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 :

Dhouha AYED

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

Déploiement sensible au contexte d'applications à base de composants

Soutenue le 18 novembre 2005 devant le jury composé de :

Rapporteurs :

Mme Yolande Berbers
Mme Isabelle Demeure

Examinateurs :

M. Noureddine Belkhatir
Mme Virgine Watine
M. Guy Bernard — Directeur de thèse
Mme Chantal Taconet — Encadrant

Thèse numéro : 05INT006

Remerciements

Mes remerciements s'adressent, en premier lieu, à Madame Chantal Taconet. J'ai eu beaucoup de chance de l'avoir comme encadrante. Je la remercie infiniment pour ses conseils précieux et sa patience. Elle a toujours été là pour m'écouter, m'encourager et m'indiquer la bonne voie. Ce fut un plaisir de travailler avec elle. Je voudrais exprimer ma profonde reconnaissance à Monsieur Guy Bernard pour avoir suivi et dirigé cette thèse.

Je tiens à remercier tous les membres du jury pour m'avoir fait l'honneur d'accepter de juger ce travail de thèse :

- Monsieur Noureddine Belkhatir, professeur à l'université Pierre Mendes-France, Grenoble II, président du jury.
- Madame Isabelle Demeure, professeur à l'ENST Paris et Madame Yolande Berbers, professeur à l'université catholique de Louvain, rapporteurs de thèse.
- Madame Virginie Watine, ingénieur responsable R&D à Thales Communications, examinateur.
- Monsieur Guy Bernard, professeur à l'INT, directeur de la thèse et examinateur.
- Madame Chantal Taconet, Maître de conférence à l'INT, encadrante et examinateur.

Je tiens à témoigner ma gratitude à Abdelkrim Beloued et Tomas Hagg qui m'ont aidé à réaliser une partie des travaux de cette thèse lors de leurs stages.

Je remercie tous les membres du département informatique de l'INT et la secrétaire Brigitte Houassine pour leur aide et leur soutien. Je pense particulièrement à Samir Tata et Denis Conan pour leurs relectures et leurs nombreux conseils.

J'exprime également ma reconnaissance à Nawel Sabri, Ronaldo Ramos, Celso Maciel, Slim Ben Atallah et Nizar Ben Youssef pour leurs idées et leurs conseils.

Je remercie chaleureusement les thésards de mon équipe Lydialle Chateigner, Nabiha Belhanafi et Nabil Kouci pour l'échange et les bons moments que nous avons passés ensemble.

Je ne sais comment remercier Abed Ellatif Samhat pour son amitié, son exemple, ses conseils précieux et sa présence aux moments difficiles ainsi que tous mes amis qui ont contribué de près ou de loin à cette thèse, je citerais particulièrement Linda Salahadin, Tijani Chahed, Nihel Chabrak et Salah Eddine Elayoubi.

Une seule ligne ne saurait transcrire ma gratitude envers mes parents, ma sœur et mon frère qui m'ont offert leur amour et leur soutien pendant toutes ces années d'études. Sans eux cette thèse n'aurait pas été possible.

Je remercie, enfin, très chaleureusement mon mari pour sa patience, son soutien et ses encouragements durant ces trois années de thèse. Il a toujours su m'apaiser durant les moments difficiles.

Résumé

Les technologies de communication et d'information permettent aujourd'hui de déployer une large gamme d'applications sur des terminaux mobiles tels que des PDAs et des téléphones portables. Le contexte d'exécution des applications dans un environnement mobile se caractérise par un changement constant dû à la variation de la localisation de l'utilisateur, de sa connexion réseau, du terminal qu'il utilise ainsi que d'autres paramètres de son environnement. Ces changements de contexte amènent l'utilisateur à effectuer plusieurs fois les tâches de déploiement d'une application (configuration, installation et désinstallation), ceci afin d'obtenir une application dont la configuration répond au mieux aux besoins du contexte. La difficulté de ces tâches de déploiement et leur fréquence nous amène à étudier le déploiement d'applications dans un environnement mobile et à chercher une solution à l'automatisation de l'adaptation du déploiement au contexte.

Cette thèse propose une infrastructure intitulée CADeComp pour l'adaptation au contexte du déploiement des applications à base de composants. CADeComp est conçu avec un modèle indépendant de la plate-forme qui est constitué d'un modèle de données et d'un modèle d'exécution. Le modèle de données décrit les méta-informations utilisées pour adapter le déploiement au contexte. Ces méta-informations décrivent le contexte de déploiement ainsi que les règles qui définissent les variations des paramètres de déploiement en fonction de ce contexte. Le modèle d'exécution spécifie les entités qui incarnent des mécanismes d'adaptation en s'appuyant sur des algorithmes qui utilisent ces méta-informations. Cette thèse propose une projection du modèle CADeComp pour le modèle CCM. CADeComp a été implémenté et évalué sur cette plate-forme.

Mots clés : déploiement, composants, adaptation au contexte, applications distribuées, CCM.

Abstract

The expansion of wireless communication and mobile hand-held devices allows the deployment of a broad range of applications on mobile terminals such as PDAs and mobile phones. Execution context of applications in mobile environments undergoes constant changes due to the variation of the user location, his network connection, the characteristics of his terminal and other parameters of his physical environment. These context changes lead the user to carry out several times many deployment tasks of the same application such as its configuration, installation and uninstallation, in order to obtain an application whose configuration satisfies the context requirements. The difficulty and the frequency of these deployment tasks lead us to study the application deployment in a mobile environment and to look for a solution for the automation of the deployment adaptation to the context.

This thesis proposes a platform for the deployment adaptation of component-based applications to the context, entitled CADeComp. CADeComp is conceived with a platform independent model which consists of a data model and an execution model. The data model describes méta-information used to adapt the deployment to the context. This méta-information describes the deployment context as well as the rules which define the variations of the deployment parameters according to this context. The execution model specifies the entities that incarnate adaptive mechanisms. It defines algorithms which use this the deployment méta-information. This thesis proposes a projection of CADeComp model on the CCM model. CADeComp was implemented and evaluated on this platform.

Key words : Deployment, components, context-awareness, distributed applications, CCM.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte de la thèse

Avec les progrès réalisés ces cinq dernières années dans le domaine des réseaux sans fil et de l'informatique mobile, les terminaux tels que les téléphones portables et les assistants personnels numériques (PDAs) sont devenus des plates-formes suffisamment évoluées pour y déployer une large gamme d'applications tant dans le monde des affaires que du divertissement : les applications permettant d'effectuer des opérations bancaires, les jeux et les services multimédias sur terminaux mobiles, sont des exemples d'applications fréquemment utilisées dans la vie de tous les jours.

Grâce à ces progrès, les terminaux mobiles sont devenus capables d'être inclus dans des systèmes distribués complexes. Ces systèmes ne peuvent pas être assimilés aux systèmes distribués fixes pour deux principales raisons. D'abord, les utilisateurs mobiles utilisent des terminaux qui ont une taille de plus en plus petite de manière à pouvoir les enfiler dans leurs poches. En conséquence, les capacités de ces terminaux restent limitées en terme de ressources comparées à celles des terminaux traditionnels : ces appareils sont dotés d'un processeur plus lent, d'une taille de mémoire restreinte et rarement d'une mémoire secondaire en configuration de base. De plus, contrairement aux systèmes distribués fixes caractérisés par un environnement stable (bande passante élevée et peu variable, localisation des utilisateurs fixe et ajout, suppression ou déplacement des hôtes peu fréquent), les systèmes distribués mobiles ont un caractère dynamique : variation constante de l'emplacement des utilisateurs, de l'environnement physique qui les entoure et de la qualité de la connexion.

Le caractère dynamique du contexte des systèmes mobiles a créé de nouveaux défis dans différents domaines informatiques. Ces défis consistent principalement à adapter les services aux changements constants dans l'environnement de l'utilisateur. Les adaptations peuvent par exemple être réalisées pour empêcher une discontinuité du service offert. En effet, dans certains cas, un changement de contexte comme une coupure de la connexion réseau, peuvent empêcher l'utilisation du service. Les adaptations permettent aussi d'améliorer l'utilisation d'un service de façon à mieux répondre aux besoins de l'utilisateur.

La réalisation de ce défi nécessite la révision et l'extension de plusieurs solutions informatiques utilisées dans le cadre de systèmes distribués fixes, le déploiement d'applications est l'un d'entre eux.

1.2 Problématique et motivations

Le déploiement logiciel représente un des domaines qui doivent être révisés et étendus pour trouver des solutions au support des applications dans un environnement mobile. L'idée d'étendre les solutions de déploiement existantes est motivée d'une part par la diversité des applications déployées par les utilisateurs sur leurs terminaux mobiles et d'autre part par la fréquence de plus en plus importante de ces déploiements.

Dans un environnement mobile, un utilisateur a besoin de déployer une même application sur un nombre de terminaux de plus en plus importants : sur son PDA à partir de sa voiture, sur son PC à partir de son bureau ou sur un ordinateur portable de l'étranger. Ces différents terminaux sont dotés d'environnements matériels et logiciels différents. Le processus de déploiement gagnerait à être adapté à chaque environnement de déploiement.

Les utilisateurs mobiles changent également de situation, de localisation ou d'environnement physique externe. Ces changements peuvent avoir pour effet de faire varier les besoins et les préférences de l'utilisateur. Sans déploiement sensible au contexte, les variations des contextes d'utilisation contraignent l'utilisateur à faire des installations répétitives avec des configurations différentes pour une même application.

La limitation des ressources des terminaux mobiles pose à son tour un problème au moment du déploiement. Par exemple, les utilisateurs ayant des terminaux non dotés de mémoires secondaires se voient obligés d'installer l'application dont ils ont besoin à chaque utilisation. En ce qui concerne les terminaux ayant un disque, leur taille très limitée ne permet pas toujours d'installer simultanément toutes les applications dont l'utilisateur a besoin. L'utilisateur doit donc désinstaller des applications pour en installer de nouvelles et réinstaller les anciennes lors de leur prochaine réutilisation.

Le changement constant du contexte des utilisateurs mobiles et la limitation des ressources des terminaux qu'ils utilisent amènent donc les utilisateurs à effectuer des tâches de déploiement répétitives (installations, configurations, désinstallations). Ces tâches peuvent être rudes et complexes en plus d'être consommatrices en temps pour l'utilisateur. Elles présentent donc un frein à l'utilisation des applications mobiles.

La difficulté de ces tâches de déploiement et leur fréquence nous ont amenés à étudier le déploiement d'applications dans un cadre mobile et à dégager les besoins imposés par ce cadre.

Les utilisateurs mobiles présentent les besoins de déploiement suivants :

– **déploiement automatique :** pour épargner les utilisateurs des tâches de déploiement répétitives, l'installation, la configuration et l'activation des différentes applications doivent être réalisées automatiquement par l'outil de déploiement. La seule tâche de l'utilisateur doit

être de choisir l'application à déployer, les autres tâches doivent s'exécuter d'une manière transparente, sans son intervention. Le but est de donner à l'utilisateur l'impression que l'application choisie est déjà installée sur son terminal et que son choix consiste juste à l'activer ;

- **déploiement adaptable :** l'utilisateur subit un changement constant du contexte d'utilisation des applications, l'outil de déploiement doit être sensible à ce contexte. Il doit prendre en compte les ressources du terminal utilisateur, l'activité de l'utilisateur, ses préférences et l'environnement physique qui l'entoure pour déployer des applications qui répondent aux besoins de l'utilisateur changeant en fonction du contexte ;
- **déploiement à la volée :** les applications doivent être déployées " just-in-time ", juste au moment de la demande de l'utilisateur et repliées automatiquement dès leur désactivation par l'utilisateur. En effet, le cumul d'applications sur le terminal utilisateur réduit l'espace mémoire du terminal et gêne l'exécution des applications nouvellement déployées ;
- **déploiement sans retard :** l'automatisation des tâches de déploiement et leur adaptation au contexte ne doivent pas rajouter des retards importants par rapport aux temps de déploiements classiques sans adaptation. Les surcoûts en temps rajoutés par l'adaptation ne doivent pas être perceptibles par les utilisateurs.

1.3 Objectifs de la thèse

L'objectif de cette thèse est d'étudier la sensibilité du processus de déploiement au contexte et de proposer un cadriciel de déploiement adaptable au contexte qui répond aux besoins que nous avons exprimés à la section 1.2.

Nous nous intéressons aux applications construites à base de composants étant donné leur structure flexible qui les rend plus appropriées à une utilisation dans un cadre mobile variable.

Le choix de proposer une approche de déploiement sensible au contexte a été motivé par la constatation de l'inexistence d'un outil de déploiement destiné à une utilisation dans un environnement mobile alors même que de nombreux outils permettant la prise en compte du contexte s'appliquent dans d'autres domaines.

La proposition de nouvelles approches générales pour la gestion des informations de contexte ne représente pas un but de cette thèse mais plutôt la prise en compte du contexte dans le cadre du déploiement.

1.4 Contributions

Les contributions de cette thèse sont les suivantes :

- **analyse du domaine** : nous présentons une analyse du domaine de déploiement des applications à base de composants en faisant une étude comparative des outils de déploiement offertes par différentes plates-formes de composants. Nous dégageons de cette étude un cadre générique pour le déploiement des applications à base de composants qui définit les

différentes tâches de déploiement devant être couvertes par un outil de déploiement ainsi que les méta-informations nécessaires à l'accomplissement de ces tâches ;

- **étude du déploiement dans un environnement mobile** : cette étude nous permet d'observer la sensibilité du déploiement au contexte et d'en dégager les paramètres qui varient en fonction du contexte ;
- **proposition d'un nouveau cadriciel intitulé CADeComp pour le déploiement des applications à base de composants** : l'apport de CADeComp par rapport aux autres outils de déploiement est sa capacité d'adaptation au contexte. Son avantage réside dans le fait qu'il est conçu avec un modèle indépendant de la plate-forme qui est constitué d'un modèle de données et d'un modèle d'exécution. Le modèle de données permet la description des méta-informations utilisées pour adapter le déploiement au contexte. Ces méta-informations décrivent le contexte de déploiement ainsi que les règles définissant les variations des paramètres de déploiement en fonction de ce contexte. Le modèle d'exécution spécifie les entités qui incarnent des mécanismes d'adaptation en s'appuyant sur des algorithmes qui utilisent ces méta-informations ;
- **projection du modèle de CADeComp sur CCM (Corba Component Model)** : nous définissons des extensions des modèles de données et d'exécution de CADeComp pour qu'ils puissent être supportés par la plate-forme CCM. Nous définissons également des règles de transformation du modèle de données en un langage XML pour la description des méta-informations permettant l'adaptation du déploiement des applications CCM au contexte ;
- **implantation et évaluation** : nous implémentons les différents éléments de l'architecture CADeComp au-dessus de OpenCCM. Pour bénéficier d'une utilisation facile, CADeComp offre un éditeur pour les méta-informations nécessaires à l'adaptation du déploiement au contexte. Cet éditeur est doté d'un vérificateur de règles d'adaptation. Nous évaluons CADeComp en comparant ses temps de déploiement aux temps de déploiement sans adaptation au contexte.

1.5 Organisation du document

Ce document comprend trois parties principales : la présentation du contexte scientifique, la description du cadriciel CADeComp que nous proposons pour adapter le déploiement au contexte. Enfin, l'implémentation et l'évaluation de CADeComp.

Contexte scientifique. La première partie introduit le contexte dans lequel nous nous plaçons. Les chapitres 2 et 3 présentent les deux composantes de base de notre travail : les différents aspects d'adaptation au contexte et les concepts de déploiement. Les deux chapitres proposent une étude de l'état de l'art d'adaptation au contexte et de solutions de déploiement. Le chapitre 2 introduit les notions de base sur les mécanismes et les étapes de l'adaptation au contexte. Le chapitre 3 présente une étude comparative d'outils de déploiement proposés dans le cadre des modèles à composants et définit un cadre générique pour le déploiement des applications à base de composants.

Proposition. Dans la deuxième partie, nous détaillons la proposition CADeComp. Le chapitre 4 introduit cette solution par une étude de la sensibilité du processus de déploiement au contexte

à travers deux scénarios et la présentation des principes fondamentaux de CADeComp. Les chapitres 5 et 6 présentent le modèle de CADeComp indépendant de la plate-forme. Le chapitre 5 détaille le modèle de données et le chapitre 6 détaille le modèle d'exécution.

Modèle spécifique à la plate-forme CCM et implémentation. La troisième partie est constituée du chapitre 7 qui décrit la projection du modèle PIM de CADeComp sur la plate-forme CCM (PSM pour CCM) et détaille l'implémentation de ce PSM. Le chapitre se termine par une évaluation du modèle et de cette implémentation.

Le document se termine par le chapitre 8 qui présente une synthèse de l'approche ainsi que des perspectives sur les environnements de déploiement sensibles au contexte.

Première partie

Contexte scientifique

Chapitre 2

Adaptation au contexte

L'étude de l'adaptation du déploiement au contexte nous amène à étudier les travaux de recherche qui se sont intéressés à l'adaptation des applications au contexte afin d'en dégager les concepts essentiels.

Dans ce chapitre, nous définissons le contexte et nous citons ses caractéristiques (section 2.1). Ensuite, nous présentons les mécanismes utilisés dans les différentes phases de prise en compte du contexte à savoir l'acquisition des informations de contexte, leur modélisation et l'adaptation au contexte (section 2.2). Enfin, nous dressons un état de l'art de travaux de recherche réalisés sur l'adaptation au contexte (section 2.3). Nous montrons dans la conclusion l'importance des intergiciels sensibles au contexte dans la simplification du développement des applications adaptables (section 2.4).

2.1 Caractéristiques du contexte d'un logiciel

Nous commençons par définir un contexte logiciel par la suite nous présentons les caractéristiques des informations de contexte.

2.1.1 Définition du contexte

Nous avons trouvé dans les articles de recherche plusieurs définitions du terme " contexte logiciel ". Certains auteurs le définissent comme étant la situation de l'utilisateur ou son environnement [Bro96, FF98], d'autres le considèrent plutôt comme l'environnement de l'application [WH97, RCDD98]. [SW94, DAW98, Pas98] définissent le contexte comme un environnement d'exécution qui subit un changement constant. Cet environnement se divise en trois catégories :

- l'environnement matériel qui représente les ressources du matériel comme la bande passante de la connexion réseau, la disponibilité de la mémoire et la charge du processeur ;
- l'environnement de l'utilisateur qui représente la situation de l'utilisateur comme son emplacement, les personnes qui l'entourent, sa situation sociale ou son état émotionnel ;

– l'environnement physique tels que la lumière ou le bruit.

Dey [DAS01] a défini le contexte comme l'information pouvant être utilisée pour caractériser l'état de plusieurs entités (personne, endroit, objet) qui sont considérées comme pertinentes pour l'utilisateur, l'application et l'interaction entre l'utilisateur et l'application. Dey a étudié les définitions des autres auteurs, que nous venons de citer, et a fait une synthèse en définissant le contexte comme étant tout élément pouvant influencer le comportement d'une application. Nous utilisons cette définition pour le reste du document.

2.1.2 Caractéristiques des informations de contexte

Les informations de contexte ont des caractéristiques variées. Elles proviennent de sources variées, ce qui amène à une large hétérogénéité en terme de modélisation, de traitement et de qualité. Elles peuvent être imparfaites, ambigües et parfois même incohérentes ou incomplètes. Ceci est dû au fait que ces informations sont généralement collectées à partir de capteurs physiques (ou sondes) qui peuvent être l'origine d'une haute dynamicité, des erreurs de capture et d'un bruitage des informations de contexte. La nature dynamique de ces informations rend leur gestion difficile parce que la description de leur état devient rapidement obsolète. Cette dynamicité est d'autant plus difficile à gérer lorsque les sources de contexte sont distribuées et que les contextes nécessitent un traitement supplémentaire tels que leur abstracion. Ces facteurs créent de larges délais entre la production et l'utilisation des informations de contextes.

La complexité des informations de contexte nécessite des moyens de modélisation et d'interprétation spécifiques ainsi que des mécanismes d'adaptation qui permettent de prendre en compte leur nature. Nous définissons dans la section suivante intitulée " l'adaptation au contexte " les mécanismes utilisés pour collecter les informations de contexte, la modélisation de ces informations ainsi que les mécanismes utilisés pour adapter une application au contexte.

2.2 L'adaptation au contexte

Les termes informatiques "sensible au contexte", "adaptable au contexte" ou "qui prend en compte le contexte ", proviennent tous du terme anglais " context-aware computing " qui a été introduit la première fois par Schilit et al en 1994 [SW94]. Schilit a défini ce terme comme étant un logiciel qui s'adapte en fonction de sa localisation d'utilisation, de l'ensemble des objets et des personnes qui l'entourent ainsi que de la variation de ces objets à travers le temps. Une autre définition a été donnée par Dey [DAS01] qui définit un système comme sensible au contexte s'il utilise le contexte pour offrir des informations ou des services pertinents pour l'utilisateur.

Toute adaptation au contexte nécessite d'abord l'acquisition des informations de contexte qui seront traitées et analysées afin de réaliser l'adaptation. Nous présentons dans cette section les différents mécanismes utilisés pour les acquérir, les interpréter, les modéliser et adapter une application aux changements du contexte.

2.2.1 Acquisition des informations de contexte

Diverses méthodes sont utilisées pour collecter les informations de contexte. Mostéfaoui [MPRB04] distingue trois types de contextes du point de vue de leur acquisition :

- **le contexte capturé** : ce type de contexte est acquis par le moyen de capteurs matériels ou logiciels appelés aussi " sondes " telles que les capteurs de température, de pression, de niveau d'éclairage et de bruit. Les informations de contextes capturées les plus fréquemment citées dans les travaux de recherche sont la localisation des utilisateurs ou des objets et les informations sur le réseau. La localisation à l'extérieur des bâtiments est généralement capturée à l'aide d'un système GPS alors que la capture de la bande passante du réseau nécessite généralement l'utilisation de systèmes spécifiques tels que Odyssey [NSN+97] et le système proposé par Andersen [ABC+00] ;
- **le contexte explicitement fourni** : par exemple, les préférences de l'utilisateur sont explicitement communiquées par l'utilisateur à l'application ;
- **Le contexte dérivé ou interprété** : contrairement au contexte capturé et au contexte explicitement fourni qui représentent des informations de contexte de bas niveau (qui proviennent directement de leur source), le contexte dérivé représente un contexte de haut niveau qui peut être calculé à la volée à partir d'autres informations de contexte. Par exemple, le pays où se trouve l'utilisateur est un contexte de haut niveau déduit à partir du contexte de bas niveau qui représente ses coordonnées collectées à partir d'un GPS.

Afin de pouvoir utiliser les informations de contexte, l'application doit avoir un mécanisme qui lui délivre ces informations. Il existe deux méthodes permettant de délivrer les informations de contexte à l'application, la première est basée sur une approche où l'application est conduite par les capteurs et la deuxième est basée sur la séparation entre l'acquisition du contexte et l'application [DAS01].

Approche conduite par les capteurs

L'approche conduite par les capteurs relie les pilotes des capteurs directement à l'application, ce qui oblige les développeurs de l'application à écrire le code se rapportant aux capteurs en utilisant les protocoles dictés par ces derniers. Cette approche a l'inconvénient de mélanger les détails de bas niveau de l'acquisition du contexte avec la sémantique de l'application. De plus, elle rend l'application très dépendante des capteurs et incapable d'interagir avec d'autres capteurs pour une même information de contexte.

Approche de séparation entre le contexte et l'application

L'approche de séparation entre le contexte et l'application permet de cacher à l'application les détails se rapportant aux capteurs. Un intergiciel générique se charge d'acquérir et stocker les informations de contexte. L'application interagit avec cet intergiciel selon deux modes :

- le mode par interrogation qui consiste à envoyer une requête vers l'intergiciel pour récupérer la valeur du contexte ;

– le mode par notification qui consiste à s'inscrire auprès de l'intergiciel pour être notifiée par ce dernier de la valeur du contexte.

Le mode requête est plutôt utile dans le cas où l'application utilise l'information de contexte une seule fois et suppose que l'application est proactive, c'est à dire qu'elle 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 établir des conditions qui précisent quand elle doit être notifiée.

La séparation de l'acquisition des informations de contexte de leur utilisation présente un grand intérêt puisqu'elle permet la réutilisation des mécanismes d'acquisition par plusieurs applications, ainsi que l'utilisation de plusieurs types de capteurs par une même application.

2.2.2 Modélisation des informations de contexte

Les caractéristiques variées des informations de contexte comme leur hétérogénéité, leur dynamicité et leur imperfection (voir section 2.1.2) nécessitent des modèles abstraits de description de contexte. Plusieurs travaux ont défini des modèles de contexte. Nous en citons quelques uns dans ce qui suit.

paires de clé-valeur : un des premiers travaux sur la modélisation du contexte est celui de Schilit, Theimer et Gallois [STW93]. L'information de contexte est modélisée par une paire clé-valeur. Par exemple la liste des noms des occupants d'une salle est la suivante *OCCUPANTS = adams :schilit :theimer :weiser :welch*.

Schmidt et al.[SAT+99] fournissent un modèle de traitement du contexte basé sur des couches dans lesquels la sortie des capteurs est transformée en un ou plusieurs signaux. Ces signaux subissent un traitement pour former une description abstraite du contexte comportant un ensemble de valeurs, chacune liée à une mesure de certitude qui estime sa précision.

Modélisation orientée objet : Henriksen et al. [HIR02] ont proposé un ensemble de concepts de modélisation basés sur une approche orientée objet. Dans cette approche, les informations de contexte sont regroupées en un ensemble d'entités. Chaque entité représente un objet conceptuel ou physique tel qu'une personne, un dispositif ou un réseau. Les propriétés des entités tel que le nom d'une personne sont représentées par des attributs. Les entités sont liées à leurs attributs à travers des associations.

Modèle d'objets sensibles : appelé " Sentient Object " [HHSW99], ce modèle a proposé une manière plus formelle pour la présentation du contexte. Le contexte est décrit en utilisant un langage basé sur le modèle entité-association et l'information de contexte est stockée au moment de l'exécution dans une base de données relationnelle.

Modélisation par une ontologie : une ontologie est une spécification explicite d'une conceptualisation. Dans une ontologie, les connaissances d'un certain domaine sont représentées formellement comme un ensemble d'objets ayant des relations entre eux. La modélisation du contexte par une ontologie a un réel intérêt ; elle permet le partage des informations de contexte dans un système distribué et avec une sémantique bien définie elle permet l'utilisation d'agents intelligents pour faire un raisonnement sur le contexte. Une ontologie

nécessite un langage de représentation basé sur un modèle sémantique. Dans les dernières années, plusieurs langages d'ontologies basés sur les technologies web ont été proposés. Nous pouvons citer parmi eux OWL(Ontology Web Language) [BHH+]. Ces langages sont utilisés pour la modélisation du contexte [PB05].

La modélisation des informations de contexte est primordiale pour leur gestion à savoir leur transport, leur échange et leur stockage mais vu la diversité des informations de contexte, cette modélisation est étroitement liée au domaine de leur utilisation.

2.2.3 Mécanismes d'adaptation

Il existe plusieurs mécanismes qui permettent aux applications d'être adaptables et réagir ainsi à des changements de contexte. Les mécanismes les plus connus sont les suivants [YTL04, AC03].

Réflexivité

La réflexivité représente la capacité d'un système à s'observer et agir sur lui même [BGW93]. Un système réflexif doit être capable de manipuler des données représentant des informations sur son état durant son exécution. Cette manipulation comporte deux aspects : l'introspection et l'intercession. L'introspection est la capacité d'un système à observer et raisonner sur son propre état et l'intercession est la capacité d'un programme à modifier son propre état d'exécution. Un tel mécanisme est appelé " réification ".

Un système réflexif est composé de deux niveaux : le niveau de base qui comporte les données et le code fonctionnel du système (le "quoi" du système) et le méta-niveau qui comporte les données et le code non fonctionnel du système (le "comment" du système). Le méta-niveau permet au système de s'observer et d'agir sur lui-même pour s'adapter.

Programmation par aspects

Le paradigme de programmation orientée aspect permet de développer des applications en séparant leur code métier de leur code technique (non fonctionnel) selon le principe de séparation des préoccupations [Kic96]. Cette séparation permet de structurer l'application en modules indépendants représentant différents aspects :

- un noyau qui représente le cœur fonctionnel de l'application (le " quoi " de l'application) ;
- plusieurs aspects qui représentent des propriétés transversales au noyau (le " comment " de l'application).

La construction d'une application en utilisant ces différents modules nécessite une étape d'intégration du noyau avec les différents aspects. Cette intégration se traduit par l'établissement d'une jonction qui se situe au niveau d'un ensemble de points du flot d'exécution du noyau, appelés points de jonction.

A l'origine, le paradigme de la programmation par aspect a été mis en place afin d'élargir la possibilité de réutilisation du code, puisque la séparation du code correspondant aux propriétés de l'infrastructure du code métier rend ce code indépendant de l'infrastructure utilisée. Ce paradigme peut aussi être utilisé comme technique d'adaptation. Dans ce cas, si nous voulons intégrer un procédé d'adaptation dans une application, ce procédé doit être considéré comme du code technique.

Adaptation par contrat

D'une manière générale, un contrat signifie une convention qui oblige une ou plusieurs personnes à faire ou ne pas faire quelque chose pour une ou plusieurs autres personnes.

Il existe deux types de contrats. Le premier permet la spécification de contraintes devant être vérifiées de sorte que l'application se comporte comme prévu [Mey92]. Le second type décrit l'adaptation [AF03, KC03]. La politique d'adaptation n'est pas codée en dur dans le procédé d'adaptation, mais elle est décrite dans un fichier appelé contrat d'adaptation. Celui-ci est interprété par un procédé d'adaptation et peut être chargé et déchargé au moment de l'exécution.

L'utilisation de moteurs d'inférence

Lorsque les informations de contexte sont représentées à l'aide d'une ontologie, l'intelligence d'un système sensible au contexte peut être implémentée à l'aide d'un moteur d'inférence qui raisonnera sur les informations de contexte. Un moteur d'inférences est implémenté par un ensemble de faits et de règles appliquées sur les informations de contexte collectées à partir des capteurs. Parmi les implémentations existantes de moteurs d'inférences, nous pouvons citer Flora 2 [CFJ03] et Jena 2 [JEN].

2.2.4 Synthèse

Nous avons vu dans cette section différents aspects qui permettent l'adaptation de l'application au contexte, à savoir les techniques de collecte des informations de contexte, leur modélisation ainsi que certains mécanismes d'adaptation. Nous verrons dans la section suivante la manière avec laquelle ces différents mécanismes sont utilisés par des intergiciels et applications sensibles au contexte.

2.3 Étude de systèmes adaptables

Nous présentons dans cette section des exemples d'applications sensibles au contexte qui ont été développées ainsi que différents intergiciels et cadres qui ont été conçus dans le but de faciliter la prise en compte des informations de contexte. Le but de cette section est d'étudier les différentes informations de contexte qui ont été utilisées par les systèmes adaptables existants, la

manière avec laquelle ces informations ont été manipulées ainsi que les mécanismes employés pour prendre en compte le contexte.

2.3.1 Applications sensibles au contexte

Le premier système sensible au contexte issu des travaux de recherche dans ce domaine est *Active Badge* qui a été conçu et développé entre 1989 et 1992 [WHFG92]. Depuis, plusieurs travaux de recherche ont été réalisés donnant lieu à des applications sensibles à des contextes variés dans plusieurs domaines. Nous en citons quelques exemples dans ce qui suit.

Le *Shopping Assistant guide* [ACK94] utilise la localisation de l'utilisateur pour le guider dans un magasin. Il lui offre des détails sur les articles du rayon où il se trouve et lui indique où se trouvent les produits soldés.

CyberGuide [AAH+97] offre des informations pour les touristes sur une carte interactive et leur propose des endroits à visiter selon leur emplacement courant et l'historique des endroits déjà visités.

Conference Assistant [DFSA99] utilise une variété de contextes pour guider les participants à une conférence. L'assistant examine le programme de la conférence, les sujets des présentations, la localisation de l'utilisateur et l'intérêt de l'utilisateur et lui suggère une présentation à laquelle assister. Lorsque l'utilisateur entre dans une salle de conférences, l'assistant lui affiche automatiquement tous les détails liés à la présentation comme son titre et le nom de celui qui la présente.

Adaptive GSM phone and PDA [SAT+99] a utilisé une plus grande variété de contextes comme l'activité de l'utilisateur, le niveau de luminosité, la pression atmosphérique et la proximité des personnes. Il permet à un utilisateur de PDA de changer la taille d'écriture en fonction de son activité (par exemple une grande taille lorsque l'utilisateur est en train de marcher et une petite lorsque l'utilisateur est en position stable). Il permet aussi de sélectionner des profils de téléphones portables en fonction du contexte. Le téléphone choisit automatiquement lors d'un appel de sonner, vibrer ou de rester silencieux selon son emplacement, c'est à dire dans un sac, sur une table, dans la main ou à l'extérieur.

Office Assistant [YS00] est un agent qui interagit avec les visiteurs d'un bureau dès leur entrée par la porte pour gérer l'agenda du possesseur du bureau. L'entrée des visiteurs est automatiquement détectée et suivie par une demande d'identité. Selon l'agenda et la disponibilité du possesseur du bureau un rendez-vous est automatiquement fixé.

Call Forwarding [WHFG92] est basé sur le système de localisation d'*Active Badge*, le réceptionniste utilise la localisation de l'utilisateur pour faire suivre les appels téléphoniques au téléphone le plus proche de l'utilisateur.

Toutes ces applications sont construites selon l'approche conduite par les capteurs, tous les mécanismes d'acquisition et d'adaptation au contexte sont inclus dans le code de l'application, ce qui ne permet pas la réutilisation de ces mécanismes par une autre application. Par ailleurs, les mécanismes utilisés par ces applications pour interagir avec le contexte sont spécifiques à une

technologie d'acquisition donnée et ne supportent pas des technologies hétérogènes. Plusieurs versions de méthodes d'interaction avec les capteurs doivent donc être développées pour interagir avec chaque type de technologie d'acquisition de contexte. Par exemple, il existe plusieurs technologies pour acquérir la localisation de l'utilisateur : le GPS à l'extérieur, les fréquences radio et l'infrarouge à l'intérieur. Il est important que l'application puisse utiliser facilement l'un ou l'autre des capteurs. L'approche conduite par les capteurs ne le permet pas. L'utilisation d'un intergiciel sensible au contexte permet de développer de manière plus souple les applications sensibles au contexte.

2.3.2 Intergiciels sensibles au contexte

Les développeurs de systèmes sensibles au contexte ont besoin de suivre des procédures uniformes et standards [YKW+02] pour construire des applications adaptables au contexte. Ces procédures peuvent être fournies par des cadres. Un cadre, cadriciel ou canevas logiciel (Framework) est un ensemble de classes abstraites collaborant entre elles pour faciliter la création de tout ou d'une partie d'un système logiciel. Un cadriciel fournit un guide architectural en partitionnant le domaine visé en classes abstraites et en définissant les responsabilités de chacune ainsi que les collaborations entre classes.

Plusieurs architectures et modèles d'intergiciels ont été proposés et développés pour offrir des cadres de développement standards et proposer des interfaces uniformes et réutilisables pour interagir avec le contexte. Nous allons présenter certains de ces travaux.

Le Context Toolkit

Le cadre le plus fréquemment cité qui offre une solution réutilisable pour l'acquisition de contexte est le "context toolkit" [DAS01, Dey01]. C'est un cadre générique qui permet le prototypage rapide des applications sensibles au contexte. Le "context toolkit" est inspiré des boites à outils des interfaces graphiques qui jouent le rôle de médiateur entre l'application et l'utilisateur. Le "context toolkit" fait la médiation entre l'application et l'information de contexte capturée. Il utilise pour cela un ensemble de composants abstraits "widgets, interpréteurs et agrégateur" qui permettent l'acquisition des informations de contexte à partir des capteurs et leur transformation en information de contexte de haut niveau.

Un widget est un composant logiciel qui permet d'accéder à l'information de contexte par une interface uniforme. Il cache la complexité des capteurs et fournit des modules réutilisables pour la capture du contexte. Les applications intéressées par une information de contexte spécifique peuvent souscrire aux widgets correspondants.

Le problème d'interprétation est résolu au moyen des interpréteurs. Un interpréteur est responsable de la transformation de la représentation du contexte en une forme utile à employer par l'application. Par exemple, un interpréteur a la capacité de convertir des coordonnées GPS en adresse (n°, rue, ville).

Un agrégateur est responsable du contexte d'une entité particulière (personne, endroit ou objet) et obtient un agrégat centralisé d'informations de plusieurs widgets distribués.

Le "context toolkit" offre aussi un ensemble de services communs qui peuvent être activés par plusieurs applications en réagissant à certains contextes pour éviter leur réimplantation par chaque application. Il offre également un ensemble de composants de recherche qui permettent de découvrir les différents widgets, interpréteurs, agrégateurs et services disponibles.

Le "context toolkit" n'offre pas de mécanismes permettant de raisonner sur le contexte tels qu'un moteur d'inférence. D'autres travaux [dI01, SAT+99] ont étudié les mécanismes de raisonnement sur le contexte, mais sans réaliser un modèle de programmation clair et sans adresser les défis de la mobilité. Le "context toolkit" n'offre pas non plus de mécanismes d'adaptation qui peuvent être utilisés par des applications pour réagir aux différents changements de contexte d'un utilisateur mobile.

CARISMA

CARISMA (Context-Aware Reflective mIddleware System for Mobile Applications) [CEM03] est un intergiciel personnalisable en fonction des besoins d'une application. Cette personnalisation est réalisée par l'intermédiaire de profils d'applications. Un profil d'application représente une liste d'associations entre un service que l'intergiciel propose, une politique d'utilisation et une configuration de contexte dans laquelle la politique peut être utilisée. Ce profil est ensuite passé à l'intergiciel pour qu'il puisse déterminer le comportement à adopter lors de l'exécution d'un service pour des conditions particulières du contexte. Dans le cadre de la réflexivité, les profils d'application définissent des méta-données qui déterminent le comportement du système réflexif. Ces méta-données sont accessibles via une API réflexive qui permet l'introspection et la modification des associations (service, politique, contexte) prédéfinies.

K-Components

K-Components [DC01] est un cadre qui utilise la réflexivité pour construire des logiciels adaptables. La logique de l'adaptation dans K-components est basée sur la spécification de contrats qui décrivent le comportement adaptatif du logiciel à l'aide du langage ACDL (Adaptation Contracts Declarative Language). L'adaptation se produit en réponse à des événements adaptatifs envoyés par les composants de l'application. Si une adaptation s'avère nécessaire, un composant peut être retiré du graphe qui représente la configuration du système et remplacé par un autre composant ayant la même interface.

RAM

RAM [DL02] est basé sur une approche qui sépare complètements les aspects fonctionnels et non fonctionnels d'une application d'une façon reliée à la programmation par aspects. En utilisant cette approche de séparation de préoccupations, seules les fonctions métiers font partie du code

de l'application, les fonctionnalités non-fonctionnelles sont représentées sous forme de services intergiciel. L'application est adaptée en cours d'exécution au moyen d'un moteur d'adaptation qui applique des politiques spécifiques à l'application et des politiques propres au système. Les politiques du système sont des politiques de bas niveau qui contiennent des règles d'adaptation du système réalisées d'une façon transparente par rapport à l'application. Les politiques de l'application sont des politiques de haut niveau qui contiennent des règles d'adaptation du système réalisées d'une façon non transparente par rapport à l'application.

ReMMoC

ReMMoc (Reflective Middleware for Mobile Computing) [GBS03] est un intergiciel configurable et reconfigurable qui a été conçu pour offrir une solution aux problèmes de l'hétérogénéité du réseau. Il est capable de communiquer avec tout service présent dans le réseau, quelle que soit l'infrastructure qui implémente ce service. La conception de ReMMoc se base sur un modèle réflexif de composants appelé OpenCOM [CBCP01] qui est construit au-dessus d'un sous-ensemble de fonctions COM.

ReMMoC est défini par un cadre de composants qui contient à son tour trois autres cadres de composants.

- Le cadre *liaison* assure l'intéropérabilité entre les divers services disponibles sur le réseau qui sont implémentés sur d'autres types d'intergiciels. Ce cadre est basé sur un mécanisme de greffe de différents types d'implémentations de liaisons ;
- le cadre *service de recherche* qui permet la découverte de services qui sont habituellement découverts par une diversité de protocoles de recherche de services. Ce cadre est également basé sur un mécanisme de greffe de différents protocoles de découverte de services ;
- le cadre *exécution* est responsable de l'exécution d'une application. Il est capable de créer des composants, de les supprimer ou bien de les assembler.

Grâce aux cadres de liaison, de recherche et d'exécution, un utilisateur mobile qui utilise un service sur un réseau et qui entre dans une zone couverte par un autre type de réseau offrant le même service peut utiliser ce service même si les protocoles diffèrent.

OpenORB

OpenORB [BCRP98] est un intergiciel réflexif qui a été conçu à l'université de Lancaster. Au moment du chargement, des composants appropriés sont sélectionnés pour former une instance de l'intergiciel. En utilisant la réflexion, des composants peuvent être modifiés ou chargés durant l'exécution. Chaque objet dans OpenORB est associé à un méta-espace qui peut être accédé à travers une des interfaces du méta-modèle. Une deuxième version d'OpenORB appelée OpenORBv2 [BCA+01] a été implémentée à l'aide d'OpenCOM et offre une interface conforme à la spécification CORBA.

Chisel

Chisel [KC03] est un cadre d'adaptation dynamique sensible au contexte reflectif basé sur la description de politiques d'adaptation. L'approche de Chisel consiste à décomposer les différents aspects d'un service en plusieurs comportement possibles. Chaque comportement sera utilisé dans un contexte donné. Le changement de comportement du service est conduit par un script basé sur des politiques d'adaptation accessibles à l'utilisateur. L'avantage de Chisel est qu'il permet de rajouter de nouveaux comportements non anticipés au moment de l'exécution du service. Cela est réalisé en définissant les différents comportements comme méta-types.

CORTEX

CORTEX (CO-operating Real-time senTient objects) a des capacités de configuration et de reconfiguration afin de répondre aux exigences imposées par des environnements ad hoc. Cette nouvelle architecture utilise, comme ReMMoc, les techniques de la réflexivité et des cadres de composants. L'implémentation de l'intergiciel est basée sur OpenCOM . CORTEX contient quatre cadres de composants :

- le cadre de composants *contexte* : il permet de collecter et gérer les informations de contexte. Il est basé sur le modèle des "objets sensibles" [FBCC02]. Les objets sensibles sont définis comme des entités intelligentes capables de consommer des événements depuis des sondes (capteurs) et de produire des événements réfléchis vers des déclencheurs (actuators). L'intelligence du cadre de composants *contexte* est due à sa capacité de transformation des événements en informations de contexte, 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* : il est basé sur un modèle d'événements, appelé STEAM [MC02] et permet la dissémination des événements dans un réseau ad hoc ;
- le cadre de composants *gestionnaire de ressources* : permet de répartir les tâches par rapport aux ressources disponibles ;
- le cadre de composants *service de recherche* : ce cadre de composants est le même que celui présenté dans ReMMoC, il donne une solution aux problèmes posés par l'hétérogénéité du réseau, puisqu'il permet de découvrir des services annoncés par différents protocoles de découverte de services.

CORTEX a les mêmes objectifs que CARISMA puisqu'il met la réflexivité au service d'applications orientées vers la connaissance de contexte. Néanmoins CORTEX est utilisé avec des applications à petite échelle.

Autres intergiciels

Plusieurs intergiciels ont également été réalisés pour offrir des solutions aux problèmes de l'hétérogénéité des technologies d'interaction avec le contexte. Parmi eux nous pouvons citer ceux qui supportent des technologies de localisation différentes comme Oracle iASWE [IAS00], Nexus [FKV00], Alternis [ALT], SignalSoft [SS000] et CellPoint [CEL00].

D'autres intergiciels ont été réalisés pour supporter les opérations de déconnexions et de partage des données pour les utilisateurs mobiles comme Coda [SKK+90], son successeur Odyssey [Sat96], Bayou [TTP+95] et Xmiddle [MCZE02]. Le but de ces intergiciels est de maximiser la disponibilité des données en offrant aux utilisateurs la possibilité d'accéder à des réplicas.

2.3.3 Synthèse

Nous avons étudié dans cette section deux classes de systèmes adaptables : les applications adaptables qui sont basées sur l'approche conduite par les capteurs et les intergiciels sensibles au contexte qui séparent entre l'application et le contexte. Les intergiciels sensibles au contexte que nous avons étudiés utilisent les mécanismes d'adaptation cités dans la section 2.2.3.

Tous les intergiciels sensibles au contexte présentent un ensemble de mécanismes réutilisables pour la collecte des informations de contexte, leur interprétation, leur analyse et leur stockage, et peuvent même offrir des méthodes d'adaptation basés sur différentes techniques tels que la réflexivité ou l'utilisation de moteurs d'inférence. Ces intergiciels jouent un rôle très important dans la diminution des temps de développement d'un logiciel. En effet, les logiciels placés au-dessus de ces intergiciels n'ont pas à redévelopper ces mécanismes.

2.4 Conclusion

Nous avons étudié dans ce chapitre les caractéristiques des informations de contexte et les mécanismes utilisés dans les différentes phases d'adaptation au contexte qui sont l'acquisition des informations de contexte, leur analyse et l'adaptation au contexte. L'analyse des informations de contexte nécessite leur modélisation d'une façon abstraite. Nous avons ensuite étudié différents systèmes sensibles au contexte existants qui utilisent ces mécanismes.

Les informations de contexte sont généralement collectées à partir de capteurs différents. Ces informations se caractérisent par leur hétérogénéité et leur imperfection. Ces informations peuvent nécessiter plusieurs traitements tels que leur analyse, leur interprétation et leur présentation abstraite avant d'être utilisées dans le processus d'adaptation. Il est préférable de séparer ces traitements du système à adapter afin de dissimuler la complexité de ces traitements et permettre leur réutilisation par d'autres systèmes : c'est le rôle des intergiciels sensibles au contexte. Ces intergiciels sont de différents types. Ils peuvent ne tenir que le rôle de collecteur d'informations de contexte comme ils peuvent intégrer plusieurs fonctionnalités de traitements de contexte voir même des mécanismes d'adaptation. Le placement d'applications au dessus de tels intergiciels raccourcit et facilite les différentes phases du cycle de développement de l'application, puisque l'application va directement réutiliser les mécanismes offerts par l'intergiciel sans avoir à les redévelopper.

Dans le cycle de vie d'un logiciel, la phase de déploiement vise à installer l'application et à la mettre à disposition des utilisateurs finaux. Ceci constitue le premier contact de l'application avec son environnement dans une situation contextuelle donnée. Il nous semble important que

le déploiement de l'application prenne en compte le contexte courant et y adapte l'application à déployer. Les applications, outils et intergiciels sensibles au contexte que nous avons étudiés permettent d'adapter le comportement d'une application particulière ou facilitent les différentes phases du cycle de vie du logiciel, mais n'adressent pas l'adaptation du déploiement des applications au contexte. Cependant, les différentes techniques utilisées par ces applications et les mécanismes réutilisables offerts par les intergiciels sensibles au contexte peuvent être utilisés pour l'adaptation du déploiement au contexte. Dans le chapitre 4, nous donnons une vue générale sur l'exploitation de ces mécanismes dans le cadre du déploiement sensible au contexte.

Chapitre 3

Déploiement d'applications

Le déploiement d'un logiciel regroupe toutes les activités qui suivent l'achèvement du processus de développement du logiciel et qui visent à le mettre à la disposition des utilisateurs. L'objectif de ce chapitre est de définir les différents concepts et activités du déploiement en parcourant les différents mécanismes de déploiement logiciel utilisés à ce jour.

Nous commençons par définir d'une manière générale ce que nous appelons "cycle de vie du déploiement" (section 3.1). Ensuite, nous présentons les mécanismes de déploiement des applications monolithiques. A travers cette présentation, nous exposons un état de l'art des outils de déploiement utilisés pour les applications monolithiques (section 3.2). Le reste du chapitre se focalise sur les concepts de déploiement des applications à base de composants. Il présente les notions de base du déploiement de composants en établissant un cadre de caractérisation des différents outils de déploiement (section 3.3), ensuite il étudie des outils de déploiement offerts par les plates-formes de composants (section 3.4) et les compare entre eux. Le chapitre se termine par une discussion sur les différentes technologies de déploiement.

3.1 Cycle de vie du déploiement

Le déploiement d'un logiciel a toujours été défini comme étant la simple installation d'un logiciel. Cette définition nous paraît simpliste et incomplète, nous préférons dire comme Hall [HHW99] que le déploiement représente un ensemble d'activités qui incluent la diffusion du logiciel, son installation, son activation, sa mise à jour, sa reconfiguration, sa désactivation, sa désinstallation et même son retrait de la vente. Toutes ces activités interagissent pour former " le cycle de vie du déploiement logiciel " [HHW99, CFH+98, LB03]. Nous définissons dans ce qui suit chacune des activités du cycle de vie du déploiement.

La diffusion du logiciel : cette activité suit l'étape de développement du logiciel. Elle inclut les étapes de paquetage et de mise à disposition du logiciel. Le résultat de cette activité est un paquetage pouvant être transféré sur un site cible de déploiement (par exemple à travers le réseau ou par un CD). Un paquetage, appelé aussi archive, contient les fichiers exécutables du

logiciel et un ou plusieurs descripteurs qui décrivent les caractéristiques du paquetage et pointent vers ses différents fichiers.

L'installation : l'installation représente l'activité de déploiement initiale qu'effectue traditionnellement l'utilisateur au niveau de son poste de travail. C'est une activité complexe qui consiste à configurer et assembler toutes les ressources nécessaires pour l'utilisation du logiciel. Le processus d'installation utilise le paquetage créé lors de l'activité de diffusion. Il doit étudier la description des caractéristiques du logiciel et examiner le site utilisateur cible pour déterminer la configuration qui s'adapte le mieux aux propriétés du site.

La désinstallation : la désinstallation doit défaire toutes les modifications introduites sur le poste de l'utilisateur par les activités du déploiement du logiciel. Au moment de la désinstallation, il faut faire attention de ne pas détruire les fichiers et librairies partagées en étudiant l'ensemble des applications sur le site utilisateur, leurs dépendances et leurs contraintes.

L'activation : elle consiste à lancer l'exécution du logiciel une fois qu'il est installé. Dans les cas les plus simples, l'activation consiste à exécuter une commande pour démarrer le logiciel.

La désactivation : elle consiste à mettre fin à l'exécution d'un logiciel. Cette activité est utilisée avant de réaliser d'autres activités de déploiement telles que la mise à jour du logiciel, sa désinstallation ou sa reconfiguration.

La mise à jour : l'activité de mise à jour modifie un logiciel déjà installé. Elle déploie une nouvelle version du logiciel proposée par le producteur.

La reconfiguration : l'activité de reconfiguration modifie un logiciel déjà installé mais en sélectionnant une configuration différente de la configuration existante. A la différence de la mise à jour, elle est à l'initiative de l'utilisateur et non du producteur, et elle est effectuée localement sur la machine sans faire appel à d'autres paquetages du producteur.

Fin de support : lorsqu'un producteur de logiciels ne supporte plus un logiciel, il le retire de la vente. Ce processus consiste à éliminer le paquetage du logiciel du site du producteur afin de le rendre non disponible pour un prochain déploiement. Cette activité n'implique pas la désinstallation du logiciel du site de l'utilisateur, mais peut nécessiter l'information de l'utilisateur de l'indisponibilité du logiciel.

Il existe plusieurs outils de déploiement qui couvrent toutes ou une partie de ces activités de déploiement. Le reste de ce chapitre porte sur ces outils. Nous étudions d'abord les outils de déploiement monolithiques. Puis, nous étudions les outils de déploiement à base de composants.

3.2 Déploiement des applications monolithiques

Une application monolithique est une application qui constitue une entité unique déployée sur un seul site. Nous présentons dans cette section différents outils permettant de déployer les applications monolithiques ainsi que les différents mécanismes qu'ils utilisent. Nous ne présentons pas les outils de déploiement un par un mais nous présentons deux classes principales de ces

outils : les outils d'installation destinés à fonctionner sur une plate-forme Windows et les gestionnaires de paquetages qui fonctionnenet sur une plate-forme unix. Ces deux classes d'outils sont basées sur une description de méta-informations que nous présentons à la fin de cette section.

3.2.1 Outils d'installation

Contrairement à ce qu'indique leur nom, les outils d'installation ne couvrent pas que l'activité d'installation des logiciels sur les sites des utilisateurs. Ils sont aussi utilisés par les producteurs d'applications pour créer des paquetages (activité de diffusion).

Les outils d'installation utilisent une description du logiciel fournie par le producteur. Cette description comporte des méta-informations qui décrivent le nom du produit et sa version, le type de la plate-forme d'installation, le nom du répertoire d'installation par défaut, les espaces disque et mémoire nécessaires, ainsi qu'une liste de fichiers qui constituent le logiciel.

Le système de paquetage des outils d'installation rassemble la distribution binaire du logiciel et ses fichiers de description dans un même paquetage. Il ajoute une procédure d'installation destinée à être exécutée sur le site de l'utilisateur. La procédure d'installation utilise les méta-informations attachées au paquetage. Elle effectue une séquence d'opérations telles que la vérification de l'espace disque requis, la demande d'informations à l'utilisateur (comme le répertoire d'installation), le dépaquetage du logiciel et la création des fichiers de configuration nécessaires.

Parmi les outils pouvant réaliser ces opérations, nous pouvons citer Net-Install [NI], OpenWeb [OW] et InstallShield [Cor]. Ces outils ne permettent que l'installation des applications monolithiques qui sont généralement destinées à fonctionner sur une seule plate-forme (essentiellement Windows Microsoft).

3.2.2 Gestionnaires de paquetages

Contrairement à ce qu'indique leur nom, les gestionnaires de paquetages ne couvrent pas que l'activité de diffusion du logiciel sur les sites des utilisateurs. Ils supportent les activités d'installation, de mise à jour et permettent de gérer les logiciels installés. Ils peuvent couvrir tout le cycle de vie du déploiement à l'exception de l'activation et de la désactivation. Parmi ces outils, nous pouvons citer *RPM* [ET96] de Linux RedHat et la commande *pkg* [PK92] de SUN Solaris.

Les gestionnaires de paquetages offrent un référentiel local qui représente une base de données stockant l'état (des méta-informations) de tous les paquetages installés. Il est possible d'envoyer des requêtes vers cette base pour avoir des informations sur les paquetages du site utilisateur.

3.2.3 Description des applications monolithiques

Tant pour les outils d'installation que pour les outils de gestion de paquetages, la description de méta-informations se rapportant aux applications est essentielle pour l'utilisation de processus de

déploiement génériques qui automatisent les différentes activités de déploiement. La définition d'un format standard pour la description de ces méta-informations devient donc nécessaire. Il existe plusieurs formats de description permettant la création de méta-informations qui décrivent les propriétés d'un logiciel. Parmi ces formats émergent OSD (Open Software Description), le standard proposé par le consortium W3C et créé par Microsoft et Marimba [VHPT], le format MIF (Management Information Format) créé par DMTF [Inc96, For95] et le format DSD [Hal99] proposé dans le cadre des recherches réalisées dans [HHVdHW97].

Un format de description de logiciels spécifie les informations suivantes [HHW98a] :

– description des artefacts du logiciel : les fichiers constituant physiquement le logiciel tels que les exécutables, les librairies, les données et les documents ;
– description des contraintes imposées par le logiciel : les contraintes sont décrites par des valeurs d'attributs qui doivent être vérifiées pour que le logiciel soit installé (exemple : taille mémoire du site cible supérieure à 24MO). Si la contrainte n'est pas vérifiée le logiciel ne sera pas installé ;
– description d'informations de dépendances du logiciel : ces informations expriment des dépendances envers d'autres systèmes ou valeurs d'attributs. A la différence d'une contrainte, lorsqu'une dépendance n'est pas vérifiée, le système peut être reconfiguré de manière à ce que le logiciel puisse être déployé. Par exemple, si l'installation d'un logiciel sur un site utilisateur nécessite un fichier et que le fichier n'existe pas sur ce site, il est téléchargé ;
– description d'activités : un format de description logiciel doit offrir un moyen de décrire des activités supplémentaires qui complètent le déploiement ;
– Description des propriétés de configuration du logiciel : elle permet d'attribuer des valeurs aux attributs configurables du logiciel.

En plus de la description du logiciel, un format de description de logiciel doit permettre de décrire le site utilisateur où le logiciel va être déployé [HHW98b]. La description du site utilisateur contient des informations sur l'état du site à un instant donné comme ses propriétés ainsi que la disponibilité de ses ressources. La description du site utilisateur est complémentaire à celle du logiciel. Elle permet lors d'une installation sur un site de vérifier des contraintes sur des propriétés tels que le système d'exploitation ou les ressources du site ou bien des dépendances à des sous-systèmes du site de l'utilisateur.

La description de méta-informations sur les logiciels et les sites de déploiement permet d'utiliser des processus de déploiement génériques. Ces processus interprètent les contraintes, les dépendances et les configurations possibles d'une application en prenant en compte les ressources disponibles sur le site utilisateur.

3.2.4 Synthèse

Nous avons présenté dans cette section des mécanismes qui permettent de gérer le déploiement des applications monolithiques et nous avons montré le rôle joué par la description des méta-informations de déploiement dans l'automatisation des activités de déploiement. Nous présentons dans la section suivante les mécanismes utilisés pour déployer des applications à base de composants et nous montrons que les mécanismes de déploiement monolithiques servent aussi dans le cadre du déploiement des applications à base de composants.

3.3 Déploiement des applications à base de composants

Un composant logiciel est une unité de composition dont les interfaces sont spécifiées d'une façon contractuelle. Il est sujet à une composition par des tierces parties [Szy02]. Chaque composant possède un certain nombre de ports et représente une unité de déploiement qui peut être déployée d'une façon isolée ou assemblée avec d'autres composants.

Une application à base de composants est une collection de composants logiciels indépendants interconnectés entre eux à travers leurs ports.

L'approche composant est apparue pour palier les défauts de l'approche objet dont l'objectif principal était d'améliorer la modélisation d'une application et d'optimiser la réutilisation du code produit. Mais l'intégration d'objets logiciels existants est restée difficile. En effet, l'assemblage des objets entre eux est opaque (non décrit). L'assemblage d'objets écrits dans des langages différents et destinés à être exécutés sur des plates-formes différentes est impossible. Les composants représentent des abstractions de type boîte noire dans lesquelles aucun détail sur l'implantation du composant n'est connu par l'utilisateur, seules l'interface et sa spécification sont connus. Les implantations des composants sont réutilisables sous forme binaire contrairement aux implantations des objets qui sont réutilisables au niveau langage.

En ce qui concerne notre travail, l'apport fondamental de l'approche composant est la capacité de déployer une application distribuée de manière automatique en utilisant un ensemble de méta-informations qui décrivent son assemblage. Les composants d'une même application peuvent être développés avec des langages différents et s'exécuter sur des plates-formes différentes.

Nous présentons dans cette section les notions de base de l'approche composant nécessaires à la compréhension des différentes notions de déploiement. Ensuite, nous détaillons les particularités du cycle de vie du déploiement d'une application à base de composants par rapport à une application monolithique.

3.3.1 Notions de base de l'approche composant

Cette section présente les notions de plate-forme de composant, type de composant, implantation d'un type de composant, instance de composant et propriétés de composants [CCM02] [EJB02] [MP00].

Plate-forme de composants

Une plate-forme de composants appelée aussi intergiciel orienté composant est une infrastructure qui accueille les composants et gère d'une façon transparente leur cycle de vie (création, supression, recherche) , leur distribution et les services non fonctionnels dont ils ont besoin. Par exemple, CCM [CCM02] définit un modèle de plate-forme pour les composants CORBA, J2EE [J2E03] est une plate-forme qui accueille des composants EJB [EJB02] et .NET [COM95] est une plate-forme pour les composants Microsoft.

Type de composant

Le type d'un composant ou sa structure représente sa définition abstraite. Elle consiste en trois éléments : ses interfaces, ses modes de coopération avec les autres composants et ses attributs.

Les interfaces d'un type de composant peuvent être de deux types : des interfaces fournies par le composant et des interfaces requises par le composant. La définition des interfaces requises présente un progrès par rapport à l'approche objet puisque dans un objet, une référence sur un autre objet utilisé est enfouie dans le code. La déclaration des interfaces requises au niveau du modèle abstrait du composant permet de gérer les connexions entre composants d'une façon plus simple avec la possibilité de substituer un composant par un autre.

Deux modes de coopération entre composants sont possibles : le mode synchrone avec l'invocation de méthode et le mode asynchrone avec l'envoi d'événements.

Les attributs d'un composant représentent des propriétés configurables permettant d'adapter le comportement d'une instance de composant à l'application qui l'utilise.

Implémentation de composant

L'implémentation d'un type de composant ou son implantation représente deux aspects : l'implémentation fonctionnelle et l'implémentation non fonctionnelle. L'implémentation fonctionnelle représente la logique de traitement du composant : son code métier. L'implémentation non fonctionnelle représente un code non lié au code métier du composant comme le code qui sert à la gestion des connexions entre composants, la persistance d'un composant ou ses besoins transactionnels. Un composant peut avoir une ou plusieurs implémentations.

Généralement, dans une approche composant, seul le code fonctionnel d'un composant est programmé, le reste est décrit sous forme de propriétés non fonctionnelles qui sont prises en charge par la plate-forme qui héberge le composant. Dans le reste de ce document, on considère que l'implémentation d'un composant représente son code fonctionnel.

Instance de composant

Une instance de composant est, au même titre qu'une instance d'objet, une entité qui s'exécute sur un système. Elle est instanciée à partir d'un type de composant donné et une implantation particulière de ce type et possède une référence unique.

Propriétés de composant

Suite à l'étude du type d'un composant et de son implantation, nous pouvons distinguer deux types de propriétés d'un composant :

– des propriétés non fonctionnelles qui sont des propriétés de qualité de service indépendantes du code métier du composant. Elles peuvent décrire des propriétés comme la persistance d'un composant, ses politiques de sécurité ou son caractère transactionnel ;
– des propriétés fonctionnelles du composant qui sont les attributs d'un composant.

L'approche composant introduit des spécificités dans le cycle de vie du déploiement que nous allons présenter dans ce qui suit.

3.3.2 Particularités du cycle de vie du déploiement

Nous détaillons dans cette section les particularités que présente le cycle de vie du déploiement d'une application à base de composants en précisant le rôle de ses acteurs.

La diffusion d'un composant : cette activité est réalisée par le producteur du composant qui produit un paquetage de composant. Un paquetage de composant est un paquetage logiciel spécial qui contient une description du type du composant, son implémentation ainsi qu'un ou plusieurs descripteurs qui décrivent les propriétés fonctionnelles et non fonctionnelles du composant ainsi que ses dépendances.

La diffusion d'un assemblage de composants : l'approche composant a amené à une nouvelle approche de construction d'applications qui consiste à décrire une application en termes d'instances de composants interconnectées. La description d'assemblage de composants est réalisée par l'**assembleur** de l'application. Ce dernier définit la relation entre les composants venant d'un ou plusieurs producteurs de composants logiciels en étudiant la compatibilité de leurs interfaces et construit ainsi une application. L'**assembleur** de l'application construit lors de cette activité un paquetage d'assemblage de composants contenant les différents paquetages des composants qui constituent l'application ainsi que les méta-informations qui décrivent cet assemblage.

L'installation d'une application : cette activité place les instances de composants d'une application sur un environnement où elles peuvent s'exécuter. Elle inclut le dépaquetage des différents paquetages de composants ou le paquetage d'assemblage de composants de l'application (installation des paquetages des composants), leur instanciation et leur connexion. L'installation d'une application est réalisée par un **déployeur**. Ce dernier suit les instructions d'assemblage définies par l'assembleur de l'application et prend en compte les dépendances de chaque composant ainsi que ses propriétés fonctionnelles et non fonctionnelles spécifiées par le producteur du composant. Le déployeur utilise des outils de déploiement offerts par la plate-forme de composants pour dépaqueter les composants, les instancier et les interconnecter. Les tâches du déployeur peuvent être partiellement ou totalement automatisées.

L'activation et la désactivation : une fois que l'application est installée, le déployeur peut l'activer. L'activation d'une application consiste à activer ses différents composants en démarrant leur exécution. De la même manière, la désactivation de l'application entraîne automatiquement la désactivation de tous ses composants.

Les définitions du reste des activités du cycle de vie de déploiement citées dans la section 3.1 tels que la désinstallation, la mise à jour, la reconfiguration et la fin du support restent les mêmes sauf qu'elles sont appliquées à chaque composant de l'application.

Nous étudions dans les sections qui suivent les différents mécanismes et méta-informations utilisés par différents outils de déploiement pour réaliser les étapes du cycle de vie de déploiement.

3.3.3 Cadre pour le déploiement

Cette section présente un cadre de caractérisation d'outils de déploiement d'applications à base de composant. Il définit les activités de déploiement devant être couvertes par un outil de déploiement ainsi que les méta-informations nécessitées par ces activités. Pour définir ce cadre, nous nous sommes basés sur l'étude de plusieurs outils de déploiement des applications à base de composants que nous présentons dans la section 3.4. Nous préférons présenter ce cadre avant l'étude de ces outils pour que cette étude soit mieux dirigée et plus claire.

Dans cette section, nous commençons par décrire les méta-informations nécessaires pour le déploiement des applications à base de composants, ensuite nous identifions les activités de déploiement devant être fournies par les interfaces de déploiement.

Méta-informations pour le déploiement

Le déploiement d'applications à base de composants s'appuie sur la description d'informations qui sont propres à chaque composant de l'application ainsi que des informations qui décrivent et caractérisent l'assemblage général de l'application. Nous exposons dans ce qui suit ces différents types d'informations.

Cas d'un composant

- **Informations générales sur le composant logiciel** : ces descriptions représentent des informations d'ordre général sur le paquetage du composant, comme son nom, sa version, sa licence ou des informations sur le producteur du composant.
- **Informations structurelles sur le composant logiciel** : ces informations définissent le type du composant (interfaces et attributs) ainsi que des informations sur les opérations permettant de créer le composant et de l'activer.
- **Informations sur les dépendances du composant logiciel** : ces informations spécifient la dépendance du type de composant à d'autres paquetages et ressources telles qu'une base de données ou un fichier.
- **Informations sur l'implémentation du composant logiciel** : ces informations décrivent une implémentation du composant. Si le composant a plusieurs implémentations, des informations spécifiques à chacune de ses implémentations doivent être fournies comme l'emplacement de l'implémentation (par exemple une URL à partir de laquelle on peut la

télécharger), une liste de dépendances nécessaires à l'utilisation de cette implémentation (des dépendances d'implémentation qui s'additionnent aux dépendances du type du composant), le langage de programmation utilisé, le compilateur utilisé pour créer l'implémentation et des informations sur les contraintes qu'impose l'implémentation sur la future plate-forme d'exécution.
- **Informations sur les propriétés non fonctionnelles du composant logiciel** : ces informations attribuent des valeurs aux propriétés de qualité de service du composant.
- **Informations sur les propriétés fonctionnelles du composant logiciel** : ces informations spécifient les valeurs que vont prendre les attributs de configuration du composant.

Cas d'une application

- **Modélisation du domaine de déploiement** : ces informations décrivent le domaine sur lequel l'application va être distribuée lors de son déploiement. Elles décrivent la structure du réseau en terme de noeuds et de liaisons entre eux en précisant les capacités matérielles et logicielles disponibles au niveau de chaque nœud (processeur, mémoire, SE, ORB, compilateurs, logiciels installés, etc.) et au niveau de chaque liaison (type de réseau, bande passante, etc.)
- **Description d'assemblage abstrait de composants** : ces informations permettent de décrire la structure de l'application en terme de types de composants qui la constituent et leurs connexions sans indication du nombre de ces types de composants ni leur instanciation.
- **Plan de déploiement de l'application** : ces informations décrivent un assemblage d'instances de composant en spécifiant les informations nécessaires au déploiement de chaque instance : la façon dont ils sont répartis, interconnectés et configurés. C'est à dire qu'il sert de guide pour le déploiement de l'application.

L'ensemble des méta-informations que nous venons de citer au niveau composant et au niveau application est utilisé par des APIs de déploiement pour réaliser des activités que nous allons détailler dans ce qui suit.

3.3.4 Interfaces de déploiement

Les interfaces de déploiement des applications à base de composants sont très liés au modèle abstrait des composants et à la plate-forme utilisée, ils fournissent tout ou partie des opérations suivantes :
- assignation d'un site cible pour chaque composant d'une application,
- installation des paquetages sur les sites sélectionnés,
- instanciation des composants sur les sites sélectionnés,
- connection des instances de composants,
- fourniture d'informations sur l'environnement cible,
- activation et désactivation de l'assemblage des composants,
- fourniture d'informations sur l'assemblage en cours d'exécution,
- désinstallation des paquetages de composants installés.

En se basant sur le cadre de déploiement que nous avons établi dans cette section, nous étudions les outils de déploiement offerts par différentes plates-formes de composants et nous les comparons.

3.4 Outils de déploiement

Plusieurs plates-formes de composants offrent leurs propres solutions de déploiement. Nous pouvons citer parmi eux les modèles de déploiement offerts par le modèle de composants de CORBA, CCM (Corba Component Model) [CCM02] de l'OMG, le modèle de composants EJB (Enterprise Java Beans) [EJB02] de Sun Microsystems, le modèle de composants .Net [COM95] de Microsoft, la spécification de déploiement et de configuration des applications à base de composants de l'OMG connue sous le nom D&C [OMG03] et le modèle de déploiement de OSGi défini par Open Service Gateway Initiative [ope03]. Nous présentons dans cette section chacun de ces modèles de déploiement.

3.4.1 Déploiement CCM

Le modèle de déploiement CCM est relié au modèle abstrait du composant CORBA. Nous commençons par présenter le modèle abstrait de composant ensuite nous détaillons le modèle de déploiement.

Modèle abstrait des composants CORBA

Un composant CORBA est défini par un ensemble de ports et d'attributs qui sont représentés dans la figure 3.1. Quatre types de ports sont définis :

- une facette qui représente une interface offerte de mode synchrone ;
- un réceptacle qui représente une interface requise de mode synchrone ;
- une source d'événements qui représente une interface requise de mode asynchrone ;
- un puit d'événement qui représente une interface offerte de mode asynchrone.

Pour chaque type de composant est défini un ou plusieurs types de maisons de composants. Une maison de composant gère les instances de composants en prenant en charge la création et la destruction de ces instances ainsi que leur recherche.

Les composants CORBA s'exécutent dans un conteneur qui leur offre un environnement d'exécution, les services système qui leur sont nécessaires et gère les différents aspects non fonctionnels dont ils ont besoin.

Modèle de déploiement

Le modèle de déploiement CCM définit deux types de paquetages : le paquetage de composant et le paquetage d'assemblage.

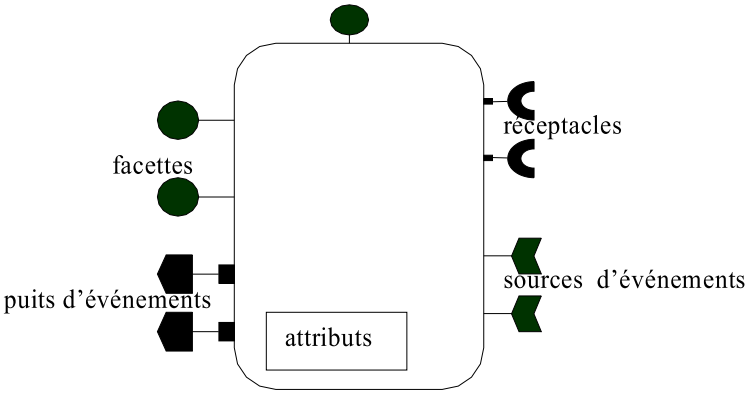


FIG. 3.1 – Modèle abstrait des composants CORBA

Le paquetage de composant est associé à un seul type de composant, il contient une ou plusieurs implantations du composant et trois descripteurs de méta-informations permettant de décrire le paquetage : le descripteur logiciel de composant, le descripteur de composant CORBA et le descripteur de propriétés du composant :

– le descripteur logiciel de composant contient des informations générales concernant un composant : son nom, des informations sur son producteur, sa licence, une référence sur le fichier IDL3 du composant qui décrit son type (son modèle abstrait) et des références sur les autres descripteurs de déploiement se rapportant au composant (le descripteur de composant CORBA et le descripteur de ses propriétés). Ce descripteur comporte aussi des informations sur les implémentations du composant, l'implémentation de ses maisons ainsi que leur point d'entrée ; pour chaque implémentation, il fournit une description générale, une référence sur le code de cette implémentation, le nom du compilateur utilisé et du langage de programmation et des informations de dépendance.
– le descripteur de composant CORBA décrit les informations structurelles et non fonctionnelles se rapportant au composant. Les informations structurelles décrivent les facettes, les réceptacles, les sources d'événements et les puits d'événements du composant ainsi que ceux des composants dont il hérite. Ce descripteur est défini pour chaque implémentation du composant ;
– le descripteur de propriétés du composant contient des valeurs pour les attributs de configuration du composant ;

Le paquetage d'assemblage contient un plan de déploiement qui décrit l'instanciation des composants et leurs interconnexions. Le plan de déploiement spécifie pour chaque instance de composant son emplacement, son implémentation et un descripteur de propriétés.

La figure 3.2 montre un diagramme de collaboration UML [Gro03] qui illustre les étapes de déploiement et l'interaction entre les interfaces de déploiement CCM. L'application de déploiement commence par installer les paquetages de composants sur les sites cibles en utilisant l'objet *ComponentInstallation* de chaque site. Ensuite, elle crée un objet *Assembly* en utilisant sa fabrique *AssemblyFactory*. L'objet `Assembly` démarre l'assemblage de l'application et coordonne le reste des étapes de déploiement. Il crée pour cela un serveur de composants à travers un activateur de serveurs qui crée à son tour une instance du conteneur du composant. *Assembly* envoie ensuite

une requête vers le conteneur pour qu'il crée les maisons des composants qu'il utilisera pour instancier les composants, puis il configure ces derniers et les connecte entre eux.

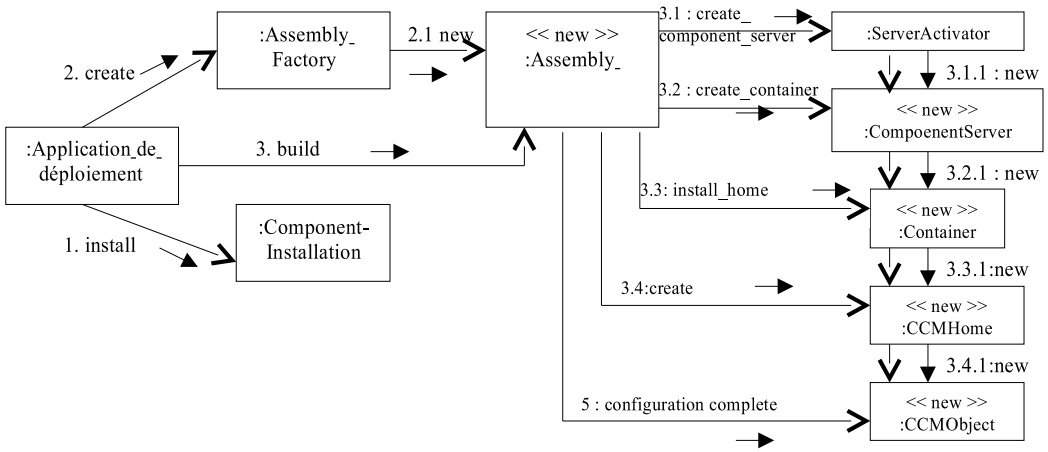


FIG. 3.2 – Diagramme de collaboration UML du déploiement CCM.

Le modèle de déploiement CCM est un modèle riche qui fournit l'outillage qu'il faut (objets de traitements et informations) pour déployer un assemblage de composant. Il est dommage que les étapes de déploiement (installation des paquetages, création des serveurs de composants, instanciation des conteneurs, création des maisons et instanciation des composants) soient exécutées en séquence sans l'intervention de l'application de déploiement. L'application de déploiement ne peut que construire ou défaire l'assemblage à travers le coordinateur d'assemblage. Il aurait été souhaitable qu'un ensemble d'opérations plus riche puisse exécuter les étapes de déploiement une par une. Cela permetterait de paramétrer leur exécution par des événements externes et ainsi de rendre le déploiement plus flexible.

3.4.2 Déploiement EJB

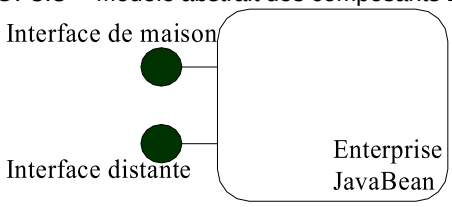
Le modèle de déploiement EJB est relié au modèle abstrait du composant EJB. Nous commençons par présenter le modèle abstrait de composant ensuite nous détaillons le modèle de déploiement.

Modèle abstrait des composants EJB

Un composant EJB ou un e-bean (voir figure 3.3) est un composant serveur représenté par une interface métier Java et une interface maison qui permet de gérer le cycle de vie du composant (sa création, sa destruction, sa recherche).

Les EJBs s'exécutent dans des conteneurs EJB qui hébergent les composants et gèrent tous leurs aspects non fonctionnels.

FIG. 3.3 – Modèle abstrait des composants EJB



Modèle de déploiement

Les composants EJB sont diffusés dans des paquetages qui contiennent des classes Java ou des références sur des classes qui implémentent les composants et leur maisons ainsi qu'un descripteur de déploiement. Il y a deux types d'informations dans le descripteur de déploiement :

- les informations se rapportant aux composants, et
- les informations se rapportant à l'assemblage des composants.

Les informations se rapportant au composant sont fournies par le producteur du composant EJB. Parmi ce type d'information, on trouve le nom du composant ainsi que celui de la classe l'implémentant, les interfaces qu'il offre, l'interface de sa maison et des informations non fonctionnelles.

Si un e-bean est utilisé lors d'un assemblage, l'assembleur de l'application complète le descripteur de déploiement en spécifiant la façon de lier les différents e-beans de l'application et définit les droits d'accès pour les différentes méthodes des e-beans.

Les composants EJB sont destinés à être déployés sur une plate-forme J2EE [J2E03]. Les interfaces de l'outil de déploiement J2EE ont été spécifiés dans la spécification " J2EE Deployment " [Sea03]. L'interface la plus importante dans cette spécification est l'interface *DeploymentManager*, elle permet de :

- configurer une application pour une utilisation spécifique. Cette configuration est réalisée par la mise à jour du descripteur de déploiement du paquetage de l'application. Cette mise à jour est réalisée par un outil qui utilise l'API de déploiement pour demander les informations de déploiement spécifiques au serveur et interagir avec le déployeur qui remplie les éléments du descripteur de déploiement qui nécessitent une mise à jour. Le résultat de cette opération est un paquetage qui contient le contenu du paquetage initial de l'application avec des classes d'introspection qui sont spécifiques au conteneur de la plate-forme J2EE ;
- distribuer une application sur une cible. Cela peut consister en un ou plusieurs serveurs J2EE en y installant des paquetages configurés ;
- activer et désactiver une application ;
- désinstaller une application ;
- réaliser la supervision des applications déployées. Les opérations d'activation, de désactivation, de distribution ou de repliement retournent un ProgressObject qui permet le suivi de l'exécution des activités de déploiement par l'outil de déploiement.

Dans le modèle de déploiement EJB/J2EE, il est à déplorer que le descripteur de déploiement ne permette pas de planifier le déploiement en décrivant la distribution des instances de composants sur des sites cibles.

3.4.3 Déploiement COM/COM+/.NET

COM (Common Object Model) et DCOM (Distributed COM) [COM95] sont les modèles de composants de Microsoft. Dans le cadre de COM, un composant est une entité binaire pour laquelle sont définis une interface et un mode d'interaction. L'implémentation d'un composant est distribuée avec une bibliothèque DLL. DCOM est une extension complémentaire à COM qui prend en compte l'aspect distribué d'une application.

La diffusion des composants se fait au travers de bibliothèques DLLs qui sont à déployer manuellement sur les machines cibles puisqu'aucun processus de déploiement n'est prévu dans COM. L'assemblage de composants est basé sur la possession d'une référence sur la bibliothèque qui implémente ce composant.

Le déploiement des applications COM/DCOM pose le problème des " conflits DLLs " [And00] selon lequel l'installation d'une application peut causer l'arrêt de fonctionnement d'autres applications déjà installées. Cet arrêt de fonctionnement est causé par le fait que les applications Windows utilisent des DLLs privées (propres à l'application) et des DLLs publiques utilisées par plusieurs applications. En installant une application, chaque DLL publique est automatiquement installée en écrasant une éventuelle version existante en cours d'utilisation par d'autres applications.

Pour résoudre ce problème de version qui amène aux conflits de DLLs et pour simplifier le déploiement des applications, Microsoft a conçu une nouvelle unité de déploiement .NET appelée " assemblage " et a introduit le concept " side by side " qui consiste à rendre possible l'exécution simultanée de plusieurs versions d'un même assemblage (donc d'une même DLL) sur la même machine [Ang00].

Un assemblage .NET ne représente pas un assemblage de composants mais un paquetage de composants. Ce paquetage est composé par du code (des DLLs), des fichiers de données et d'un descripteur de déploiement appelé manifeste. Ce descripteur contient les informations suivantes :

- le nom de l'assemblage,
- le numéro de version,
- des informations sur le langage utilisateur supporté par l'assemblage,
- une clé publique fournie par le producteur de l'assemblage,
- une liste de tous les fichiers de l'assemblage (code et données), et
- une liste de références statiques sur d'autres assemblages.

Il existe deux types d'assemblages : les assemblages privés et les assemblages publics qui sont partagés par plusieurs applications. Les assemblages publics doivent suivre des règles rigoureuses de nommage pour avoir des noms uniques et pouvoir ainsi s'exécuter simultanément. Ces noms uniques sont réalisés à l'aide de clés publiques contenant essentiellement un jeton du producteur, le nom de l'assemblage et sa version.

3.4.4 Déploiement OSGi

La plate-forme OSGi [ope03] est une plate-forme Java qui a été spécifiée dans le but de faciliter le déploiement et la gestion à distance des services utilisés dans un environnement local. Les unités de diffusion d'OSGi sont appelées des " bundles ".

Les " bundles " contiennent des services qui sont les composants à partir desquels les applications sont construites. Un bundle est une archive Java au format jar décrite par un fichier manifeste. Cette archive contient le code interprétable d'un composant, la classe de son activation et les ressources qui lui sont nécessaires. " bundle " est utilisé pour déployer un seul composant qui donne lieu à une instance unique au moment de l'exécution.

Un " bundle " peut importer ou exporter un code source sous forme de paquetage. Ce mécanisme permet aux bundles d'accéder ou de donner accès au code des interfaces des services de manière à ne pas intégrer ce code dans chaque " bundle ".

Dans le cadre de OSGi, les applications subissent de fréquentes évolutions. Ces applications sont construites à partir de " bundles " connectés à travers les services qui peuvent apparaître ou disparaître au cours de l'exécution de l'application.

Le manifeste OSGi décrit les informations suivantes :
- les informations générales comme le nom du " bundle ", sa description, sa version, l'URL de sa documentation, les coordonnées de son producteur, sa licence et sa catégorie ;
- l'emplacement des classes et des ressources du " bundle " ;
- les bibliothèques natives à charger ;
- les paquetages requis et les paquetages fournis ;
- les services requis et les services fournis ;
- le nom de la classe Activator ;
- l'URL à partir de laquelle le bundle sera mis à jour ;
- la liste d'environnements qui doivent être présents sur la plate-forme ;
- la liste de paquetages qui pourront être importés en cours d'exécution.

Une fois déployés, les composants contenus dans les " bundles " interopèrent suivant l'approche orientée service.

L'environnement d'exécution fournit des mécanismes permettant de réaliser des activités de déploiement qui incluent l'installation, l'activation, la déactivation, la mise à jour et le retrait des " bundles ". De plus, l'environnement d'exécution prend en charge la gestion des dépendances de code qui doivent être satisfaites après l'installation d'un " bundle " pour permettre de réaliser son activation. L'activation d'un " bundle " déclenche la création de l'instance du composant.

3.4.5 Déploiement D&C

D&C représente un modèle de déploiement d'applications à base de composants proposé par l'OMG qui est indépendant de la plate-forme d'exécution.

D&C spécifie trois modèles de données pour la description de méta-informations de déploiement : un modèle de données des composants, un modèle des sites cibles et un modèle de données d'exécution du déploiement :

– le modèle de données des composants permet de décrire la configuration d'un paquetage de composants en décrivant le type de composant et ses différentes implémentations. Un composant peut avoir une implémentation monolithique concrète contenue dans un artefact (fichier exécutable ou librairie) ou peut être implémenté d'une manière récursive par un assemblage : un ensemble de sous composants interconnectés entre eux. Dans les deux cas, plusieurs implémentations peuvent être supportées pour un même composant. A chaque implémentation est associé un ensemble de descriptions d'artefacts qui spécifient l'emplacement de l'implémentation, ses paramètres d'exécution, ses dépendances et les contraintes que l'implémentation impose sur le domaine de déploiement ;
– le modèle de sites cibles permet de décrire le domaine de déploiement cible en terme de noeuds, d'interconnexions entre les noeuds et éléments de routage ;
– le modèle de données de gestion d'exécution du déploiement représente un moyen de description d'un plan de déploiement qui décrit comment créer les instances de composants à partir des artefacts et comment les interconnecter, et où les instancier. Dans le plan de déploiement, tous les composants composites sont remplacés par leur boîte blanche (leurs sous composants) et des implémentations concrètes sont choisies pour chaque sous-composant.

La figure 3.4 montre le diagramme de classes du modèle de gestion d'exécution du déploiement proposé par D&C. Le déploiement d'une application est réalisé en deux phases. La première est la préparation du plan pour l'exécution avec l'opération *ExecutionManager.preparePlan()*. Cette opération a comme résultat un objet *ApplicationManager* qui peut mettre le plan en action à plusieurs reprises. La deuxième phase est l'activation de l'application qui est à son tour divisée en deux étapes : la première étape, *ApplicationManager.startLaunch()*, retourne des références aux ports offerts par l'application et la deuxième étape, *Application.finishLaunch()*, attribue des références aux ports utilisés par l'application.

Un objet *Application* permet l'activation d'une application, la navigation entre les différents ports de l'application et son introspection au moment de son exécution. *Application* est une classe abstraite spécialisée par *DomainApplication*, qui représente une application globale et *NodeApplication* qui représente une application qui s'exécute sur un seul noeud.

ExecutionManager divise l'application globale en sous-applications (composants et connexions) qui peuvent s'exécuter sur des nœuds différents. Le déploiement d'une sous-application sur un nœud est décrit (comme une application) avec un plan de déploiement. *ExecutionManager* crée des plans de déploiement pour les sous-applications et les offre au *NodeManager* responsable d'instancier les composants sur le nœud.

Les interfaces gérant le déploiement des sous-applications et des applications (*ApplicationManager* et *Application*) sont les mêmes mais pour des raisons de sémantique, elles sont préfixées par Domain ou Node.

En plus du modèle de gestion d'exécution de déploiement, D&C propose un modèle de gestion de composants qui permet de gérer les données se rapportant au composant au niveau d'un

référentiel (installation et recherche d'un paquetage de composant au niveau d'un référentiel) ainsi qu'un modèle de gestion de domaine cible de déploiement qui permet de récupérer des informations sur les ressources des sites cibles.

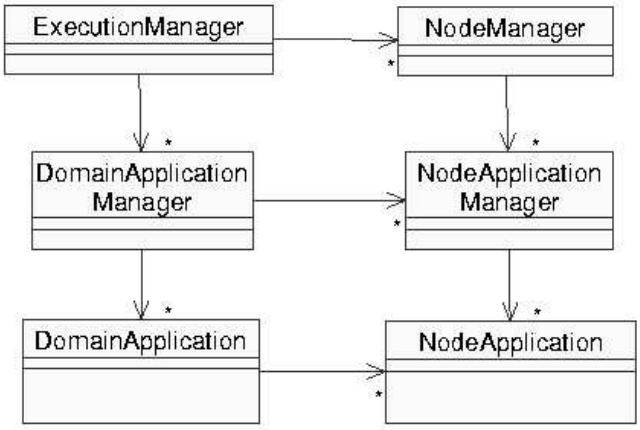


FIG. 3.4 – Modèle de gestion d'exécution de déploiement D&C (diagramme de classes)

En dépit de la richesse de ses modèles de données et d'exécution du déploiement, la spécification D&C est assez compliquée et n'a pas encore d'implémentation.

3.4.6 Autres outils de déploiement

D'autres solutions de déploiement sont issues de travaux de recherche. Nous en citons quelques uns dans cette section de manière à dégager les évolutions des outils de déploiement.

La spécification de déploiement et de configuration de COACH [COA03] étend la spécification de CCM en définissant un modèle permettant de décrire l'environnement cible de déploiement ainsi qu'un outil d'assignation des composants aux différents nœuds. COACH couvre des aspects de déploiement non couverts par CCM comme le téléchargement des implémentations des composants, la supervision de l'assemblage des applications en cours d'exécution et le support d'interfaces spéciales pour la négociation de la qualité de service.

Le travail [Sab03] permet le déploiement de services personnalisés. La personnalisation se fait à travers des préférences utilisateurs. Les préférences utilisateurs permettent de donner des valeurs à la volée aux propriétés de configuration des composants de l'application et de choisir un assemblage de l'application parmi plusieurs assemblages possibles. Ce travail est une extension du modèle de déploiement CCM.

ORYA [LB03] et Software Dock [HHW99] offrent des environnements de déploiement génériques qui automatisent les différents processus de déploiement dans un environnement distribué et hétérogène, et qui couvrent toutes les étapes du cycle de vie du déploiement. Ces deux environnements ont presque la même approche conceptuelle dans la modélisation du processus de

déploiement et des applications à déployer. ORYA est plus orienté vers le déploiement de logiciels à grande échelle et offre une approche plus flexible basée sur la technologie de la fédération d'outils coopératifs et sur un modèle de processus de déploiement décomposable en plusieurs activités de déploiement de base. ORYA et software Dock ont des mécanismes permettant d'étudier les contraintes matérielles et logicielles de placement d'une application ainsi que la résolution de ses dépendances au moment du déploiement.

L'environnement pour services partitionnables [IHAK02] inclue un module de plannification qui détermine le placement des services en fonction des conditions du réseau et des machines.

[MRM02] décrit une approche pour le déploiement d'applications à base de composants sur des terminaux mobiles en proposant des entités légères pour le déploiement et la gestion d'architecture d'applications. au cours de l'exécution sans description de règles précisant suite à quels besoins ces reconfigurations d'architectures seront réalisées.

Les ADLs (Langages de Description d'Architecture) [MT00] sont des outils qui permettent de construire une application par assemblage de composants en raisonnant avec un modèle abstrait de composants plutôt que de raisonner directement à partir d'un modèle de composants industriel. Ces outils offrent aussi des solutions de déploiement mais leur description d'assemblage est trop formelle. Ils sont plus adaptés à la vérification de l'architecture d'applications et la gestion de leur complexité. Leur vision abstraite permet une meilleure compréhension de l'architecture, une meilleure réutilisation des modèles et une analyse de l'assemblage de l'application.

La plate-forme de composants Fractal [BCS04, BCS02] définit un outil de déploiement pour les composants Fractal qui permet de construire des applications à partir d'un modèle à composants supportant la création de compositions hiérarchiques (composants composites). Le modèle abstrait de fractal définit un composant comme étant une entité qui fournit et requiert des interfaces fonctionnelles et qui fournit un ensemble d'interfaces de contrôle (interface de gestion du cycle de vie, interface de liaison entre instances, interface de contrôle du contenu dans le cas où le composant serait composite). L'API de Fractal permet de réaliser des activités de déploiement tels que créer des instances de composants à partir des types de composants, les configurer et les connecter entre elles. Fractal définit un ADL qui permet de décrire l'architecture d'une application de façon déclarative.

3.4.7 Étude comparative des outils de déploiement des applications

Les tableaux 3.1 et 3.2 montrent une étude comparative des méta-informations prises en compte par les différents outils de déploiement d'applications à base de composant, selon le cadre décrit dans la section 3.3.3. Le tableau 3.3 compare les activités offertes par les interfaces de ces outils. Nous analysons dans ce qui suit ces tableaux.

Informations sur l'implémentation du composant

Chacun des cinq outils étudiés supporte la description d'information sur l'implémentation des composants. CCM et D&C permettent la description de plusieurs implémentations d'un même

composant tandis que OSGi, EJB et .Net se contentent de décrire une seule implémentation du composant. Cela est dû au fait que dans OSGi et EJB les implémentations des composants sont en Java, vont s'exécuter sur une machine virtuelle Java et suivent le principe "Write Once, Run Anywhere" [Cur98]. Les implémentations des applications .Net sont écrites en un langage tels que C#, Visual Basic .NET ou Jscript et tournent sur CLR (*Common Language Runtime*). L'indépendance de l'implémentation du composant de la plate-forme d'exécution représente, en théorie, une facilité de déploiement puisque cela ne nécessite pas une implémentation par type de plate-forme d'exécution. Cependant, la réalité est différente [CA01]. Par exemple dans le cas d'EJB, les archives (ainsi que les descripteurs) étant créées par des outils de déploiement spécifiques à une implémentation donnée (comme par exemple : BEA [BEA], WebSphere [WS] ou Jboss [JB]), elles ne sont pas aussi indépendantes de la plate-forme que cela.

Le support d'implémentations multiples n'est pas seulement utile dans le cadre de diversité de le plate-formes. Un composant peut par exemple avoir plusieurs implémentations, chacune utilisée dans un contexte particulier. Par exemple, un composant interface graphique peut avoir une version d'implémentation allégée utilisable sur un PDA.

OSGi, EJB et .Net ne permettent pas la description de contraintes sur l'environnement d'exécution. CCM et D&C supportent les informations sur les contraintes de la future plate-forme d'exécution. L'expression des contraintes au niveau de D&C n'est définie que suite à la projection du modèle indépendant de la plate-forme sur un modèle spécifique à une plate-forme donnée. Ces informations sont utilisées au moment du placement des composants sur les sites d'exécution.

Informations structurelles sur le composant

La description d'informations structurelles peut être réalisée avec tous les outils de déploiement que nous avons étudiés à l'exception de .Net. Ces informations permettent à un outil de conception (ou d'assemblage) d'afficher les interfaces d'un composant et de connecter les composants entre eux. Par exemple un composant qui requiert l'interface X pourrait être connecté à un composant qui offre une interface X.

Informations sur les dépendances du composant

La description des dépendances des composants de différents paquetages et ressources est supportée par chacun des outils. CCM supporte la description des dépendances d'un composant d'une façon générale ainsi que des dépendances qui sont propres aux différentes implémentations du composant.

Informations sur les propriétés fonctionnelles et non fonctionnelles du composant

La description des propriétés non fonctionnelles ne sont pas prises en compte par les modèles de déploiement OSGi et .NET. Dans EJB et CCM, les propriétés non fonctionnelles d'un composant sont utilisées pour déterminer le type de conteneur dans lequel le composant a besoin d'être

déployé et pour fournir au conteneur les informations sur le composant qui spécifient les services non-fonctionnels (transaction, persistance, etc.). Or, dans .Net et OSGi les composants ne sont pas déployés dans des conteneurs.

Les propriétés non fonctionnelles sont supportées par D&C suite à une projection du modèle indépendant de la plate-forme sur un modèle spécifique à la plate-forme tels que CCM ou EJB.

.NET et OSGi ne supportent pas la description des propriétés fonctionnelles au niveau de leur manifeste.

Modélisation du domaine de déploiement

Seul D&C permet la modélisation du domaine de déploiement.

Plan de déploiement et description d'assemblage abstrait

CCM et D&C supportent la description de plan de déploiement. Dans EJB seule la description des assemblages abstraits est supportée. La notion d'assemblage abstrait et de plan de déploiement n'existe ni pour OSGi ni pour .Net. Dans .Net l'assemblage de composants est basé sur la possession de références sur les bibliothèques contenant leurs implémentations.

Interfaces de déploiement

Les activités de déploiement de base des composants à savoir l'instanciation, la configuration et leurs connexions sont supportées par chacun des outils de déploiement. Ils peuvent être regroupés en une seule opération comme dans CCM ou ils peuvent être séparés ; la séparation en plusieurs opérations permet plus de flexibilité et de dynamicité. Ces activités sont précédées par l'installation du paquetage de composant sur le site cible.

Seul D&C permet de donner des informations sur les sites cibles de déploiement et spécifie une méthode qui permet l'assignation automatique d'un composant à un site cible.

Les interfaces de supervision des assemblages de composants déployés ou en cours de déploiement sont supportées par D&C et J2EE.

Le déploiement d'applications .NET se distingue par sa simplicité : il suffit de copier les applications dans leur répertoire [Sco00]. Plus besoin de modifier les registres Windows et ensuite de redémarrer l'ordinateur. Pour réussir cela, le framework .NET a introduit les concepts d'assemblage et de manifeste afin de résoudre les problèmes de version en utilisant la technologie du déploiement dit "side by side".

3.4.8 Synthèse

Nous avons étudié dans cette section différents modèles de déploiement en nous basant sur le cadre de déploiement défini dans la section 3.3.3. A l'exception de D&C, ces modèles sont rattachés à une plate-forme donnée et dépendent du modèle abstrait de composant qu'ils déploient. Cependant, ils présentent des points communs et des différences que nous avons discutés. Le modèle de déploiement D&C est le plus complet. Il définit la description de méta-informations la plus riche et l'API la plus complète. Son avantage réside dans son modèle indépendant de la plate-forme. Ce modèle est malheureusement très complexe et n'a pas encore été implémenté. Les outils de déploiements EJB, CCM, .NET et OSGi ont tous des implémentations. Le modèle de composants EJB est le plus utilisé dans le monde industriel mais le modèle de déploiement CCM est plus complet que celui de EJB ; il supporte la description de plusieurs implémentations d'un même composant et sa description d'un plan de déploiement est son point fort.

Parmi les solutions de déploiement que nous venons de citer aucune ne s'intéresse à l'adaptation du déploiement au contexte. Elles prennent en compte presque toutes des mécanismes qui étudient les contraintes de placement mais ne comportent pas de mécanismes qui étudient les informations de contexte d'une façon générale.

3.5 Conclusion

Ce chapitre a permis de mettre en évidence les concepts liés au déploiement. Nous avons commencé par les concepts de déploiement monolithiques en présentant un panel d'outils représentatif des approches existantes. Ensuite, nous nous sommes concentrés sur les concepts de déploiement des applications à base de composants. Nous avons pour cela étudié les modèles de déploiement offerts pour différentes plate-formes de composants comme CCM, J2EE et .Net ainsi que le modèle de déploiement D&C et le modèle de déploiement d'OSGi.

L'apport de ce chapitre est la définition du cadre de caractérisation d'outils de déploiement et la comparaison de ces outils. Cette étude a été faite dans le but d'identifier les méta-informations ainsi que les opérations utilisées dans le déploiement des applications à base de composant afin d'étudier la possibilité de les enrichir pour supporter l'adaptation au contexte.

L'étude du déploiement des applications à base de composants nous a montré que l'apport fondamental du paradigme composant est la capacité de déployer une application distribuée d'une manière automatique à partir de la description d'un ensemble de méta-informations. Ces méta-informations se situent au niveau composant et au niveau assemblage de composants. Elles décrivent l'assemblage de l'application et les différentes informations fonctionnelles et non fonctionnelles de chaque composant de l'application.

Aucun des outils de déploiement à base de composants que nous avons étudié ne prend en compte le contexte général de l'utilisateur au moment du déploiement des applications, le seul contexte pris en compte par quelques outils de déploiement est le contexte qui représente le domaine de déploiement de l'application (disponibilité de ressources matérielles et logicielles).

Notre travail consiste à introduire la prise en compte du contexte dans le déploiement d’applications à base de composants. En effet, l’adaptation du déploiement des applications à base de composants est plus facile que celui des applications monolithiques grâce à la simplicité d’ajout et de retrait de composants et pratique grâce à la possibilité de distribution des différents composants de l’application sur différents nœuds sans avoir à intervenir sur chaque site.

	CCM	EJB	OSGi	D&C	.NET
Implémentation du composant	Support de plusieurs implémentations. Pour chacune, description, référence sur son code , nom du compilateur, dépendances, langage de programmation, contraintes de placement	Nom des classes d'implémentation java.	Emplacement des classes et ressources du bundle.	Support de plusieurs implémentations (monolithique ou composite). Pour chacune, dépendances, paramètres d'exécution, référence sur son code, contraintes de placement	liste des fi chiers qui composent l'implémentation.
Structure du composant	Le modèle abstrait du composant (facettes, réceptacles, puits d'événement, attributs) et de sa maison	Les noms des interfaces offertes et des interfaces maison	Services requis et fournis, nom de la classe Activator, paquetages requis et fournis.	Interface du composant : ses opérations, ses attributs et ses ports	Pas d'informations structurelles au niveau du manifeste
Dépendances du composant	Paquetages, ressources, fi chiers	Ressources	Bibliothèques natives, liste de paquetages qui pourront être importés en cours d'exécution	Paquetages de composants et artefacts	liste d'autres assemblages statiquement référencés par l'assemblage, liste de ressources

Table 3.1 – Comparaison des méta-informations prises en compte par les différents outils de déploiement (partie 1)

	CCM	EJB	OSGi	D&C	.NET
Propriétés non fonctionnelles du composant	Type du composant, transaction, persistance, QoS des ports d'événements, threading	Type du composant, transaction, persistance, relations et attributs gérés par le conteneur, gestion d'état	Non gérées	Peuvent être supportées au niveau du PSM	Non gérées
Propriétés fonctionnelles du composant	Description des propriétés de configuration	Description des propriétés de configuration	Non gérées	Description des propriétés de configuration	Non gérées
Modélisation du domaine de déploiement	Non supporté	Non supporté	Non supporté	Modélisation des nœuds, leurs connexions et éléments de routages	Non supporté
Plan de déploiement	Supporté	Non supporté	Non supporté	Supporté	Non supporté
Description d'assemblage abstrait	Non supporté	Supporté	Non supporté	Supporté au niveau description de l'assemblage des composants composites.	Non supporté
Informations générales supportées	Nom, description générale, type de paquetage, producteur, licence, références sur tous les descripteurs de déploiement.	Nom, description générale	Nom du bundle, sa description, sa version, URL documentation, coordonnées du producteur, sa licence, sa catégorie, URL des mises à jour	Nom	Nom, numéro de version, clé

Table 3.2 – Comparaison des méta-informations prises en compte par les différents outils de déploiement (partie 2)

	CCM	J2EE	OSGi	D&C	.NET
Installation de paquetage	manuelle	manuelle	automatique	automatique	manuelle
Instanciation	supportée	supportée	supportée	supportée	supportée
Connexion	supportée	supportée	supportée	supportée	supportée
Assignation d'un site	non supporté	non supporté	non supportés	supportés	non supportés
Supervision assemblages	non supportés	non supportés	non supportés	supportés	non supportés
Activation/ désactivation	supportée	supportée	supportée	supportée	supportée
Désinstallation	manuelle	manuelle	manuelle	manuelle	manuelle

Table 3.3 – Comparaison des activités de déploiement prises en compte par les différents outils de déploiement

Deuxième partie

Solution Proposée

Chapitre 4

Étude de la sensibilité du déploiement au contexte

Suite à l'étude des méta-informations et des entités d'exécution nécessaires au bon déroulement du déploiement des applications à base de composants présentée dans le chapitre 3, nous illustrons dans ce chapitre la sensibilité du déploiement au contexte à travers des scénarios de déploiement d'applications (section 4.1) et nous identifions les paramètres du déploiement qui sont sensibles au contexte (section 4.2). Enfin, nous présentons la démarche suivie dans la solution que nous proposons pour adapter le déploiement au contexte (section 4.3).

4.1 Scénarios d'illustration

Dans cette section, nous décrivons deux applications : une application de vente en ligne pour utilisateurs mobiles et une application de gestion de crises. Ces applications se rapportent à deux domaines différents qui permettent d'illustrer les types de contextes qui peuvent avoir un impact sur le déploiement et les paramètres de déploiement qui peuvent varier en fonction du contexte.

4.1.1 Application de vente en ligne pour utilisateurs mobiles

Une application de vente en ligne peut se révéler intéressante pour des utilisateurs mobiles qui souhaitent diminuer les temps passés dans les magasins. Grâce à cette application, l'utilisateur gagne du temps en commandant ses courses avant d'arriver au magasin.

L'utilisateur peut déployer l'application de vente en ligne à la volée et d'une façon automatique à partir du poste de son bureau, de chez lui, de chez des amis ou bien à partir de son PDA lorsqu'il est dans sa voiture. Cette installation automatique épargne à l'utilisateur la réinstallation manuelle de l'application. Une fois l'application de vente en ligne installée, l'utilisateur peut visiter virtuellement le magasin le plus proche et commander en ligne les produits dont il a besoin.

L'application de vente en ligne est constituée de quatre composants.

– Un composant Interface Graphique (GUI) qui permet à l'utilisateur de parcourir les rayons du magasin et de sélectionner les produits souhaités. Ce composant est déployé sur le terminal de l'utilisateur et a deux implémentations : une est dédiée aux terminaux ayant de petits écrans tels que des PDAs, la seconde est dédiée aux terminaux ayant un écran supérieur ou égal à 14 pouces. La première implémentation présente une version allégée de la deuxième qui peut tenir sur un petit écran. La langue d'utilisation de ce composant (français, anglais, arabe ou chinois) peut être configurée selon le langage défini par les préférences de l'utilisateur.
– Le composant Base de Données (BD) fournit des informations sur les produits disponibles dans le magasin. Le composant BD est un composant préinstancié sur un serveur distant du domaine offert par le prestataire du service. Cela veut dire que ce composant n'est pas instancié au moment du déploiement de l'application mais qu'il a été instancié avant et que le composant qui l'utilise va s'y connecter au moment du déploiement. Il existe un composant BD pour chaque magasin, l'utilisateur est connecté au composant BD du magasin qui lui est le plus proche.
– Le composant Vue Locale (VL) représente une vue partielle du composant BD. Le composant VL sert à se substituer au composant BD quand celui-ci n'est plus accessible : typiquement, quand l'utilisateur est déconnecté du réseau volontairement ou involontairement. Si le composant BD est accessible, VL n'est pas déployé.
– Le composant Panier contient les produits sélectionnés par l'utilisateur. Selon les ressources du terminal, le composant Panier peut être déployé aussi bien sur le terminal de l'utilisateur que dans un serveur de proximité. L'utilisateur peut choisir d'utiliser ce même panier ou remplir un nouveau panier lors d'un prochain achat. Ce choix a un impact sur la propriété non fonctionnelle qui représente la persistance du composant panier.

La figure 4.1 montre deux assemblages possibles de l'application (assemblage sans et avec une vue locale) et la figure 4.2 montre deux exemples possibles pour le choix de l'emplacement du composant Panier et deux choix possibles pour l'implémentation du composant GUI.

Après avoir étudié la structure de l'application de vente en ligne et ses possibilités de déploiement en fonction du contexte, nous pouvons distinguer quatre types de contextes ayant un effet sur le déploiement de l'application de vente en ligne :

– les ressources offertes par le terminal utilisateur ainsi que par les nœuds fournis par le prestataire du service (dans le choix de l'implémentation du GUI et le placement du composant panier) ;
– la localisation de l'utilisateur : sa zone de connexion (dans le choix de du composant BD auquel va être connecté le composant GUI) ;
– les préférences de l'utilisateur (le langage utilisé et la persistance du composant Panier) ;
– la connexion réseau de l'utilisateur (connecté, non connecté)

4.1.2 Application de gestion de crises

On se place dans le cadre de grandes catastrophes tels qu'un crash d'avion, une collision de trains ou un tremblement de terre, donnant lieu à un nombre important de victimes qui nécessite un grand nombre d'équipes et d'équipements de sauvetages. On suppose que les équipes

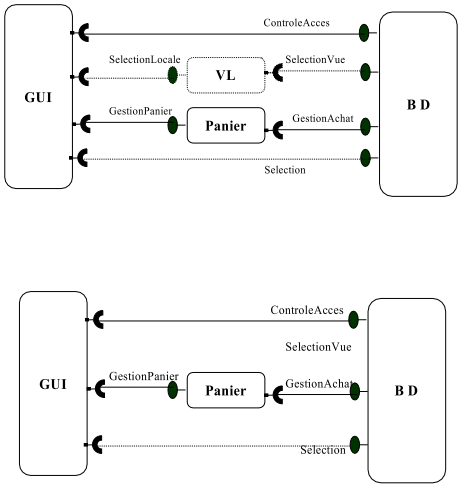


FIG. 4.1 – Application de vente : variation d'architecture

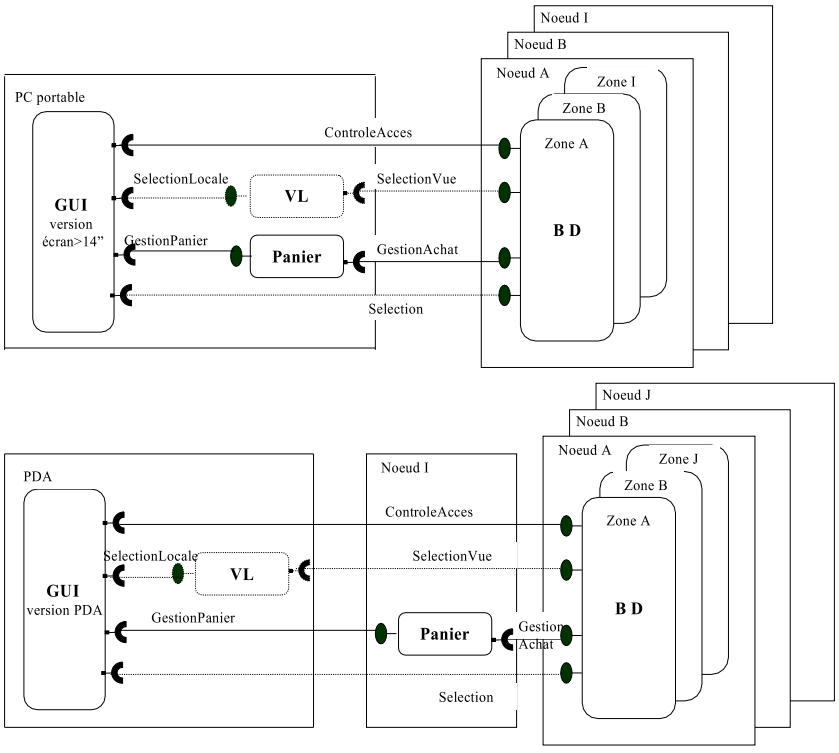


FIG. 4.2 – Application de vente : variation des emplacements et des implémentations des instances de composants

utilisent une technologie mobile et distribuée pour gérer des tâches telles que le partage d'informations, la communication entre les membres d'équipes différentes, l'obtention d'informations sur la disponibilité des hôpitaux, la communication des diagnostiques médicaux, etc. Cette application a été développée comme application de démonstration dans le cadre du projet AMPROS [AMP].

La zone de catastrophe est divisée en trois :

– la zone des victimes : la couverture réseau de cette zone dépend de l'endroit (montagne, tunnel, sous-sol, etc.) ;
– la zone médicale primaire : offre les premiers soins. Elle comporte des équipements médicaux et des nœuds d'exécution (serveurs de composants) qui peuvent héberger des composants qui n'ont pas pu être déployés sur les PDAs des sauveteurs. Si les victimes ne peuvent pas être traitées dans cette zone, elles sont évacuées selon la gravité de leurs blessures et la disponibilité des hôpitaux ;
– le centre de commandement : il inclue des policiers et des pompiers.

Les membres des équipes de sauvetage ont des PDAs interchangeables. Les PDAs ont des ressources limitées ne leur permettant pas d'héberger toutes les applications de sauvetage. Le déploiement d'applications à la volée permet d'économiser les ressources du terminal.

Nous voulons illustrer le déploiement sensible au contexte d'une application qui permet à un sauveteur de saisir l'état d'une victime et de le diffuser à plusieurs de ses collaborateurs appartenant à son équipe ou à d'autres équipes. Cette application comporte les composants suivants :

– un composant (GUI) représentant une interface graphique permettant aux sauveteurs de saisir l'état des victimes. Ce composant doit être placé sur le terminal du sauveteur ;
– un composant "serveur de partenaires" qui reçoit les informations sur les victimes, les traite si nécessaire et les diffuse vers les sauveteurs partenaires concernés. Le placement de ce composant dépend de la disponibilité des ressources ; il peut être placé sur le terminal du sauveteur chargé de la saisie de l'état des victimes ou sur un des serveurs de la zone médicale primaire. Si l'état du patient nécessite une évacuation vers un hôpital, ce composant aura besoin d'utiliser une base de données qui contient les anciennes fiches des victimes. Ce composant a donc une dépendance qui varie en fonction du contexte ;
– un ou plusieurs composants "partenaires" permettant aux sauveteurs de recevoir l'état des victimes. Le nombre de composants de ce type dépend du nombre de sauveteurs avec lesquels collabore le sauveteur. Ces composants doivent être placés sur les terminaux des collaborateurs concernés.
– un composant "journalisation" qui permet à l'utilisateur de garder une trace des activités qu'il a réalisées lors d'une déconnexion réseau. Ce composant n'est déployé que s'il y a un risque de déconnexion.

Dans ce qui suit, nous décrivons les différents profils des équipes de sauvetage. Nous avons six profils possibles :

– le profil "Chef des Pompiers" (CP) : il gère la zone de sinistre. Il a besoin de déployer plusieurs applications sur son PDA pour coordonner toutes les opérations de sauvetage. Il gère les informations suivantes : le nombre total de victimes, la localisation des sauveteurs sur le terrain. Il envoie un rapport régulier au centre de commandements ;

- le profil "Chef Médical" (CM) : il a une vue générale sur la situation médicale. Il organise les opérations de sauvetage et affecte les sauveteurs aux victimes ;
- le profil "Gestionnaire de Tri" (GT) : il trie les victimes selon la gravité de leur état ;
- le profil "Gestionnaire des Évacuations" (GE) : médecin qui décide l'évacuation des victimes vers les hôpitaux selon les diagnostiques des médecins ;
- le profil "Gestionnaire de l'Hôpital sur le Terrain " (GHT) : il affecte les médecins aux victimes évacuées ;
- Sauveteurs (SAUV) : ils s'occupent des victiment sur le terrain.

Les composants de type GUI et partenaire sont associés à plusieurs implémentations qui dépendent du profil de l'utilisateur. Par exemple, l'implémentation du composant GUI associée au gestionnaire de tri lui fournit des boutons radio sur lesquels il clique pour saisir l'état du patient. Par contre celle associée au chef des pompiers permet de saisir des rapports. L'implémentation du composant de type partenaire associée au sauveteur lui permet d'afficher une affectation à une victime et l'état de cette victime. Par contre, l'implémentation du composant de type partenaire associée au chef médical lui permet d'afficher des rapports et un état global de toutes les victimes.

La figure 4.3 montre des assemblages possibles de l'application (avec ou sans composant de journalisation, nombre de composants "partenaires" différents pour chaque utilisation) , avec différents emplacements du composant "serveur de partenaires" ainsi que des choix d'implémentations pour les composants "partenaires" et GUI selon le profil des sauveteurs.

Si nous faisons un bilan des contextes pris en compte pour le déploiement de l'application de gestion de crises. Nous trouvons les contextes suivants :

- le profil des utilisateurs,
- le nombre de sauveteurs,
- les préférences des sauveteurs,
- la connexion réseau,
- l'état du patient,
- les ressources offertes par les terminaux, ainsi que les noeuds offerts par la zone médicale primaire.

Les deux applications que nous avons décrites mettent en évidence la sensibilité du déploiement au contexte et illustrent l'intérêt pour que les outils de déploiement soient adaptables au contexte. Les deux applications nous ont aidé à dégager les paramètres de déploiement qui peuvent varier en fonction du contexte ainsi que les types de contextes qui peuvent agir sur le déploiement. Nous discutons ces deux points dans la section suivante.

4.2 Sensibilité du déploiement au contexte

Nous constatons à partir des scénarios que nous avons étudiés que les types de contexte pouvant agir sur le déploiement dépendent de la sémantique de l'application. Des contextes tels que les ressources du terminal de l'utilisateur, les ressources des nœuds de déploiement et les préférences de l'utilisateur présentent des contextes qui doivent généralement être toujours pris

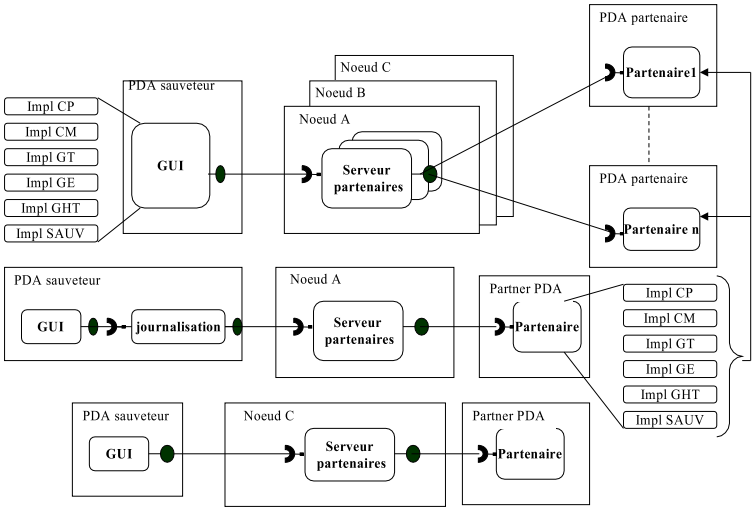


FIG. 4.3 – Application de gestion de crise : variation des paramètres de déploiement

en compte lors du déploiement mais la considération d'autres types de contextes dépend de l'application à déployer.

Dans le chapitre 3, nous avons étudié les solutions de déploiement offertes par différentes plate-forme de composants et nous avons identifié dans la section 3.3.3 un ensemble de méta-informations utilisées par ces outils pour réaliser le déploiement des applications à base de composants. Parmi ces méta-informations, nous avons identifié cinq paramètres de déploiement qui peuvent varier en fonction du contexte :

- **L'architecture** de l'application. Le nombre d'instances de composants constituant l'application à déployer et les connexions entre elles peuvent varier en fonction du contexte. L'application peut à la base contenir plusieurs instances de composants et connexions mais seuls les instances de composants et les connexions requis par le contexte doivent être déployés. Ceci est le cas des instances de composants "Vue Locale" et "journalisation" et leurs connexions avec les autres instances de composants. Le nombre des instances de composants "partenaires" varie à son tour en fonction du nombre de sauveteurs.
- La sélection de **l'emplacement des instances de composants**. Si les ressources du terminal de l'utilisateur sont insuffisantes pour installer toutes les instances de composants de l'application, seulement une partie de ces instances sera installée sur le terminal utilisateur et le reste sera distribué sur des machines distantes. Ces machines peuvent être sélectionnées selon plusieurs paramètres de contexte tels que la disponibilité de leurs ressources et la localisation de l'utilisateur. Ceci est le cas des instances Panier et "serveur de partenaires"
- Le choix des **implémentations** des composants : ce choix dépend essentiellement de la plate-forme logicielle et matérielle sur laquelle va être déployée l'instance de composant mais peut aussi dépendre d'autres types de contextes tel que le profil de l'utilisateur, comme dans le cas des composants GUI et "partenaires".

- **Les valeurs des propriétés** : chacune des propriétés fonctionnelles et non fonctionnelles d'un composant dépendent du contexte. Le langage de l'utilisateur est une propriété fonctionnelle du composant GUI et la persistance du composant Panier est une propriété non fonctionnelle. Ces deux propriétés dépendent de la préférence de l'utilisateur.
- **Les dépendances du composant** : lorsque le contexte d'utilisation du composant change, ses dépendances peuvent changer. Comme dans le cas du composant " serveur de partenaires " dont la dépendance à la base de données des fichiers des victimes dépend de l'état de ces derniers.

Les outils de déploiement étudiés dans le chapitre 3 décrivent ces paramètres d'une manière statique à travers les descripteurs de déploiement, ce qui ne permet pas de les faire varier selon le contexte. Cela ne leur permet pas de répondre aux besoins de déploiement des utilisateurs mobiles identifiés dans la section 1.2. Nous décrivons dans la section qui suit deux scénarios qui illustrent la variabilité de ces paramètres en fonction du contexte.

4.3 Démarche suivie

Nous présentons dans cette section une vue générale sur notre proposition, CADeComp (**C**ontext-**A**ware **D**eployment of **C**omponents), pour adapter le déploiement au contexte. Nous commençons par étudier l'impact de l'adaptation du déploiement sur le cycle de vie de développement des applications. Ensuite nous expliquons les principes de base de CADeComp.

4.3.1 Adaptation du déploiement et cycle de vie du développement d'applications

Nous avons étudié dans le chapitre 2 les différentes étapes nécessaires pour l'adaptation d'un logiciel au contexte, nous montrons dans ce qui suit que l'intégration des mécanismes d'adaptation au contexte dans un logiciel et en particulier dans le déploiement nécessite sa prise en compte dans toutes les étapes de production de ce logiciel à savoir sa spécification, sa conception et son développement.

La prise en compte du contexte au stade de la spécification consiste à définir l'ensemble des informations de contexte auxquelles l'application est sensible et spécifier les différents comportements de l'application correspondant à chaque état de contexte.

La phase de conception consiste à spécifier les différentes méthodes et techniques permettant d'interagir avec le contexte, c'est-à-dire les techniques, qui vont être utilisées pour collecter, interpréter et modéliser le contexte ainsi que les techniques d'adaptation au contexte. Les techniques d'adaptation définies doivent permettre d'aboutir aux résultats et comportements attendus. La phase de conception peut consister ainsi à choisir un intergiciel sensible au contexte existant qui implémente les différentes techniques identifiées dans cette phase.

Enfin, la phase de développement implémente les méthodes et techniques définis dans la phase de conception ou réalise le placement de l'application au-dessus d'un intergiciel sensible au contexte.

L'adaptation du service de déploiement au contexte suit le même principe. Elle nécessite également la définition pour chaque application à déployer des informations de contexte qui peuvent agir sur son déploiement et la définition de la variation du comportement du déploiement de l'application en fonction du contexte. Comme la production d'une application à base de composants consiste d'abord à produire ses différents composants (spécification, conception et développement) ensuite à les assembler. Nous proposons de définir ces informations de contexte au moment de la spécification de ses différents composants et au moment de leur assemblage. Par exemple, au moment de la spécification d'un composant, il est possible de définir la plateforme qui peut l'accueillir et au moment de l'assemblage d'une application il possible d'identifier la sensibilité de son architecture à d'autres contextes comme la localisation de l'utilisateur ou ses préférences. Nous nous proposons aussi de définir des entités intergicielles qui implémentent des techniques d'adaptation afin de simplifier la prise en compte du contexte lors de la conception et la réalisation des composants.

Suite à l'identification des différentes informations nécessaires à l'adaptation du déploiement d'une application, c'est-à-dire les informations de contexte auxquelles le déploiement est sensible et les différents comportements du déploiement en fonction de ces contextes vient l'étape d'adaptation du déploiement qui nécessite à son tour des mécanismes d'adaptation spécifiques au déploiement.

Notre but est donc d'offrir les moyens de décrire toutes les informations nécessaires pour adapter le déploiement d'une application au contexte durant les phases de sa production et de définir les mécanismes qui utilisent ces informations pour réaliser l'adaptation du déploiement au contexte d'une façon automatique. Nous proposons pour cela un cadriciel pour le déploiement sensible au contexte des applications à base de composants intitulé CADeComp.

4.3.2 Vue générale de CADeComp

Dans le chapitre 2, nous avons montré l'importance de l'utilisation des intergiciels sensibles au contexte pour la séparation entre le traitement des informations de contexte et l'application. Pour adapter le déploiement au contexte, nous allons suivre ce même principe en plaçant CADeComp au-dessus d'une infrastructure de contexte qui représente un intergiciel sensible au contexte (voir figure 4.4). Cette infrastructure a comme rôle de masquer l'hétérogénéité des sources de contextes. Elle communique avec les sources d'informations de contexte pour leur acquisition et leur interprétation afin d'en obtenir des informations de contexte de haut niveau. Elle joue un rôle déterminant dans la simplification de la prise en compte du contexte dans les étapes de conception et développement des composants et d'assemblage des applications.

Comme notre objectif est de déployer des applications à base de composants, notre service de déploiement sensible au contexte est par conséquent amené à être intégré dans une plateforme de composants qui offre aux composants des services tels que la gestion de leur cycle de vie, leur persistance, leur politique de sécurité et leur contrôle d'accès.

Dans le chapitre 3, nous avons étudié plusieurs outils de déploiement offerts par différentes plate-formes de composants tels que J2EE, CCM et .Net. Ces outils offrent des fonctionnalités élémentaires de déploiement tels que l'instanciation des composants, leur interconnexion et leur configuration mais ne prévoient pas l'adaptation du déploiement au contexte. Nous nous fixons comme objectif de rajouter une couche d'adaptation au-dessus de ces outils de déploiement sans introduire de modification dans leur fonctionnement interne.

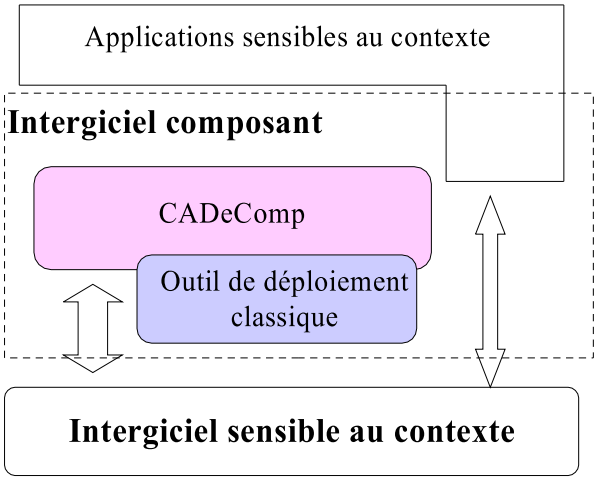


FIG. 4.4 – Vue générale de l'architecture de CADeComp

Dans le cadre de cette thèse, le service de déploiement est fourni par un prestataire de service qui offre un ensemble de nœuds d'exécution (serveurs d'exécution) permettant d'alléger le terminal de l'utilisateur en hébergeant les composants qui ne peuvent pas être déployés sur ce dernier. L'ensemble de ces nœuds sont connectés entre eux par des liens réseaux pour former ce que nous appelons "un domaine de déploiement".

Suite à l'étude que nous avons faite dans de ce chapitre sur la sensibilité du déploiement au contexte, nous estimons que l'adaptation du déploiement au contexte ne nécessite pas la modification du mécanisme de déploiement lui-même puisque ce sont les paramètres de déploiement qui varient en fonction du contexte et non le mécanisme d'instanciation des composants ou leur configuration. Cela veut dire que nous n'avons, par exemple, pas besoin de définir différentes manières d'instancier un composant de manière à ce que l'outil de déploiement change de mécanisme d'instanciation selon le contexte. Cela nous amène à écarter des mécanismes d'adaptation telles que la réflexion ou la programmation par aspects et à choisir d'utiliser plutôt des contrats pour la spécification de la variation du déploiement d'une application donnée au contexte.

Pour prendre en compte la variabilité des différents paramètres de déploiement cités dans 4.2, nous proposons l'écriture d'un ensemble de règles d'adaptation qui spécifient la variabilité de ces paramètres pour chaque application en fonction du contexte.

Ces règles forment un contrat pour le déploiement d'une application donnée. Ce contrat dépend de la sémantique de l'application : il identifie les informations de contexte qui peuvent agir

sur le déploiement d'une application et définit la manière avec laquelle les paramètres de déploiement vont varier en fonction de ce contexte.

Les règles d'adaptation permettent à un mécanisme d'adaptation de déploiement basé sur un algorithme générique de déterminer à la volée les valeurs des paramètres variables de déploiement, après prise en considération de l'état du contexte au moment du déploiement au lieu de leur donner une valeur fixe à priori.

Comme les paramètres de déploiement peuvent être étroitement liés à l'outil de déploiement utilisé et que les informations de contexte dépendent de l'infrastructure de contexte responsable de la collecte et du traitement des informations de contexte, nous voulons que la description de ces règles d'adaptation ainsi que leur traitement soient indépendants des plates-formes de composants et de contextes. Nous définissons à cette fin un modèle d'adaptation de déploiement indépendant de la plate-forme qui est conforme à MDA (Model Driven Architecture) de l'OMG [OMG01].

Dans le cadre de cette thèse, nous ne nous intéressons qu'à l'adaptation du déploiement initial des applications c'est à dire l'installation des paquetages de composants, leur instanciation et leur activation. L'adaptation de ces activités a un impact direct sur la diffusion des paquetages de composants. Les composants installés sont désinstallés suite à leur désactivation afin de conserver les ressources du terminal de l'utilisateur. Dans le reste des chapitres de ce document le terme déploiement d'un composant désigne son installation, son instanciation et son activation. Le terme déploiement d'application désigne le déploiement de ses composants.

4.4 Conclusion

Ce chapitre a mis en évidence la sensibilité du déploiement au contexte en identifiant cinq paramètres de déploiement variables en fonction du contexte. Cette sensibilité a été illustrée par deux exemples de déploiement d'applications dans un environnement mobile.

Nous avons présenté dans ce chapitre les principes de base de CADeComp en insistant sur le fait que CADeComp est placé au-dessus d'un intergiciel sensible au contexte et s'intègre dans un intergiciel à base de composants. CADeComp est basé sur un ensemble de règles et d'algorithmes définis d'une façon indépendante de la plate-forme. La définition des règles d'adaptation du déploiement est réalisée au long de différentes étapes de production de l'application :la spécification des composants de l'application et l'assemblage de ces composants.

Cette partie du rapport de la thèse comporte, en plus de ce chapitre, les chapitres 5 et 6. Le chapitre 5 détaille le modèle de données qui permet la description des règles d'adaptation et le chapitre 6 décrit les algorithmes et les entités qui utilisent ces règles pour réaliser l'adaptation du déploiement au contexte.

Chapitre 5

Modèle de données de CADeComp indépendant de la plate-forme

5.1 Introduction

Dans le chapitre 4 nous avons identifié les différents paramètres de déploiement qui peuvent varier en fonction du contexte. Nous proposons, dans ce chapitre, un modèle indépendant de la plate-forme pour la description des méta-informations nécessaires à la variation de ces paramètres. Pour adapter le déploiement des applications au contexte, nous définissons deux niveaux de méta-informations :

- des méta-informations d'adaptation situées au niveau composant. Ces dernières se rapportent à un seul type de composant et se situent dans le paquetage d'un composant. Elles sont spécifiées par le développeur du composant ;
- des méta-informations d'adaptation situées au niveau application. Ces dernières se rapportent à l'ensemble des instances de composants qui constituent l'application. Elles sont spécifiées par le planificateur de déploiement de l'application.

Chacun des deux niveaux de méta-informations utilise des informations sur le contexte pour spécifier la variation des différents paramètres de déploiement.

Pour respecter cette structure à deux niveaux, nous commençons ce chapitre par présenter le modèle de description des informations de contexte commun à ces deux niveaux (section 5.2), ensuite nous détaillons le modèle de paquetage de composant qui inclut les méta-informations du niveau composant (section 5.3) suivi par le modèle de la planification du déploiement qui inclut les méta-informations du niveau application (section 5.4). Le modèle de paquetage de CADeComp est une extension du modèle de paquetage D&C. Une fois que tous les éléments du modèle ont été détaillés, nous expliquons les rôles que jouent les différents acteurs du déploiement dans la description de ces méta-informations (section 5.5). La fin de ce chapitre détaille la structure des règles qui ont été utilisées pour adapter le déploiement au contexte et analyse leur cohérence (section 5.6). La conclusion montre l'impact de la prise en compte du contexte au moment du déploiement sur le cycle de vie du développement logiciel.

Au fur et à mesure que nous détaillons le modèle de données indépendant de la plate-forme, nous donnons des exemples issus de la projection de ce modèle sur le modèle spécifique à la plate-forme CCM pour le langage XML. Ces exemples ont comme but d'illustrer et faciliter la compréhension du modèle, mais la projection spécifique à la plate-forme CCM n'est détaillée que dans le chapitre 7.

5.2 Modèle de description du contexte

Le déploiement d'une application nécessite l'identification des types de contextes qui peuvent agir sur lui, et pour chaque type de contexte, il faut identifier les états qui sont pertinents pour le déploiement de l'application. Ces informations (types de contextes et informations de contextes pertinentes) varient d'une application à une autre et dépendent de la sémantique de l'application. Nous présentons dans ce qui suit le modèle de description des informations de contexte de l'utilisateur et du domaine de déploiement ainsi que le modèle de description des informations de contexte pertinentes pour le déploiement d'une application.

5.2.1 Description des contextes de l'utilisateur

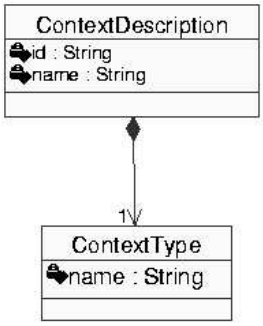


FIG. 5.1 – Structure de la description d'un contexte

Pour bénéficier d'un déploiement sensible au contexte, chaque application doit fournir une description des contextes qui peuvent avoir un impact sur son déploiement. Ces contextes présentent des contextes de l'utilisateur tels que ses préférences, les propriétés de son terminal, son identité ou son activité. La description de ces contextes est réalisée dans le but de fournir toutes les informations nécessaires à leur collecte à partir de l'intergiciel sensible au contexte sous-jacent.

La figure 5.1 montre la structure de la description d'un contexte. Chaque contexte est défini par un nom de contexte et un identifiant de contexte lui permettant d'être référencé lors de son utilisation et est associé à un type de contexte (*ContextType*).

```
<ContextDescription idref="ul"  name="userLocation">
  <ContextType name="Location"/>
</ContextDescription>
```

FIG. 5.2 – Exemple de description de contexte

5.2.2 Description du domaine de déploiement

Comme expliqué dans la section 4.3.2, nous considérons que le service de déploiement est fourni par un prestataire de service qui offre un domaine de déploiement. La figure 5.3 montre la structure de la description des ressources offertes par les différents nœuds du domaine de déploiement et leurs liens réseaux. L'ensemble de ces ressources représente un contexte, appelé contexte du domaine, qui est systématiquement pris en considération pour le déploiement de toutes les applications sans avoir besoin de les décrire à chaque fois, pour chaque application comme les contextes de l'utilisateur. Ce contexte joue un rôle important dans la détermination des nœuds sur lesquels vont être placés les composants. L'attribut *consumable* d'une ressource indique si la ressource peut être partagée par plusieurs composants. Tous les détails sur l'algorithme de placement des composants et le rôle de cet attribut sont fournis dans la section 6.2.

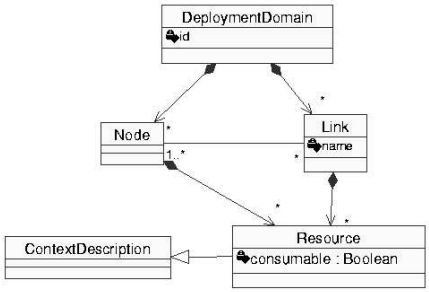


FIG. 5.3 – Domaine de déploiement

5.2.3 Description des contextes pertinents

Une fois que les types de contextes pouvant agir sur le déploiement d'une application sont identifiés, il faut déterminer les ensembles de valeurs pertinentes de ces contextes qui ont une incidence particulière sur le déploiement de l'application. Le modèle de données permettant de décrire ces contextes pertinents est représenté dans la figure 5.4.

Un contexte pertinent est modélisé par l'élément *RelevantContext*, il est associé à l'un des contextes qui agissent sur le déploiement de l'application. Il est décrit par un opérateur de comparaison ($>$, $<$, $=$, $>=$ ou $<=$) et une valeur de contexte. Il est possible de définir une combinaison de contextes pertinents (RelevantContextComposition) collectés à partir de sources différentes (une conjonction ou une disjonction de contextes pertinents).

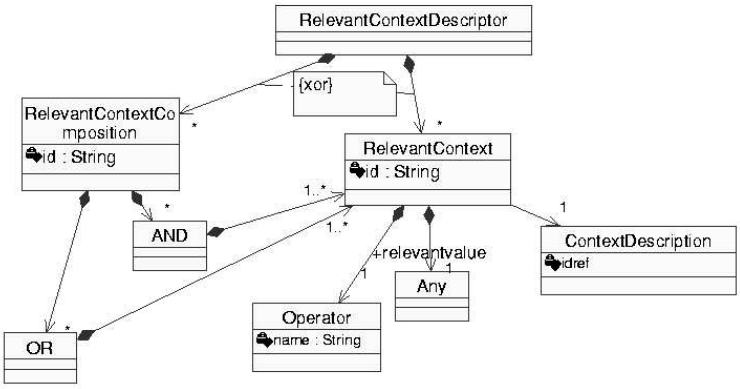


FIG. 5.4 – Contextes pertinents

```
<relevantcontext id= "PMnodeZone"  contextref="nodeZone">
  <operator  name="equal">
     <relevantvalue>PrimaryMedicalZone</relevantvalue>
  </operator>
</relevantcontext>
```

FIG. 5.5 – Exemple de description d'un contexte pertinent simple

Par exemple, si nous considérons le contexte représentant la zone où se trouve un nœud, le contexte pertinent "le nœud se trouve dans la zone médicale primaire", peut être décrit comme dans la figure 5.5.

La figure 5.6 montre un exemple de conjonction de deux contextes pertinents. Le premier contexte pertinent est décrit dans la figure 5.5 et le deuxième représente à son tour une disjonction de deux contextes pertinents qui se rapportent au contexte qui représente le système d'exploitation d'un nœud.

Les contextes pertinents jouent le rôle de contraintes sur le contexte dans les règles d'adaptation du déploiement (voir section 5.6.1).

5.3 Modèle de description de paquetages de composants

Dans la section 3.3.3, nous avons établi un cadre dans lequel nous avons identifié les méta-informations décrites pour déployer un composant. Ces informations ne permettent pas la prise en compte des informations de contexte. En se basant sur ce cadre, nous avons défini un modèle de description de paquetages de composants étendu pour le déploiement sensible au contexte (voir figure 5.7).

Comme un modèle de description de paquetage de composant classique, ce modèle décrit les informations suivantes :

– des informations descriptives générales (*GeneralDescription*) ;

```
<relevantcontextcomposition idref="PmnodeZone&Winos">
  <and>
    <relevantcontextref idref="PMnodeZone">
    <relevantcontextref idref="Winos">
   </and>
</relevantcontextcomposition>
<relevantcontextcomposition idref="Winos">
  <or>
      <relevantcontextref idref="osnameNT">
      <relevantcontextref idref="osname2000">
  </or>
</relevantcontextcomposition>
<relevantcontext id= "osnameNT"  contextref="osname">
  <operator  name="equal">
      <relevantvalue>WinNT</relevantvalue>
  </operator>
</relevantcontext>
<relevantcontext id= "osname2000"  contextref="osname">
      <operator  name="equal">
          <relevantvalue>Win2000</relevantvalue>
     </operator>
</relevantcontext>
```

FIG. 5.6 – Exemple de descripteur de contextes pertinents composés

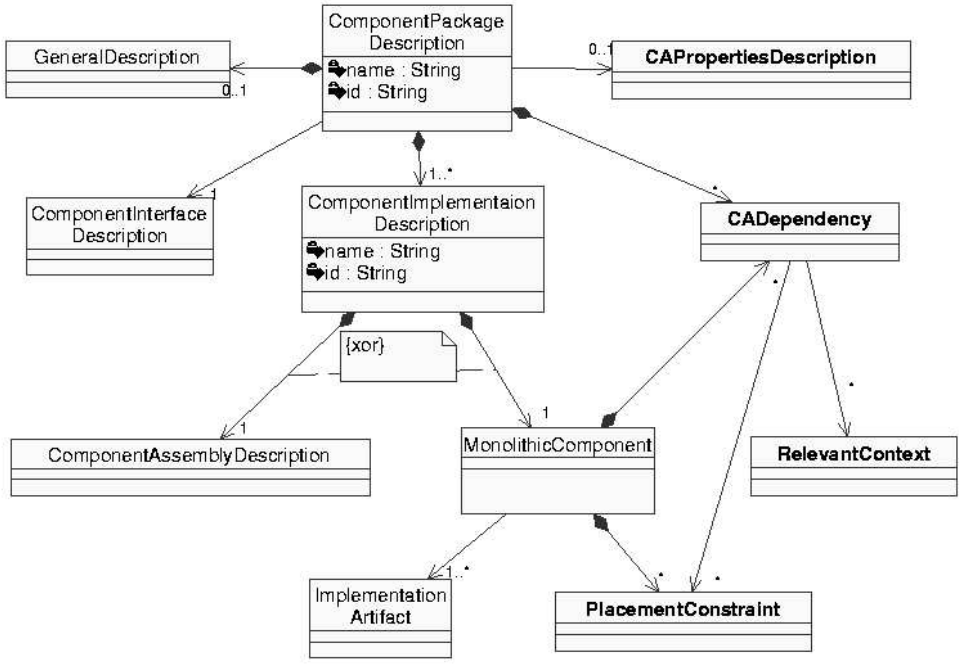


FIG. 5.7 – Modèle de paquetage de composant

– des informations structurelles (*ComponentInterfaceDescription*) ;
– la description des différentes implémentations du composant (*ComponentImplementation-Description*) ;
– des informations sur les dépendances du composant (*CADependency*) ;
– la description des valeurs de propriétés de configuration des composants (*CAProperties-Description*).

Pour prendre en considération le contexte auquel de déploiement est sensible, ce modèle décrit en plus :

– la variation des propriétés de configuration du composant en fonction du contexte (*CAPropertiesDescription*) ;
– la variation des dépendances du composant en fonction du contexte (*CADependency*) ;
– les contraintes de placement de chaque implémentation (*PlacementConstraint*).

Dans la figure 5.7, les éléments qui incluent des informations de contexte sont en gras.

Nous détaillons dans le reste de cette section les différents éléments que nous venons de citer du modèle de description du paquetage d'un composant. Comme ce modèle consiste en une extension d'un modèle de description de paquetage classique, nous avons choisi d'étendre le modèle de description de paquetage de la spécification D&C [OMG03]. Nous avons fait ce choix parce que le modèle D&C est complet et indépendant de la plate-forme. Nous commençons par décrire les éléments de base que nous avons utilisés du modèle de description de paquetage de D&C. Ensuite, nous présentons les éléments qui permettent de prendre en compte le contexte lors du déploiement d'un composant : la description des contraintes de placement, la description de la variabilité des propriétés d'un composant et la description de la variabilité de ses dépendances.

5.3.1 Eléments de base du modèle de description d'un paquetage de composant D&C

Conformément aux spécifications D&C, un composant peut être simple ou composite. L'implémentation d'un composant simple représente du code compilé (dans ce cas nous parlons d'une implémentation monolithique : *MonolithicComponent*). L'implémentation d'un composant composite représente un assemblage de composants (dans ce cas nous parlons d'implémentation d'assemblages récursifs : *ComponentAssemblyDescription*).

Un composant composite représente un ensemble d'instances de composants connectés entre eux (voir figure 5.8). Une connexion (*AssemblyConnectionDescription*) relie plusieurs ports de composants de manière à pouvoir faire circuler un message vers plusieurs ports. Il existe trois types de connexions [OMG03] :

– une connexion vers un port de composant interne à l'assemblage, externe au composant (modélisée par *ComponentExternalPortEndPoint*),
– une connexion vers un port de sous composant *interne au composant* (modélisée par *SubComponentPortEndPoint*) et
– une connexion vers un port de composant *externe à l'assemblage* via une URL (modélisée par *ExternalReferenceEndPoint*).

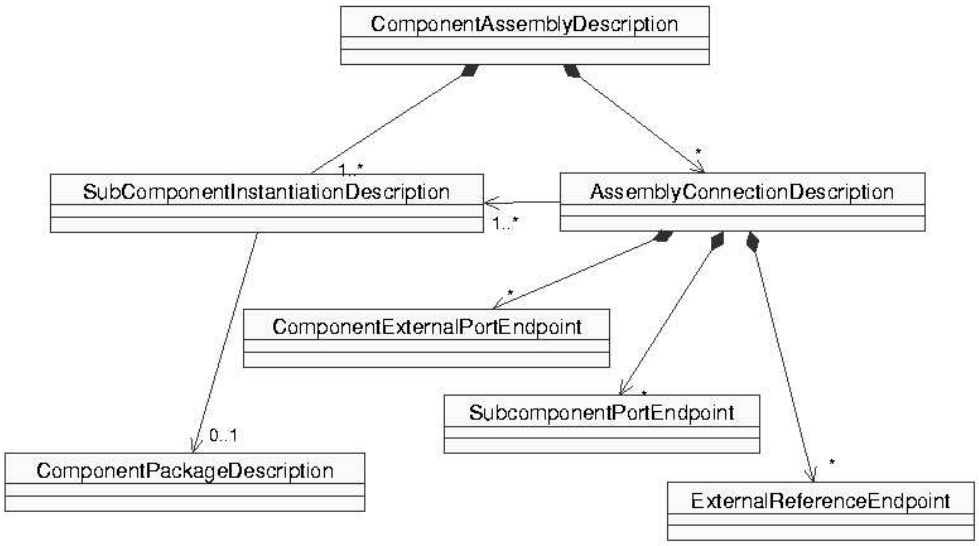


FIG. 5.8 – Structure d'un composant composite

Une interface de composant est décrite dans D&C par son type, les types supportés par héritage et un ensemble de ports (voir figure 5.9). Chaque port est décrit par son nom, les types qu'il supporte, un booléen qui définit si c'est une interface fournie ou requise, un booléen qui montre s'il y a au plus un seul fournisseur et un booléen qui montre s'il y a au plus un seul consommateur de la connexion.

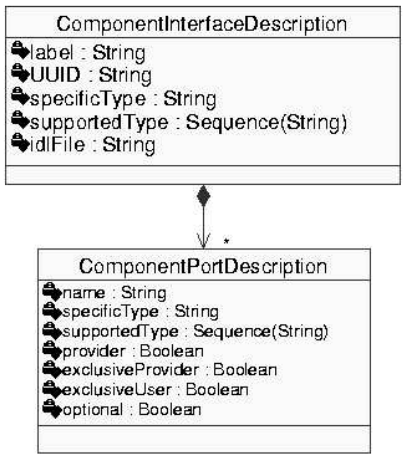


FIG. 5.9 – Description d'une interface de composant D&C

5.3.2 Contraintes de placement

Un même type de composant peut avoir plusieurs implémentations. Chacune doit posséder une description des contextes dans lesquels elle peut s'exécuter. Ce contexte d'exécution inclut aussi bien les contextes du domaine de déploiement que des contextes de l'utilisateur. Le développeur de chaque implémentation du composant spécifie les contraintes sur le contexte exigées par cette implémentation. Ces contraintes représentent des contextes pertinents étendus par deux attributs *strong* et *weight*. Le booléen *strong* spécifie si la contrainte est forte ou faible (voir figure 5.10). Une implémentation ne peut pas être sélectionnée durant la phase d'assignation d'un composant à un nœud si ses contraintes fortes ne sont pas satisfaites. La satisfaction des contraintes faibles n'est pas obligatoire mais préférable. Si la contrainte de placement représente une contrainte sur une ressource consommable, elle nécessite la spécification d'un attribut "weight" qui permet de déterminer la priorité du composant dans le partage des ressources du domaine de déploiement (voir détails dans la section 6.2). Les contraintes de placement permettent de déterminer le placement d'un composant et de choisir une de ses implémentations. La figure 5.11 montre l'exemple de description de contrainte de placement d'une implémentation d'un composant "partenaire" de l'application de gestion de crises (voir section 4.1.2). Cette implémentation est dédiée aux utilisateurs ayant un profile de sauveteur et ne peut être installée que sur un terminal ayant "Microsoft Pocket PC".

Cette description de contraintes de placement est semblable à la description de contraintes de placement de D&C et CCM excepté que ces derniers ne supportent que les contraintes sur la plate-forme sur laquelle va être placé le composant et ne spécifient que les contraintes fortes.

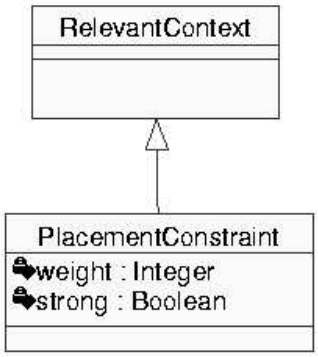


FIG. 5.10 – Description des contraintes de placement

5.3.3 Variabilité des propriétés

Un composant peut avoir zéro ou plusieurs propriétés fonctionnelles et non fonctionnelles (voir section 3.3.3). Dans notre modèle, chaque propriété peut prendre différentes valeurs selon le contexte. Le développeur du composant a la possibilité de spécifier les différentes valeurs qui peuvent être prises par chaque propriété en les associant à différents contextes pertinents (voir

```
<componentpackagedescription name="partnercomponentpackage"
                             id="partnerpack">
  <componentimplementationdescription name="rescuerimplementation"
                                      id="rescimpl">
     <monolithiccomponent>
        <placementconstraint type=strong
                             relevantcontextref="PocketPCOS">
        <placementconstraint type= strong
                             relevantcontextref="rescuerProfile">
      </monolithiccomponent>
  </componentimplementationdescription>
</componentpackagedescription>
```

FIG. 5.11 – Exemple de description de contraintes de placement

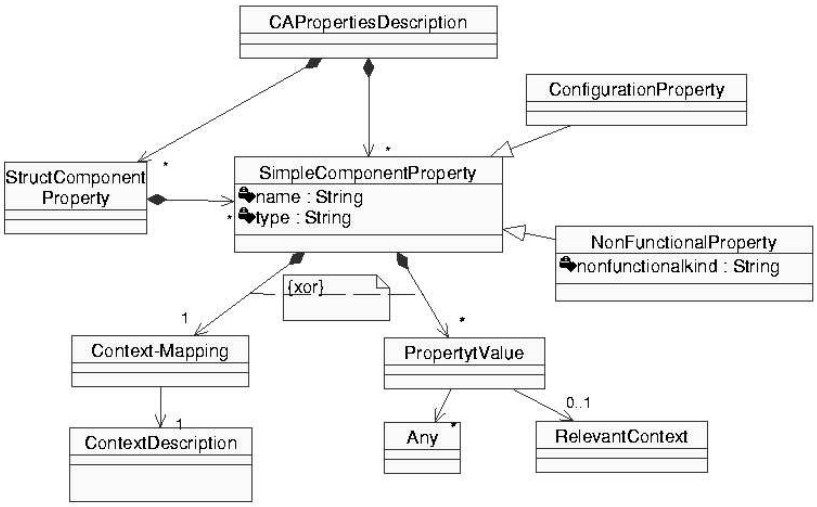


FIG. 5.12 – Variation des valeurs des propriétés en fonction du contexte

figure 5.12). Le développeur peut aussi associer directement la propriété à un contexte particulier pour que la valeur de la propriété prenne la valeur de ce contexte au moment du déploiement. Cette projection des valeurs des propriétés sur le contexte est modélisée par l'élément *Context-Mapping*. L'exemple de la figure 5.20 montre une projection de la propriété de configuration " langage de l'utilisateur " sur le contexte qui représente le langage de l'utilisateur. Une propriété peut avoir une structure simple ou complexe, dans ce dernier cas elle est considérée comme un ensemble de sous propriétés.

Une propriété non fonctionnelle nécessite la spécification d'un type non fonctionnel (*nonfunctionalkind*) comme le type transaction ou persistance. La figure 5.13 donne un exemple de propriété non fonctionnelle, celle de la persistance du composant Panier de l'application de vente en ligne (voir section 4.1.1). Cette propriété dépend de la préférence de l'utilisateur qui peut choisir lors d'un prochain achat d'utiliser le même panier (se traduit par un contexte pertinent appelé *statefulcart*) ou remplir un nouveau panier (se traduit par un contexte pertinent appelé *transientcart*).

```
<property name="pers" type="string"
                      nonfunctionaltype="persistency" >
  <propertyvalue relevantcontextref="transientcart"
                 value="transient"/>
  <propertyvalue relevantcontextref="statefulcart"
                 value="stateful"/>
</property>
```

FIG. 5.13 – Exemple de description de variabilité de propriété non fonctionnelle

5.3.4 Variabilité des dépendances

Un paquetage de composant peut avoir des dépendances sur des ressources (voir section 3.3.3). Ces dépendances peuvent varier en fonction du contexte, cela explique l'association de l'élément *CADependency* à zéro ou plusieurs contextes pertinents (voir figure 5.7). Chaque implémentation du composant peut avoir ses propres dépendances qui sont rajoutées à celles du composant. Par exemple, si au moment du déploiement sur un nœud du composant "serveur de partenaires" de l'application de gestion de crise, la bande passante de la connexion réseau de ce nœud est faible, ce composant aura une dépendance sur un logiciel qui lui permet de compresser les données avant de les envoyer (voir figure 5.14).

```
<componentpackagedescription name="serverpackage" id="serverpack">
   <cadependency  type="package" action="install"
                                 relevantcontextref="lowbandwidth">
      <fileinarchive name="zippack.rar"/>
   </cadependency>
</componentpackagedescription>
```

FIG. 5.14 – Exemple de description de variabilité de dépendances d'un composant

5.3.5 Synthèse

Dans cette section, nous avons décrit les méta-informations qui permettent de faire varier les paramètres de déploiement liés au composant : ses propriétés, ses dépendances, son implémentation et son emplacement. Nous pouvons remarquer que les composants sont développés et placés dans des paquetages pour être vendus et intégrés par la suite dans des applications que le producteur du composant ignore. Les méta-informations au niveau du composant ne prennent donc pas en compte la sémantique de l'application, en conséquence certains d'entre elles pourront être surchargées au niveau application.

5.4 Plan de déploiement sensible au contexte

Un plan de déploiement sensible au contexte représente un ensemble de méta-informations qui permettent de planifier le déploiement d'une application constituée d'un assemblage de composants monolithiques et composites en prenant en compte le contexte. Planifier le déploiement consiste à spécifier la variabilité des paramètres de déploiement associés à l'application. Un plan de déploiement sensible au contexte doit par conséquent décrire la variabilité de l'architecture de l'application en fonction du contexte. Il peut en plus définir la variabilité des paramètres associés à chaque composant en complétant ou remplaçant si nécessaire les méta-informations définis au niveau composant par des méta-informations qui prennent en compte la sémantique de l'application. Nous commençons la section par la spécification de la variabilité de l'architecture de l'application et la recherche dynamique de composants préinstanciés. Ensuite, nous détaillons la description de l'emplacement des instances de composants et la description de la variabilité des propriétés des instances de composants. Ces deux paramètres de déploiement peuvent aussi être décrits à l'aide de méta-informations communes à toutes les instances de composants d'une application. Enfin, nous décrivons la structure du plan de déploiement final que nous générons dynamiquement suite à la prise en compte du contexte.

5.4.1 Spécification de la variabilité de l'architecture de l'application

Le plan de déploiement sensible au contexte est représenté par l'élément de données *ContextAwareDeploymentPlan* (voir figure 5.15). Il décrit l'ensemble des instances de composants (*CAComponentInstance*) formant l'application à déployer, leurs connexions (*CAConnectionDescription*) ainsi que leurs paramètres de déploiement (emplacement et propriétés).

Au niveau du plan de déploiement sensible au contexte, chaque composant composite est récursivement remplacé par ses sous-composants afin de pouvoir spécifier les différents paramètres de déploiement associés à chaque instance de sous-composant.

Chaque instance de composant a un identifiant et un ordre d'exécution qui indique si cette instance va s'exécuter en parallèle avec d'autres instances de l'application. Cet attribut est utile dans le placement de l'instance de composant (voir 6.2.2).

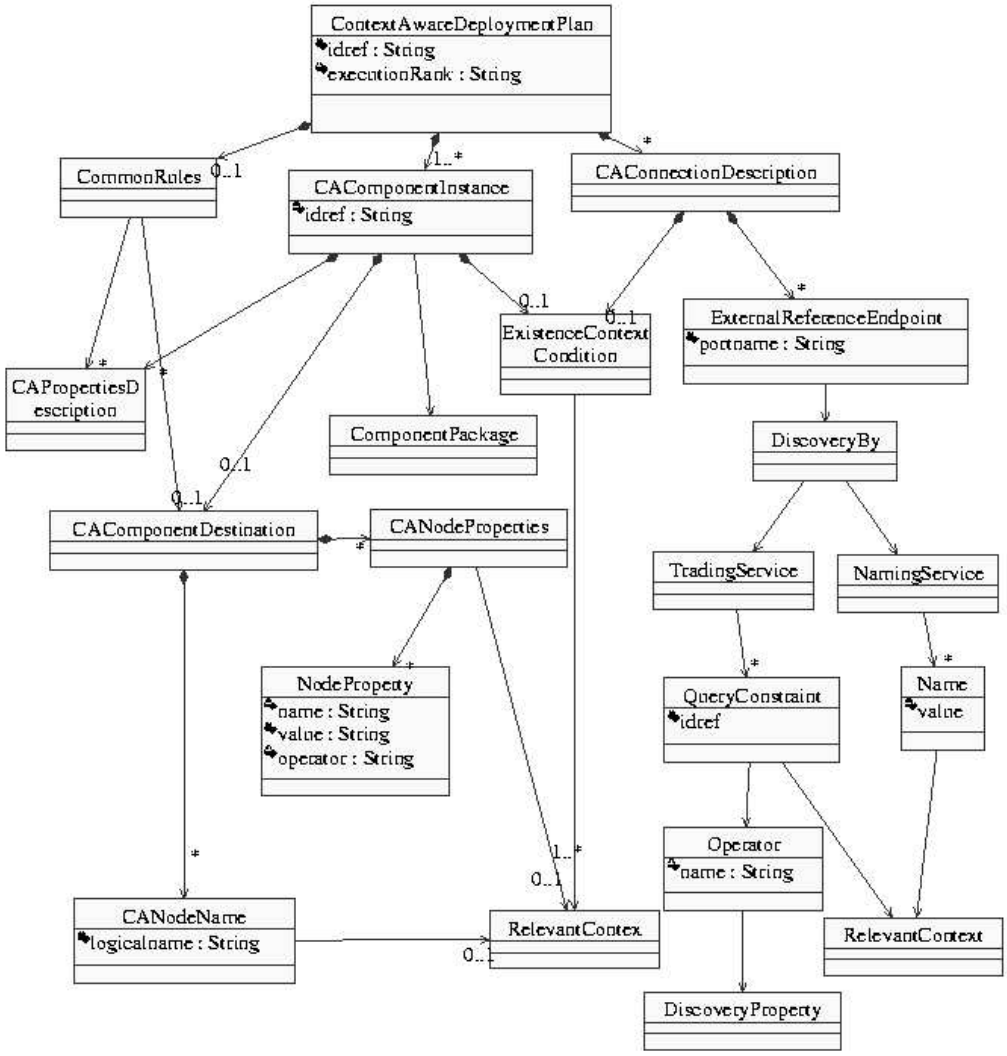


FIG. 5.15 – Plan de déploiement sensible au contexte

Pour adapter l'architecture de l'application au contexte, nous distinguons entre deux types d'instances composants et de connexions au sein d'une application : des composants et des connexions dits obligatoires, qui doivent être déployés quel que soit le contexte et d'autres optionnelles dont l'existence au moment du déploiement dépend du contexte.

Une instance de composant ou une connexion optionnelle est toujours associée à une condition d'existence modélisée par l'élément *ExistenceCondition*. Cette condition d'existence se concrétise par un ensemble de contraintes sur le contexte représenté par une disjonction de contextes pertinents (voir figure 5.15). Une instance de composant ou une connexion obligatoire n'est pas associée à une condition d'existence, en effet son existence ne dépend pas du contexte.

Ce mode de description évite la description de toutes les architectures possibles de l'application. Par contre cela peut mener à des incohérences dans l'architecture de l'application. En effet, la connexion de deux instances de composants optionnels ou d'une instance de composant obligatoire et d'une autre optionnelle n'est possible que si l'un des contextes pertinents représentant le contexte d'existence des instances de composants optionnelles est vérifié. Cela veut dire que l'établissement d'une connexion est conditionné non seulement par la vérification des contextes pertinents qui leur sont associés mais aussi par l'existence des instances de composants optionnels à connecter. La cohérence de l'architecture d'une application est décrite et étudiée dans la section 5.6.3.

La figure 5.16 montre l'exemple de description de l'instance optionnelle du composant *Vue Locale* de l'application de vente en ligne. Sa connexion optionnelle avec le composant *Base de Données* est décrite dans la figure 5.17. Cette instance de composant et cette connexion ne seront déployées que si l'utilisateur se trouve proche d'une zone non couverte par le réseau ou que l'utilisateur préfère travailler en mode déconnecté avec une vue locale.

```
<cacomponentinstance idref="LocalViewinstance">
    <existencecontextcondition>
      <relevantcontextref idref="nearUncoveredZone"/>
      <relevantcontextref idref="disconnectedMode"/>
    </existencecontextcondition>
</cacomponentinstance>
```

FIG. 5.16 – Exemple de description de composant optionnel

En plus de l'architecture de l'application, le plan de déploiement sensible au contexte permet la recherche dynamique de composants préinstanciés et spécifie si nécessaire l'emplacement et la configuration des propriétés des différents composants, en prenant en compte la sémantique de l'application.

5.4.2 Recherche dynamique de composants préinstanciés

Le plan de déploiement sensible au contexte permet l'utilisation d'un service de recherche sur propriétés [ODP93] pour trouver un composant déjà instancié en décrivant les contraintes de recherche qui peuvent inclure des informations de contexte. Par exemple, pour trouver le composant préinstancié BD représentant la base de données du magasin se trouvant dans la

```
<connectinterface idref="DB-LV">
    <existencecontextcondition>
      <relevantcontextref idref="nearUncoveredZone"/>
      <relevantcontextref idref="disconnectedMode"/>
    </existencecontextcondition>
     <usesport>
       <usesidentifier>LVinterfaceid</usesidentifier>
       <componentinstantiationref idref=" DBinstance"/>
     </usesport>
     <providesport>
       <providesidentifier>LVinterfaceid</providesidentifier>
       <componentinstantiationref idref=" LocalViewinstance "/>
     </providesport>
</connectinterface>
```

FIG. 5.17 – Exemple de description de connexion optionnelle

zone où se trouve l'utilisateur, le plan de déploiement sensible au contexte peut décrire la requête de recherche présentée dans la figure 5.18.

```
<componentinstance idref="DBinstance">
    <discoveryby><tradingservice>
        <queryconstraint idref="qconst">
          <operator value="equal">
            <property name ="DBzone"/>
            <contextmapping contextref=
                  "UserConnectionZone"/>
          </operator>
        </queryconstraint >
      <tradingservice></discoveryby>
</componentinstance>
```

FIG. 5.18 – Exemple de recherche d'un composant préinstancié

5.4.3 Placement des instances de composants

La description des contraintes de placement au niveau du paquetage d'un composant monolithique est généralement suffisante pour déterminer son emplacement (par l'algorithme décrit dans la section 6.2), seulement, dans certains cas, le placement d'une instance de composant peut nécessiter la prise en compte de la sémantique de l'application dans laquelle elle est intégrée. Dans ce cas, la specification de l'emplacement de l'instance de composant est réalisée au niveau du plan de déploiement sensible au contexte de l'application. Le composant de journalisation de l'application de gestion de crises est un exemple de composant dont l'emplacement des instances doit être spécifié au niveau du plan de déploiement sensible au contexte. En effet, le développeur crée ce composant pour être utilisé par n'importe quel composant qui a besoin de la journalisation et qui a un port compatible avec ce composant. Il spécifie pour cela ses contraintes de placement pour qu'il puisse être automatiquement placé sur un noeud qui vérifie ces contraintes. Mais, nous savons que dans le cas de l'application de gestion de crises, le

composant de journalisation ne sera déployé que s'il y a un risque de déconnexion réseau. Nous savons aussi que l'intérêt de ce composant consiste à garder une trace des activités réalisées par l'utilisateur durant sa déconnexion réseau. Dans ce cas, le composant de journalisation doit être placé au niveau du terminal utilisateur et son placement automatique sur un autre noeud satisfaisant uniquement les contraintes de placement n'aurait donc aucun sens. Dans le cas de l'application de gestion de crises, la description du placement d'une instance du composant de journalisation doit par conséquent être réalisée au niveau du plan de déploiement sensible au contexte.

```
<cacomponentdestination>
    <canodename logicalname="UserTeminal"
                relevantcontextref="SufficientResources"/>
    <canodeproperties relevantcontextref="InsufficientResources
                                          &Connected" >
            <nodeproperty name ="nodezone" operator ="equal"
                          value="userzone"/>
    </canodeproperties>
</cacomponentdestination>
```

FIG. 5.19 – Exemple de description de placement

La description de l'emplacement d'une instance d'un composant au niveau du plan de déploiement sensible au contexte se fait à travers la spécification d'un ou plusieurs noms de noeuds logiques (*CAComponentDestination*) dont chacun est associé à une contrainte de contexte représentée par un contexte pertinent. La détermination d'un nom physique qui correspond à un nom de nœud logique est réalisée au moment du déploiement grâce à des descriptions de propriétés de nœuds associées à ces noms logiques. Le prestataire de service de déploiement a, pour chaque application, une description d'un ensemble de noms logiques de nœuds qui sont associés à des propriétés qui permettent la détermination d'un nœud physique au moment du déploiement. Par exemple, le nom logique "UserTerminal" est utilisé par toutes les applications pour désigner le terminal de l'utilisateur. Le nœud physique va varier d'une utilisation à une autre et sera un paramètre de déploiement. Un exemple d'un autre nom logique est "RescuerTerminal" qui désigne un terminal de sauveteur dans l'application de gestion de crise caractérisé par l'existence d'une application de sauvetage sur son terminal.

La description du placement d'une instance de composant peut aussi se réaliser à travers la spécification d'une ou plusieurs propriétés de nœuds associées à des contextes pertinents. Ces nœuds sont dynamiquement déterminés au moment du déploiement en se basant sur la description du domaine de déploiement. L'exemple de la figure 5.19 montre la spécification du placement du composant Vue Locale qui sera placé sur le terminal de l'utilisateur si ce dernier a des ressources suffisantes. Dans le cas contraire, il sera placé sur un nœud se trouvant dans la même zone que l'utilisateur s'il n'y a pas un risque de coupure de connexion réseau.

Il est possible de ne spécifier que le nom logique ou les propriétés d'un seul noeud sans l'associer à un contexte pertinent et cela veut dire que l'instance de composant sera placée sur ce noeud quel que soit le contexte.

Lorsque le placement d'une instance de composant est spécifié au niveau du plan de déploiement sensible au contexte, l'algorithme de placement, cherche automatiquement une implémentation du composant qui puisse s'exécuter sur ce nœud. S'il n'en existe aucune, le déploiement du composant ne pourra pas être réalisé.

5.4.4 Variabilité des propriétés des instances de composants

En prenant en compte la sémantique de l'application, il est possible de découvrir d'autres valeurs que peuvent prendre les attributs des instances de composants ainsi que leurs propriétés non fonctionnelles qui varient en fonction du contexte. L'élément *CAPropertiesDescription* sera donc associé à l'instance de composant. Cet élément a une structure identique à celui défini dans le paquetage de composant (voir section 5.3.3). Cet élément surcharge les valeurs définies dans le paquetage de composant.

5.4.5 Méta-informations communes aux instances de composants

L'élément *CommonRules* du plan de déploiement sensible au contexte permet de spécifier la variation des paramètres de déploiement applicables sur toutes les instances de composants de l'application. Ces règles peuvent par exemple spécifier un emplacement général pour la totalité des instances de composants de l'application. Par exemple, un contexte donné (ressources suffisantes) peut nécessiter le déploiement de tous les composants sur le site de l'utilisateur. L'emplacement de tous les composants de l'application sera donc spécifié par une seule règle. De la même manière, tous les composants d'une application peuvent avoir des propriétés en commun. Les règles communes peuvent spécifier les valeurs possibles de ces propriétés selon le contexte. La figure 5.20 montre l'exemple de spécification de la valeur de la propriété de configuration représentant le langage utilisateur de tous les composants de l'application à déployer. Cette configuration est réalisée par la mise en correspondance de cette propriété sur le contexte représentant le langage utilisateur.

```
<commonrules>
    <cacomponentproperties>
         <property name="UserLanguage"
                   type="string">
             <contextmapping contextref=
             "UserLanguage"/>
         </property>
    </cacomponentproperties>
  </commonrules>
```

FIG. 5.20 – Exemple de règle commune

5.4.6 Plan de déploiement final

Le plan de déploiement final est généré automatiquement après étude de l'état du contexte et prise de décisions d'adaptation du déploiement. Il spécifie l'architecture de l'application qui va être finalement déployée. Il décrit ses instances de composants (*ComponentInstance*) et leurs interconnexions (*ConnectionDescription*). Il spécifie pour chacune de ces instances de composants un nom de noeud physique de placement (*InstanceDestination*), une implémentation qui va être utilisée lors de l'instanciation du composant (*ComponentImplementationDescription*), des dépendances (*Dependency*) ainsi que des valeurs fixes de ses différentes propriétés (*PropertiesDescription*). La structure du plan de déploiement final est présentée dans la figure 5.21.

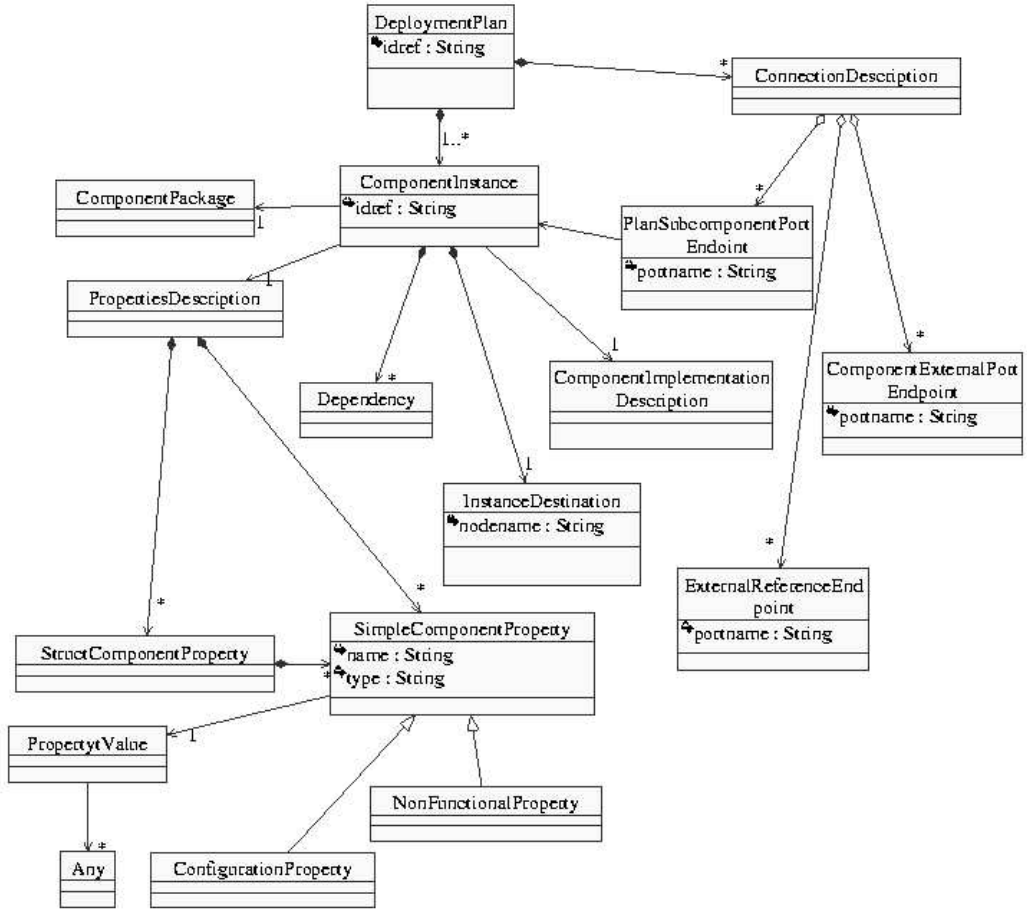


FIG. 5.21 – Plan de déploiement fi nal

5.4.7 Synthèse

Dans cette section, nous avons décrit les méta-informations qui permettent de faire varier les paramètres de déploiement liés à l'application. Ces méta-informations sont définies par un plan

de déploiement sensible au contexte qui spécifie la variabilité de l'architecture de l'application, de l'emplacement de ses instances de composants et de leurs propriétés. Seul la spécification de la variabilité de l'architecture de l'application est obligatoire au niveau du plan de déploiement sensible au contexte, la spécification de la variabilité des autres paramètres est optionelle, elle n'est réalisée qu'en cas de besoin (si la sémantique de l'application l'exige). La possibilité de description de requêtes de recherche d'instances de composants assure une dynamicité dans le déploiement. En cas de contradiction entre les méta-informations décrites au niveau du paquetage de composant et celles décrites au niveau du plan de déploiement sensible au contexte, les méta-informations du plan de déploiement sensible au contexte sont toujours prioritaires par rapport à celles du paquetage de composant puisqu'elles prennent en compte la sémantique de l'application.

5.5 Acteurs du déploiement adaptable au contexte

Nous avons vu dans les sections précédentes de ce chapitres que les méta-informations qui permettent de prendre en compte le contexte dans le processus de déploiement se situent à deux niveaux : composant et application, cela nécessite leur prise en compte aux différents stades du processus de production de l'application pour arriver à l'étape de déploiement. Chaque acteur dans la production et le déploiement du composant joue donc un rôle dans la description des méta-informations permettant la prise en compte du contexte lors du déploiement. Nous citons dans ce qui suit les différents acteurs de production des composants et les acteurs de déploiement des applications et nous décrivons leur rôle dans la prise en compte du contexte.

La production des composants consiste à développer les composants et à les mettre dans des paquetages. Le paquetage des composants fait partie du cycle de vie du déploiement. Deux acteurs réalisent la production du composant : le développeur et le constructeur de paquetages.

Nous avons deux types de développeurs de composants ; des développeurs de composants monolithiques qui créent une implémentation du composant en écrivant son code et des développeurs de composants composites qui créent une implémentation du composant basée sur un assemblage de sous-composants existants. Les développeurs de composants composites créent une description de l'implémentation en décrivant l'architecture du composant. Chacun des développeurs de composants monolithique crée une description de l'implémentation du composant en décrivant ses contraintes de placement ainsi que la variation de ses dépendances et ses propriétés.

Le constructeur de paquetages rassemble les différentes implémentations d'un même composant dans un paquetage en y intégrant leurs descriptions et crée une description du paquetage. Le paquetage est par la suite placé dans un référentiel de paquetages de composant, au niveau du prestataire de déploiement.

Une fois que les paquetages de composants sont disponibles au niveau du référentiel de paquetages de composant, le planificateur de déploiement peut créer une application à partir de ces composants en décrivant un plan de déploiement sensible au contexte. Le planificateur de déploiement est un producteur d'applications qui assemble des composants monolithiques et

composites pour former des applications en décrivant la variation de leurs paramètres de déploiement en fonction du contexte.

5.6 Règles d'adaptation du déploiement au contexte

Nous présentons dans cette section les règles d'adaptation du déploiement utilisées par CA-DeComp. Ensuite, nous montrons comment réaliser une analyse de cohérence d'un plan de déploiement sensible au contexte qui utilise ces règles à savoir la cohérence de l'architecture et la cohérence de l'assignation de valeurs aux paramètres de déploiement.

5.6.1 Structure des règles d'adaptation

Le modèle de données que nous avons proposé dans ce chapitre est basé sur un système de règles permettant de déterminer les valeurs des paramètres de déploiement. Les règles associées aux différents paramètres de déploiement sont écrites suivant le schéma suivant :
$R : condition \Longrightarrow deploymentparameter = value.$
La condition de la règle est exprimée par une contrainte sur le contexte : un contexte pertinent qui peut être simple ou présenter une conjonction ou une disjonction sur un ensemble de contextes pertinents ($condition = ContextConstraint$).

Nous distinguons deux niveaux de règles d'adaptation : des règles d'adaptation au niveau composant et des règles d'adaptation au niveau application.

Les règles d'adaptation au niveau composant

De par les paramètres de déploiement qu'ils permettent de déterminer, ces règles sont de trois types :

- règles de choix d'implémentation et de placement. Les conditions de ces règles représentent les contraintes de placement associées à chaque composant. Si le composant a n implémentations, un système de n règles ayant la structure suivante lui seront associées :
$$PlacementConstraint \Longrightarrow ComponentImpl = impl_i, i = 1..n$$
- règles de variation des propriétés fonctionnelles et non fonctionnelles. A chaque propriété du composant est associé un systèmes de règles ayant une des deux structures suivantes (n représente le nombre de valeurs possibles que peut prendre la propriété) :
$$ContextConstraint \Longrightarrow PropertyValue = value_i, i = 1..n, ou$$
$$ContextConstraint \Longrightarrow PropertyValue = ContextMapping(context)$$
- règles de variations de dépendances. Si le composant a n dépendances possibles qui dépendent du contexte, un système de n règles ayant la structure suivante lui seront associées.
$$ContextConstraint \Longrightarrow ComponentDependency = dependency_i, i = 1..n$$

Les règles d'adaptation du niveau application

Ces règles sont de trois types :

- des règles d'adaptation de l'architecture des applications à déployer en fonction du contexte. Ces règles sont définies en spécifiant les composants et les connexions optionnels et obligatoires de l'application. A chaque connexion ou composant optionnel de l'application est associé une contrainte sur le contexte qui conditionne son existence
- des règles d'adaptation de l'emplacement des instances de composant. Ces règles étant optionnelles, il est possible d'associer zéro ou plusieurs règles de placement à chaque instance de composant de l'application selon le nombre d'emplacements que veut spécifier le planificateur de déploiement

$$ContextConstraint \Longrightarrow ComponentDestination = logicalNode_i, i = 1..n$$

- règles de variation des propriétés qui ont la même structure que celles du niveau composant.

Nous étudions dans ce qui suit les cas où les règles d'adaptation du déploiement qui sont écrites par les différents acteurs du déploiement (développeur de composant ou planificateur de déploiement) peuvent contenir des incohérences. Puis, nous décrivons comment ces incohérences peuvent être détectées par un vérificateur de règles de déploiement. Le vérificateur peut avertir l'acteur de déploiement et lui proposer une description sans incohérence.

5.6.2 Cohérence du plan de déploiement final

Il est important que le planificateur de déploiement ait les outils lui permettant d'obtenir un plan de déploiement final cohérent. Ce plan est dit cohérent si la description de l'architecture de l'application à déployer est correcte et que les différents paramètres de déploiement associés à chaque instance de composant sont décrits d'une façon correcte.

Nous définissons qu'une architecture d'application est correcte lorsque chaque instance de composant a au moins une connexion qui le relie au reste des composants de l'application. Il ne faut pas qu'il y ait une instance de composant qui présente un détachement par rapport au reste des instances de composants de l'application, sinon elle ne pourrait pas être utilisée.

Pour que la description des paramètres de déploiement d'une instance de composant soit correcte, chacun de ses paramètres doit avoir au maximum une seule valeur possible au moment de la génération du plan de déploiement final. Cela veut dire que chaque instance de composant de l'application doit avoir un et un seul emplacement, une et une seule implémentation, chacune de ses dépendances doit avoir au maximum une seule valeur et chacune de ses propriétés doit avoir au une et une seule valeur pour un état de contexte donné.

A partir de cette définition d'un plan de déploiement cohérent, nous distinguons deux types d'incohérences qui peuvent être rencontrées au moment de la génération du plan de déploiement final :

- des incohérences dans la description de l'architecture de l'application ;

– des incohérences dans l'assignation d'une valeur à un paramètre de déploiement d'une instance de composant de l'application.

Nous étudions dans ce qui suit les causes de ces incohérences et nous discutons les moyens qui permettent de les éviter.

5.6.3 Cohérence de l'architecture

L'incohérence de l'architecture d'une application se traduit par le détachement d'une ou plusieurs instances de composants du reste des instances de composants de l'application. Ce détachement peut se produire au niveau des instances de composants obligatoires ou au niveau des instances de composants optionnels.

Détachement de composants obligatoires

Il est important de s'assurer que l'application est fonctionnelle dans le cas où aucune des instances de composants optionnelles ne serait déployés. Il faut pour cela vérifier qu'il n'y a pas de détachement entre les instances de composants obligatoires.

Pour ne pas avoir de détachements entre les instances de composants obligatoires dans le plan de déploiement final, il ne faut pas qu'il y ait de détachements entre les instances de composants obligatoires décrites dans le plan de déploiement sensible au contexte.

L'ensemble des instances de composants obligatoires et les connexions entre elles, décrites dans un plan de déploiement sensible au contexte, forment un graphe orienté où les nœuds du graphe représentent les instances de composants et les arcs représentent les connexions entre elles. Comme une connexion est une liaison entre un composant qui fournit une interface et un composant qui utilise cette interface, un arc est toujours orienté vers le composant qui utilise l'interface.

Pour vérifier qu'il n'y a pas d'instances de composants détachées, nous vérifions que ce graphe est connexe. Pour ce faire, nous appliquons un algorithme d'exploration de graphe orienté en profondeur d'abord [CLR90] qui explore les nœuds en commençant par le nœud qui représente la première instance de composant décrite dans le plan de déploiement sensible au contexte. Si l'algorithme arrive à explorer tous les nœuds du graphe, il n'y a pas de détachement entre les instances de composants obligatoires de l'application. Dans le cas contraire, les instances de composants correspondantes aux nœuds non explorés sont indiquées au planificateur de déploiement.

Détachement entre composants optionnels

Si l'application comporte des instances de composants optionnels, la vérification de la cohérence de la description de l'assemblage dans le plan de déploiement sensible au contexte est nécessaire afin de s'assurer que si un contexte donné amène au déploiement d'un ensemble

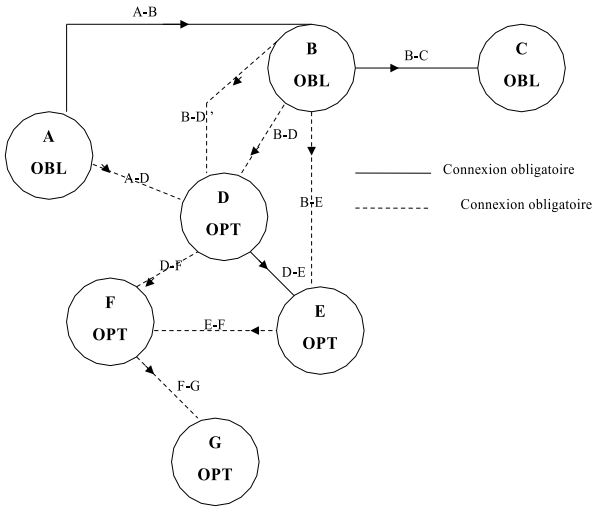


FIG. 5.22 – Exemple d'assemblage de composants

d'instances de composants optionnels, aucune de ces instances n'est détachée du reste des instances de composants (obligatoires ou optionnels) et ceci quelque soit l'état du contexte.

La cohérence de l'assemblage des instances de composants optionnels dépend de la cohérence de leurs conditions d'existence ainsi que celle de leurs connexions. Nous étudions dans ce qui suit les différentes possibilités de description d'assemblage des composants optionnels et la détection de détachement dans chacun des cas.

Si une instance optionnelle est directement connectée à un ou plusieurs instances de composants obligatoires, nous vérifions que toutes les contraintes sur le contexte qui conditionnent les connexions de l'instance optionnelle à ces instances obligatoires conditionnent aussi l'existence de cette instance. De cette manière il n'est pas possible de déployer un composant sans sa connexion. Par exemple, la condition d'existence de l'instance de composant optionnel D de la figure 5.22 doit être spécifiée comme suit :
$existence(D) \Longrightarrow existence(A-D) \vee existence(B-D) \vee existence(B-D')$, avec $existence(X-Y)$= condition d'existence de la connexion (X-Y).

Si une instance de composant optionnel est directement connectée à un ou plusieurs instances de composants optionnels, il faut vérifier que toutes les contraintes sur le contexte qui conditionnent les chemins optionnels de cette instance vers des instances de composants obligatoires conditionnent aussi l'existence de ce composant. Un chemin est un ensemble de connexions. Par exemple, la condition d'existence du composant optionnel G doit être la suivante :

$$existence(G) \Longrightarrow$$
$$[existence(F-G) \wedge existence(D-F) \wedge existence(B-D)]$$
$$\vee [existence(F-G) \wedge existence(D-F) \wedge existence(A-D)]$$
$$\vee [existence(F-G) \wedge existence(D-F) \wedge existence(B-D')]$$
$$\vee [existence(F-G) \wedge existence(E-F) \wedge existence(B-E)]$$

$\vee[existence(F-G) \wedge existence(E-F) \wedge existence(D-E) \wedge existence(B-D)]$

$\vee[existence(F-G) \wedge existence(E-F) \wedge existence(D-E) \wedge existence(B-D')]$

$\vee[existence(F-G) \wedge existence(E-F) \wedge existence(D-E) \wedge existence(A-D)]$

Pour vérifier qu'il n'y a pas de détachement d'une instance optionnelle directement connectée à des instances optionnelles, nous explorons tous les chemins optionnels qui connectent cette instance à des instances de composants obligatoires. Ensuite, nous vérifions que les contraintes sur le contexte qui conditionnent ces chemins conditionnent aussi l'existence de cette instance. Si une ou plusieurs de ces contraintes manquent, elles sont signalées au planificateur de déploiement

Il est possible de vérifier la cohérence d'autres paramètres liés à l'architecture d'une application tels que la compatibilité des ports de connexions, mais nous ne nous intéressons qu'à des incohérences causées par les règles d'adaptation du déploiement et non par la description de méta-informations du déploiement.

5.6.4 Cohérence des assignations des valeurs aux paramètres de déploiement

Nous avons une incohérence au niveau de l'assignation de valeurs aux paramètres de déploiement d'une instance de composant si dans un état de contexte donné, le système de règles attribue plus d'une valeur possible à ce paramètre. Il peut par exemple attribuer plus d'un emplacement ou plus d'une implémentation à une instance de composant ou attribuer plus d'une valeur possible à une de ses propriétés. Ce genre d'incohérence se produit lorsqu'il existe plus d'une règle qui puisse être appliquée dans un système de règles associé à un même paramètre de déploiement. Cette situation cause une contradiction entre les règles.

La possibilité d'appliquer plus d'une règle d'un système de règles associé à un même paramètre de déploiement, pour un état donné du contexte, a deux causes :

- les contraintes sur le contexte du système de règles sont reliées au même type de contexte et il y a une intersection entre ces contraintes, ou
- les contraintes sur le contexte du système de règles sont reliées à des types de contexte différents qui peuvent être vérifiés simultanément.

Nous étudions dans ce qui suit chacun de ces deux cas.

Intersection entre les contraintes

Il y a une intersection entre les contraintes de deux règles si ces contraintes se rapportent aux mêmes types de contextes et chaque sous contrainte de l'un a une intersection avec la sous contrainte qui lui correspond de l'autre règle. Considérons les deux règles suivantes :
$R1 : (50MB < diskCapacity < 80MB) \wedge (50\% < CPUuse < 70\%) \wedge (50MB <$ freeMemory $< 70MB) \Longrightarrow ComponentImpl = implementation_p$
$R2 : (30MB < diskCapacity < 55MB) \wedge (60\% < CPUuse < 80\%) \wedge (60MB < freeMemory <$

$80MB) \Longrightarrow ComponentImpl = implementation_q$

Il y a une intersection entre les contraintes de ces deux règles puisque chacune de leurs contraintes se rapportent aux mêmes types de contextes à savoir la taille du disque, l'utilisation du processeur et la taille mémoire. En plus, chaque sous contrainte de l'un a une intersection avec la sous contrainte de l'autre. Si nous avons l'un de ces états de contexte vérifiés : $50MB<disk<55MB \wedge 60\%<CPUuse<70\% \wedge$ $60MB<$memory$<70MB$, les deux règles peuvent être appliquées.

Ce genre d'incohérence est annoncé au développeur du composant ou au planificateur de déploiement, selon le niveau des règles.

Satisfaction simultanée des contraintes des règles

Si les contraintes des règles se rapportent à un même type de contexte et qu'ils ne présentent pas d'intersection, ils ne peuvent jamais être vérifiées en même temps, parce qu'un même contexte prend un seul état à un instant t donné. Ceci n'est pas le cas lorsque ces états se rapportent à des types de contextes différents comme le montre l'exemple de placement du composant Vue Locale suivant :
$R1 : (userzone = "nearnoncoveredzone") \wedge (userterminalCPUuse < 80\%) \Longrightarrow ComponentDestination = userterminal$
$R2 : (useLocalView = "true") \wedge (userterminalCPUuse > 80\%) \Longrightarrow ComponentDestination = thenearestnodetotheuser$
$R3 : (useLocalView = "true") \wedge (userterminalCPUuse < 80\%) \Longrightarrow ComponentDestination = userterminal$

Les deux contraintes sur le contexte" Localisation de l'utilisateur proche d'une zone non couverte " et " le choix de l'utilisateur d'utiliser une vue locale " associés aux deux premières règles peuvent se réaliser en même temps puisqu'elles se rapportent toutes à des contextes de types différents : localisation et préférence de l'utilisateur. La troisième règle ne peut pas être en conflit avec la première règle puisqu'elles ont toujours le même résultat et ne peut pas être en conflit avec la deuxième règle puisqu'elles se rapportent au même contexte et ne présentent pas d'intersection.

Lorsque des règles comme les deux premières règles, qui présentent des résultats contradictoires et qui se rapportent à des contextes de types différents, sont rencontrées, leurs contraintes sur le contexte doivent être complétées par la négation de toutes les autres contraintes associées aux autres règles qui peuvent se produire en même temps qu'elles. Cela revient à compléter les règles de manière qu'elles se rapportent toutes aux mêmes types de contextes et vérifier qu'elles ne présentent pas d'intersection entre elles. Dans ce cas, l'exemple précédent doit être décrit de cette manière :
$R1 : (userzone = "nearnoncoveredzone") \wedge (userterminalCPUuse < 80\%) \wedge (useLocalView = "false") \Longrightarrow userterminal$
$R2 : (useLocalView = "true") \wedge (userterminalCPUuse > 80\%) \wedge (userzone =$

$"nearcoveredzone") \Longrightarrow thenearestnodetotheuser$
$R3 : (useLocalView = "true") \wedge (userterminalCPUuse < 80\%) \Longrightarrow userterminal$

Parfois aucune des règles associées à un paramètre de déploiement n'est applicable parce qu'aucune de leurs contraintes sur le contexte n'est vérifiée, ceci ne constitue pas une incohérence pour les raisons suivantes :

- Si le paramètre représente l'existence d'une instance de composant ou d'une connexion, cela veut dire tout simplement que le contexte ne nécessite pas leur déploiement.
- S'il représente le placement d'un composant, cela veut dire que la tâche de placement sera déléguée à l'algorithme de placement (voir section 6.2).
- S'il représente une valeur d'une propriété de configuration, la propriété aura sa valeur par défaut.
- S'il représente une dépendance, cela veut dire que le contexte ne nécessite pas de dépendances.

Synthèse

Nous avons présenté dans cette section la structure des règles que nous utilisons dans le modèle de données de CADeComp et les incohérences qui peuvent être engendrées par ces règles dans le plan de déploiement final. Deux types d'incohérences peuvent survenir : des incohérences dans l'architecture de l'application ou des incohérence dans l'assignation des valeurs au paramètres de déploiement. Les incohérences de l'architecture d'une application sont causées par des anomalies dans la description des règles d'adaptation de l'architecture du niveau application. Les incohérences des assignations de valeurs aux paramètres de déploiement sont causées par des anomalies qui peuvent se situer au niveau de tous les types de règles du niveau application ou composant à l'exception des règles d'adaptation de l'architecture.

Les incohérences que nous avons présentées dans cette section sont détectées par un outil de vérification de règles qui est utilisé par le développeur du composant ou par le planificateur du déploiement à la suite de la description des méta-informations du paquetage de composant ou du plan de déploiement sensible au contexte.

5.7 Conclusion

Ce chapitre montre que l'adaptation du déploiement d'une application au contexte nécessite sa prise en compte dès les différentes phases du développement des composants de l'application (leur implémentation et leur paquetage) et se poursuit lors de la phase de la planification de déploiement. Cette prise en compte du contexte consiste en une description d'un ensemble de méta-informations nécessaires à l'adaptation du déploiement au contexte. Ces descriptions sont réalisées par le développeur du composant et le planificateur de déploiement. Le développeur du composant spécifie les contextes auxquels le déploiement du composant est sensible et la variation des paramètres de déploiement du composant en fonction du contexte (les valeurs de ses propriétés, ses dépendances et ses contraintes de placement). Le planificateur assemble

les différents composants produits par le développeur pour construire une application. Il décrit alors la variabilité de l'architecture de l'application en fonction du contexte. Le planificateur de déploiement peut aussi décrire la variation des emplacements des instances des composants ainsi que celles de leurs propriétés si la sémantique de l'application l'exige.

L'apport de ce chapitre consiste en un modèle de données des méta-informations décrites par les acteurs de déploiement : les développeurs de composants et les planificateurs de déploiement. L'avantage de ce modèle est son indépendance de la plate-forme, il est défini en se basant sur le cadre de déploiement que nous avons établi à partir de plusieurs modèles de déploiement (voir section 3.3.3). Il est basé sur des contrats qui spécifient des règles de variation des paramètres de déploiement définis dans le chapitre 4. La structure des règles est simple dans le but de simplifier la tâche des acteurs de déploiement. Les incohérences qui peuvent être engendrées dans l'écriture de ces règles ont été étudiées.

Chapitre 6

Adaptation du déploiement dans CADeComp

Nous décrivons dans ce chapitre les algorithmes d'adaptation utilisés par CADeComp ainsi que son modèle d'exécution qui s'associe au modèle de données que nous avons décrit dans le chapitre 5. Ce modèle d'exécution présente les entités qui implantent les mécanismes d'adaptation du déploiement au contexte.

Nous commençons par lister les différentes étapes d'un algorithme général d'adaptation du déploiement (section 6.1) et détailler le processus de placement des composants sur les nœuds (section 6.2). Ensuite, nous donnons le modèle d'exécution (section 6.3) avec une vue sur l'architecture de CADeComp et les interfaces des différentes entités responsables de l'adaptation du déploiement au contexte. Nous finissons le chapitre par une conclusion sur le modèle d'exécution et les algorithmes d'adaptation de CADeComp.

6.1 Étapes de l'algorithme d'adaptation du déploiement au contexte

D'après l'étude des systèmes adaptables que nous avons réalisée dans le chapitre 2, l'adaptation au contexte se réalise suite à la collecte des informations de contexte et leur analyse pour la prise de décisions d'adaptation. Nous suivons les mêmes étapes pour adapter le déploiement d'une application à base de composants au contexte. Cela donne lieu aux étapes suivantes :

1. collecte des informations de contexte et filtrage de celles qui sont pertinentes pour le déploiement de cette application ;
2. extraction des règles d'adaptation du déploiement de l'application du plan de déploiement sensible au contexte ;
3. pour chaque instance de composant décrite, dans le plan de déploiement sensible au contexte faire :

(a) vérifier son existence dans le plan de déploiement final : si le composant est obligatoire, il sera déployé quel que soit le contexte, mais s'il est optionnel, il faut vérifier si l'une des conditions de son existence est satisfaite. Si oui, l'instance est déployée, sinon elle ne l'est pas et il faut revenir au début de l'étape 3 avec l'instance suivante ;

(b) si l'instance de composant doit exister dans le plan de déploiement final, faire

i. si le placement de l'instance du composant est spécifié dans le plan de déploiement sensible au contexte, vérifier si l'une des contraintes des règles de placement est satisfaite. Si oui, assigner le placement, sinon passer à l'étape (ii) ;

ii. déterminer l'ordre d'exécution de l'instance du composant par rapport aux autres composants de l'application, pour voir si elle s'exécute en série avec d'autres instances de composants ou en parallèle ;

iii. assigner des valeurs pour les propriétés de configuration de l'instance du composant :

A. extraire les règles de variation des propriétés de l'instance du composant qui sont décrites au niveau du plan de déploiement sensible au contexte.

B. Pour chaque propriété :

– appliquer les règles décrites au niveau du plan de déploiement sensible au contexte ;

– si pour une propriété donnée, aucune des règles n'est applicable (parce que leur contrainte n'est pas satisfaite par le contexte), extraire les règles au niveau composant qui sont associées à cette propriété ;

– si aucune règle du niveau composant n'est applicable, attribuer à la propriété sa valeur par défaut ;

iv. déterminer les dépendances de l'instance du composant : extraire les règles de variation de dépendances décrites au niveau composant et les appliquer ;

4. détermination des connexions entre les composants : pour chaque connexion entre instances de composants décrite au niveau du plan de déploiement sensible au contexte, vérifier son existence. Si la connexion est obligatoire, elle est déployée quelque soit le contexte, mais si elle est optionnelle, il faut vérifier si l'une des conditions de son existence est satisfaite. Si oui, la connexion est déployée, sinon elle ne l'est pas ;

5. déterminer à l'aide d'un algorithme générique de placement, l'emplacement et l'implémentation de toutes les instances de composants qui vont être déployées et dont l'emplacement n'a pas été spécifié au niveau du plan de déploiement sensible au contexte.

Toutes ces étapes utilisent les méta-informations du modèle de données défini dans le chapitre précédent. Nous détaillons dans la section suivante la cinquième étape de cet algorithme d'adaptation du déploiement.

6.2 Placement des composants sur les noeuds

Pour affecter les composants sur des nœuds et choisir l'implémentation de chaque composant, il nous faut étudier les contraintes de placement de chaque implémentation de composant. Le placement des composants sur les noeuds se réalise en deux phases :

– la phase de recherche d'affectations implémentation/noeud valides, et
– le choix de l'affectation optimale.

6.2.1 Recherche des affectations valides

Cette phase consiste à sélectionner pour chaque composant les implémentations qui peuvent être déployées sur le domaine de déploiement en les affectant à un ou plusieurs nœuds possibles. Cette sélection dépend du contexte d'utilisation, qui doit satisfaire les contraintes fortes imposées par l'implémentation, et de la disponibilité d'au moins un nœud pouvant lui offrir les ressources dont elle a besoin.

Nous rappelons que nous avons deux types de contextes : des contextes utilisateur et des contextes du domaine (voir section 5.2). Pour chaque implémentation d'un composant, nous vérifions si toutes les contraintes fortes sur les contextes de l'utilisateur sont satisfaites. Si au minimum une seule n'est pas satisfaite, l'implémentation n'est pas sélectionnée. Dans le cas contraire, nous vérifions les contraintes fortes sur le domaine. Cette étape consiste à parcourir tous les noeuds du domaine de déploiement et vérifier si chaque ressource du noeud satisfait les contraintes fortes de l'implémentation. S'il existe au moins une ressource qui ne satisfait pas les contraintes fortes de l'implémentation, le noeud n'est pas sélectionné pour le placement de cette implémentation.

Une implémentation est donc sélectionnée si toutes ses contraintes fortes sur le contexte de l'utilisateur sont vérifiées et s'il existe au moins un nœud dont toutes les ressources satisfont les contraintes fortes sur le domaine de cette implémentation. Un nœud est sélectionné s'il existe au moins une implémentation, pour n'importe quel composant, pour laquelle il offre toutes les ressources dont elle a besoin.

À la fin de cette phase, nous pouvons trouver plusieurs placements (affectations implémentation/noeud) valides pour un même composant, comme il peut ne pas y avoir de placements valides pour un ou plusieurs composants de l'application. Si le deuxième cas se produit et que le composant est de type obligatoire, il n'existe pas de déploiement valide pour l'application. Si par contre le composant est optionnel, l'utilisateur peut choisir de continuer le déploiement du reste des composants de l'application sans ce composant.

Si à la fin de cette phase nous trouvons un seul placement valide pour chaque composant de l'application, la deuxième phase est inutile. L'objectif de cette première phase est de réduire l'espace de recherche pour le choix des affectations optimales.

6.2.2 Choix des affectations optimales

Dans le cas où nous nous trouvions avec plusieurs placements valides pour un même composant à la fin de la première phase, nous choisissons une affectation optimale en sélectionnant celle qui maximise le nombre de contraintes faibles satisfaites ou optimise l'utilisation des ressources du domaine. L'optimisation de l'utilisation des ressources du domaine revient à maximiser le nombre de ressources restantes offertes par le domaine après le placement des composants,

nous appelons ces dernières des ressources supplémentaires. Nous commençons par présenter une vue générale sur l'algorithme de placement ensuite nous détaillons le calcul des ressources supplémentaires et du nombre de contraintes faibles satisfaites.

Algorithme de placement

Aprés avoir choisi les implémentations à déployer et les nœuds d'instanciation possibles, cet algorithme permet d'affecter ces implémentations aux nœuds suivant une fonction d'évaluation f égale au pourcentage de contraintes faibles de placement (P) ou la moyenne des ressources supplémentaires (M_{supp}), selon le choix du planificateur du déploiement ou selon ce qui suit :

$$f = \begin{cases} .\mathcal{P} & \textit{si l'implémentation a des contraintes faibles} \\ .\mathcal{M}_{supp} & \textit{si l'implémentation n'a pas de contraintes faibles ou Plusieurs implémentations ont le même } \mathcal{P} \end{cases}$$

Lors de l'affectation des composants aux nœuds, nous prenons en compte les composants qui s'exécutent en parallèle pour un calcul des ressources supplémentaires proche de la réalité.

L'algorithme est présenté comme suit :

1. *Pour chaque ensemble de composants pouvant s'exécuter en parallèle faire :*
   (a) *initialiser f :* $\mathcal{P} = 0$ *et* $\mathcal{M}_{supp}$*=-1*
   (b) *Appliquer A* : Tant que au moins un composant n'est pas encore placé faire :*
      i. *CompToPlace = un des composants qui n'est pas placé*
      ii. *Pour chaque implémentation du composant faire :*
         A. *Pour chaque nœud où l'implémentation du CompToPlace peut être placée faire :*
            – *E = estimation (placer les implémentations qui ne sont pas encore placées sur les nœuds où leurs* f *est max et* $\mathcal{M}_{supp}$ *!= -1)*
            – f *= f(implémentations placées + E)*
      iii. *Sélectionner l'affectation où* f *est max*
2. *Fin*

L'algorithme, basé sur A* [Nil80], consiste à explorer l'espace de recherche en passant d'une affectation à une autre et en essayant à chaque fois de se rapprocher de l'affectation optimale en choisissant celle où f est maximale (ligne *iii*). Pour chaque itération (ligne *ii*) où certains composants ont été déjà placés, une affectation est obtenue en affectant les autres composants (E) aux nœuds qui leurs donnent un f maximal (estimation basée sur les résultats de la première étape).

[BTAB05] présente un exemple détaillé d'utilisation de l'algorithme de placement dans le cadre de l'application de vente en ligne. Nous détaillons dans ce qui suit le calcul des ressources supplémentaires (M_{supp}) et du pourcentage de contraintes faibles de placement vérifiées (P) utilisés par cet algorithme.

Calcul des ressources supplémentaires

L'optimisation de l'utilisation des ressources du domaine consiste à maximiser les ressources supplémentaires offertes par les nœuds. Nous distinguons deux types de ressources :

- des ressources consommables qui peuvent être partagées par plusieurs composants telles que la taille du disque et la taille de la mémoire. La valeur de ces ressources varie en fonction des composants installés sur les nœuds, c'est pourquoi on dit que ces ressources se consomment ;
- des ressources non consommables comme un écran, elles n'ont pas une valeur quantifiable, mais un état d'existence.

La ressource supplémentaire $R_{supp}(r, i)$ offerte par une ressource consommable r pour une implémentation i d'un composant, est égale à la valeur de la ressource offerte par r moins la valeur de la ressource requise par i. Une ressource non consommable n'offre pas de ressource supplémentaire.

Pour calculer une valeur des ressources supplémentaires offertes par un nœud proche du réel, nous prenons en compte les composants qui s'exécutent en parallèle après leur placement sur un même nœud. Les implémentations qui s'exécutent en parallèle (sur un même noeud) doivent donc être placées comme s'ils constituaient un seul élément, pour cela nous les regroupons dans une seule entité qui peut être considérée comme une nouvelle implémentation dont les contraintes sont l'union de celles des implémentations la constituant. Nous appelons cette entité *granule*.

Nous avons expliqué dans 5.3.2 que chaque implémentation d'un composant a une contrainte de placement. Si cette contrainte représente une contrainte sur une ressource consommable, elle exprime la valeur minimale de la ressource requise par l'implémentation. La contrainte de placement a un poids qui détermine la priorité dans le partage des ressources. Ce poids représente le poids de la ressource pour l'implémentation.

La valeur de la ressource supplémentaire R_j offerte par le nœud N à l'implémentation I est donnée par la formule suivante :

$$\mathcal{R}_{supp}(R_j, N, I) = \frac{\mathcal{P}_j * \mathcal{R}_{supp}(R_j, N, G)}{\sum_{k=1}^{m} \mathcal{P}_{k_j}}$$

m : le nombre d'implémentations partageant la ressource R_j,

$\mathcal{P}_{kj}$: *le poids de la ressource* R_j *défini par l'implémentation* I_k,

$\mathcal{R}_{supp}(R_j, N, G)$: *la valeur de la ressource supplémentaire* R_j *offerte par le nœud* N *au granule* G *qui contient l'implémentation* I*, donnée par la formule suivante :*

$$\mathcal{R}_{supp}(R_j, N, G) = \mathcal{R}_{off} - \sum^{G} \mathcal{R}_{min}$$

$\mathcal{R}_{off}$ *: la ressource offerte par* N*,*

$\mathcal{R}_{min}$ *: la valeur minimale de la ressource requise par* G*.*

La moyenne des ressources supplémentaires offertes par le nœud N à l'implémentation I est donc donnée par la formule suivante :

$$\mathcal{M}_{supp}(N, I) = \frac{\sum_{j=1}^{\mathcal{N}bResCons} \mathcal{P}_j * \mathcal{R}_{supp}(R_j, N, I)}{\sum_{l=1}^{\mathcal{N}bResCons} \mathcal{P}_l}$$

$\mathcal{N}bResPart$ *: le nombre de ressources consommables.*

$\mathcal{P}_j$ *: le poids de la ressource* R_j *défini par l'implémentation I.*

$\mathcal{R}_{supp}(R_j, N, I)$ *: la valeur de la ressource supplémentaire* R_j *offerte par le nœud N à l'implémentation I*

Supposons par exemple que deux composants C_1 et C_2 doivent s'exécuter simultanément et qu'ils ont été affectés sur un même nœud avec les implémentations respectives I_1 et I_2. Si l'espace disque offert par le nœud est égal à 100 MO, et que chaque implémentation a respectivement besoin de 10 et 20 MO au minimum, le granule sera défini par la contrainte : *espace disque* ≥ 30. L'espace disque supplémentaire total est égal à :

$$\mathcal{R}_{supp}(disque, N, G) = 100 - (10 + 20) = 70.$$

Si nous commençons par placer I_1, nous devons prendre en compte le placement de I_2 qui va suivre. L'espace disque supplémentaire pour I_1, dépend du poids de la ressource pour I_1, si ce poids est par exemple égal à 6 pour I_1 et 4 pour I_2, la ressource supplémentaire pour I_1 et I_2 sera égale à :

$$\mathcal{R}_{supp}(disque, N, I_1) = \frac{6 * 70}{(6 + 4)} = 42$$

$$\mathcal{R}_{supp}(disque, N, I_2) = \frac{4 * 70}{(6 + 4)} = 28$$

Calcul du pourcentage de contraintes faibles satisfaites

Le pourcentage de contraintes faibles satisfaites est aussi déterminé en fonction des poids de ces contraintes, comme suit :

$$\mathcal{P}(D,E) = \frac{\sum_{i=1}^{card(E)} \sum_{j=1}^{\mathcal{N}bWeakConst} \mathcal{P}_{ij} * \mathcal{V}erifier(i,j)}{\sum_{k=1}^{card(E)} \sum_{l=1}^{\mathcal{N}bWeakConst} \mathcal{P}_{kl}}$$

E : l'ensemble des implémentations et des connexions

D : l'ensemble des noeuds du domaine

$\mathcal{P}(D,E)$ *: le pourcentage de contraintes faibles vérifiées.*

$\mathcal{P}_{ij}$ *: le poids de la contrainte faible j défini pour l'implémentation i*

$\mathcal{N}bWeakConst$ *: le nombre de contraintes faibles de l'implémentation i*

$\mathcal{V}erifie(i,j) = 1$ *si la contrainte faible j est satisfaite pour l'implémentation i*

$\mathcal{V}erifie(i,j) = 0$ *si la contrainte faible j n'est pas satisfaite pour l'implémentation i*

Suite à l'étude des différentes étapes d'adaptation du déploiement au contexte, nous détaillons dans la section qui suit le modèle des entités qui assurent ces différentes étapes.

6.3 Modèle d'exécution du déploiement sensible au contexte

Contrairement au modèle de données qui est un modèle d'informations descriptives, le modèle d'exécution modélise les entités d'exécution qui permettent de traiter, créer ou offrir ces informations. Nous commençons la section par une vue d'ensemble sur l'architecture de CADeComp. Ensuite, nous présentons les deux entités qui traitent les méta-informations de l'adaptation du déploiement à base de composants : le *Filtre de Contexte* et l'*Adaptateur de Déploiement*. Enfin, nous présentons le processus d'adaptation.

6.3.1 Vue sur l'architecture de CADeComp

La figure 6.1 représente une vue sur l'architecture de la plate-forme de déploiement sensible au contexte. La partie gauche de la vue présente les entités devant être disponibles sur le terminal utilisateur alors que la partie droite présente les entités fournies par un prestataire de service de déploiement.

Comme nous l'avons expliqué dans la section 4.3.2, CADeComp fait partie d'une plate-forme de composants et se situe au-dessus d'un intergiciel sensible au contexte.

Le prestataire de service de déploiement offre un ensemble de noeuds que nous appelons aussi des serveurs d'exécution qui permettent d'héberger les composants qui n'ont pas été placés sur le terminal utilisateur lors d'un déploiement à la volée. Ces noeuds disposent tous d'un intergiciel de composants et d'un intergiciel sensible au contexte. L'ensemble de ces nœuds représente le domaine de déploiement.

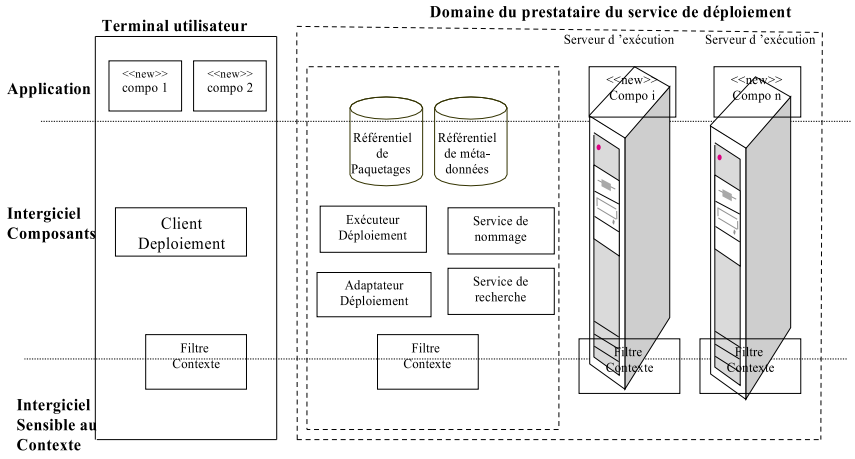


FIG. 6.1 – Vue sur l'architecture de CADeComp

L'entité Exécuteur du Déploiement (*DeploymentExecutor*) représente l'outil de déploiement de base offert par l'intergiciel orienté composants. Il offre les activités de déploiement que nous avons défini dans la section 3.3.3 et ne supporte pas l'adaptation au contexte.

Pour déployer une application, l'utilisateur doit disposer d'un intergiciel de composants, d'un intergiciel sensible au contexte ainsi que d'un client de déploiement sur son terminal.

Le Client de Déploiement (*Deployment Client*) est une entité qui initie le déploiement d'une application suite à la demande de l'utilisateur ou d'un autre programme qui remplace l'utilisateur.

Pour lancer le déploiement d'une application (son installation et son activation), le Client de Déploiement envoie une requête de déploiement vers l'Adaptateur de déploiement (*DeploymentAdapter*) en lui indiquant le nom de l'application à déployer, le nom du terminal de l'utilisateur ainsi que les préférences de l'utilisateur. A la réception de cette requête, l'Adaptateur considère le terminal de l'utilisateur comme faisant partie du domaine de déploiement pour le déploiement de cette application.

Le fournisseur du service de déploiement dispose de deux référentiels : le référentiel de paquetages de composants et le référentiel de méta-données d'applications.

Le référentiel de paquetages contient les paquetages de composants des applications à déployer. Un paquetage de composant contient différentes implémentations du composant (chaque implémentation est dédiée à un environnement d'exécution et des contextes particuliers), les méta-informations nécessaires au déploiement du composant (voir section 5.3) ainsi que les méta-informations qui décrivent les contextes et les contextes pertinents pouvant agir sur le déploiement du composant (voir section 5.2).

Le référentiel de méta-données contient les méta-informations nécessaires au déploiement des applications offertes par le prestataire de service (ces applications seront assemblées à partir des composants dont les paquetages se trouvent au niveau du référentiel des paquetages de composants). Ces méta-informations représentent les méta-informations de contexte, les contextes pertinents auxquels le déploiement de l'application est sensible et les informations sur la planification du déploiement de l'application (voir section 5.4).

Comme nous plaçons CADeComp au-dessus d'un intergiciel sensible au contexte, ils nous faut exprimer nos besoins en terme de collecte et gestion d'informations de contexte afin de choisir l'intergiciel qui répond à nos besoins. CADeComp adapte le déploiement à des contextes variées de bas et de haut niveau. L'intergiciel sensible au contexte utilisé par CADeComp doit alors permettre de collecter différents types de contextes de bas niveau à partir de sources différentes et de déduire des informations de contexte de haut niveau à partir des contextes de bas niveau. Les deux modes de collecte d'informations de contexte par interrogation et par notification doivent être supportés (pour la reconfiguration de l'application).

L'adaptation du déploiement au contexte dans CADeComp est assuré par deux entités principales : le Filtre de Contexte (*ContextFilter*) et l'Adaptateur de Déploiement. Les modèles de ces entités sont présentés dans ce qui suit.

6.3.2 Filtre de contexte

Le *Filtre de Contexte* filtre les états de contextes pertinents pour le déploiement d'une application donnée. Grâce à lui, l'information de contexte ne circule sur le réseau que si elle est pertinente pour le déploiement de l'application. Cela économise la bande passante réseau. Le filtre de contexte peut être utilisé selon les deux modes : par interrogation et par notification.

Dans le mode par interrogation, l'entité faisant appel au filtre (l'*Adaptateur de déploiement*), lui fournit la description de l'état ou des états de contexte qui lui sont pertinents. Le filtre s'adresse à l'intergiciel sensible au contexte par interrogation pour collecter les contextes qui intéressent l'entité appelante et vérifie ensuite si les valeurs de contexte récupérées correspondent aux états recherchés par l'entité appelante.

Dans le mode par notification, le filtre s'enregistre auprès de l'intergiciel sensible au contexte selon le mode par notification pour collecter les contextes qui intéressent l'entité appelante et à chaque notification d'une valeur de contexte, le filtre compare l'état actuel du contexte aux états recherchés par l'entité appelante. S'ils sont vérifiés, il notifie l'application.

Chaque nœud du domaine de déploiement doit disposer d'un filtre puisque nous devons collecter et filtrer des contextes à partir de chaque noeud. L'interface du filtre est distribuée. Une entité distante peut donc interroger le filtre ou s'inscrire auprès de lui.

Les opérations offertes par le filtre de contexte sont montrées dans la figure 6.2.

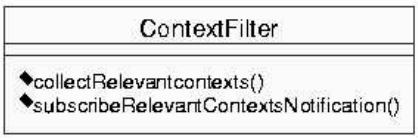


FIG. 6.2 – Diagramme de classe UML du Filtre de Contexte

Les opérations *collectRelevantContext()* et *subscribeRelevantContextsNotifications()* filtrent les états de contexte pertinents respectivement selon le mode par interrogation et par notification. Ces deux opérations utilisent une description des informations de contexte que l'intergiciel sensible au contexte doit collecter et une description des contextes pertinents qu'ils doivent filtrer.

Pour l'instant, CADeComp ne traite que le déploiement initial des applications à base de composants c'est-à-dire l'installation sensible au contexte sans reconfiguration dynamique. Seul le mode par interrogation est utilisé pour le filtrage et la collecte des informations de contexte. Cependant l'architecture a été prévue pour être étendue à la reconfiguration.

6.3.3 Adaptateur de déploiement

L'*Adaptateur de Déploiement* est le cœur du service de déploiement sensible au contexte, il est responsable de toutes les décisions d'adaptation du déploiement. A la réception d'une requête d'un *Client de Déploiement* pour le déploiement d'une application donnée, l'*Adaptateur de*

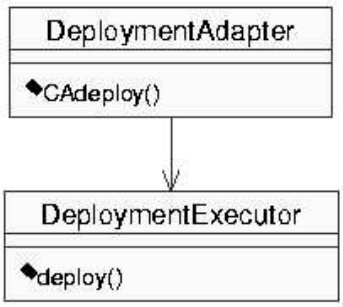


FIG. 6.3 – Adaptateur de déploiement

Déploiement récupère les méta-informations associées à cette application à partir du référentiel de méta-données (son plan de déploiement sensible au contexte, la description des contextes et des contextes pertinents). Il utilise ensuite le plan de déploiement sensible au contexte pour identifier les paquetages de composants du référentiel de paquetages qui vont être utilisés lors du déploiement de l'application. L'*Adaptateur de Déploiement* étudie tous les contextes pertinents qui sont décrit au niveau de l'application et au niveau de chaque paquetage de composant et fait la distinction entre deux types de contextes pertinents : des contextes pertinents se rapportant à des contextes du domaine et des contextes pertinents se rapportant à des contextes de l'utilisateur (tout contexte pertinent se rapportant à une ressource matérielle ou logicielle est considéré comme contexte pertinent se rapportant à un contexte du domaine). L'adaptateur de Déploiement utilise le Filtre du terminal utilisateur pour filtrer les contextes pertinents de l'utilisateur et interroge tous les filtres du domaine en envoyant à chacun d'eux la description des contextes pertinents du domaine. Chaque filtre va interagir avec l'intergiciel sensible au contexte pour filtrer les informations pertinentes pour le déploiement de l'application.

L'Adaptateur de Déploiement analyse les contextes pertinents collectés à partir des Filtres de contexte pour décider les instances de composants qui constituent l'application, les connexions entre ces instances, la version d'implémentation, l'emplacement, les dépendances et les propriétés de configuration et non fonctionnelles de chaque instance de composant et produire ainsi un plan de déploiement final de l'application. Le plan de déploiement ainsi produit est envoyé vers L'Exécuteur de Déploiement représentant un outil de déploiement classique qui se charge de déployer l'application (voir figure 6.3).

Les décisions de l'adaptateur sont basées sur l'algorithme décrit dans la section 6.1.

6.3.4 Processus d'exécution de l'adaptation du déploiement

L'Adaptateur de Déploiement comporte un Adaptateur d'architecture (*ArchitectureAdapter*), un Adaptateur de Placement (*PlacementAdapter*), un Adaptateur de Propriétés de Configurations (*PropertyConfigurationAdapter*) et un Gestionnaire de méta-informations (*MetaInformationMana-*

ger). La figure 6.4 présente le diagramme de classes UML reliant ces différentes entités. Nous expliquons d'une façon brève le rôle de ces différents éléments.

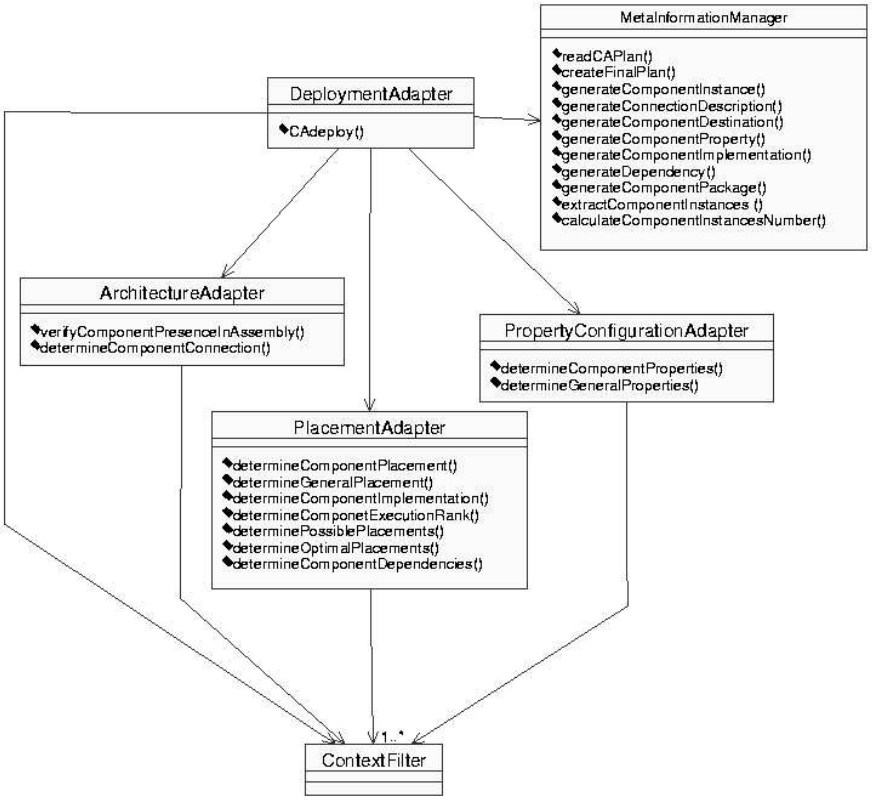


FIG. 6.4 – Structure de l'adaptateur de déploiement

L'Adaptateur d'Architecture applique les règles d'adaptation de l'architecture, il utilise les contextes pertinents collectés pour déterminer l'architecture de l'application à savoir ses composants et ses connexions. Pour ce faire, il a deux opérations qui parcourent le plan de déploiement sensible au contexte et identifient les composants et les connexions obligatoires ainsi que les composants et les connexions optionnels dont la condition d'existence est vérifiée. Ces composants seront générés par l'Adaptateur d'Architecture dans le plan de déploiement final.

L'Adaptateur de propriétés assigne les valeurs aux propriétés d'une instance de composant donnée. Il prend pour cela en considération les règles générales et spécifiques décrites au niveau du plan de déploiement sensible au contexte, les différentes règles d'adaptation de propriétés décrites au niveau du paquetage de composant et les valeurs des propriétés par défaut.

L'Adaptateur de placement applique l'algorithme de placement décrit dans la section 6.2. Il a des opérations lui permettant de :

– déterminer un placement général pour tous les composants de l'application (*determineGeneralPlacement()*), s'il y a un placement général qui a été spécifié dans le plan de déploiement sensible au contexte.

– déterminer l'emplacement et l'implémentation de tous les composants dont l'emplacement a été spécifié dans le plan de déploiement sensible au contexte (*determineComponentImplementation()*).
– déterminer les emplacements possibles puis l'emplacement optimal des différents composants à déployer si leur emplacement n'a pas été spécifié au niveau du plan de déploiement sensible au contexte (*determinePossiblePlacements()*).

Le Gestionnaire de méta-informations offre à l'Adaptateur de Déploiement les opérations lui permettant d'extraire le contenu du plan de déploiement sensible au contexte et des méta-informations qui décrivent chaque composant de l'application ainsi que de créer et générer le plan de déploiement final une fois que toutes les décisions de déploiement sont prises.

Le diagramme de séquences qui montre l'interaction entre ces différents éléments dans le but de réaliser le processus d'adaptation du déploiement, selon l'algorithme décrit dans la section 6.1, est présenté dans la figure 6.5.

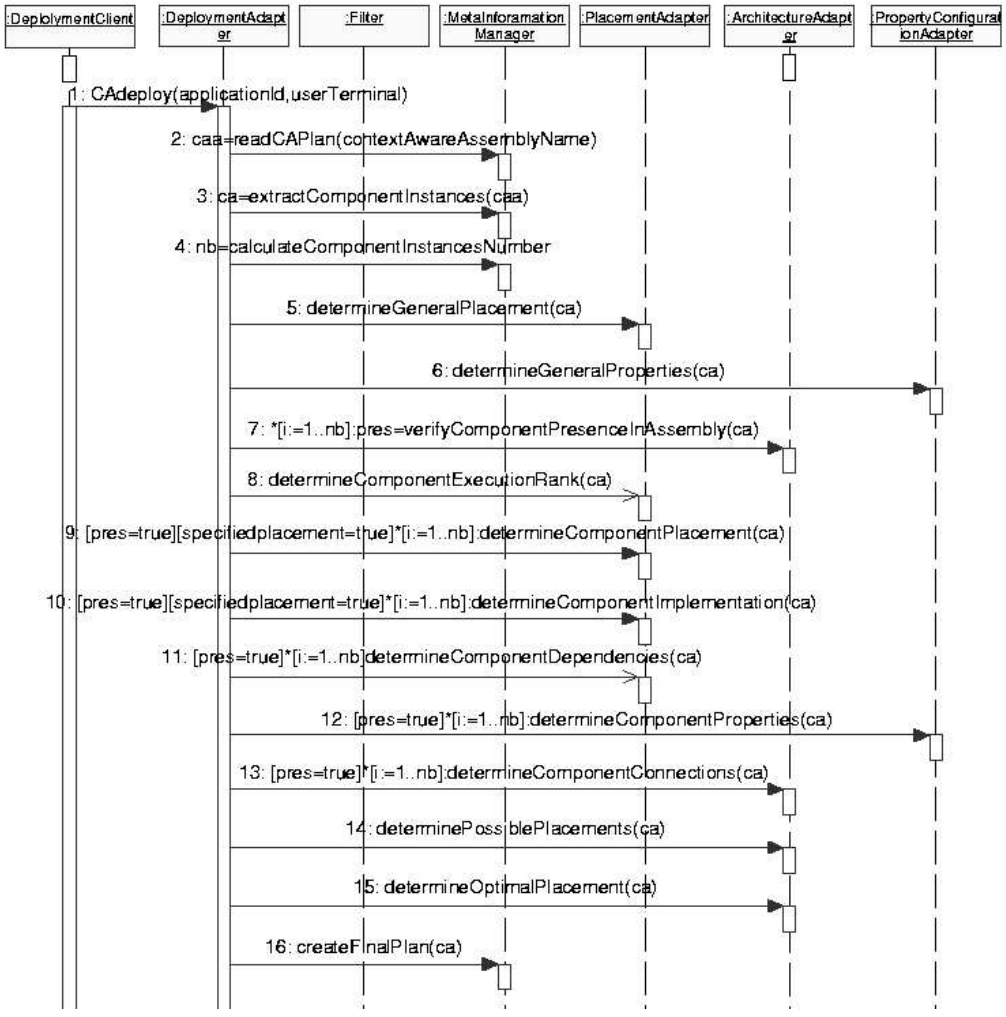


FIG. 6.5 – Diagramme de séquences du déploiement sensible au contexte

6.4 Conclusion

Nous avons présenté dans ce chapitre les éléments de l'architecture de CADeComp en détaillant les différents mécanismes, étapes et algorithmes utilisés pour adapter le déploiement au contexte.

Les différentes étapes d'adaptation de déploiement consistent principalement à collecter et filtrer les informations de contexte pertinentes pour le déploiement, puis à analyser ces informations afin de déterminer les valeurs des cinq paramètres de déploiement que nous avons identifiés dans le chapitre 4.

L'architecture de CADeComp est basée sur un ensemble d'entités conçus d'une façon indépendante de la plate-forme. Ces entités peuvent être rajoutées au-dessus de n'importe quel outil de déploiement classique. Ces entités réalisent les différentes étapes de déploiement en se basant sur les méta-informations dont le modèle a été défini dans le chapitre 5 et un algorithme d'adaptation.

Comme le placement automatique des composants sur le domaine du déploiement est indépendant de la sémantique des applications à déployer, nous avons défini un algorithme générique pour accomplir cette tâche. Cet algorithme permet de trouver une solution pour le placement des composants qui s'approche d'une solution optimale qui maximise les ressources disponibles du domaine de déploiement après le placement de ces composants ou le nombre de contraintes faibles satisfaites. La particularité de l'algorithme de placement par rapport aux solution existantes de placement est sa capacité de prise en compte de différents types de contextes : non seulement il prend en compte le contexte qui représente la plate-forme matérielle et logicielle sur laquelle vont être déployés les composants mais aussi le contexte général de l'utilisateur.

Le principe général de l'algorithme d'adaptation proposé, utilise les informations fournies par un intergiciel sensible au contexte et permet une grande souplesse quant à la variabilité du déploiement en enrichissant un processus de déploiement classique. Cet algorithme a la possibilité d'assigner des valeurs par défaut aux paramètres de déploiement, de les calculer ou déterminer leur valeur à l'aide de règles d'adaptation prédéfinies.

Troisième partie

Modèle spécifique à la plate-forme CCM et implémentation

Chapitre 7

Mise en œuvre dans CCM, évaluation et expérimentation

Pour définir les modèles de données et d'exécution de CADeComp, nous nous sommes basés sur le cadre de déploiement défini dans la section 3.3.3 qui a été établi à partir de plusieurs modèles de déploiement. Cela constitue une base pour la généricité de notre modèle et pour son indépendance vis à vis de la plate-forme. Pour pouvoir utiliser CADeComp sur une plate-forme particulière, il faut le projeter sur celle-ci. CADeComp nécessite une projection sur un intergiciel composant et une projection sur un intergiciel sensible au contexte.

Nous commençons ce chapitre par décrire la projection du modèle de CADeComp, indépendant de la plate-forme, sur CCM et sur l'intergiciel sensible au contexte que nous avons développé (7.1). La description de la projection est suivie par une proposition d'une implémentation de CADeComp sur OpenCCM, une implémentation de la spécification CCM (7.2). Nous décrivons ensuite des outils associés à CADeComp que nous avons implémentés. Enfin, nous présentons une expérimentation permettant d'évaluer CADeComp(7.4).

7.1 Projection du modèle de CADeComp

Cette section présente la projection des modèles de données et d'exécution de CADeComp (PIM : Platform Independent Model) sur des modèles spécifiques à l'intergiciel orienté composant (CCM) et l'intergiciel sensible au contexte que nous avons développé (PSM : Platform Specific Model).

La projection concrétise les méta-concepts abstraits du modèle de CADeComp et les aligne aux différents éléments et classes des intergiciels choisis. Elle consiste à étudier les spécificités du modèle de chaque intergiciel et leurs répercussions sur le modèle de CADeComp. Le modèle de CADeComp peut être sur-spécifié ou sous-spécifié par rapport aux services offerts par ces intergiciels. Dans le premier cas, nous ne considérons pas les classes qui présentent une sur-

spécification en restreignant le modèle aux classes dont nous avons besoin. Dans le deuxième cas, nous étendons le modèle avec des classes spécifiques aux modèles des intergiciels.

Nous commençons par décrire la projection du modèle de CADeComp sur CCM. Ensuite, nous décrivons sa projection sur le collecteur de contextes, l'intergiciel sensible au contexte que nous avons développé. Enfin, nous définissons un ensemble de règles pour la génération automatique du PSM CADeComp.

7.1.1 Modèle de CADeComp spécifique à CCM

Nous avons choisi de projeter le modèle de CADeComp sur le modèle d'intergiciel composant CCM parce qu'il nous a semblé être le modèle de déploiement le plus complet (voir section 3.4.7). La projection du modèle de CADeComp sur CCM concerne les éléments de CADeComp ayant un rapport avec l'outil de déploiement de l'intergiciel composant tels que le modèle de description de paquetages de composants (voir section 5.3), le plan de déploiement sensible au contexte (voir section 5.4), le plan de déploiement final (voir section 5.4) et l'adaptateur du déploiement (voir section 6.3). Nous ne décrivons dans ce qui suit que les éléments nécessitant un alignement avec CCM qui sont le modèle de description de paquetages de composants et le plan de déploiement final. Le plan de déploiement sensible au contexte et l'adaptateur de déploiement ne nécessitent aucune modification dans le cadre de la projection CCM parce qu'ils sont à un niveau d'abstraction plus haut par rapport à la plate-forme.

Modèle de description de paquetages de composants

Le modèle de composant CCM actuel ne supporte pas les composants composites. L'implémentation d'un composant ne peut donc être que monolithique. Cela nous amène à omettre la classe *ComponentAssemblyDescription* (voir figure 5.7) et réduire le modèle de description de paquetage de composant de CADeComp pour CCM à celui le la figure 7.1.

Dans le cadre de CCM, la description des interfaces et des ports d'un composant (voir figure 5.9) doit être étendue comme le montre la figure 7.2. Un attribut *idlFile* est rajouté à la description d'une interface pour spécifier une référence sur le fichier IDL du composant. Un type est additionné à la description d'un port d'un composant ou le port d'une connexion au moment de l'assemblage. Ce type spécifie le type de port concret utilisé (facette, réceptacle, source d'événements, puit d'événement, etc.) [OMG03].

Plan de déploiement final

En alignant le plan de déploiement final généré par l'adaptateur de déploiement avec les descripteurs définis dans CCM, nous trouvons que le descripteur d'assemblage CCM (Component Assembly Descriptor ayant l'extension ".cad") décrit dans 3.4.1 contient les différents éléments spécifiés dans le plan de déploiement final. Par contre le descripteur d'assemblage a une structure différente par rapport à celle définie par le plan de déploiement final (voir figure 5.21). Le

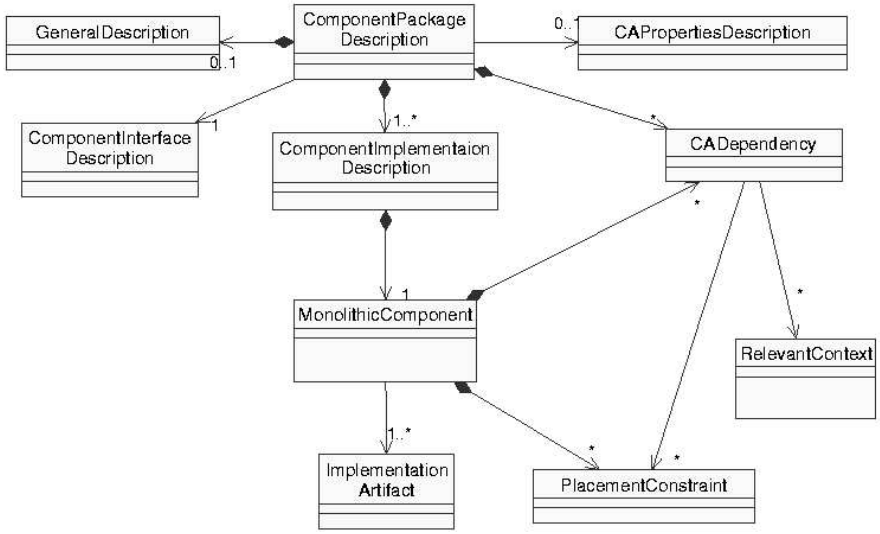


FIG. 7.1 – Projection CCM du modèle de paquetage de composant

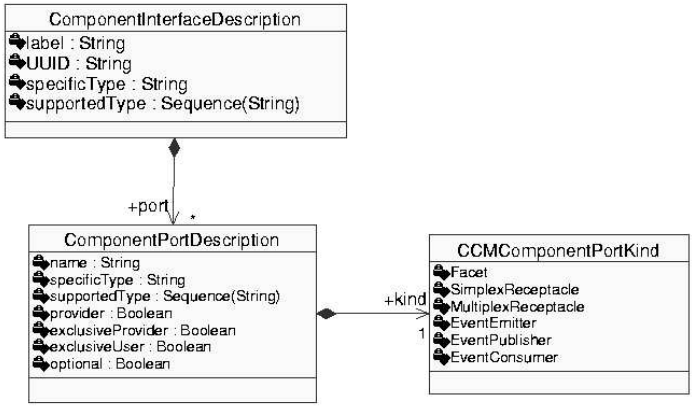


FIG. 7.2 – Description d'une interface de composant CCM

modèle général du descripteur d'assemblage est illustré dans la figure 7.3. Le descripteur d'assemblage CCM décrit les informations suivantes :

- les paquetages des composants qui constituent l'assemblage désignés par leur nom (*componentfiles*) ;
- des informations de distribution (*partitioning*) qui décrivent quels sont les composants qui seront instanciés et où les instancier. L'instanciation d'un composant est toujours relative à une maison de composant. Le déploiement d'une maison est décrit par un élément *homeplacement*. Il est possible de décrire le déploiement d'instances de composant en les affectant à des machines ou à des processus avec la possibilité de spécifier des colocations d'un ensemble d'instances de composants sur une seule machine ou un seul processus (*hostcollocation*, *processcollocation*). La description de l'instanciation d'un composant nécessite la spécification d'un descripteur de propriétés de l'instance de composant ;
- des connexions entre les instances de composants. Deux types de connexions sont supportés dans CCM : les connexions synchrones (*connectinterface*) et les connexions asynchrones (*connectevent*). Le premier type connecte une facette d'un composant à un réceptacle d'un autre et le deuxième type connecte une source d'événements d'un composant à un puits d'événements d'un autre.

Le descripteur d'assemblage CCM contient les méta-informations spécifiées par le plan de déploiement final du modèle de données indépendant de la plate-forme. Il ajoute les méthodes d'instanciation et de connexion de composants dans CCM et la possibilité de décrire des colocations d'instances de composants.

Dans le cadre de CCM, l'adaptateur de déploiement génère un descripteur d'assemblage à la place d'un plan de déploiement final, cela n'introduit pas de modifications au niveau de l'interface externe de l'adaptateur de déploiement. Mais nécessite l'extension de ses interfaces internes. Ces extensions sont détaillées au niveau de l'implémentation (voir section 7.2.4).

7.1.2 Modèle de CADeComp spécifique à un intergiciel sensible au contexte

Nous avons vu dans la section 4.3.2 que CADeComp doit être placé au-dessus d'un intergiciel sensible au contexte qui se charge de la collecte et de la gestion des informations de contexte. La projection du modèle de CADeComp sur un intergiciel sensible au contexte concerne les éléments ayant un rapport avec la collecte des informations de contexte tels que le modèle de description de contextes, le modèle de description de contextes pertinents et le filtre de contexte. Dans [ABTB05], nous avons étudié le placement de CADeComp au-dessus de l'intergiciel sensible au contexte CAMidO [BTB05]. Son implémentation n'étant pas finie à ce jour, nous ne pouvons pas faire les expérimentations de CADeComp au-dessus de CAMidO. En attendant CAMidO, nous avons développé un outil de collecte d'informations de contexte appelé "collecteur de contexte" qui bien que simple, joue le rôle d'intergiciel sensible au contexte. Cet outil a été développé dans le but de tester CADeComp, il permet la collecte des contextes simples et de bas niveau tels que les préférences des utilisateurs, les informations sur les ressources d'un nœud et

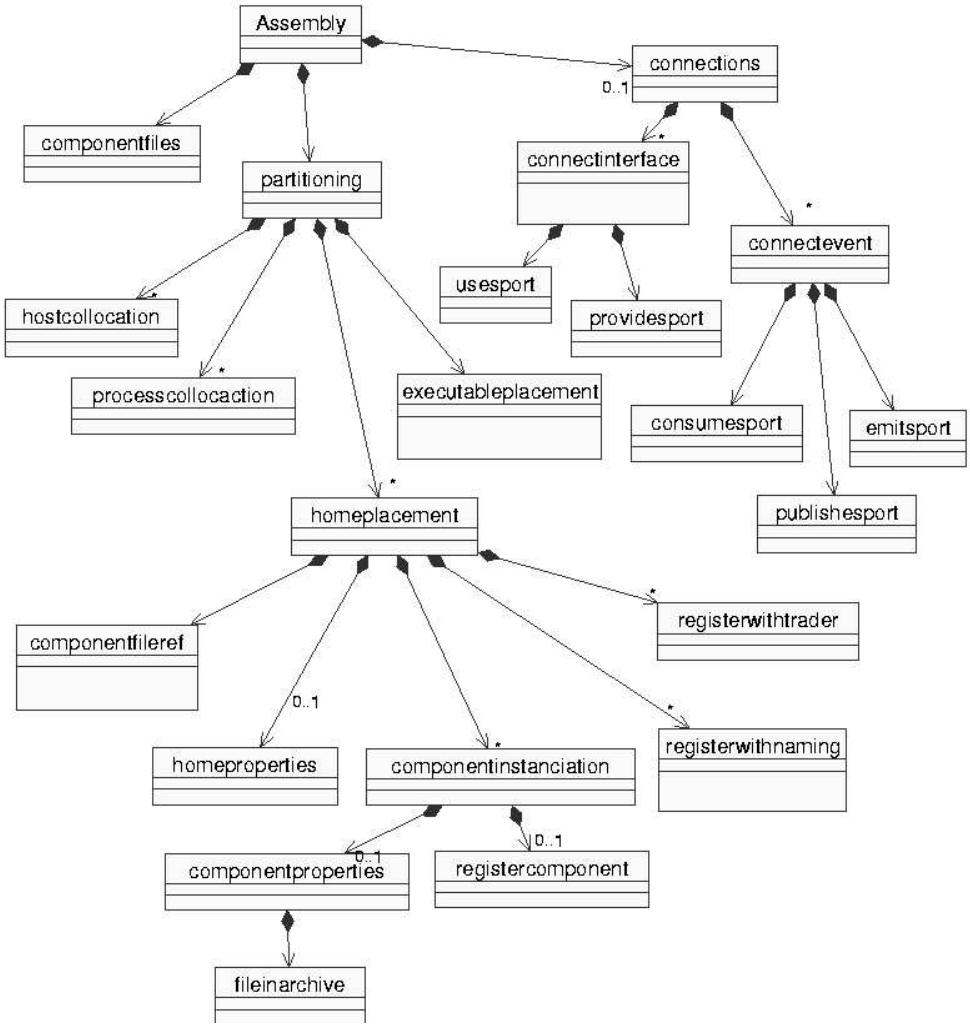


FIG. 7.3 – Modèle de descripteur d'assemblage CCM

la localisation de l'utilisateur. Nous commençons par décrire brièvement le collecteur de contextes puis nous spécifions la projection du modèle de CADeComp sur cet intergiciel.

Collecteur de contextes

Le collecteur de contextes (voir figure 7.4) est constitué d'un ensemble de sous-collecteurs de contextes qui interagissent avec les sources physiques du contexte afin de rendre l'information de contexte accessible à l'application. Chaque collecteur de contexte offre une interface permettant d'interagir avec une source de contexte bien déterminée selon deux modes : le mode par interrogation, qui permet à une entité de récupérer l'état du contexte suite à sa demande, et le mode par notification qui permet à une entité de s'inscrire en fournissant une description d'un contexte pertinent. A chaque fois que ce contexte pertinent est vérifié, l'entité est notifiée avec indication de l'état du contexte. Chaque source de contexte fournit une méthode d'acquisition à partir d'un capteur, un fichier de données ou un logiciel de simulation.

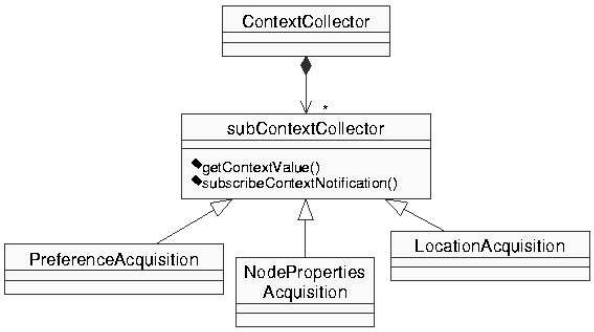


FIG. 7.4 – Diagramme de classes UML du collecteur de contextes

Projection de CADeComp sur le collecteur de contextes

Dans le but de supporter les contextes de haut niveau dans le cadre de l'utilisation de notre collecteur comme intergiciel sensible au contexte, la description du contexte, présentée dans la section 5.2, doit être étendue par une description des éléments nécessaires à l'interprétation des contextes. La figure 7.5 montre le modèle de description du contexte de déploiement d'une application qui inclut une description d'un contexte de haut niveau. Un contexte de haut niveau est un contexte interprété (*ContextInterpretationDescription*) qui nécessite la spécification de l'ensemble de contextes à partir desquels le contexte est dérivé ainsi que l'opération permettant de le dériver (*InterpretationOperationDescription*). Cette opération est définie par un nom, une classe dans laquelle elle est définie (modélisée par l'élément *InterpretationClass*) et un ensemble de paramètres définis par leur type et leur valeur (*InterpretationParameter*).

Par exemple (voir figure 7.6), le contexte "userZone", qui représente la zone où se trouve l'utilisateur, est un contexte de haut niveau, dérivé du contexte de bas niveau qui représente la

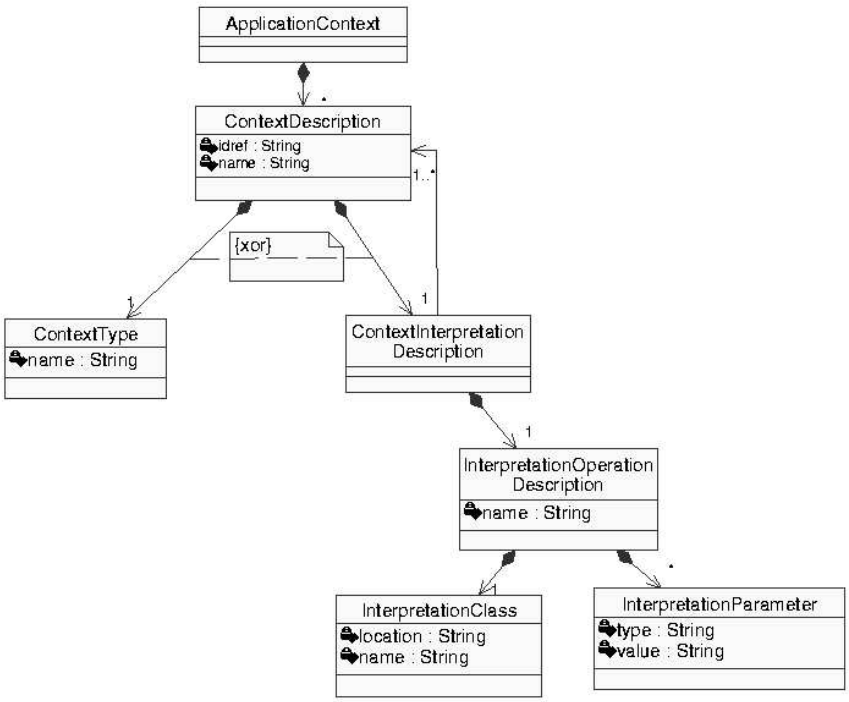


FIG. 7.5 – Contexte de déploiement d'une application

localisation de l'utilisateur à l'aide de l'opération *deduceUserZone()*. Cette opération ne nécessite pas de paramètres.

```
<ContextDescription idref="uz"  name="userZone">
  <ContextInterpretationDescription>
    <InterpretationOperationDescription name= "deduceUserZone">
       <ContextDescription idref= "UserLocationContext">
       <InterpretationClass name= "UserZoneDeducer"
             location= "http://www-inf.int-evry.fr/~ayed/deducers/">
       </InterpretationClass>
    </ContextDescription>
   </InterpretationOperationDescription>
  </ContextInterpretationDescription>
</ContextDescription>
```

FIG. 7.6 – Exemple de description de contexte de haut niveau

Nous n'avons pas décrit la projection du modèle de description des contextes pertinents et du filtre de contexte parce qu'ils ne nécessitent aucune modification dans le cadre de la projection sur notre collecteur de contextes. Le filtrage des contextes de haut et de bas niveau et la description de leurs contextes pertinents sont les mêmes.

7.1.3 Règles de transformation pour la définition d'un modèle spécifique à CCM et à l'intergiciel de contexte

Les éléments du modèle de données de CADeComp sont destinés à être utilisés pour générer des DTDs XML et les éléments du modèle d'exécution de CADeComp sont destinés à être utilisés pour générer des interfaces IDL. Nous définissons dans cette section les règles pouvant être utilisées pour générer de l'IDL concret et des DTDs XML.

Règles de transformation XML

Nous transformons les classes importantes du modèle de données spécifique CCM en fichiers XML que nous appellerons descripteurs.

Les classes importantes du modèle de données sont les suivantes (voir chapitre 5) :
– *ContextDescription*,
– *DeploymentDomain*,
– *RelevantContextDescriptor*,
– *ComponentPackageDescription*,
– *CAPropertiesDescription*,
– *ComponentInterfaceDescription*,
– *CADeploymentPlan*, et
– *DeploymentPlan*.

Nous utilisons les règles de transformation suivantes pour obtenir des descripteurs XML à partir des classes que nous venons de citer :

- une classe est transformée en un élément XML ;
- les relations de composition entre deux classes se traduisent par deux éléments XML se trouvant dans le même descripteur XML ;
- les relations de composition entre deux classes se traduisent par un élément père qui représente la classe source et un élément fils qui représente la classe cible ;
- les relations d'associations simples entre deux classes représentent deux éléments XML appartenant à deux descripteurs différents ;
- toutes les relations d'héritages sont enlevées et les attributs ainsi que les associations de la classe de base sont attachés à la classe dérivée ;
- les relations d'associations simples entre deux classes se traduisent par un attribut qui représente un identifiant au niveau de l'élément cible de la navigation et un attribut qui représente une référence qui correspond à cet identifiant au niveau de l'élément source ;

Les noms et extensions des descripteurs que nous avons obtenus en projetant le modèle données sur un modèle spécifique à CCM pour XML sont les suivants [1] :

- un descripteur de contextes ayant une extension *.cd* (*Context Descriptor*) ;
- un descripteur de contextes pertinents ayant une extension *.rcd* (*Relevant Context Descriptor*) ;
- un descripteur de domaine de déploiement ayant une extension *.ddd* (*Deployment Domain Descriptor*) ;
- un descripteur de paquetage de composant sensible au contexte ayant une extension *.cacd* (*Context Aware Component Descriptor*) ;
- un descripteur de variation de propriétés en fonction du contexte des différents composants, ayant une extension *.capf* (*Context Aware Property File*) ;
- un descripteur de plan de déploiement sensible au contexte ayant une extension *.cadp* (*Context Aware Deployment Plan*).

La classe *DeploymentPlan* est transformée en classe *Assembly* (voir section 7.1.1), les éléments de *PropertiesDescription* du plan de déploiement final sont partagés entre le descripteur *ccd* et le descripteur *cpf* de CCM : les propriétés fonctionnelles sont placées dans le descripteur *cpf* et les propriétés non fonctionnelles sont placées dans le descripteur *ccd*. Enfin, les éléments de *ComponentInterfaceDescription* sont décrits au niveau du descripteur *ccd*. Ces transformations nous amènent à utiliser les quatre descripteurs CCM suivants [2] :

- le descripteur logiciel de composant ayant une extension *.csd* (*Component Software Descriptor*) ;
- le descripteur de composant CORBA ayant une extension *.ccd* (*Corba Component Descriptor*) ;
- le descripteur de propriétés du composant ayant une extension *.cpf* (*Component File Descriptor*) ;
- le descripteur d'assemblage d'application ayant une extension *.cad* (*Component Assembly Descriptor*).

[1] Les DTDs de ces descripteurs sont sur le site de CADeComp http ://picolibre.int-evry.fr/cgi-bin/cvsweb/cadecomp/src/
[2] Les DTDs de ces descripteurs sont sur le site d'OpenCCM http ://openccm.objectweb.org/dtd/ccm/

Toutes les règles de transformations que nous avons définies peuvent être implémentées par une transformation basée sur une représentation XMI du modèle indépendant de la plate-forme.

Règles de transformation IDL

Les classes importantes du modèle d'exécution sont :
– DeploymentAdapter, et
– ContextFilter.

La transformation IDL est réalisée en utilisant les règles transformant les classes UML en interfaces IDL qui ont été défini dans le profile UML pour CORBA [UML02]. Il nous faut pour cela transformer les classes CADeComp spécifiques à CCM et au collecteur de contextes de manière à ce qu'ils correspondent aux classes UML définis pour CORBA dans le profil UML pour CORBA. Pour appliquer ces règles, les classes de CADeComp doivent avoir des stéréotypes définis dans le profil. Comme nous voulons transformer les deux classes DeploymentAdapter et ContextFilter en interfaces CORBA, nous leur rajoutons un stéréotype «CORBAInterface».

7.1.4 Synthèse

Nous avons décrit dans cette section la projection du modèle de CADeComp sur CCM et sur notre intergiciel sensible au contexte, le "collecteur de contextes". Le résultat de la projection du modèle d'exécution CADeComp est un ensemble d'interfaces IDL CORBA et le résultat de la projection du modèle de données est un ensemble de descripteurs nécessaire à l'adaptation du déploiement au contexte. Nous utilisons les descripteurs de déploiement CCM et nous les étendons par d'autres descripteurs qui permettent de prendre en compte le contexte.

7.2 Implémentation de CADeComp au-dessus d'OpenCCM

Nous avons implémenté CADeComp au-dessus d'OpenCCM, une implémentation libre de la specification du modèle CCM. OpenCCM permet la conception, l'implémentation, la compilation, le paquetage, le déploiement, l'exécution et la gestion des applications distribuées à base de composants. Nous nous intéressons au déploiement OpenCCM. OpenCCM a une version installable sur PDA. Nous commençons par présenter les descripteurs et les interfaces de CCM implémentés par OpenCCM puis nous décrivons l'architecture de CADeComp au-dessus d'OpenCCM.

OpenCCM offre un outil de paquetage qui supporte les deux types de paquetages CCM : les paquetages d'assemblage et les paquetages de composants. Il supporte aussi tous les descripteurs de déploiement CCM décrits dans 3.4.1. OpenCCM offre aussi une infrastructure de déploiement qui utilise les APIs de déploiement CCM présentées dans 3.4.1 et permet la gestion des domaines de déploiement telle que définie dans le projet COACH [COA03].

Dans ce qui suit, nous montrons l'utilisation des outils de paquetage et de déploiement d'OpenCCM dans le cadre de CADeComp tout en expliquant le rôle de chacun d'eux. Nous commençons par décrire une vue générale de l'architecture de CADeComp au-dessus d'OpenCCM.

Ensuite, nous détaillons l'implémentation du Client de Déploiement, des référentiels de paquetages et de méta-données, et de l'Adaptateur de Déploiement que nous avons décrits dans la section 6.3.1.

7.2.1 Architecture de CADeComp au-dessus d'OpenCCM

Afin de déployer et d'exécuter des composants sur le terminal utilisateur et sur les serveurs d'exécution du prestataire du service de déploiement, il nous faut suivre les étapes suivantes :

- installer un référentiel de configuration OpenCCM sur tous les nœuds (chacun des serveurs d'exécution et les terminaux utilisateurs). Ce référentiel contient les références de tous les services OpenCCM démarrés sur ces nœuds ;
- lancer un serveur Commanche sur tous les nœuds. Commanche est un micro serveur HTTP et Multicast qui permet l'accès à distance au référentiel de configuration OpenCCM (par exemple pour connaître les références des services OpenCCM lancés sur un nœud) ;
- démarrer sur un des serveurs d'exécution du prestataire du service de déploiement le service de nommage CORBA ;
- démarrer sur un des serveurs d'exécution du prestataire du service de déploiement le service de recherche sur proprités CORBA ;
- démarrer un serveur de composants java OpenCCM sur tous les nœuds. Ce dernier est la structure d'exécution qui héberge le conteneur CORBA, les maisons et les instances de composants. Il implémente le serveur de composants CCM (voir section 3.4.1) et les interfaces du conteneur ;
- démarrer sur un des serveurs d'exécution du prestataire du service du déploiement un gestionnaire OpenCCM d'infrastructure distribuée (un DCI Manager) appelé aussi gestionnaire de domaine. Un domaine est une fédération d'un ensemble de nœuds distribués.
- démarrer sur un des serveurs d'exécution du prestataire du service du déploiement l'*Adaptateur de Déploiement*.

Un exemple de placement des différents serveurs OpenCCM et CADeComp est présenté sur la figure 7.7.

Dans OpenCCM, le processus de déploiement peut être lancé manuellement à l'aide d'un script "ccm_deploy " qui utilise le nom du paquetage d'assemblage de l'application à déployer. Dans le cadre de CADeComp, ce script est automatiquement lancé par l'*Adaptateur de Déploiement* suite à la réception d'une requête de déploiement venant d'un *Client de Déploiement*. Nous détaillons dans ce qui suit le rôle du client de déploiement, de l'adaptateur de déploiement ainsi que le contenu des deux référentiels de paquetages et de méta-données.

7.2.2 Client de déploiement

L'interface graphique du client de déploiement que nous avons développée ressemble à celle présentée dans la figure 7.8. Ce client peut être développé de plusieurs manières et son implémentation est indépendante d'OpenCCM. Cette interface affiche une liste d'applications disponibles pour le déploiement (les applications auxquelles l'utilisateur est abonné). L'utilisateur

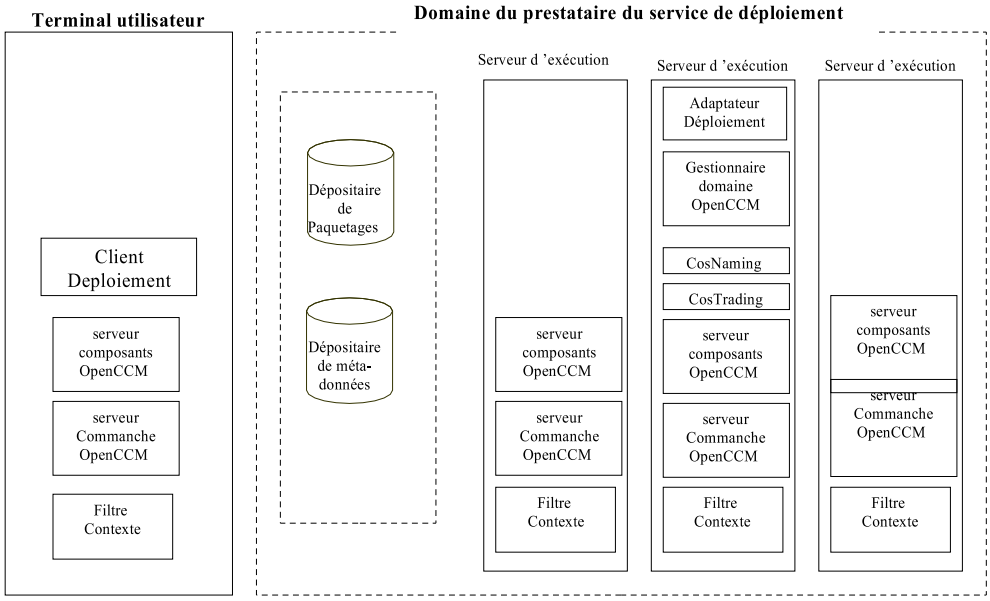


FIG. 7.7 – Architecture de CADeComp au-dessus d'OpenCCM

choisit une application dans cette liste et peut, s'il le souhaite, saisir ses préférences pour cette application en modifiant les préférences par défaut. L'exemple de la figure 7.8 montre le choix de l'utilisateur de l'application de gestion de crises et la saisie de ses préférences qui sont : le langage de l'utilisateur, le profile de l'utilisateur et le choix des profils des sauveteurs avec lesquels l'utilisateur va collaborer.

Pour lancer le déploiement de l'application choisie (son installation et son activation), le *Client de Déploiement* envoie une requête de déploiement vers l'*Adaptateur de déploiement* en lui indiquant le nom de l'application à déployer, le nom du terminal de l'utilisateur ainsi que les préférences de l'utilisateur.

Le client de déploiement est responsable du lancement automatique du serveur Commanche et du serveur de composants OpenCCM sur le terminal utilisateur.

7.2.3 Référentiels CADeComp

Le référentiel de paquetages de CADeComp contient les paquetages des composants à déployer. En plus des implémentations du composant et des descripteurs de déploiement CCM propres au composant : Component Software Descriptor (.csd), Corba Component Descriptor (.ccd) et Context Aware Property File (.capf), un paquetage de composant contient les descripteurs de déploiement CADeComp : Context Descriptor (.cd), Relevant Context Descriptor (.rcd) et Context Aware Component Descriptor (.cacd). Il y a des informations redondantes entre le .csd et le .cacd, mais nous préférons garder le .csd tel qu'il est afin de pouvoir assurer le déploiement sans prise en compte du contexte. Les paquetages de composants ont une extension .car (Component ARchive) utilisé par OpenCCM.

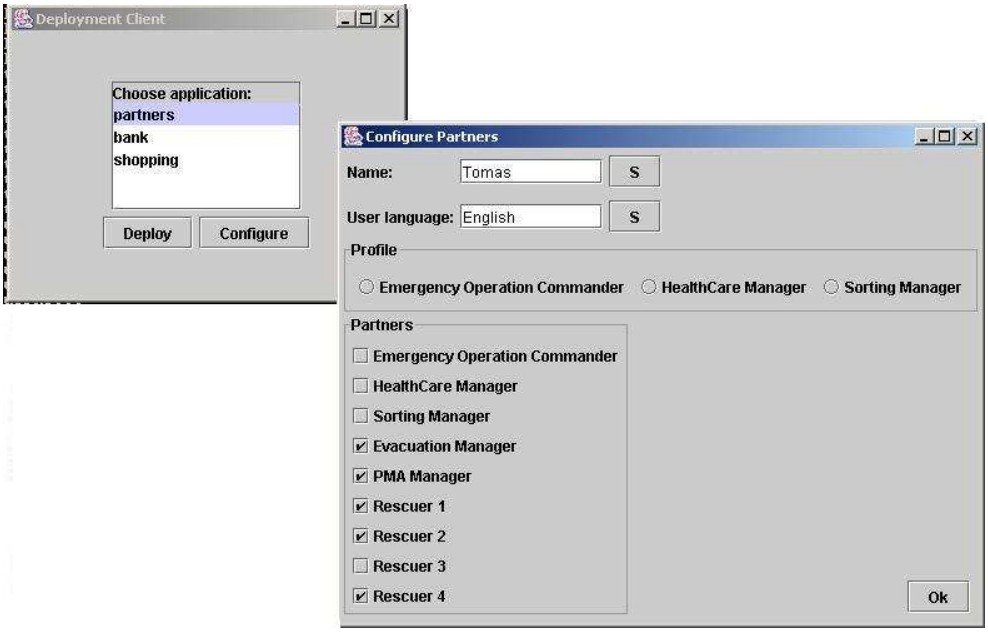


FIG. 7.8 – Client de déploiement

Le référentiel de méta-données contient des descripteurs .cd, .rcd et Context Aware Deployment Plan (.cadp) pour chaque application à déployer.

7.2.4 Adaptateur de déploiement

Dans le cadre d'une implémentation de CADeComp au-dessus de CCM, le *Gestionnaire de Méta-informations* de l'adaptateur de déploiement (voir figure 6.4) doit générer un descripteur d'assemblage CCM comme plan de déploiement final. Comme nous l'avons expliqué dans le 7.1.1, ce descripteur fait référence à des descripteurs qui décrivent les valeurs des propriétés de configuration de chaque instance de composant (des descripteurs de propriétés). Quant aux propriétés non fonctionnelles, elles sont décrites dans le descripteur de composant CORBA. Le gestionnaire de méta-informations génère donc en plus du descripteur d'assemblage le descripteur de propriétés de chaque instance de composant et met à jour le descripteur de composant CORBA de chaque composant.

L'entité *Adaptateur de Placement* CCM de la figure 7.9 étend l'Adaptateur de Placement avec des opérations qui permettent de déterminer les composants qui sont déployés sur un même nœud (*determineHostCollocation()*), les composant qui s'exécutent dans un même processus (*determineProcessCollocation()*) et les composants qui sont gérés par une même maison (*determineHomePlacement()*).

L'entité *Gestionnaire de Méta-informations* (*CCMMetaInformationManager*) CCM de la figure 7.9 étend le *Gestionnaire de Méta-informations* avec des opérations qui permettent de générer les différents éléments d'un descripteur d'assemblage CCM tels que *createFinalAssem-*

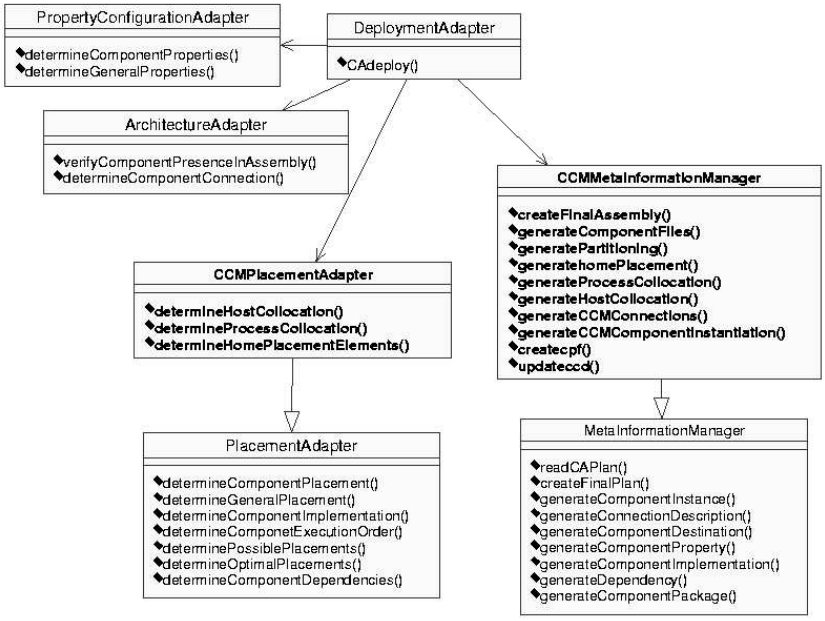


FIG. 7.9 – Modèle d'exécution spécifi que à CCM

bly(), *generateComponentFiles()*, *generatePartitioning()*, *generateHomePlacement()*, *generate-ProcessCollocation()*, *generateCCMConnections()* et *generateCCMComponentInstantiation()*. Le *Gestionnaire de Méta-informations CCM* comporte aussi des méthodes qui permettent de générer les descripteurs de propriétés (*createcpf()*) et de mettre à jour les descripteurs de composants CORBA(*updateteccd()*).

7.2.5 Synthèse sur le développement de CADeComp au-dessus d'OpenCCM

Le tableau 7.1 présente un récapitulatif des éléments développés dans CADeComp et de leurs propriétés

7.3 Outils associés à CADeComp

Nous avons développé deux outils pour faciliter l'utilisation de CADeComp : un éditeur de descripteurs de déploiement et un vérificateur de règles.

7.3.1 Éditeur de descripteurs de déploiement

L'éditeur de descripteurs de déploiement a été créé afin de faciliter la tâche des développeurs de composants et les planificateurs de déploiement. Il offre une interface graphique qui permet de

Source	– 347 KO de sources dont 45 classes java et une déclaration en OMG IDL – script Apache Ant pour la compilation de la distribution – deux démonstrations (application de gestion de crises et application de vente en ligne)
Exécutable	– 104KO pour l'archive (.jar) – 32KO nécessaire pour les archives externes utilisées par CADeComp (nanoXML)
Testé avec l'ORB	OpenORB1.3.1
Testé sur le système d'exploitation	– Linux RedHat 9.0 (noyau 2.4.20-8) – Windows 2000
Requiert les logiciels	– OpenCCM (version 0.8.1) – OpenORB (version 1.4.0) – Apache Ant (version >=1.6) – nanoXML (2.2.3)

TAB. 7.1 – Caractéristiques du prototype de CADeComp

générer automatiquement les descripteurs de contextes, les descripteurs de contextes pertinents, les descripteurs de paquetages de composants sensibles au contexte et les descripteurs de plan de déploiement sensible au contexte. Pour créer un descripteur de déploiement, il s'agit de créer son élément racine en indiquant son nom et la valeur de ses attributs, puis créer ses différents éléments fils et ainsi de suite jusqu'à atteindre les feuilles de l'arbre du descripteurs. Les différents fils possibles sont automatiquement indiqués à chaque fois et pour chaque élément. Des zones textes (ou des choix) sont proposés pour indiquer la valeur de l'élément ainsi que la valeur de ses attributs.

L'éditeur de descripteurs de déploiement se base sur Apollon [APO]. Apollon est un outil qui permet la génération d'une interface graphique à partir d'une DTD. Le fait que l'éditeur soit basé sur les DTDs des différents descripteurs de déploiement, un développeur de composant ou un planificateur de déploiement ne peut saisir que des éléments valides des différents descripteurs.

7.3.2 Vérificateur de la cohérence des règles d'adaptation

Le vérificateur de la cohérence des règles d'adaptation est associé à l'éditeur de descripteurs de déploiement. Il permet de vérifier la cohérence des descripteurs de déploiement. Ce vérificateur est démarré juste après la saisie du développeur d'un composant ou d'un planificateur de déploiement des éléments d'un descripteur. Il vérifie la cohérence de l'architecture et l'intersec-

tion entre les contextes pertinents associés au même paramètre de déploiement selon les règles de détection d'incohérences décrites dans la section 5.6.

7.4 Expérimentation et évaluation de CADeComp

L'adaptation du déploiement au contexte présente une réelle valeur ajoutée pour les utilisateurs mobiles. Elle prend en compte leur contexte pour déployer une application qui répond à leur besoin. Ce déploiement est réalisé à la volée (les composants sont téléchargés, instanciés et activés juste au moment de la demande) et d'une manière automatique de façon à épargner l'utilisateur de réaliser des tâches manuelles et répétitives de déploiement. Cependant, quelle que soit la valeur ajoutée, l'utilisateur ne supportera pas des temps d'attente très importants. Il nous a donc semblé essentiel de faire une étude expérimentale de CADeComp pour évaluer les temps de déploiement.

Pour tester et valider notre cadriciel CADeComp nous avons développé les applications décrites dans la section 4.1, d'autres applications de démonstrations ont aussi été développées et se trouvent sur le site de CADeComp [3]. Nous commençons par décrire notre plate-forme de démonstrations. Ensuite, nous détaillons l'étude des temps de déploiement. Enfin, nous décrivons le démonstrateur qui a été développé dans le cadre du projet AMPROS pour CADeComp.

7.4.1 Plate-forme de démonstrations

Nous avons expérimenté le déploiement CADeComp sur une plate-forme composée de quatre machines qui jouent le rôle de serveurs d'exécution du prestataire du service de déploiement (voir figure 7.10). L'Adaptateur de Déploiement est démarré sur une machine de 256 MO de RAM ayant un processeur Pentium Intel 1,8 Ghz et sur laquelle est installé Red Hat Linux 9.0. Les trois autres machines sont sous Windows 2000 et ont les mêmes capacités. Les quatre machines sont reliées par un réseau Ethernet.

Nous utilisons des terminaux de capacités différentes qui jouent le rôle de clients de déploiements :

- trois PDAs HP iPAQ h 6340 dotés de Microsoft Pocket PC 4.20.0, un processeur TI OMAP1510 et une mémoire de 56MO.
- deux PDAs HP iPAQ H3900 dotés de Linux Familiar 0.8.2, un processeur X Scale - PXA250 rev 4 (v51) et une mémoire de 61 MO.
- trois PCs portables dotés de Windows 2000, un processeur Pentium Intel 1,8 Ghz et 256 MO de mémoire.

Chacun des terminaux est équipé d'une carte WIFI. Les types et le nombre de terminaux utilisés dépendent de la démonstration testée.

[3] http ://picolibre.int-evry.fr/projects/cadecomp/

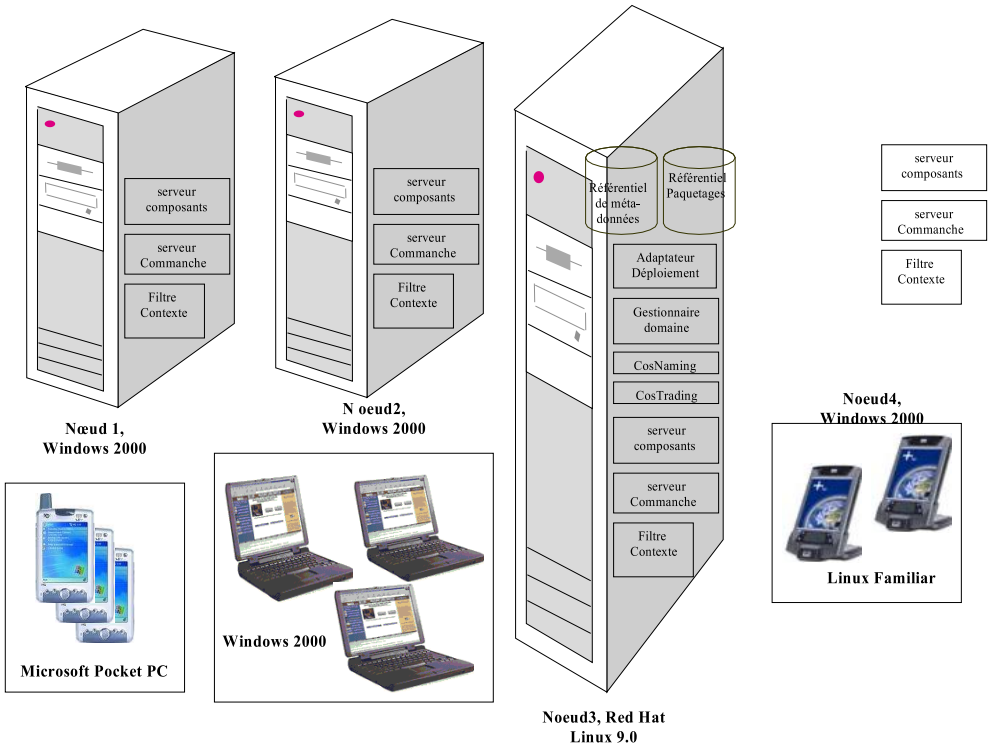


FIG. 7.10 – Plate-forme de démonstrations

7.4.2 Étude des temps de déploiement de CADeComp

Le déploiement tel que nous l'avons conçu consiste à collecter des informations de contexte, prendre des décisions d'adaptation et réaliser un déploiement à la volée (téléchargement des composants, installation et instanciation des composants) selon les décisions d'adaptation qui ont été prises. Le temps nécessaire au déploiement d'une application représente le temps consommé par ces différentes étapes. Bien que le temps d'attente utilisateur soit la somme des temps des trois étapes, le temps qui nous concerne le plus est le temps d'adaptation puisqu'il constitue l'apport de CADeComp. C'est sur ce temps là que porte notre étude.

L'adaptation dans le cadre de CADeComp peut impliquer la variation de toute la structure de l'application (le nombre des instances de composants qui la constituent). Il nous semble donc intéressant d'étudier la variation des temps de déploiement en fonction du nombre d'instances de composants déployées. Nous commençons par décrire la plate-forme d'évaluation. Ensuite, nous donnons une analyse du temps de déploiement total incluant toutes ses étapes. Enfin, nous analysons plus particulièrement le temps d'adaptation.

Plate-forme d'évaluation

Les résultats d'expérimentations que nous présentons dans le reste de ce chapitre ont été mesurés avec l'application de gestion de crises développée dans le cadre du projet AMPROS.

Le tableau 7.2 rappelle les différents composants de l'application de gestion de crises (voir figure 4.3). Nous utilisons les différents éléments de la plate-forme de démonstration décrite dans la section 7.4.1. Nous utilisons le PDA HP iPAQ H3900 comme terminal utilisateur. La figure 7.11 présente la plate-forme d'évaluation en montrant l'endroit d'instanciation des différents composants de l'application. Comme nous ne disposons pas d'un nombre suffisant de PDAs, nous démarrons plusieurs serveurs de composants sur l'un des ordinateurs portables chacun d'entre eux joue le rôle d'un PDA pour les différents sauveteurs. Comme premier jeu de tests, nous varions le nombre d'instances de composants entre cinq (une instance du composant GUI, une instance du composant "serveur de partenaires" et 3 instances du composant "partenaire") et cinquante (une instance du composant GUI, une instance du composant "serveur de partenaires" et 48 instances du composant "partenaire").

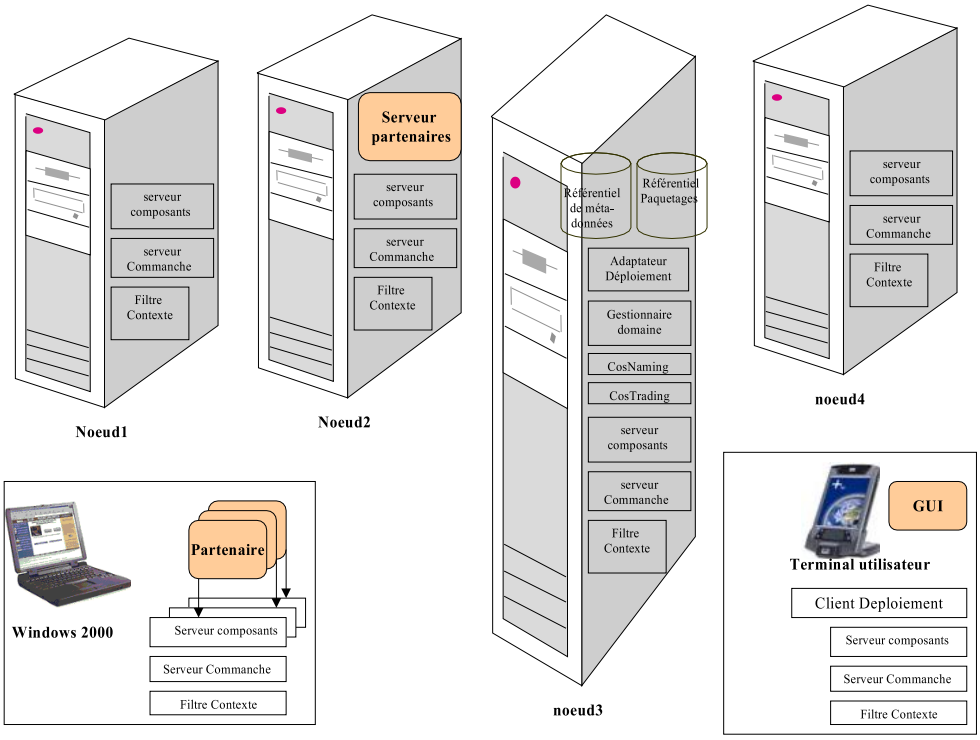


FIG. 7.11 – Déploiement de l'application de gestion de crises sur la plate-forme de d'évaluation

Analyse du temps de déploiement total

La figure 7.12 montre la variation du temps de déploiement moyen de l'application de gestion de crises, en fonction du nombre des instances de composants déployées, avec et sans adaptation. Chaque mesure est refaite vingt fois, nous donnons le temps de déploiement moyen, la variance moyenne des mesures est égale à 0,76.

Le temps de déploiement total se décompose comme suit :
D=Collect+Adapt+Depl, avec

D : le temps de déploiement total de l'application à déployer.
Collect : le temps de collecte des informations de contextes.
Adapt : le temps d'adaptation du déploiement de l'application à déployer.
Depl : le temps de déploiement sans adaptation.
Les courbes de la figure 7.12 présentent D et $Depl$. Les temps de collecte d'informations de contexte (profil des utilisateurs, préférences des sauveteurs, état du domaine de déploiement et nombre de sauveteurs) sont négligeables (de l'ordre de quelques micro secondes) c'est pourquoi ils n'apparaissent pas sur ces courbes. Cela est du au fait que nous les lisons à partir de descripteurs spécifiques à ces contextes. En réalité, les temps de collecte des informations de contexte dépendent de l'intergiciel sensible au contexte utilisé.

Types de composants	GUI	serveur de partenaires	partenaire	journalisation
Nombre d'implémentations	6	1	6	1
Taille moyenne de paquetage	120KO	130KO	125KO	100KO
Nombre	1	1	dépend du nombre de sauveteurs	0 ou 1
Emplacement	terminal de l'utilisateur	un des serveurs d'exécution	terminal du sauveteur	terminal de l'utilisateur

TAB. 7.2 – Structure de l'application de gestion de crises

Les résultats que nous avons obtenus, pour un nombre d'instances de composants variant entre 5 et 50 montrent que les temps de déploiement sans adaptation réalisés par l'outil de déploiement d'OpenCCM augmentent d'une manière presque linéaire avec la taille de l'application avec un taux moyen de 1,3% et passent de 9,5 secondes pour 5 composants à 16,5 secondes pour 50 composants. Ce temps de déploiement à la volée dépend de l'outil de déploiement utilisé.

Le temps d'adaptation augmente également linéairement avec le nombre des instances de composants de l'application avec un taux moyen de 0,21%. Nous avons un nombre de règles d'adaptation associées à chaque instance de composant à déployer au niveau application et au niveau composant : des règles pour son placement, sa configuration, ses dépendances, sa connexion et son existence s'il est optionnel. Dans le cas de l'application de gestion de crise, nous avons en moyenne 20 règles d'adaptation (une moyenne de 4 règles pour la variation de chaque paramètre). Le nombre de règles augmente avec le nombre de composants de l'application. Cela explique l'augmentation du délai d'adaptation avec le nombre de composants.

L'adaptation au contexte rajoute un délai allant de 0.5 secondes à 1 seconde au temps de déploiement sans adaptation. Ce délai représente en moyenne 5,48% du temps de déploiement total moyen. Nous décomposons dans le paragraphe suivant le temps d'adaptation d'une façon détaillée.

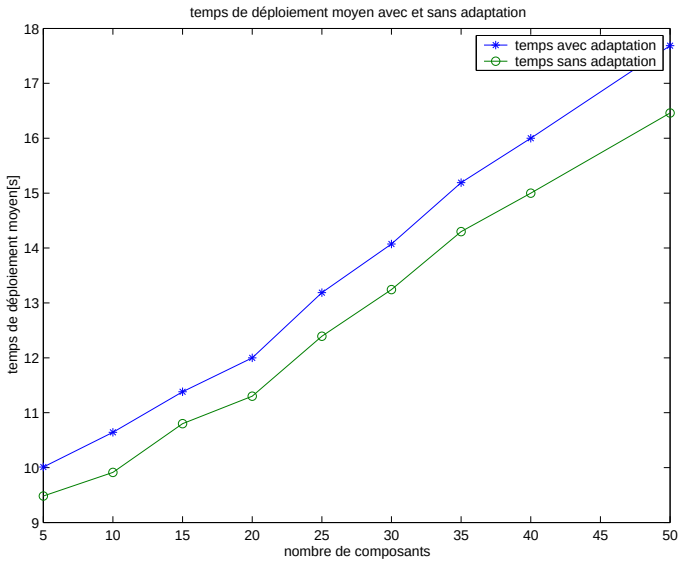


FIG. 7.12 – Variation du temps de déploiement moyen de l'application de gestion de crises, en fonction du nombre des composants déployés

Etude du temps d'adaptation

Le temps total d'adaptation représente le temps de traitement des règles d'adaptation et le temps nécessaire pour le pré-traitement de ces règles et la génération du plan de déploiement final :
Adapt= RulesPreprocess+RulesProcess+DeploymentPlanGeneration

Le pré-traitement des règles ($RulesPreprocess$) consiste principalement à lire et extraire le contenu des descripteurs contenant les règles d'adaptation du niveau composant et application (au niveau composant, nous avons un descripteur par composant).

Le temps de traitement des règles d'adaptation ($RulesProcess$) présente le temps de traitement des règles d'architecture, le temps de traitement des règles de configuration de propriétés et le temps de traitement des règles de placement (détermination de l'implémentation du composant, de son placement et de ses dépendances) :

RulesProcess(n)=Arch+Conf+Place

La figure 7.13 montre le temps moyen nécessaire pour le traitement des règles d'architecture, les règles de configuration des propriétés, les règles de placement ainsi que le temps d'adaptation total de l'application de gestion de crises. Ce dernier est largement supérieur à la somme des trois courbes parce qu'il inclut le temps nécessaire pour le pré-traitement de ces règles et la génération du plan de déploiement final.

Le temps de traitement des règles d'architecture ne dépend pas du nombre d'instances de composants qui seront déployées mais du nombre total d'instances de composants dans l'application (incluant tous les composants optionnels). Cela est dû au fait que ces règles sont traitées

même dans le cas où le composant n'est pas déployé (elles permettent au processus d'adaptation de décider ou non l'existence de l'instance du composant). Le temps de traitement des règles d'architecture est la somme du temps de traitement de l'existence des instances des composants et de l'existence de leurs connexions. Il peut être écrit comme suit :

$Arch(n) = (n_{total} \times t_{exsist} \times x_{reComp}) + (n_{total} \times y_{conn} \times t_{exist} \times z_{reConn}) = n_{total} \times t_{exsist}(x + yz)$, *avec*

n_{total} *: nombre total d'instances de composants de l'application*
t_{exsist} *: temps moyen de traitement d'une règle d'existence (d'architecture)*
x_{reComp} *: nombre de règles d'existence associées à un composant donné*
y_{conn} *: nombre de connexions d'un composant*
z_{reConn} *: nombre de règles d'existence associées à une connexion donnée*

Le temps de traitement des règles de configuration de propriétés est le plus élevé. Ceci est causé par le fait que chaque composant a plusieurs propriétés et que dans le cadre de CCM, nous générons un descripteur de propriétés pour chaque instance de composant après la détermination des valeurs de ses propriétés. Le temps de traitement des règles de configuration de propriétés peut être écrit comme suit : $Conf(n) = n \times y_{prop} \times x_{reprop} \times t_{prop}$
n *: nombre d'instances de composants à déployer de l'application (dont l'existence est vérifiée)*
x_{reprop} *: nombre de règles de variation de propriétés associées à une propriété donnée d'un composant*
y_{prop} *: nombre moyen de propriétés d'un composant*
t_{prop} *: temps moyen de traitement d'une règle de variation de propriété*

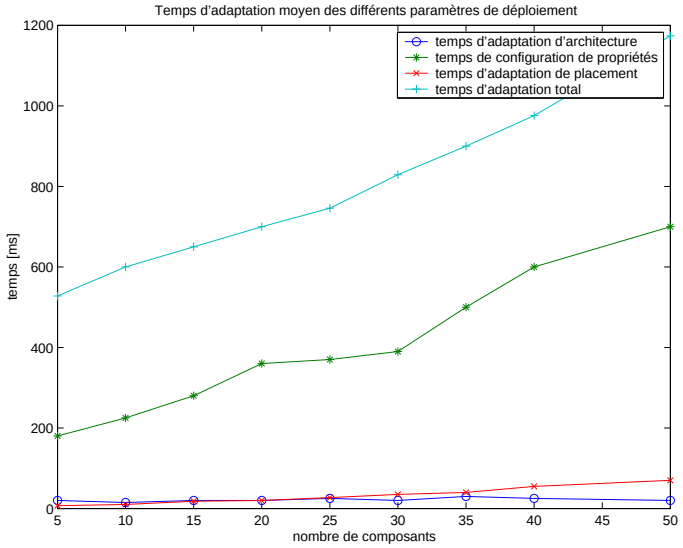
Le temps de traitement des règles de placement est négligeable parce que le placement du composant GUI et de tous les composants partenaires a été spécifié au niveau application (puisque nous savons qu'ils doivent être placés sur les terminaux des sauveteurs), ce qui ne demande pas beaucoup de temps de traitement. Seul le composant "serveur des partenaires" est placé d'une façon automatique en appliquant l'algorithme de placement.

Lorsque le placement et l'implémentation du composant sont explicitement spécifiés et qu'ils ne sont pas déterminées en utilisant l'algorithme de placement décrit dans la section 6.2, le temps de traitement des règles de placement peut être écrit de la manière suivante : $Place(n) = n \times (x_{replace} \times t_{place} + x_{reimpl} \times t_{impl} + x_{redep} \times t_{dep})$
n *: nombre d'instances de composants à déployer de l'application (dont l'existence est vérifiée)*
t_{place} *: temps moyen de traitement d'une règle d'adaptation de placement*
t_{impl} *: temps moyen de traitement d'une règle de choix d'implémentation*
t_{dep} *: temps moyen de traitement d'une règle de variation de dépendance*
$x_{replace}$ *: nombre de règles d'adaptation de placement associées à un composant donné*
x_{reimpl} *: nombre de règles de choix d'implémentation associées à un composant donné*
x_{redep} *: nombre de règles de variation de dépendance associées à un composant donné*

Le délai total d'adaptation sans application de l'algorithme automatique de placement est égal à :

$RulesProcess(n) = n \times (y_{prop} \times x_{reprop} \times t_{prop} + x_{replace} \times t_{place} + x_{reimpl} \times t_{impl} + x_{redep} \times t_{dep}) + n_{total} \times t_{exsist}(x + yz) + RulesPreprocess(n) + DeploymentPlanGeneration.$
Cette équation explique la linéarité de la courbe d'adaptation en fonction du nombre d'instances de composants de la figure 7.13.

FIG. 7.13 – Temps d'adaptation moyen des différents paramètres de déploiement de l'application de gestion de crises



Lorsque nous ne spécifions pas le placement des composants du composant GUI et de tous les composants partenaires au niveau application dans l'objectif d'observer le temps de placement des composants, nous obtenons les temps de placement présentés sur la figure 7.14. En effet, les temps de placement additionnels que nous avons obtenus représentent le temps de traitement de l'algorithme de placement présenté dans la section 6.2. La complexité de cet algorithme est de l'ordre de $n^2 \times n_{impl}^2 \times n_{noeuds}^2$, avec :
n *: nombre d'instances de composants à déployer de l'application (dont l'existence est vérifiée)*
n_{impl} *: nombre d'implémantation d'un composant donné*
n_{noeuds} *: nombre de nœuds offerts par le prestataire de service.*

La taille de l'arbre utilisé dans A* augmente avec le nombre d'instances de composants à déployer, le nombre de nœuds et le nombre d'implémentations des différents composants.

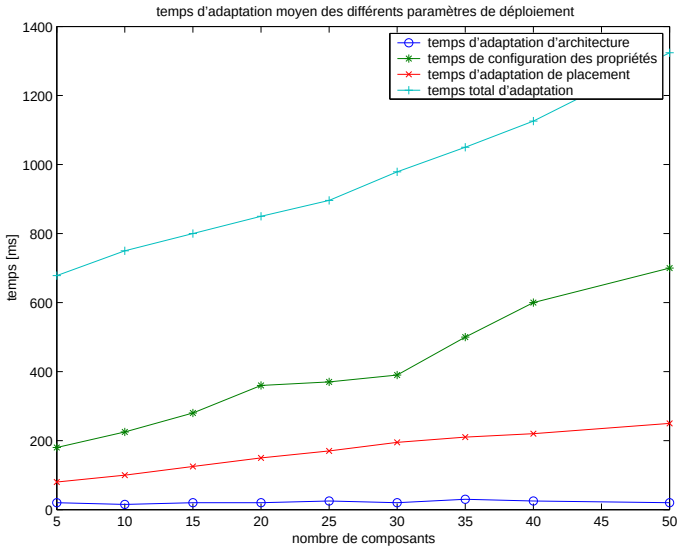
Augmenter la taille des paquetages de ses différents composants ne fait pas croître le temps d'adaptation. Il est donc très intéressant d'utiliser CADeComp avec des applications n'ayant pas un grand nombre d'instances de composants même si ses composants sont de grande taille.

7.4.3 Utilisation de CADeComp dans le cadre du projet AMPROS

AMPROS [4] (Adaptive Middleware Platform for PROactive reconfigurable Systems) est un projet dont le but est de réaliser un intergiciel pour les réseaux mobiles et de le valider dans le

[4] http ://www-inf.int-evry.fr/AMPROS/

FIG. 7.14 – Temps d'adaptation moyen des différents paramètres de déploiement : placement automatique des composants



domaine de gestion de catastrophes. Cet intergiciel consiste à adapter les applications à la variabilité des informations de contexte. Pour cela, il offre un ensemble de services pour les applications sensibles au contexte tels que la gestion de déconnexions et de cache pour les utilisateurs mobiles, la gestion de réconciliation, le déploiement dynamique et l'adaptation de contenu. CADeComp a été utilisé dans AMPROS pour réaliser l'adaptation du déploiement au contexte.

CADeComp a été testé et validé dans le cadre du projet AMPROS par une application de démonstration intitulée "Collect Victims" qui se place dans le même cadre décrit dans la section 4.1.2 pour l'application de gestion de crises. Le rôle de cette application est de créer et mettre à jour les données concernant les victimes dans une base de données. Les acteurs principaux de cette application sont les sauveteurs et le chef médical. Dès qu'une victime est récupérée, les sauveteurs la placent dans un sarcophage. Le sarcophage inclut un ordinateur intégré et plusieurs capteurs qui détectent l'état d'une victime (capteur de température, de pouls, de tension artérielle, etc.). L'application comporte les composants suivants (voir figure 7.15) :

– Plusieurs composants "Rescuer". Chaque composant de ce type permet à un sauveteur de déclarer la récupération d'une victime, d'établir un rapport primaire succinct sur l'état de la victime et la saisie de quelques informations tels que l'âge, le sexe et l'état de gravité de la victime (indemne, blessé, critique ou mort). Le nombre de composants de ce type dépend du nombre de sauveteurs sur le terrain de la catastrophe. Ces composants doivent être placés sur les terminaux des sauveteurs. Ils comportent deux versions d'implémentations : une pour ordinateur portable et une version allégée pour PDA. La propriété de configuration représentant la langue de l'utilisateur peut être variée.
– Plusieurs composants "Sensor". Ces composants sont déployés sur l'ordinateur du sarcophage et fournissent les informations collectées par les capteurs auxquels ils sont connectés.

- Un composant "Victim Manager" qui représente la base de données dans laquelle sont collectées les informations sur l'état de toutes les victimes.
- Un composant optionnel "Global View" qui est une interface graphique permettant d'afficher le contenu de la base de données. C'est un client de la base de donnée qui affiche pour chaque victime toutes les informations saisies par les sauveteurs ainsi que l'état des capteurs de son sarcophage.

Le scénario de déploiement se déroule comme suit.
Le déploiement de l'application est à l'initiative du chef médical. Ce dernier est informé du nombre de sauveteurs, des terminaux qu'ils utilisent et du nombre des victimes dans la zone des victimes. Il saisit ces informations au niveau du client de déploiement en indiquant le nombre d'instances de composants "Global View" qu'il veut déployer ainsi que les terminaux sur lesquels il veut les placer. Une fois qu'il lance une requête de déploiement, les étapes suivantes sont réalisées :

1. collecte des informations de contexte par l'intergiciel sensible au contexte AMPROS :
   - collecte des informations saisies par le chef médical ;
   - étude des préférences de chacun des sauveteurs et des propriétés de leur terminal comme le type du terminal et la langue utilisée. Ces propriétés sont lues à partir de fichiers se trouvant sur le terminal du sauveteur ;
   - détermination du nombre et des types de capteurs installés sur le sarcophage de chaque victime ;
2. réalisation du déploiement adaptable au contexte :
   - déploiement d'une instance du composant "Rescuer" sur chaque terminal de sauveteur. Chaque instance a une version d'implémentation et une langue qui correspondent à celles indiquées sur le fichier de propriétés du terminal ;
   - déploiement d'une instance du composant "Sensor" pour chaque capteur, sur le terminal du sarcophage de chaque victime ;
   - déploiement d'une instance du composant "VictimManager" en le plaçant sur le serveur d'exécution ayant le plus de ressources au niveau de la zone médicale primaire ;
   - déploiement d'une instance du composant "GlobalView" sur les terminaux indiqués par le chef médical.

L'utilisation de cette application dans le cadre d'AMPROS pour valider CADeComp a permis de tester la variation de plusieurs paramètres de déploiement en fonction du contexte comme l'architecture de l'application, l'emplacement de ses composants, leur implémentation et leurs propriétés de configuration.

7.4.4 Synthèse

Nous avons réalisé une première expérimentation et évaluation de CADeComp. L'évaluation a surtout porté sur le temps de déploiement de CADeComp qui représente un paramètre important à considérer pour pouvoir répondre aux besoins de déploiement des utilisateurs mobiles exprimés dans la section 1.2. Le temps d'adaptation du déploiement au contexte dépend essentiellement du nombre d'instances de composants à déployer et du nombre de règles associées aux différents paramètres de déploiement de chaque instance de composant, du nombre de nœuds de

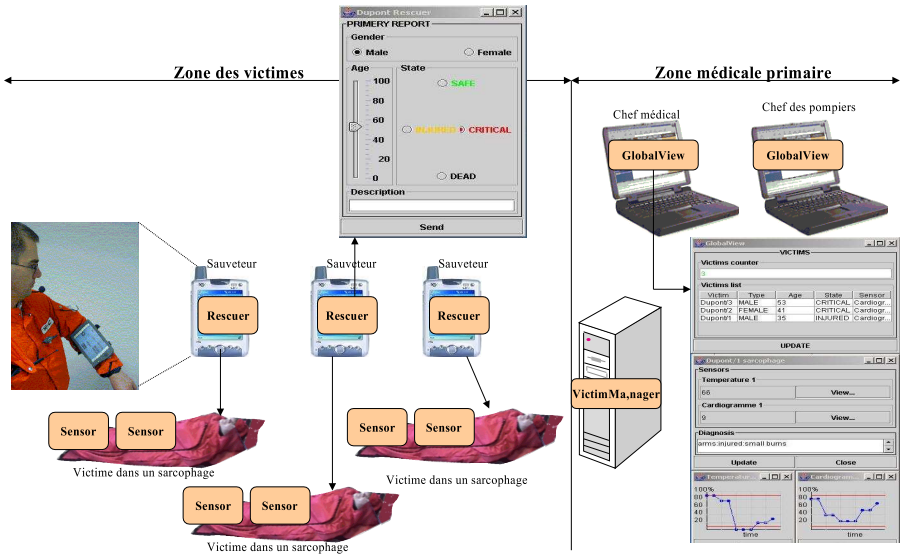


FIG. 7.15 – Déploiement de l'application

placement et du nombre d'implémentations des composants. Le temps d'adaptation du déploiement au contexte reste négligeable par rapport à l'apport de l'adaptation.

Une grande partie des temps d'adaptation représentent des temps d'écriture dans des descripteurs CCM tels que le descripteur d'assemblage ou le descripteur de propriétés. Cela veut dire que les temps de déploiement peuvent être réduits si nous utilisons une API de déploiement autre que celle de CCM, qui inclut directement ces informations sans avoir à les écrire dans des descripteurs.

7.5 Conclusion

Dans ce chapitre, nous avons défini un modèle de CADeComp spécifique à CCM. Nous avons pour cela projeté le modèle de données et d'exécution de CADeComp sur CCM. La projection a affecté les entités en rapport avec l'outil de déploiement. La projection de CADeComp sur CCM représente une projection sur un intergiciel composant. CADeComp nécessite aussi une projection sur un intergiciel sensible au contexte sur lequel il peut être utilisé. Cette projection a consisté à étendre les méta-informations et les entités reliées au contexte. Le résultat de la projection du modèle de données consiste en un ensemble de descripteurs XML qui permettent de prendre en compte le contexte lors du déploiement et qui incluent les descripteurs CCM et le résultat de la projection du modèle d'exécution consiste en un ensemble d'interfaces IDL qui réalisent l'adaptation du déploiement au contexte. Nous avons défini un ensemble de règles pour la génération automatique des DTDs XML et des interfaces IDL.

Nous avons implémenté CADeComp sur OpenCCM et nous l'avons évalué. CADeComp fournit des outils qui facilitent son utilisation. Il offre un client de déploiement qui permet l'accès à

ses différentes fonctionnalités et fournit aux développeurs de composants et aux planificateurs de déploiement deux outils qui permettent respectivement d'éditer les descripteurs de déploiement et de vérifier la cohérence de leur contenu.

L'évaluation de CADeComp a tourné autour des délais rajoutés par l'adaptation au contexte aux temps de déploiement normaux. Ces délais dépendent de plusieurs paramètres tels que le nombre d'instances de composants à déployer, le nombre de règles associées aux différents paramètres de déploiement de chaque instance de composant, le nombre de nœuds de placement et le nombre d'implémentations des composants. Nous trouvons que les délais d'adaptations constatés sont raisonnables par rapport à l'apport important de l'adaptation pour les utilisateurs mobiles.

Nous nous sommes limités au déploiement de cinquante instances de composants comme premier jeu de tests. Mais il sera intéressant d'observer la variation du temps de déploiement pour un nombre plus grand d'instances.

Notre choix de CCM s'est basé sur la richesse de son API et de ses descripteurs de déploiement. Mais malgré l'existence de versions d'implémentations de CCM pour les terminaux mobiles, l'environnement CORBA reste pas très adapté à ces derniers puisque le temps de déploiement de l'outil de déploiement reste élevé. Il serait intéressant d'étudier la projection de CADeComp sur d'autres types de plate-formes surtout dans le cas où nous voulons déployer une application sur plusieurs types d'intergiciels en même temps (par exemple une partie des composants sur CCM et l'autre sur J2EE). Nous envisageons actuellement de faire une projection de CADeComp sur EJB/J2EE.

Chapitre 8

Conclusion

Les terminaux mobiles tels que les téléphones portables et les PDAs sont devenus de véritables plates-formes pouvant accueillir une large gamme d'applications. Le contexte d'exécution des applications dans un environnement mobile se caractérise par un changement constant dû à la variation de la localisation de l'utilisateur, de sa connexion réseau, du terminal qu'il utilise ainsi que d'autres paramètres de son environnement.

Nous avons proposé dans ce document un cadriciel intitulé CADeComp pour l'adaptation au contexte du déploiement des applications à base de composants. Ce cadriciel est basé sur une approche qui consiste à prendre en compte le contexte lors des différentes phases du développement des composants de l'application et de leurs assemblages. Cette prise en compte du contexte consiste en une description d'un ensemble de méta-informations qui sont par la suite interprétées par des entités qui réalisent l'adaptation du déploiement au contexte.

Dans la suite de ce chapitre nous dressons un bilan des différentes contributions de notre thèse puis nous présentons quelques perspectives de recherches.

8.1 Contributions

8.1.1 Analyse du domaine de déploiement

Nous avons effectué une analyse du domaine de déploiement des applications à base de composants en faisant une étude comparative des outils de déploiement offerts par différentes plates-formes de composants. La contribution réalisée lors de cette analyse consiste en un cadre générique pour le déploiement des applications à base de composants qui définit les différentes activités de déploiement devant être couvertes par un outil de déploiement ainsi que les méta-informations nécessaires à l'accomplissement de ces activités. Cette analyse a été réalisée dans le but d'étudier la possibilité d'enrichir ces méta-informations et tâches pour supporter l'adaptation du déploiement au contexte.

8.1.2 Etude du déploiement dans un environnement mobile

Nous avons étudié les types de contextes qui peuvent agir sur le déploiement ainsi que les paramètres de déploiement qui peuvent varier en fonction du contexte. Parmi les méta-informations identifiées lors de l'analyse du domaine de déploiement, nous avons identifié cinq paramètres de déploiement variables en fonction du contexte qui sont l'architecture de l'application, l'emplacement de ses instances de composants, le choix de leurs implémentations, la valeurs de leurs propriétés et leurs dépendances. Les types de contexte qui agissent sur le déploiement d'une application dépendent de sa sémantique.

8.1.3 Modèle de données de CADeComp

Nous avons défini un modèle, indépendant de la plate-forme, pour décrire les méta-informations utilisées pour adapter le déploiement au contexte. Ces méta-informations se situent au niveau composant et au niveau application et sont liées à la sémantique de l'application. Elles décrivent le contexte qui agit sur le déploiement de l'application ainsi que des contrats de règles qui spécifient la variation des cinq paramètres de déploiement en fonction de ce contexte. Les incohérences qui peuvent être engendrées par l'écriture de ces règles ont été étudiées.

8.1.4 Algorithmes d'adaptation du déploiement au contexte

Nous avons défini un ensemble d'algorithmes génériques qui réalisent les différentes étapes d'adaptation du déploiement au contexte. Ces étapes ne dépendent pas de la sémantique des applications à déployer, ils consistent à collecter et filtrer les informations de contexte pertinentes pour le déploiement, ensuite analyser ces informations afin de déterminer les valeurs des cinq paramètres variables de déploiement. La détermination de l'emplacement et de l'implémentation d'une instance de composant est l'étape la plus difficile car elle nécessite l'étude des ressources de chaque nœud de placement. La particularité de l'algorithme de placement est sa capacité de prise en compte de différents types de contextes en plus les ressources des nœuds de placement.

8.1.5 Modèle d'exécution de CADeComp

Nous avons défini une architecture pour l'adaptation du déploiement des applications à base de composants au contexte. Cette architecture est basée sur un ensemble d'entités indépendantes de la plate-forme qui peuvent être rajoutées au-dessus d'un outil de déploiement classique. Le fonctionnement de ces entités est basé sur les algorithmes d'adaptation du déploiement au contexte que nous avons définis. Ces entités collectent les informations de contexte à partir d'un intergiciel sensible au contexte et utilisent les méta-informations d'adaptation au contexte décrites pour chaque application pour prendre des décisions de déploiement.

8.1.6 Projection du modèle de CADeComp sur un modèle spécifique

Nous avons projeté le modèle de données et d'exécution de CADeComp sur l'intergiciel composant CCM et sur le "collecteur de contextes", un intergiciel sensible au contexte que nous avons développé. Le résultat de la projection du modèle de données consiste en un ensemble de descripteurs XML qui incluent les descripteurs CCM tout en permettant de prendre en compte le contexte lors du déploiement. Le résultat de la projection du modèle d'exécution consiste en un ensemble d'interfaces IDL qui réalisent l'adaptation du déploiement au contexte.

8.1.7 Implantation et évaluation

Nous avons implémenté le modèle de CADeComp spécifique à CCM au-dessus de OpenCCM et nous avons développé un ensemble d'outils qui facilitent l'utilisation de CADeComp tels qu'un éditeur de descripteurs de déploiement et un vérificateur de la cohérence des règles d'adaptation. Nous avons réalisé une première évaluation de CADeComp en comparant ses temps de déploiement aux temps de déploiement sans adaptation au contexte. Nous avons trouvé que les délais rajoutés par l'adaptation au contexte sont négligeables par rapport à la valeur ajoutée de l'adaptation au contexte.

8.2 Perspectives

8.2.1 La reconfiguration en fonction du contexte

Nos travaux ont essentiellement porté sur l'adaptation du déploiement initial des applications à savoir l'installation des composants, leur instanciation et leur activation. Il serait intéressant d'étudier la reconfiguration dynamique des applications déployées en fonction du contexte de manière à pouvoir varier les paramètres de déploiement en cours d'exécution tels que rajouter, supprimer des composants ou modifier la valeur d'une propriété en cours d'exécution. La reconfiguration dynamique pose des problèmes critiques parce qu'elle peut remettre en cause l'intégrité de l'application. Elle nécessite la définition d'autres entités d'adaptation qui permettront de garantir l'intégrité des interactions entre les composants en cours de la reconfiguration ainsi que l'extension du modèle de données pour la description des reconfiguration suite à des changements de contexte. Dans [ATB04], il y a une ébauche de solution pour le support de la reconfiguration dans CADeComp.

8.2.2 Gestion du cache

CADeComp est basé sur le déploiement à la volée des applications à base de composants qui amène à désinstaller l'application juste après sa désactivation dans le but d'économiser les ressources du terminal de l'utilisateur. Cette solution pourrait bénéficier d'être utilisée conjointement

avec un système de gestion de cache. En effet, si les utilisateurs ont besoin d'utiliser la même application ultérieurement, la sauvegarde des applications installées après leur désactivation dans un cache permettrait de diminuer les temps de déploiement de manière significative. Lorsqu'il est nécessaire de libérer des ressources, différentes politiques pour désinstaller les applications déployées peuvent être utilisées telles que l'application la moins récemment utilisée (LRU) ou l'application la plus anciennement installée (FIFO).

8.2.3 Modélisation du contexte par une ontologie

Il serait intéressant de modéliser le contexte de déploiement à l'aide d'une ontologie. Cette modélisation permettra l'utilisation d'un moteur d'inférence qui raisonne sur les informations de contexte et déduit de nouvelles règles d'adaptation selon les relations entre les informations du contexte décrites par le développeur de composant et le planificateur de déploiement. Cette extension permettra à CADeComp de supporter les adaptations non anticipées.

8.2.4 Déploiement de fragments de composant

CADeComp permet le déploiement d'une application entière ou d'une partie d'une application en déployant que quelques composants de ses composants de base. Il serait intéressant d'étudier la possibilité de décomposer un composant monolithique et de ne déployer qu'une partie selon les exigences du contexte. Dans ce cas, il serait par exemple possible de déployer une seule interface parmi les interfaces du composant.

8.2.5 Projection de CADeComp sur d'autres plates-formes

Nous avons projeté le modèle d'adaptation du déploiement au contexte, indépendant de la plate-forme, sur un modèle spécifique à la plate-forme CCM. Nous avons choisi cette plate-forme parce qu'elle présente le modèle de déploiement le plus riche parmi les modèles que nous avons étudiés et qui sont implémentés. Malgré l'existence d'une implémentation de CCM pour les terminaux mobiles, cette plate-forme reste peu adaptée aux environnements mobiles. Il est sonc important de projeter le modèle CADeComp sur d'autres plates-formes tels que .Net ou J2EE/EJB.

Le domaine de l'automatisation du déploiement et en particulier de l'adaptation au contexte est un domaine important qui permettrait de faciliter la vie des utilisateurs informatiques et en particulier des utilisateurs de terminaux mobiles.

Bibliographie

[AAH+97] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide : A mobile context-aware tour guide. *Wireless Networks*, 3(5) :421–433, October 1997.

[ABC+00] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System support for bandwidth management and content adaptation in internet applications. In *Proceedings of 4th Symposium on Operating Systems Design and Implementation*, pages 213–226, San Diego, CA, October 2000.

[ABTB05] D. Ayed, N. Belhanafi, C. Taconet, and G. Bernard. Deployment of component-based applications on top of a context-aware middleware. In *The IASTED International Conference on Software Engineering (SE 2005)*, Innsbruck, Austria, February 2005.

[AC03] M. Aksit and Z. Choukair. Dynamic, adaptive and reconfigurable systems overview and prospective vision. In *ICDCS Workshops 2003*, pages 84–94, 2003.

[ACK94] A. Asthana, M. Cravatts, and P. Krzyzanowski. An indoor wireless system for personalized shopping assistance. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 69–74, Santa Cruz, California, December 1994.

[AF03] L.F. Andrade and J.L. Fiadeiro. Architecture based evolution of software systems. In *Third International School on Formal Methods for the Design of Computer, communication and Software Systems : Software Architectures*, Italy, November 2003.

[ALT] Alternis s.a. solutions for location data mediation. http ://www.alternis.fr.

[AMP] AMPROS (Adaptive Middleware Platform for PROactive reconfigurable Systems) Project Home Page. `http ://www-inf.int-evry.fr/AMPROS/`.

[And00] R. Anderson. The end of dll hell. In *Technical Article of Microsoft Corporation. http ://msdn.microsoft.com/library/*, January 2000.

[Ang00] D. Angeline. Side-by-side execution of .net framework. In *Technical Article of Microsoft Corporation. http ://msdn.microsoft.com/library/*, January 2000.

[APO] *http ://forge.objectweb.org/projects/apollon/*.

[ATB04] D. Ayed, C. Taconet, and G. Bernard. Deployment and reconfiguration of component-based applications in ampros. In *PROactive computing Workshop (PROW 2004)*, Helsinki, Finland, 2004.

[BCA+01] G. S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T.Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikoski. The design and implementation of openorb v2. *IEEE DS Online, Special Issue on Reflective Middleware*, 2(6), 2001.

[BCRP98] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, London, 1998.

[BCS02] E. Bruneton, T. Coupaye, and J.B. Stefani. Recursive and dynamic software composition with sharing. In *Proceedings of the Seventh International Workshop on Component-Oriented Programming (WCOP02)*, Malaga, Spain, June 10, 2002.

[BCS04] E. Bruneton, T. Coupaye, and J.B. Stefani. The fractal component model. In *ObjectWeb Specification*, 2004.

[BEA] Bea's weblogic. http ://www.bea.com.

[BGW93] D. BOBROW, R. GABRIEL, and J. WHITE. Clos in context - the shape of the design space. In *Object-Oriented Programming : the CLOS Perspective*, MIT Press, 1993.

[BHH+] S. Bechhofer, F. V. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Owl web ontology language reference. *W3C Recommendation, available at : http ://www.w3.org/TR/owl-ref/*, February 2004.

[Bro96] P.J. Brown. The stick-e document : a framework for creating context-aware applications. In *Proceedings of the IFIP Electronic Publishing*, pages 259–272, Laxenburg, Austria, 1996.

[BTAB05] A. Beloued, C. Taconet, D. Ayed, and G. Bernard. Placement automatique des composants lors du déploiement d'applications à base de composants. In *Actes des Journées Composants (JC 05)*, Le Croisic, France, 2005.

[BTB05] N. Belhanafi, C. Taconet, and G. Bernard. Camido, a context-aware middleware based on ontology meta-model. In *CAPS 2005, Workshop on Context Awareness for Proactiv Systems*, Helsinki,Finland, 16-17 June,2005.

[CA01] J. Caplen and M. H. Altarace. The art of ejb deployment. *Technical Article. http ://www.javaworld.com/javaworld/jw-08-2001*, Augut 2001.

[CBCP01] M. Clarke, G.S. Blair, G. Coulson, and N. Parlavantzas. An efficient component model for the construction of adaptive middleware. In *the IFIP / ACM International Conference on Distributed Systems Platforms (Middleware'2001)*, Heidelberg, Germany, November 2001.

[CCM02] OMG. *CORBA Components Version 3.0 :An adopted Specification of the Object Management Group*, June 2002.

[CEL00] Cellpoint, inc. the cellpoint system. http ://www.cellpt.com/thetechnology2.htm, 2000.

[CEM03] 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, October 2003.

[CFH+98] A. Carzaniga, A. Fuggetta, R. S. Hall, A. van der Hoek, D. Heimbigner, and A. L. Wolf. A characterization framework for software deployment technologies. Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, April 1998.

[CFJ03] H. Chen, T. Finin, and A. Joshi. An ontology for a context aware pervasive computing environment. In *The Proceedings of the IJCAI Workshop on Ontologies and Distributed systems (IJCAI 03)*, Acapulco MX, August 2003.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[COA03] *COACH WP2 : Specification of the Deployment and Configuration*, July 18, 2003.

[COM95] Microsoft. *Component Object Model Specification 0.9. http ://www.microsoft.com*, 1995.

[Cor] InstallShield Corp. Installshield. *http ://www.installshield.com*.

[Cur98] M. Curtin. Write once, run anywhere : Why it matters. *Technical Article. http ://java.sun.com/features/1998/01/wo*, Augut 1998.

[DAS01] A.K. Dey, G.D. Abowd, and D. Salber. A conceptual framework and toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer Interaction*, 16(2-4 (special issue on context-aware computing)) :97–166, December 2001.

[DAW98] A. K. Dey, G. D. Abowd, and A. Wood. CyberDesk : A framework for providing self integrating context aware services. *IEEE Personal Communications*, 11(1) :3–13, 1998.

[DC01] J. Dowling and V. Cahill. The k-component architecture meta-model for self-adaptive software. In *Reflection 2001*, 2001.

[Dey01] A.K Dey. Understanding and using context. *Personal and Ubiquitous Computing Journal*, 5(1) :4–7, 2001.

[DFSA99] A. K. Dey, M. Futakawa, D. Salber, and G. D. Abowd. The conference assistant : Combining context-awareness with wearable computing. In *Proceedings of the 3rd International Symposium on Wearable Computers (ISWC '99)*, pages 21–28, San Francisco, CA, October 1999.

[dI01] Diego López de Ipia. An eca rule-matching service for simpler development of reactive applications. *Middleware 2001*, 2(7), November 2001.

[DL02] P.C. David and T. Ledoux. An infrastructure for adaptable middleware. In *DOA'02*, Irvine, California, USA, October 2002. Springer-Verlag.

[EJB02] Sun Microsystems. *Enterprise JavaBeans Specification 2.0*, 2002.

[ET96] M. Ewing and E. Troan. The rpm packaging system. In *The first Conference on Freely Redistributable Software*, Cambridge, MA, USA, February 1996.

[FBCC02] A. Fitzpatrick, G. Biegel, S. Clarke, and V. Cahill. Towards a sentient object model. In *Workshop on Engineering Context-Aware Object Oriented Systems and Environments (ECOOSE)*, Seattle, WA, USA, November 2002.

[FF98] D. Franklin and J. Flaschbart. All gadget and no representation makes jack a dull environment. In *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, pages 155–160, Menlo Park, CA, 1998.

[FKV00] D. Fritsch, D. Klinec, and S. Volz. Nexus positioning and data management concepts for location aware applications. In *2nd International Symposium on Telegeoprocessing*, 2000.

[For95] Desktop Management Task Force. Software standard groups definition. http ://www.hpl.hp.com/people/arp/dmtf/ver2.htm. November 1995.

[GBS03] P. Grace, G. S. Blair, and S. Samuel. Remmoc : A reflective middleware to support mobile client interoperability. In *International Symposium on Distributed Objects and Applications(DOA)*, Catania, Sicily, Italy, November 2003.

[Gro03] Object Management Group. OMG Unified Modeling Language Specification, version 1.5. March 2003.

[Hal99] R.S. Hall. *Agent-based Software Configuration and Deployment*. PhD thesis, University of Colorado, 1999.

[HHSW99] A. Harter, A. Hopper, P. Steggles, and A. Ward. The anatomy of a context-aware application. In *Mobile Computing and Networking*, pages 59–68, 1999.

[HHVdHW97] R.S. Hall, D.M. Heimbeigner, A. Van der Hoek, and A.L. Wolf. An architecture for post-development configuraion management in a wide-area network. In *The International Conference on Distributed Computing Systems*, May 1997.

[HHW98a] R. S. Hall, Dennis Heimbigner, and A. L. Wolf. Requirements for software deployment languages and schema. *Lecture Notes in Computer Science*, 1439, 1998.

[HHW98b] R.S. Hall, D. Heimbeigner, and A.L. Wolf. Evaluating software deployment languages and schema. In *the International Conference on Software Maintenance, IEEE Computing Society*, November 1998.

[HHW99] R. S. Hall, Dennis Heimbeigner, and A. L. Wolf. A cooperative approach to support software deployment using the software dock. In *International Conference on Software Engineering*, pages 174–183, 1999.

[HIR02] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In *Pervasive 2002*, pages 167–180, Zurich, Switzerland, 2002.

[IAS00] Oracle technology network. oracle9i application server wireless. http ://technet.oracle.com/products/iaswe/content.html, 2000.

[IHAK02] Anca-Andreea Ivan, Josh Harman, Michael Allen, and Vijay Karamcheti. Partitionable services : A framework for seamlessly adapting distributed applications to heterogeneous environments. In *11th IEEE International Symposium on High Performance Distributed Computing*, pages 103–112, Edinburgh, Scotland, UK, 2002.

[Inc96] Desktop Management Task Force Inc. Desktop management interface specification. http ://www.dmtf.org/tech/specs.html. March 1996.

[J2E03] Sun Microsystems. *Java 2 Platform, Enterprise Edition Specification Version 1.4*, 2003.

[JB] Jboss homepage. http ://www.jboss.org.

[JEN] *Jena2. http ://jena.sourceforge.net/*.

[KC03] J. Keeney and V. Cahill. Chisel : A policy-driven, context-aware dynamic adaptation framework. In *POLICY 2003*, pages 3–14, Italy, 2003.

[Kic96] Gregor Kiczales. Aspect-oriented programming. *surveys*, 28A(4), December 1996.

[LB03] V. Lestideau and N. Belkhatir. Providing highly automated and generic means for software process deployment. In *Proceedings of 9th European Workshop on Software Process Technology (EWSPT 2003)*, Helsinki, Septembre 2003.

[MC02] R. Meier and V. Cahill. Steam : Event-based middleware for wireless ad hoc networks. In *International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02)*, pages 639–644, Vienna, Austria, 2002.

[MCZE02] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE : A Data-Sharing Middleware for Mobile Computing. *Int. Journal on Personal and Wireless Communications*, 21(1) :77–103, April 2002.

[Mey92] B. Meyer. Applying design by contract. In *IEEE Computer*, pages 40–51, 1992.

[MP00] R. Marvie and M.C. Pellegrini. *Etat de l'art des modèles de composants*. Délivrable du projet RNRT CESURE, 2000.

[MPRB04] G. K. Mostéfaoui, G. Pasquier-Rocha, and P. Brézillon. Context-aware computing : A guide for the pervasive computing community. In *ICPS*, pages 39–48, 2004.

[MRM02] M. Mikic-Rakic and N. Medvidovic. Architecture-level support for software component deployment in resource constrained environments. In *The first International IFIP/ACM Working Conference on Component Deployment*, Berlin, Germany, June 2002.

[MT00] N. Medvidovic and R. N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactiond Software Engineering*, 26(1) :70–93, January 2000.

[NI] Netinstall. *http ://www.twenty.com*.

[Nil80] Nils J. Nilson. *Principles of Artificial Intelligence*. San Francisco : Morgan Kaufmann, 1980.

[NSN+97] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, October 1997.

[ODP93] Information technology - open distributed computing - odp trading function. ISO/IEC JTC1/SC21.59 Draft, ITU-TS-SG 7 Q16 report, Novembre 1993.

[OMG01] OMG. *Model Driven Architecture*, Jully 2001.

[OMG03] OMG. *Specification for Deployment and Configuration of Component Based Distributed Applications*, Mars 2003.

[ope03] open services gateway initiative. *OSGi services platform specification, Release3*, March 2003.

[OW] Openweb netdeploy. http ://www.osa.com.

[Pas98] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2nd International Symposium on Wearable Computers (ISWC'98)*, Los Alamitos, CA, 1998.

[PB05] D. Preuveneers and Y. Berbers. Semantic and syntactic modeling of component-based services for context-aware pervasive systems using owl-s. In *First International Workshop on Managing Context Information in Mobile and Pervasive Environments*, pages 30–39, 2005.

[PK92] Sun microsystems. sunos 5.5 manual, 1992.

[RCDD98] T. Rodden, K. Cheverst, K. Davies, and A. Dix. Exploiting context in hci design for mobile systems. In *Proceedings of the Workshop on Human Computer Interaction with Mobile Devices*, Glasgow, Scotland, 1998.

[Sab03] Nawel Sabri. *Une architecture à base de composants CORBA pour Services Personnalisés*. PhD thesis, Université d'Evry Val d'Essonne, Juin 2003.

[Sat96] M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1), 1996.

[SAT+99] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced interaction in context. In *Proceedings of the First International Symposium on Handheld and Ubiquitous Computing,HUC'99*, pages 89–101, Karlsruhe, Germany, September 1999.

[Sco00] S. Scott. Structuring a .net application for easy deployment. In *Technical Article of Microsoft Corporation. http ://msdn.microsoft.com/library/*, February 2000.

[Sea03] Rebecca Searls. Java 2 Enterprise Edition Deployment API Specification, Version 1.1. *http ://java.sun.com/j2ee/tools/deployment/*, Août 2003.

[SKK+90] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. Coda : A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4), 1990.

[SS000] Signalsoft. wireless location services. http ://www.signalsoftcorp.com, 2000.

[STW93] B. Schilit, M. Theimer, and B. Welch. Customising mobile applications. In *Proceedings of USENIX Symposium on Mobile and Location-Independent Computing*, pages 129–138, August 1993.

[SW94] N. Schilit, B.and Adams and R. Want. Context-aware computing applications. In *Proceedings of the 1st IEEE International Workshop on Mobile Computing Systems and Applications*, Los Alamitos, CA, 1994.

[Szy02] C. Szyperski. *Component Software, Beyond Object-Oriented Programming, 2nd Edition*. Addison Wesley Professional, 2002.

[TTP+95] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *15th ACM Symposium on Operating Systems Principles(SOSP-15)*, Cooper Mountain, Colorado, 1995.

[UML02] OMG. *UML Profile for CORBA Specification*, April 2002.

[VHPT] Arthur Van Hoff, Hadi Partovi, and Tom Thai. The open software description format (osd). *Microsoft corp. and Marimba, Inc. (http ://www.w3.org/TR/NOTE-OSD)*.

[WH97] A. Ward, A.and Jones and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5) :42–47, 1997.

[WHFG92] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1) :91–102, January 1992.

[WS] Ibm's websphere homepage. http ://www-3.ibm.com/software/info1/websphere.

[YKW+02] 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, July-September 2002.

[YS00] H. Yan and T. Selker. Context-aware office assistant. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, pages 276–279, New Orleans, LA, January 2000.

[YTL04] N. Yahiaoui, B. Traverson, and N. Levy. Classification and comparison of adaptable platforms. In *First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04) held in conjunction with ECOOP '04*, Oslo, Norway, June, 2004.

Publications

Articles publiés dans des colloques avec actes et comité de lecture

1. Dhouha Ayed, Chantal Taconet, Nawel Sabri, Guy Bernard - Plate-forme de déploiement sensible au contexte des applications à base de composants. Actes de la 4ème Conférence Française sur les Systèmes d'Exploitation CFSE'4. Le Croisic France. 6-8 Avril 2005.

2. Abdelkrim Beloued, Chantal Taconet, Dhouha Ayed, Guy Bernard - Placement automatique des composants lors du déploiement d'applications à base de composants. Actes des Journées Composants (JC 05). Le Croisic France. 6-8 Avril, 2005

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 Engineering (SE 2005). Innsbruck, Austria. February 15-17, 2005.

4. Douha Ayed, Chantal Taconet and Guy Bernard - Deployment and Reconfiguration of Component-based Applications in AMPROS. in Proc. PROactive computing Workshop (PROW 2004). Helsinki, Finland. November 25-26, 2004.

5. Dhouha Ayed, Chantal Taconet, Nawel Sabri and Guy Bernard - Context-aware Distributed Deployment of Component-based Applications. in Proc. OTM Workshops of Distributed Objects and Applications 2004. DOA'04. Agia Napa, Cyprus. Oct 25-29, 2004.

6. Dhouha Ayed, Chantal Taconet and Guy Bernard - A data model for context-aware deployment of component-based applications onto distributed systems . Component-oriented Approaches to Context-aware Systems Workshop ECOOP'04. Oslo, Norway. June 2004.

7. Dhouha Ayed, Chantal Taconet and Guy Bernard - Architecture à base de composants pour le déploiement adaptatif des applications multicomposants. Actes des Journées Composants 2004. Lille, France. Mars 2004.

8. Denis Conan, Chantal Taconet, Dhouha Ayed, Lydialle Chateigner, Nabil Kouici, and Guy Bernard - A Pro-Active Middleware Platform for Mobile Environments. in Proc. IASTED International Conference on Software Engineering. Innsbruck, Austria. February 2004

9. Dhouha Ayed , Chantal Taconet, and Guy Bernard - Etude comparative des services de recherche sur propriétés. Actes des Journées Scientifiques Francophones (JSF'03) en Electronique, Télécommunication et Informatique. Tozeur Tunisie. Décembre 2003.

10. Dhouha Ayed, Chantal Taconet, and Guy Bernard - Context-Aware Deployment of multi-component applications. in Proc. 5th Generative Programming and Component Engineering (GPCE03) Young Researchers Workshop. September 2003, Erfurt, Germany.

Articles publiés dans des colloques sans actes ou actes à diffusion limitée

C. TACONET, D. AYED, N. BELHANAFI - Context aware deployment. Object-Web Architecture Meeting, Deployment Workshop. Séville, Espagne, Janvier 2004.