sur la figure). Le *CAMidOProxy* modifie cette requête selon le type d'adaptation qu'il doit effectuer (changement du nom et/ou des paramètres de l'opération, redirection vers un autre composant), puis la remet sur le bus ORB (O4 sur la figure).

La figure 4.14 illustre l'architecture des entités qui participent au mécanisme d'adaptation proactive côté serveur. L'intercepteur *CAServerRequestInterceptor* intercepte les requêtes reçues via le bus ORB (I1 sur la figure). Il redirige les requêtes vers le *CAMidOProxy* (I2 sur la figure). Après avoir consulté le *ProactivAdaptor* associé au composant qui fournit les opérations invoquées (I3 sur la figure), le *CAMidOProxy* se charge d'effectuer les adaptations nécessaires sur ces requêtes. Il transmet ensuite la requête vers le composant (I4 sur la figure).

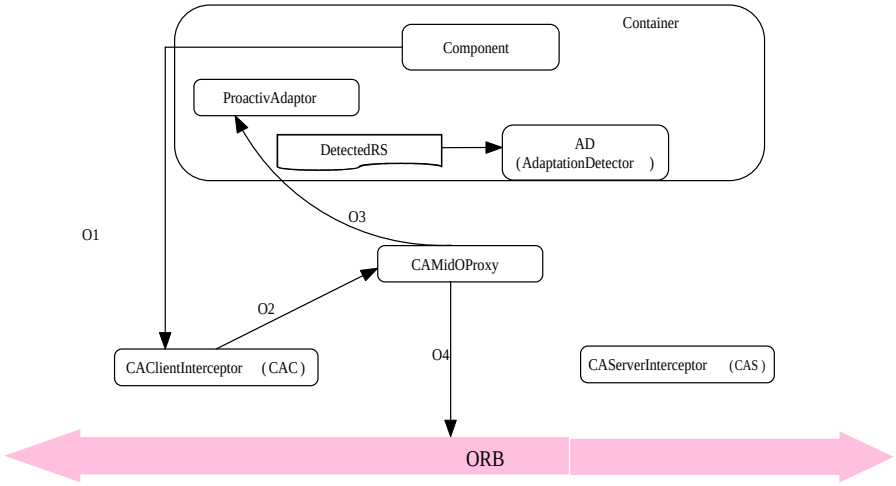

FIG. 4.13 – Mécanisme d'adaptation proactive côté client

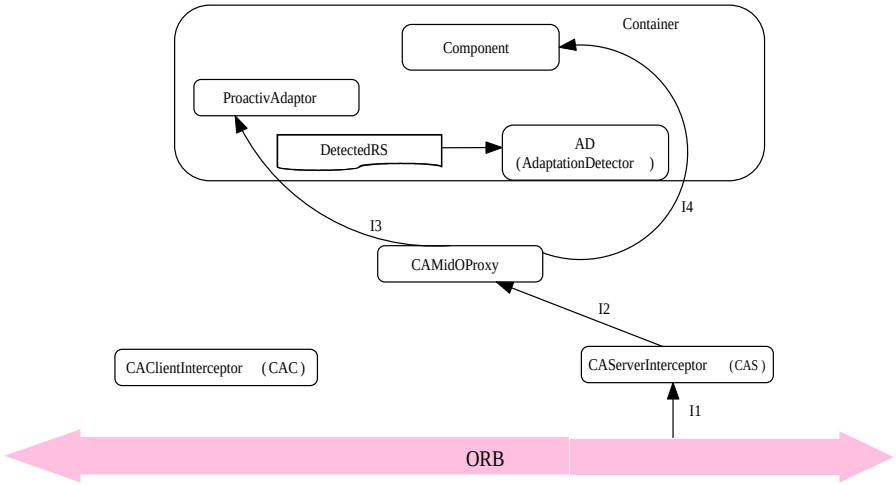

FIG. 4.14 – Mécanisme d'adaptation proactive côté serveur

La figure 4.15, illustre le diagramme de classes du mécanisme d'interception des requêtes. La mise en place d'un intercepteur portable nécessite son installation. Cette installation est effectuée par la classe *CAMidOInterceptor*. Elle se charge d'enregistrer chaque type d'intercepteur auprès de l'ORB. Le bus ORB active le *CAClientRequestInterceptor* à chaque fois qu'une requête est émise par un composant. Il active le *CAServerRequestInterceptor* à chaque fois qu'une requête est transmise vers un composant. Le *CAClientRequestInterceptor* (*CAServerRequestInterceptor*) implémente l'interface *ClientRequestInterceptor* (*ServerRequestInterceptor*) pour pouvoir intercepter les requêtes émises par (transmises vers) un composant. La redirection des requêtes vers le *CAMidOProxy* est effectuée en levant l'exception CORBA LOCATION_FORWARD. Cette exception permet de rediriger la requête émise par un composant avant qu'elle n'atteigne le bus ORB. Elle permet aussi de rediriger les requêtes transmises vers un composant avant qu'elles n'atteignent le composant cible.

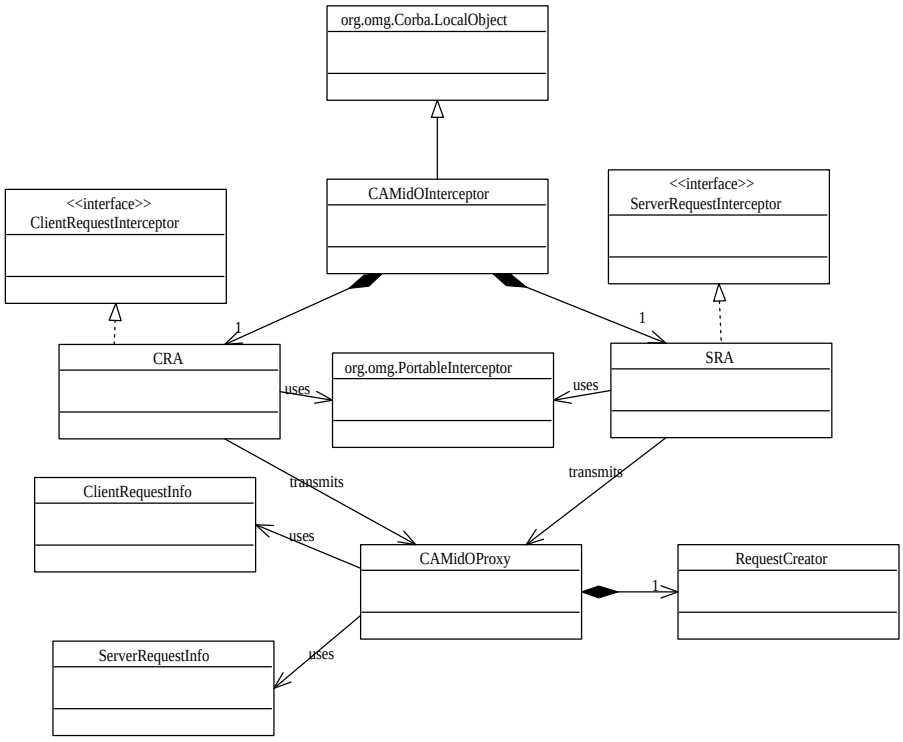

FIG. 4.15 – Diagramme de classes du mécanisme d'interception des requêtes

4.4 Modes d'exécution de CAMidO

L'intergiciel CAMidO offre deux modes d'exécution : le mode configuration statique et le mode reconfiguration dynamique. La configuration statique de l'intergiciel consiste à lui fournir les informations nécessaires à la gestion des applications sensibles au contexte[2]. Le changement de

[2]Ces informations concernent les contextes pertinents auxquels l'application est sensible, les situations pertinentes, les politiques d'interprétation des contextes de haut niveau si elles n'existent pas déjà ou si le concepteur veut en utiliser

ces informations après l'exécution de l'application n'est pas possible. En mode reconfiguration dynamique, au contraire il est possible de changer l'un des éléments de configuration y compris pendant l'exécution de l'application soit pour améliorer le comportement de l'application, ou pour prendre en compte de nouvelles données non considérées lors de sa conception, par exemple une situation pertinente qui n'a pas été prise en compte lors de la conception, mais pour laquelle l'utilisation de l'application a montré que cette situation pertinente nécessite l'adaptation de l'application. Dans ce qui suit, nous allons décrire ces deux modes d'exécution.

4.4.1 Mode configuration statique

Le mode configuration statique permet à CAMidO de gérer des applications orientées composant sensibles au contexte en considérant la description initiale du modèle associé à chaque application. En effet, ce mode ne prend pas en compte les modifications effectuées sur le modèle de l'application après le lancement de son exécution. Par conséquent, la prise en compte des nouvelles données ajoutées au modèle après le lancement de l'application nécessite l'arrêt, la recompilation, puis le redéploiement de l'application concernée.

Le mode configuration statique utilise le compilateur CAMidO que nous décrivons dans la section 4.5 et la description fournie par le concepteur de l'application pour générer un ensemble d'informations dont l'intergiciel a besoin pour gérer la sensibilité au contexte. Le compilateur CAMidO est un processus lancé lors de la compilation de l'application. Il parcourt le modèle de chaque application pour générer du code de gestion du contexte.

4.4.2 Mode reconfiguration dynamique

L'exécution de CAMidO en mode reconfiguration dynamique permet à l'intergiciel de prendre en compte les mises à jour effectuées sur le modèle après le lancement de l'application. Ces modifications peuvent être nécessaires pour ajouter des éléments non pris en compte lors de la conception de l'application. Ces éléments concernent les situations pertinentes, les règles d'interprétations et les politiques d'adaptations.

Le mode reconfiguration n'utilise que la partie du compilateur qui concerne la génération de code. Ce mode se base sur la réflexivité comportementale (cf. section 2.4.3) pour inspecter dynamiquement le modèle. En effet, dans le mode reconfiguration dynamique, CAMidO utilise des processus génériques d'interprétation, d'analyse du contexte et d'adaptation de l'application qui réifient le méta-modèle pour diriger leur comportement. Grâce à cela, l'intergiciel CAMidO peut prendre en compte toutes les mises à jours effectuées dans le méta-modèle sans arrêter l'exécution de l'application.

d'autres, et les règles d'adaptation des composants qui constituent l'application. Ces informations permettent de diriger l'intergiciel dans les tâches d'analyse, d'interprétation du contexte et d'adaptation de l'application.

4.5 Compilateur CAMidO

L'exécution de l'intergiciel CAMidO en mode configuration statique et en mode reconfiguration dynamique nécessite l'utilisation préalable pour chaque application d'un compilateur. Comme le montre la figure 4.16, le compilateur CAMidO utilise le modèle de description du contexte associé à l'application et un fichier de patrons (*templates*) pour générer d'une part le code source d'adaptation et d'autre part des fichiers de règles de logique de premier ordre. Les fichiers et le code générés servent à automatiser les processus d'adaptation et d'interprétation du contexte. Le code source d'adaptation représente le code des contrôleurs que nous avons intégrés à la nouvelle architecture du conteneur (cf. section 4.3.2). Les fichiers de règles regroupent des règles d'interprétation du contexte et des règles de détection des situations pertinentes. Ces règles sont une traduction des politiques d'adaptation et des règles d'interprétation fournies par le modèle de l'application, ainsi que des règles d'interprétation et de détection des situations pertinentes déduites à partir des relations existantes entre les contextes observables et les situations pertinentes.

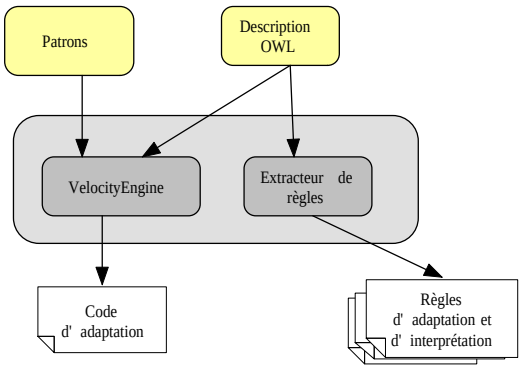

FIG. 4.16 – Architecture du compilateur CAMidO

Pour atteindre cet objectif, le compilateur CAMidO utilise un moteur de vélocité [1] et un parseur d'ontologie. Le parseur d'ontologie sert à parcourir l'ontologie et à extraire les informations nécessaires à la génération des fichiers de règles. Le moteur de vélocité, qui est un outil basé sur des patrons, sert à générer le code source nécessaire aux mécanismes d'analyse du contexte, d'interprétation du contexte et d'adaptation de l'application en se basant sur des partons qui décrivent le modèle de chaque code à générer.

La figure 4.17, illustre le diagramme de classes du compilateur CAMidO. La classe *CAMidOCompiler* représente la classe de base de ce compilateur. Cette classe utilise la classe *VelocityEngine* et la classe *RuleFileGenerator* pour générer respectivement le code source d'adaptation et les fichiers de règles de logique de premier ordre. La classe *ParseOntology* parcourt le modèle de l'application pour extraire des informations sur les situations pertinentes (*RelevantSituation*), les règles d'interprétation (*InterpretationPolicies*), et des informations sur les composants sensibles au contexte (*CAComponentInformation*). Ces informations sont exploitées respectivement par le *RuleFileGenerator*, qui représente la classe associée à l'extracteur de règles illustré dans

la figure 4.16, et le *VelocityEngine*. En plus de ces informations, la classe *VelocityEngine* utilise un ensemble de patrons pour générer le code d'adaptation de chaque contrôleur appartenant au composant ainsi que le code associé à l'interprétation et à la détection des situations pertinentes. La classe *ComponentName_ProactivAdaptorImpl* représente le code associé au contrôleur *ProactivAdaptor*. La classe *ComponentName_ReactivAdaptorImpl* représente le code associé au contrôleur *ReactivAdaptor*. La classe *ComponentName_AdaptationDetectorImpl* représente le code associé au contrôleur *AdaptationDetector*. La classe *Method_Impl* représente les classes générées par le compilateur CAMidO qui héritent de la classe *BaseBuiltin*. Ces classes sont utilisées par le moteur d'inférence pour interpréter un contexte de haut niveau ou pour détecter des situations pertinentes.

Dans le suite de cette section, nous décrivons plus en détail le mécanisme de génération des fichiers de règle et le mécanisme de génération de code.

4.5.1 Génération de fichier de règles

En mode configuration statique, le compilateur CAMidO utilise les descriptions fournies par le modèle de l'application pour générer deux types de fichiers de règles de logique de premier ordre : le type *InterpretationRule* et le type *RelevantSituationRule*. Ces règles sont destinées à être utilisées par le moteur d'inférence représenté par l'entité *InferenceComponent* que nous avons intégré à la plate-forme de CAMidO pour raisonner sur les informations de contexte (cf. section 4.3). Le type *InterpretationRule* regroupe les règles d'interprétation des contextes de haut niveau, alors que le type *RelevantSituationRule* regroupe des règles qui permettent de filtrer le contexte et de détecter la présence de situations pertinentes à partir des observations du contexte. Le compilateur ne génère pas ces fichiers de règles lorsque CAMidO est exécuté en mode reconfiguration dynamique. En effet, dans ce mode, l'ontologie est parcourue à l'exécution de l'application et les règles sont alors recalculées à chaque traitement d'un contexte observé.

Règles d'interprétation

La génération des règles d'interprétation en logique de premier ordre nécessite l'utilisation des règles d'interprétation décrites dans le modèle de l'application ainsi que les relations d'équivalence entre les informations de contexte (cf. figure 3.13). Les règles générées sont des traductions des règles fournies par le modèle en forme de règles de logique du premier ordre et des règles déduites en utilisant les relations d'équivalences entre les informations de contexte.

Comme nous l'avons décrit dans le chapitre 3 à la section 3.5.2, l'intergiciel CAMidO gère deux types d'interprétation. Le premier type permet de faire appel à une méthode externe à la plate-forme pour calculer la valeur d'un contexte indirect. Le second type utilise des règles qui utilisent une condition sur des valeurs du contexte pour déterminer la valeur que doit prendre un contexte indirect.

La figure 4.18 illustre un exemple de règle d'interprétation en logique du premier ordre tel qu'elle est générée par le compilateur CAMidO à partir d'une règle OWL qui fait appel à une

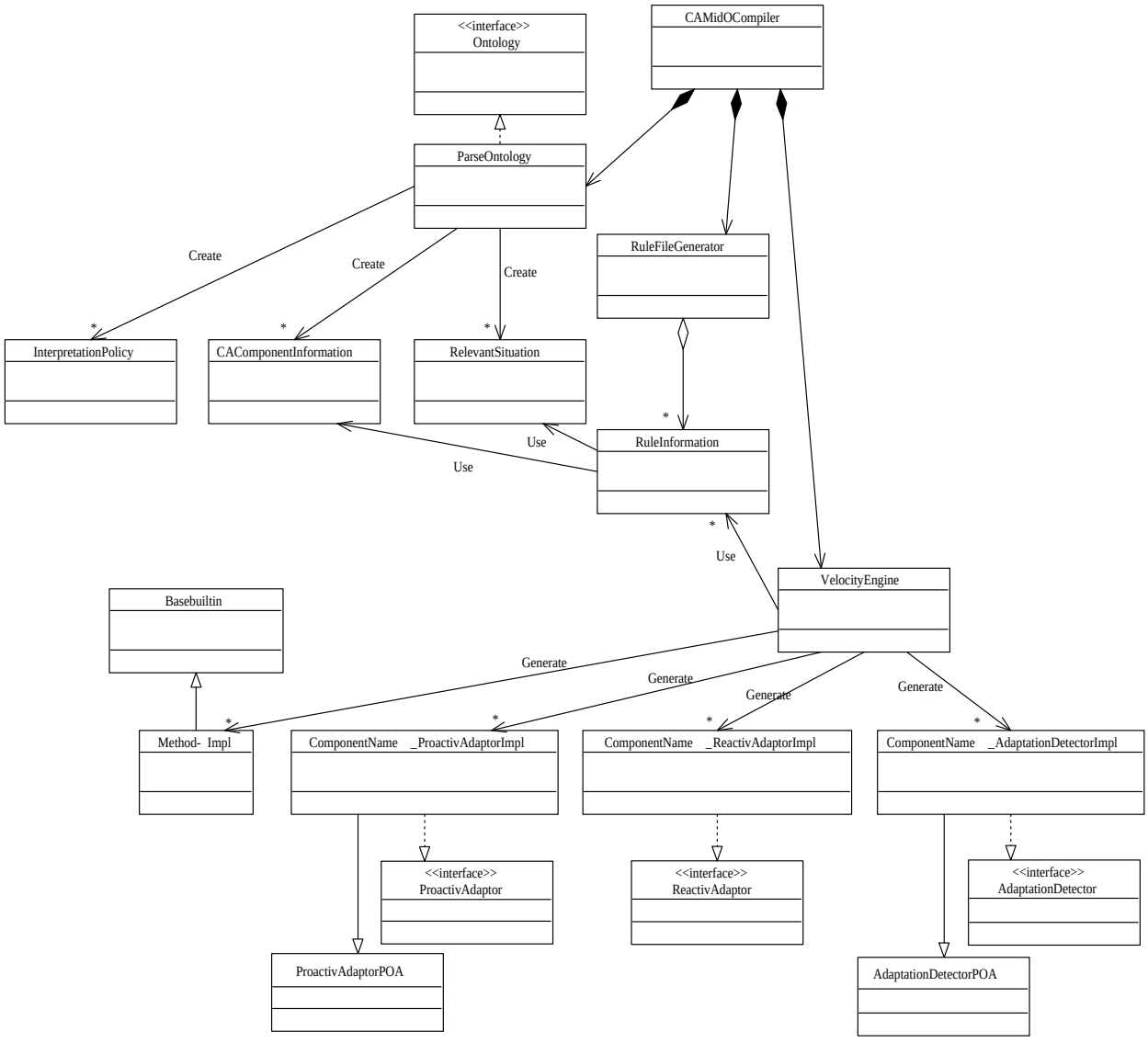

FIG. 4.17 – Diagramme de classes du compilateur CAMidO

méthode externe. La règle fournie par le modèle de l'application, stipule que la déduction du contexte indirect *LocationTownName* est faite en invoquant la méthode *ComputeLocationTown-Name* fournie par le paquetage *Sensor.Location.LocationDeduction*. En utilisant l'extracteur de règles, le compilateur CAMidO génère la règle en logique de premier ordre *DeduceLocationTown*. Cette règle est structurée en deux parties, la partie gauche décrit une condition qui vérifie que le contexte à déduire est une instance de la classe *IndirectContext* et la partie droite décrit une action. Cette action dont le code est généré par le compilateur CAMidO, consiste à faire appel à la méthode *headAction* (cf. section 1.6.4) de la classe *ComputeLocationTownName_Impl* en lui passant le nom du contexte indirecte à déduire en paramètre, et à mettre à jour l'ontologie avec la valeur du contexte déduite.

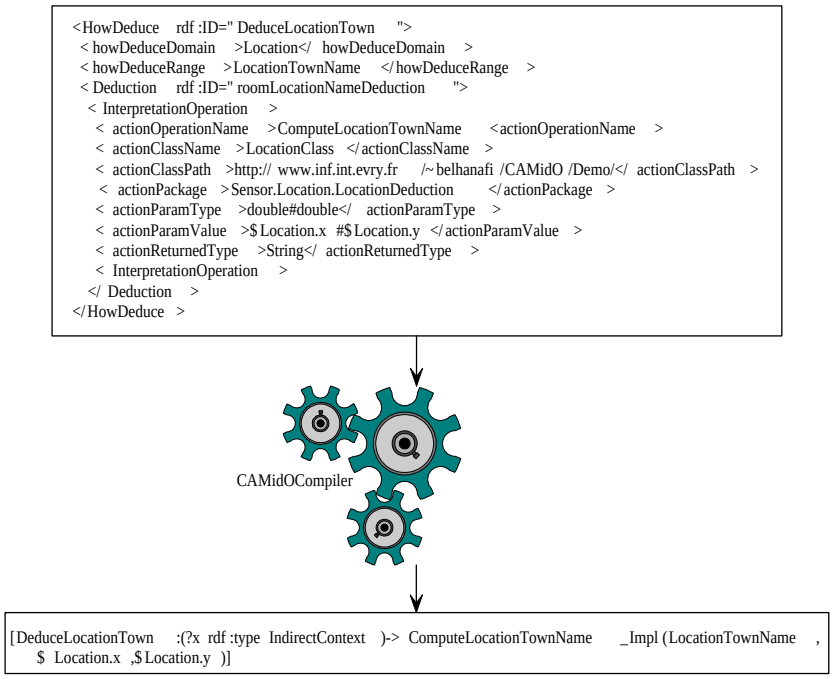

FIG. 4.18 – Exemple de la règle *DeduceLocation* générée

La figure 4.19 illustre une règle d'interprétation générée par le compilateur CAMidO pour le type d'interprétation qui utilise des conditions sur les valeurs du contexte. Les règles générées *DeduceConnexionState* et *DeduceConnexionState2* permettent de déduire l'état de la connexion réseau en utilisant des informations sur la bande passante. La règle *DeduceConnexionState* permet d'affecter la valeur *Connected* au contexte indirect *ConnexionState* si la valeur de la bande passante est supérieure à 1%, cette affectation est effectuée en invoquant la méthode *headAction* fournie par la classe *NewIndirectContextVal* définit par la plate-forme CAMidO. Cette classe se charge de déduire la nouvelle valeur du contexte et de la sauvegarder dans l'ontologie. La règle *DeduceConnexionState2* permet d'affecter la valeur *Disconnected* au contexte indirect *ConnexionState* en faisant appel à la même méthode.

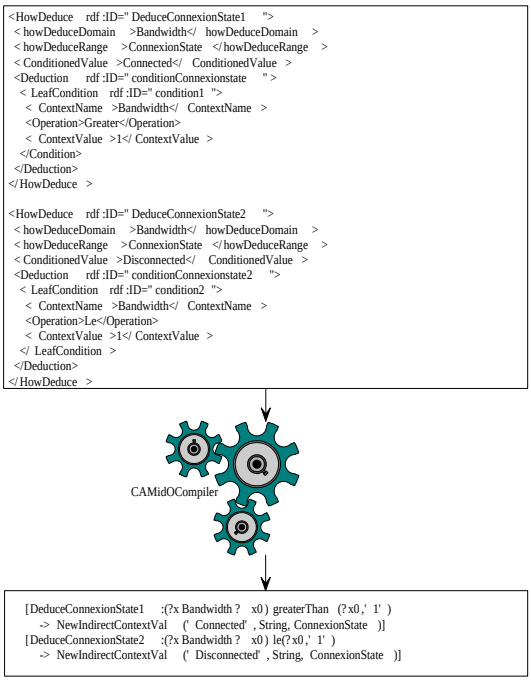

FIG. 4.19 – Génération des règles d'interprétation *DeduceConnexionState1* et *DeduceConnexionState2* à partir de conditions

Les règles d'interprétation sont exécutées par le moteur d'inférence instancié par l'entité *InferenceComponent* (cf. section 4.3) qui se charge d'automatiser le processus d'interprétation du contexte.

Règles de détection de situation pertinente

En plus des règles d'interprétation, le compilateur CAMidO génère des règles de détection de situations pertinentes en se basant sur les descriptions fournies dans l'ontologie (cf. figure 3.17).

En suivant le même schéma que pour les règles d'interprétation, le compilateur CAMidO génère deux types de règles de détection de situations pertinentes. Le premier type utilise des conditions sur les valeurs du contexte pour filtrer le contexte et ainsi détecter l'existence d'une situation pertinente. Le second type fait appel à des méthodes fournies par l'application pour déduire si une situation pertinente est détectée ou non.

La figure 4.20, illustre les deux types de règles de déduction de situations pertinentes générées par le compilateur CAMidO. La règle *RelevantTemperature* permet de détecter la pertinence de la température actuelle représentée par la variable $*Temperature* pour l'application. Ceci est réalisé en faisant appel à la méthode *bodyCall* (cf .section 1.6.4) fournie par la classe *TemperatureChange_Impl* (cf. figure 4.24). Cette méthode se charge d'invoquer la méthode *TemperatureChange* décrite dans le modèle de l'application qui retourne la valeur booléenne *true* si une situation pertinente a été détectée ou *false* dans le cas contraire. La classe *TemperatureChange_Impl*

est générée à partir de la description de le situation pertinente *RelevantTemperature* fournie par l'ontologie et d'un patron. Le patron représente le modèle que le *VelocityEngine* doit suivre pour la générer. Quand la condition de cette règle est vérifiée, le moteur d'inférence fait appel à la méthode *headAction* de la classe *Notification*. Cette classe est fournie par la plate-forme CAMidO, elle se charge de notifier les entités sensibles à la situation pertinente détectée (cf. section 4.3).

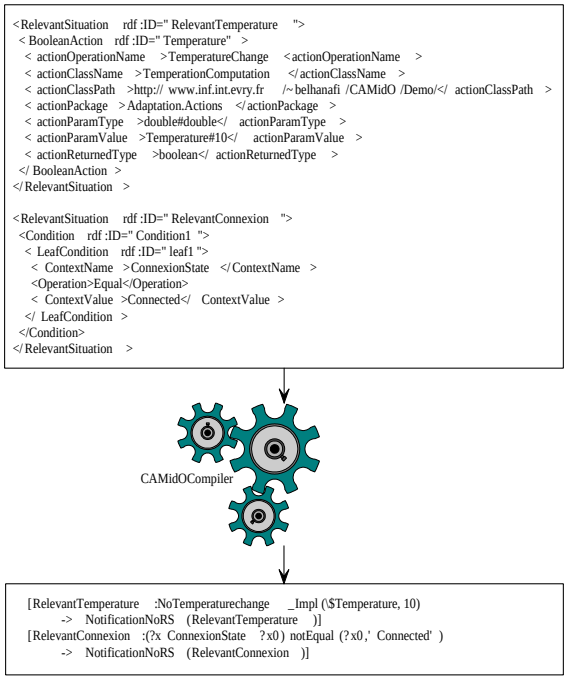

FIG. 4.20 – Exemple de génération de règles de détection de situations pertinentes

De même que pour l'interprétation, pour la détection des situations pertinentes, le compilateur CAMidO utilise les relations entre les informations de contexte décrites dans le méta-modèle (cf. figure 3.22) pour générer de nouvelles règles de filtrage. La figure 4.21 décrit une règle de détection de situation pertinente générée par le compilateur CAMidO en exploitant l'équivalence entre la situation pertinente *RelevantBandwidth*, et la situation pertinente *RelevantConnexion*. En effet, puisque la situation pertinente *RelevantBandwidth* est équivalente à la situation pertinente *RelevantConnexion*, alors les adaptations associées au *RelevantConnexion* sont associées aussi au *RelevantBandwidth*. De ce fait, la règle *RelevantConnexion2* est générée en exploitant cette équivalence. Cette règle permet de notifier tous les composants sensibles à la situation pertinente *RelevantConnexion* lors de la détection de la situation pertinente *RelevantBandwidth*.

Pour chaque situation pertinente décrite dans le modèle de l'application, le compilateur CAMidO génère des règles qui permettent de raisonner sur le contexte et de détecter la non existence de cette situation à un instant donné. Ces règles sont utiles pour l'adaptation proactive. Elles sont utilisées pour notifier les composants de la non validité d'une situation pertinente qui a été préalablement détectée. A la réception de cette notification, le contrôleur *proactivAdaptor* associé à chaque composant supprime cette situation pertinente de la structure *DetectedRS*. La figure 4.22, illustre la génération de ce type de règles à partir de la description d'une situation

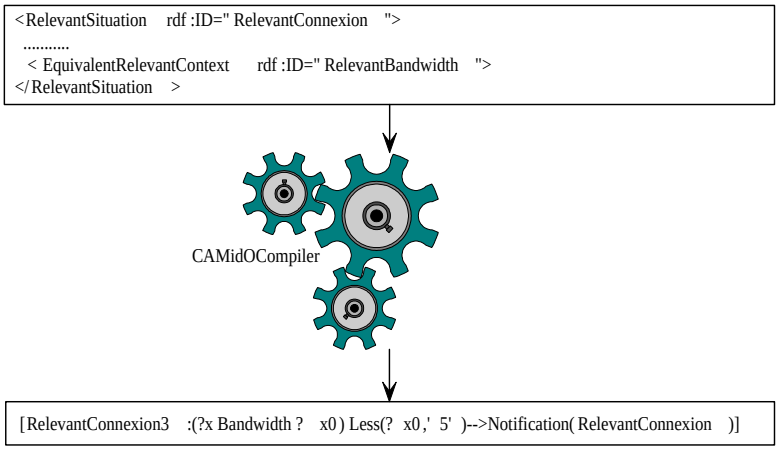

FIG. 4.21 – Exemple de règles de fi ltrage déduite à partir des relations entre les informations de contexte

pertinente. Quand la condition de cette règle est vérifiée, le moteur d'inférence fait appel à la méthode *headAction* de la classe *NotificationNoRS*. Cette classe est fournie par la plate-forme CAMidO, elle se charge de notifier les entités qui se sont enregistrés pour la situation pertinente à laquelle la règle est associé en leur indiquant que cette situation n'est plus valide.

En mode reconfiguration dynamique la génération des règles n'est pas effectuée au moment de la compilation de l'application, mais lors de son exécution à chaque fois qu'une nouvelle observation de contexte doit être traitée par CAMidO. Dans ce mode, les règles d'interprétation ou de détection des situations pertinentes qui consistent à évaluer des conditions sur les valeurs du contexte générées sont les mêmes que ceux générées par le compilateur CAMidO pour le mode configuration statique, alors que les règles qui consistent à faire appel à des méthodes d'évaluation décrites dans l'ontologie ne sont pas exactement les même qu'en mode configuration statique. Au lieu de faire appel à des classes générées par le compilateur CAMidO, les règles générées en mode reconfiguration dynamique font appel à des classes fournies par la plate-forme CAMidO nommées *ReconfigInterpretation* dans le cas des règles d'interprétation et *ReconfigDeduction* dans le cas des règles de détection des situations pertinentes. Ces classes se chargent de faire appel aux méthodes décrites dans le modèle de l'application en utilisant le mécanisme de réflexivité.

4.5.2 Génération de code

En plus des fichiers de règles générés, le compilateur CAMidO génère du code source pour l'interprétation du contexte, la détection de situations pertinentes et l'adaptation de l'application. En mode configuration statique, le code source est généré pour l'interprétation d'un contexte de haut niveau, pour la détection des situations pertinentes et pour l'adaptation de l'application. En mode reconfiguration dynamique, le code est généré uniquement pour l'adaptation de l'application. En effet, l'intergiciel utilise alors des méthodes génériques fournies par CAMidO qui font

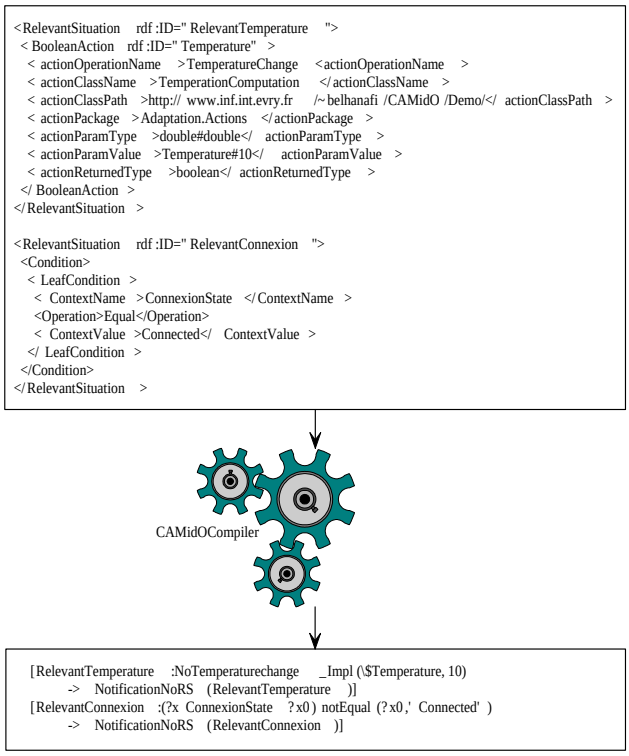

FIG. 4.22 – Exemple de génération de règles pour détecter la non existance de situation pertinente

```
1  import  ${ActionPackage};
2  public class ${ActionClassName}_Impl  extends BaseBuiltin {
3  ContextManager contextmanager;
4  public ${ActionClassName}_Impl(ContextManager contextmanager)
5  {
6    super();
7    this.contextmanager=contextmanager;
8  }
9
10 public void headAction(Node[] args, int length, RuleContext context)
11 {
12   ${ActionClassName} A =new ${ActionClassName}();
13   String retour=A.${actionOperationName}(${actionParameters});
14   contextmanager.UpdateOntology(retour,...);
15 ...
16 }
17
18 }
```

FIG. 4.23 – Patron utilisé pour générer le code d'interprétation

appel aux méthodes d'interprétation et de détection des situations pertinentes décrites dans le modèle en utilisant le mécanisme de réflexivité.

Le compilateur utilise la classe *VelocityEngine* illustrée dans la figure 4.17 ainsi que des templates pour générer le code approprié. Les templates contiennent les modèles des codes source à générer.

Code associé à l'interprétation du contexte

Le compilateur CAMidO utilise le patron décrit dans la figure 4.23 pour générer le code d'interprétation d'un contexte de haut niveau. Les classes générées permettent de calculer les valeurs des contextes de haut niveau en faisant appel aux méthodes d'interprétation décrites dans le le méta-modèle. Nous avons convenu que le nom des classes générées pour l'interprétation du contexte est sous la forme <actionOperationName>_Impl, où *actionOperationName* est une variable de la classe *InterpretationOperation* de la figure 3.13 qui représente le nom de l'opération qui effectue l'interprétation.

Code associé à la détection des situations pertinentes

La figure 4.24, illustre le patron utilisé pour générer le code de détection d'une situation pertinente. La classe générée fait appel à la méthode de détection d'une situation pertinente décrite dans le modèle. Le nom de la classe générée est sous la forme <actionOperationName>_Impl, où *actionOperationName* représente une variable de la classe *BooleanAction* (cf. figure 3.17).

Pour détecter la non existance d'une situation pertinente, le compilateur CAMidO utilise le patron décrit dans la figure 4.25. La classe générée fait appel à la méthode de détection d'une situation pertinente décrite dans le modèle, le nom de la classe générée est sous la forme No<actionOperationName>_Impl. Le *bodyCall* retourne *false* lorsque la situation pertinente est détectée, et *true* dans le cas contraire.

```
1  import ${ActionPackage};
2  public class ${ActionClassName}_Impl  extends BaseBuiltin {
3  ContextManager contextmanager;
4  public boolean bodyCall(Node[] args,int length,RuleContext context)
5  {
6    ${ActionClassName} t=new ${ActionClassName}();
7    if( t.{ActionOperationName}(${actionParameters}))
8       return true;
9    else return false;
10
11 }
12 }
```

FIG. 4.24 – Patron utilisé pour générer le code de détection d'une situation pertinente

```
1  import ${ActionPackage};
2  public class No${ActionClassName}_Impl  extends BaseBuiltin {
3  ContextManager contextmanager;
4  public boolean bodyCall(Node[] args,int length,RuleContext context)
5  {
6    ${ActionClassName} t=new ${ActionClassName}();
7    if( t.{ActionOperationName}(${actionParameters}))
8       return false;
9    else return true;
10 //$
11 }
12 }
```

FIG. 4.25 – Patron utilisé pour générer le code de détection de la non existance d'une situation pertinente

Code d'adaptation

Dans le cadre de CAMidO la gestion de l'adaptation est effectuée par des contrôleurs que nous avons intégré à la nouvelle architecture du conteneur. Pour chaque composant, le compilateur génère trois types de contrôleurs : l'*AdaptationDetector*, le *ProactivAdaptor* et le *ReactivAdaptor*. Ces contrôleurs implémentent les interfaces OMG IDL décritent dans la section 4.3.2. Par convention, les noms des classes générées sont sous la forme suivante : <nom_du_composant>_<Type_du_controleur>.

En mode reconfiguration dynamique, le code source d'adaptation utilise le mécanisme de réification qui permet de consulter le modèle de l'application pour connaître le type d'adaptation à effectuer et l'opération à invoquer. Les opérations fournies par un paquetage externe sont invoquées à l'aide de la réflexivité, alors que les opérations fournies par un autre composant sont invoquées à l'aide du mécanisme de construction dynamique de requête (DII) [13].

4.6 Discussion

CAMidO est un intergiciel orienté composant sensible au contexte, il fournit un méta-modèle d'ontologie pour décrire le contexte et des mécanismes pour gérer le contexte et l'adaptation de l'application.

L'architecture de CAMidO est divisée en deux structures : le gestionnaire de contexte et le gestionnaire d'adaptation. Le gestionnaire de contexte se charge d'interagir avec les capteurs pour collecter les informations de contexte, d'interpréter ces données et de les analyser pour détecter les situations pertinentes du contexte. Le gestionnaire d'adaptation se charge d'adapter l'application à chaque fois qu'une situation pertinente a été détectée. Ce gestionnaire implante la gestion de l'adaptation au niveau du conteneur du composant. Il utilise à cet effet le paradigme composant/conteneur. Ainsi, dans CAMidO, l'adaptation est gérée par le conteneur en ajoutant des contrôleurs à son architecture. Le code source associé à ces contrôleurs est généré automatiquement par le compilateur CAMidO, à partir des descriptions du concepteur de l'application.

L'intergiciel offre deux modes d'exécution, le mode configuration statique et le mode reconfiguration dynamique. Le mode configuration statique se base essentiellement sur le code source et les règles d'interprétation et de détection de situations pertinentes du contexte générées par le compilateur CAMidO. Alors que le mode reconfiguration dynamique génère les règles d'interprétation du contexte et de détection des situations pertinentes à chaque fois d'une interprétation ou une analyse des informations de contexte est sollicitée par l'intergiciel. De ce fait, le mode reconfiguration dynamique peut prendre en compte les changements que peut subir le méta-modèle sans arrêter l'exécution de l'application, il sera par contre plus lent à l'exécution.

Comparé aux intergiciels sensibles au contexte décrits dans la section 2, l'intergiciel CAMidO facilite la tâche de création d'applications sensibles au contexte. Le méta-modèle basé sur des ontologies est le modèle de référence pour décrire les informations de sensibilité au contexte associées à une application. Grâce au compilateur CAMidO, ces informations seront utilisées par les collecteurs, les interpréteurs, et les analyseurs de contexte ainsi que par les entités d'adaptation. En résumé, nous proposons au développeur de décrire plutôt que de programmer.

Cependant, des ambiguïtés et des contradictions peuvent être introduites par le modèle de l'application au niveau de la description des règles d'adaptation, ou par le compilateur CAMidO en déduisant les nouvelles règles. Plusieurs types de conflits peuvent apparaître. Nous avons identifié deux types de conflits : les conflits intra-composants et les conflits inter-composants.

Les conflits intra-composants peuvent apparaître lors du processus d'adaptation côté client lorsque la même opération peut être ré-aiguillée vers deux opérations différentes, alors que les conflits inter-composants peuvent apparaître lors du processus d'adaptation côté serveur quand le serveur peut fournir deux comportements différents de l'opération invoquée. Certains de ces conflits peuvent être détectés durant la compilation de la description fournie par le concepteur, mais cela n'élimine pas la possibilité de leur apparition en mode reconfiguration dynamique. Par conséquent, un mécanisme de résolution de conflits inter et intra-composant devra être incorporé à CAMidO. Ce mécanisme peut se résumer à l'ajout de priorité à toutes les opérations décrites dans l'application. Lorsqu'un conflit apparaît, l'opération ayant la plus haute priorité est invoquée à ce moment là.

D'autres types de conflits, plus complexes, peuvent apparaître durant le processus d'adaptation réactive. Pour expliquer ce type de conflit, prenons l'exemple de deux composants installés sur la même machine et appartenant à deux applications différentes. Le premier composant est sensible à la puissance de la batterie et éteint la machine si elle est à moins de 20%, alors que le second composant doit transférer des données vers une machine distante quand la puissance

de la batterie est à moins de 20%. Dans ce scénario, quand la situation pertinente (batterie moins de 20%) est détectée, le premier composant éteint la machine, de ce fait, le second ne peut pas fournir le service approprié (transférer les données vers une autre machine). Ce conflit est appelé un conflit intra-application, sa résolution nécessite l'ajout d'un mécanisme de consensus à CAMidO. Ce mécanisme doit être lancé à chaque installation d'une application.

Un changement fréquent des valeurs du contexte peut générer un effet ping-pong. En effet, si un composant est sensible à un contexte dont la valeur change très fréquemment en un laps de temps très court, l'intergiciel passera son temps à modifier le contenu de la structure *DetectedRs* ou à invoquer des services d'adaptation (la réaction dépend du type d'adaptation associé à ce contexte). Cet effet doit être résolu en ajoutant un mécanisme qui détecte les changements très fréquents du contexte dans l'objectif de ne prendre en compte que les changements significatifs du contexte.

4.7 Conclusion

Dans ce chapitre, nous avons présenté l'architecture de l'intergiciel CAMidO dont le but consiste à faciliter le développement des applications sensibles au contexte. Pour atteindre cet objectif, nous avons combiné un ensemble de techniques de séparation des préoccupations, entre autres la réflexivité, le paradigme composant/conteneur et les mécanismes d'interception. La réflexivité a été utilisée pour accéder au méta-modèle et prendre en compte les mises à jour de l'utilisateur quand CAMidO est exécuté en mode reconfiguration dynamique ainsi que pour effectuer les adaptations réactives de l'application. Le paradigme composant/conteneur a été utilisé pour gérer l'adaptation de l'application par des propriétés extra-fonctionnelles ce qui permet aux développeurs d'effectuer des descriptions d'adaptations au lieu de les programmer. Le mécanisme d'interception quant à lui a été utilisé pour intégrer l'adaptation d'un appel de service lors d'une adaptation proactive.

L'intergiciel CAMidO utilise un compilateur qui se charge de générer les informations nécessaires à la gestion du contexte et l'adaptation de l'application en se basant sur le modèle fourni. De ce fait, la création d'une application sensible au contexte consiste principalement à décrire le niveau application du méta-modèle. Les autres niveaux peuvent être complétés si nécessaire.

Dans le chapitre suivant, nous décrivons l'implémentation de cette solution au dessus de la plate-forme OpenCCM et les outils que nous avons utilisés.

Chapitre 5

Implantation de CAMidO et étude de performances

5.1 Introduction

Nous avons présenté dans les deux chapitres précédents l'ensemble de nos propositions pour la gestion et la création des applications orientées composant sensibles au contexte c'est à dire le méta-modèle et l'intergiciel CAMidO. Dans ce chapitre, nous présentons l'implantation et l'évaluation du prototype de CAMidO. Dans la section 5.2, nous décrivons l'implémentation du prototype de CAMidO que nous avons effectué au dessus de la plate-forme OpenCCM [80]. Ensuite, la section 5.3 décrit les évaluations qualitatives et quantitatives de ce prototype. Enfin, nous synthétisons dans la section 5.4 les différents tests d'évaluations que nous avons effectués.

5.2 Implantation et Intégration de CAMidO sur OpenCCM

Cette section présente l'implantation et l'intégration d'un prototype de CAMidO à la plate-forme OpenCCM (une plate-forme logicielle ouverte implantant la spécification CCM [79]). Ce prototype comprend un méta-modèle de description du contexte, une chaîne de production sous la forme d'un compilateur, un gestionnaire de contexte et un gestionnaire d'adaptation. Le méta-modèle de CAMidO décrit en Annexe A a été réalisé avec le langage OWL. Le compilateur CAMidO, le gestionnaire de contexte et le gestionnaire d'adaptation ont été développés à l'aide de Jena [57] et d'un outil de vélocité [1]. La version de CAMidO que nous avons implanté associe un gestionnaire de contexte à chaque application. Le tableau 5.1, présente les caractéristiques de la version actuelle de CAMidO (version 0.1.1). Dans ce qui suit, nous décrivons plus en détail l'implantation et l'intégration de CAMidO à la plate-forme OpenCCM.

Source	– 324 Ko de sources dont 45 classes Java, 2 déclarations en OMG IDL, 4 patrons (templates) – Fichier de confi guration (*confi g.properties*) – Script Jakarta Ant pour la compilation de la distribution – Scripts en ligne de commande
Exécutable	– 68.5 Ko pour l'archive (*.jar*) – 7.33 Mo nécessaire pour les archives externes utilisées (Jena, L'outil de vélocité)
Testé avec l'ORB	ORBacus 4.1.0
Testé sur les systèmes d'exploitation	– Windows 2000 – Windows XP
Requiert les logiciels	– OpenCCM (version 0.8.2) – ORBacus (version 4.1.0) – Jakarta Ant (version >= 1.6) – Jena (version 2.2) – Velocity tools (version 1.4)

TAB. 5.1 – Caractéristiques du prototype de base

5.2.1 Compilateur CAMidO

Le compilateur CAMidO utilise la description du contexte fournie par le méta-modèle de CAMidO, le modèle de l'application et des fichiers de *templates* pour générer des fichiers de règles de logique de premier ordre et le code source d'adaptation et d'interprétation. Les règles générées permettent à CAMidO de raisonner sur les informations de contexte et le code généré permet d'automatiser le processus de gestion du contexte et d'adaptation de l'application.

Comme nous l'avons décrit dans la section 4.5, le compilateur CAMidO utilise un moteur de vélocité et un parseur d'ontologie pour parcourir l'ontologie et générer les règles et le code source d'interprétation et d'adaptation. Pour implémenter le moteur de vélocité nous avons utilisé le velocity tool version 1.4 [1] et un ensemble de templates (que nous avons écrit) contenant des modèles de chaque type de fichier que le compilateur CAMidO doit générer. Nous avons développé le parseur d'ontologie à l'aide du *framework* Jena version 2.2. Ce parseur permet de parcourir l'ontologie et d'extraire les informations nécessaires à la génération des fichiers de règles décrits dans la section 4.5.1.

L'implantation du compilateur CAMidO s'est faite indépendamment de la plate-forme OpenCCM, mais son intégration à cette plate-forme a nécessité son ajout à la chaîne de production d'OpenCCM. Pour cela, nous avons écrit un script de lancement du compilateur CAMidO appelé *LaunchCAMidOCompiler* que nous avons invoqué dans le script associé à la chaîne de production d'OpenCCM. Le script *LaunchCAMidOCompiler* se charge de parser l'ontologie associée à l'application et de générer les fichiers de règles et le code source d'interprétation et

d'adaptation. La figure 5.1 présente l'ensemble du cycle de développement et de déploiement des applications à base de composants sensibles au contexte. Elle montre le découpage du processus de production en différentes étapes, où chaque étape utilise un ensemble d'outils fournis par OpenCCM et CAMidO qui travaillent en collaboration pour générer l'application à exécuter.

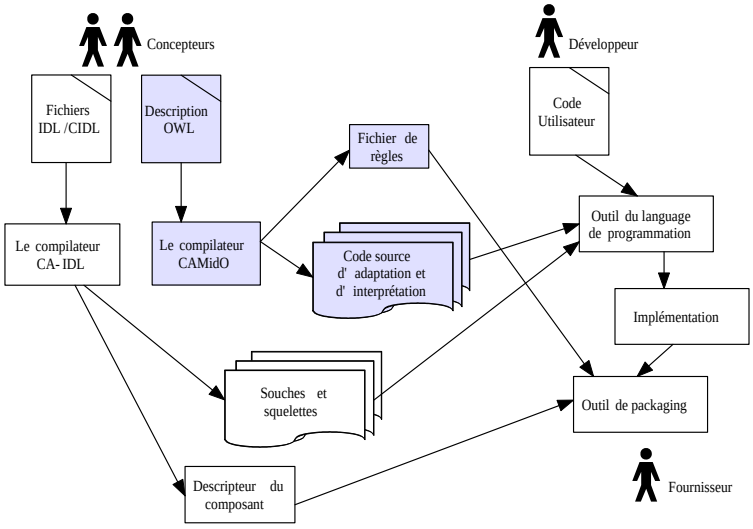

FIG. 5.1 – Intégration du compilateur CAMidO dans la chaîne de production d'OpenCCM

Plusieurs concepteurs peuvent définir les types des composants à l'aide du langage IDL3 et les informations de sensibilité au contexte à l'aide du langage OWL. Les définitions IDL sont projetées en souches et squelettes et en IDL CORBA2 pour permettre l'implantation des composants par le développeur. Les descriptions OWL sont compilées par le compilateur CAMidO pour générer des fichiers de règles et le code java nécessaire à la gestion de la sensibilité au contexte. La phase de projection produit aussi un descripteur de composant qui doit être complété par le développeur. Ce dernier doit implanté les méthodes d'interprétation et de détection des situations pertinentes si le modèle de l'application utilise des règles qui font appel à ces méthodes (cf. section 4.5). Les données générées par ces étapes sont packagées au sein d'une archive pour être utilisées lors de la création du descripteur d'assemblage. Les paquetages de composant et d'assemblages sont déployés pour instancier les applications sur leurs sites d'exécution.

5.2.2 Gestionnaire de contexte

Le gestionnaire de contexte est constitué d'un ensemble d'entités qui se chargent de collecter et de gérer le contexte (cf. section 4.3). Il est constitué de l'interface IDL *ContextManager* qui permet aux composants de s'enregistrer pour les contextes auxquels ils sont sensibles, et d'un ensemble de classes qui se chargent de collecter et de gérer le contexte (interprétation et analyse). Ces classes constituées du *CollectionManager*, du *ContextInterpreter*, du *ContextAnalyser* et de l'*InferenceComponent* ont été développées en langage java et font appel aux classes du *framework* Jena. Ce *framework* nous a permis dans le cadre de l'implantation du gestionnaire

de contexte de parcourir l'ontologie et d'exécuter le moteur d'inférence intégré à l'entité *InferenceComponent*. Ce moteur permet de raisonner sur les informations de contexte collectées afin d'interpréter des informations de haut niveau et de détecter ses variations pertinentes.

Le gestionnaire de contexte est indépendant de la plate-forme OpenCCM, son exécution est effectuée en exécutant le script *LaunchCAMidO*. Le gestionnaire de contexte est considéré par cette plate-forme comme un service auquel chaque application sensible au contexte peut faire appel à l'aide des contrôleurs générés par le compilateur CAMidO. Ce code permet à chaque composant de s'enregistrer auprès du gestionnaire de contexte pour toutes les situations pertinentes auxquelles il est sensible.

5.2.3 Gestionnaire d'adaptation

Comme nous l'avons décrit dans la section 6, le gestionnaire d'adaptation est constitué des contrôleurs générés par le compilateur CAMidO, des intercepteurs portables que nous associons à l'ORB et d'un proxy. Ces entités travaillent en collaboration pour gérer l'adaptation de l'application.

Les contrôleurs ont été implantés selon le principe de conception décrit dans la section 4.5.2, la section 4.3.2, et la section 6. Le code de ces contrôleurs (le *ReactivAdaptor*, le *ProactivAdaptor* et l'*AdaptationDetector*) est généré automatiquement par le compilateur CAMidO lors de la compilation de l'application.

Dans OpenCCM, l'utilisation de toute application nécessite le déploiement des composants qui la constitue. Ce déploiement consiste à installer les implantations des composants, les instancier puis configurer les composants et les connexions. La fin du déploiement de chaque instance se traduit par l'invocation de l'opération *configuration_complete* pour notifier aux instances de composants que leurs connexions initiales ont été effectuées. Lors de l'intégration de CAMidO dans OpenCCM, l'enregistrement de chaque composant auprès du gestionnaire de contexte est effectué lors de l'invocation de *configuration_complete*. Pour réaliser cela, nous avons modifié le patron *java_skeleton_component.template* utilisé par OpenCCM pour générer les codes sources associés au squelette de chaque composant. Cette modification consiste à solliciter le contrôleur *AdaptationDetector* pour enregistrer le composant auprès du gestionnaire de contexte en invoquant l'opération *Subscribe* fournie par l'interface *ContextManager* (cf. section 4.3).

Les intercepteurs portables ont été implantés selon le principe de conception décrit dans la section 6, leur activation a nécessité la création d'un script que nous avons intégré aux scripts de lancement des applications OpenCCM. L'utilisation du *CAMidOProxy* pour récupérer les paramètres des requêtes et leurs redirections est due aux limites de l'implantation java des intercepteurs portables. En effet, l'implantation java ne donne pas le moyen aux intercepteurs d'accéder à certaines informations associées aux opérations invoquées tel que les arguments de l'opération par exemple [87].

Le *CAMidOProxy* est un proxy dynamique qui utilise les mécanismes DII (*Dynamic Invocation Interface*) et DSI (*Dynamic Skeleton Interface*) [13] pour respectivement invoquer dynamiquement

des requêtes et lire les requêtes redirigées par le *CAClientInterceptor*. Comme nous l'avons spécifié dans la section 6, le traitement des requêtes redirigées par le *CAClientInterceptor* nécessite de connaître le composant qui a émis cette requête. Cette information n'est pas disponible au niveau *ClientRequestInfo* [77]. Pour remédier à ce problème, nous proposons deux solutions : la première solution consiste à utiliser l'attribut *contexts* de la structure *RequestInfo* [77] pour sauvegarder la référence du composant qui a émis la requête. Cette affectation peut être effectuée lors de la création de la requête soit en modifiant le code source d'OpenCCM ou le code source de l'ORB utilisé, grâce à cela le composant qui a émis la requête peut être identifié en consultant l'attribut *contexts*. La seconde solution consiste à imposer aux développeurs d'applications la contrainte d'enregistrer tous les composants dans le service de nommage et d'ajouter comme paramètre le nom du composant dans toutes les signatures des opérations implantées. Lors de l'interception d'une requête, le composant qui l'a émise peut être identifié en consultant les arguments de l'opération invoquée.

Dans l'implantation actuelle de *CAMidOProxy*, nous avons considéré la seconde solution, car bien qu'elle impose une forte contrainte aux développeurs d'applications, elle est plus rapide à implanter. Au niveau du *CAServerInterceptor*, nous n'avons imposé aucune contrainte, car la référence du composant cible est disponible à partir de la structure *ServerRequestInfo* [77].

5.2.4 Synthèse

L'implantation de CAMidO à été effectuée en utilisant un ensemble d'archives externes pour parcourir l'ontologie, raisonner sur les informations de contexte et générer du code source java. Nous avons imposé une contrainte aux développeurs d'applications, qui consiste à faire passer en paramètre le nom du composant qui invoque une opération dans le but d'identifier le composant appelant lors de l'interception des requêtes.

L'intégration de CAMidO à la plate-forme OpenCCM à nécessité l'écriture d'un ensemble de scripts qui permettent de lancer le compilateur CAMidO lors de la phase de compilation de l'application, de lancer le gestionnaire de contexte et d'intégrer les intercepteurs portables à l'ORB lors de la phase d'exécution. Nous avons modifié une template appartenant à la plate-forme OpenCCM pour effectuer l'enregistrement des composants lorsque l'opération *configuration_complete* est invoquée.

5.3 Évaluation qualitative et étude des performances

La gestion de la sensibilité au contexte présente une réelle valeur ajoutée aux applications distribuées. Elle surveille l'environnement et prend en compte ses changements pertinents pour adapter le comportement des applications d'une manière automatique.

Sans l'utilisation d'un intergiciel sensible au contexte, la création et la gestion d'une application sensible au contexte est une tâche difficile à mettre en oeuvre, car le développeur doit programmer toutes les étapes de gestion du contexte et d'adaptation de l'application. L'objectif de

CAMidO consiste à faciliter cette tâche aux développeurs. Ceci est réalisé en leur offrant d'une part un méta-modèle d'ontologie pour décrire les informations relatives à la sensibilité au contexte et d'autre part des entités qui se chargent d'automatiser le processus de gestion du contexte et d'adaptation de l'application. De plus l'intergiciel CAMidO sépare la gestion du contexte de la gestion de l'adaptation. Par conséquent, CAMidO peut être utilisé soit pour gérer le contexte seulement soit pour gérer tous les éléments fonctionnels d'une application orienté composant sensible au contexte allant de l'observation du contexte à l'adaptation de l'application.

Pour valider nos travaux, nous avons effectué des évaluations qualitatives de CAMidO et des tests de performances sur la gestion de la sensibilité au contexte et l'adaptation des applications. L'évaluation qualitative a pour objectif de montrer l'utilité des abstractions et du principe de séparation des préoccupations fournies par CAMidO pour faciliter le développement des applications sensibles au contexte ainsi que la possibilité de son utilisation par des services sensibles au contexte. Cette évaluation nous permet tout d'abord d'étudier l'utilisabilité de CAMidO, d'examiner les extensions qu'il faut lui apporter pour qu'il puisse être exploité par des services sensibles au contexte et d'analyser les degrés de difficultés pour réaliser ces extensions. Les tests de performances ont pour objectif d'évaluer le coût du moteur d'inférence, d'étudier la réactivité du système pour déterminer la fréquence de changement des informations de contexte que le système CAMidO peut traiter pour que les adaptations effectuées soient pertinentes, d'évaluer le coût du mode reconfiguration dynamique pour déterminer le type d'application qui peuvent utiliser ce mode et enfin, d'évaluer le coût de traitement engendré par les adaptations réactives et proactives.

5.3.1 Évaluation qualitative

L'évaluation qualitative de CAMidO est réalisée en deux étapes. La première étape consiste à présenter pas à pas une utilisation concrète de CAMidO pour le développement d'une application sensible au contexte dans le but d'évaluer la facilité apportée par CAMidO pour la création de ce type d'applications. Dans la deuxième étape, nous intégrons CAMidO au-dessous d'un service sensible au contexte appelé CADeComp [1] (Context-Aware Deployment of Component) [6] pour étudier l'utilisation de CAMidO entant que gestionnaire de contexte, d'examiner les extensions nécessaires à CAMidO pour que CADeComp puisse utiliser toutes les fonctionnalités offertes par notre plate-forme, et d'analyser le degré de difficulté de la réalisation de cette extension.

Développement d'applications sensibles au contexte avec CAMidO

Dans cette section, nous décrivons pas à pas l'implantation et l'exécution de l'application *SensitivePurchase* décrite dans la section 3.4.2. Cette application permet aux utilisateurs de diminuer le temps passé dans les magasins en commandant leurs achats avant d'arriver au point de vente.

L'application SensitivePurchase est structurée en quatre composants : le composant Interface Utilisateur (UI), le composant Base de Données (DB), le composant Vue Local (LV), et le com-

[1] CADeComp est un service qui permet le déploiement sensible au contexte des applications orientées composants

```
module SensitivePurchase{
        interface CatalogueVisualisation{
          void  GetAllCatalogue(in string catalogueName);
          void  GetTextCatalogue(in string catalogueName);
        };

        interface RequestManagement{
          void  TransferRequest(in string requestInfo);
            };
        interface ProductManagement{
          void AddProduct(in string productName);
          void RemoveProduct(in string productName);
            };
        interface CatalogueCacheManagement{
          void SaveCatalogue(in string catalogueName);
        };
        component UI {
                uses CatalogueVisualisation catalogueVisualisation;
                uses ProductManagement productManagement;
                provides CatalogueCacheManagement catalogueCacheManagement;
        };
        component DB {
                provides CatalogueVisualisation catalogueVisualisation;
                provides RequestManagement requestManagement;
        };
        component LV {
                provides CatalogueVisualisation catalogueVisualisation;
                 uses RequestManagement requestManagement;
                uses CatalogueCacheManagement catalogueCacheManagement;
        };
        component CART {
                provides ProductManagement productManagement;
        };
...
}
```

FIG. 5.2 – Description le l'interface IDL de l'application *SensitivePurchase*

posant Panier (Cart). Les contextes pertinents pour cette application sont l'état de la connexion réseau et la variation de la bande passante.

L'objectif de l'utilisation de CAMidO est de séparer le code spécifique à la gestion de la sensibilité au contexte du code métier de l'application. L'implantation de cette application se fait en plusieurs étapes. Tout d'abord, il faut décrire l'interface IDL de chaque composant qui constitue l'application. Ensuite, les informations de sensibilité au contexte associées à l'application doivent être décrites en se basant sur le méta-modèle de CAMidO. Puis, il faut implanter les interfaces IDL de chaque composant. Enfin, il faut implanter les classes décrites dans l'ontologie et utilisées lors de la gestion de la sensibilité au contexte telles que les méthodes de détection des situations pertinentes (cf. section 3.5.3).

La figure 5.2, illustre la description de l'interface IDL des composants qui constituent l'application. Cette interface contient tous les comportements de l'application y compris ceux de l'adaptation, elles doivent être implémentées par le développeur de l'application.

L'ajout de la sensibilité au contexte à cette application consiste à décrire le modèle de l'application en se basant sur l'ontologie associée au niveau application du méta-modèle de CAMidO. Cette description peut être faite en utilisant l'outil graphique Protégé [2] en y exportant le méta-modèle de CAMidO. Cette description se fait en plusieurs étapes. Tout d'abord le concepteur doit

vérifier la présence de la description du contexte de bas niveau *Bandwidth* et du capteur qui permet de le collecter dans l'ontologie, si ces données n'existent pas, il doit les créer et les intégrer à l'ontologie comme l'illustre la figure 5.3.

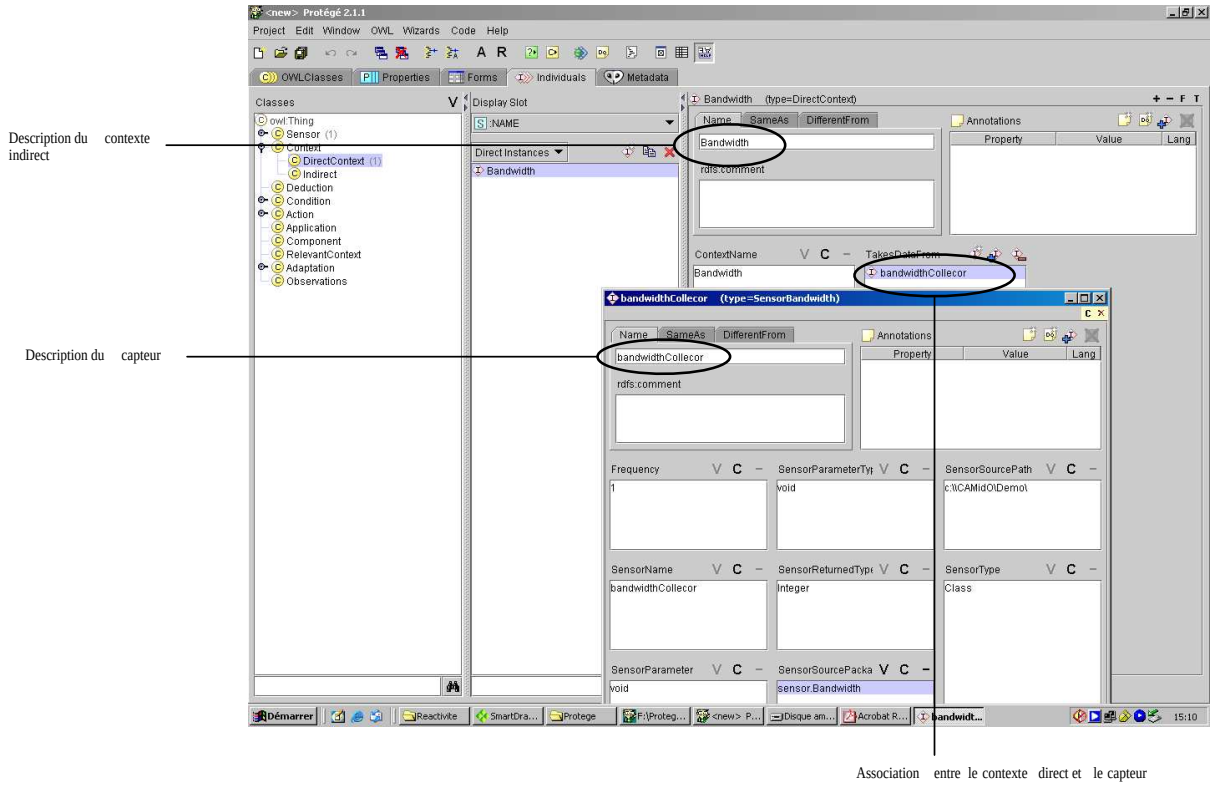

FIG. 5.3 – Description du contexte direct *Bandwidth* et du capteur *BandwidthSensor* avec l'outil Protege

Ensuite, Il doit décrire les contextes de haut niveau et les règles de leur interprétation comme l'illustre la figure 5.4. Cette figure montre la description du contexte *ConnexionState* et de la règle d'interprétation qui lui est associée. Après cela, le concepteur doit associer les composants qui constituent l'application *SensitivePurchase* avec les situations pertinentes auxquelles ils sont sensibles comme l'illustre la figure 5.5.

Après avoir défini les composants qui constituent l'application et les situations pertinentes auxquelles ils sont sensibles, le concepteur doit associer à chacune de ces associations (composant, situation pertinente) les actions d'adaptation appropriées comme l'illustre la figure 5.6 qui montre la description de l'adaptation réactive associée au composant LV lors de la détection de la situation pertinente *RelevantConnection*.

Dans le cadre de l'application *SensitivePurchase* les règles d'interprétation d'un contexte de haut niveau et de détection des situations pertinentes utilisent des conditions sur les valeurs du contexte, elle ne font pas appel à des méthodes qui effectuent ces calculs. Si une application fait appel à des méthodes pour effectuer ces calculs, ces méthodes doivent être implémentées par le développeur de l'application.

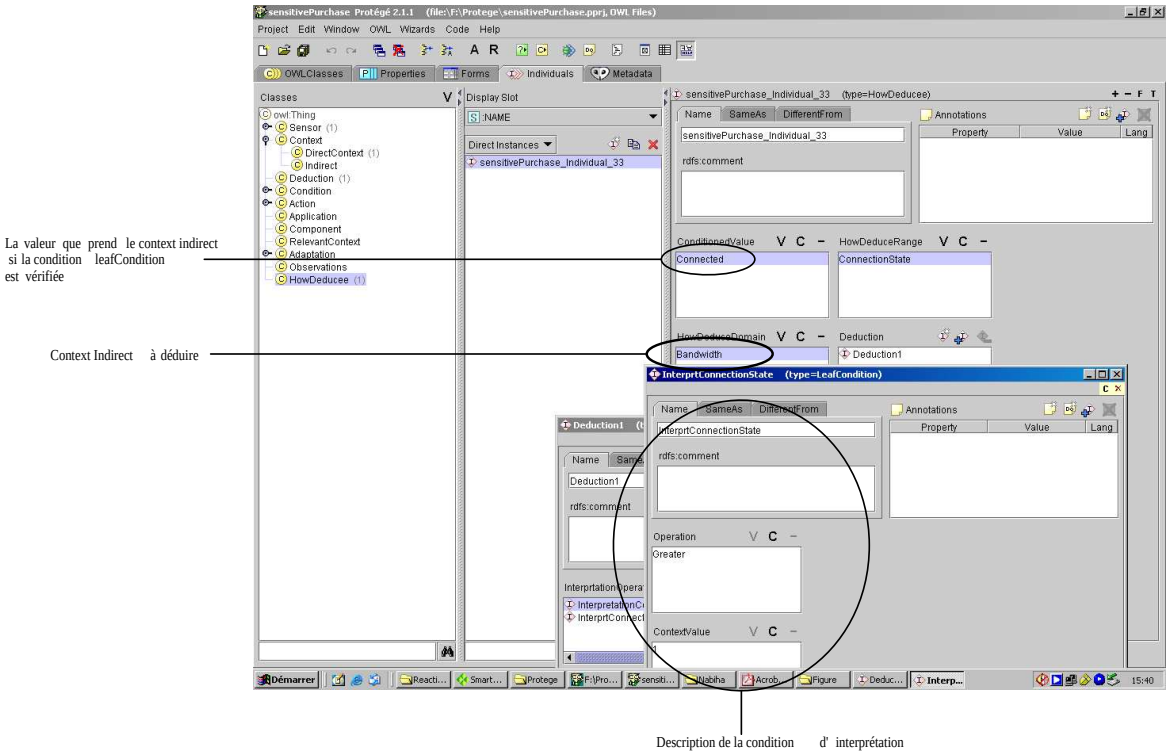

FIG. 5.4 – Description du context indirect *ConnexionState* et de la règle associée

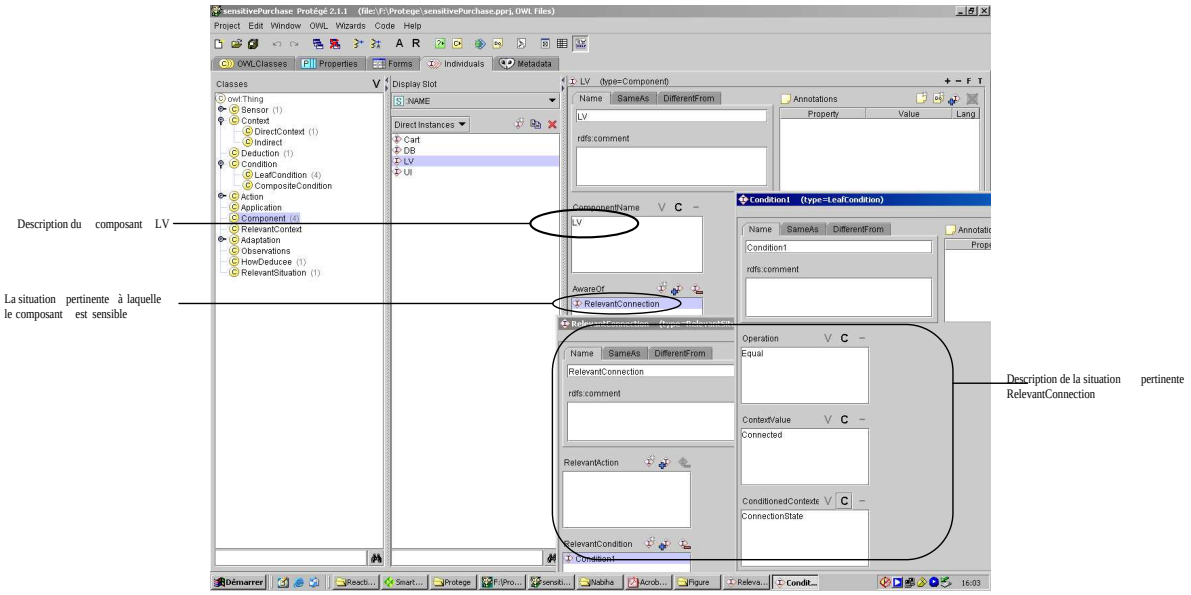

FIG. 5.5 – Description du composant LV

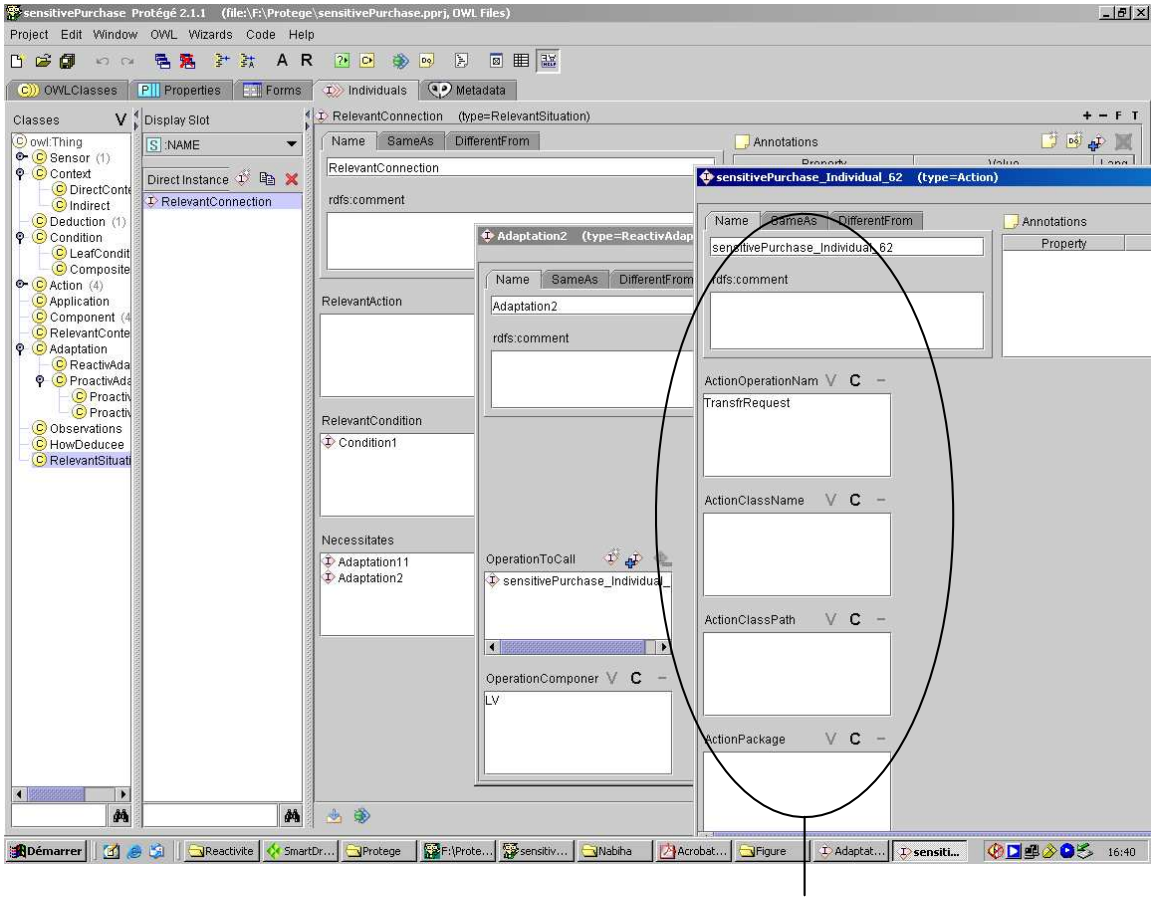

FIG. 5.6 – Description de la règle d'adaptation associée à la situation pertinente *RelevantConnexion*

Après la description de l'assemblage de l'application et son empaquetage, elle est compilée et exécutée en lançant les scripts appropriés. La compilation de l'application est effectuée à l'aide de l'instruction *build* qui exécute le script de compilation d'openCCM et celui de CAMidO. Lors de l'exécution, le lancement de l'application *SensitivePurchase* doit être précédé du lancement du gestionnaire de contexte avec la commande *LaunchCAMidO*. Ensuite, l'application doit être lancée à travers l'instruction *bin start_java –xml*.

Pendant l'exécution de l'application, le gestionnaire de contexte utilise les règles de raisonnement sur le contexte (règles d'interprétation et de détection des situations pertinentes) générées par le compilateur CAMidO pour gérer le contexte. Le gestionnaire d'adaptation utilise les contrôleurs générées par le compilateur CAMidO afin d'adapter chaque composant aux situations pertinentes auxquelles il est sensible.

Le développement d'une application sensible au contexte à l'aide de CAMidO consiste principalement à décrire le modèle de sensibilité au contexte associé à l'application. Ce modèle est décrit en se basant sur le méta-modèle de CAMidO. La description du modèle de l'application sans utiliser un éditeur graphique (protégé par exemple) est fastidieuse. Car le concepteur d'application doit connaître toute la syntaxe du méta-modèle de CAMidO.

Par rapport aux travaux décrits dans le chapitre 2, l'apport de l'intergiciel CAMidO réside dans le fait qu'il gère tous les éléments fonctionnels d'une application sensible au contexte.

Utilisation de CAMidO par le service CADeComp

Dans cette section, nous évaluons l'utilisation de CAMidO par un service sensible au contexte et nous analysons le volume de travail que peut engendrer son extension. Avant de commencer la description de cette évaluation, il est important de souligner qu'un couplage entre CAMidO et CADeComp a été effectué au sein de notre équipe par un stagiaire de master de recherche. Dans cette section, nous faisons référence aux travaux de couplage et aux évaluations de CAMidO faites par ce stagiaire [115].

Comme nous l'avons spécifié dans le chapitre 4, CAMidO offre des mécanismes de gestion du contexte et des mécanismes d'adaptation pour faciliter la création des applications sensibles au contexte. Cet intergiciel est constitué de deux parties distinctes, le gestionnaire de contexte et le gestionnaire d'adaptation. Cette séparation permet à des services sensibles au contexte d'utiliser CAMidO comme un simple gestionnaire de contexte, l'adaptation étant gérée par le service lui-même. De ce fait, CAMidO se charge simplement de collecter les informations de contexte, de les interpréter et de les analyser en se basant sur la description du contexte fournie par le service.

CADeComp est un service qui établit l'adaptation au contexte du déploiement des applications à base de composants [6]. Cette adaptation structurelle consiste à choisir les implantations des composants qu'il faut déployer, à choisir les nœuds de déploiement[2] de chaque instance, à effectuer la configuration des propriétés de chaque instance et à définir l'assemblage final de l'application en fonction du contexte de l'environnement. Deux niveaux d'informations de contexte

[2] un noeud de déploiement est représenté par un ordinateur. Les noeuds sont regroupés dans un domaine de déploiement fournit par un prestataire de service

peuvent affecter le déploiement : le niveau plate-forme qui regroupe les informations matérielles et logicielles relatives aux nœuds qui vont accueillir l'application à déployer et le niveau utilisateur qui regroupe des informations de contexte sur l'utilisateur de l'application comme ses préférences, sa localisation et son profil. Lors du déploiement initial de l'application, CADeComp a besoin de toutes les informations de contexte qui affectent le déploiement, pour cela, il utilise les services de CAMidO entant que collecteur de contexte. Après l'installation de l'application, CADeComp s'enregistre auprès de CAMidO pour toutes les situations pertinentes qui nécessitent une reconfiguration structurelle de l'application. Lors de la réception d'une notification, CADeComp se charge de réinstaller l'application en fonction des nouvelles valeurs du contexte.

L'utilisation des services de CAMidO par CADeComp nécessite la description des contextes pertinents pour CADeComp. Cette description est effectuée en créant des instances et des classes spécifiques à CADeComp dans le méta-modèle de CAMidO. La figure 5.7, illustre le méta-modèle de CAMidO contenant des classes spécifiques à CADeComp. L'ajout de ces classes demande une très bonne connaissance du méta-modèle de CAMidO, cela consiste à ajouter des triplets dans les ontologies appropriées.

Le niveau *Déploiement* intégré au méta-modèle de CAMidO permet de définir un ensemble de nœuds de domaine sur lequel une application va être déployée. Cela est exprimé par la relation *DeployedOn* entre l'entité *Application* et l'entité *ProviderDomain*. Une application, bien entendu, est déployée à partir du terminal de l'utilisateur cela est décrit à l'aide de la relation *DeployedFrom* entre l'entité *Application* et l'entité *Terminal*. Ce terminal peut être vu comme un nœud particulier du domaine sur lequel quelques composants peuvent être déployées. Les nœuds du domaines et les liaisons entre ces nœuds offrent un ensemble de resources, cela est exprimé par les relations *LinksProvides* et *NodeProvides*. Un nœud peut être sujet à des observations du contexte, la relation *NodeContext* permet de déterminer les observables associés à chaque noeud.

Le service CADeComp à besoin de deux modes de communication pour s'exécuter : le mode requête lors du déploiement initial et le mode notification lors de l'exécution de l'application. A travers le mode requête, le service CADeComp récupère les observations du contexte effectuées par CAMidO, ainsi qu'une structure de données contenant entre autre les références des situations pertinentes détectées. Or, la version de base de CAMidO ne fonctionne qu'en mode notification, par conséquent, une extension de CAMidO s'avère être nécessaire. Cette extension consiste à ajouter à CAMidO le composant *CommunicationModeManager* qui lui permet de fonctionner en mode requête et de fournir au service toutes les valeurs du contexte dont il a besoin. La figure 5.8 illustre la description de l'interface IDL du *ContextControlBlock*.

Le *CommunicationModeManager* se charge d'interagir avec le *ContextRepository* pour récupérer des observations du contexte (toutes les observations du contexte, les observations du contexte propres au domaine, les observations du contexte associées à un nœud) à l'aide des opérations décrites dans les lignes 9 à 11 de la figure 5.8, de récupérer les situations pertinentes détectées (toutes les situations pertinentes, les situations pertinentes pour un domaine, les situations pertinentes pour un nœud) à l'aide des opérations décrites dans les lignes 12 à 14 de la figure 5.8. Surveiller le domaine de déploiement pour détecter des situations pertinentes nécessite l'interaction du *CommunicationModeManager* avec le moteur d'inférence *InferenceComponent*. Cette interaction nécessite de fournir au composant *InferenceComponent* des règles qui

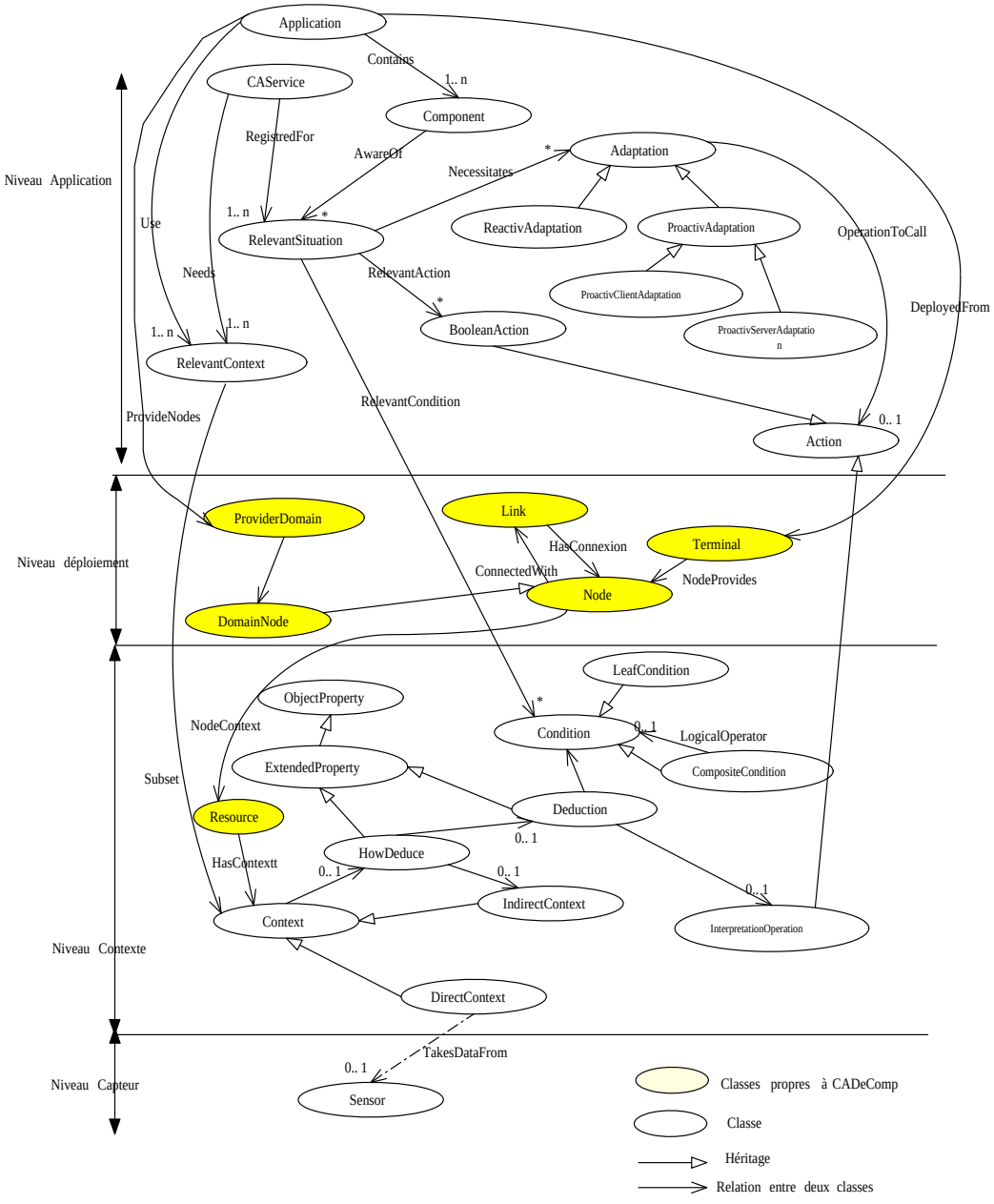

FIG. 5.7 – Méta-modèle de CAMidO étendu pour le couplage anec CADeComp

```
module CAMidOExtension{
     struct ContextValue{string id; string name; string idnode; string val; string type};
     struct RelevantSituation{string id; string idnode};
     typedef sequence<Contextvalue> ContextValSeq;
     typedef sequence<RelevantSituation> RelSituationseq;
     typedef sequence<string> Seq;

  interface CommunicationModeManager{
    . ContextValSeq getContextValues(in String appname);
      ContextValSeq getDomaineContextValues(in string appname);
      ContextValSeq getNodeContextValues(in string node,in string appname);
      RelContextSeq getRelevantSituation(in string appname);
      RelContextSeq getDomainRelevant(in string appname);
      RelContextSeq getNodeRelevant(in string node, in string appname);
      string subscribe(in Seq contexte);
    };
};
```

FIG. 5.8 – Interface IDL du composant *CommunicationModeManager*

lui permettent de détecter les situations pertinentes et de remplir la structure *RelevantSituation* avec les situations pertinentes détectées. Ces règles sont générées par le compilateur CAMidO en utilisant une extension implantée dans le cadre du couplage.

L'extension de CAMidO en lui ajoutant le *CommunicationModeManager* n'est pas très difficile à mettre en œuvre puisqu'il s'agit d'une nouvelle entitée qui doit être intégrée à son architecture. Cette entité est constituée d'une interface qui interagit avec le *ContextRepository* et qui lance le raisonnement sur les informations de contexte pour les utiliser en mode requête.

L'extension du compilateur CAMidO est une tâche fastidieuse qui nécessite une très bonne connaissance du méta-modèle de CAMidO, de la structure des règles utilisées et du code du compilateur CAMidO, néanmoins nous fournissons un ensemble de bibliothèques pour faciliter cette tâche.

La séparation entre le gestionnaire de contexte et le gestionnaire d'adaptation facilite l'utilisation de CAMidO par des services sensibles au contexte ayant leurs propres mécanismes d'adaptation. Quant à l'extension de CAMidO, selon les tests menés par le stagiaire, il est préférable de développer ces extensions et de les tester séparément avant de les intégrer à la plate-forme CAMidO, cela permet de ne pas modifier le code source de CAMidO et de développer ces modules plus rapidement. Le compilateur CAMidO est la partie la plus difficile à étendre malgré les bibliothèques fournies par la plate-forme.

Les extensions effectuées à l'intergiciel CAMidO lui permettent de communiquer en mode requête et en mode notification. L'utilisation de CAMidO par d'autres services sensibles au contexte consiste à décrire les modèles de sensibilité au contexte de ces services en se basant sur le méta-modèle de CAMidO. Ces descriptions peuvent se passer d'une extension du méta-modèle si ces services n'ont pas besoin de spécialiser des entités observables au niveau méta comme c'est la cas du service CADeComp. Le mode de communication entre chaque service et CAMidO peut se faire en mode requête ou en mode notification grâce à l'interface *CommunicationModeManager*.

5.3.2 Évaluation des performances

Les tests que nous décrivons dans cette section portent sur les performances de CAMidO. Le premier objectif de test consiste à évaluer le coût engendré par le moteur d'inférence et les paramètres qui font varier son temps de traitement. Le deuxième objectif consiste à étudier la réactivité du système afin de déterminer la fréquence de changement des informations de contexte que le système peut collecter et utiliser en toute efficacité. Le troisième objectif consiste à évaluer le mode reconfiguration dynamique offert par CAMidO dans le but de déterminer le coût de la reconfiguration en terme de temps d'exécution et le type d'applications qu'il est préférable d'utiliser dans ce mode. Enfin, le dernier objectif de nos tests porte sur l'étude du coût engendré par l'adaptation réactive et proactive.

L'environnement de test que nous avons utilisé est constitué d'un PC portable PentiumIII dont la fréquence du processeur est de 1,3 GHz avec une mémoire RAM de 256 Mo. Le système d'exploitation utilisé est Microsoft Windows 2000 avec la machine virtuelle J2SDK 1.4.2. Le bus logiciel utilisé est l'ORB ORBACUS 4.1.0 pour java au-dessous d'OpenCCM 0.8.2. Puisque l'objectif de nos tests n'est pas d'étudier la distribution des applications qui s'exécutent au-dessus de CAMidO, mais, plutôt d'évaluer le coût engendré par la gestion de la sensibilité au contexte, nous avons effectuer nos tests sur une seule machine. Pour étudier le coût de gestion de la sensibilité au contexte, nous avons choisi d'étudier le cas le plus défavorable, c'est à dire, le cas où tous les composants de l'application sont installés dans la même machine où s'exécute CAMidO.

Pour tous nos tests, nous avons utilisé une application constituée de trois composants. Les autres informations de sensibilité au contexte ainsi que l'ontologie associées à l'application ont changé au fil des expérimentations, nous donnons dans chaque section des informations sur leur taille[3] et le nombre de règles d'interprétation et d'adaptation qu'elle contient. Ce qui est important de souligner, c'est que les contextes observés changent à chaque collecte du contexte. Les règles d'interprétation et de détection des situations pertinentes utilisent des conditions pour déduire le contexte de haut niveau et pour détecter les situations pertinentes. Chaque observation de contexte engendre la détection de 50% des situations pertinentes décritent dans le modèle de l'application. Concernant l'interprétation du contexte de haut niveau, chaque observation de contexte engendre la déduction d'un ensemble de contextes de haut niveau, le nombre de contexte interprété est égale à la moitié des règles d'interprétation décrites dans le modèle de l'application.

Chaque test est exécuté 20 fois afin d'avoir des moyennes significatives. La moyenne des écarts types de tous les tests que nous avons effectués est égale à 0,70.

Évaluation du moteur d'inférence

Le moteur d'inférence est représenté par le composant *InferenceComponent* de l'architecture de CAMidO (cf. section 4.3). Il est utilisé par le composant *ContextInterpreter* pour interpréter

[3]la taille de l'ontologie que nous considérons englobe le nombre de composants décrits dans l'ontologie, le nombre de contextes pertinents pour l'application ainsi que la taille du méta-modèle de CAMidO qui est égale à 100 triplets. Le nombre de règles d'interprétation et de détection des situations pertinentes n'est pas comptabilisé

un contexte de haut niveau et par le composant *ContextAnalyser* pour détecter des situations pertinentes. Dans cette section, nous évaluons le coût de traitement engendré par le moteur d'inférence en étudiant le temps d'analyse et d'interprétation d'une observation de contexte. Afin de déterminer les paramètres qui peuvent influer ce temps, nous faisons varier pour chaque test la taille de l'ontologie en terme de triplet[4] et le nombre de règles d'interprétation et de détection des situations pertinentes.

Le temps de traitement d'une information de contexte est représenté par la formule suivante :
$Temps_{traitement}$=$Temps_{Analyse}$ +$Temps_{Deduction}$.
$Temps_{Analyse}$=$Temps_{Nv}$+$Temps_{Maj}$+$Temps_{Sp}$
$Temps_{Maj}$=$Temps_{MajOnto}$+ $Temps_{MajArchive}$
$Temps_{Deduction}$=$Temps_{Interpretation}$+$Temps_{Maj}$
$Temps_{Analyse}$: représente le temps d'analyse total d'une information de contexte.
$Temps_{Nv}$: représente le temps que le *ContextAnalyser* prend pour déterminer si l'information de contexte collectée a changée.
Maj : représente le temps de sauvegarde d'une information de contexte.
$Temps_{MajOnto}$: représente le temps de sauvegarde d'une information de contexte dans l'ontologie.
$MajArchive$: représente le temps de sauvegarde d'une information de contexte dans une archive.
$Temps_{Sp}$: représente le temps de détection des situations pertinentes.
$Temps_{Interpretation}$: représente le temps que prend le moteur d'inférence pour exécuter les règles d'interprétation.
$Temps_{Deduction}$: représente le temps total de l'interprétation, qui regroupe le temps de l'interprétation du contexte et le temps de la sauvegarde de la valeur interprétée.

Lors de nos tests, nous n'avons considéré que le temps d'exécution des tâches effectuées par le moteur d'inférence, à savoir l'interprétation représenté par $Temps_{Deduction}$ et la détection des situations pertinentes représenté par $Temps_{Sp}$.

Les figures 5.9 et 5.10 illustrent les résultats que nous avons obtenus en exécutant le moteur d'inférence pour le traitement d'une observation de contexte. Nous avons utilisé une ontologie contenant 800 triplets et nous avons fait varier le nombre de règles d'interprétation et de détection des situations pertinentes de 10, 50 jusqu'à 200 règles. Ces figures montrent que le moteur d'inférence est sensible au nombre de règles qu'il doit exécuter et à la taille de l'ontologie. En d'autres termes, le temps d'interprétation et de détection des situations pertinentes augmente en fonction du nombre de règles et en fonction de la taille de l'ontologie. En effet, pour un contexte observé, le moteur d'inférence exécute toutes les règles d'interprétation pour calculer des contextes de haut niveau, et toutes les règles de détection des situations pertinentes pour notifier les composants sensibles aux situations détectées. De plus, lors de l'exécution d'une règle, le moteur d'inférence parcourt toute l'ontologie pour évaluer les conditions des règles et raisonner sur les observations du contexte.

En étudiant les graphes de la figure 5.9 et ceux de la figure 5.10, nous constatons que pour une taille d'ontologie spécifique, le moteur d'inférence prend plus de temps pour exécuter les

[4]un triplet en RDF est représenté par <une classe, une propriété, un objet> l'objet peut être classe ou une valeur.

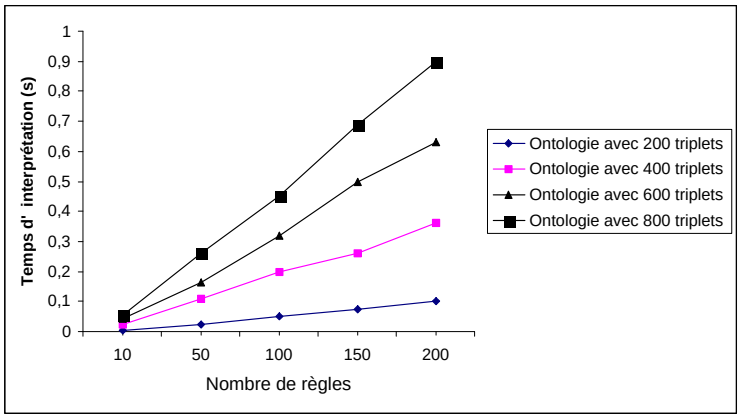

FIG. 5.9 – Temps d'interprétation du contexte en fonction du nombre de règles d'interprétation et de la taille de l'ontologie

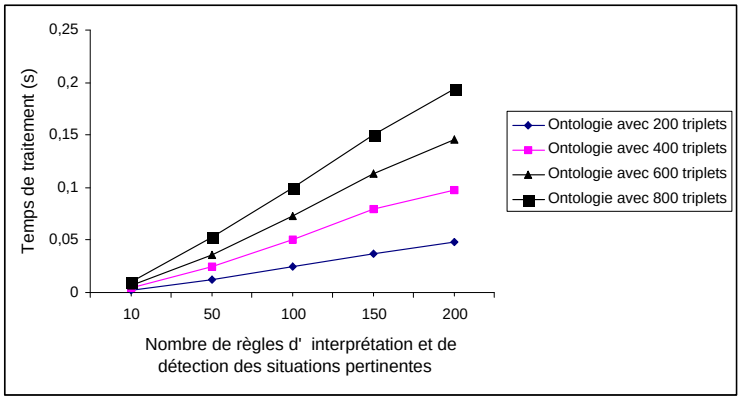

FIG. 5.10 – Temps de détection des situations pertinentes en fonction du nombre de règles de détection des situations pertinentes et de la taille de l'ontologie

règles d'interprétation que les règles de détection situations pertinentes. En effet, pour une ontologie à 800 triplets et un nombre de règles d'interprétation et de détection des situations pertinentes égale à 200 règles, le moteur d'inférence prend 0,9 secondes pour exécuter les règles d'interprétation et 0,2 secondes pour exécuter les règles de détection des situations pertinentes. Ceci s'explique par le contenu de la classe *NewIndirectContextVal* offerte par CAMidO (cf. section 4.5.1) et exécutée par le moteur d'inférence. Lors de l'invocation de cette classe par le moteur d'inférence, elle se charge de déduire la nouvelle valeur du contexte et sauvegarde cette valeur dans l'ontologie. Par conséquent, si le moteur d'inférence déduit 2 contextes de haut niveau en exécutant 2 règles, l'ontologie est mise à jour deux fois.

Pour diminuer le temps d'interprétation du contexte, il est nécessaire que la valeur du contexte déduite ne soit pas sauvegardée dans l'ontologie à chaque exécution d'une règle d'interprétation, il faut la sauvegarder dans une structure de donnée. Une fois que toutes les règles d'interprétation sont exécutées, le composant *ContextAnalyser* (cf. section 4.3.1) se charge de mettre à jour l'ontologie avec les nouvelles valeurs interprétées.

Étude de la réactivité du système

L'étude de la réactivité du système consiste à analyser le temps pris par CAMidO pour traiter une information de contexte 4.3. Ce temps de traitement regroupe le temps de détection de la variation du contexte collecté $Temps_{Nv}$, le temps de mise à jour de l'ontologie $Temps_{Maj}$, le temps d'exécution des règles d'interprétation $Temps_{Deduction}$ et le temps de détection des situations pertinentes $Temps_{Sp}$. Cette étude nous permet de déterminer les fréquences minimales du changement de contexte que peut surveiller CAMidO pour que les adaptations effectuées restent toujours pertinentes [5].

L'étude de la réactivité du système est liée à l'étude du moteur d'inférence puisque ce dernier est utilisé pour interpréter le contexte et détecter les situations pertinentes. De ce fait, lors de nos évaluations, nous avons fait varier le nombre de règles utilisées par CAMidO ainsi que la taille de l'ontologie, puis nous avons étudié le temps nécessaire à CAMidO pour effectuer chaque étape du traitement pour une seule information de contexte.

La figure 5.11 illustre le temps de traitement du contexte en fonction de la taille de l'ontologie, lors de ces tests nous avons fixé le nombre des règles d'interprétation et de détection des situations pertinentes à 200 règles. Les résultats que nous avons obtenus montrent que le temps de détection de la variation du contexte est négligeable et ne dépend pas de la taille de l'ontologie, par contre le temps de mise à jour de l'ontologie varie en fonction de sa taille car cette mise à jour consiste à faire une recherche de l'ancienne valeur du contexte dans l'ontologie, de la supprimer pour la sauvegarder dans une archive qui contient l'historique des variations du contexte, et enfin de sauvegarder la nouvelle valeur du contexte dans l'ontologie. Nous constatons aussi que le temps d'interprétation du contexte et le temps de la détection des situations pertinentes varient en fonction de la taille de l'ontologie, chose que nous avons étudié lors de l'évaluation du moteur d'inférence. Nous avons expliqué cette constatation par le fait que le moteur d'inférence parcourt

[5] Pour qu'une adaptation soit pertinente, il faut être dans le même contexte pertinent lors du lancement de l'adaptation.

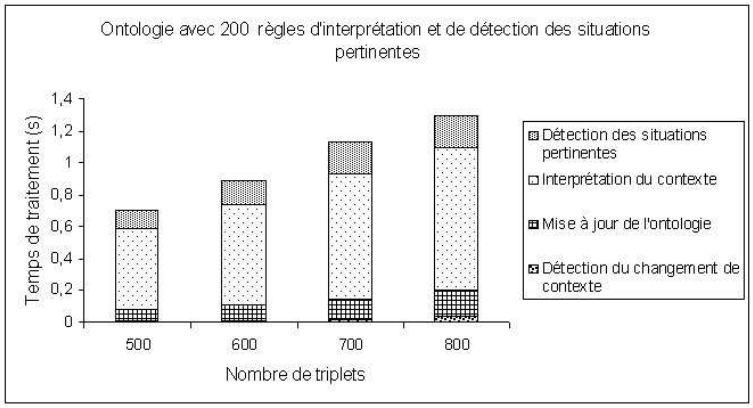

FIG. 5.11 – Temps de traitement d'une information de contexte en fonction de la taille de l'ontologie

toute l'ontologie, pour chaque règle d'interprétation et de détection des situations pertinentes, afin de vérifier la nécessite d'invoquer l'action associée à la règle.

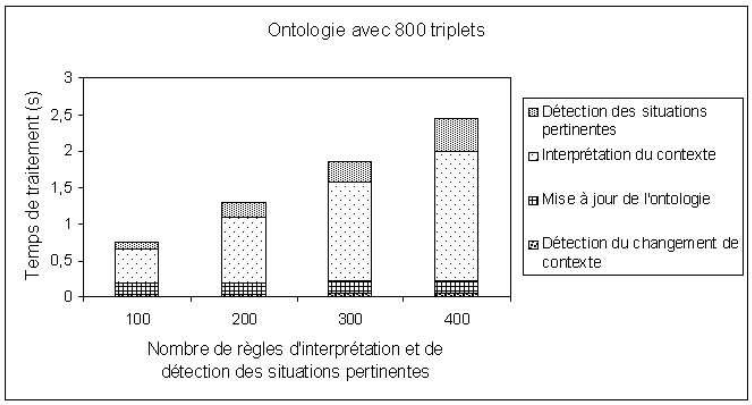

FIG. 5.12 – Temps de traitement d'une information de contexte

La figure 5.12 illustre les variations du temps de traitement d'une information de contexte en fonction du nombre de règles d'interprétation et de détection des situations pertinentes. Lors de ces tests, nous avons fixé la taille de l'ontologie à 800 triplets. Les résultats obtenus montrent que le nombre de règles à une grande influence sur le temps d'interprétation du contexte et le temps de détection des situations pertinentes, par contre ce dernier n'influe pas sur le temps de mise à jour de l'ontologie et le temps de détection d'une nouvelle valeur du contexte collecté.

En comparant les histogrammes de la figure 5.11 et ceux de la figure 5.12, nous remarquons que le nombre de règles à plus d'influence sur le temps de traitement d'une information de contexte que la taille de l'ontologie. En effet, en considérant l'histogramme (800 triplets, 200 règles) comme référence, nous remarquons que l'ajout de 100 règles à plus d'impact sur le temps de traitement d'une information de contexte que l'ajout de 100 triplets. Ceci s'explique par le fait que l'augmentation du nombre de règles à un impact sur toutes les étapes de traitement d'une

information de contexte. De plus, l'ajout des règles d'interprétation et de détection des situations pertinentes peut engendrer la vérification des conditions associées à ces règles et ainsi l'exécution des actions d'interprétation des nouvelles valeurs du contexte et de détection des situations pertinentes appropriées. Or, l'augmentation de la taille de l'ontologie en ajoutant de nouveaux contextes pertinents pour l'application par exemple, n'influe pas sur toutes les étapes de traitement d'une information de contexte et n'engendre pas forcément la vérification des conditions associées aux règles pour le traitement de cette même information de contexte.

Les informations de l'environnement peuvent changer à des fréquences très variables. Pour détecter ces variations, CAMidO doit collecter les informations de contexte à des fréquences déterminées soit par le concepteur de l'application ou par l'agent de maintenance de l'intergiciel. Pour avoir le maximum d'efficacité et de réactivité, CAMidO doit lancer une adaptation à chaque détection de situation pertinente. Lors de l'exécution de cette adaptation, il est nécessaire d'être toujours dans la situation pertinente. Pour vérifier cet objectif, il est nécessaire que la fréquence de collecte de contexte soit inférieur à la plus pertite fréquence de ses variations et que la plus petite fréquence de changement du contexte que CAMidO doit surveiller soit supérieure au temps de traitement du contexte.

A l'issue de l'étude effectuée dans cette section, nous avons déduit que le temps de traitement augmente en fonction de la taille de l'ontologie et du nombre de règles. Selon les histogrammes de la figure 5.11 et la figure 5.12, nous pouvons constater que le temps de traitement du contexte augmente en moyenne de 0,5 secondes à chaque fois que la taille de l'ontologie augmente de 100 triplets. Ce temps augmente aussi en moyenne de 0,1 seconde à chaque fois que le nombre de règles augmente de 100. Cette constatation n'est pas une règle générale, les résultats obtenus dépendent en effet des caractéristiques de la machine utilisée, mais ce qui est important de retenir, c'est le fait que le temps de traitement augmente linéairement avec la taille de l'ontologie et le nombre de règles. De ce fait, il est possible de déterminer le degré de cette variation, pour une machine donnée, et ainsi déduire la valeur minimale de la fréquence de changement du contexte que CAMidO peut surveiller lorsqu'il est installé sur cette machine. Cette variation est calculée en fonction de la taille de l'ontologie et du nombre de règles d'interprétation et de décetion des situations pertinentes en effectuant un ensemble de tests. Dans notre cas, si nous exécutons une application dont l'ontologie est constituée de 800 triplets et 200 règles, la fréquence de changement de contexte minimale que CAMidO peut gérer doit être supérieure à 1,5 secondes selon l'histogramme de la figure 5.11.

Les résultats obtenus lors des tests présentés dans cette section nous permettent de déduire que CAMidO ne peut pas être utilisé pour gérer des applications temps réel sensibles au contexte dont la taille de l'ontologie associé excède 800 triplets avec un nombre de de règles qui excède 400 règles. En effet, nous pouvons constater dans les histogrammes présentés dans cette section que les temps de réaction à un changement pertinent de contexte sont de l'ordre de 2,5 secondes dans le cas d'une ontologie contenant 800 triplets et 400 règles, ce qui est inconcevable comme temps de réaction dans des applications où le temps de réaction est critique comme les applications temps réel.

Évaluation de la reconfiguration dynamique

Le mode reconfiguration dynamique est offert par CAMidO pour donner la possibilité aux utilisateurs de configurer des applications au cours de leur exécution. L'intergiciel CAMidO prend en compte ces modifications sans arréter l'exécution de ces applications. Pour atteindre cet objectif, CAMidO consulte l'ontologie pour chaque étape de traitement de l'information collectée et génère les règles d'interprétation et d'adaptation avant chaque analyse ou interprétation du contexte. Par conséquent, il est nécessaire d'étudier le coût engendré par ce mode d'exécution.

En mode reconfiguration dynamique, le temps de traitement d'une information de contexte est donné par la formule suivante :

$Temps_{Traitement_reconf}$=$Temps_{Analyse_reconf}$ +$Temps_{Interpretations_reconf}$.

$Temps_{Analyse_reconf}$=$Temps_{Nv}$+$Temps_{Maj}$+ $Temps_{Grsp}$+ $Temps_{Sp}$

$Temps_{Interpretations_reconf}$= $Temps_{Gri}$+$Temps_{Deduction}$

$Temps_{Maj}$=$Temps_{MajOnto}$+ $Temps_{MajArchive}$

$Temps_{Analyse_reconf}$: représente le temps d'analyse total d'une information de contexte en mode reconfiguration dynamique.

$Temps_{Grsp}$: représente le temps nécessaire pour la génération des règles de détection des situations pertinentes.

$Temps_{Interpretations_reconf}$: représente le temps total de l'interprétation en mode reconfiguration dynamique.

$Temps_{Gri}$: représente le temps nécessaire pour la génération des règles d'interprétation.

$Temps_{Nv}$: représente le temps que le *ContextAnalyser* prend pour déterminer s'il s'agit d'une nouvelle valeur du contexte.

Maj : représente le temps de sauvegarde d'une information de contexte.

$Temps_{MajOnto}$: représente le temps de sauvegarde d'une information de contexte dans l'ontologie.

$MajArchive$: représente le temps de sauvegarde d'une information de contexte dans une archive.

$Temps_{Sp}$: représente le temps de détection d'une situation pertinente.

$Temps_{Interpretation}$: représente le temps que prend le moteur d'inférence pour exécuter les règles d'interprétation.

Pour étudier le coût de la reconfiguration dynamique nous avons utilisé une ontologie contenant 800 triplets et 200 règles d'interprétation et de détection des situations pertinentes. Puis, nous avons calculé le temps nécessaire à CAMidO pour traiter et adapter l'application lors de la collecte d'une informations de contexte. Enfin, nous avons comparé les résultats avec ceux obtenus lors de l'exécution de la même ontologie en mode configuration statique.

La figure 5.13 illustre les variations du temps d'interprétation en fonction du nombre de règles d'interprétation en mode configuration statique et en mode reconfiguration dynamique pour le traitement d'une information de contexte collectée. Ce temps d'interprétation est représenté par $Temps_{Interpretations_reconf}$ en mode reconfiguration dynamique et par $Temps_{Dedution}$ en mode configuration statique. Les résultats des tests montrent que le mode reconfiguration dynamique est très sensible au nombre de règles d'interprétation. En effet, le temps d'interprétation lors du traitement d'une information de contexte passe de 2,95 secondes pour 100 règles d'interpréta-

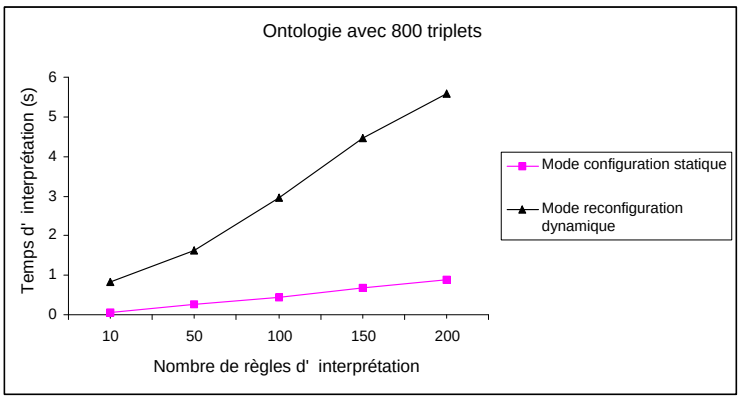

FIG. 5.13 – Mesures du temps d'interprétation en mode confi guration statique et reconfi guration dynamique

tion à 4,45 secondes pour 150 règles d'interprétation. De plus pour le traitement de la même information de contexte avec 100 règles d'interprétation, CAMidO prend 0,45 secondes en mode configuration statique et 2,95 secondes en mode reconfiguration dynamique. Ce coût est engendré par la génération des règles d'interprétation avant le traitement de chaque information de contexte.

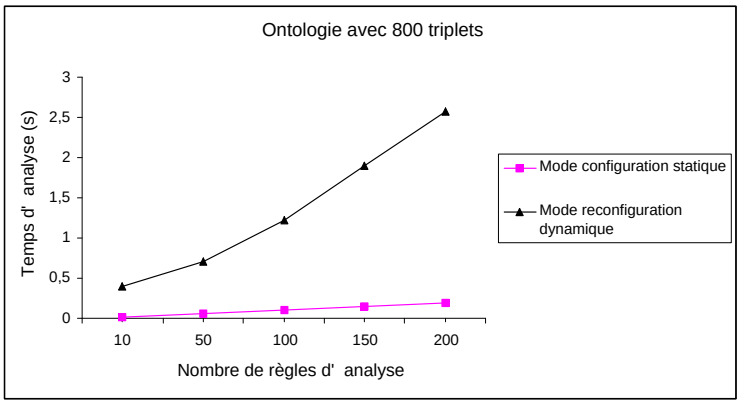

FIG. 5.14 – Mesures du temps d'analyse du contexte en mode confi guration statique et reconfi guration dynamique

La figure 5.14 illustre le temps nécessaire à CAMidO pour analyser une information de contexte en mode reconfiguration dynamique et en mode configuration statique. Le temps d'analyse est représenté par $Temps_{Analyse}$ en mode configuration statique et par $Temps_{Analyse_reconf}$ en mode reconfiguration dynamique sachant que $Temps_{Analyse_reconf}$= $Temps_{Analyse}$+$Temps_{Grcp}$. Les résultats de nos tests montrent que le temps de génération des règles de détection des situations pertinentes n'est pas négligeable et dépend du nombre de règles de détection des situations pertinentes considérés . En effet, l'analyse d'une information de contexte en mode reconfiguration dynamique passe de 1,9 secondes pour 150 règles à 2,57 secondes pour 200 règles. De plus, pour le traitement de la même information de contexte avec

150 règles de détection des situations pertinentes, CAMidO prend 0,15 seconde en mode configuration statique contre 1,9 secondes en mode reconfiguration statique ce qui nous emmène à déduire que $Temps_{Grsp}$ est égale à 1,75 secondes. Il est important de souligner qu'en mode reconfiguration dynamique, CAMidO doit générer ces règles pour chaque information de contexte qu'il doit traiter. Donc pour traiter n informations de contexte, CAMidO a généré n fois les règles de détection des situations pertinentes.

A l'issue des résultats obtenus lors de ces évaluations, nous concluons que le mode reconfiguration dynamique est très gourmand en temps de calcul à cause de la nécessité de générer les règles d'interprétation et de détection des situations pertinentes pour chaque information de contexte. Par conséquent, ce mode ne peut pas être utilisé lorsque le temps de réactivité du système est critique.

Il est possible d'envisager des améliorations du temps de traitement en mode reconfiguration dynamique. Pour cela, il ne faut pas générer les règles d'interprétation du contexte de haut niveau et de détection des situations pertinentes pour chaque information de contexte à traiter. La génération des règles ne doit s'effectuer que si le modèle de l'application a changé. L'intergiciel CAMidO peut détecter le changement du modèle de l'application en utilisant un thread démon qui surveille constamment le modèle et qui notifie l'entité qui se charge de générer ces règles pour tout changement que peut subir le modèle.

Étude du coût de l'adaptation

Dans cette section, nous évaluons le coût engendré par l'adaptation réactive et l'adaptation proactive, pour cela nous calculons le temps qu'un composant sensible au contexte prend pour exécuter un ensemble de requêtes tout en effectuant des adaptations. Puis, nous comparons les résultats obtenus avec le temps d'exécution des même requêtes par un composant installé au-dessus d'OpenCCM sans utiliser CAMidO. Il est important de souligner que lors des tests sur les adaptations réactives nous avons désinstallé les intercepteurs portables pour pouvoir évaluer le coût effectif de ce type d'adaptation.

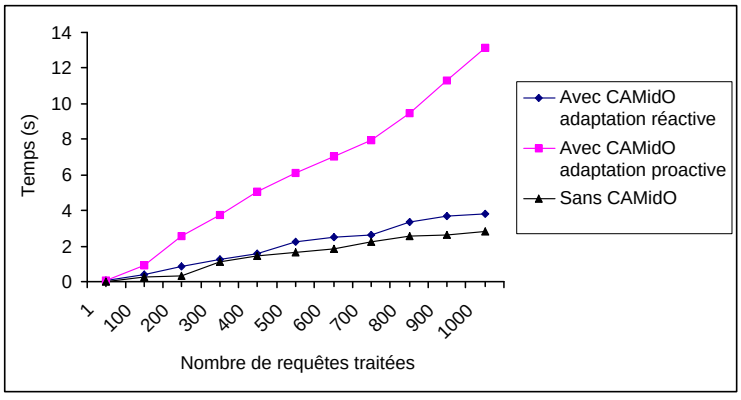

FIG. 5.15 – Mesures du coût des adaptations réactives et proactives

La figure 5.15, illustre les résultats obtenus lors de ces expérimentations. En prenant comme repère le graphe associé au composant installé au-dessus d'openCCM, nous constatons que l'adaptation réactive engendre un coût négligeable par rapport aux adaptations proactives. En effet, le temps de traitement de 400 requêtes par un composant qui effectue 70 adaptations réactives est presque identique au temps de leur traitement par un composant installé au dessus d'OpenCCM (1.5 secondes pour le composant qui effectue des adaptations réactives contre 1.4 secondes pour le composant installé au dessus d'OpenCCM). Ce résultat s'explique par le fait que l'adaptation réactive consiste à invoquer une opération, dont le temps de traitement est négligeable.

Les adaptations proactives engendrent un coût de traitement plus important, ceci s'explique par l'utilisation des intercepteurs portables à chaque invocation de requête. Toutes les requêtes sortantes ou entrantes vers un composant sont interceptées respectivement par le *CAClientInterceptor* et le *CAServerInterceptor* puis redirigées vers le *CAMidOProxy* pour vérifier si une adaptation proactive est nécessaire. Le coût de cette redirection représente le coût de l'adaptation proactive. Ce coût diminue considérablement le nombre de requêtes qu'un composant peut traiter en un laps de temps bien déterminé. En effet, 800 requêtes sont traitées en 9.45 secondes lorsque les adaptations proactives sont prises en compte par le système contre 3.34 secondes dans le cas des adaptations réactives avec désinstallation des intercepteurs portables, et 2.57 secondes dans le cas d'un composant qui n'utilise pas CAMidO.

Dans la réalité, les intercepteurs portables sont automatiquement installés au-dessus de l'ORB lors de l'exécution de CAMidO, par conséquent, toutes les requêtes transmises sur le bus sont interceptées puis redirigées vers le *CAMidOProxy*. De ce fait, le coût de cette redirection concerne autant l'adaptation réactive que l'adaptation proactive.

Selon le graphe de la figure 5.15, au-dessous de 200 requêtes, le coût engendré par la redirection des requêtes n'est pas très important comparé au coût engendré par le traitement de plus de 200 requêtes. Par conséquent, il est préférable d'installer au-dessus de CAMidO des applications qui ne doivent pas répondre à plus de 200 requêtes en moins d'une seconde. Dans le cas où le temps de réponse est critique, et que l'application doit traiter plus de 200 requêtes par seconde, nous conseillons de ne pas installer les intercepteurs portables et de ne considérer que les adaptations réactives.

5.4 Conclusion

Dans ce chapitre, nous avons présenté l'implantation d'un prototype de CAMidO au-dessus d'OpenCCM et nous avons discuté les résultats de son évaluation. Nous avons utilisé l'application *SensitivePurchase* et le service CADeComp pour évaluer les résultats qualitatifs en terme d'efficacité d'utilisation et de possibilité d'extension, tandis que nous avons effectué un ensemble de tests pour étudier les performances de CAMidO.

En terme de résultats qualitatifs, nous avons montré que CAMidO offre des abstractions et des séparations des préoccupations utiles pour le développement des applications orientées composants sensibles au contexte. En effet, le méta-modèle de CAMidO qui sépare la description du

contexte de son utilisation facilite le développement des applications sensibles au contexte en offrant aux concepteurs le moyen de créer ce type d'applications en procédant par description. De plus, la séparation entre la gestion du contexte et la gestion de l'adaptation élargit le champ d'utilisation de CAMidO par des services ou des applications sensibles au contexte qui possèdent leurs propres mécanismes d'adaptation. L'extension de CAMidO nécessite une bonne connaissance du méta-modèle qu'il offre, une bonne connaissance des *templates* utilisées par le compilateur et une bonne connaissance de la structure des règles générées, ce qui rend difficile cette extension. Néanmoins, pour faciliter cette tâche, nous offrons un ensemble de bibliothèques qui peuvent aider les développeurs à étendre le modèle de base de CAMidO.

L'évaluation des performances de CAMidO nous a conduit à étudier les performances du moteur d'inférence et des paramètres auxquels son temps de traitement est sensible, la réactivité du système, le coût de la reconfiguration dynamique et celui de l'adaptation réactive et l'adaptation proactive. A l'issue de cette étude, nous avons conclu que la taille de l'ontologie et le nombre de règles d'interprétation sont des facteurs qui augmentent le temps de traitement des informations de contexte. Ce temps peut être utilisé pour déterminer la fréquence minimale de collecte des informations de contexte. Le mode reconfiguration dynamique engendre un coût de traitement non négligeable à cause de l'accès permanent à l'ontologie et à la génération des fichiers de règles lors du traitement de chaque information de contexte. Par conséquent, nous déconseillons l'utilisation de ce mode pour des applications temps réel. CAMidO prend en compte deux types d'adaptations, les adaptations réactives et les adaptations proactives. Les adaptations proactives nécessitent l'utilisation des intercepteurs portables qui se chargent d'intercepter toutes les requêtes et de les rediriger vers le *CAMidOProxy* afin d'appliquer l'adaptation appropriée si cela est nécessaire. Par conséquent, le nombre de requêtes auxquelles un composant peut répondre en un laps de temps déterminé diminue à cause de cette redirection. Dans le cas où le temps de réponse est critique, et qu'une application doit traiter plus de 200 requêtes pas seconde, nous conseillons de ne pas installer les intercepteurs portables et de ne considérer que les adaptations réactives.

Troisième partie

Conclusions et perspectives

Conclusions

Les dispositifs mobiles sont caractérisés par un environnement d'exécution dynamique. De ce fait, il est nécessaire de considérer un nouveau type d'applications qui détectent le changement de l'environnement et qui adaptent leurs comportements en conséquence.

Le développement des applications sensibles au contexte est une tâche fastidieuse qui nécessite plusieurs étapes de programmation. L'objectif de cette thèse étant de faciliter le développement des applications orientées composant sensibles au contexte, nous avons entamé notre étude par un état de l'art sur la modélisation du contexte et la gestion de la sensibilité au contexte par des intergiciels. Cette étude nous a permis de mieux cerner les difficultés engendrées par la création des applications sensibles au contexte du point de vue intergiciel orienté composant. Tout d'abord, nous avons étudié les différentes définitions du contexte, ses caractéristiques et son importance dans le domaine informatique. Puis, nous avons étudié différentes approches de modélisation du contexte et l'apport de chacune d'elles concernant l'expressivité du modèle, la possibilité de sa réutilisation et son extension, et la possibilité de partager les informations décrites. Ceci nous a permis de choisir une approche de modélisation qui nous semble être la mieux adaptée à la description du contexte dans un environnement distribué. Ensuite, nous avons étudié les éléments fonctionnels des applications sensibles au contexte. Cette étude nous a permis de souligner la complexité du développement de ce type d'applications. Nous avons étudié aussi les caractéristiques des intergiciels et les techniques d'adaptation qu'ils peuvent utiliser pour gérer l'adaptation des applications distribuées. Enfin, nous avons étudié les intergiciels sensibles au contexte dont le but consiste à faciliter le développement des applications sensibles au contexte. Cela nous a permis de déduire que ces intergiciels ne prennent en compte qu'une partie des éléments fonctionnels de la gestion du contexte, et laissent le reste à la charge du développeur. Pour palier à ce manque, nous avons proposé un intergiciel sensible au contexte qui décharge le développeur de toutes ces tâches.

Contributions

Le développement d'une application sensible au contexte consiste tout d'abord à décrire le contexte et à programmer sa gestion. Les éléments fonctionnels de la gestion du contexte regroupent l'interaction de l'application avec les capteurs pour collecter les informations de contexte, l'interprétation du contexte de haut niveau, l'analyse des données collectées et interprétées et

l'adaptation de l'application lors de la détection d'une situation pertinente. Ces étapes de développement sont difficiles à mettre en œuvre. Les solutions existantes que nous avons étudiées dans la première partie de cette thèse, permettent en effet de faciliter le développement des applications sensibles au contexte en prenant en compte une partie de ces éléments fonctionnels par un intergiciel, par contre les tâches restantes sont à la charge du développeur.

L'objectif de cette thèse est de faciliter encore plus le développement des applications orientées composants sensibles au contexte, en prenant en compte tous les éléments fonctionnels de la gestion du contexte par un intergiciel sensible au contexte appelé CAMidO. Nos contributions s'étendent de la modélisation du contexte à la mise en oeuvre d'un intergiciel pour la sensibilité au contexte. Nous résumons ces contributions en cinq points essentiels.

Méta-modèle de description du contexte

Nous avons défini le méta-modèle de CAMidO qui permet de décrire des informations liées à la sensibilité au contexte pour des applications orientées composant. L'objectif de ce méta-modèle n'est pas de définir une nouvelle ontologie pour décrire le contexte, mais de s'appuyer sur des ontologies standards et de les étendre et ceci, afin de décrire les informations nécessaires à l'automatisation des processus de gestion du contexte et d'adaptation dans un environnement orienté composant sensible au contexte. Le méta-modèle proposé est structuré en trois niveaux, les deux premiers niveaux permettent de décrire des concepts généraux aux applications sensibles au contexte, alors que le dernier niveau permet de décrire des informations plus spécifiques aux applications orientées composant sensibles au contexte. La valeur ajoutée du méta-modèle de CAMidO, par rapport aux travaux cités dans la première partie de cette thèse, réside dans le fait qu'il permet non seulement la description des caractéristiques du contexte, mais considère aussi le fait que ce contexte est capturé puis utilisé par un système qui se charge d'adapter l'application aux situations pertinentes du contexte. Par conséquent, cette description peut être utilisée pour automatiser le processus de gestion du contexte et celui de l'adaptation.

Gestionnaire de contexte

Nous avons défini un gestionnaire de contexte qui regroupe un ensemble de composants qui se chargent d'interagir avec les capteurs, d'interpréter le contexte, de l'analyser et de détecter les situations pertinentes. Ce gestionnaire utilise le modèle fourni par le concepteur d'application afin d'exécuter ces tâches. Il permet aussi à chaque composant de l'application de s'enregistrer pour les situations pertinentes auxquelles il est sensible, et se charge de les notifier lorsque ces situations sont détectées.

Le gestionnaire de contexte prend en charge tous les éléments fonctionnels de la gestion du contexte, ce qui facilite ainsi aux développeurs la création d'applications orientées composants sensibles au contexte. En effet, la tâche des développeurs se résume uniquement à décrire le modèle de sensibilité au contexte associé à leurs applications en utilisant le méta-modèle de CAMidO.

Modèle de conteneur pour la gestion de l'adaptation au contexte

Nous avons proposé la gestion de l'adaptation dans les conteneurs des composants. Le conteneur que nous proposons offre un lien entre le composant et le gestionnaire de contexte grâce aux contrôleurs que nous avons intégrés à son architecture. L'un de ces contrôleurs permet d'enregistrer le composant auprès du gestionnaire de contexte pour toutes les situations pertinentes auxquelles il est sensible, alors que les autres se chargent d'appliquer les adaptations réactives et proactives lorsque cela est nécessaire. Le contrôleur qui se charge de l'adaptation proactive travail en collaboration avec des intercepteurs portables et un proxy pour intercepter les invocations des composants et appliquer les adaptations proactives appropriées si cela est nécessaire.

Comme décrit dans l'état de l'art, il existe peu de solutions qui abordent la gestion de l'adaptation pour des applications à base de composants. Notre solution se distingue des autres par le fait qu'elle exploite le paradigme composant/conteneur en collaboration avec les intercepteurs portables pour la gestion des adaptations au contexte. Ainsi, nous avons démontré avec le modèle de CAMidO, le gestionnaire de contexte et notre modèle de conteneur, que le principe de séparation des préoccupations fonctionnelles et extra-fonctionnelles dans le paradigme composant/conteneur est approprié à la gestion des adaptations au contexte.

Deux modes d'exécution : configuration statique et reconfiguration dynamique

CAMidO fournit deux modes d'exécution : le mode configuration statique et le mode reconfiguration dynamique. Le mode configuration statique utilise les informations générées par le compilateur CAMidO. Ce compilateur se charge de générer d'une part des règles de logique de premier ordre pour l'interprétation du contexte et la détection des situations pertinentes et d'autre part le code source nécessaire à l'interprétation du contexte, la détection des situations pertinentes et l'adaptation de l'application. Le mode reconfiguration dynamique permet à l'intergiciel CAMidO de prendre en compte les modifications effectuées sur le modèle de l'application après son exécution. Ces modifications peuvent être nécessaires pour ajouter des éléments non pris en compte lors de la conception de l'application. Ces éléments concernent les contextes pertinents pour l'application, les situations pertinentes, les règles d'interprétation et les politiques d'adaptation.

Évaluation de CAMidO

Nous avons implanté et intégré un prototype de CAMidO à la plate-forme OpenCCM, ensuite nous avons effectué des évaluations qualitatives et quantitatives de ce prototype. Les évaluations qualitatives ont montré que les abstractions et les séparations des préoccupations facilitent le développement des applications orientées composants sensibles au contexte et élargissent le champ d'utilisation de CAMidO à des services ou des applications sensibles au contexte ayant leurs propre mécanismes d'adaptation. Les évaluations des performances ont montré que la taille de l'ontologie et le nombre de règles d'interprétation et d'adaptation augmentent le temps de

traitement des informations de contexte surtout en mode reconfiguration dynamique. Par conséquent, ce mode n'est pas adapté aux applications temps réel. De plus, les adaptations proactives qui utilisent les intercepteurs portables augmentent le temps de traitement des requêtes à cause de leur redirection, donc les adaptations proactives ne doivent pas être considérées dans des applications où les composants sont massivement sollicités et où le temps de réponse est critique.

Perspectives

Le travail que nous avons proposé ne résout pas tous les problèmes liés à la sensibilité au contexte, plusieurs perspectives de recherches complémentaires nous semblent intéressantes à explorer.

Fonctionnement de CAMidO sur terminaux mobiles

Actuellement l'implantation de CAMidO a été testée sur des ordinateurs portables. Nous estimons qu'il est important de tester son fonctionnement sur des dispositifs mobiles ayants des ressources limitées.

Distribution de CAMidO

Le gestionnaire de contexte de CAMidO est structuré en un ensemble d'entités qui travaillent en collaboration pour gérer le contexte. Il nous semble important de distribuer les entités qui constituent ce gestionnaire. Cette distribution permet d'installer chaque entité de gestion du contexte sur un terminal différent ce qui permet d'augmenter la fiabilité et la disponibilité de ces entités.

Intégration de CAMidO à d'autres intergiciels à composants

Nous avons intégré l'intergiciel CAMidO à la plate-forme OpenCCM. Il nous semble intéressant d'intégrer nos propositions à d'autres plates-formes à composants comme les modèles EJB et Fractal. Tout d'abord, ceci nous permettrait de démontrer la généricité des mécanismes proposés. Ensuite, les réalisations en source libre de Fractal et EJB (par exemple, JBoss ou Jonas) sont intéressantes pour la mise en oeuvre de notre conteneur.

Prise en compte d'autres type d'adaptation

CAMidO prend en compte deux natures d'adaptations comportementales : les adaptations comportementales réactives et les adaptations comportementales proactives. Il nous semble intéressant d'étendre le gestionnaire d'adaptation pour gérer d'autre types d'adaptations, comme l'adaptation structurelle de l'application. la modification de la structure d'une application consiste

à modifier les interconnexions des composants, de supprimer des instances de composants ou d'ajouter de nouvelles instances de composant.

Gestion des conflits d'adaptation

L'intergiciel CAMidO permet la gestion de plusieurs applications sensibles au contexte, ce qui peut engendrer l'apparition de conflits intra-application. Ces conflits se traduisent par la nécessité d'exécuter des adaptations conflictuelles comme par exemple le téléchargement d'un fichier, et la déconnexion de la machine. Nous estimons qu'il est intéressant d'intégrer à CAMidO un mécanisme de consensus pour résoudre ce type de conflits. Ce mécanisme doit avoir connaissance des adaptations associées à chaque situation pertinente et des ressources nécessaires à son exécution.

Gestion de l'intimité (*privacy protection*)

Les applications sensibles au contexte utilisent différents types de données allant de la température d'une pièce à la localisation de l'utilisateur ainsi que son profil et ses préférences. Ces informations de contexte sont accessibles n'importe où et n'importe quand. Les informations concernant l'utilisateur rentrent dans le domaine de sa vie privée. Par conséquent, il est nécessaire de restreindre l'accès à ces données en les protégeant. Nous estimons qu'il est intéressant d'intégrer à CAMidO un gestionnaire d'intimité (en anglais, *privacy protection manager*). La gestion de l'intimité peut se faire au niveau de l'ontologie, en donnant la possibilité à l'utilisateur de définir des règles qui autorisent ou pas l'accès aux informations privées stockées dans l'ontologie, et au niveau du gestionnaire de contexte en intégrant un composant qui se charge de permettre ou de refuser l'enregistrement des composants sensibles au contexte auprès du *ContextManager* et cela, selon les règles d'accès fournies par l'utilisateur.

Gestion de la cohérence des applications

Dans le cadre de CAMidO des incohérences peuvent apparaître lorsqu'un ensemble de requêtes dépendantes est interrompu par une redirection d'une de ces requêtes vers un autre composant. Ce dernier n'ayant aucune connaissance de l'historique des requêtes peut altérer le bon fonctionnement de l'application. Il nous semble donc primordial d'intégrer un gestionnaire de cohérence pour s'assurer que les applications restent dans un état cohérent après l'exécution d'une adaptation.

L'informatique sensible au contexte est un domaine très vaste qui donne libre court à l'imagination des développeurs dans la conception d'applications sensibles au contexte. Nos contributions aident ces développeurs à concrétiser ces applications en procédant par description plutôt que par programmation.

Quatrième partie

Annexes

Annexe A

Description OWL du méta-modèle de CAMidO

A.1 Niveau *Capteur* du méta-modèle

```
<?xml version="1.0"?> <rdf:RDF
    xmlns="http://etna.int-evry.fr/~belhanaf/Sensor_CAMidO#"
    xml:base="http://etna.int-evry.fr/~belhanaf/Sensor_CAMidO#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<owl:Ontology rdf:about=""/>

  <owl:Ontology rdf:about="Sensor_CAMidO">
    <owl:versionInfo>$Version 1  $</owl:versionInfo>
    <rdfs:label>CAMidO Sensor level Ontology</rdfs:label>
    <rdfs:comment></rdfs:comment>
  </owl:Ontology>

<!--Sensor Level description -->

  <owl:Class rdf:ID="Sensor"/>

  <owl:DatatypeProperty rdf:ID="sensorType">
    <rdfs:range rdf:resource="#SensorTypes"/>
    <rdfs:domain rdf:resource="#Sensor"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:ID="SensorTypes">
    <owl:oneOf rdf:parseType="Collection">
      <Op rdf:about="CORBA"/>
```

```
      <Op rdf:about="Class"/>
    </owl:oneOf>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="sensorName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Sensor"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="sensorSourcePath">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Sensor"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="sensorSourcePackage">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Sensor"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="sensorParameter">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Sensor"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="sensorParameterType">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Sensor"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="sensorReturnedType">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Sensor"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="frequency">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
    <rdfs:domain rdf:resource="#Sensor"/>
  </owl:DatatypeProperty>

</rdf:RDF>
```

A.2 Niveau *Contexte* du méta-modèle

```
<?xml version="1.0"?> <rdf:RDF
    xmlns="http://etna.int-evry.fr/~belhanaf/Context_CAMidO#"
    xml:base="http://etna.int-evry.fr/~belhanaf/Context_CAMidO#"
```

```
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<owl:Ontology rdf:about=""/>

  <owl:Ontology rdf:about="Context_CAMidO">
    <owl:versionInfo>$Version 1  $</owl:versionInfo>
    <rdfs:label>CAMidO Context level Ontology</rdfs:label>
    <owl:imports rdf:resource="http://etna.int-evry.fr/~belhanaf/Sensor_CAMidO"/>
    <rdfs:comment></rdfs:comment>
  </owl:Ontology>

<!-- Association class  description-->

<owl :Class rdf ID="ExtendedProperty">
  <rdfs:subClassOf OWL:ObjectProperty/>
</owl:Class>

<!-- Context type description-->

  <owl:Class rdf:ID="Context"/>

  <owl:Class rdf:ID="Direct">
    <rdfs:subClassOf rdf:resource="#Context"/>
    <owl : Restriction>
      <owl: maxCardinality rdf:datatype="&xsd;#int">
        1
      </owl:maxCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:resource="#TakesDataFrom">
      </owl:onProperty>
    </owl : Restriction>
  </owl:Class>

  <owl:Class rdf:ID="Indirect">
    <rdfs:subClassOf rdf:resource="#Context"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="contextName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Context"/>
  </owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:ID="contextType">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Context"/>
  </owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="T">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#time"/>
    <rdfs:domain rdf:resource="#Context"/>
  </owl:DatatypeProperty>

  <owl:ObjectProperty rdf:ID="TakesDataFrom">
    <rdfs:range rdf:resource="#Sensor"/>
    <rdfs:domain rdf:resource="#Indirect"/>
  </owl:ObjectProperty>

<!-- Action Class type description-->

  <owl:Class rdf:ID="Action"/>

  <owl:DatatypeProperty rdf:ID="actionOperationName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Action"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="actionClassName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Action"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="actionClassPath">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Action"/>
  </owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:ID="actionPackage">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Action"/>
  </owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:ID="actionParamType">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Action"/>
  </owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:ID="actionParam">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Action"/>
  </owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:ID="actionReturnedType">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Action"/>
  </owl:DatatypeProperty>
```

```
<!-- Condition description-->

  <owl:Class rdf:ID="Condition"/>

  <owl Class rdf:ID="LeafCondition">
    <rdfs:subClassOf rdf:resource="#Condition"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#Operator">
      <owl:allValuesFrom rdf:resource="#Operat">
    </owl:Restriction>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="Operator">
    <rdfs:range rdf:resource="#Condition"/>
    <rdfs:domain rdf:resource="#LeafCondition"/>
  </owl:ObjectProperty>

  <owl:Class rdf:ID="Operat">
      <owl:oneOf rdf:parseType="Collection">
        <Op rdf:about="Equal"/>
        <Op rdf:about="NotEqual"/>
        <Op rdf:about="Greater"/>
        <Op rdf:about="Less"/>
        <Op rdf:about="Ge"/>
        <Op rdf:about="Le"/>
      </owl:oneOf>
  </owl:Class>

  <owl:Class rdfID="CompositeCondition">
    <rdfs:subClassOf rdf:resource="#Condition"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#LogicalOperator">
      <owl:allValuesFrom rdf:resource="#LogicOperator">
    </owl:Restriction>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="LogicalOperator">
    <rdfs:range rdf:resource="#CompositeCondition"/>
    <rdfs:domain rdf:resource="#Condition"/>
  </owl:ObjectProperty>

  <owl:Class rdf:ID="LogicOperator">
      <owl:oneOf rdf:parseType="Collection">
      <Op rdf:about="And"/>
      <Op rdf:about="Or"/>
      </owl:oneOf>
  </owl:Class>
```

```
<!-- Interpretation Rule description-->

  <owl:Class rdf:ID="HowDeduce">
    <rdfs:subClassOf owl:ExtendedProperty/>
    <rdfs:range rdf:resource="#Indirect"/>
    <rdfs:domain rdf:resource="#Context"/>
  </owl:Class>

  <owl:Class rdf:ID="Deduction">
    <rdfs:subClassOf owl:ExtendedProperty/>
    <rdfs:domain rdf:resource="#ContextHowDeduce"/>
  </owl:Class>

  <owl :Class rdf ID="InterpretationOperation">
    <rdfs:subClassOf rdf:resource="#Action"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="ExecuteDeduction">
    <rdfs:domain rdf:resource="#Deduction"/>
    <rdfs:range rdf:resource="#InterpretationOperation"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="IfDeduction">
    <rdfs:domain rdf:resource="#Deduction"/>
    <rdfs:range rdf:resource="#Condition"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

A.3 Niveau *Application* du méta-modèle

```
<?xml version="1.0"?> <rdf:RDF
    xmlns="http://etna.int-evry.fr/~belhanaf/Application_CAMidO#"
    xml:base="http://etna.int-evry.fr/~belhanaf/Application_CAMidO#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<owl:Ontology rdf:about=""/>

  <owl:Ontology rdf:about="Application_CAMidO">
    <owl:versionInfo>$Version 1  $</owl:versionInfo>
    <rdfs:label>CAMidO Application level Ontology</rdfs:label>
    <owl:imports rdf:resource="http://etna.int-evry.fr/~belhanaf/Sensor_CAMidO"/>
    <owl:imports rdf:resource="http://etna.int-evry.fr/~belhanaf/Context_CAMidO"/>
    <rdfs:comment></rdfs:comment>
```

```
  </owl:Ontology>

<!--Application description -->

  <owl:Class rdf:ID="Application"/>

<owl:DatatypeProperty rdf:ID="ApplicationPath">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  <rdfs:domain rdf:resource="#Application"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="ApplicationClassPath">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  <rdfs:domain rdf:resource="#Application"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="ApplicationPackage">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  <rdfs:domain rdf:resource="#Application"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="Contains">
  <rdfs:range rdf:resource="#Component"/>
  <rdfs:domain rdf:resource="#Application"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Use">
  <rdfs:range rdf:resource="#RelevantContext"/>
  <rdfs:domain rdf:resource="#Application"/>
</owl:ObjectProperty>

<!--RelevantContext description -->

<owl:Class rdf:ID="RelevantContext">
  <owl:Restriction>
     <owl:onProperty>
       <owl:objectProperty rdf:resource="#Subset">
     <owl:onProperty>
     <owl:allValuesFrom rdf:resource="#Context"/>
  </owl:Restriction>
</owl:Class>

<owl:ObjectProperty rdf:ID="Subset">
  <rdfs:range rdf:resource="#Context"/>
  <rdfs:domain rdf:resource="#RelevantContext"/>
</owl:ObjectProperty>

<!--Component description -->
```

```
<owl:Class rdf:ID="Component"/>

<owl:DatatypeProperty rdf:ID="ComponentName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  <rdfs:domain rdf:resource="#Component"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="AwareOf">
  <rdfs:range rdf:resource="#RelevantSituation"/>
  <rdfs:domain rdf:resource="#Component"/>
</owl:ObjectProperty>

<!--Context aware service description -->

<owl:Class rdf:ID="CAService"/>

<owl:DatatypeProperty rdf:ID="CAServiceName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  <rdfs:domain rdf:resource="#CAService"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="RegistredFor">
  <rdfs:range rdf:resource="#RelevantSituation"/>
  <rdfs:domain rdf:resource="#CAService"/>
</owl:ObjectProperty>

<!-- Relevant situation description-->

  <owl:Class rdf:ID="RelevantSituation"/>

 <owl:ObjectProperty rdf:ID="Necessitates">
    <rdfs:range rdf:resource="#Adaptation"/>
    <rdfs:domain rdf:resource="#RelevantSituation"/>
  </owl:ObjectProperty>

 <owl:ObjectProperty rdf:ID="RelevantAction">
    <rdfs:range rdf:resource="#Adaptation"/>
    <rdfs:domain rdf:resource="#BooleanAction"/>
  </owl:ObjectProperty>

  <owl:Class rdf:ID="BooleanAction">
    <rdfs:subClassOf owl:Action/>
    <actionReturnedType>Boolean</actionReturnedType>
  </owl:Class>

<!--  Adaptation description-->
```

```
  <owl:Class rdf:ID="Adaptation"/>


  <owl:ObjectProperty rdf:ID="OperationToCall">
    <rdfs:range rdf:resource="#Action"/>
    <rdfs:domain rdf:resource="#Adaptation"/>
  </owl:ObjectProperty>

  <owl:Class rdf:ID="AdaptationAction">
    <rdfs:subClassOf owl:Action/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="OperationLocation">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Adaptation"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="OperationComponent">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#Adaptation"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:ID="ProactivAdaptation">
    <rdfs:subClassOf OWL:Adaptation/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="OperationToAdapt">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
    <rdfs:domain rdf:resource="#ProactivAdaptation"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:ID="ReactivAdaptation">
    <rdfs:subClassOf OWL:Adaptation/>
  </owl:Class>

  <owl:Class rdf:ID="ProactivClient">
    <rdfs:subClassOf OWL:ProactivAdaptation/>
  </owl:Class>

  <owl:Class rdf:ID="ProactivServer">
    <rdfs:subClassOf OWL:ProactivAdaptation/>
  </owl:Class>

</rdf:RDF>
```

Bibliographie

[1] Velocity, the Apache Jakarta Project . `http ://jakarta.apache.org/velocity/index.html`, 1999-2005 The Apache Software Foundation.

[2] The protégé Ontology Editor and Knowledge Acquisition System. `http ://protege.stanford.edu/`, 2006 Stanford Medical Informatics.

[3] A.Costa, A. Gloria, P.Faraboschi, A.Pagni, and G.Rizzoto. Hardware Solutions for fuzzy Control. In *FUZZIEEE 95, IEEE International Conference on fuzzy systems*, volume 83, pages 422–434.

[4] H. A.Duran-Limon, G. S.Blair, T. Sivaharan, G. Samartzidis, and A. Friday. Resource Management for the Real-Time Support of an Embdded Publish/Subscribe System. Technical report, MPG-03-02.

[5] AMPROS. Adaptive Middleware Platform for PROactive reconfigurable Systems . `http ://www-inf.int-evry.fr/AMPROS/`.

[6] D. Ayed. *Déploiement sensible au contexte d'applications à base de composants*. PhD thesis, Institut National des Télécommunications, 2005.

[7] J. Bauer. Identification and Modeling of context for different information in Air Traffic. Master's thesis, Université d'eléctronique et d'informatique de Berlin, Mars 2003.

[8] C. Berthouzoz. A Model of Context Adapted to Domain-Independent Machine Translation. In *CONTEXT '99 : Proceedings of the Second International and Interdisciplinary Conference on Modeling and Using Context*, pages 54–66, London, UK, 1999. Springer-Verlag.

[9] I. Bokun and K. Zielinski. Active Badges–The Next Generation. *Linux Journal.Specialized Systems Consultants, Inc,Seattle, WA, USA*, 54 :24–26, 1998.

[10] D. Box. *Essential COM*. Adisson Wesley Professional, 1997.

[11] P. Brezillon and J. Pomerol. Contextual knowledge sharing and cooperation in intelligent assistant systems. In *Le Travail Humain*, volume 3, pages 223–246, Paris, 1999.

[12] S. Brockmans, R. Volz, A. Eberhart, and P. Loffler. Visual Modeling of OWL DL Ontologies Using UML. In *International Semantic Web Conference*, pages 189–213, 2004.

[13] G. Brose, A. Vogel, and K. Duddy. *Java programming with CORBA : advanced techniques for building distributed applications*. Wiley Computer, 2001.

[14] 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.

[15] P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In H.-W. Gellerson, editor, *Handheld and ubiqitous computing*, number 1707 in Lecture Notes in Computer Science, pages 304–307. Springer, September 1999.

[16] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The Fractal Component Model and its Support in Java. *Software Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12) :1257–1284, 2006.

[17] L. Capra. Mobile Computing Middleware for Context-Aware Applications. In *Proc. of the 24^{th} International Conference of Software Engineering (ICSE 2002), Doctoral Synposium*, Orlando, Florida, May 2002.

[18] L. Capra. *Reflexive Mobile Middleware for Context-Aware Applications*. PhD thesis, Université de Londre, 2003.

[19] L. Capra, W. Emmerich, and C. Mascolo. Middleware for Mobile Computing : Awareness vs. Transparency (Position Summary). In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 142–142, Schloss Elmau, Germany, MAY 2001.

[20] L. Capra, W. Emmerich, and C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. In *Proc. of REFLECTION 2001. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, volume 2192 of *lncs*, pages 126–133, Kyoto, Japan, September 2001.

[21] L. Capra, W. Emmerich, and C. Mascolo. A micro-economic approach to conflict resolution in mobile computing. In *SIGSOFT '02/FSE-10 : Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering*, pages 31–40, New York, NY, USA, 2002. ACM Press.

[22] L. Capra, W. Emmerich, and C. Mascolo. CARISMA : Context-Aware Reflective mIddleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, 29(10) :929–945, oct 2003.

[23] L. Capra, G. S.Blair, C. Mascolo, W. Emmeric, and P.Grace. Exploiting Reflection in Mobile Computing Middleware. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4) :34–44, OCT 2002.

[24] D. Chalmers, N. Dulay, and M. Sloman. Towards Reasoning About Context in the Presence of Uncertainty. In *Proceedings of Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004*, 2004.

[25] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[26] H. Chen, T. Finin, and A. Joshi. A Context Broker for Building Smart Meeting Rooms. In *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium*, pages 53–60, Stanford, California, 2004.

[27] H. Chen, T. Finin, and A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3) :197–207, May 2004.

[28] H. Chen, F. Perich, D. Chakraborty, T. Finin, and A. Joshi. Intelligent Agents Meet Semantic Web in a Smart Meeting Room. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004)*, New York City, NY, July 2004.

[29] M. Clarke, G. S. Blair, G. Coulson, and N. Parlavantzas. An Efficient Component Model for the Construction of Adaptive Middleware. *Lecture Notes in Computer Science*, 2218 :160–179, 2001.

[30] CLIPS. CLIPS Rule based language. `http ://www.siliconvalleyone.com/clips.htm`, 2005.

[31] COACH. Component Based Open Source Architecture for Distributed Telecom Application. `http ://coach.objectweb.org`, 2004.

[32] P.-C. DAVID. *Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation*. PhD thesis, École des mines de Nantes, 2005.

[33] N. Davies, K. Cheverst, K. Mitchell, and A. Efrat. Developing a context sensitive tour guide. In *Proceedings of First Workshop on Human-Computer Interaction for Mobile Devices, Glasgow, UK*, 1998.

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

[35] A. K. Dey. Supporting the construction of context-aware applications. In *Dagstuhl Seminar on Ubiquitous Computing*, September 2001.

[36] A. K. Dey, G. D. Abowd, and D. Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16 :97–166, 2001.

[37] A. K. Dey, D. Salber, G. D. Abowd, and M. Futakawa. The Conference Assistant : Combining Context-Awareness with Wearable Computing. In *3rd IEEE International Symposium on Wearable Computers*, pages 21–28, Whashington, DC,USA, 1999. IEEE Computer Society.

[38] J. Dowling and V. Cahill. The K-Component Architecture Meta-model for Self-Adaptive Software. In *REFLECTION '01 : Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, pages 81–88, London, UK, 2001. Springer-Verlag.

[39] P. Fahy and S. Clarke. CASS-Middleware dor Mobile Context-Aware Applications. In *2nd International Conference on Mobile Systems, Applications and Services (Mobisys 2004)*, Boston, MA, June 2004.

[40] J. Ferber. Computational Reflection in class based object-oriented languages. In *Conference on Object Oriented Programming Systems Languages and Applications*, pages 317–326, New Orleans, Louisiana, USA, October 1989.

[41] A. Fitzpatrick, G. Biegel, S. Clarke, and V. Cahill. Towards a Sentient Object model. In *Engineering Context-Aware Object-Oriented Systems and Environments (ECOOSE)*, Seattle, WA, USA, November, 2002.

[42] Fractal Home Page. `http ://fractal.objectweb.org`, 2004.

[43] E. Gamma, R. Helm, and R. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995. GAM e 95 :1 1.Ex.

[44] T. Gu, H. Pung, and D. Zhang. A Middleware for Building Context-Aware Mobile Services. In *IEEE Vehicular Technology Conference*, Milan, Italy, May 2004.

[45] T. Gu, H. K. Pung, and D. Q. Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1) :1–18, 2005.

[46] R. Guha. *Contexts : a formalization and some applications*. PhD thesis, Stanford, CA, USA, 1992.

[47] L. Hart, P. Emercy, B. Colomb, K. Raymond, S. Taraporewalla, D. Chang, Y. Ye, E. Kendall, and M. Dutra. OWL Full and UML 2.0 Compared. Technical report, 2004.

[48] A. Held, S. Buchholz, and A. Schill. Modeling of Context Information for Pervasive Computing Applications. 2002.

[49] K. Henricksen and J. Indulska. Modelling and Using Imperfect Context Information. In *PerCom Workshops*, pages 33–37, 2004.

[50] K. Henricksen and J. Indulska. Developing context-aware pervasive computing applications : Models and approach. *Journal of Pervasive and Mobile Computing*, volume 2(1) :pages 37–64, Elsevier, 2006.

[51] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. In *the First International Conference on Pervasive Computing*, pages 167–180, 2002.

[52] R. Hull, P. Neaves, and J. Bedford-Roberts. CyberDesk : The Use of Perception in Context-Aware Computing. In *Extended abstract in the Proceedings of the Workshop on Perceptual User Interfaces (PUI 97)*, pages 47–54, 1997.

[53] R. Hull, P. Neaves, and J. Bedford-Roberts. Towards Situated Computing. In *The First International Symposium on Wearable Computing (ISWC)*, pages 146–153, October 1997.

[54] J. Indulska, R. Robinson, A. Rakotonirainy, and K. Henricksen. Experiences in Using CC/PP in Context-Aware Systems. In *4th international Conference on Mobile Data Management*, pages 247–261, London, UK, 2003. Springer-Verlag.

[55] J2EE. Java 2 Platform Enterprise Edition Specification. `http ://java.sun.com/j2ee/`, 2002, version 1.4.

[56] Jena2 Inference support, http ://jena.sourceforge.net/inference/index.html, 2006.

[57] Jena2 support, http ://jena.sourceforge.net/, 2006.

[58] X. Jiang, N. Y. Chen, J. I. Hong, K. Wang, L. Takayama, and J. A. Landay. Siren : Context-aware Computing for Firefighting. In *Pervasive*, pages 87–105, 2004.

[59] JMS. Java Messaging Service Specification. `http ://java.sun.com/jms/`, 2002, version 1.1.

[60] J. I. Karen Henricksen and T. McFadden. Modelling context information with ORM. volume volume 3762 of Lecture Notes in Computer Science, pages pages 626–635, 2005.

[61] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Akcsit and S. Matsuoko, editors, *Proceeding European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Spring-Verlag, Berlin, Heidelberg, and New York, 1997.

[62] M. Kifer and G. Lausen. F-logic : a higher-order language for reasoning about objects, inheritance, and scheme. In *SIGMOD '89 : Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 134–146, New York, NY, USA, 1989. ACM Press.

[63] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran. Composite Capability/Preference Profile (CC/PP) : Structure and vocabularies 1.0. Technical report, W3C recommandation, 15 january 2004.

[64] G. Kortuem, Z. Segall, and M. Bauer. Context-Aware Adaptive Wearable Computers as Remote Interfaces to 'Intelligent' Environments. In *2nd IEE International Symposium on Wearable Computers*, pages 58–65, 1998.

[65] S. Krakowiak. Patrons et canevas pour l'intergiciel. Présentation, Quatriéme École d'été sur les intergiciels et sur la Construction d'Applications Réparties, Autrans, France, August 2003.

[66] S. Long, R. Kooper, G. D. Abowd, and C. G. Atkeson. Rapid Prototyping of Mobile Context-Aware Applications : The Cyberguide Case Study. In *Mobile Computing and Networking*, pages 97–107, 1996.

[67] C. Lopes and W. Hursch. Separation of Concerns. Technical report, College of ComputerScience Northeastern University, Boston, February 1995.

[68] H. Maass. Location-aware mobile applications based on directory services. In *3rd annual ACM/IEEE international conference on Mobile computing and networking*, pages 23–33, New York, NY, USA, 1997. ACM Press.

[69] F. Manola and E. Miller. RDF Primer. Technical report, W3C recommandation, 10 February 2004.

[70] J. McCarthy. Notes on Formalizing Contexts. In T. Kehler and S. Rosenschein, editors, *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 555–560, Los Altos, California, 1993. Morgan Kaufmann.

[71] J. McCarthy and S. Buvac. Formalizing Context (Expanded Notes). Technical report, Stanford, CA, USA, 1994.

[72] R. Meier and V. Cahill. Steam : Event-based middleware for wireless ad hoc networks. In *the 1st International Workshop on Distributed Event-Based Systems (DEBS '02)*, Vienna, Austria. July 2002.

[73] MicoCCM. MicoCCM home page. `http ://www.fpx.de/MicoCCM`, 2004.

[74] S. Microsystems. Java Remote Method Invocation specification. Revision 1.4.1. Technical report, Sun Microsystems, February 1997.

[75] G. K. Mostéfaoui, J. Pasquier-Rocha, and P. Brézillon. Context-Aware Computing : A Guide for the Pervasive Computing Community. In *ICPS : IEEE International Conference on Pervasive Services*, pages 39–48, 2004.

[76] C. national de la recherche scientifique. *Dictionnaire de la langue franÃ§aise du XIXeme et XXeme siècle*. Centre national de la recherche scientifique.

[77] OMG. Portable Interceptor. Omg document ptc/2001-03-04, Object Management Group, March 2001. Published Draft.

[78] OMG. The Common Object Request Broker - Architecture and Specifications. Revision 2.6. Omg document formal/o1-12-01, `http://www.omg/org`, December 2001.

[79] OMG. CORBA Components Version 3.0 an Adopted Specification of the Object Management Group. OMG Document formal/02-06-65, June 2002.

[80] OpenCCM. OpenCCM home page. `http ://openccm.objectweb.org`, 2006.

[81] M. Pagels. The DARPA Agent Markup Language. available at :`http ://www.daml.org/`, feb 2006.

[82] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12) :1053–1058, 1972.

[83] J. Pascoe. The stick-e note architecture : extending the interface beyond the user. In *IUI '97 : Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 261–264, New York, NY, USA, 1997. ACM Press.

[84] J. Pascoe, N. S. Ryan, and D. R. Morse. Human Computer Giraffe Interaction : HCI in the Field. In C. Johnson, editor, *Workshop on Human Computer Interaction with Mobile Devices*, IST Technical Report G98-1. University of Glasgow, May 1998.

[85] E. Pitoura and B. Bhargava. Building information systems for mobile environments. In *CIKM '94 : Proceedings of the third international conference on Information and knowledge management*, pages 371–378, New York, NY, USA, 1994. ACM Press.

[86] Quedo. Quedo home page. `http ://quedo.berlios.de`, 2004.

[87] R.Baldoni, C.Marchetti, and L.Verde. CORBA Request Portable Interceptors : Analysis and Applications. Concurrency and Practice and Experience. 2003. 15(6) : p.551-579, 2003.

[88] N. Ryan. ConteXtML : Exchanging contextual information between a mobile client and the fieldnote server. http ://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html, Accessed in 2006.

[89] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced Reality Fieldwork : the Context-aware Archaeological Assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.

[90] M. Samulowitz, F. Michahelles, and C. Linnhoff-Popien. CAPEUS : An Architecture for Context-Aware Selection and Execution of Services. In *Proceedings of the IFIP TC6 / WG6.1 Third International Working Conference on New Developments in Distributed Applications and Interoperable Systems*, pages 23–40, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[91] SART home page. `http ://www-poleia.lip6.fr/ brezil/SART/index.html`, 2006.

[92] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[93] B. Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5) :22–32, 1994.

[94] B. Schilit, M. Theimer, and B. Welch. Customizing Mobile Application. In *USENIX Symposium on Mobile and Location-independent Computing*, pages 129–138, Cambridge, MA, US, 1993.

[95] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. de Velde. Advanced Interaction in Context. In *HUC '99 : Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 89–101, London, UK, 1999. Springer-Verlag.

[96] Q. Z. Sheng and B. Benatallah. ContextUML : A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. In *The 4th International Conference on Mobile Business (ICMB'05), IEEE Computer Society. Sydney, Australia.*, July 11-13 2005.

[97] M. Sintek and S. Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *ISWC '02 : Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 364–378, London, UK, 2002. Springer-Verlag.

[98] B. Smith. *Procedural reflection in programming languages*. PhD thesis, Massachussetts Institute of Technimogy, USA, 1982.

[99] T. Stang and C. Linnhoff-Popoen. A Context Modeling Survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, September 2004*.

[100] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL : A Context Ontology Language to enable Contextual Interoperability. In J.-B. Stefani, I. Dameure, and D. Hagimon, editors, *LNCS 2893 : Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, volume 2893 of *Lecture Notes in Computer Science (LNCS)*, pages 236–247, Paris/France, November 2003. Springer Verlag.

[101] S. S.Yau and F. Karim. An Adaptive Middleware for Context-Sensitive Communications for Real-Time Applications in Ubiquitous Computing Environments. *Real-Time Systems*, pages 29–61, 2004.

[102] C. Szyperski. *Component software : Beyond object-oriented programming*. Second Edition ACM Press, component software series, Addisson-wesley, 2002.

[103] M. Uschold and M. Grüninger. Ontologies : principles, methods, and applications. *Knowledge Engineering Review*, 11(2) :93–155, 1996.

[104] M. Vadet. *Un modèle de services Logiciels pour la Spécialisation des Intergiciels à Composants*. PhD thesis, Laboratoire d'informatique fondamentale de Lille, Université des science et technologies de Lille, Lille, France, Novembre, 2004.

[105] P. Veríssimo, V. Cahill, A. Casimiro, K. Cheverst, and A. F. andJ Kaiser. CORTEX : Towards Supporting Autonomous and Cooperating Sentient Entities. In *Proceeding of European Wireless 2002*, pages 595–601, Florence, Italy, February 2002.

[106] M. Volk. The Automatic Translation of Idioms. Machine Translation vs. Translation Memory Systems. In : Nico Weber(ed) :Machine Translation : Theory, Application, and Evaluation. An assessment of the state of the art. St.Augustin : gardez-Verlag. 1998.

[107] W3C. OWL Web Ontology Language Use Cases and Requirements. `http ://www.w3.org/TR/webont-req/`, W3C Recommendation 10 February 2004.

[108] W3C. Resource Description Framework (RDF) : Concepts and Abstract Syntax. `http ://www.w3.org/TR/rdf-concepts/`, W3C Recommendation 10 February 2004.

[109] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology Based Context Modeling and Reasoning using OWL. In *PerCom Workshops*, pages 18–22, 2004.

[110] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. an Ontology Based Context Model in intelligent Environment. In *Communication networks and Distributed System modeling ans simulation Conference (CNDS 2004)*, pages 270–275, San Diego, Clalifornia, USA, January 2004.

[111] A. Ward, A. Jones, and A. Hopper. A New Location Technique for the Active Office. In *IEEE Personnel Communications*, October 1997.

[112] W. Xiaohang. The context gateway : a parvesive computing infrastructures for context aware services. Technical report, National university of singapore, 2003.

[113] S. S. Yau and F. Karim. Context-Sensitive Middleware for Real-Time Software in Ubiquitous Computing Environments. In *4th International Symposium on Object-Oriented Real Time Distributed Computing*, pages 163–170, 2001.

[114] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta. Reconfigurable Context-Sensitive Middleware for Pervasive Computing. *IEEE Pervasive Computing*, 1(3) :33–40, 2002.

[115] S. Zeghad. Configuration et reconfiguration de service en fonction du contexte d'exécution. Master's thesis, Institut National des Télécommunications, 2006.

Publications

Conférences internationales

1. **Nabiha Belhanafi-Behlouli**, Chantal Taconet, Guy Bernard - An architecture for supporting Developpment and Execution of Context-Aware Component Applications. IEEE International Conference on Pervasive Service 2006 (ICPS06)-Lyon, France. June 26-29, 2006, pp.57-66.
2. **Nabiha Belhanafi**, Chantal Taconet, Guy Bernard - Mécanismes de réactivité au contexte dans un intergiciel orienté composant. in Proc. New Technologies for Distributed Systems (NOTERE 2005). Université du Québec en Outaouais, Canada. 29 August-01 Semptember 2005. pp.41-49.
3. Dhouha Ayed, **Nabiha Belhanafi**, Chantal Taconet, Guy Bernard - Deployment of Component-based Applications on Top of a Context-aware Middleware. in Proc. The IASTED International Conference on Software Engeneering (SE 2005). Innsbruck, Austria. February 15-17, 2005. pp 414-419.

Workshops

1. **Nabiha Belhanafi**, Chantal Taconet, Guy Bernard - CAMidO, A Context-AWare Middleware based on Ontology meta-model. in Proc. workshop on Context Awareness for proactive Systems. CAPS2006, Helsinki, Finland. June 16-17, 2005. pp 93-103.

Posters

1. **Nabiha Belhanafi-Behlouli**, Chantal Taconet, Guy Bernard - CAMidO : a Middleware for component-based context-aware application development. in The first Conferences located in Europe, EuroSys 2006, Leuven, Belgium. April 18-21,2006.

Autres

1. **Nabiha Belhanafi-Behlouli**, Chantal Taconet - CAMidO : un intergiciel pour la sensibilité au contexte. Ubimob 2006. Atelier d'étude du contexte. Paris. September 05, 2006.