Vers un environnement pour le déploiement logiciel autonomique

Mohammed El Amine Matougui

TELECOM SudParis

Université d'Évry Val d'Essonne

Thèse de doctorat de Télécom SudParis, préparée dans le cadre de l'école doctorale S&I, en accréditation conjointe avec l'Université d'Evry-Val d'Essonne

Spécialité Informatique

Par :

Mohammed El Amine MATOUGUI

Vers un environnement pour le déploiement logiciel autonomique

Thèse présentée pour l'obtention du grade de Docteur de TELECOM SudParis

Soutenue le 21 novembre 2013 devant le jury composé de :

Rapporteurs :			
	M.	Noël De Palma	Professeur à l'Université Joseph Fourier
	M.	Philippe Roose	Maître de Conférences HDR à l'IUT Bayonne
Examinateurs :			
	M.	Zidani Abdelmadjid	Professeur à l'université de Batna (Algérie)
	M.	Bruno Defude	Professeur à TELECOM SudParis - SAMOVAR
	Mme.	Chantal Taconet	Maître de Conférences HDR à TELECOM SudParis - SAMOVAR (Directeur de thèse)
	M.	Sébastien Leriche	Maître de Conférences à l'ENAC - Université de Toulouse (Encadrant)

Thèse numéro : 2013TELE0022

Résumé

Le déploiement de logiciels répartis dans des environnements à grande échelle et ouverts (tels les systèmes ubiquitaires, les systèmes mobiles et les systèmes P2P) est une problématique actuelle ouverte. Ces environnements sont distribués, hétérogènes et peuvent être de nature instable (dotés d'une topologie dynamique du réseau). Le déploiement dans ces environnements met en jeu un très grand nombre de machines, de liens réseau ainsi qu'un ensemble de contraintes de déploiement. Quelques solutions de déploiement existent aujourd'hui, mais ne sont exploitables que dans le cadre d'architectures figées. Dans la plupart des propositions de déploiement, une personne en charge du déploiement doit décrire la topologie de déploiement. En outre, la majorité de ces outils ne prennent pas en compte les problèmes dûs à la variabilité de la qualité de service du réseau, aux pannes des hôtes, aux défaillances des liens du réseau ou encore aux changements dynamiques de topologie, qui caractérisent les environnements ouverts.

Dans ce mémoire, nous présentons les motivations concernant une infrastructure de déploiement logiciel autonomique et les exigences sous-jacentes. Nous présentons un état de l'art du déploiement logiciel que nous analysons au regard du contexte visé. Ensuite, nous présentons notre contribution pour le déploiement autonomique. Nous proposons j-ASD, un intergiciel qui exploite la complémentarité de ces technologies pour réaliser un déploiement logiciel autonomique. Notre proposition concerne le déploiement de composants logiciels avec le support d'agents mobiles adaptables, d'intergiciel et de langage dédié. Le processus de déploiement proposé est en trois étapes : description des contraintes de déploiement, résolution, et déploiement autonomique. Pour la première étape, nous avons défini un langage dédié (DSL) comme langage de haut niveau pour exprimer des contraintes de déploiement. Pour la deuxième, nous avons conçu une infrastructure répartie pour collecter les propriétés des sites cibles, ce qui permet de résoudre les contraintes de déploiement. Pour la troisième étape, nous proposons un intergiciel à base d'agents mobiles pour la réalisation et la supervision du déploiement autonomique. Enfin, nous donnons les éléments de conception du prototype que nous avons implémenté, ainsi que les résultats des expérimentations de validation.

Mots clés

Intergiciel, déploiement de logiciel, informatique ubiquitaire, autonomicité, agents mobiles

Abstract

Software deployment in large-scale and open distributed systems (such as ubiquitous systems, mobile systems and P2P systems) is still an open issue. These environments are distributed, heterogeneous and can be naturally unstable (fitted with a dynamic network topology). Deployment in such environments require the management of a large number of hosts, network links and deployment constraints.

Existing distributed deployment solutions are usable only within static and reliable topologies of hosts, where a man in charge of the deployment has to describe more or less manually the topology. Moreover, majority of these tools do not take into account network and computer QoS variabilities, hosts crashes, network link failures and network topology changes, which characterize open and mobile environments.

In this thesis, we discuss the motivations for an autonomic software deployment and the requirements underlying for such a platform. We carefully study and compare the existing work about software deployment. Then, we propose a middleware framework, designed to reduce the human cost for setting up software deployment and to deal with failure-prone and change-prone environments. We also propose an autonomic deployment process in three steps : deployment constraints description step, constraints resolution step and the autonomic deployment step. For the first step, we defined a high-level constraint-based dedicated language (DSL) as support for expressing deployment constraints. In the second step, we have designed a distributed infrastructure to collect target hosts properties used to solve deployment constraints. For the third step, we propose an agent-based system for establishing and maintaining software deployment. At last, we give an overview of our working prototype with some details on some experimental results.

Keywords

Middleware, software deployment, ubiquitous computing, autonomic computing, mobile agents

Remerciements

Qu'il me soit permis, tout d'abord, d'exprimer mes vifs remerciements à toutes les personnes sans lesquelles ce travail de thèse n'aurait pu voir le jour.

Je voudrais aussi exprimer ma grande reconnaissance à tous les membres du jury pour la bienveillante attention qu'ils ont bien voulu porter à mon travail.

Je voudrais également faire part de ma pleine gratitude à mon encadrant M. Sebastien Leriche et à Mme Chantal Taconet, ma directrice de thèse, qui n'ont ménagé aucun effort pour m'assurer un encadrement de qualité tout au long de mes travaux. Leur précieuse aide, leurs conseils, leurs encouragements et la confiance qu'ils m'ont cessé de témoigner, ont été pour moi autant de stimulants et autant de facteurs de motivation et de réussite dans ma tâche.

Je remercie très sincèrement M. Noël De Palma et M. Philippe Roose d'avoir voulu accepter d'être mes rapporteurs et d'avoir bien voulu lire et évaluer mon travail de thèse. Je les remercie pour leurs lectures approfondies de mon mémoire de thèse, pour tout le temps qu'ils m'ont accordé et pour les remarques très constructives qu'ils m'ont données.

Je remercie également M. Zidani Abdelmadjid et M. Bruno Defude qui m'ont fait l'honneur d'accepter de faire partie de mon jury de thèse. Leur participation dans l'évaluation de ce travail est d'une très grande valeur.

Je m'omettrais aussi pas de remercier tous les membres de l'équipe MARGE et du département informatique de Télécom SudParis : Mohamed Sallemi pour sa précieuse aide à plusieurs occasions, Brigitte Houassine qui nous accord beaucoup de facilités au sein du département. Je remercie aussi toutes les personnes que j'ai connu au cours de cette thèse.

Mes remerciements iront également à Mohamed Boulkour et son épouse Karima pour tout ce qu'ils ont fait pour moi, depuis mon arrivée en France. Ainsi que tout mes amis qui n'ont cessé de m'encourager : Abdraouf, Azzedine, Badis, Charif, Jamel, Fycel, Hassen[2], Mohamed[3], Khaled[2], Yaakoub, Youcef et Zohir.

Un grand merci à ma famille pour son soutien tout au long de cette thèse. Mes chères parents, Said et Souad, pour les sacrifices qu'ils ont fait en faveur de mon éducation et sans qui je n'aurais pas pu réaliser cette thèse. Ma chère épouse Djawaher pour m'avoir supporté, encouragé et motivé. Je lui suis pour toujours reconnaissant. Mes beaux parents Abdelfetah et Hakima pour leurs encouragements constants. Je tiens également à remercier mes belles-sœurs Houda, Sarah et Soundous ainsi que mes deux sœurs Samira et Ikram et mon frères Lotfi.

Enfin, un grand merci à ALLAH de m'avoir accorder son aide à travers toutes ces personnes.

Table des matières

Table des figures

Chapitre 1

Introduction

Sommaire

1.1 Introduction

Le déploiement de logiciel est le processus qui suit celui de sa production. Son objectif est de rendre disponible le logiciel là où il doit être exécuté, puis de l'y maintenir.

Le déploiement d'un système à grande échelle est un processus complexe qui inclut l'installation, la configuration et l'activation de logiciels divers allant du système d'exploitation aux applications utilisateur. Il implique la gestion d'un très grand nombre de machines, de liens de réseau hétérogènes et d'une multitude de versions logicielles. La gestion de tous ces aspects nécessite des outils adaptés qui permettent le contrôle et l'automatisation du processus de déploiement.

Aujourd'hui, pour déployer de manière satisfaisante une application répartie, il est impératif de privilégier une intervention humaine pour gérer les activités de déploiement. En effet, un administrateur du déploiement installe, configure, adapte, effectue les mises à jours, etc. de l'application. Quelques outils permettent de simplifier et d'automatiser le déploiement sous certaines conditions (stabilité de la topologie sous-jacente, uniformité des composants, absence de pannes...), tout en se révélant inefficaces dès lors qu'apparaissent des conditions imprévues dans l'environnement (panne de machine, variations de qualité de service...).

Un exemple de déploiement d'actualité : sur la grille expérimentale GRID5000, les chercheurs qui programment un déploiement sur un certain nombre de machines peuvent se retrouver avec moins de ressources que prévues, si les machines tombent en panne entre le moment de leur réservation et leur allocation définitive. Grid5000 est une grille faite de machines homogènes (même environnement) relativement stables, cela permet de donner une idée de la difficulté qu'il existe à déployer de manière automatisée une application répartie à grande échelle dans des topologies plus complexes, ouvertes et plus instables tels les réseaux pair à pair (P2P) et les réseaux mobiles. Dans les environnements ubiquitaires par exemple, les différentes activités de déploiement sont rendues difficiles par l'absence d'hypothèses sur les spécificités ou la disponibilité du matériel.

Nous proposons dans cette thèse d'étudier, puis de réaliser un intergiciel de déploiement qui soit capable de répondre à cette problématique, c'est à dire de déployer «au mieux» une application répartie quel que soit le contexte d'exécution et en particulier dans les environnements ouverts et dynamiques par nature. Par «au mieux», nous entendons que cet intergiciel effectue de manière autonome un déploiement d'un logiciel réparti dans un environnement ouvert qui satisfasse un ensemble de contraintes spécifiées par l'administrateur en charge du processus de déploiement.

Pour passer à l'échelle du déploiement de grands systèmes, le déploiement autonome est nécessairement décentralisé (une approche centralisé ne permet pas le passage à l'échelle) et doit être capable de s'adapter localement aux variations de qualité de service, aux pannes, à l'apparition et à la disparition de machines. Cette capacité d'auto-gestion décentralisée nous a amené à l'utilisation du terme de déploiement autonomique dans le sujet de cette thèse.

1.2 Présentation de l'équipe d'accueil

J'ai effectué mes recherches au sein de l'équipe MARGE [1] (Middleware for Autonomous distRibuted applications with context manaGEment). MARGE est un *projet structurant* de la recherche de l'Institut Télécom. Ses membres font partie du département d'informatique de Télécom SudParis (Évry), un établissement d'enseignement supérieur et de recherche dans le domaine des technologies de l'information et de la communication. MARGE fait également partie du CNRS, unité mixte de recherche SAMOVAR (équipe ACMES).

L'équipe MARGE a 15 ans d'expérience dans les systèmes distribués et les intergiciels. Le vaste domaine des activités de recherche tourne autour de la conception et l'évaluation des paradigmes et des outils pour construire des objets, composants, architectures intergicielles ou à base de services visant à l'exécution d'applications

1. http://www-inf.it-sudparis.eu/MARGE

distribuées. Le contexte est celui des environnements de toutes tailles, comportant des dispositifs de toutes sortes (y compris les terminaux mobiles, les réseaux de capteurs sans fil et les étiquettes électroniques). Les travaux ciblent l'informatique ubiquitaire et diffuse, en se concentrant sur des modèles, des algorithmes et des outils permettant une adaptation dynamique (au démarrage de l'application ou lors de l'exécution d'applications) à des situations dont le contexte varie.

A la fin de mon travail de thèse, mon encadrant (Sébastien LERICHE) a changé d'établissement. Il est actuellement en poste à l'École Nationale de l'Aviation Civile (ENAC - Université de Toulouse), au sein du laboratoire d'informatique interactive (LII) [2].

1.3 Motivations et problématique

Pour comprendre la problématique et la spécificité du déploiement logiciel en environnement ouvert et à grande échelle, nous présenterons deux scénarios. Dans le premier, on souhaite déployer une application répartie sur une architecture ubiquitaire. Dans le second, on souhaite déployer une application de simulation dans une grille dont la taille implique des défaillances régulières de certains de ses nœuds.

1.3.1 Scénario 1 : topologie imprévisible

Dans ce premier scénario, nous souhaitons déployer un logiciel de démonstration de la plate-forme de gestion de contexte COSMOS [Conan 2007, Rouvoy 2008] dans un environnement réparti ubiquitaire constitué d'un ensemble de participants à une conférence. Ce logiciel fournit à l'ensemble des participants des informations statistiques sur leurs équipements connectés à un réseau WiFi au moment de l'expérimentation via l'intergiciel COSMOS. Le nombre de participants à cette expérimentation n'est pas connu à l'avance, il peut être d'une centaine de participants, équipés de machines diverses en terme de puissance, de système d'exploitation et de débit réseau.

COSMOS (COntext entitieS coMpositiOn and Sharing) est un intergiciel de gestion de contexte, il fournit un moyen puissant et flexible pour la collecte et le traitement des données lors de l'exécution. Dans la taxonomie COSMOS, un nœud de contexte représente une information de contexte modélisée par un composant FRACTAL [Bruneton 2004]. Cette unité de composition offre des interfaces pour exporter de l'information et déclare des dépendances explicites vers d'autres nœuds de contexte. Les nœuds de contexte sont organisés de manière hiérarchique, la racine étant l'expression la plus abstraite de la politique de gestion de contexte, les feuilles

2. http://lii-enac.fr

étant des collecteurs qui fournissent l'information de contexte sous la forme la plus brute. Le rôle des nœuds de contexte intermédiaires étant de filtrer, spécialiser, combiner, de l'information brute pour en renforcer le sens.

Enfin, chaque nœud de contexte peut être configuré pour traiter de manière différente l'information de contexte. Par exemple, un composant collecteur connecté à un capteur de température, configuré comme actif observant et bloquant en notification, perçoit de manière autonome les variations de température et met à jour une variable interne la représentant, mais ne les propage pas aux nœuds (composants) parents auxquels il est connecté. Ceux-ci devront réaliser une observation pour connaître la valeur de la température.

Les informations récoltées dans le cadre de cette expérimentation sont la taille de la mémoire (RAM) disponible, le type du système d'exploitation utilisé dans le terminal, le pourcentage de l'occupation du/des processeurs et la taille de l'espace disque disponible. Les informations affichées au courant de l'expérimentation sont la moyenne de la mémoire disponible, la moyenne de l'espace disque disponible, la moyenne du pourcentage de l'occupation des processeurs et les types des systèmes d'exploitation utilisés dans chaque site cible de déploiement.

Les équipements impliqués dans cette expérimentation sont connectés au réseau WiFi. Chaque équipement connecté au réseau sans fil est considéré comme un site cible de déploiement potentiel. Les sites sont de natures différentes, ils peuvent être des ordinateurs portables, des Smartphones, des Netbooks, des Macbook et des PDAs.

L'application à déployer est composée de huit composants (nœuds de contexte COSMOS) ; chaque composant représente une unité de déploiement. Les huit composants sont respectivement :

- Le composant `OS-Type` est un collecteur utilisé pour détecter et renvoyer le type du système d'exploitation installé dans un site cible de déploiement.
- Le composant `RAM-Size` est un collecteur qui permet de fournir la taille de la mémoire vive disponible dans un site.
- Le composant `Disk-Size` est un collecteur utilisé pour le calcul et l'envoi de la taille du disque disponible.
- Le composant `CPU-Occupation` est un collecteur qui permet de fournir le pourcentage d'occupation du/des processeurs.
- Les composants `Average-RAM`, `Average-Disk` et `Average-CPUS-Occupation` sont des nœuds de contexte intermédiaires permettant respectivement le calcul de la moyenne de la taille mémoire disponible, la moyenne de l'espace disque disponible et la moyenne du pourcentage d'occupation des processeurs.
- Le composant `Display-Results` est le composant client de l'intergiciel de gestion de contexte, responsable de l'affichage des résultats statistiques récupérés dans cette expérimentation. Pour fonctionner correctement, ce composant à besoin d'être installé et exécuté sur un site fonctionnant sous Linux et

disposant de 400 Mb de mémoire RAM disponible et d'une charge système (occupation CPU) inférieure à 20% .

La figure 1.1 présente un plan de déploiement souhaitable pour satisfaire les besoins de cette application.

FIGURE 1.1 – Plan de déploiement souhaitable

Dans un site cible de déploiement équipé d'un système d'exploitation Linux doté d'une capacité d'affichage, de suffisamment de mémoire requise (400 Mb) et une charge du/des processeurs inférieure à 20% nous pouvons déployer tous les composants de l'application. Soit sur la figure 1.1, le composant d'affichage des résultats Display-Results (en orange) ainsi que les composants du calcul des moyennes Average-RAM, Average-Disk et Average-CPUS-Occupation (en vert) et les quatre composants collecteurs OS-Type, RAM-Size, Disk-Size, CPU-Occupation (en jaune).

Dans tous les autres sites disponibles, nous déployons uniquement les composants collecteurs (en jaune) pour pouvoir récupérer respectivement la taille de la mémoire disponible, la taille de l'espace disque disponible, le pourcentage d'occupation des processeurs et le type du système d'exploitation.

Pour réaliser un tel déploiement, il est nécessaire de choisir les sites cibles qui vont être utilisés. Pour cela, avant le lancement du processus de déploiement, il nous faut d'abord résoudre deux problèmes : celui de la découverte du réseau et celui de l'administration multiple.

Le besoin de résoudre le problème de découverte du réseau est justifié par le caractère ouvert et la nature dynamique de l'environnement ciblé. En effet, au moment de la conception de l'application les sites cibles de déploiement ne sont pas forcément connus ; de plus, au départ du processus de déploiement, le nombre de participants est imprévisible et on ne connaît pas au préalable les nombres et le type de machines impliquées. Dans un environnement distribué classique, la solution la plus simple pour résoudre ce problème consiste à établir une liste de sites cibles d'une manière manuelle ou semi-automatique (par un script par exemple). Par contre dans un environnement ouvert (P2P ou ubiquitaire) dans lequel des machines sont amenées a apparaître et disparaître fréquemment et de manière imprévisible, il n'est pas possible d'adopter cette solution. Pour cela, le système de déploiement doit prévoir un mécanisme automatique qui permet la détection et la sélection des sites cibles de déploiement d'une manière automatique.

Le deuxième type de problème à satisfaire pour déployer notre logiciel, est celui de l'administration multiple. En effet, l'environnement cible de notre application est un ensemble de sites cibles de déploiement indépendants où chaque site dispose de son propre administrateur. Par conséquent, le système doit obtenir les droits d'accès sur chaque site cible pour être en mesure de déployer notre application.

Pour réaliser cette étape, nous ne voulons pas contourner les principes de sécurité des systèmes distribués. Nous voulons explicitement obtenir l'autorisation de chaque administrateur de site pour déployer notre application. On peut par exemple demander à chaque site connecté au réseau WiFi d'accepter d'établir une connexion sécurisé SSH (par un moyen à définir, éventuellement de manière similaire à la gestion des permissions dans les systèmes Android), en laissant le choix d'accepter ou de refuser cette connexion à l'administrateur de la machine. En utilisant cette connexion, on peut par la suite envoyer notre application ainsi que toutes les dépendances logicielles nécessaires et réaliser les activités de déploiement demandées.

Une fois les problèmes de découverte du réseau et d'administration multiple résolus, le processus de déploiement peut être lancé. Il doit couvrir toutes les activités de déploiement. Dans ce cas, l'activité d'installation inclut la résolution des dépendances logicielles, le téléchargement des composants dans les sites cibles et l'installation physique des composants dans chaque site. L'activité d'activation consiste en le lancement de l'exécution des composants installés.

Le système de déploiement doit aussi couvrir les activités de mise à jour et de désinstallation. Enfin, il doit fournir des mécanismes de reconfiguration (adaptation) dynamique du processus de déploiement au moment de l'exécution pour réagir aux variations de l'environnement et prendre en compte les situations de pannes de machines et des liens du réseau ainsi que les situation de déconnexion et de nouvelles connexions de sites cibles.

1.3.2 Scénario 2 : déploiement dans une grille

Le deuxième scénario présente le déploiement d'une application de simulation à grande échelle. Nous avons expérimenté ce scénario de déploiement lors d'un travail de recherche précédent dans le cadre du projet BROCCOLI [3].

L'objectif de ce projet était de construire une plateforme permettant de décrire, déployer, exécuter, observer, administrer et reconfigurer des applications de simulation à événement discret en utilisant des composants logiciels distribués à grande échelle. Les applications de simulation ainsi que les différents logiciels utilisés sont basés sur le modèle de composant Fractal et sur le langage de description d'architecture Fractal ADL. Le projet BROCCOLI utilise, plusieurs plates-formes logicielles pour la réalisation des applications de simulation à grande échelle.

Il utilise OSA [Ribault 2010] (Open Simulation Architecture) une plate-forme collaborative à base de composants Fractal pour la simulation à événement discret. Elle permet la modélisation des systèmes simulés et différents scénarios de tests.

Les applications de simulation sont conçues pour évoluer dans des environnements répartis à grande échelle et doivent continuellement gérer le contexte de simulation dans lequel elles s'exécutent afin de permettre la détection des différentes situations d'adaptation. La plate-forme de gestion de contexte utilisée est l'intergiciel COSMOS.

Les deux plates-formes OSA et COSMOS utilisent une implémentation java du modèle de composants Fractal (Julia, AOKell). La figure 1.2 illustre le processus de simulation dans le cadre du projet BROCCOLI.

La simulation commence par l'étape de préparation qui consiste en le développement d'un modèle de simulation ainsi qu'un ensemble de scénarios de tests de ce modèle. La deuxième étape consiste en le déploiement du modèle de simulation sur une Grille. La troisième étape consiste à analyser les différents résultats de la simulation.

Pour déployer les composants de l'application de simulation sur une grille telle que Grid'5000, le Framework DeployWare (FDF) [Flissi 2008] à été utilisé.

En utilisant DeployWare, l'utilisateur fournit un ou plusieurs descripteurs de déploiement décrivant un ensemble d'informations sur le logiciel à déployer ainsi qu'une spécification du déploiement. L'utilisateur spécifie les propriétés logicielles telles que le chemin vers les archives du logiciel, les ports de communications utilisés et les dépendances logicielles. Il établit également manuellement une liste de sites cibles (description des nœuds physiques) ou d'une manière semi-automatique en utilisant des scripts spécifiques. Ensuite, il produit aussi un plan de déploiement dans lequel il décrit comment installer et activer chaque partie du logiciel

3. http://www-sop.inria.fr/mascotte/Contrats/broccoli

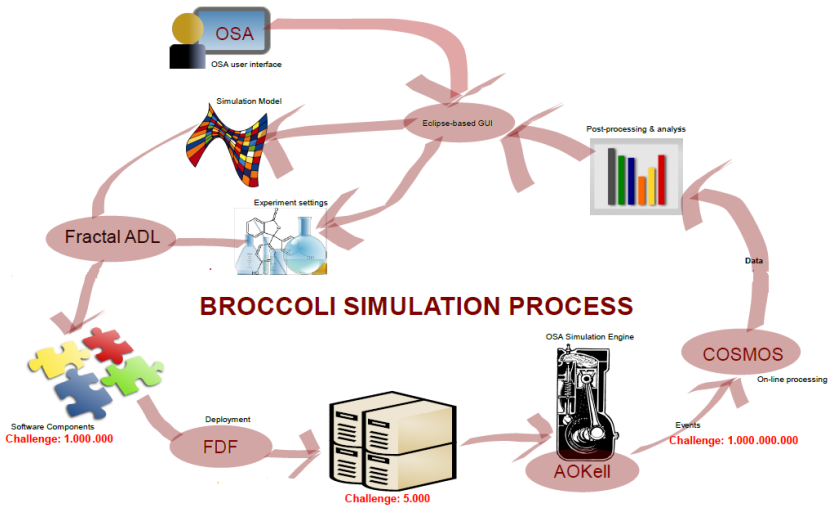

FIGURE 1.2 – Processus de simulation dans le projet BROCCOLI

(configurations d'installation et de démarrage). Une fois que l'utilisateur a terminé la description des contraintes, le processus de déploiement peut commencer. FDF (Fractal Deployment Framework) interprète les descripteurs DeployWare (description de l'application) comme une composition de composants Fractal, il utilise une librairie de composants de bas niveau pour abstraire les détails et masquer l'hétérogénéité des infrastructures physiques ; DeployWare est aussi en mesure de résoudre automatiquement les dépendances déclarées par un automate interne.

Une fois que l'application est installée avec succès l'utilisateur peut manuellement démarrer (activer) l'application, stopper (désactiver) l'application et désinstaller l'application. Les différentes activités de déploiement sont réalisées à travers l'outil graphique FDF-Explorer.

Contrairement au scénario précédent, le système de déploiement ne requiert pas un mécanisme de détection automatique des hôtes cibles de déploiement parce qu'on connaît au préalable la liste de machines cibles. De même, la résolution des problèmes d'administration multiples n'est pas pertinente car on est dans le cadre d'un système mono-administrateur. En effet, l'administrateur nous fournit les différents droits d'accès aux ressources physiques disponibles à l'opposé des systèmes ouverts tels que les systèmes P2P et les systèmes ubiquitaires.

Néanmoins, nous avons détecté d'autres types de problèmes et de spécificités dans ce scénario.

En effet, l'établissement manuel du plan de déploiement est une tâche complexe qui prend un temps important de mise en œuvre, en particulier lors du déploiement sur des milliers d'hôtes. En même temps, nous avons constaté qu'un environnement cible comportant un millier de sites cibles comporte, de temps en temps, une

ou plusieurs pannes de machines ou des liens du réseau. Par conséquent, le plan de déploiement risque d'être invalide et doit être souvent changé avant même le lancement du processus de déploiement si on dispose de moins de machines que prévues.

De plus, DeployWare ne prévoit pas des outils de reconfiguration dynamique au moment de l'exécution. Autrement dit, si une ou plusieurs situations de pannes de machines et de liens du réseau se présentent au moment de déploiement ; la seule solution possible était l'interruption du processus de déploiement, puis la modification du plan en prenant en compte la nouvelle situation et enfin le redémarrage du processus depuis le début. Par conséquent, DeployWare est une solution de déploiement logiciel difficilement utilisable dans les grilles large échelle et encore moins utilisable dans le cadre de systèmes ouverts tels que les systèmes P2P et ubiquitaires.

1.3.3 Synthèse

Dans cette introduction, nous avons présenté deux scénarios de déploiement logiciel dans des infrastructures réparties à grande échelle (environnement ubiquitaire et grille). Le premier scénario décrit le déploiement d'un logiciel dans un environnement ouvert avec une topologie imprévisible et des machines qui ne sont pas connues au début du processus de déploiement.

Le deuxième scénario présente le déploiement d'un logiciel de simulation sur une grille de calcul. Ces deux scénarios nous ont permis de mettre en évidence les spécificités et les problèmes rencontrés lors de déploiement en environnement instable et / ou à grande échelle.

La première spécificité, consiste en la nécessité de l'utilisation d'un service de découverte du réseau. En effet, dans les systèmes ouverts, l'administrateur de déploiement ne connaît pas forcément, à l'avance, les hôtes cible de déploiement. La seconde spécificité consiste en la satisfaction du problème d'administration multiple ; après la détection des sites il nous faut obtenir les droits d'accès aux machines pour permettre le déploiement de logiciels. La dernière spécificité, consiste à prévoir des outils de reconfiguration dynamique et d'auto-adaptation pour pouvoir traiter les situations de pannes de machines et les déconnexions.

A partir de ces deux scénarios, nous avons pu conclure qu'une plate-forme de déploiement autonomique en environnements ouverts doit répondre aux sept exigences suivantes :

1. être capable de détecter, gérer et accéder aux sites cibles d'une manière automatique.
2. être capable de gérer les hétérogénéités logicielles et matérielles des sites détectés.

3. être capable de fournir un moyen pour la déclaration des dépendances logicielles, les préférences matérielles et les contraintes de déploiement.
4. être capable de calculer un plan de déploiement qui satisfasse les contraintes de déploiement d'une manière automatique.
5. être capable d'exécuter les activités de déploiement avec un minimum d'intervention humaine.
6. être capable d'assurer des mécanismes d'adaptation automatique au moment de l'exécution pour prendre en charge les situations de panne de machines et des déconnexions.
7. être capable de s'exécuter dans des topologies dynamiques à grande échelle.

1.4 Contributions

Dans cette thèse, nous étudions plusieurs aspects liés à la problématique du déploiement autonomique de logiciel. Nous proposons une architecture logicielle et un intergiciel de déploiement qui permettent de répondre à cette problématique.

Cet intergiciel se base sur un système d'agents mobiles adaptables pour exécuter, contrôler et adapter le processus de déploiement. L'objectif est de permettre à un administrateur de déploiement voulant déployer un logiciel dans un environnement réparti à grande échelle (ouvert ou non) de déployer son logiciel automatiquement et de façon transparente avec un minimum d'intervention humaine. Le déploiement doit pouvoir être réalisé quelque soit le contexte de l'exécution, tout en respectant un ensemble de contraintes fixées au départ du processus de déploiement.

Grâce à l'utilisation des agents mobiles, l'intergiciel supporte aussi l'activité de reconfiguration dynamique au moment de l'exécution pour corriger les situations de pannes de machines et de déconnexion grâce à des décisions autonomes et décentralisées, prises par les agents de déploiement.

Nous avons voulu, en parallèle avec les réflexions sur l'élaboration de l'environnement de déploiement autonomique, réaliser un prototype fonctionnel permettant de valider l'approche proposée. La mise en œuvre de ce prototype et quelques expérimentations sont détaillées également dans cette thèse.

Les principales contributions de cette thèse sont les suivantes :

- La proposition d'une architecture logicielle pour le déploiement autonomique ;
- La proposition d'un langage dédié à la description de contraintes de déploiement et une approche atomique pour la transformation des contraintes et la génération d'un plan de déploiement initial ;
- La proposition d'un système d'agents mobiles et d'un algorithme de déploiement en trois étapes (la vérification de la validité du plan de déploiement, l'exécution du plan et enfin le contrôle et la reconfiguration du déploiement).

– L'implémentation de l'architecture proposé en utilisant des technologies Java et la plate-forme OSGi comme support de déploiement ;
– Le développement d'un prototype et des expérimentations pour évaluer l'approche proposée et prouver ça faisabilité.

1.5 Plan du document

Cette thèse se compose d'une introduction qui s'achève ici, d'un chapitre d'étude de l'état de l'art, d'un chapitre de description de la contribution, d'un chapitre d'expérimentation et d'une conclusion.

Le chapitre 2 propose un état de l'art des travaux relatifs à notre travail de thèse. Nous étudions dans ce chapitre des travaux sur le déploiement de logiciels dans des environnements distribués et des environnements ouverts tels que les systèmes ubiquitaires, les système P2P et les grilles de calcul. Nous nous focalisons sur les mécanismes de déploiement, les activités de déploiement et la reconfiguration du déploiement. Nous présentons, par la suite, les technologies agents logiciels et nous identifions les points pouvant être projetés sur notre contexte. Enfin, nous présentons les limites des plates-formes de déploiement existantes.

Le chapitre 3 constitue le cœur de notre travail. Nous détaillons respectivement l'architecture de notre intergiciel pour le déploiement autonomique de logiciel. Nous détaillons notre langage de description des contraintes de déploiement, la transformation des contraintes de déploiement, la résolution de contraintes pour le calcul du plan de déploiement, le système de découverte du réseau, le système d'agent mobile et l'exécution du processus de déploiement.

Dans le chapitre 4, nous nous sommes intéressés aux différentes expérimentations et résultats mis en œuvre pour la validation des différentes contributions présentées dans les chapitres 3.

Enfin, le chapitre 5 résume nos contributions et dégage les perspectives directes de nos travaux de recherche.

Chapitre 2

État de l'art pour le déploiement logiciel autonomique

Sommaire

Dans ce second chapitre, nous présentons un état de l'art des concepts et les technologies qui nous ont semblé essentiels pour le déploiement autonomique dans environnements instable et / ou à grande échelle. Nous décrivons le concept d'informatique autonomique en 2.1, puis nous réalisons un état de l'art sur le déploiement de logiciels en 2.2 et 2.3. Notre solution étant basée sur l'utilisation du paradigme agent logiciel, nous présentons les concepts liés aux agents en 2.4 à fin du chapitre.

2.1 Informatique autonomique

L'informatique autonomique (Autonomic Computing) est un concept proposé en 2001 par IBM [Horn 2001]. Il vise à construire des systèmes capables de prendre eux-mêmes en charge les tâches relatives à leur administration. Les auteurs partent du constat que la complexité croissante des systèmes risque d'atteindre les limites des capacités humaines et a rendu très complexe les différentes taches d'administration. Le terme autonomique se réfère à l'idée de restreindre au strict minimum l'intervention humaine dans les différentes activités d'administration du système. Autrement dit, l'objectif d'un système autonome est de soulager au maximum un opérateur humain en l'assistant dans les tâches d'administration et de le remplacer là où cela est possible. Un système autonomique quant à lui peut être vu comme un système autonome où la responsabilité et la prise de décision relative à son fonctionnement sont automatisées. Le rôle de l'opérateur humain dans ce contexte est alors réduit à la définition des politiques d'administration qu'il souhaite voir appliquées au système dont il a la charge. Le système autonomique quant à lui prend en charge la mise en application et le maintien de ces politiques définies sans le besoin de l'intervention de l'opérateur humain. Par conséquent, un système autonomique permet de simplifier considérablement les différentes tâches d'administration et offre une meilleur réactivité et efficacité en cas d'incident, car il est capable de s'y adapter plus rapidement. De plus, par ses facultés d'adaptation dynamique, il peut optimiser son fonctionnement en écourtant au maximum les délais de diagnostic et d'intervention.

Le fonctionnement des systèmes autonomes s'inspire largement de celui du système nerveux du corps humain. L'article fondateur du concept des systèmes autonomes [Kephart 2003] s'inspire du système nerveux humain qui permet par exemple de faire des efforts physiques sans avoir à se soucier du calcul du rythme cardiaque. Autrement dit, ils tentent de reproduire certains comportements qui lui permettent de se maintenir en bonne condition. Par conséquent, un système autonome doit offrir des fonctionnalités (propriétés) autonomes telles que :

1. L'auto-configuration, qui illustre la capacité du système à déterminer et à s'adapter de manière automatique et dynamique aux caractéristiques de l'environnement d'exécution. Elle illustre aussi sa capacité de paramétrage afin de fournir une configuration acceptable permettant au système de fonctionner correctement ;
2. L'auto-réparation, qui illustre la capacité du système à détecter, diagnostiquer et réparer des pannes survenant dans le système. Un système auto-réparable peut isoler les ressources défaillantes et répare la situation, par exemple, en choisissant des ressources alternatives disponibles dans l'environnement ;
3. L'auto-optimisation, qui illustre la capacité du système à assurer des niveaux de performances, par l'adaptation de la configuration du système en réponse aux événements liés à l'évolution de son état et de son environnement ;
4. L'auto-protection, qui représente la capacité du système à assurer sa protection contre des attaques internes ou externes pouvant le rendre inopérant.

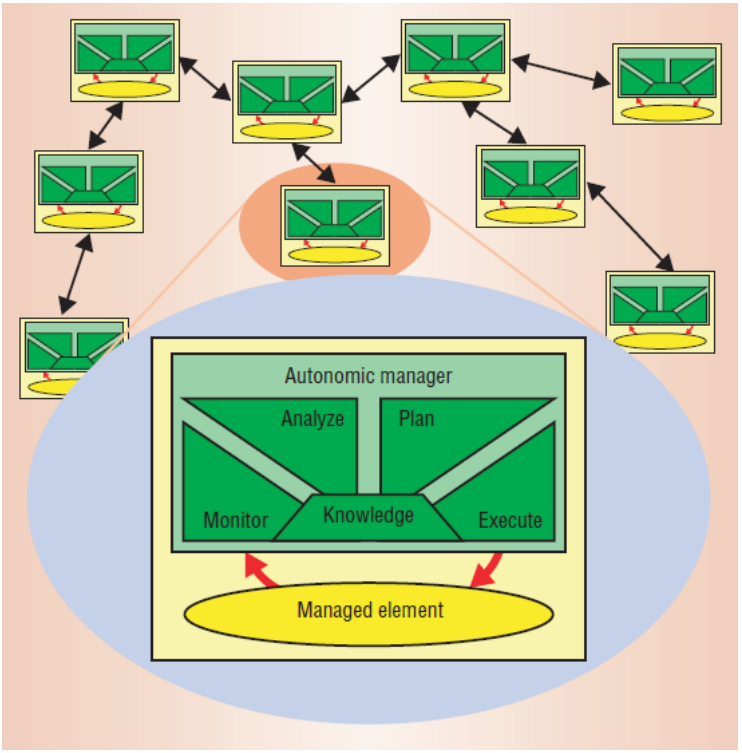

FIGURE 2.1 – Boucle de contrôle autonomique MAPE-K

La figure 2.1, présente le modèle de référence pour la boucle de contrôle autonomique en informatique suggéré par IBM dans [Kephart 2003]. Le modèle MAPE-K est constitué de cinq fonctionnalités qui sont respectivement : Monitor, Analyse, Plan, Execute et Knowledge. Le `Monitor` est le responsable de la gestion de ressources (matérielles et logicielles) et des observations. `Analyse` est responsable de l'analyse d'observations. `Plan` est le responsable de la planification des réactions

éventuelles. `Execute` est responsable de l'exécution du plan de réaction. `Knowledge` fournit la connaissance du système et de ses règles de fonctionnement.

Dans cette thèse, nous proposons un intergiciel pour le déploiement autonomique de logiciels. Autrement dit, l'intergiciel doit être capable de déployer «au-mieux» une application répartie. L'objectif est de réduire les interventions humaines dans le processus de déploiement quel que soit le contexte d'exécution (en particulier dans les environnements ouverts, dynamiques par nature). Nous visons un processus de déploiement de logiciels, auto-configurable et auto-adaptable capable de s'exécuter d'une manière autonome dans un environnement ouvert, qui satisfait un ensemble de contraintes spécifiées par l'administrateur de déploiement et qui est capable de réagir d'une manière automatique aux variations de l'environnement cible de déploiement.

La section suivante décrit l'étude que nous avons menée pour clarifier les concepts liés au déploiement de logiciels.

2.2 Déploiement de logiciels

2.2.1 Définitions

Le déploiement de logiciels est défini dans [van der Hoek 1998, Hall 1999] et [Dearle 2007] comme un processus complexe qui comporte un ensemble d'activités. Il arrive en fin du cycle de vie du logiciel. Le cycle de vie de déploiement regroupe toutes les activités depuis la validation du logiciel par le producteur jusqu'à sa désinstallation des sites cible de déploiement, cela inclue les activités de mise en place sur les sites cible comme l'installation et l'activation ainsi que les activités de maintenance du logiciel comme la reconfiguration et la mise à jour.

2.2.2 Cycle de vie de déploiement

Dans [van der Hoek 1998], les auteurs ont défini le cycle de vie de déploiement comme un processus formé de huit activités indépendantes et un ensemble de transition entre les activités. Les activités en question sont considérées comme les activités minimales devant appartenir au processus de déploiement de logiciels. Pour cela, le processus de déploiement proposé est considéré comme minimal et générique. Les activités appartenant à ce processus sont respectivement : la mise à disposition (Release), l'installation (Install), l'activation (Activate), la désactivation (DeActivate), la mise à jour (Update), la désinstallation (DeInstall), l'adaptation (Adapt) et la fin du support (DeRelease).

La figure (2.2), représente la définition du cycle de vie de déploiement.

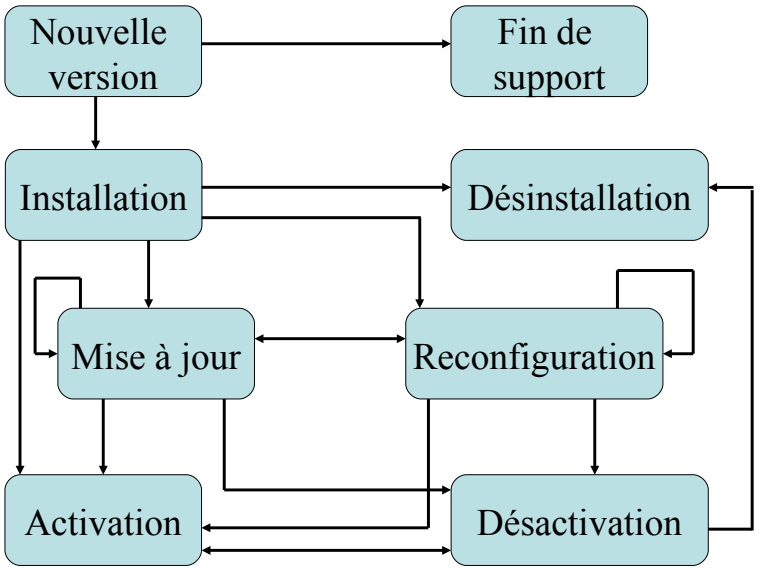

FIGURE 2.2 – Cycle de vie de déploiement définie dans [van der Hoek 1998]

2.2.2.1 Activités de déploiement

La définition des activités du cycle de vie de déploiement sont les suivantes :

- **La mise à disposition** [**van der Hoek 1998**, **Hall 1999**] : cette activité est appelé aussi la diffusion logiciel, elle fait l'interface entre le processus de développement et le processus de déploiement du logiciel. Elle comporte des opérations qui permettent de préparer le système logiciel afin qu'il puisse être correctement assemblé au niveau du site cible de déploiement. Elle inclut les étapes de paquetage et de mise en place du logiciel. Le résultat de cette activité est une archive (paquetage) qui contient les composantes du logiciel, ses dépendances, ses contraintes, ainsi que des informations nécessaires aux autres activités de déploiement. Ce paquetage va être déployé par la suite sur un ou plusieurs sites cibles de déploiement.
- **L'installation du logiciel** [**van der Hoek 1998**, **Hall 1999**] : une activité qui permet d'insérer un logiciel dans un ou plusieurs sites cibles de déploiement. Elle est considérée comme l'activité la plus complexe et la plus importante du cycle de vie de déploiement. Elle inclut la résolution des dépendances du logiciel à déployer en matière de ressources physique et logicielles nécessaires pour son utilisation. Autrement dit, elle permet de trouver et d'assembler toutes les ressources nécessaires au logiciel à installer en in-

terprétant les informations fournies dans l'archive, cela en inspectant le / les sites cibles pour configurer correctement le système logiciel. Elle comporte aussi, le transfert des unités de déploiement vers le / les sites cibles ainsi que leur configuration pour rendre le logiciel déployé activable.

- **La désinstallation** [van der Hoek 1998, Hall 1999] : une activité qui consiste à remettre à l'état initial le système et enlever toutes les modifications introduites sur toutes les machines cibles par les différentes activités du processus du déploiement. Au moment de désinstallation, le système doit gérer les ressources et dépendances partagées avec d'autres logiciels.
- **L'activation** [van der Hoek 1998, Hall 1999] : cette activité consiste en le lancement de l'exécution d'un logiciel installé. Elle peut se résumer simplement à l'invocation d'une méthode dans sa forme la plus simple. Par contre, dans sa forme la plus complexe, il peut être nécessaire d'activer des éléments de l'environnement cible comme les dépendances logicielles ou de serveurs avant de pouvoir activer l'application.
- **La désactivation** [van der Hoek 1998, Hall 1999] : une activité utilisée en général avant une activité de mise à jour, de désinstallation ou même de reconfiguration. Elle consiste à interrompre l'exécution du logiciel ou d'une partie du logiciel (un composant) déployé.
- **La mise à jour** [van der Hoek 1998, Hall 1999] : une activité qui consiste a modifier un logiciel déjà installé, en installant par exemple une nouvelle version du logiciel. La mise à jour peut être de type statique ou dynamique. Elle inclut la désactivation du logiciel déployé (dans le cadre de la mise à jour statique), le transfert et l'installation des nouveaux paquetages dans les sites cibles et enfin l'activation des parties désactivées. Elle est considérée comme un cas particulier de l'installation, mais elle reste moins complexe puisque la plupart des ressources nécessaire ont déjà été obtenues durant l'activité d'installation.
- **La reconfiguration** [van der Hoek 1998, Hall 1999] : cette activité permet de modifier un logiciel déjà installé en sélectionnant une configuration différente (pas similaire) a la configuration existante. Elle peut être par exemple, l'ajout ou la suppression d'un composant, la réparation d'une panne ou simplement le changement des paramètres de configuration. Elle est utilisée généralement suite à un ou plusieurs événements nécessitant une opération d'adaptation afin d'ajuster le comportement du logiciel déployé. Elle à pour but, la correction de situation pour maintenir en vie le logiciel déployé.
- **La fin du support** [van der Hoek 1998, Hall 1999] : une activité qui intervient à la fin du cycle de vie de déploiement. Elle est différente de la désinstallation. Elle illustre le fait d'informer l'utilisateur final que le logiciel déjà déployé n'est plus disponible pour des futures installations.

2.2.3 Unités de déploiement

Le terme déploiement de logiciels est souvent utilisé et laisse à penser que l'unité de base de la phase de déploiement est le logiciel. Or les éléments qui sont effectivement déployés sont des unités de déploiement.

Une unité de déploiement est une entité logicielle archivée (packagée) de façon à pouvoir être déployée. Autrement dit, une unité de déploiement est une archive exécutable dans un environnement d'exécution sans le besoin d'intervention d'un opérateur humain pour le modifier pour le rendre prêt à être déployé. Ainsi, un logiciel acquis peut être une unité de déploiement en lui-même ou être constitué d'un ensemble d'unités de déploiement. L'unité de déploiement peut être aussi un composant logiciel ou une application composée d'un ou plusieurs composants.

2.2.4 Types de déploiement

Dans la littérature, le déploiement de logiciels peut se réaliser par deux types de stratégies : `push` et `pull`. Dans le mode pull, le déploiement est réalisé à la demande des sites cibles de déploiement. Tandis que dans le mode push, l'administrateur de déploiement commence le processus à partir d'un site initiateur de déploiement.

Le déploiement, peut être sensible au contexte d'exécution de l'application déployé. La gestion du contexte peut être réalisé par le gestionnaire de déploiement qui est alors le responsable de la récolte et la gestion des informations de contexte. La gestion des informations de contexte peut aussi être assurée par l'application déployée elle-même, en implémentant des modules de gestion et adaptation au contexte de l'exécution.

Il existe deux types de déploiement, le déploiement initial et le déploiement ultérieur. Le déploiement initial, fait référence au premier déploiement de l'application dans l'environnement cible. Tandis que le déploiement ultérieur, est le résultat d'une activité de reconfiguration (évolution du système) après une panne ou une déconnexion par exemple.

2.2.5 Plan de déploiement

Un plan de déploiement traite le déploiement d'un ensemble d'unités de déploiement (composants logiciels par exemple) sur un ensemble de sites cibles de déploiement. Il peut être unitaire ou global. Un plan de déploiement unitaire, permet d'établir une liste d'activités à réaliser pour déployer une unité sur une machine cible. Tandis qu'un plan de déploiement global, est concerne le déploiement de N unités de déploiement sur M sites cibles de déploiement. Il est établi à partir d'un

ensemble de plan de déploiement unitaire (couple <unité de déploiement, machine cible>).

2.2.6 Contrôle du déploiement

Le contrôle du déploiement peut être centralisé ou décentralisé. Le contrôle centralisé, permet de facilité la mise en œuvre du processus de déploiement. Cependant, il n'est pas adapté au déploiement logiciel à grand échelle et aux environnements ouverts. En effet, si on réalise un déploiement sur plusieurs millier de machines, le contrôle centralisé peut atteindre très vite les limites de l'unité centrale de gestion à cause du nombre très important de machines impliquées et les messages échangés.

Dans le contrôle décentralisé, il n'existe pas une unité centralisé pour la gestion du déploiement. Chaque site cible, dispose d'un gestionnaire de déploiement qui collabore avec les gestionnaires voisins pour réaliser le déploiement. L'inconvénient de ce type de contrôle est la difficulté de mise en œuvre, mais il a l'avantage d'être adapté aux environnements répartis à grande échelle ouverts (par nature il permet de répartir les décisions d'adaptation au niveau des sites cibles de déploiement) ; Pour cette thèse, une approche hybride qui offre un contrôle centralisé limité avec un contrôle décentralisé au niveau des sites cibles est adoptée pour permettre des décisions d'adaptations hiérarchiques et décentralisées.

2.3 Plates formes de déploiement de logiciels

2.3.1 Introduction

Après avoir présenté les définitions du processus de déploiement et du cycle de vie de déploiement nous présentons dans ce qui suit un ensemble de plates-formes de déploiement de logiciels. Ces outils permettent de couvrir une partie ou toutes les activités présentées précédemment et disposent de plusieurs types d'unités de déploiement (logiciel packagé, composant logiciel...). Les outils de déploiement sont des outils de déploiement d'application monolithique (application constitué d'une seul entité à déployer sur un seul site) tel que RPM [Bailey 2000] et des outils de déploiement d'applications réparties (par exemple des applications à base de composants logiciel).

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 [Szyperski 2002]. 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.

Les plates-formes de composants logiciels telles que .NET [Microsoft 1999], D&C [Specification 2006], CCM [Object Management Group 2006], EJB [Burke 2006] et OSGi [The OSGi Alliance 2009] offrent leurs propres solutions pour le déploiement de logiciels (composants). Nous présentons dans la suite ces outils de déploiement et des d'autres plates-formes de déploiement de logiciels et de composants logiciels comme DeployWare [Flissi 2008], RPM [Bailey 2000], ORYA [Cunin 2005, Merle 2005], kalimucho [Louberry 2011], Jade [De Palma 2008], ProActive [Baduel 2006] et MADME [Dearle 2010].

2.3.2 Corba Component Model (CCM)

La spécification CCM [Object Management Group 2006] de l'OMG, présente un modèle de composant logiciel et son modèle de déploiement. Pour mieux comprendre la phase de déploiement de ce modèle, nous présentons d'une façon brève le modèle de composant CORBA.

Un composant CORBA est défini par un ensemble de ports (interfaces) et d'attributs. Chaque port possède un ensemble d'opérations, il peut être utilisé par d'autres composants ou utilise lui-même d'autres composants, il s'agit, respectivement, de ports fournis et de ports requis par le composant. Les ports peuvent avoir quatre types différents :

- Une facette qui représente une interface fournie par un type de composant et qui est utilisée par des clients en mode synchrone.
- Un réceptacle qui est une interface utilisée en mode synchrone.
- Un puits d'événement qui représente une interface utilisée par les clients du composant et qui fournit une interface en mode asynchrone.
- Une source d'événements qui est une interface requise en mode asynchrone.

La figure 2.3 illustre l'aspect générale du modèle abstrait de composant CORBA.

La spécification CCM précise aussi qu'un ensemble de composants, interconnectés ou non, est appelé un assemblage. L'entité qui permet la gestion des instances d'un même type de composant est la maison de composant. Elle gère le cycle de vie des instances en offrent une fabrique d'instances de composants et des opérations de recherche des composants. Les maisons de composants prennent en charge aussi la création et la destruction de ces instances. Un type de maison doit spécifier le type de composant qu'il va gérer. Notons que plusieurs types de maison peuvent gérer un même type de composant, mais pas les mêmes instances.

Les composants CORBA s'exécutent dans un conteneur qui représente un support d'exécution générique qui peut accueillir différents types de composants. Il offre des services système et gère plusieurs aspects non fonctionnels des types de composant.

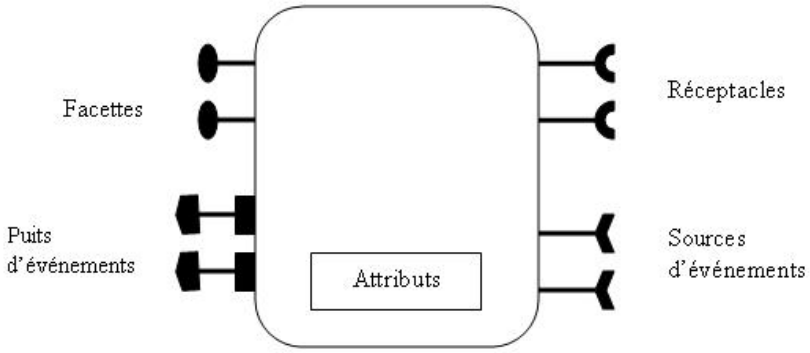

FIGURE 2.3 – Modèle abstrait de composant CORBA

Le modèle de déploiement offre des moyens pour automatiser la diffusion et la mise en place d'une application répartie. Le modèle de déploiement dans CCM (Corba Component Model) utilise deux types de paquetage, le paquetage de composant et le paquetage d'assemblage. Dans le cadre de CORBA un paquetage est une entité qui regroupe un ensemble d'implémentations de composants et un ou plusieurs descripteurs écrits en OSD [1] (Open Software Description) qui est un langage basé sur XML et qui décrit le contenu du paquetage. Le paquetage de composant *"Component Package"* représente l'unité de déploiement, il contient une ou plusieurs implantations du composant et trois descripteurs : **le descripteur logiciel de composant, le descripteur de composant CORBA et le descripteur de propriétés du composant**. Ce paquetage permet le déploiement d'un (s'il contient un seul composant) ou plusieurs composants (s'il contient un assemblage de composants).

Le descripteur logiciel de composant contient les informations générales concernant un composant : son nom, sa licence, la référence sur le fichier IDL3 qui décrit le type de composant et des références sur les autres descripteurs de déploiement. Ce descripteur contient aussi des informations sur les implémentations du composant, de ses maisons ainsi que leur point d'entrée. Il fournit une description générale pour chaque implémentation, une référence sur le code de cette implémentation, le nom du compilateur utilisé, le langage de programmation et des informations sur les dépendances.

Le descripteur de propriétés du composant contient des valeurs pour les attributs de configuration du composant, il est utilisé pour configurer un composant ou une

1. http://www.w3.org/TR/NOTE-OSD

maison en fournissant des propriétés modifiables par l'utilisateur.

Le descripteur de composant CORBA est défini pour chaque implémentation du composant. Il décrit la structure d'un composant : les facettes, les réceptacles, les sources d'événements et les puits d'événements ainsi que l'héritage. Ce descripteur permet la connexion des différents composants entre eux une fois la phase de déploiement déclenché.

Le paquetage d'assemblage permet de déployer des composants dépendants les uns des autres. Il contient un plan de déploiement qui décrit l'instanciation des composants et leurs interconnexions. Le plan de déploiement (plan d'assemblage) spécifie pour chaque instance de composant son emplacement, son implémentation et un descripteur de propriétés. Il définit des regroupements logiques pour les composants qui seront projetés sur des sites physiques dans l'étape de déploiement.

Le déploiement est réalisé à l'aide d'un outil qui projette les composants et les assemblages sur des sites cibles. Les quatre étapes principales du déploiement sont, le choix des sites d'installation, l'installation des implémentations des composants, l'instanciation des maisons et des composants et enfin la connexion des composants dans le cas où on a un assemblage.

La figure 2.4 présente le processus de déploiement des composants CCM en treize points comme présenté dans la spécification [Object Management Group 2006].

L'application de déploiement commence par l'interrogation d'utilisateur pour connaitre les sites cible de déploiement (1). Une fois les sites connus, les paquetages de composants sont installés sur les sites cibles en utilisant l'objet *ComponentInstallation* de chaque site concernés (2). Ensuite, la fabrique d'assemblage *AssemblyFactory* est utilisée pour la création d'une instance de l'objet *Assembly* sur un seul site pour toute l'application (3).

L'objet *Assembly* démarre l'assemblage de l'application et coordonne le reste des étapes de déploiement (4). Il crée pour cela un serveur de conteneurs à travers un activateur de serveurs puis crée à son tour une instance de conteneur dans ce serveur de conteneur du composant.

Assembly envoie par la suite une requête vers le conteneur pour qu'il créé les maisons des composants, l'interface du conteneur fournit une opération pour installer une maison de composants, une fois l'implémentation de la maison est installée, le conteneur crée une instance et retourne sa référence. L'instance d'*Assembly* utilisera la maison de composants pour instancier les composants. Une fois tous les composants installés, l'instance *Assembly* connecte les différentes instances des composants en se basant sur le descripteur d'assemblage, puis il configure ces instances afin de notifier aux instances de composants que les connexions initiales ont été effectuées.

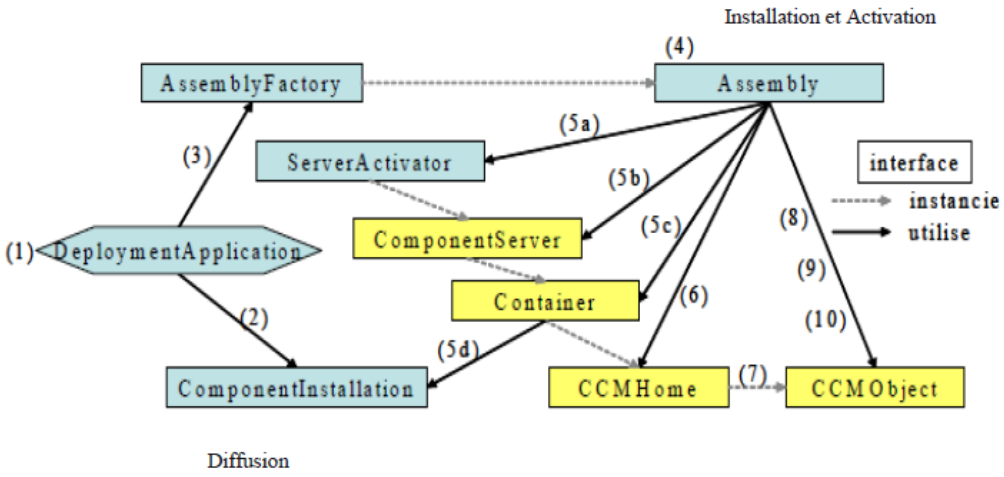

FIGURE 2.4 – Étapes de déploiement dans CCM

Le tableau 2.1 présente une synthèse du processus de déploiement dans le cadre de CCM.

TABLE 2.1 – Synthèse du déploiement de CCM

	CCM
Unité de déploiement	Le Component Package, il contient les implantations du composant et trois descripteurs.
Activité de déploiement	L'installation, l'activation et la désactivation.
Résolution des dépendances	Les dépendances des composants sont gérées à la main.
Sites cibles et plan de déploiement	Le plan de déploiement est géré par l'utilisateur. L'utilisateur choisi manuellement les sites cibles dans la première étape du processus de déploiement.

2.3.3 Deployment and Configuration, D&C

D&C (Deployment and Configuration of Component-based Applications) représente un modèle de déploiement d'applications à base de composants indépendant de la plate-forme d'exécution proposé par l'OMG [Specification 2006]. Il est intéressant

de noter que les définitions de la phase de déploiement de CCM et cette spécification sont naturellement liées. La spécification D&C voit la phase de déploiement comme un processus linéaire et séquentiel qui comporte les activités d'installation, de configuration, de planification, de préparation et de lancement de l'exécution. Elle définit trois pré-conditions au processus de déploiement :

- La première pré-condition stipule que le fournisseur du logiciel doit packager le logiciel conformément à la spécification D&C. Ainsi, tout package doit contenir des métadonnées décrivant le logiciel et les binaires exécutables des différents éléments composant le logiciel. Le package doit, ensuite, être publié et mis à la disposition via un CDROM ou une URL par exemple.
- La seconde pré-condition impose l'existence et la disponibilité d'un environnement cible. Cet environnement est une infrastructure composée de systèmes distribués (ordinateurs, réseaux, etc.) sur laquelle le logiciel est destiné à être exécuté.
- La troisième et dernière pré-condition rend obligatoire la présence d'un dépôt (repository). Ce dépôt étant, au minimum, une zone dans laquelle le logiciel packagé est stocké avant le début du processus de déploiement.

L'unité de déploiement est le logiciel packagé selon la spécification et qui contient un ensemble de méta-informations de déploiement. 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 but de la spécification D&C est de fournir un processus de déploiement indépendant de toutes les plates-formes de composants.

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. En effet, un composant peut avoir une implémentation concrète dans un artefact ou peut être implémenté d'une manière récursive par un assemblage. Pour un même composant, plusieurs implémentations peuvent être supportées, a chaque implémentation est associé un ensemble de descriptions d'artefacts (les contraintes sur l'emplacement de l'implémentation, les paramètres d'exécution et les dépendances).

Le modèle de sites cibles permet de décrire le domaine où les applications seront déployées en terme de nœud, d'interconnexions entre les nœuds et des éléments de routage.

Le modèle de données de gestion d'exécution du déploiement représente un moyen pour décrire un plan de déploiement. Ce plan décrit la manière de créer des instances de composants à partir des artefacts, comment les interconnecter, et où les instancier.

En ce qui concerne la résolution de dépendances, la spécification ne décrit ni ne propose de mécanisme particulier.

Le processus de déploiement d'une application D&C tel qu'il est décrit par la spécification est constitué des étapes suivantes :

- L'installation qui inclut la phase de paquetage de l'application. Elle définie la récupération et l'acquisition d'un paquetage logiciel (software package) publié et son rapatriement dans un dépôt logiciel.
- La configuration qui permet de créer l'aspect fonctionnel de l'application.
- La planification : dans cette phase le logiciel de déploiement prend en entrée les dépendances logicielles et matérielles ainsi que les sites cibles ce qui permet de prendre la décision d'où et comment déployer le logiciel.
- La préparation qui permet d'installer le logiciel dans l'environnement cible.
- L'activation consiste à démarrer l'exécution du logiciel. Dans cette étape le logiciel de déploiement réalise l'instanciation des différents composants ainsi que l'établissement des liaisons entre les composants instanciés. Plus précisément la préparation du plan d'exécution qui retourne comme résultat un objet qui peut mettre le plan en action à plusieurs reprises.

Nous précisons ici, que l'installation dans le cadre de L'OMG consiste à insérer une unité de déploiement dans un dépôt et non pas dans l'environnement d'exécution. La phase d'activation de l'application est divisée en deux étapes : la première étape retourne des références vers les ports offerts par l'application et la deuxième étape attribue des références aux ports utilisés par l'application.

Le tableau 2.2 présente une synthèse des différents points que nous avons abordés dans cette sous-section sur le déploiement dans le cadre de D&C.

Table 2.2 – Synthèse du déploiement de D&C

	D&C
Unité de déploiement	Un paquetage selon la spécification D&C et qui contient des méta-informations de déploiement.
Activité de déploiement	L'installation, l'activation, la désactivation, et la configuration.
Résolution des dépendances	La spécification ne décrit ni propose un mécanisme particulier.
Sites cibles et plan de déploiement	Les sites cibles et le plan de déploiement sont gérés dans le modèle des sites cibles.

2.3.4 Enterprise Java Beans EJB

Ce modèle de déploiement est spécifique au modèle abstrait de composant EJB. Un composant EJB [Burke 2006] 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 beans sont les briques de base pour la programmation d'applications destinées à être déployées du côté serveur.

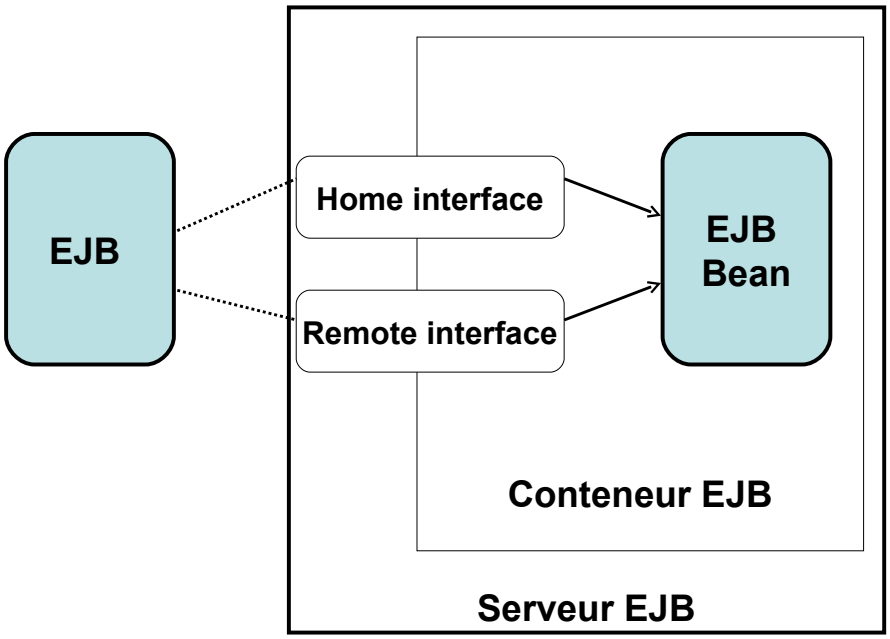

FIGURE 2.5 – Modèle abstrait des composants EJB

Les composants EJB son hébergés dans un conteneur EJB (voire la figure 2.5) ce dernier gère tout les aspects non fonctionnel des composants et qui permet leur exécution. Les principaux conteneurs EJB existants aujourd'hui sont Jonas [2], JBoss [3] et Apache Geronimo [4].

Un composant EJB est diffusé dans un paquetage. Il contient des références sur les classes java implémentant les composants ainsi que leurs maisons et peut contenir un descripteur de déploiement écrit en XML.

L'unité de déploiement dans le cadre des EJB est une archive qui peut être un fichier JAR dans le cas d'un ensemble composé uniquement de beans et un

2. http://wiki.jonas.objectweb.org/xwiki/bin/view/Main/JOnAS5
3. http://www.redhat.com/jboss/
4. http://geronimo.apache.org/

fichier EAR dans le cas d'applications contenant à la fois des beans et une partie de l'IHM web. Le descripteur de déploiement comporte deux types d'information, des informations sur le composant (nom du composant, classe d'implémentation du composant, l'interface de sa maison) et des informations sur l'assemblage des composants.

Dans EJB 3.0 les descripteurs de déploiement sont remplacés par des annotations placées directement dans le code source des EJB (exemple : l'injection de dépendances d'un Bean via l'annotation EJB). Tous les paramètres de déploiement sont dotés de valeurs par défaut qui suffissent dans la plupart des cas pour le déploiement d'une application. Notons qu'il est possible dans EJB 3.0 d'utiliser une combinaison entre les annotations et le descripteur de déploiement. De plus, plusieurs implémentations d'interface on été remplacées par des annotations java. Le déploiement d'une archive est réalisé sur une plate-forme J2EE qui comporte plusieurs interfaces et outils permettant la réalisation du déploiement.

Le cycle de vie d'un bean est constitué de quatre phases, à savoir, le développement, le déploiement, l'exécution (appelé aussi disponibilité) et le retrait (undeployment). Dans la phase de développement le bean est développé puis packagé dans une archive java.

La phase de déploiement selon cette spécification consiste en :

- La configuration : dans laquelle l'outil de déploiement suit les instructions d'assemblage fournies par l'application (dans le descripteur de déploiement s'il existe) en résolvant toutes les dépendances déclarés par l'application.
- La distribution : dans cette étape, l'archive de l'application ainsi que les informations de déploiement sont installés sur les serveurs via l'API de déploiement. Autrement dit, cette activité consiste en l'installation de l'archive Java dans un répertoire du conteneur EJB.
- Le démarrage de l'exécution : dans cette étape l'outil de déploiement demande aux serveurs de démarrer l'exécution de l'application. La troisième phase (phase d'exécution) commence lorsque le conteneur de l'archive java active l'unité de déploiement afin de lancer l'exécution des beans qu'elle contient.
- La quatrième et dernière phase est la phase de retrait de l'unité de déploiement. Elle consiste à enlever l'unité de déploiement du répertoire du conteneur dans lequel elle a été déposée.

Parmi les interfaces les plus intéressantes pour le déploiement dans le cadre de la plate-forme J2EE on peut citer l'interface DeploymentManager. Cette interface permet de configurer, distribuer, activer, désactiver, désinstaller et superviser l'application. La configuration consiste en des mises à jour du descripteur de déploiement du paquetage de l'application, les mises à jour sont réalisées grâce à l'API de déploiement qui permet de demander des informations spécifique au déploiement et l'interaction avec le concepteur de l'application. La distribution de l'application

consiste en l'installation du paquetage (des paquetages) configuré à l'aide de serveurs J2EE. La spécification EJB ne fait pas de proposition pour la résolution de dépendances et des contraintes de déploiement. Ainsi, si un bean à besoin d'un autre bean et si ce dernier n'est pas installé et activé, alors, le bean ne pourra pas fonctionner correctement ou fonctionnera d'une manière partielle. Les activités du cycle de vie de déploiement dans la spécification EJB sont l'installation, l'activation, de désactivation et de désinstallation. La spécification ne couvre néanmoins pas la totalité du cycle de vie de déploiement puisqu'elle n'apporte aucune précision ni sur les mises à jour (statiques ou dynamiques), ni sur les possibilités d'adaptation et de reconfiguration (l'évolution et le dynamisme du système).

Le tableau ci-dessous (2.3) présente une synthèse des différents points que nous avons abordés dans le cadre du déploiement dans les EJB.

Table 2.3 – Synthèse du déploiement d'EJB

	EJB
Unité de déploiement	Archive jar contenant soit un ou plusieurs beans, soit une archive ear qui comporte un ensemble de beans et une IHM web.
Activité de déploiement	L'installation, l'activation, la désactivation et la désinstallation.
Résolution des dépendances	Aucun mécanisme de résolution automatique de dépendances et de contraintes.
Site cibles et plan de déploiement	Le plan de déploiement et les sites cibles sont gérés à la main par l'utilisateur.

2.3.5 Microsoft .Net Framework

Le Framework .NET [Hashimi 2006] est proposé par Microsoft pour résoudre les problèmes de déploiement et de Versionnement dus au DLLs. COM et DCOM sont deux modèles de composants de Microsoft [Microsoft 1999]. Un composant DCOM est une entité binaire avec des interfaces et un mode d'interaction implémenté et distribué avec une bibliothèque DLL qui est déployé manuellement sur les machines cibles puisqu'aucun processus de déploiement n'est prévu dans COM.

Le déploiement retenu par le Framework .Net est constituée par les activités suivantes : l'installation, la maintenance et la désinstallation. Les activités d'activation

et de désactivation sont supportées par le Framework, mais ne sont pas présentées comme faisant partie du déploiement. Il faut noter que l'activité maintenance définie par le Framework .Net contient les sous activités de mise à jour et de réparation d'applications.

Microsoft .Net Framework propose trois outils pour réaliser effectivement les activités de déploiement retenues. Ces outils sont Windows Installer, ClickOnce et Windows Update. Windows Installer est l'outil de déploiement standard. Il couvre l'installation, la maintenance (mise à jour) et la désinstallation d'applications. L'outil ClickOnce couvre les activités d'installation, de mise à jour et de désinstallation. Windows Update, se focalise sur l'activité de mise à jour.

Pour résoudre les problèmes de versionnements, les applications Windows utilisent des DLL privées et DLL publiques et afin de simplifier le processus de déploiement, Microsoft a créé une nouvelle entité de déploiement appelé assemblage *"assembly"* qui est constituée par le code des composants, les ressources des composants (bibliothèques) et un manifeste. Concrètement, un *"assembly* est soit un fichier exécutable *".exe*, soit un fichier .dll.

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly xmlns="urn:schemas-microsoft-com:asm.v2" xmlns:asmv1="urn:schemas-microsoft-com:asm.v1"
xmlns:asmv2="urn:schemas-microsoft-com:asm.v2" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
manifestVersion="1.0" >
  <asmv1:assemblyIdentity name="Sample.application" version="1.0.0.0" publicKeyToken="3cb8aba78866dbbf"
language="neutral" processorArchitecture="msil" />
  <asmv1:description asmv2:publisher="Microsoft" asmv2:product="Sample" > This is a sample </asmv1:description>
  <deployment install="true">
    <subscription> <update> <beforeApplicationStartup /> </update> </subscription>
    <deploymentProvider codebase="http://myserver/Sample.application" />
  </deployment>
  <dependency>
    <dependentAssembly codebase="Sample_1.0.0.0\Sample.exe.manifest" size="3422">
      <assemblyIdentity name="Sample.exe" version="1.0.0.0" publicKeyToken="3cb8aba78866dbbf" language="neutral"
processorArchitecture="msil" />
      <hash>...</hash>
    </dependentAssembly>
  </dependency>
  <dsig:Signature Id="StrongNameSignature">...</dsig:Signature>
</asmv1:assembly>
```

FIGURE 2.6 – Exemple de manisfeste de déploiement

Le manifeste contient des informations sur le nom, le numéro de version, la clé publique, la liste de tous les fichiers et une liste des références statiques d'assemblage. La figure 2.6 illustre un manifeste de déploiement pour l'outil ClickOnce. Les assemblages sont de deux types privés et publiques. Ils sont partagés par plusieurs applications et qui doivent suivre des règles rigoureuses de nommage pour avoir des noms uniques et pouvoir ainsi s'exécuter simultanément.

Les deux types d'assemblages permettent de réaliser :

– Un déploiement privé qui consiste à installer l'application ainsi que les bibliothèques non partagées (privées) par d'autres applications dans un répertoire

privé.
– Un déploiement qui consiste à copier les assemblages des applications dans un répertoire appelé Global Assembly Cache. Cela permet le partage d'assemblages de DLLs.

Le tableau ci-dessous (2.4) présente une synthèse des points abordés dans cette sous-section.

Table 2.4 – Synthèse du déploiement de .NET

	.NET
Unité de déploiement	L'Assembly qui peut contenir une partie de l'application ou une application entière.
Activité de déploiement	L'installation, l'activation et la désactivation, la mise à jour et la désinstallation.
Résolution des dépendances	Aucun mécanisme de résolution automatique de dépendances.
Site cibles et plan de déploiement	Le plan de déploiement est géré par l'utilisateur dans un site cible de déploiement.

2.3.6 Open Service Gateway Initiative OSGi

Open Service Gateway Initiative (OSGi) est une spécification ouverte qui définit les API de serveurs embarqués destinés à héberger des services qui peuvent être installés, activés, mis à jour, désactivés et désinstallés sans interruption de service du serveur embarqué [The OSGi Alliance 2009].

OSGi est une plate-forme java dont l'unité de déploiement est appelée un Bundle. Cette plate-forme se divise en deux parties, un Framework OSGi et des services standards. Les applications dans OSGi sont construites à partir des Bundles connectés à travers les services fournies par ces bundles, les services peuvent dynamiquement apparaître ou disparaitre au courant de l'exécution de l'application. Un Bundle est une archive java au format JAR qui contient un fichier qui le décrit appelé manifeste, un ensemble de classes et des bibliothèques de code. Les mécanismes d'administrations dans le Framework OSGi permettent l'installation, l'activation, la désactivation, la mise à jour et le retrait d'un Bundle. Un Bundle est l'unité de déploiement dans le cadre de la plate-forme OSGi.

La figure 2.7 illustre la structure d'un Bundle.

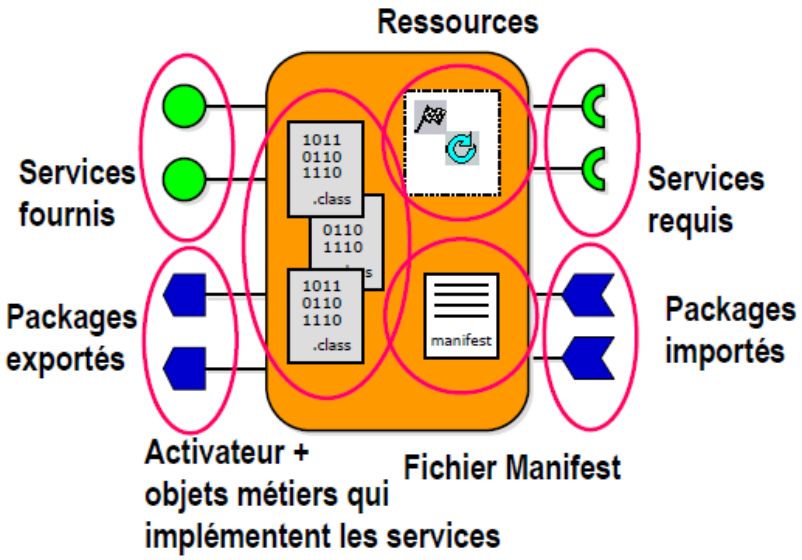

FIGURE 2.7 – Structure générale d'un Bundle OSGi

Un Bundle peut avoir l'un des états suivants : installé, résolu, actif, arrêté, en démarrage et désinstallé. Les Bundles disposent d'un mécanisme permettent l'importation et l'exportation de code sous forme de paquetage, ce mécanisme permet l'accès aux services du bundle sans avoir le besoin d'intégrer le code de ces services dans chaque bundle.

OSGi permet de créer des applications mais ne peut pas décrire l'architecture des applications ce qui rend difficile la vision globale d'applications constituées de plusieurs composants.

Les activités de déploiement supportées par OSGi sont l'installation, l'activation, la mise à jour, la désactivation et la désinstallation de bundles. L'environnement d'exécution fournit des mécanismes permettant de réaliser les activités de déploiement. Les différentes activités sont réalisées a l'aide de plusieurs commandes. On peut citer par exemple *install url* qui permet d'installer un bundle à partir d'un url, *start id* qui permet de démarrer un bundle invoquant la méthode *start()* de l'activateur, *update id* qui permet de faire une mise à jour d'un bundles et *obr* qui permet d'installer et démarrer un bundle et résoudre ses dépendances.

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.

Le tableau 2.5 synthétise les points que nous avons abordés dans cette sous-section.

TABLE 2.5 – Synthèse du déploiement d'OSGi

	OSGi
Unité de déploiement	Un bundle ou une application constituée de plusieurs bundles.
Activité de déploiement	L'installation, l'activation, la désactivation, la mise à jour et la désinstallation.
Résolution des dépendances	Présence de mécanismes de gestion automatique des dépendances de paquetage.
Sites cibles et plan de déploiement	Les sites cibles et le plan de déploiement sont gérés manuellement par l'utilisateur.

2.3.7 Red Hat Package manager RPM

Nous allons maintenant nous intéresser au déploiement de logiciels dans le cadre des systèmes Linux et précisément au **RPM Red Hat Package manager**. Le RPM [Bailey 2000] est un gestionnaire de paquets, qui prend en charge l'installation, l'interrogation, la vérification, la mise à jour et la suppression des paquets. Le RPM comporte un dépôt qui contient des détails sur les paquets qui ont été installés sur une distribution Linux.

Un paquet RPM contient en général des binaires exécutables et un fichier de configuration et de documentation. Les dépendances implicites entre un paquet et le système d'exploitation et l'architecture matérielle sont traités en utilisant un ensemble de bibliothèques C, et en annotant les paquets avec l'architecture pour laquelle ils ont été compilés. Chaque paquet RPM est étiqueté avec une étiquette (Label) qui contient le nom du paquet, sa version, et l'architecture cible. Chaque RPM contient quatre sections appelées la tête, la signature, l'en-tête et les archives.

La tête est utilisée par des commandes Unix comme un fichier de commande. Chaque paquet RPM contient une ou plusieurs en-têtes de différentes structures, représentées comme un ensemble d'entrées indexées contenants des informations sur certaines références. La signature contient des informations cryptographiques qui sont utilisées pour la vérification de l'intégrité et l'authenticité de l'en-tête et de l'archive contenues dans le paquet.

Les fichiers RPM peuvent également contenir un certain nombre de scripts écrits dans le langage de script Unix standard. Ces scripts sont des scripts de construction de logiciel, des scripts d'installation, des scripts de suppression et des scripts de vérification. Les scripts de construction sont responsables du dépaquetage des fichiers

sources, la construction de paquet, l'installation et la désinstallation du logiciel, et la préparation de la post-installation. Les scripts d'installation et de désinstallation sont exécutés dans quatre reprises avant et après l'installation et avant et après la désinstallation. Le script de post-installation permet de vérifier que l'installation est correcte.

Un ensemble d'outils de haut niveau ont été construits en utilisant les RPM. Par exemple YUM (Yellowdog Updater Modifie) est conçu pour déterminer les dépendances inter-paquets et pour automatiser l'installation de paquets. Il permet aussi la gestion d'une ou plusieurs machines sans avoir à configurer manuellement chaque machine en utilisant les RPM.

Il faut aussi souligner que le point fort de ce type d'outils est qu'ils permettent de prendre en charge les dépendances (résolution automatique) pour l'installation et la désinstallation ou de mettre à jour les paquetages installés avec les dernières versions existantes. Notons que ce type d'outil, offre un support très limité pour le déploiement à grande échelle et pour le déploiement des systèmes distribués.

Le tableau (2.6) ci-dessous présente une synthèse des caractéristiques de déploiement dans le cadre des RPM.

Table 2.6 – Synthèse du déploiement d'RPM

	RPM
Unité de déploiement	L'unité de déploiement est un paquet RPM qui contient des binaires exécutables, un fichier de configuration et de documentation.
Activité de déploiement	L'installation, la désinstallation et la mise à jour.
Résolution des dépendances	Résolution automatique des dépendances des paquets.
Sites cible et plan de déploiement	Le /les sites cibles sont désigné par l'utilisateur et le plan de déploiement est généralement géré manuellement pour un seul site.

2.3.8 DeployWare

DeployWare [Flissi 2008] est un Framework utilisé pour l'administration, la description, et le déploiement de logiciels distribués de manière automatique. Il est

constitué d'une couche logicielle complexe, hétérogène et distribuée sur une infrastructure répartie comme la grille.

Cet outil propose un langage dédié au domaine du déploiement et une machine virtuelle pour ce langage. Cette machine virtuelle est appelée Fractal Deployment Framework (FDF), elle est implantée sous la forme de composants Fractal réifiant les logiciels à déployer ainsi que l'infrastructure répartie. De plus, DeployWare propose de mettre en place un mécanisme de distribution de la machine virtuelle DeployWare permettant de décentraliser le déploiement. Cette approche est basée sur un serveur principal appelé DeployWareMaster (DMMaster) et des serveurs auxiliaires appelés DeployWare Server (DWServer).

Pour réaliser le déploiement d'une application, les développeurs de l'application doivent fournir une description DeployWare de l'application, cela consiste en :

- La déclaration des propriétés du logiciel à déployer, par exemple le chemin vers les archives de l'application et les ports utilisés pour la communication.
- La déclaration des références vers d'autres logiciels. Autrement dit, la déclaration des dépendances logicielles.
- La déclaration des configurations d'installation et de démarrage de l'application.
- La déclaration des différents nœuds physiques dans lesquelles les composants de l'application doivent être déployés.

L'unité de déploiement est une archive qui contient l'application et le descripteur de déploiement. Pour déployer une application la machine virtuelle FDF est installée dans un ou plusieurs nœuds de la Grille qui exécutent à partir de la description DeployWare le processus de déploiement sur les autres nœuds de la Grille.

Une fois l'application déployée avec succès, l'utilisateur peut manuellement démarrer (activer), stopper (désactiver) et désinstaller l'application.

Les différentes activités de déploiement sont faites manuellement via l'outil graphique FDF-Explorer ou peuvent être automatiques en utilisant un scripte qui détermine les activités à réaliser.

Le tableau 2.7 présente une synthèse des caractéristiques de déploiement dans le cadre de DeployWare.

TABLE 2.7 – Synthèse du déploiement de DeployWare

	DeployWare
Unité de déploiement	L'unité de déploiement est une archive qui contient l'application et le descripteur de déploiement.
Activité de déploiement	L'installation, l'activation, la désactivation et la désinstallation.
Résolution des dépendances	Les dépendances des composants sont gérées par l'utilisateur.
Sites cibles et plan de déploiement	Les sites cible et le plan de déploiement sont définis manuellement par l'administrateur du déploiement (la déclaration des configurations de l'installation et la déclaration des nœud physique).

2.3.9 Software Dock

Software Dock est un prototype conçu pour le déploiement de logiciels développé par Richard S. Hall [Hall 1999]. Le cycle de vie de déploiement tel qu'il a été décrit dans ce travail de recherche comporte un ensemble d'activités de déploiement qui sont : *Release*, *DeRelease*, *Installion*, *l'Activation*, *l'Update*, *l'Adapt*, *la DeActivation* et la *DeInstallation*. Dans ce travail, on note l'utilisation de quelques termes comme *Retire* qui désigne l'activité *DeRelease*, le terme *Remove* qui désignent l'activité de *DeInstallation* et enfin les termes *ReConfigure* et *Adapt* qui désignes l'activité *Adapt*.

Dans Software Dock, il y a deux concepts importants : les *Docks*, qui modélisent les clients et les serveurs ainsi que les fichiers descripteurs appelées "*Deployable Software Description* (DSD)" qui modélisent les applications.

A chaque Dock, on associe un "*registre*" et un ou plusieurs agents. Un registre est une base de données que les agents modifient. Chaque modification de la base de données entraîne des événements, auxquels les agents peuvent souscrire. Il y a deux types d'agents, des agents qui peuvent migrer vers d'autres Docks, ce sont les agents externes, comme l'agent d'installation par exemple et des agents qui restent fixés à leurs Docks, ce sont les agents internes, comme l'agent de site qui gère les modifications sur les sites.

La figure 2.8 présente l'architecture de l'outil de déploiement Software Dock.

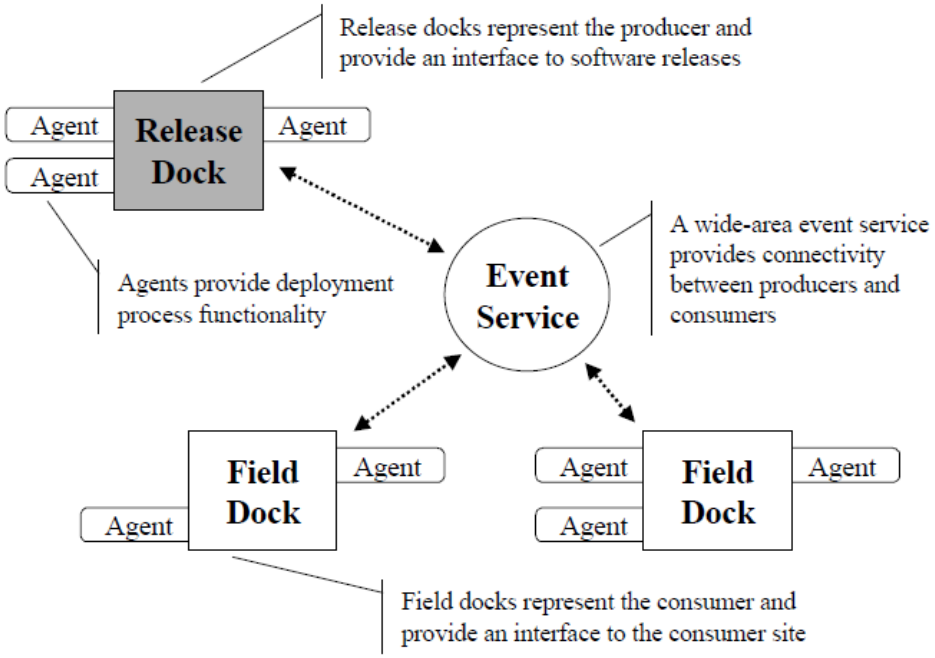

FIGURE 2.8 – Architecture de Software Dock

L'unité de déploiement est un logiciel packagé. Cependant le logiciel en question peut exprimer différentes dépendances via le descripteur DSD. Ces dépendances sont essentiellement des dépendances vers d'autres logiciels qui doivent être déployés avant le déploiement du logiciel par Software Dock.

Les auteurs précisent que le nombre total des dépendances logiciels pour un logiciel à déployer est généralement inférieur à dix. Une autre particularité qu'on peut préciser ici, est que Software Dock prend en compte les dépendances logicielles déjà déployées dans l'environnement cible de déploiement. L'idée est de ne pas déployer un logiciel déjà déployé dans l'environnement cible. Avant de déployer un logiciel avec Software Dock l'administrateur de déploiement doit fournir à Software Dock le descripteur DSD correspondant à l'unité de déploiement. Le DSD contient principalement cinq champs d'informations qui sont respectivement *Configuration, Assertion, Dependency, Artifact* et enfin *Activity.*

- La partie *Configuration* se focalise sur l'expression de la configuration de l'unité de déploiement, ainsi que sur les ressources (physiques et logicielles) qu'elle fournit.
- La partie *Assertion* décrit des propriétés que l'environnement cible de déploiement doit satisfaire (type du système d'exploitation, machines virtuelles...).
- La partie *Dependency* regroupe les informations concernant les dépendances que Software Dock devra satisfaire afin de réaliser le déploiement du logiciel.
- La partie *Artifact* décrit les informations sur le paquetage logiciel (liste de

fichiers, les archives qu'il contient...).
- La partie *Activity* définit les activités de déploiement qui ne sont pas couvertes par Software Dock. Notons que les auteurs ne précisent pas le contenu de ce champ d'information ni comment ces informations sont traitées par Software Dock.

Enfin pour chaque DSD, on spécifie les agents, qui seront fournis par le système (agent d'installation, de mise à jour).

Software Dock couvre principalement les activités d'installation, de désinstallation, de mise à jour et de reconfiguration.

- **L'installation** : Dans cette activité un agent logiciel est envoyé vers un Dock du client, cela se réalise soit par l'utilisateur de Software Dock, soit par un agent logiciel dans le cas où on a des dépendances vers d'autres logicielles non installés sur le site cible. Après avoir résolu toutes les dépendances, l'agent récupère la bonne configuration de l'application, puis il crée un nœud d'application pour le logiciel dans le registre du Dock client, ce qui génère un événement qui va être reçu par l'agent du site qui va par la suite créer un répertoire pour l'application. Ensuite, l'agent d'installation insère dans le registre les nœuds des fichiers et répertoires de l'application, ce qui provoque des événements reçus par l'agent du site qui créé ainsi toute l'arborescence physique de l'application. A la fin de l'installation, un agent de mise à jour est ajouté.
- **La désinstallation** : il s'agit de supprimer une application installée et d'enlever les nœuds des fichiers et répertoires de l'application.
- **La mise à jour** : dés qu'une nouvelle version d'une application est disponible, elle est ajoutée au registre du Dock du coté serveur. Cette insertion génère un événement reçu par la suite par l'agent de mise à jour du Dock client. Ensuite, cet agent consulte le registre du Dock client, puis il communique avec le Dock serveur pour retrouver la bonne configuration du logiciel. Enfin, il insère les nœuds des fichiers et répertoires correspondants. L'agent de site modifie l'arborescence en fonction de cette configuration.
- **La reconfiguration** : l'activité de reconfiguration consiste à réinstaller le logiciel sous une nouvelle configuration. On note qu'il existe une activité appelée l'activité de Vérification de la cohérence d'un logiciel qui consiste à vérifier que tous les fichiers et répertoires de l'application sont présents.

Le tableau 2.8 présente une synthèse des différents points abordés dans le cadre du processus de déploiement de Software Dock.

TABLE 2.8 – Synthèse du déploiement de Software Dock

	Software Dock
Unité de déploiement	L'unité de déploiement est un package qui contient un logiciel.
Activité de déploiement	L'installation, la désinstallation, la diffusion du logiciel, la mise à jour, l'activation, la désactivation et la reconfiguration.
Résolution des dépendances	Résolution automatique des dépendances des paquets.
Sites cibles et plan de déploiement	Les sites cibles sont listés à la main et le plan de déploiement est établie manuellement par l'utilisateur.

2.3.10 ProActive

ProActive [Baduel 2006] est une bibliothèque Java pour le calcul parallèle, distribué et concurrent. Elle permet de déployer une application à grand échelle sur un très grand nombre de machines avec un maximum de transparence. Cette bibliothèque permet la création d'objets distants, la gestion de la mobilité des applications, la gestion des appels synchrones et asynchrones et la gestion des communications entre les groupes. Cela est réalisé par un ensemble de primitives réduit. Le langage Java a été retenu pour permettre la gestion d'hétérogénéité des machines impliquées dans une grille de calcul.

ProActive fournit une API permettant la programmation d'applications distribuées pouvant être déployées sur des réseaux locaux et sur des grilles de calcul interconnectées via Internet.

L'un des concepts de base qui permet le développement des applications distribués avec ProActive sont les objets actifs. Un objet actif dispose d'un thread qui permet l'exécution des méthodes invoquées par d'autres objets, il peut être créé sur n'importe quel hôte utilisé pour le déploiement d'une application, une fois crée l'objet peut être manipulé comme les objets java. Dans ProActive, une application est structurée en sous-systèmes qui se compose d'un objet actif et plusieurs objets passifs. Les objets passifs ne sont pas partageables entre les sous-systèmes.

Un autre concept de base dans ProActive est les nœuds qui sont des entités logiques capables d'accueillir plusieurs objets actifs. Les nœuds permettent aussi d'identifier la machine virtuelle java sur laquelle on souhaite créer les objets actifs.

Le déploiement dans ProActive se base sur le Grid Component Model (GCM). Le bût de ce modèle de composant est de permettre le déploiement de n'importe quelle application n'importe où dans le système est cela sans toucher au code de l'application. Pour cela, ProActive fournit un ensemble de protocoles qui permet l'enregistrement et la découverte de ressources, et offre la possibilité de décrire d'une façon abstraite une application donnée en terme d'activités en précisant les parties parallèles et distribuées.

Le GCM utilise deux descripteurs :

Un descripteur d'application GCMA, qui permet de décrire comment lancer l'application, cela en déclarant les différentes dépendances logicielles de l'application.

Un descripteur de déploiement GCMD, qui permet de décrire les nœuds cibles de déploiement et quel protocole utiliser pour l'installation des différentes dépendances logicielles par la suite.

Pour la réalisation du déploiement, ProActive utilise la notion de nœud virtuel qui est décrit dans le descripteur d'application et qui va être utilisé après la phase d'activation de l'application. Le déploiement d'une application ProActive commence par le chargement du descripteur de déploiement. Ensuite, un processus d'installation à distance est lancé pour installer les dépendances de l'application sur les sites cibles de déploiement déclarés dans le descripteur de déploiement. Enfin, l'application démarre l'exécution des JVMs installées dans chaque nœud et commence l'exécution de l'application active et des objets actifs.

Le tableau 2.9, présente une synthèse des points abordés dans le déploiement avec ProActive.

Table 2.9 – Synthèse du déploiement de ProActive

	ProActive
Unité de déploiement	L'unité de déploiement est les objets actifs et les composants GCM.
Activité de déploiement	L'installation, la désinstallation, l'activation, et la désactivation.
Résolution des dépendances	Résolution automatique des dépendances.
Sites cibles et plan de déploiement	Les sites cibles sont décrits dans le descripteur de déploiement et le plan de déploiement est établi manuellement par l'utilisateur.

2.3.11 ORYA

ORYA [Cunin 2005, Merle 2005] est un environnement ouvert basé sur l'utilisation d'outils de déploiement existant. Il permet d'automatiser et de coordonner certaines activités du cycle de vie du déploiement. Il permet d'offrir un procédé de déploiement adaptable capable de couvrir toutes les activités du cycle de vie du déploiement. L'adaptation est réalisée par l'utilisation des modèles paramétrables pour décrire les applications, les sites, les entreprises et les procédés. C'est un outil qui permet de prendre en compte la structure de l'entreprise et ses stratégies de déploiement.

L'approche adoptée par ORYA est une approche dirigée par les modèles, les concepts de déploiement sont donc décrits par un méta-modèle. Le méta-modèle d'ORYA contient les descriptions de l'environnement de l'entreprise, les machines cibles et les applications à déployer. Ce méta-modèle vise à représenter les concepts communs de déploiement et à s'abstraire de toute notion spécifique. Enfin, un plan de déploiement est créé pour réaliser toutes les étapes de déploiement de plusieurs applications sur plusieurs machines. Le plan de déploiement représente l'ensemble des actions à réaliser pour effectuer une activité de déploiement. L'unité de déploiement est déployée selon un plan de déploiement qui peut être spécialisé en fonction des différentes activités du cycle de vie.

Le modèle du plan de déploiement global est décrit pour refléter le résultat de l'application des contraintes et des stratégies de déploiement, pour ordonner les déploiements unitaires (associés à chaque unité de déploiement) et enfin pour indiquer les choix possibles d'unités à déployer. Un plan de déploiement global concerne le déploiement de N unités sur P machines. Les différents plans de déploiement unitaires, concernent un couple "unité de déploiement, machine cible", qui représentent les briques de base pour construire le plan de déploiement global dans ORYA.

Les travaux réalisés ont porté essentiellement sur les activités de sélection et l'installation, le reste des activités de déploiement fait partie des perspectives de ce travail.

Le tableau 2.10, présente une synthèse des points abordés dans le déploiement avec ORYA.

TABLE 2.10 – Synthèse du déploiement de ORYA

	ORYA
Unité de déploiement	L'unité de déploiement est le composant logiciel.
Activité de déploiement	La sélection et l'installation.
Résolution des dépendances	Résolution automatique des dépendances.
Sites cibles et plan de déploiement	Les sites cibles sont décrit dans le méta-modèle des machines cibles. Le plan de déploiement est généré automatiquement à partir des méta-données.

2.3.12 Kalimucho

Kalimucho est une plate-forme qui permet le déploiement dynamique et la reconfiguration des applications [Louberry 2011]. L'unité de déploiement dans Kalimucho est une application composée d'un ou plusieurs services, chaque service est réalisé par un ou plusieurs assemblages de composants. Les différents composants sont interconnectés par des connecteurs encapsulant les flux d'informations.

Kalimucho est une plate-forme capable de répondre aux changements de contexte en modifiant la structure de l'application à deux niveaux (composition des services et déploiement des services). Au niveau de la composition des services, elle est capable d'ajouter ou de supprimer des composants et de proposer des configurations de différentes qualités tout en réalisant le même service. Au niveau du déploiement des services, elle est capable de déplacer des composants et donc de modifier les connexions tout en gardant la même architecture. De plus Kalimucho est une plate-forme de supervision. Elle est constituée de cinq services collaborant. La plate-forme étant distribuée sur tous les périphériques supportant l'application afin d'avoir une vision globale de l'application.

La figure 2.9 présente le Schéma de la plate-forme Kalimucho.

- Le service `Superviseur` fourni les mécanismes de surveillance de l'application à déployer. Il permet la capture du contexte via des événements, ensuite il interprète ces événements et prend la décision d'adaptation adéquate à apporter à l'application.
- Le service `Générateur de Reconfiguration` permet d'évaluer le schéma de reconfiguration proposé lors de la décision de reconfiguration.
- Le service `Routage` permet de tenir à jour la topologie réseau de l'application. Ce service permet de garder une application efficace lors du déplacement, la

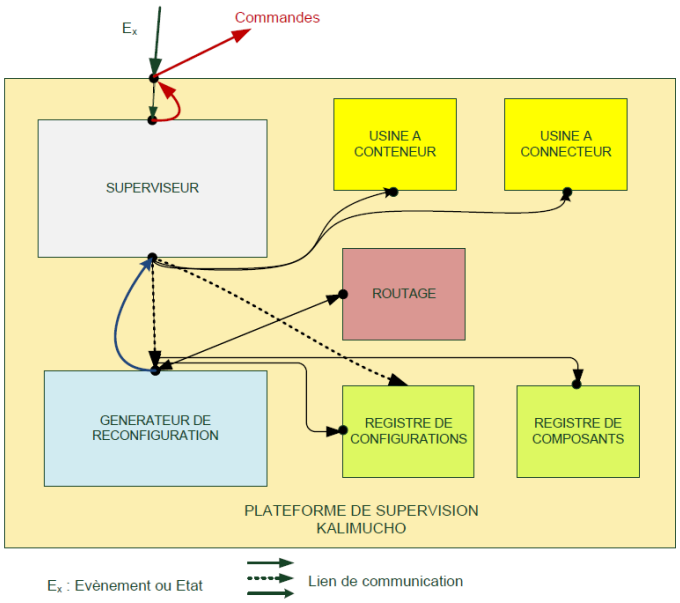

FIGURE 2.9 – La plate-forme Kalimucho

suppression et l'ajout de nouveaux composants dans le cas d'une reconfiguration de l'application.
- Le service `Usine à Conteneur` permet de créer des composants encapsulés dans un conteneur ad-hoc adapté au périphérique sur lequel le composant est installé. Les conteneurs ont une durée de vie limitée à l'utilisation du composant. Un conteneur est créé lorsque que le composant est installé et détruits lorsque que le composant est détruit.
- Le service `Usine à Connecteur` permet de créer des connecteurs encapsulé dans des conteneurs et qui on une durée limitée à l'utilisation du composant.

La plate-forme Kalimucho permet la réalisation d'un certain nombre d'activités de déploiement, à savoir l'installation de l'application (la création des conteneurs des composant et l'installation des composants dans les sites cible) ; la désinstallation qui consiste à retirer tous les services fournis par les composants de l'application ; la reconfiguration qui consiste en l'ajout, la suppression, ou la migration des composants de l'application et la création de schéma d'assemblage des composants. On peut dire aussi que l'activité de mise à jour est supportée.

Le tableau 2.11, présente une synthèse des points abordés dans cette sous-section.

TABLE 2.11 – Synthèse du déploiement de Kalimucho

	Kalimucho
Unité de déploiement	L'unité de déploiement est un composant qui offre des services.
Activité de déploiement	L'installation, la désinstallation, l'activation, la reconfiguration et la mise à jour.
Résolution des dépendances	Résolution manuelle des dépendances.
Sites cibles et plan de déploiement	Les sites cibles et le plan de déploiement sont générés à la main par l'utilisateur.

2.3.13 Jade

Jade [De Palma 2008] est un environnement pour l'administration autonome d'applications réparties. Il est composé de deux parties : un canevas (ME pour Manged Elements) qui encapsule les ressources administrées, qui leur donne une interface d'administration uniforme, et un canevas de construction de gestionnaires autonomes (AM pour Autonomic Managers), qui administrent à l'exécution un ensemble de ressources suivant une politique particulière. Un AM est souvent implémenté sous la forme d'une boucle de contrôle qui supervise et reconfigure suivant une certaine politique une infrastructure de ME.

Comme illustré dans la figure 2.10, Jade offre une bibliothèque de gestionnaires autonomes spécialisés dans divers domaines comme l'auto-réparation, l'auto-optimisation et l'auto-protection. Il s'appuie sur le modèle de composant Fractal pour la représentation du système administré et offrir une administration basé sur les composants. Autrement dit, les éléments administrés (les unités de déploiement) sont représentés sous la forme de composants Fractal toute en disposant d'une interface d'administration uniforme définie par les contrôleurs du composant. Par exemple, le contrôleur d'attribut permet de modifier le port utilisé par l'application et le contrôleur du cycle de vie permet d'activer et de désactiver un élément administré.

Jade dispose d'un gestionnaire de déploiement centralisé qui prend en charge l'exécution des plans de déploiement / re-déploiement commandées par la plateforme. Le logiciel administré (déployé) est décrit en utilisant le langage de description d'architecture Fractal ADL.

Pour déployer une application avec JADE, l'administrateur décrit dans un ou plusieurs fichiers Fractal ADL, l'architecture logicielle et les paramètres de confi-

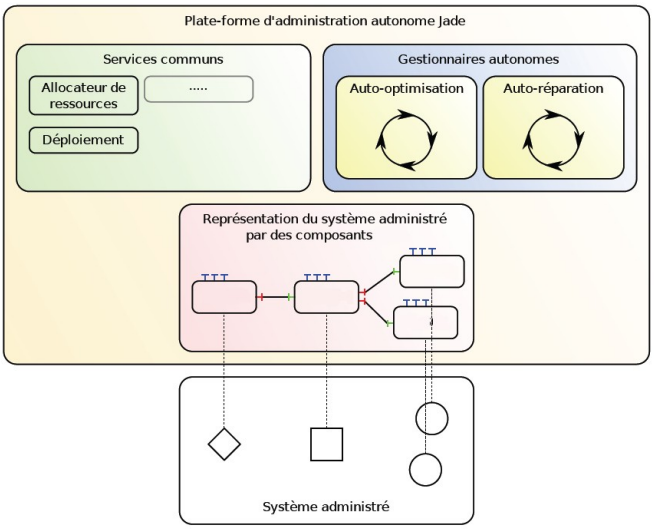

FIGURE 2.10 – Architecture de la plate-forme d'administration autonome Jade)

guration de chaque instance d'entité. Par la suite, le gestionnaire de déploiement transforme cette description en une architecture de composants Fractal (ME) distribuée sur les machines cibles en utilisant le mécanisme de déploiement intégré à Fractal. Les dépendances logicielles dans Jade sont représentées uniquement entre les éléments administrés.Un programme déclenche le processus de déploiement en appelant les interfaces appropriées sur les composants au niveau des machines cibles.

Les activités de déploiement supporté sont respectivement : l'installation, l'activation, la désactivation et la reconfiguration. Les auteurs ne donnent aucune précision sur l'activité de mise à jour et de désinstallation. L'activité d'installation consiste à l'installation des implémentations d'un module (implémentation des composants Fractal) tandis que l'activité d'activation / désactivation consiste au démarrage /l'arrêt de l'exécution du module installé. L'activité de reconfiguration s'appuie sur des capteurs permettant la détection des pannes de machines ou de composants Fractal ; et des actionneurs autonome permettant le redéploiement ou le remplacement d'un composant (ou un site cible) pour corrigé la situation.

Le tableau 2.12, présente une synthèse des points abordés dans cette sous-section.

TABLE 2.12 – Synthèse du déploiement dans Jade

	Jade
Unité de déploiement	L'unité de déploiement est une unité administrée encapsulée dans un composant Fractal.
Activité de déploiement	L'installation, l'activation, la reconfiguration et la désactivation.
Résolution des dépendances	Résolution automatique des dépendances.
Sites cibles et plan de déploiement	Les sites cibles et les plans de déploiement sont gérées à la main par l'administrateur.

2.3.14 Monitoring Autonomic Deployment and Management Engine MADME

A. Dearle et al propose dans [Dearle 2010] un middleware pour le déploiement et la gestion autonomique des applications constituées d'un ou plusieurs composants. Les unités de déploiement sont des composants logiciels appelés Cingal-bundle. Les composants peuvent être déployés sur des hotes cible équipés avec l'environnement de l'exécution de cingal. Le support de l'exécution cingal est écrit en java et dispose des mécanismes nécessaires pour assurer l'exécution et la sécurité des bundles.

Les contraintes de déploiement et le logiciel à déployer sont spécifiés avec le langage dédié à la description de système autonomique DELADAS (DEclarative LAnguage for Describing Autonomic Systems). L'administrateur de déploiement spécifie un objectif de déploiement initial, ensuite le système de déploiement tente de générer automatiquement une configuration qui décrit la manière de déploiement des composants de l'application. Autrement dit, l'ensemble des contraintes définit par l'administrateur, sont utilisées par un solveur de contraintes pour générer une ou plusieurs solutions. Chaque solution représente les interconnexions de composants et l'adresse d'une machine pour chacun d'entre eux.

Par la suite, le système de déploiement réalise un déploiement initial, en vérifiant périodiquement la satisfaction de l'objectif initial et re-déploie l'application si nécessaire. Le cycle de vie de déploiement est contrôler par un dispositif appeler MADME (Monitoring Autonomic Deployment and Management Engine). Il a le rôle d'exécuter et superviser les activités de déploiement. Il a aussi le rôle, de contrôler l'état de l'application déployé ainsi que les prises de décisions d'adaptations de l'application en cas de panne par exemple.

Les activités de déploiement supportées sont : l'installation, l'activation et la reconfiguration. Les auteurs, ne donne aucun précision sur les activités de mise à jour, de désinstallation et de désactivation. Les auteurs, ne précise pas aussi leurs manière de gestion des dépendances de l'application.

Cette approche comporte des motivations similaires aux notre. En effet, l'une des motivations est la réduction des interventions humaines dans le processus de déploiement par la génération automatique du plan de déploiement et la proposition d'un dispositif de contrôle autonomique. Toutefois, le middleware proposé n'est pas utilisable dans des environnements dotés d'une topologie imprévisible (aucun mécanisme de détection automatique des sites cibles). De plus, le contrôle de déploiement étant centralisé et assurer par le MADME, le processus de déploiement sera souvent redémarrer pour calculer une nouvelle configuration dans le cas des déconnexions et des pannes de lien du réseau par exemple. Notre solution, comporte des décisions d'adaptations hiérarchiques et décentralisées prises par les agents mobiles sur plusieurs niveaux (agents de supervision locale ou globale) et permet de faire des reconfigurations réalisables et très légères au niveau local et global dans des environnements répartis ouverts.

Le tableau 2.13, présente une synthèse des points abordés dans cette sous-section.

Table 2.13 – Synthèse du déploiement de MADME

	MADME
Unité de déploiement	L'unité de déploiement est bundle-Cingal.
Activité de déploiement	L'installation, l'activation et la reconfiguration.
Résolution des dépendances	Aucun précision.
Sites cibles et plan de déploiement	Les sites cibles sont gérés manuellement par l'utilisateur tandis que le plan de déploiement est généré automatiquement par un solveur de contraintes.

2.3.15 Synthèse des outils de déploiement

L'état de l'art abordé dans ce chapitre présente plusieurs plates-formes et outils de déploiement de logiciels sur des environnements différents. Il existe de nombreux outils dédiés au déploiement logiciel. Ces outils proposent des solutions variées à

plusieurs problèmes de déploiement de logiciels. Dans certains cas, ils sont très efficaces (sous certains conditions comme la stabilité de l'environnement et l'absence de pannes), mais ils ne couvrent pas en général toutes les activités du cycle de vie de déploiement et spécialement l'activité de reconfiguration dynamique au moment de l'exécution. Nous pouvons constater que généralement chaque plate-forme de déploiement dispose de sa propre définition du déploiement ainsi que ses propres mécanismes de déploiement. Elles prennent généralement en compte l'hétérogénéité des ressources, la distribution des ressources, les dépendances (logicielles et matérielles) de l'application à déployer. Nous allons dans ce qui suit développer la synthèse correspondante aux plates-formes de déploiement que nous avons présentées dans ce chapitre.

2.3.15.1 Activités de déploiement

La diffusion du logiciel est supportée par tous les outils de déploiement étudiées. Tous ces outils disposent d'un environnement de développement qui permet à la fin du processus de développement de produire une archive prête à déployer sur un ou plusieurs sites.

Presque toutes les plates-formes supportent les activités d'installation, d'activation, de désactivation et de désinstallation à l'exception de CCM qui supporte l'installation, l'activation et la désactivation, D&C qui supporte l'installation, l'activation la désactivation et la reconfiguration, RPM qui ne supporte pas l'activation et la désactivation et enfin ProActive qui ne supporte pas la désinstallation.

La mise à jour est supportée dans OSGi, RPM, .NET, Software Dock, et KALIMUCHO. On constate même l'utilisation de plusieurs stratégies de mise à jour dans le cas de la plate-forme .NET.

Software Dock, KALIMUCHO, MADME, Jade et D&C supportent l'activité de reconfiguration. Dans le cas de Software Dock la reconfiguration consiste à réinstaller le logiciel sous une nouvelle version, dans le cas de D&C la reconfiguration consiste en l'utilisation d'une nouvelle configuration de l'application déjà déployée. Dans MADME l'opération de reconfiguration consiste au re-déploiement de l'application dans le cas de défaillances, tandis que dans KALIMUCHO la reconfiguration consiste en l'ajout, la suppression, ou la migration des composants de l'application et la création d'un nouveau schéma d'assemblage des composants. L'activité de reconfiguration pour kalimucho, reste la mieux adaptée pour les environnements répartis ouverts.

En matière d'autonomie, la majorité des plates-formes de déploiement ne propose des mécanismes de gestion autonomique, seulement MADME propose un cycle autonomique pour la gestion de quelques activités de déploiement comme l'installation, l'activation et l'adaptation. De plus, elles ne disposent pas de mécanismes de

reconfiguration automatique adaptés aux environnements ouverts qui ne nécessite pas des interventions humaines.

Le tableau 2.14 présente la prise en compte des différentes activités de déploiement par les outils de déploiement étudiés.

Les lignes représentent les activités de déploiement et les colonnes représentent les plates-formes étudiées. Le signe + signifie que l'activité en question est supportée par la plate-forme correspondante.

TABLE 2.14 – Activités de déploiement et leurs prises en compte par les différents outils de déploiement

	CCM	EJB	OSGi	D&C	.NET	DeployWare	RPM	Software Dock	ProActive	ORYA	KALIMUCHO	Jade	MADME
Installation	+	+	+	+	+	+	+	+	+	+	+	+	+
Désinstallation		+	+		+	+	+	+			+		
Mise à jour			+		+		+	+			+		
Diffusion logiciel	+	+	+	+	+	+	+	+	+	+	+		
Activation	+	+	+	+	+	+		+	+		+	+	+
Désactivation	+	+	+	+	+	+		+	+		+	+	
Reconfiguration				+				+			+	+	+

2.3.15.2 Unité de déploiement et résolution de dépendance

La majorité des outils de déploiement propose leurs propres unités de déploiement. Généralement, les unités de déploiement peuvent encapsuler l'implémentation d'un composant et un ou plusieurs types de descripteurs (de déploiement, de sites,...). Les descripteurs sont, soit obligatoires comme dans CCM, D&C, ou optionnels dans le cas des EJB par exemple.

En matière de résolution des dépendances CCM, D&C, EJB, .NET, MADME, ProActive et DeployWare ne proposent aucun mécanisme de résolution automatique des dépendances. En effet, toutes les dépendances doivent être gérées à la main. Par exemple dans le cas de CCM et D&C les dépendances doivent être décrites dans les méta-données du paquetage de composant puis installées. Dans DeployWare les dépendances sont décrites dans le descripteur de déploiement par la suite, elles peuvent être installé en utilisant l'interface graphique de DeployWare. De leur coté, Software Dock, OSGi et RPM proposent un ou plusieurs mécanismes de résolution automatique de dépendances. OSGi propose des mécanismes de résolution automatique des dépendances entre les Bundles. De même, Software Dock propose une

résolution automatique des dépendances déclarer dans le fichier DSD d'un paquetage.

2.3.15.3 Limites

Les principales limites des plates-formes étudiées sont les suivantes :

Le caractère statique du déploiement

Dans les plates-formes de déploiement de logiciels étudiées, on a pu constater que c'est l'utilisateur des plates-formes qui doit choisir les différents sites dans lesquels l'application doit être déployée. Les plans de déploiement sont écrits à la main dans presque toutes les plates-formes. En effet, un opérateur humain prend en charge l'établissement du plan de déploiement en désignant pour chaque composant (logiciel) son site cible de déploiement avant le lancement du processus de déploiement. Dans le cadre de CCM, le processus de déploiement commence par l'interrogation de l'utilisateur pour connaitre les sites cible de déploiement. Dans le cadre d'EJB, l'utilisateur doit préciser les sites cibles de déploiement via le descripteur de déploiement ou à travers les annotations EJB. DeployWare utilise un descripteur de déploiement pour fixer au départ les sites cible de déploiement. Dans le cadre de D&C, le domaine où les applications seront déployées est décrit au début dans le modèle des sites cibles. Ce caractère statique de déploiement est vu comme une limite majeure du fait que si un ou plusieurs sites tombent en panne, il peut causer un crash du processus de déploiement, si on ne prévoit pas un mécanisme d'adaptation et de reprise adéquat. En plus, ce type de déploiement n'est pas adapté aux systèmes répartis de nature dynamique tels que les systèmes ouverts et les systèmes pervasifs. Dans ce type de système, on peut avoir une situation dans la quelle on ne connaît pas au préalable les sites cibles de déploiement. Par conséquent, ce type de déploiement n'est malheureusement pas applicable dans des environnements répartis ouverts caractérisés par : la mobilité des sites cibles, une fréquence importante de déconnexion et pour lesquels en ne connais pas forcément la liste des sites cibles au départ du processus de déploiement. Une détection automatique des sites cibles avec une génération automatique d'un plan de déploiement représente une réponse à cette limite.

L'intervention humaine dans les activités de déploiement

La deuxième limite constatée dans les plates-formes étudiées concerne les interventions humaine dans les activités de déploiement. Cette intervention est différente dans chaque outil, par exemple dans le cadre de DeployWare, l'utilisateur peut effectuer toutes les activités d'une façon manuelle ou d'une façon automatique en utilisant un script Unix standard dans lesquels il spécifie les différentes tâches à réaliser par le Framework. Dans le cadre d'OSGi, l'installation de l'application ainsi que la résolution des différentes dépendances sont réalisées d'une façon au-

tomatique, mais la désinstallation et la mise à jour de l'application se font d'une façon manuelle. Toutes les activités réalisées d'une façon manuelle nécessite une intervention humaine ce qui est considéré comme la deuxième limite des outils de déploiement étudiés. Nous rappelons que dans cette thèse, nous visons la limitation des interventions humaines dans le processus de déploiement en proposant un processus de déploiement autonomique.

L'absence d'outils de reconfiguration automatique

Dans les plates-formes de déploiement étudiées on a remarqué l'absence d'outils de reconfiguration automatique qui permettent le traitement des différentes situations de panne (de machines ou des liaisons réseau) ainsi que l'apparition de nouvelles machines (nœuds ou paires dans le cas des systèmes P2P) au moment de l'exécution sans l'arrêt du processus de déploiement et faire un nouveau déploiement (re-déploiement). De plus, la majorité des outils optent pour un modèle de contrôle de déploiement centralisé. On note l'utilisation d'outils de reconfiguration dans le cadre Software MADME, Dock, D&C, et KALIMUCHO. Seul ce dernier comporte un le service superviseur qui fournit des mécanismes de surveillance de l'application, il permet la capture du contexte via des événements, ensuite il interprète ces événements et prend la décision d'adaptation adéquate à apporter à l'application et un service de routage qui permet de tenir à jour la topologie du réseau de l'application. La présence d'outils de reconfiguration est primordiale pour permettre l'utilisation des différentes plates-formes de déploiement dans des systèmes dynamiques tels que les systèmes ubiquitaire et P2P.

Liaison à un modèle de composant

La dernière limite qu'on a constaté est la liaison à un modèle de composant. En effet, les plates-formes de déploiement comme OSGi, .NET, EJB utilisent leurs propres modèles de composant (Bundle, DCOM, EJB) et ne permettent pas le déploiement d'applications encapsulées dans d'autres modèles de composant. Autrement dit, ces outils ne permettent pas la description de plusieurs implémentations d'un même composant. Le support d'implémentations multiples n'est pas seulement utile dans le cadre de diversités de plate-forme, un composant peut par exemple avoir plusieurs implémentations, chacune utilisée dans un contexte particulier. Par exemple, un composant d'une interface graphique peut avoir une version d'implémentation allégée utilisable sur un Smartphone. Seulement les plates-formes ORYA, D&C et FDF, ces plates-formes permettent la description de plusieurs implémentations d'un même composant ainsi que son déploiement.

2.4 Agents mobiles

Avant de passer à la description de l'architecture de notre intergiciel de déploiement autonomique j-ASD qui se base sur les agents mobiles pour la réalisation et

la supervision du processus de déploiement et qui constitue le sujet du prochain chapitre, nous présentons un certain nombre de points servant à définir le paradigme d'agent mobile et motivant son utilisation. Nous donnons ici des éléments de définition des agents logiciels, des agents mobiles et des systèmes multi-agents suivant différents auteurs on discutons des leurs avantages et leurs limites.

2.4.1 Modèle d'acteur

Le modèle d'acteur est un modèle formel pour le calcul de processus concurrents et distribués proposé initialement par Hewitt [Hewitt 1977] et complété par Agha [Agha 1986]. Il sert de base pour l'implémentation de nombreuses plateformes d'agents mobiles. Les acteurs sont des entités anthropomorphes qui communiquent par messages. Au sein d'une communauté, un acteur est uniquement connu par sa référence. La communication est point à point, asynchrone, unilatérale et supposée sûre. Les messages reçus par un acteur sont stockés dans une boîte aux lettres et traités en série (en exclusion mutuelle). Les acteurs sont caractérisés par un comportement qui décrit la réaction de l'acteur à un message reçu. Lors du traitement d'un message, un acteur peut créer (dynamiquement) de nouveaux acteurs, envoyer des messages aux acteurs qu'il connaît et changer de comportement c'est-à-dire définir son comportement pour le traitement du prochain message. Le changement de comportement est un mécanisme puissant d'évolution et d'adaptation individuelle. D'une part, il permet à un acteur de modifier dynamiquement la nature des services qu'il fournit (évolution de la sémantique), d'autre part il permet de faire évoluer son interface. Par nature, le comportement est une entité de première classe qui matérialise l'état et qui est manipulable par le programme. Il est alors possible de créer à distance un autre acteur défini à partir du comportement. On obtient ainsi simplement une évolution de l'acteur d'origine à qui on peut faire suivre tous les messages qui étaient en attente. L'état est donc entièrement transporté (via le comportement) et restauré sans que le programmeur n'ait à développer de code spécifique pour cela. Ainsi, après migration, l'acteur peut poursuivre son activité à distance en bénéficiant des acquis résultant des traitements de messages précédents.

2.4.2 Agents logiciels

Un agent logiciel peut être défini comme un programme autonome caractérisé par des données et un comportement privé [Bradshaw 1997]. Les agents logiciels communiquent généralement par envoi de messages. Un système d'agents mobiles est un framework qui implémente le paradigme des agents mobiles [Filho 2000] ; il fournit des primitives et des services permettant l'implémentation, les communications et la migration des agents logiciels. Dans [Ferber 1999], les agents sont définis comme des entités, physiques ou virtuelles, pouvant agir volontairement sur leur environnement et communiquant entre elles.

Milojicic et al. [Dejan 1999] proposent la définition suivante : un agent est un élément autonome possédant une activité interne disposant de ses propres ressources, œuvrant généralement au nom d'un utilisateur ou d'une application, communiquant avec d'autres agents afin de réaliser la tache pour laquelle il a été créé [Dejan 1999]. Un agent agit pour le compte d'un tiers (un autre agent, un utilisateur) qu'il représente sans être obligatoirement connecté à lui. L'agent étend le concept d'objet, en y ajoutant des capacités d'autonomie (indépendance lors de l'exécution), de proactivité (capacité à prendre des décisions de manière autonome) et de communication. Un système multi-agent appartient initialement au domaine de l'intelligence artificielle où la programmation tente d'imiter l'esprit humain lors de l'exécution [Alliot 2002]. Le modèle de programmation par agent est adapté aux environnements décentralisés et évolutifs grâce à ses propriétés. A partir de ces définitions générales, on peut décliner les agents de deux manières distinctes : soit selon leur utilisation fonctionnelle (agents pour la recherche d'informations, la surveillance, le commerce électronique, des agents assistants, des agents conversationnels...) soit selon leurs fonctionnalités (capacité de collaboration, capacité d'apprentissage, mobilité, flexibilité. . .). Comme nous avons déjà précisé, les systèmes multi-agents (SMA) [Ferber 1995] forment une branche de l'Intelligence Artificielle dans laquelle des métaphores sociologiques ou biologiques sont employées pour la conception et la mise en œuvre de systèmes artificiels intelligents. Pour J. Ferber, un agent est une entité physique ou virtuelle :

- qui est capable d'agir dans un environnement
- qui peut communiquer directement avec d'autres agents
- qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser)
- qui possède des ressources propres
- qui est capable de percevoir (mais de manière limitée) son environnement
- qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune)
- qui possède des compétences et offres des services
- qui peut éventuellement se reproduire
- dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

Plusieurs modèles d'agent permettent d'implémenter différentes stratégies de fonctionnement. Par exemple dans le modèle BDI [Lesser 1995] (pour Beliefs Desires Intentions), un agent possède trois composantes principales : les croyances, les désirs et les intentions. Un mécanisme d'ordonnancement permet de choisir les tâches à accomplir pour réaliser ses intentions, en tenant compte des autres composantes. Dans le cadre de cette thèse, nous utilisons l'agent comme outil de génie logiciel et par conséquent, nous nous intéressons plutôt à leurs caractéristiques physiques (autonomie, mobilité, communication. . .).

2.4.3 Agents mobiles

Pour Harrison, un agent mobile est un logiciel capable de se déplacer au moment de l'exécution avec son code et ses données [Harrison 1995]. Les apports de la mobilité sont discutés dans [Fuggetta 1998, Bernard 2002] : un agent logiciel peut se déplacer d'un site à un autre en cours d'exécution pour se rapprocher de données ou de ressources. Il se déplace avec son code et ses données propres, mais aussi avec son état d'exécution. L'agent décide lui-même de manière autonome de ses mouvements. En pratique, la mobilité ne se substitue pas aux capacités de communication des agents mais les complète (la communication distante, moins coûteuse dans certains cas, reste possible). Afin de satisfaire aux contraintes des réseaux de grande taille ou sans fil (latence, non permanence des liens de communication), les agents communiquent par messages asynchrones. Il existe deux types de migration. Si la migration est initiée par le système, alors la migration est dite réactive [Quitadamo 2008]. Si la migration de l'agent est initiée par l'agent lui-même la migration est dite donc migration proactive [Quitadamo 2008]. L'agent mobile est généralement implémenté par un thread. Si l'ensemble du code avec le contexte d'exécution (pile, état, etc.) est déplacé au moment de la migration d'un agent alors la migration est qualifiée de forte. La migration est dite faible si l'agent déplace seulement le code ainsi que l'état. Le principe de délégation permet à l'agent d'opérer en étant déconnecté du client. La mobilité des agents et leur autonomie améliorent la sûreté (tolérance aux pannes par redéploiement des agents sur des nœuds en service) et permettent le suivi de matériel ou d'utilisateurs mobiles (informatique nomade, informatique ubiquitaire). Les agents mobiles sont donc un outil à part entière pour l'adaptation des systèmes. Dans [Chess 1994], les auteurs précisent le champ d'application des agents mobile : sources d'information multiples réparties, volumes importants, prise en compte des spécificités du client, interactions avec le client et la source. Ainsi, les avantages des agents mobiles sont nombreux :

- Les agents mobiles ont un avantage sur l'architecture client/serveur : leur capacité d'adaptation dans les environnements dynamiques hétérogènes. Cette faculté est essentiellement due à leur aptitude au déplacement en fonction de leurs besoins propres pour accomplir au mieux la tache qui leur incombe. De plus, les agents mobiles peuvent procéder à deux types de migration (proactive et réactive). La mobilité d'agent permet à un client d'interagir localement avec un serveur et donc de réduire le trafic sur le réseau qui ne transporte plus que les données utiles (éventuellement pré traitées). En outre, cela permet des transactions plus robustes que les transactions distantes.

- Dans le cadre de la migration proactive, c'est l'agent même qui initie son déplacement, ceci permettant de garder intégralement leur caractère autonome. Par contre dans le cadre de la migration réactive, c'est le système qui initie les déplacements sans avoir besoin d'une demande explicite de l'agent. Dans le cadre de cette thèse, nous utilisons les deux types de migration proactive et

réactive pour la réalisation de nos activités d'adaptation du processus de déploiement. La migration réactive intervient généralement lorsque l'on souhaite mettre en place des migrations totalement transparentes aux unités déplacées. Dans ce contexte, chaque unité retrouve, avant et après la migration, le même environnement et n'a absolument pas conscience de son déplacement. Par contre, l'adaptation d'un agent est nécessaire lorsque l'environnement change et qu'il ne peut plus trouver les ressources requises.

– L'exécution d'agents spécialisés offre davantage de souplesse que l'exécution d'une procédure standard sur les sites serveurs.

– L'asynchronisme, l'autonomie et la réactivité des agents leur permettent de réaliser une tâche tout en étant déconnectés du client, ce qui est particulièrement utile dans le cas de supports physiquement mobiles (clients ou serveurs d'information). Ce mode de communication permet de réduire le plus possible les communications distantes aux seuls transferts d'agents mobiles.Cela permet de diminuer considérablement les périodes de connexion entre deux sites. Cette diminution d'utilisation des communications réseaux permet de moins se soucier des ruptures de liens physiques qui peuvent intervenir fréquemment dans les environnements sans fil et ubiquitaires.

2.4.4 Agents mobiles adaptables

Un agent logiciel est dit adaptable [Leriche 2006, Leriche 2010] si certains de ses mécanismes internes, opérationnels (envoi de messages, déplacement) ou fonctionnels (comportement), sont modifiables en cours d'exécution. Conformément à la propriété d'autonomie, l'agent contrôle lui-même ses propres évolutions. Enfin un agent mobile peut se déplacer de manière proactive sur le réseau, en transportant ses données, son code et son état d'exécution.

L'architecture des agents mobiles adaptables se présente sous la forme de micro-composants en étoile autour d'un contrôleur (figure 2.11). Les différentes capacités des agents ont été réifiées sous la forme de micro-composants, par exemple :
– SendCt et ReceiveCt pour l'envoi et la réception de message,
– CreateCt pour la création locale et distante d'agents,
– BecomeCt pour le changement de comportement,
– MoveCt pour la mobilité.
– MailBoxCt pour la gestion de la boîte aux lettres,
– LifeCycleCt pour activer et gérer le cycle de vie de l'agent.

Le micro-composant de réception de message ReceiveCt, constitue le point d'entrée d'activation d'un agent. Il n'est pas modifiable car modifier ce micro-composant reviendrait à perdre la connaissance de ce point d'entrée, ce qui empêcherait toute communication entrante avec des agents qui possédaient la référence

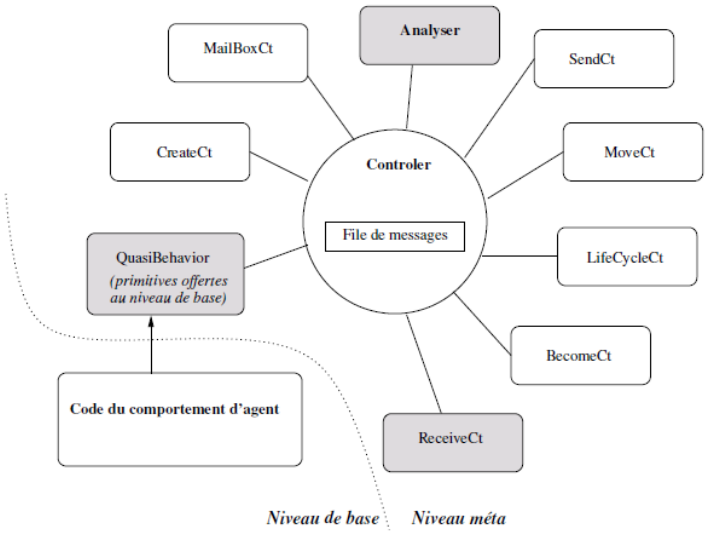

FIGURE 2.11 – Architecture d'agent mobile adaptable

du micro-composant remplacé. Enfin, le QuasiBehavior, n'est pas réellement un micro-composant, il sert d'interface entre le niveau de base et le méta-niveau, pour renforcer la sûreté et la sécurité de l'architecture. L'utilisation d'agents conduits à la décentralisation de la connaissance et du contrôle, les agents mobiles adaptables constituent ainsi un outil privilégié pour l'adaptation et le déploiement autonomique dans les environnements répartis (ubiquitaires, P2P et grille).

2.5 Conclusion

Ce chapitre a présenté les technologies et concepts qui nous ont semblé essentiels pour le déploiement autonomique et qui nous ont inspirées pour réaliser nos travaux. Nous avons d'abord présenté des définitions des principaux concepts et éléments clés appartenant au thème de cette thèse comme l'informatique autonomique, le déploiement de logiciels et les agents mobiles. Ensuite, nous avons présenté un ensemble de plates-formes de déploiement logiciel. A travers ce chapitre, nous avons expliqué les traits marquants de chaque plate-forme de déploiement, en montrant la complexité à laquelle nous devons faire face, pour concevoir un intergiciel pour le déploiement logiciel autonomique dans des environnements ouverts.

A l'heure actuelle il n'existe aucune plate-forme de déploiement de logiciels qui permette un déploiement autonomique avec un minimum d'intervention humaine possible dans le processus de déploiement et capable de résoudre automatiquement les problèmes liés à l'instabilité et à l'ouverture de l'environnement, mais plutôt des plates-formes permettant de simplifier et d'automatiser le déploiement sous cer-

taines conditions comme la stabilité de l'environnement de l'exécution et l'absence de pannes.

Le chapitre suivant décrit notre contribution. Il présente l'architecture de notre intergiciel de déploiement autonomique ainsi que l'algorithme de déploiement et de reconfiguration.

Chapitre 3

Contribution : j-ASD un intergiciel pour le déploiement autonomique de logiciels

Sommaire

3.1 Introduction

Dans ce chapitre, nous présentons notre contribution concernant le déploiement autonomique de logiciels. Nous proposons un processus et un langage dédié (DSL) au déploiement de logiciels permettant d'exprimer des contraintes de déploiement, puis une méthode de résolution des contraintes de déploiement prenant en compte un ensemble de sites cibles découverts à l'exécution du déploiement et générant un plan de déploiement initial, qui est ensuite réalisé par un intergiciel permettant le support du déploiement autonomique.

Pour mémoire, nous visons principalement des environnements répartis ouverts et dynamiques tels que les systèmes ubiquitaires ou P2P et des environnements répartis à grande échelle comme les grilles. Ces environnements sont caractérisés par une forte hétérogénéité matérielle et logicielle, la mobilité des sites, les variations de la QoS ainsi que les pannes et les déconnexions. Ces caractéristiques impliquent une nécessité d'adaptation dynamique du processus de déploiement lors de son exécution, d'où le terme de déploiement autonomique.

Dans le chapitre 1, nous avons présenté deux scénarios de déploiement dans lesquels les outils de déploiement de logiciels actuels échouent. Le premier décrit un environnement ouvert avec une topologie réseau imprévisible et des sites cibles qui ne sont pas connus au moment de la conception du déploiement et qui appartiennent à des utilisateurs différents. Le deuxième scénario présente le déploiement d'un logiciel de simulation sur une grille de calcul dont la taille est telle que la probabilité d'avoir un nœud en panne au moment du déploiement est élevée.

Ces deux scénarios mettent en évidence plusieurs spécificités, que nous devons intégrer dans notre contribution :

- Il est nécessaire de concevoir un mécanisme de découverte de réseau afin de pouvoir détecter les sites cibles de déploiement potentiel et permettre une adaptation dynamique du processus de déploiement en cas de détection de défaillances.
- Il faut obtenir les autorisations d'accès requises pour permettre le déploiement du logiciel en respectant les principes de sécurité des systèmes répartis (problème d'administration multiple).
- Il faut un intergiciel dédié au déploiement capable de reconfiguration dynamique et d'auto-adaptation pour pouvoir traiter les situations de panne de machine et des déconnexion.
- Il faut permettre à la personne qui déploie le logiciel d'exprimer des contraintes de déploiement et des modes d'adaptation dynamique en cas de panne.

Dans le chapitre 2, nous avons présenté plusieurs approches et plates-formes de déploiement de logiciel. Parmi les plates-formes présentées, il existe des outils qui permettent le déploiement d'applications monolithiques comme RPM [Bailey 2000]

et des plates-formes qui permettent le déploiement de logiciels à base de composants répartis tels que DeployWare [Flissi 2008] ou D&C [Specification 2006]. Mais même si ces plates-formes proposent généralement des solutions de déploiement qui permettent de couvrir tout ou partie des activités de déploiement et permettent de simplifier et d'automatiser le déploiement, les solutions proposées ne sont généralement exploitables que sous certaines conditions telles que la stabilité de l'architecture et l'absence de pannes et ne sont donc pas applicables dans les environnements que nous visons.

Nous proposons j-ASD, un intergiciel constitué d'une suite logicielle complète dédiée au déploiement autonomique et qui permet de répondre à cette problématique, c'est-à-dire déployer au mieux une application répartie quel que soit le contexte d'exécution. Ce qui revient à dire que l'intergiciel effectue de manière autonome (avec un minimum d'intervention humaine possible) un déploiement qui satisfasse un ensemble de contraintes spécifiées par l'administrateur de déploiement et capable de s'auto-adapter et de résoudre automatiquement quelques problèmes liés à l'instabilité et à l'ouverture de l'environnement.

Pour réaliser l'intergiciel de déploiement, nous avons choisi une approche originale, basée sur la technologie des agents mobiles adaptables (AMA) afin offrir des possibilités d'adaptation dynamiques du processus de déploiement. En effet, grâce à leur autonomie, leur proactivité et leurs facultés d'adaptation dynamique, les agents mobiles adaptables servent de support d'exécution et de supervision du processus de déploiement autonomique.

Dans ce chapitre je détaille mes contributions au déploiement logiciel autonomique, à savoir :

1. La définition d'un langage dédié (DSL) pour exprimer des contraintes de déploiement (contraintes matérielles et contraintes de d'installation) ainsi qu'un prototype permettant la construction d'un plugin pour Eclipse pilotant les phases suivantes du déploiement.
2. Un mécanisme de découverte automatique du réseau et des sites cibles de déploiement.
3. Une modélisation des contraintes de déploiement sous la forme d'un programme de satisfaction de contraintes (CSP).
4. Une modélisation d'un plan de déploiement sous la forme d'une matrice de localisation.
5. Le développement d'une interface graphique permettant à un outil de résolution de satisfaction de contraintes de fournir la matrice de localisation à partir des contraintes de déploiement et de la liste des sites cibles.
6. Une architecture et un prototype d'intergiciel capable d'interpréter, superviser et adapter le plan de déploiement fourni sous la forme d'une matrice de localisation, en utilisant un système d'agents mobiles adaptable qui, sur

la base de décisions locales, autonomes et décentralisées, permet l'adaptation dynamique du déploiement.

Le chapitre est organisé comme suit : dans la section 3.2 nous présentons le processus de déploiement autonomique. Nous présentons par la suite dans les sections 3.3 et 3.4 notre langage de description de contraintes de déploiement, la déclaration, la transformation et la résolution des contraintes de déploiement. Les sections 3.6 et 3.5 présentent respectivement le service de découverte de réseau et le mécanisme de bootstrap. Nous présentons l'intergiciel à base d'agents mobiles et l'algorithme de déploiement dans la section 3.8. Enfin, la section 3.9 conclut le chapitre.

3.2 Processus de déploiement autonomique

La vue générale du processus de déploiement autonomique de logiciels proposé est illustré sur la figure 3.1. Il répond aux sept exigences présentées dans le l'introduction 1.

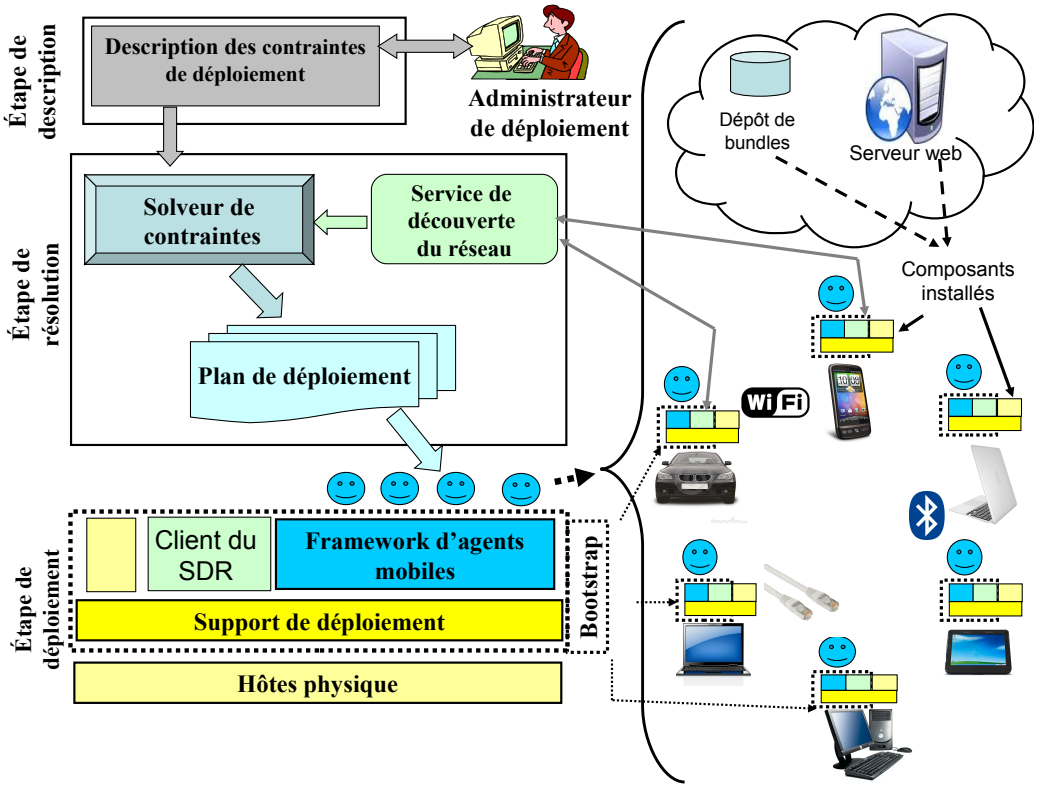

FIGURE 3.1 – Architecture de j-ASD

Le processus de déploiement est constitué de trois étapes principales qui sont respectivement : l'étape de description (spécification) de contraintes de déploiement,

l'étape de résolution de contraintes et l'étape de déploiement.

3.2.1 Description de contraintes de déploiement

Dans la première étape, l'utilisateur spécifie un ensemble de contraintes de déploiement en utilisant le langage dédié à la description de contraintes de déploiement. En effet, l'administrateur de déploiement déclare à travers ce langage des informations sur le logiciel à déployer tels que : les composants qui forment le logiciel, les services fournis et requis du logiciel, les dépendances logicielles, les préférences matérielles et les contraintes de déploiement.

Nous proposons pour cela un langage dédié (DSL) à la description des contraintes de déploiement qui permet d'exprimer les contraintes de déploiement et quelques informations sur le logiciel à déployer.

3.2.2 Résolution des contraintes de déploiement

La deuxième étape comporte plusieurs activités, dont le résultat est un plan de déploiement. Tout d'abord le système de déploiement lance le service de découverte du réseau pour la détection des sites cibles (dans le cas où l'administrateur ne les connaît pas). Ce service de découverte du réseau permet la détection automatique des hôtes cibles de déploiement dans le réseau local ou le réseau à large échelle, répondant au besoin de déploiement dans des environnements ouverts tels que les systèmes ubiquitaires. Le résultat de l'invocation de ce service est une liste de sites cibles candidats.

Pour obtenir un environnement d'exécution minimal sur les sites cibles de déploiement qui puisse jouer le rôle du client de service de découverte, nous proposons d'utiliser un logiciel d'initialisation (nommé *bootstrap* par la suite). Comme il est installé sur chaque site cible, nous l'utilisons également pour résoudre les problèmes d'administration multiples en obtenant les droits d'accès aux sites cibles.

Les contraintes de déploiement décrites au moyen du DSL sont compilées vers un niveau d'abstraction plus bas pour que, associées à la liste des sites cibles candidats, nous puissions obtenir un problème de satisfaction de contraintes résolvable par un outil tiers.

Le système procède à la fin de cette étape à la résolution du problème de satisfaction de contraintes pour calculer un plan de déploiement. Ce plan est généré dans un format interprétable par l'intergiciel de déploiement.

3.2.3 Déploiement

L'étude de l'état de l'art nous a permis de déterminer qu'il était envisageable de réutiliser un logiciel de déploiement répondant à une partie de nos besoins (comme OSGi par exemple), à savoir capable de s'exécuter dans des infrastructures matérielles hétérogènes et qui permet d'exécuter localement l'ensemble des activités de déploiement. Il doit permettre d'installer, désinstaller, activer, désactiver et mettre à jour le logiciel déployé.

Nous proposons un intergiciel à base d'agents mobiles adaptables, et des algorithmes de déploiement pour prendre en charge l'exécution du plan de déploiement, et le maintient du système en réagissant et en s'adaptant aux différentes situations de pannes de machines, des liens du réseau et des agents de déploiement par un ensemble de décisions prises au niveau local par les agents de supervision (par exemple la décision de migration d'un ou plusieurs composants vers un autre site cible de déploiement).

Des interfaces graphiques qui permettent aux administrateurs de déploiement le lancement / l'interruption du processus de déploiement. L'administrateur peut à tout moment vérifier l'état du processus de déploiement et intervenir dans une ou plusieurs activités.

Dans les sections suivantes, nous détaillons les différentes briques logicielles de cette architecture.

3.3 Langage de description des contraintes de déploiement

3.3.1 Besoin d'un langage dédié (DSL)

L'étude de l'état de l'art montre que configurer et déployer des logiciels répartis à grande échelle est une tâche complexe. La complexité est due principalement à la multitude de composants qui forment le logiciel et à l'hétérogénéité et au nombre important des sites cibles de déploiement. L'écriture d'un plan de déploiement pour un logiciel réparti à grande échelle est fastidieux et, comme discuté en introduction de cette thèse, non adapté aux environnements ouverts qui nous intéressent. Nous pensons qu'il est nécessaire de changer d'approche et de, plutôt, décrire des contraintes liant ces composants et ces sites candidats au déploiement. Il reste à trouver le bon formalisme.

Comme montré dans le chapitre 2, les plates-formes de déploiement existantes comportent plusieurs formalismes pour exprimer les contraintes de déploiement, les dépendances logicielles et les préférences matérielles des logiciels à déployer. Ces

formalismes incluent généralement les langages de description d'architecture (ADL), les descripteurs de déploiement (descripteur de déploiement XML) comme dans le cas de D& C, software Dock et CCM et les langages dédiés (DSL).

Dans le cadre de nos travaux de thèse, nous avons choisi d'utiliser un langage dédié (DSL) à la description des contraintes de déploiement. Les DSL présentent de nombreux avantages : ils utilisent les idiomes et le niveau d'abstraction du domaine ciblé, et donc sont utilisables par des spécialistes ; ils sont légers, donc facilement maintenables, portables et réutilisables ; ils sont le plus souvent très documentés, cohérents et fiables ; ils sont optimisés pour le domaine [Tolvanen 2010, Strembeck 2009].

Cette approche est similaire à celle développée dans MADME [Dearle 2004] [Dearle 2010], discuté dans l'état de l'art (chapitre 2). Ainsi, dans le cycle de vie du logiciel, nous identifions un rôle particulier, celui de l'administrateur de déploiement. L'acteur qui a ce rôle peut exprimer dans un langage expert les différentes contraintes liées au déploiement.

Une première itération pour définir le langage nous a amené à construire un arbre de syntaxe du DSL (figure 3.2). Cet arbre reprend les concepts classiques des outils de déploiement, à savoir une description d'un logiciel par son ensemble de composants, dont chaque composant possède un nom, une version, un ensemble de fichiers disponibles sur une URL donnée, des dépendances. Nous avons ajouté des concepts de préférences logicielles pour décrire des dépendances physiques (architecture, taille mémoire, puissance électrique...).

Mais nous voulons pouvoir exprimer des contraintes de déploiement au delà de ces concepts classiques. Par exemple, nous voulons pouvoir exprimer qu'un composant logiciel doit être déployé sur tous les hôtes disponibles dans le réseau au moment du déploiement, ou encore qu'un autre composant doit être présent sur exactement N sites cibles satisfaisant les préférences logicielles exprimées par l'administrateur du déploiement, sans préjuger de leur emplacement. Ces nouveaux concepts ont été décrits dans notre DSL, nommé j-ASD DSL.

3.3.2 j-ASD DSL

"j-ASD DSL" est un langage déclaratif doté d'une grammaire simplifiée et intuitive. Le code de la figure 3.3 présente la grammaire de j-ASD DSL telle que décrite au moyen d'Xtext [1], un framework de développement de langages dédiés. Xtext nous permet d'obtenir directement un greffon j-ASD dans l'environnement Eclipse permettant à l'administrateur de déploiement de décrire le logiciel à déployer ainsi que

1. http://http://www.eclipse.org/Xtext

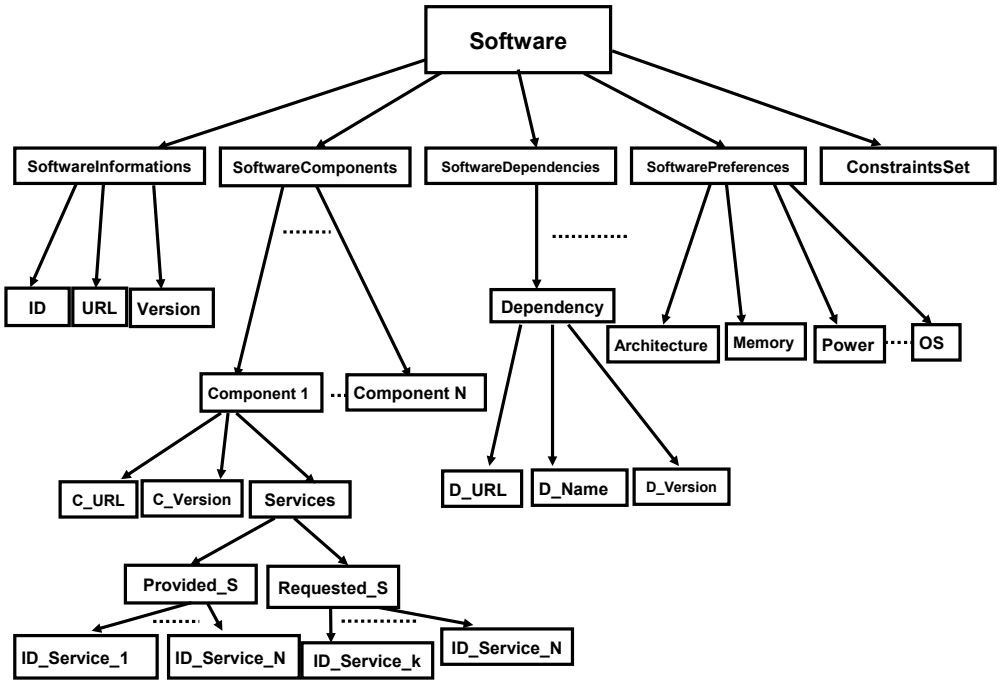

FIGURE 3.2 – Arbre de syntaxe du DSL j-ASD

les contraintes de déploiement dans un outil intégrant des vérifications de syntaxe, des propositions de correction, une complétion automatique. Le greffon pourrait être complété par d'autres vérifications (sémantiques, cohérence, validité...) mais nous n'avons pas approfondi cet aspect au cours de cette thèse.

Comme présenté dans les Figures 3.2 et 3.3, un logiciel est défini par son identifiant, sa version, son URL et les composants qui le forment. Le logiciel est défini aussi par ses dépendances logicielles, ses contraintes matérielles et ses contraintes de déploiement.

Un logiciel est composé à partir d'un ou plusieurs composants. Chaque composant est défini par son identifiant, sa version, son URL (la localisation de son implémentation). Dans notre prototype, nous avons préféré nous concentrer sur l'expression des contraintes de déploiement plutôt que sur la résolution des dépendances, ce qui explique la disparition de cet aspect dans cette proposition.

```
grammar eu.itsudparis.inf.JASDDsl with
org.eclipse.xtext.common.Terminals
generate jASDDsl
"http://www.itsudparis.eu/inf/JASDDsl"

Model:
Software=Software
Components+=Component+
(HostConstraints+=HostConstraint*)?
Deployment=Deployment;

Software:
"Software" "{"
"Name" "=" name=ID
"Version" "=" ver=INT
"Components" "=" components+=(ID)*
"}";

Component:
"Component" "{"
"Name" "=" name=ID
"Version" "=" ver=INT
"Url" "=" url=STRING
("Dependencies" "=" dependencies+=ID*)?
"}";

HostConstraint:
"HostConstraint" "{"
"Name" "=" name=ID
constraints+=(OsPref | CPUPref |
    RAMPref | HDPref | NetSpeedPref)*
"}";

Deployment:
"Deployment" "{"
{Deployment} members+=MemberDecl*
"}";

MemberDecl:
component=ID "@" localisation=Localisation
("with" constraints+=(ID)*)?;

OsPref:
"OSNameContains" name=STRING;

CPUPref:
"CPULoad" InfSup val=INT "%";

terminal InfSup:
"<" | ">" | ">=" | "<=";

RAMPref:
"RAM" sym=InfSup val=INT "MB";

HDPref:
"HD" sym=InfSup val=INT "MB";

NetSpeedPref:
"NetSpeed" sym=InfSup val=INT "kb/s";

Localisation:
IPv4 | NetName | Val | Interval | All;

terminal IPv4:
INT '.' INT '.' INT '.' INT;

NetName:
STRING;

Val:
INT;

terminal Interval:
INT ".." INT;

terminal All:
"all";
```

FIGURE 3.3 – Grammaire du langage j-ASD DSL en Xtext

Les contraintes matérielles et logicielles suivantes peuvent être exprimées :

- contraintes sur le système d'exploitation `OsPref` ;
- contraintes sur la charge du processeur `CPUPref` ;
- contraintes sur la mémoire disponible `RAMPref` ;
- contraintes sur l'affichage `HDPref` ;
- contraintes sur la vitesse du réseau `NetSpeedPref`.

Dans une vision plus large que celle de notre prototype, cette liste de contraintes n'est pas exhaustive et peut-être étendue par la suite. On pourrait ajouter d'autres types de contraintes comme par exemple l'utilisation de la batterie `PowerPref`,

les contraintes sur la latence du réseau `NetLatency` ou bien des contraintes sur la puissance des processeurs `MIPSPref` (MIPS : Millions d'Instructions Par Seconde).

Également, les contraintes de déploiement étant liées aux activités de déploiement, il faudrait prévoir l'expression de contraintes d'installation, d'activation, de désactivation et de mise à jour. Les contraintes d'activation et de désactivation sont des contraintes de type relation de précédences. Elles permettent d'exprimer des contraintes de type "activer le composant Ci avant Ck", "activer les composants Ci,..., Ck avant le composant C" et ne pas désactiver le composant Ck une fois activé.

Dans le cadre de cette thèse nous nous focalisons sur les contraintes d'installation et nous n'avons pas prévu les mécanismes d'expressions spécifiques aux autres activités. Le travail sur les contraintes d'activation, désactivation et mise à jour fait partie des perspectives de cette thèse.

3.3.3 Sémantique et exemples de contraintes d'installation

```
1 Software {
2   Name=Example_1
3   Version=1
4   Components=C C1 C2 C3 C4
5 }
6 Component {
7   Name = C
8  Version = 1
9  Url="http://x.fr/C.jar"
10 }
11 Component {
12   Name = C1
13   Version = 1
14   Url="http://x.fr/C1.jar"
15 }
16 Component {
17   Name = C2
18   Version = 1
19   Url="http://x.fr/C2.jar"
20 }
21 Component {
22   Name = C3
23   Version = 1
24   Url="http://x.fr/C3.jar"
25 }
26 Component {
27   Name = C4
28   Version = 1
29   Url="http://x.fr/C4.jar"
30 }
31 HostConstraint {
32   Name= Constraint1
33   CPULoad < 80%
34   RAM >= 40 Mb
35   OSNameContains "Linux"
36 }
37 HostConstraint {
38   Name= Constraint2
39   RAM >= 400 Mb
40   OSNameContains "Windows"
41   NetSpeed >= 4000 kb/s
42 }
43 Deployment {
44    C @ all
45    C1 @ 10
46    C2 @ all with Constraint1
47    C3 @ 5 with Constraint2
48    C4 @ 2 alone
49 }
```

FIGURE 3.4 – Exemple de programme j-ASD DSL

Les contraintes d'installation que nous avons identifiées pour permettre le déploiement de logiciels dans des environnements ouverts sont les suivantes :

- Installer un composant dans tous les sites cibles disponibles au moment du déploiement, éventuellement avec des contraintes ;
- Installer un composant dans un sous-ensemble de sites cibles disponibles, éventuellement avec des contraintes ;
- Installer un composant au même endroit qu'un autre composant, éventuellement avec des contraintes.

Le programme présenté dans la figure 3.4 illustre la description du déploiement d'un logiciel nommé Example_1, comportant cinq composants (`C, C1, C2, C3` et `C4`). Chaque composant est défini par son identifiant, la version, une URL d'où il peut être récupéré. Cette URL peut indifféremment décrire l'adresse d'un serveur web, ftp ou être un répertoire disponible localement dans le cas d'un système de fichiers partagé.

Les deux blocs HostConstraint représentent deux spécifications de contraintes matérielles et logicielles. Les contraintes déclarées dans (`Constraint1`) signifient que dans le / les sites cibles :

- La charge du processeur (`CPULoad`) doit être inférieure à 80%.
- Le système à besoin d'un minimum de 40 Mb de mémoire (RamSize >= 40 Mb).
- Le système d'exploitation doit être un système Linux.

Les contraintes(`Constraint2`) signifient que dans le / les sites cibles :

- Le composant à besoin d'un minimum de 400 Mb de mémoire.
- Le système d'exploitation doit être un système Windows.
- La vitesse de la connexion réseau `NetSpeed` doit être supérieure ou égale à 4000 kb/s.

Finalement, les contraintes de déploiement (contraintes d'installation dans cet exemple) sont une spécification de contraintes de haut niveau, qui expriment que :

- Le composant C doit être déployé dans tous les sites disponibles au moment du déploiement.
- Le composant C1 doit être déployé dans un sous-ensemble de dix sites cibles de déploiement.
- La troisième contrainte exprime que le composant C2 doit être déployé dans tous les sites qui satisfont les contraintes présentes dans `Constraint1`.
- La quatrième contrainte veut dire que le composant C3 doit être déployé dans un sous-ensemble de cinq sites cibles qui satisfont l'ensemble des contraintes `Constraint2`.
- La dernière contrainte exprime que le composant C4 doit être déployé seul dans deux sites cibles de déploiement (à l'exclusion d'autres composants).

3.4 Transformation et résolution des contraintes de déploiement

Le service de découverte du réseau permet de détecter les sites cibles de déploiement. Ce service, décrit à la section 3.6, retourne une liste de sites cibles disponibles au moment de la recherche, exploitée conjointement avec la description des contraintes de déploiement.

Le processus de transformation des contraintes de déploiement (voir la Figure 3.5.), commence par la vérification syntaxique et lexicale du programme j-ASD DSL. Un programme syntaxiquement et lexicalement correct sera compilé afin de générer automatiquement un autre type de contraintes d'un niveau d'abstraction plus bas sous la forme d'un problème de satisfaction de contraintes (CSP) résolvable par un solveur de contraintes.

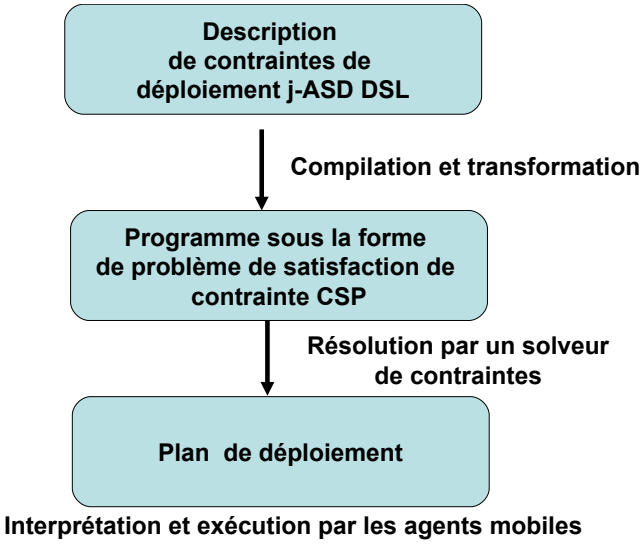

FIGURE 3.5 – Processus de transformation des contraintes

3.4.1 Traduction en CSP

Un CSP (Constraints Satisfaction Problem) se décrit par un ensemble de variables (inconnues) dont le domaine est souvent discret (entier, ensemble, booléen) et un ensemble de contraintes sur ces variables. La résolution consiste alors à trouver un tuple de valeurs tel que toutes les contraintes soient satisfaites.

Dans un CSP, une variable Xi est associée à un domaine D (Xi) qui définit l'ensemble des valeurs qu'il est possible d'affecter à cette variable. Les domaines

peuvent être, par exemple, un intervalle de valeurs ou un ensemble discontinu de valeurs.

Les contraintes expriment les relations sur les variables du problème. Une contrainte qui porte sur une variable est dite contrainte unaire, si une contrainte porte sur n variables elle est dite contrainte n-aire.

Une solution est un tuple ou une affectation d'une liste de valeurs choisies pour chacune des variables du problème (ou un sous-ensemble dans le cas d'une affectation partielle lors d'une recherche). Un tuple viole une contrainte si ses valeurs n'appartiennent pas à la relation. Il est dit consistant s'il ne viole aucune contrainte.

La transformation des contraintes déclarées dans le programme j-ASD DSL est nécessaire à cause de l'écart important entre le niveau d'abstraction dans les programmes de satisfaction de contraintes et les abstractions utilisées par l'administrateur de déploiement pour exprimer les contraintes de déploiement sous la forme d'un programme j-ASD DSL.

Dans le cadre de j-ASD, le problème de satisfaction de contraintes est construit à partir d'un ensemble de variables entières (matrices de localisation et collocation) et un ensemble de contraintes sur ces variables. Nous modélisons le programme CSP avec les éléments suivants :

- Un ensemble fini C de composants logiciels qui forment l'application à déployer. Par exemple, dans l'exemple présenté précédemment dans la Figure 3.4 C= {C, C1, C2, C3, C4}.
- Un ensemble de sites cibles de déploiement H détectés par le service de découverte du réseau.
- Une matrice de localisation (loc), qui modélise la localisation d'un composant sur un site tel que :

  loc (Ci, Hj) = 1, si le composant Ci peut être installé dans le site Hj.
  loc (Ci, Hj) = 0, si le composant Ci ne peut pas être installé dans le site Hj.

- Un ensemble P de contraintes sur les sites cible, par exemple CPULoad et la taille de la mémoire disponible.
- Un ensemble de contraintes sur les variables de localisation (loc (Ci, Hj)).
- Un problème de placement initial de composants sur l'ensemble des hôtes H en respectant les contraintes de déploiement.

En plus des matrices de localisation (loc (Ci, Cj), nous avons aussi défini une matrice de collocation (colloc (ci,cj)) telle que :

$\forall Ci$ et $Cj \in C$:

1. colloc (Ci, Cj) = 2, si le composant Ci doit être installé seul dans un ou plusieurs sites cibles de déploiement.

2. colloc (Ci, Cj) = 1, si le composant Ci doit être installé avec le composant Cj dans le même site cible de déploiement.
3. colloc (Ci, Cj) = 0, s'il n'y a aucune restriction.

Les contraintes de déploiement déclarées dans le programme j-ASD DSL seront traduites par le compilateur en contraintes sur les variables de localisation (loc (Ci, Hj)) et collocation (colloc (Ci, Cj)).

3.4.2 Exemple

Le programme j-ASD DSL suivant (figure 3.6), présente le déploiement d'un logiciel composé de quatre composants (C1, C2, C3, C4). Il comporte deux ensembles de contraintes matérielles Constr1 et Constr2. Les contraintes de déploiement de cet exemple sont des contraintes d'installation.

La première contrainte de déploiement (partie Deployment) signifie que le composant c1 doit être déployé dans tous les sites cibles de déploiement, soit :

$c1 \in C, \forall Hj \in H$, loc (c1, Hj) = 1.

La deuxième contrainte signifie que le composant c2 doit être déployé dans un sous-ensemble de trois sites, soit :

$c2 \in C$, $\exists Hi, Hj, Hn \in H$: loc (c2, Hi) = loc (c2, Hj) = loc (c2, Hn) = 1

La troisième contrainte signifie que le composant c3 doit être déployé dans tous les sites qui comportent au minimum 150 Mb de mémoire disponible qui sont sous Windows, soit :

$c3 \in C, \forall Hj \in H$: **si** $((RAM \geqslant 150Mb)$ et $(OSName = "Windows"))$ **alors** loc (c3,Hj) = 1 **sinon** loc (c3, Hj)= 0

La quatrième contrainte signifie que le composant c4 doit être installé dans un sous-ensemble de quatre sites qui respectent l'ensemble de contraintes définies dans Constr2. Autrement dit le composant C4 doit être installé dans quatre sites cibles qui disposent d'un système d'exploitation de type Linux, équipé d'un minimum de 100 Mb de mémoire disponible et leur charge de processeur doit être inférieure à soixante pour cent au moment du déploiement. Cette contrainte est traduite de la manière suivante :

$c4 \in C$, $\exists h1, h2h3, h4 \in H$ tel que :
si $((RAM \geqslant 100Mb)$ et $(OSName = "Linux"))$ et $(CPULoad < 60\%))$
alors loc (c4,h1) = loc (c4,h2) = loc (c4, h3) = loc (c4, h4) = 1
sinon loc (c4,h1) = loc (c4,h2) = loc (c4, h3) = loc (c4, h4) = 0

```
Software {
  Name=Exemple_2
  Version = 1
  Components = C1 C2 C3 C4
}

Component {
  Name = C1
  Version = 1
  Url="http://x.fr/C1.jar"
}

Component {
  Name = C2
  Version = 1
  Url="http://x.fr/C2.jar"
}

Component {
  Name = C3
  Version = 1.1
  Url="http://x.fr/C3.jar"
}

Component {
  Name = C4
  Version = 1
  Url="http://x.fr/C4.jar"
}

HostConstraint {
  Name=Constr1
  RAM >= 150 MB
  OSNameContains "Windows"
}

HostConstraint {
  Name=Constr2
  CPULoad < 60%
  RAM >= 100 MB
  OSNameContains "Linux"
}

Deployment {
  C1 @ all
  C2 @ 3
  C3 @ all with Constr1
  C4 @ 4 with Constr2
}
```

FIGURE 3.6 – Programme écrit avec le DSL j-ASD

En utilisant ces traductions formelles, le compilateur génère automatiquement un problème de satisfaction de contraintes sous la forme d'un programme. Ensuite, ce programme est dynamiquement chargé avec la liste des sites cibles au solveur de contraintes. Le solveur est alors invoqué pour résoudre le CSP et retourne la première solution consistante trouvée. La solution trouvée est sous la forme d'un ensemble de valeurs des variables de location et de collocation. Les valeurs retournées seront par la suite directement traduites comme un plan de déploiement initial par le système d'agents mobiles.

Par exemple, au moment de la résolution des contraintes de l'exemple précédent (Exemple_2) on dispose de cinq sites cibles de déploiement H1, H2, H3, H4, H5 :

Si loc (C3, H4) = 1 alors le site H4 satisfait les contraintes présentes dans Constr1 et on doit déployer le composant C3 dans ce site.
Si loc (C3, H1) = 0 alors le site H1 ne satisfait pas les contraintes matérielles présentes dans Constr1 et le composant C3 ne sera pas déployé dans ce site.

Le plan de déploiement (via la matrice loc) décrit l'endroit où les différents composants logiciels seront installés. Si le solveur de contraintes n'arrive pas à trouver une solution consistante, il peut être relancé après l'obtention d'une nouvelle liste de sites cible pour essayer de trouver une autre solution. Si le solveur n'arrive

toujours pas à trouver une solution, une erreur sera notifiée à l'administrateur de déploiement.

Il est possible que le solveur de contraintes arrive à trouver plusieurs solutions consistantes mais différentes (plusieurs plans de déploiement). Néanmoins, lorsque certains paramètres de qualité de service sont considérés, certaines de ces solutions sont meilleures que d'autres. Notre objectif n'est pas de trouver la meilleure solution possible, mais de trouver une solution consistante qui satisfasse les contraintes de déploiement. Le travail sur la sélection de la meilleure solution fait partie des perspectives de la thèse.

3.5 Le bootstrap

Nous avons discuté dans l'introduction de cette thèse du problème d'administration multiple. En effet, nous ne voulons pas contourner les principes de sécurité des systèmes distribués, et nous souhaitons explicitement obtenir l'accord des administrateurs de chaque site cible pour l'exécution de l'intergiciel de déploiement.

Pour cela, nous imaginons de prévoir un programme dédié (nommé bootstrap), qui sera installé volontairement par l'administrateur du site et mis à sa disposition par différents moyens. Par exemple, via Bluetooth, ou en envoyant son URL par e-mail ou SMS, ou même en l'intégrant dans un code QR®. Ce programme très léger peut être implémenté sous la forme d'un script qui demande aux administrateurs des sites cibles de déploiement (détectés préalablement par le service de découverte du réseau) les droits d'accès (permissions) à l'hôte et met en place l'environnement d'exécution et les dépendances logicielles pour notre intergiciel, de manière similaire à la gestion des permissions dans les environnements Android [2].

3.6 Système de découverte des sites cibles

Dans les environnements ouverts et instables, l'administrateur de déploiement ne connaît pas forcément l'ensemble des sites cibles au moment de lancer le déploiement. Pourtant, cette information est nécessaire à la résolution des contraintes de déploiements pour obtenir le plan de déploiement.

De plus, au moment de l'exécution du plan de déploiement, un ou plusieurs sites peuvent tomber en panne ou se déconnecter. Dans ce cas, le système de déploiement doit être en mesure de détecter ce type de situation pour maintenir en vie le logiciel déployé en exécutant des opérations de re-configuration.

2. http://developer.android.com/guide/topics/security/permissions.html

Pour répondre à ces besoins, nous proposons un service de découverte qui permet de réaliser trois modes de découverte des sites cibles :

1. Une découverte du réseau local (domestique) dans le cas où un utilisateur souhaite déployer son logiciel sur l'ensemble (ou un sous-ensemble) des sites disponibles connectés au réseau local.
2. Une découverte étendue (à grande échelle) dans le cas où un utilisateur souhaite déployer une application à grande échelle (dans le cadre d'un système ubiquitaire par exemple) sans forcément connaitre au préalable tous les sites cibles de déploiement.
3. Une découverte multi-échelle qui permet la réalisation d'une découverte mixte du réseau (local et à grande échelle).

Pour construire notre système de découverte du réseau nous avons réalisé une étude qui porte sur plusieurs technologies et protocoles open source qui peuvent être utilisés comme base pour notre service de découverte du réseau. Cette étude est résumée en annexe A.

Dans le cadre des infrastructures réseaux fixe (réseaux locaux et grilles de calcul), des protocoles tels que bonjour [APPLE 2009] et UPnP (Universal Plug-and-Play) [UPnP Forum 2008] sont considérés comme des solutions efficaces. Cependant, ces protocoles ne permettent pas le passage à l'échelle et ne sont pas utilisables dans des infrastructures réseaux large échelles ouvertes comme les systèmes ubiquitaires. Pour cela nous avons étudié d'autres protocoles créés initialement pour des domaines spécifiques tels que la découverte de service et gestion des sessions multimédia, qui permettent le passage à l'échelle. Les protocoles étudiés sont respectivement : le protocole SLP [Guttman 1999], le protocole SIP [Sparks 2007] et le protocole XMPP [Saint-Andre 2009], cf. annexes.

Pour la réalisation de notre système de découverte nous avons choisi les protocoles UPnP et XMPP. Les protocoles UPnP sont utilisés pour la réalisation d'un module de découverte du réseau local, tandis que le protocole XMPP est utilisé pour la réalisation d'un module de découverte étendue (découverte à grande échelle). Un module qui utilise XMPP et UPnP ensemble est utilisé pour la réalisation d'une découverte mixte (multi-échelle).

3.6.1 Découverte d'un réseau local

Dans le premier cas, l'administrateur de déploiement souhaite déployer une application dans un réseau local. L'administrateur ne connaît pas forcément les sites cibles de déploiement, mais il souhaite déployer son application sur tout l'ensemble des équipements connectés, disponibles dans le réseau local.

L'utilisation d'UPnP permet en effet, la création d'un réseau de sites cibles (devices UPnP) détectables et contrôlables par des points de contrôle UPnP. UPnP

offre des mécanismes d'envoi de messages permettant aux équipements de détecter la présence, l'insertion, la suppression et la disparition d'un équipement du réseau. En plus, la technologie UPnP est compatible avec OSGi et s'intègre facilement dans notre environnement de déploiement. UPnP ne nécessite pas une connexion internet pour fonctionner, il suffit d'avoir simplement des connexions-réseaux entre les sites cibles, les mécanismes d'UPnP fonctionnant même dans le cas de l'inexistence d'une connexion internet.

L'idée est que le système de déploiement installe un **Device** et un **point de contrôle UPnP** sur chaque site cible de déploiement et un Device et un point de contrôle global dans le site initiateur de déploiement dans le cas d'une découverte du réseau local. La création des devices et des points de contrôle UPnP est prise en charge par le logiciel de bootstrap (voir la section 3.5). Le point de contrôle global, permet la détection d'une manière totalement transparente des connexions/déconnexions de Device UPnP client dans le réseau. Les Devices créés offrent un ensemble de services permettant la récupération de certains informations sur les sites cibles comme la taille de la RAM, le type de système d'exploitation utilisé et le type de l'équipement (Pc, PDA, Smartphone,). Les informations fournies à notre système de déploiement permettant par la suite le calcul d'un ou plusieurs plans de déploiement et de détecter d'une manière autonomique les situation de pannes et de déconnexions.

3.6.2 Découverte à grande échelle

Dans le deuxième cas d'illustrations, l'administrateur de déploiement souhaite déployer une application répartie à grande échelle. Dans ce contexte, l'utilisateur qui ne connait pas l'ensemble des sites cibles de déploiement, au départ du processus de déploiement, ne pourra pas déployer son application avec les outils de déploiement existants.

Fournir un service de découverte du réseau à grande échelle, est une tache complexe, qui peut prendre un temps d'exécution (de détection) très important. La complexité est présente parce que, dans ce cadre, on dispose d'un nombre très grand des sites potentiels qui peuvent être des sites de déploiement ou pas. En plus, on doit gérer les problèmes engendrés par la multitude des supports réseaux ; en effet, on peut rencontrer plusieurs types de connexions réseaux (réseaux internet, réseau 3G), plusieurs types de protocoles réseaux (http, TCP/IP, UDP) et protocoles de communication.

Le protocole XMPP permet la détection de présences de clients Jabber connectés à un serveur Jabber et dispose de services permettant la gestion de sessions multimédia (authentification, communications). Dans le cas de la découverte à grande échelle, le système de déploiement installe et lance l'exécution d'un client XMPP sur chaque site cible de déploiement et un client spécialisé (client central) dans le

site initiateur de déploiement. Chaque site dans lequel le client XMPP est installé et activé sera considéré comme site cible de déploiement détectable d'une manière automatique. Cette approche permet de réduire considérablement le temps de découverte du réseau, mais change partiellement la notion de site cible de déploiement. Nous ne voulons pas contourner les principes de sécurité des systèmes distribués, par conséquent nous comptons sur les administrateurs des sites pour obtenir les droits d'installation et d'exécution de notre environnement de déploiement dans leurs hôtes. Le bootstrap installe et exécute l'environnement d'exécution de j-ASD, ce qui permet de lancer le client XMPP d'une manière automatique.

Tous les clients XMPP seront connectés d'une manière automatique à un serveur XMPP fixé au début du processus de déploiement (voir la figure 3.7). Une fois qu'un client est connecté au serveur XMPP (12jabber.fr par exemple), il est ajouté à la liste de contacts du client central, ensuite il envoie d'une manière périodique le couple d'informations (nom de machine, adresse IP) au client central. Le client central reste à l'écoute des autres clients et joue le rôle d'un collecteur d'informations provenant des autres clients en exploitant les fonctionnalités de gestion de présence offertes par le protocole XMPP.

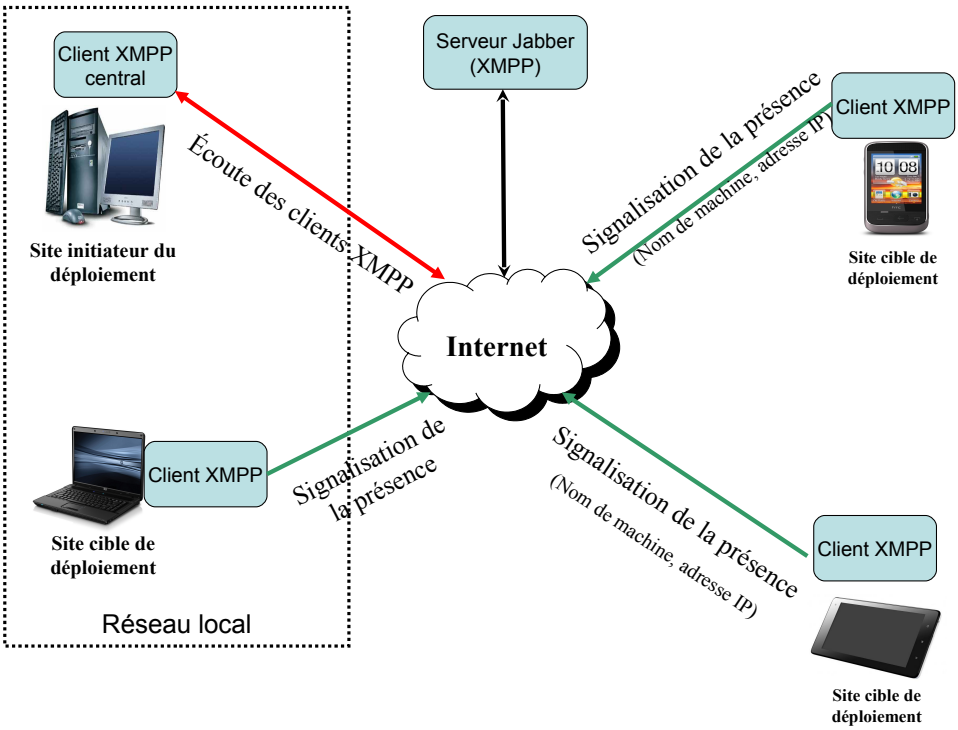

FIGURE 3.7 – Système de découverte à grande échelle

L'utilisation de XMPP permet à notre système de déploiement, par la suite, la détection des défaillances (déconnexion des hôtes cibles) et pour la réalisation des

opérations de reconfiguration.

3.6.3 Découverte mixte

Les deux modules de découverte du réseau présenté précédemment sont utilisés ensemble dans le cas d'une découverte mixte (découverte locale et découverte étendue) pour la réalisation d'un module de découverte mixte. Les deux modules de découverte sont activés ensemble. Dans chaque machine cible, un client XMPP, un device et un point de contrôle UPnP sont installés. UPnP sera utilisé dans le cadre de découverte du réseau local, et les clients XMPP seront utilisés pour la découverte étendue. Le résultat de la découverte est toujours une liste de sites cibles de déploiement sous la forme d'un couple nom de machine, adresse de la machine cible et quelques informations sur les ressources disponibles dans les sites cible.

Grâce à ce service et à la vision globale sur l'état des sites cibles, notre intergiciel est en mesure de réaliser un déploiement de logiciels même dans les environnements les plus instables comme les systèmes ubiquitaires. L'utilisation de ce service permet de détecter toutes les situations de nouvelles connexions, de pannes des machines et de déconnexions. La vision globale sur l'état des sites cibles de déploiement qu'offre ce service ainsi que la détection des défaillances et des déconnexions représente un atout qui permettra à notre système de déploiement de réaliser des opérations de reconfigurations dynamiques au moment de l'exécution.

3.7 Support de déploiement

Le support de déploiement fournit l'environnement d'exécution pour notre système de déploiement. Il doit permettre aussi l'installation, la désinstallation, l'activation, la désactivation et la mise à jour des composants déployés au moment de l'exécution sans redémarrer l'ensemble du système. Le système de déploiement doit également permettre le déploiement de logiciels sur des hôtes hétérogènes tels que les ordinateurs portables, les tablettes tactiles, les Smartphones, les voitures et les PC-Ultra mobile.

Plusieurs plates-formes de déploiement existantes comme OSGi et D&C fournissent tout ou partie des fonctionnalités souhaitées. Pour la réalisation de notre prototype, nous avons choisi la plate-forme OSGi comme support de déploiement. Les motivations de notre choix de plate-forme et d'unité de déploiement seront discutées dans la section suivante.

3.7.1 OSGi comme support de déploiement local

L'Alliance OSGi [3] a été fondée créé mars 1999. Elle propose des standards pour les services internet à destination des maisons, voitures, Smartphone, ordinateurs et d'autres environnements. Elle propose un ensemble de spécifications ouvertes pour la gestion et la livraison de services sur des réseaux locaux.

Comme présenter dans le chapitre 2, OSGi dispose d'un environnement d'exécution basé sur la technologie Java et qui supporte le déploiement local et à distance de bundles. Un bundle peut être installé, résolu, actif, arrêté, en démarrage et désinstallé. Chaque bundle dispose d'un mécanisme permettant l'importation et l'exportation de code sous forme de paquetage, ce mécanisme permet l'accès aux services du bundle sans avoir le besoin d'intégrer le code de ces services dans chaque bundle. Cela correspond directement à notre besoin de déploiement local de composants applicatifs. La figure 3.8 illustre le cycle de vie d'un Bundle OSGi.

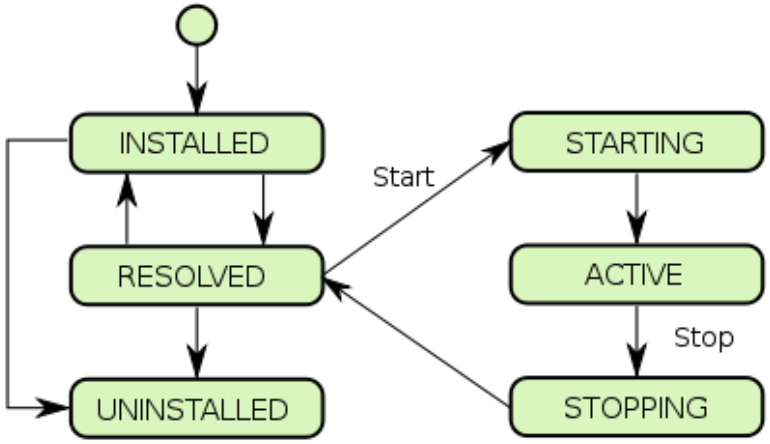

FIGURE 3.8 – Cycle de vie d'un Bundle OSGi

L'environnement d'exécution d'OSGi fournit des mécanismes permettant de réaliser les activités de déploiement principales comme l'installation, l'activation, la désactivation, la mise à jour et la désinstallation. Les différentes activités peuvent être réalisées manuellement ou via une API dédiée. Néanmoins, comme nous avons vue dans le chapitre 2 l'environnement d'exécution ne supporte pas l'activité de

3. http://www.osgi.org

reconfiguration qui nécessite son implémentation.

3.8 Intergiciel de déploiement autonomique

Dans cette section, nous décrivons le fonctionnement de notre intergiciel de déploiement autonomique, à base d'agents mobiles.

L'utilisation des agents mobiles pour l'automatisation du processus de déploiement de logiciels n'est pas une nouvelle approche. En effet, il existe plusieurs travaux dans la littérature qui ont utilisé cette technique pour réaliser quelques activités de déploiement de logiciels dans des infrastructures réseaux stable (voir par exemple [Hall 1999]). Néanmoins, les solutions proposées ne sont pas adaptées au déploiement de logiciels dans des environnements ouverts tels que les systèmes ubiquitaires.

Les agents mobiles adaptables que nous utilisons (cf. Section 2.4), sont basés sur le modèle formel d'acteur [Agha 1986]. Programmer nos agents mobiles adaptables revient donc à programmer les comportements des acteurs ; l'enchaînement des comportements étant décrit dans les comportements eux-mêmes. Le choix d'utiliser des agents mobiles qui se basent sur le modèle d'acteur est justifié par :

- d'une part, la communication par messages asynchrones est bien adaptée aux réseaux ouverts (réseaux sans fils, systèmes ubiquitaires et aux réseaux de grande taille), où les communications synchrones sont trop difficiles et coûteuses à établir,
- d'autre part, l'autonomie d'exécution et l'autonomie comportementale favorisent l'intégration de mécanismes de mobilité et garantissent un certain niveau d'adaptabilité du processus du déploiement.

3.8.1 Système d'agents pour le déploiement autonomique

Nous avons créé un système d'agents mobiles adaptables pour exécuter et superviser le processus de déploiement. Nous avons créé un système composé de deux types d'agents (figure 3.9) qui sont respectivement : les agents de supervision du déploiement (globale et locale) et les agents de déploiement qui sont des agents dédiés à la réalisation des activités de déploiement dans les sites cibles.

Les agents de supervision de déploiement ont pour rôle l'exécution, la supervision, le contrôle et la reconfiguration du processus de déploiement. Les opérations de reconfiguration et d'adaptation sont exécutées pour réagir aux changements rencontrés dans l'environnement d'exécution dans lequel le logiciel est déployé (pannes de machine ou de liens et les déconnexions, panne d'un agent,...). Pour que notre système de déploiement puisse assurer un déploiement logiciel à grande échelle, deux

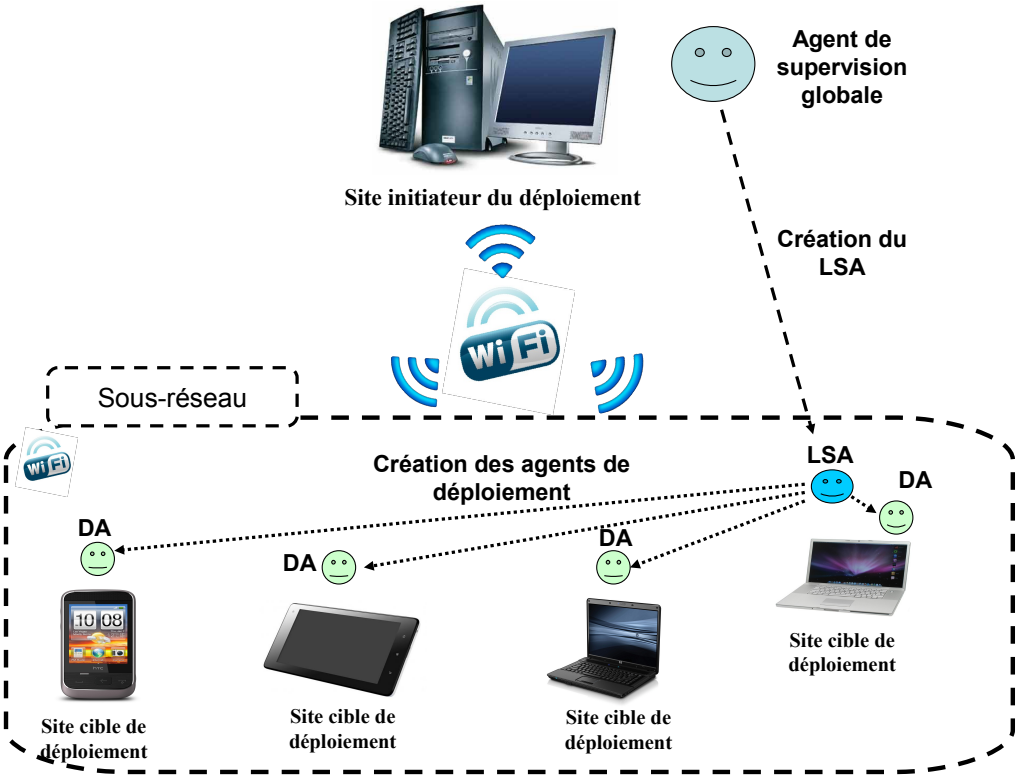

FIGURE 3.9 – Système d'agents mobiles pour le déploiement

types d'agents de supervision sont utilisés et qui sont respectivement : les agents de supervision locale (LSA) et un agent de supervision globale (GSA).

L'agent de supervision globale (GSA) est créé au début du processus de déploiement par l'intergiciel j-ASD. Il a pour rôle l'exécution et la supervision du plan de déploiement initial calculé par le solveur de contraintes. Il n'y a qu'un seul agent de supervision globale dans notre système, créé et exécuté initialement dans le site initiateur du processus de déploiement. Le GSA permet de contrôler le processus de déploiement par la création et la coordination avec les agents de supervision locale. L'agent de supervision globale, dispose aussi d'une capacité de prise de décisions, d'adaptation du processus du déploiement d'une manière autonome sans revenir à l'administrateur du déploiement. Par exemple, le GSA peut remplacer un agent de supervision locale qui ne répond pas. Il peut aussi demander la migration d'un agent de supervision locale ou un agent de déploiement vers un autre site de déploiement pour réagir aux variations de qualité de service, aux pannes de machines et aux déconnexions.

Un agent de supervision locale est déployé par l'agent de supervision globale sur un site cible d'un sous-réseau local (un sous réseau de classe C ou B pour des

raisons de passage à l'échelle). L'agent de supervision locale a pour rôle la création des agents de déploiement dans chaque site cible du sous-réseau pour installer, désinstaller, activer, désactiver et mettre à jour les composants logiciels qui forment l'application à déployer, selon le plan de déploiement transmis par l'agent de supervision globale. L'agent de supervision locale a aussi pour rôle la supervision du processus de déploiement dans le sous-réseau correspondant et la réalisation d'un certain nombre d'opérations de reconfigurations dynamiques d'une manière autonomique ou via la demande de l'agent de supervision globale.

Afin d'exécuter le plan de déploiement calculé, l'agent de supervision globale échange plusieurs messages (principalement asynchrones) avec les agents de supervision locale (LSA) pour connaître, superviser et contrôler l'état des activités de déploiement dans chaque sous-réseau.

L'agent de supervision globale fournit aussi des interfaces d'échange à l'administrateur de déploiement qui lui permettent de vérifier et d'intervenir à tout moment dans une des activités du processus de déploiement.

Enfin, l'agent de déploiement est chargé d'exécuter les activités de déploiement dans chaque site cible. Il est créé et exécuté dans un site cible selon le plan de déploiement calculé. L'agent de déploiement réalise plusieurs opérations telles que, le téléchargement des composants (à partir d'un serveur web par exemple), la résolution des dépendances logicielles (l'installation des dépendances de l'application dans l'environnement d'exécution), l'installation des composants dans le site cible et la notification de la fin de l'activité d'installation à l'agent de supervision par un message. De la même manière, l'agent de déploiement peut aussi activer, désactiver et désinstaller des composants de l'application à déployer à la demande de l'agent de supervision et notifie à la fin de chaque activité réalisée le succès/l'échec de l'activité.

Dans la sous-section suivante, nous présentons d'une manière détaillée l'algorithme de déploiement et quelques algorithmes d'adaptation du déploiement initial.

3.8.2 Algorithmes de déploiement

L'algorithme de déploiement commence son exécution dans le site initiateur de déploiement. Il commence dès qu'un plan de déploiement valide (qui satisfait les contraintes de déploiement) est calculé. Il prend en entrée le plan de déploiement initial. Nous rappelons, que pour calculer le plan de déploiement initial, le service de découverte du réseau et le bootstrap sont utilisés pour découvrir les sites cibles et préparer l'environnement d'exécution de j-ASD (support de déploiement et framework d'agent mobile).

L'algorithme comporte trois étapes qui sont respectivement : la phase de vérification, la phase d'exécution du plan de déploiement et la phase de contrôle et

de reconfiguration du déploiement. Le diagramme suivant (figure 3.10) illustre le déroulement des trois phases de l'algorithme de déploiement.

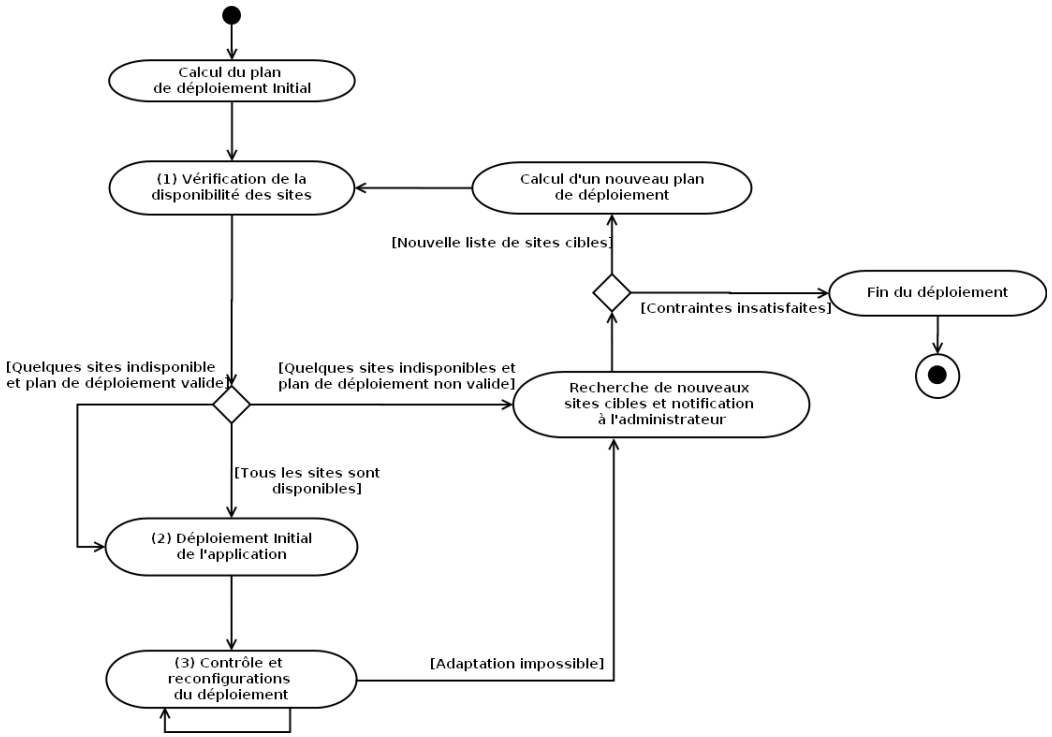

FIGURE 3.10 – Diagramme d'activités de l'algorithme de déploiement

3.8.2.1 Phase de vérification

L'algorithme commence par la phase de vérification de la disponibilité des sites cibles (voir l'algorithme 1) découverts par le service de découverte du réseau et sélectionnés dans le plan de déploiement. Pour cela, j-ASD fait appel une autre fois au service de découverte du réseau et vérifie bien que chaque site sélectionné -pour être un site cible de déploiement- est toujours disponible. Cette étape offre une possibilité d'adaptation du processus du déploiement avant même le lancement de l'étape du déploiement. En effet, elle permet de détecter l'indisponibilité d'un ou plusieurs sites cibles de déploiement et probablement l'insatisfaction d'une ou plusieurs contraintes de déploiement avant le lancement de la phase de déploiement. Autrement dit, cette phase permet de déterminer si tous les sites cible du plan de déploiement sont toujours disponibles et que toutes les contraintes de déploiement sont satisfaites. L'algorithme passe à la phase d'exécution du plan de déploiement si tous les sites sont disponibles. Sinon, deux cas de figures sont possibles :

1. Dans le premier cas, un ou plusieurs sites cibles sont indisponibles et une ou

Algorithme 1 *Phase de verification*

Entrée : *deploymentPlan, hostList* */* Plan de déploiement et liste des sites cibles*/*
Sortie : *startDeployment* */* Une variable booléenne, si vrai commencer le déploiement*/*

```
1: boolean hostDispo;   /* Variable de disponibilité de sites cibles, hostDispo est vrai si le site cible est disponible*/
2: boolean [ ] checkConstraint;   /* Tableau de validité des contraintes, checkConstraint[i] est vrai les contraintes sont satisfaites*/
3: startDeployment = Vrai;
4: i = 0;
5: pour tout host dans deploymentPlan faire
6:    hostDispo = checkHostDispo(host); /* CheckHostDispo(host) est une fonction qui vérifie la disponibilité d'un site cible*/
7:    si (hostDispo == Faux) alors
8:                                                    /* Si le site cible est indisponible*/
9:       checkConstraint[i] = checkDeploymentConstraint();   /* Vérification des contraintes de déploiement*/   /* checkDeploymentConstraint() est une fonction qui vérifie la validité des contraintes de déploiement*/
10:   sinon
11:      checkConstraint[i] = Vrai;   /* Si le site est disponible en suppose que les contraintes de déploiement sont satisfaites*/
12:   fin si
13:   si (checkConstraint[i] == Faux) alors
14:      /* Si les contraintes de déploiement sont insatisfaites, le plan de déploiement n'est plus valide*/
15:      notifyStatut() ;   /* notifyStatut() est une fonction qui notifie à l'administrateur de déploiement la non validité du plan de déploiement en cours*/
16:      startDeployment = Faux;   /* La valeur de retour sera Faux*/
17:      retour startDeployment
18:   fin si
19:   i++ ;
20: fin pour
21: retour startDeployment
```

plusieurs contraintes de déploiement sont insatisfaites. Dans ce cas, le plan de déploiement calculé n'est plus valide ; par conséquent, j-ASD informe l'administrateur du déploiement de l'invalidité du plan et procède par la suite au calcul d'un nouveau plan de déploiement qui satisfait les contraintes de déploiement avec une nouvelle liste de sites cibles.

2. Dans le deuxième cas, on dispose d'une indisponibilité d'un ou plusieurs sites cibles mais les contraintes de déploiement sont toujours satisfaites. Par exemple, si l'administrateur du déploiement utilise des contraintes de type Installer les composants dans tout ce qui est disponible (`C @ all`). Dans ce

cas, le plan de déploiement reste toujours valide ; par conséquent j-ASD informe l'administrateur du déploiement et procède au lancement de la phase de déploiement avec moins de sites cibles que prévus mais toujours avec un plan qui satisfait les contraintes.

Il existe un cas dans lequel tous les sites cibles sont disponibles mais les contraintes de déploiement ne sont pas satisfaites (on dispose de moins de mémoire que prévu par exemple). Dans ce cas, la détection de l'insatisfaction des contraintes sera dans la phase de déploiement au niveau des sites cibles. Par conséquent, la correction de cette situation sera à la charge des agents de supervision par la migration vers un autre site dans phase de déploiement ou la phase d'adaptation du déploiement.

3.8.2.2 Phase d'exécution du plan de déploiement

Une fois la vérification effectuée, le processus de déploiement initial de l'application, commence par la création de l'agent de supervision globale. L'agent de supervision globale (GSA) créé les agents de supervision locale (LSA) dans chaque sous-réseau détecté (un LSA pour chaque sous-réseau). Pour cela, le GSA utilise la primitive `CreateCt` pour créé les agents de supervision locale à distance. Le GSA utilise plusieurs types de messages pour communiquer avec les LSA. L'activité de déploiement traitée détermine le type de message utilisé. Les messages utilisés sont respectivement : les messages d'installation, les messages d'activation, les messages de désactivation, les messages de mise à jour et les messages de désinstallation. Le LSA réagit à chaque message reçu et adapte un comportement bien défini, qui se termine par l'envoi d'un message de succès ou d'échec de l'activité de déploiement en question.

Le GSA peut utiliser deux modes d'envoi de messages, un mode d'envoi asynchrone et un mode synchrone. Dans le mode asynchrone (qui est le mode le plus adapté aux systèmes ouverts), l'agent envoie le message et continue son exécution sans attendre la réponse du destinataire du message envoyé. Dans le mode synchrone, l'agent envoie un message et reste bloqué jusqu'à la réception de la réponse de l'agent destinataire du message.

Dans les deux modes de communication, les messages envoyés peuvent être perdus. Par conséquent, si GSA détecte qu'un LSA ne répond pas il renvoie une seul fois le message en question. Si le LSA en question ne répond toujours pas, le GSA exécute une procédure de reconfiguration pour remplacer le LSA en question (voir l'algorithme 2.

Après la création des LSA, le GSA reste à l'écoute des messages du succès de la création de chaque LSA. Si l'agent de supervision globale, ne reçoit pas de message de succès de la création d'un LSA, le GSA crée un nouveau LSA dans un autre site (choisi à partir de la liste des sites cibles disponible) pour remplacer le LSA qui ne

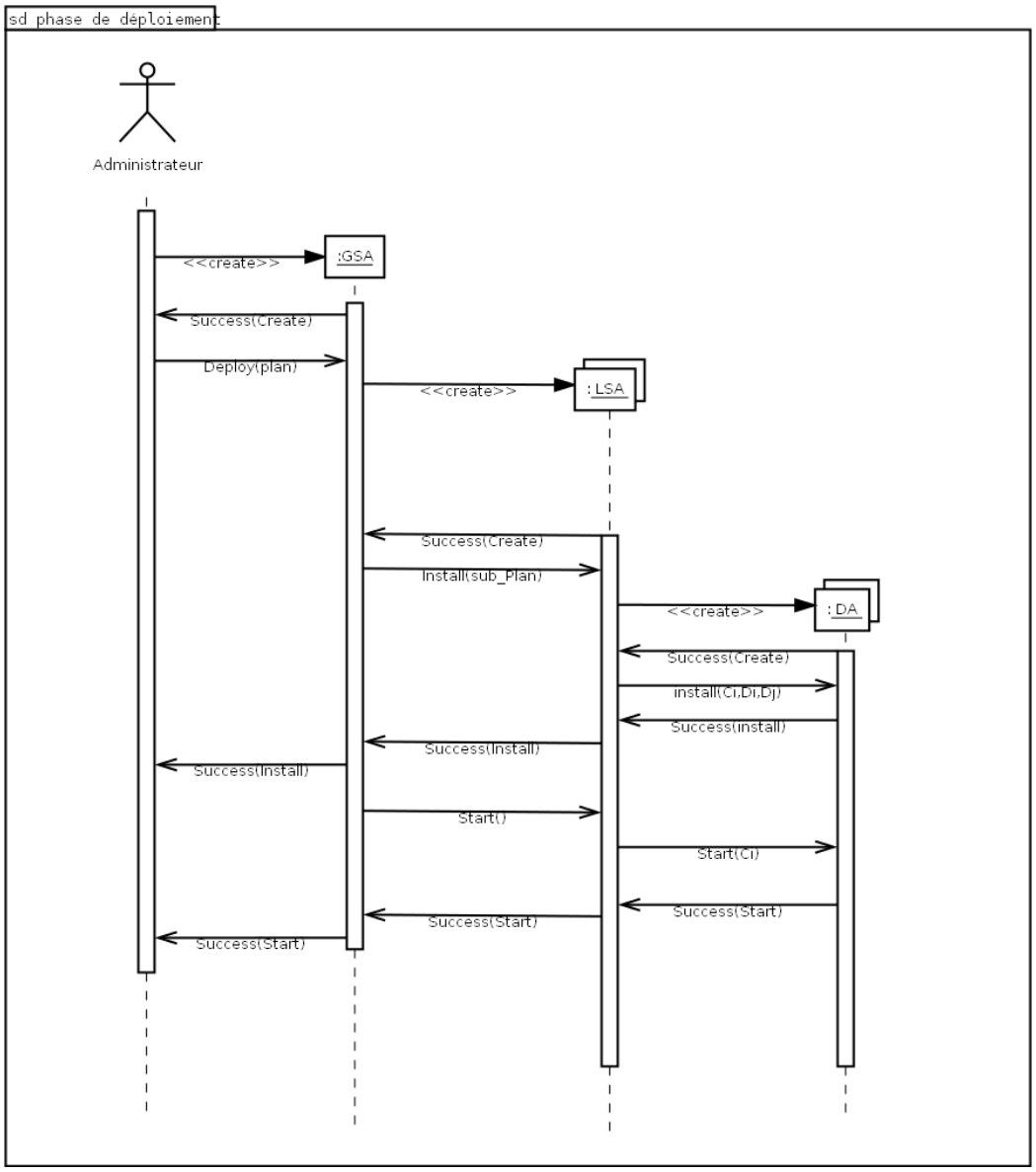

FIGURE 3.11 – Diagramme de séquence de la phase de déploiement (asynchrone)

répond pas et ignore tous les messages qu'il peut recevoir de ce dernier par la suite. Autrement dit, le succès de l'opération de création des agents de supervision locale est conditionné par la réception de tous les messages de succès de la création des LSA.

La figure 3.11 présente un diagramme de séquence pour une phase de déploiement asynchrone dérouler sans aucun besoin de reconfiguration. Comme illustré dans ce diagramme (3.11), une fois que tous les agents de supervision locale sont créés avec succès, le GSA peut alors envoyer la partie correspondante du plan de déploiement à chaque LSA. Le GSA utilise le mode de communication asynchrone, par conséquent il envoie toutes les parties du plan de déploiement en séries et se met dans une situation d'écoute permanente des messages qui proviennent des LSA.

Les agents de supervision locale, procèdent par la suite à la création des agents de déploiement (un agent par site cible) selon le plan de déploiement reçu. De la même manière que le GSA, les LSA restent à l'écoute des agents de déploiement et attendent les messages de succès de la création des agents de déploiement. Une fois que tous les messages de succès de la création des agents de déploiement ont été reçus, les messages d'installation de l'application serons envoyés. Pour cela, chaque LSA envoie un message d'installation à chaque agent de déploiement concerné selon le plan de déploiement reçu. Le message d'installation, comporte les informations URL du/des composants à installer dans le site cible et le/les URL des dépendances logicielles du composant.

L'agent de déploiement installe la /les dépendances logicielles des composants, ensuite installe le /les composants de l'application. Une fois l'installation terminée, l'agent de déploiement effectue une vérification au niveau local de l'échec/succès de l'activité d'installation et envoie un message de notification à l'agent de supervision locale selon la situation (message de succès ou d'échec de l'installation).

Notons que, si le GSA utilise le mode de communication synchrone, le GSA envoie les parties du plan en séries, mais il reste bloqué après chaque envoi jusqu'à la réception du message du succès/échec de l'installation. Dès la réception de la réponse, et si l'installation est terminée avec succès dans le sous-réseau, le GSA envoie la partie suivante du plan au deuxième agent de supervision locale. Si Le LSA ne répond pas le GSA utilise la procédure de reconfiguration des LSA pour le remplacé (voir 2).

3.8.2.3 Phase de contrôle et de reconfiguration du déploiement

Une fois les messages de notification (de l'échec/succès) de l'installation sont reçus, l'agent de supervision locale traite localement les messages et notifie par la suite l'information de l'échec/succès de l'installation à l'agent de supervision globale par l'envoi d'un message de succès ou d'échec d'installation dans le sous-réseau.

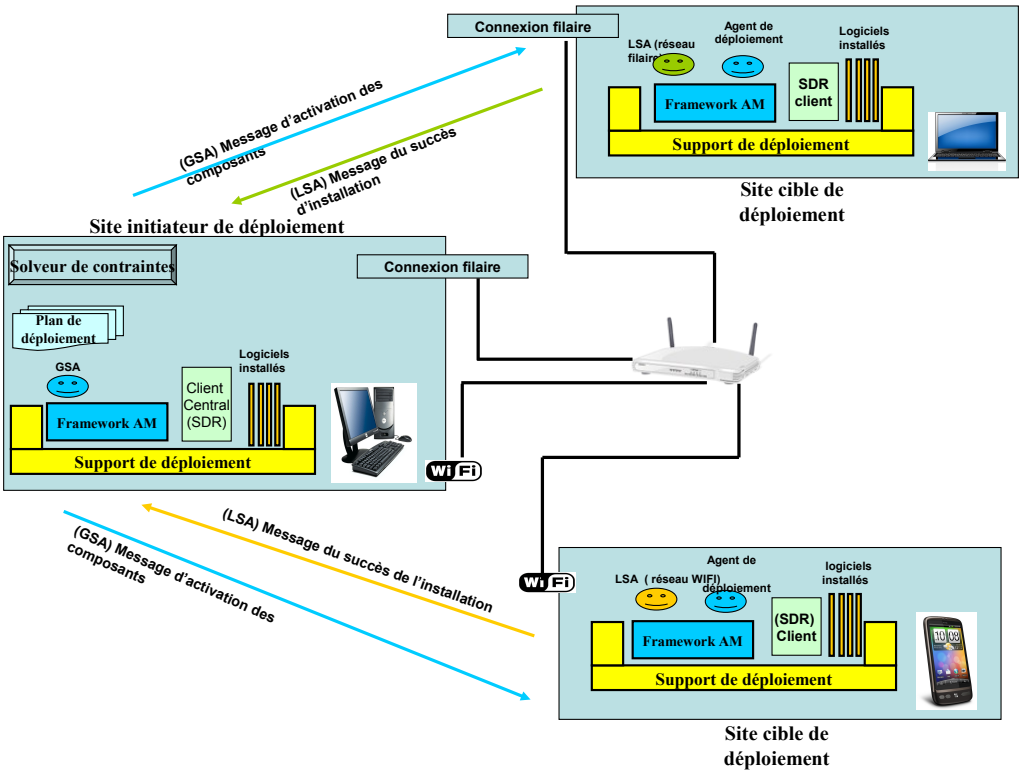

FIGURE 3.12 – Agents de supervision

L'agent de supervision globale envoie un message d'activation du logiciel déployé à tous les agents de supervision locale après la réception de tous les messages de succès de l'installation comme l'illustre les figures 3.11 et 3.12. Le comportement des agents de supervision locale consistera en la création et l'envoi des messages d'activation à tous les agents de déploiement déployés dans le sous-réseau.

La phase d'exécution du plan de déploiement se termine après l'installation et l'activation de l'application dans tous les sites cibles et passe j-ASD à la phase du contrôle du déploiement. Dans cette dernière, l'agent de supervision globale reste à l'écoute du service de découverte du réseau et échange d'une manière permanente des messages de contrôle de l'état avec les agents de supervision locale pour détecter toute situation qui nécessite une procédure d'adaptation comme la panne d'un site cible, les pannes des liens du réseau et la panne d'un agent. Notons ici, que nous ne faisons pas de différence entre une panne de machine / lien du réseau et une panne d'une agent.

Si l'agent de supervision globale ne reçoit aucun message (succès/échec de création par exemple) de la part d'un agent de supervision locale à cause d'une déconnexion ou panne de machine dans un intervalle de temps fini, le GSA exécute une procédure de reconfiguration (voir l'algorithme 2). Cette procédure consiste en la

désactivation de tous les agents de supervision locale par l'envoi d'un message de désactivation et l'envoi d'un message d'auto destruction (suicide) à l'agent qui ne répond pas. Ce dernier, s'il reçoit le message de suicide, il envoie un message d'auto destruction aux agents de déploiement créés et termine son exécution.

En parallèle l'agent de supervision globale créé un nouveau LSA dans un autre site du sous-réseau concerné en respectant les contraintes de déploiement et envoie un message d'activation à tous les LSA désactivés après la réception du message de succès de l'installation de la part du nouveau LSA. Le site choisi peut être un site déjà découvert et qui fait partie de la liste des sites cibles de déploiement utilisés où un nouveau site découvert par le service de découverte du réseau et non utilisé dans le plan de déploiement calculé.

Algorithme 2 *Procedure de reconfiguration d'un LSA*

Entrée : $sub - networkList, hostList$ */* liste des sites cibles et des sous-réseau*/*

```
1: message = null;
2: pour tout LSA qui ne répond pas faire
3:    reSendCurrentMessage(LSA)   /* reSendCurrentMessage(LSA) est une fonction
      qui permet de renvoyer un message (déjà envoyé)à un LSA*/
4:    message = waitForReplyMessage()
5:    si (message == null) alors
6:                                /* Si aucun message reçu, désactiver tous les LSA*/
7:       deactivateAllLSA(); /* deactivateAllLSA() est une fonction d'envoi de message de
         désactivation à tous les LSA*/
8:       sendSuicideMessage(LSA);   /* sendSuicideMessage(LSA) est une fonction
         d'envoi d'un message de fin d'exécution*/
9:       host = getNewLSAhost();    /* Obtention d'un nouveau site pour le LSA*/
10:      newLSA = createNewLSA(host);        /* Création d'un nouveau LSA*/
11:      updateLSAtable(newLSA);             /* Mise à jour de la table des LSA*/
12:      waitForInstallSuccesMessage();
13:      activateAllLSA();                   /* Activation des LSA désactivés*/
14:   fin si
15: fin pour
```

La figure 3.13 illustre le diagramme de séquence de l'exécution de la procédure de reconfiguration pour un système comportant un seul agent de supervision locale.

Il est à noter que l'agent de supervision locale peut migrer vers un autre site cible de déploiement dans le sous-réseau ou ordonner la migration à un agent de déploiement afin de corriger une défaillance détectée (panne de machine ou l'agent dispose de moins de ressources par exemple). Cela offre à notre intergiciel la possibilité de réaliser des opérations de reconfiguration dynamique et autonomique au moment de l'exécution de l'application et/ou processus de déploiement. Le LSA informe l'agent de supervision globale de son déplacement avant et après l'opération

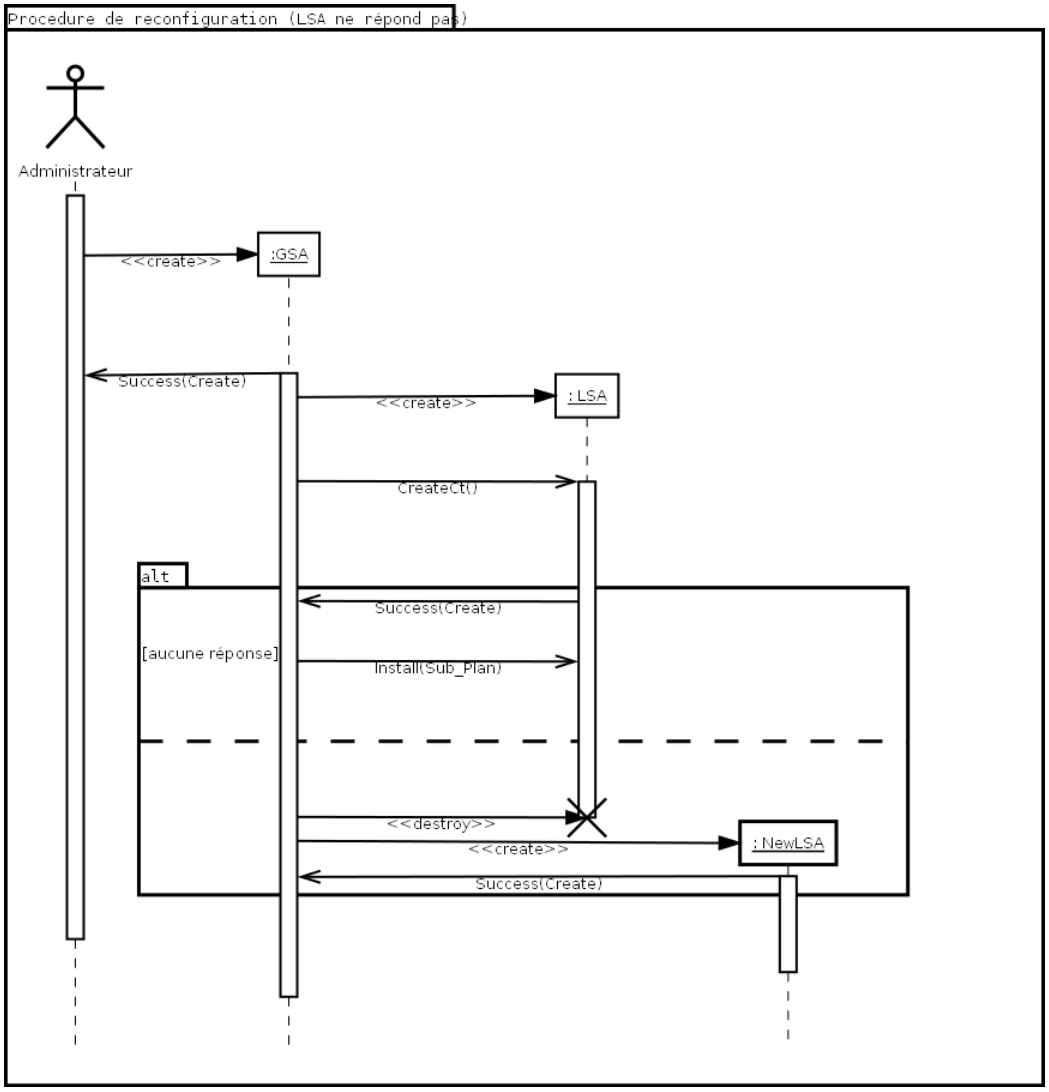

FIGURE 3.13 – Exemple de reconfiguration d'un LSA

de migration en notifiant sa nouvelle position après la correction de la situation.

Si l'agent de supervision locale ne peut pas résoudre localement une défaillance, le GSA sera informé afin de trouver une solution. Le GSA peut par exemple recalculer un nouveau plan de déploiement partiel (spécifique au LSA concerné) afin de trouver une solution à cette défaillance. Dans le cas où aucune solution n'est trouvée, le GSA mettra fin a la phase de déploiement et demande à l'administrateur de déploiement de réduire l'ensemble des contraintes proposées.

De la même manière, l'agent de supervision locale, reste à l'écoute des messages des agents de déploiement et de l'agent de supervision globale et réagit d'une manière autonomique pour corriger les situations de défaillances. Le système d'agent

mobile reste aussi à la disposition de l'administrateur du déploiement et permet la désactivation, la désinstallation et la mise à jour du logiciel déployé à travers des interfaces graphique du déploiement.

L'administrateur de déploiement peut à tout moment désactiver, mettre à jour et désinstaller tout ou partie des composants déployés. Ces activités de déploiement (la désactivation, la mise à jour et la désinstallation) sont exécutées d'une manière similaire a l'activité d'activation. Autrement dit, l'agent de supervision globale commence par la localisation des composants ciblés par la requête de l'administrateur de déploiement et envoie par la suite les messages correspondant à chaque activité de déploiement (des messages de désinstallation par exemple). Les messages générés, sont envoyés aux agents de supervision locale, qui vont réagir aux messages reçus et adoptent le comportement correspondant. Par exemple, si l'agent de supervision globale reçoit une requête de désinstallation de l'application déployée, il envoie des messages de désinstallation à tous les agents de supervision locale. Les LSA, génèrent aussi des messages de désinstallation qui sont envoyés aux agents de déploiement. Les agents de déploiement, désinstallent les composants installés et notifient le succès/ l'échec de la désinstallation aux agents de supervision locale. Ces derniers, notifiant l'échec/succès de la désinstallation à l'agent de supervision globale.

3.9 Conclusion

Dans ce chapitre nous avons présenté l'intergiciel j-ASD dédié au déploiement autonomique de logiciels en environnement ubiquitaire. L'architecture de l'intergiciel j-ASD est composé :

- d'un langage dédié (DSL) à la description des contraintes de déploiement,
- d'un service réseau pour découvrir d'une manière automatique les sites cibles de déploiement,
- d'un logiciel de bootstrap pour la préparation de l'environnement d'exécution,
- d'un solveur de contraintes pour le calcul de plans déploiement initial
- et d'un système d'agents mobiles pour l'exécution et la supervision des activités de déploiement.

Les caractéristiques du système d'agents mobiles (comportement autonome et capacité de migration) nous permettent d'effectuer des adaptations et des reconfigurations dynamiques au moment de l'exécution sans aucune intervention de l'administrateur de déploiement. Nous avons montré que les capacités d'adaptation globale de l'algorithme de déploiement et individuelle des agents mobile adaptables permettent de faire évoluer le logiciel déployé afin de répondre aux besoins d'adaptation rencontrés lors du déploiement.

Nous avons choisi d'utiliser le framework OSGi comme support de déploiement pour permettre le déploiement d'applications (des bundles OSGi) sur plusieurs types terminaux hétérogènes comme les téléphones mobiles, les Smartphones, les tablettes tactiles et ordinateurs. Le service de découverte du réseau et le logiciel de bootstrap permettent la détection et la gestion des droits d'accès aux sites cibles de déploiement avec un minimum d'intervention humaine. De ce fait, la solution générée par le solveur de contraintes est dynamiquement interprétée comme un plan de déploiement initial par le système d'agents mobiles.

Dans le chapitre suivant (chapitre 4), nous donnons des détails d'implémentation de notre intergiciel ainsi que les différentes expérimentations réalisées pour la validation de notre proposition.

Chapitre 4

Implémentation et expérimentations de j-ASD

Sommaire

4.1 Introduction

Afin d'évaluer et de valider notre proposition pour le déploiement logiciel autonomique, nous avons réalisé un prototype entièrement fonctionnel. Il implémente l'architecture logicielle de j-ASD présentée dans le chapitre précédent (chapitre 3). Plusieurs expérimentations sur les capacités de j-ASD on été réalisées avec ce prototype. Les buts des expérimentations réalisées sont respectivement : l'évaluation des performances du service de découverte du réseau et de l'algorithme de déploiement dans un environnement réel.

L'objectif de ce chapitre est de décrire l'implémentation de j-ASD ainsi que les différents résultats des expérimentations réalisées. Nous présentons en premier lieu (dans la section 4.2) les détails de l'implémentation et les différents outils et technologies impliquées pour la réalisation de j-ASD. Nous présentons par la suite (dans la section 4.3) les expérimentations que nous avons menées pour l'évaluation de j-ASD ainsi que les différents résultats.

4.2 Implémentation de j-ASD

Comme expliqué dans le chapitre 3, l'intergiciel j-ASD est une plate-forme dédiée pour le déploiement autonomique de logiciels dans des environnements répartis ouverts tels que les systèmes ubiquitaires et les systèmes P2P. Il est composé d'un langage dédié à la description des contraintes de déploiement, d'un service de découverte des sites cibles d'une manière automatique, d'un logiciel de bootstrap pour la préparation de l'environnement de l'exécution, d'un solveur de contraintes pour le calcul des plans de déploiement et d'un système d'agents mobiles adaptables pour l'exécution, la supervision et la reconfiguration des activités de déploiement.

4.2.1 j-ASD DSL

Les contraintes de déploiement sont exprimées avec le langage dédié à la description de contraintes de déploiement j-ASD DSL. Notre DSL a été conçu et implémenté sous la forme d'un plugin Eclipse en utilisant le framework libre Xtext. En effet, Xtext [1] permet le développement de langages de programmation et de DSL (Domain-Specific programming Language). Il s'appuie sur une grammaire générée ANTLR ainsi que sur le framework de modélisation EMF (Eclipse Modeling Framework).

La figure 4.1 présente un aperçu de l'environnement de développement (Plugin Eclipse) des programmes j-ASD DSL. Il est constitué d'un parseur, d'un compilateur et d'un interpréteur du langage ; l'ensemble des éléments de Xtext sont intégrés complètement dans l'environnement de développement Eclipse. Xtext nous a permis la mise en place de notre langage dédié (langage, compilateur et parseur) j-ASD DSL à partir d'un seul fichier contenant la description de la grammaire du langage.

j-ASD DSL, est un langage dédié à la description des contraintes de déploiement, il permet à l'administrateur de déploiement de décrire le logiciel à déployer, les composants qui forment le logiciel, les services fournis et requis, les dépendances logicielles, les contraintes matérielles et les contraintes liées aux activités de déploiement. La figure 4.2 illustre un extrait d'un programme j-ASD DSL. Il décrit un logiciel nommé `ExtractFromScenario_1`. Il comporte deux composants (`ramSize` et `display`). Les composants `ramSize` et `display` sont définis par leurs noms, leurs versions et leurs URL respectives. Les JARs des composants ramSize et display sont récupérés à partir d'un serveur http. La partie `hostConstraint` représente une spécification des contraintes d'affichage (`Display-Constraint`) du composant display dans les sites cibles de déploiement. Les contraintes exprimées signifient que dans les sites cibles :

– La charge du processeur (CPULoad) doit être inférieure à 80%.

1. http ://www.eclipse.org/Xtext/

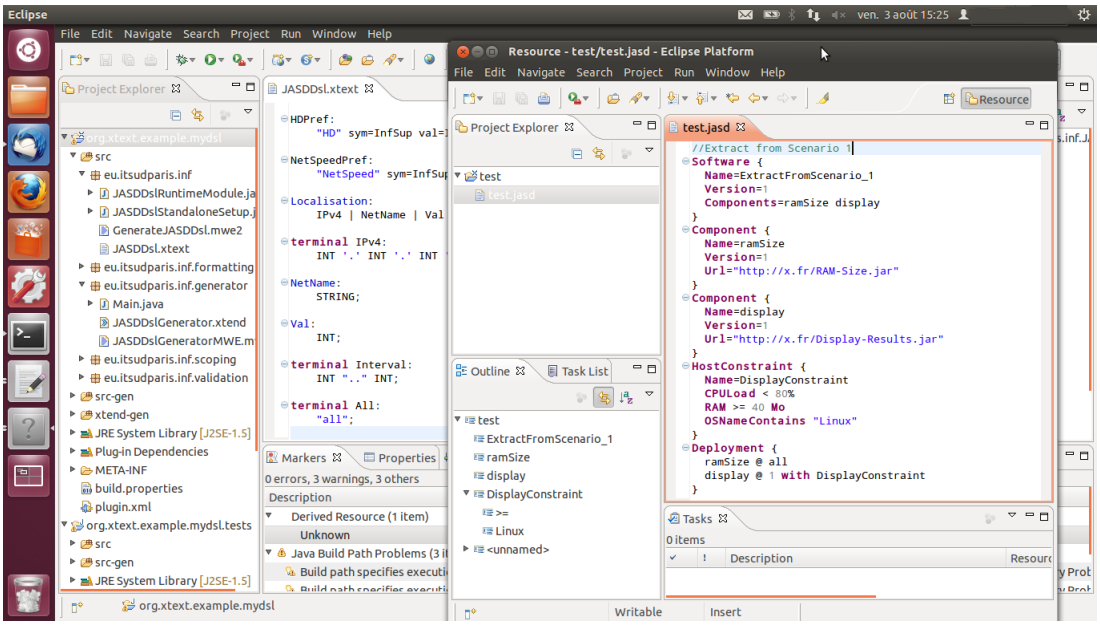

FIGURE 4.1 – Environnement de développement des programmes j-ASD DSL

– Le composant à besoin d'un minimum de 40 Mo de mémoire.
– Le système d'exploitation dans le site qui accueil le composant doit être un système Linux.

Les contraintes de déploiement sont une spécification de contraintes de haut niveau, qui expriment que le composant ramSize doit être déployé dans tous les sites disponibles au moment du déploiement et le composant display doit être déployé dans un seul hôte qui satisfait les contraintes d'affichage (`Display-Constraint`).

4.2.2 Résolution des contraintes

"j-ASD" utilise la librairie java open source CHOCO [CHOCO Team 2010] pour la résolution des contraintes de déploiement afin de calculer un plan initial de déploiement. En effet, le programme j-ASD DSL est édité par l'administrateur de déploiement en utilisant le plugin Eclipse généré, ensuite, il est compilé et transformé en un nouveau programme sous la forme d'un problème de satisfaction de contraintes de bas niveau (programme CHOCO).

Comme expliqué dans le chapitre précédent, le programme généré est chargé automatiquement par j-ASD. Il est résolu par le solveur de contraintes de CHOCO. La solution trouvée par ce dernier est sous la forme de deux matrice d'un ensemble de variables entières qui sont respectivement les variables de localisation et les variables de collocation. Nous rapelons que la matrice des variables de localisation

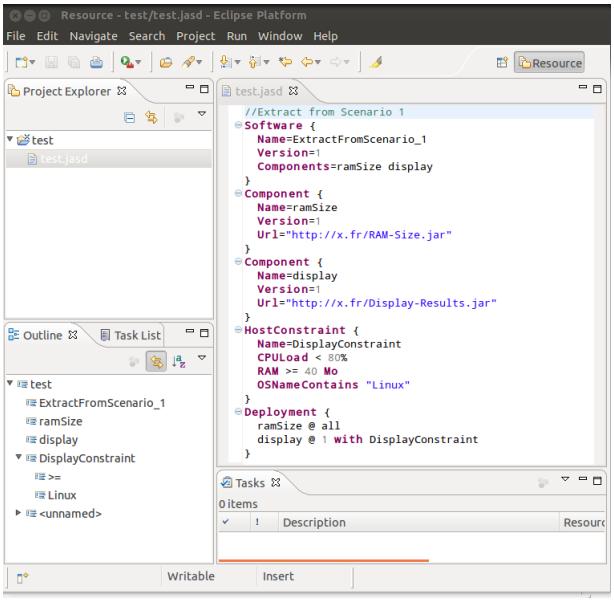

FIGURE 4.2 – Extrait d'un programme j-ASD DSL

désigne l'emplacement des composants dans les sites cible de déploiement.

La figure 4.3 illustre une matrice de localisation de cinq composants sur dix sites cibles de déploiement. Les lignes de la matrice représentent les composants de l'application à déployé et les colonnes représentent les sites cible de déploiement. Comme expliqué précédemment, pour un composant Ci et un site Hj, si Loc (Ci, Hj) = 1, le composant Ci doit être déployé dans le site Hj.

La matrice de collocation permet d'identifier si un composant Ci supporte les collocations avec d'autres composants. Pour deux composants Ci et Cj, si colloc (Ci, Cj) = 2, le composants Ci doit être installé seul dans les sites cibles d'accueil.

Les deux matrices de localisation et de collocation seront mappées en un plan de déploiement par le système d'agents mobiles. Les mécanismes de traduction en plan de déploiement sont les suivants :

1. Pour chaque variable de localisation, si loc (Ci, Hj) = 1 alors, le composant Ci doit être déployé dans le site Hj. Pour cela l'agent de supervision (locale ou globale) créé un agent de déploiement dans le site Hj. L'agent de déploiement une fois qu'il à reçu l'URL du composants et les URL des dépendances du composant (par un message) il adopte un comportement associé au message et commence automatiquement le déploiement du composant Ci dans le site cible.
2. Dans le cas où la variable de localisation loc(Ci, Hj) = 0 alors, l'agent de supervision (globale ou locale) n'effectuera aucune action, par conséquent l'agent

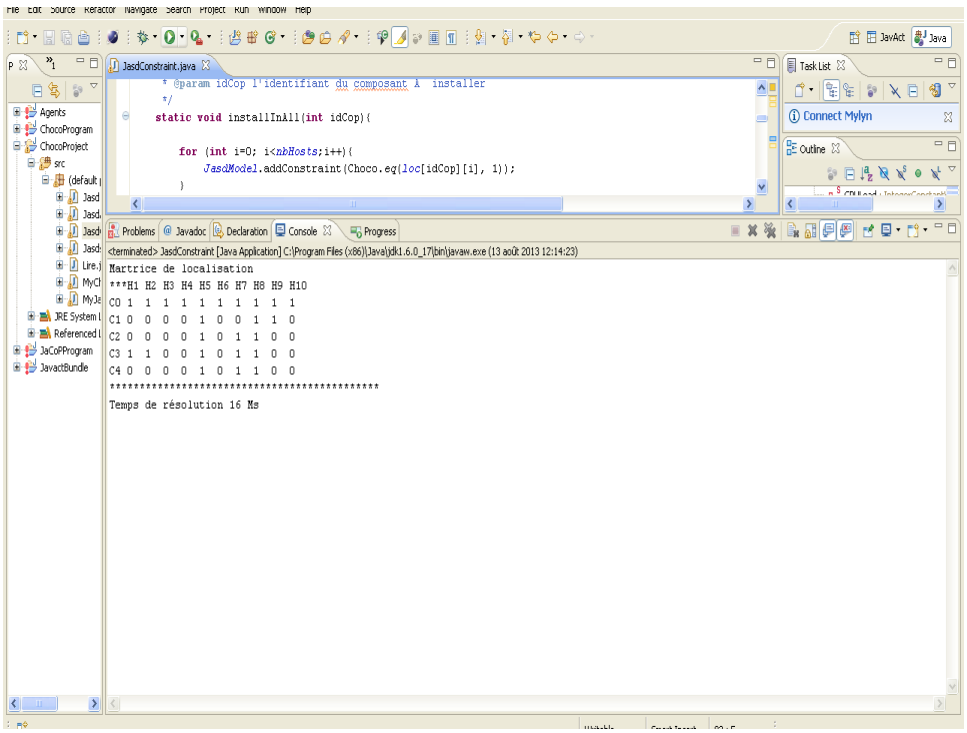

FIGURE 4.3 – Exemple d'une matrice de localisation

de déploiement ne recevra aucun message de déploiement du composant Ci dans le site Hj.

3. Si dans le même site Hj, plusieurs variables de localisation sont égales à 1 (loc (Ci, Hj) = loc (Ck, Hj) = 1), l'agent de supervision vérifiera l'état des variables de collocation entre les composants concernés. Si les variables de collocation sont égales à 0 ou 1, dans ce cas l'agent de déploiement recevoir autant de messages d'installation que de composants à déployer dans le site cible concerné.
4. Si pour un site Hj, loc (Ci, Hj) = 1 et colloc (Ci, Ck) = 2, dans ce cas l'agent de supervision doit s'assurer d'installer le composant Ci tout seul dans le site Hj.

4.2.3 Découverte des sites cibles

Pour calculer le plan de déploiement, j-ASD à besoin d'une liste de sites cible de déploiement. Pour fournir cette dernière d'une manière automatique, nous avons proposé et implémenté un service de découverte du réseau pour la détection automatique des sites cible de déploiement. En effet, nous avons étudié plusieurs technologies et protocoles existants présentés en annexe. Après cette étude, nous avons

choisi les protocoles XMPP et UPnP avec leurs implémentations java open source respectives Smack [2] et Cyberlink [3]comme protocoles de base à notre système de découverte du réseau. L'architecture d'un client central du système de découverte est présentée dans la figure 4.4. Le client comporte un module de découverte UPnP (CDevice) pour la découverte locale, un module de découverte XMPP (XMPPManager) pour la découverte étendue et un module de découverte mixte (Discovery) qui exploite les deux modules ensemble pour réaliser une découverte mixte.

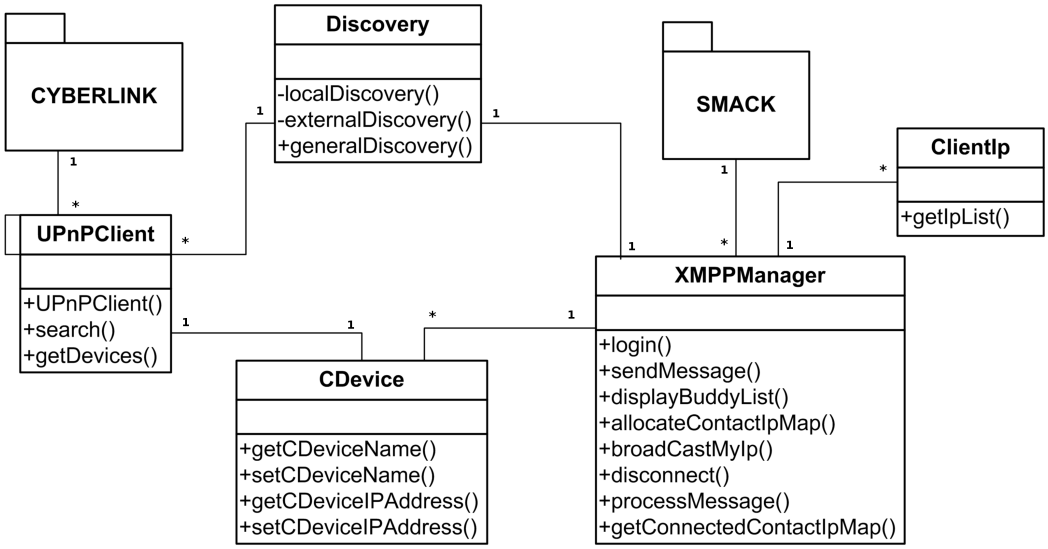

FIGURE 4.4 – Diagramme de classes du client central de découverte du réseau

Nous avons aussi implémenté un logiciel de bootstrap sous la forme d'un JAR java qui permet de préparer l'environnement d'exécution. Le logiciel de bootstrap établit une connexion ssh en utilisant la bibliothèque java jcraft [4]. Une fois la connexion établie, le logiciel de bootstrap télécharge et installe l'environnement d'exécution de j-ASD sur le site cible. L'environnement d'exécution de j-ASD intègre un client du service de découverte du réseau qui est installé et lancé automatiquement.

Le système de découverte du réseau repose sur un modèle client central (client XMPP, point de contrôle et Device UPnP) – clients périphériques (client XMPP, PC et Device UPnP). Le client central est installé et exécuté dans le site initiateur de déploiement et joue le rôle de collecteur d'informations parvenant des clients périphériques. Les clients périphériques sont installés et exécutés dans les sites cibles de déploiement. Tous les clients sont connectés à un serveur XMPP. Le client central reste à l'écoute attentive des clients périphériques pour recenser leurs présences sur le réseau et récupère leurs localisations. Autrement dit, il établit une liste de

2. http ://www.igniterealtime.org/projects/smack
3. http ://www.cybergarage.org/twiki/bin/view/Main/CyberLinkForJava
4. http ://www.jcraft.com/jsch/

sites cibles de déploiement en récupérant les informations suivantes de chaque site détecté : le site cible de déploiement, la disponibilité du site et l'adresse IP du site. Le client central récupère aussi un certain nombre d'informations sur l'état des ressources dans les sites cibles de déploiement comme la taille de la mémoire et le type du système d'exploitation.

Les informations récupérées sont actualisées toutes les quinze secondes. Nous avons choisi cet intervalle d'actualisation après plusieurs expérimentations réalisées. Les expérimentations ont montrées que quinze secondes est la période minimum nécessaire au client central pour recevoir et traiter tous les messages signalisation de présence envoyés par 100 clients périphériques. Si l'administrateur de déploiement cible plus de 100 sites cible, il est possible d'augmenter l'intervalle de rafraichissement selon le nombre de sites cible souhaités.

La figure 4.5 montre un aperçu de l'interface graphique du système de découverte du réseau au niveau du client central. Dans cette figure, le premier champ d'information affiché (Device Name) est le nom du site cible, le deuxième (Device Presence) correspond à la disponibilité du site et le dernier champ (Device Ip) correspond à l'adresse IP du site.

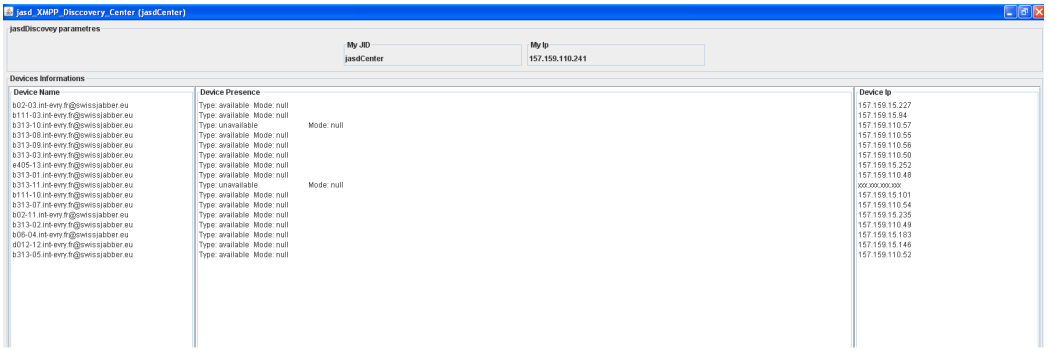

FIGURE 4.5 – Client central du système de découverte du réseau

Les communications entre les clients sont assurées par le protocole XMPP si l'administrateur de déploiement réalise une découverte à large échelle. Elles sont assurées par mécanisme de communication d'UPnP si l'administrateur réalise une découverte en local. Le système utilise le protocole XMPP et UPnP ensemble dans le cadre de la découverte multi-échelles.

4.2.4 Intergiciel de déploiement autonomique

Nous avons utilisé le Framework libre JavAct [5] comme plate-forme d'agents mobiles [Arcangeli 2001] et le Framework OSGi comme support de déploiement. OSGi

5. http ://www.javact.org

est une spécification qui définit une plate-forme d'exécution de composants appelés Bundles. Le bundle OSGi est notre unité physique de déploiement. Le déploiement de d'autres types d'unité de déploiement ne fait pas partie des objectifs de cette thèse. Concrètement, un bundle est un fichier java (archive JAR) qui contient un manifest et une combinaison de fichiers de classes java et des ressources. Le framework OSGi permet l'installation, la désinstallation, l'activation, la désactivation et la mise à jour de bundles au moment de l'exécution et inclut des mécanismes qui permettent la gestion automatique des dépendances. L'utilisation d'OSGi comme support de déploiement permet le déploiement de logiciel sur plusieurs types d'infrastructures hétérogène. La réutilisation des fonctionnalités de déploiement fournis par le Framework OSGi comme, l'installation et l'activation de bundles, nous permet de nous concentrer sur d'autres aspects du processus de déploiement.

Nous avons aussi développé le système d'agents mobiles ainsi qu'un algorithme de déploiement pour l'installation, l'activation, la désactivation, la désinstallation et la mise à jour de bundles. Le prototype nous permet de créer et d'envoyer nos agents mobiles spécialisés aux sites cible de déploiement pour exécuter et superviser les activités de déploiement dans les environnements les plus instables. Nous avons utilisé un processus qui se base sur deux types d'agents de supervisions qui sont respectivement les agents de supervision globale et les agents de supervision locale pour permettre le passage à l'échelle, et améliorer les temps de déploiement.

L'utilisation des agents mobiles nous offre des possibilités de reconfigurations et d'adaptations dynamiques au contexte de l'exécution du logiciel déployé. En effet, les agents de déploiement peuvent prendre des décisions décentralisés et autonomes, sans la consultation des agents de supervision pour corriger une situation de défaillance du processus de déploiement. Un exemple typique d'adaptation est la décision de migration vers un autre site cible si le site actuel ne dispose pas de suffisamment de ressources.

Le prototype est fonctionnel et à été expérimenté dans un environnement réel. La section suivante présente les expérimentations réalisées sur ce prototype ainsi que les différents résultats d'évaluation.

4.3 Expérimentations

Cette section présente les expérimentations menées pour évaluer notre intergiciel, elles représentent aussi une validation expérimentale de notre approche. D'après [Tempich 2003], l'évaluation peut se réaliser de deux manières qui sont respectivement : une évaluation basée sur la perception de l'utilisateur final (l'administrateur de déploiement dans notre cas) et une évaluation basée sur les capacités du système développé. Dans cette thèse, nous avons réalisées nos évaluations selon la deuxième approche, autrement dit une évaluation basée sur les capacités du système déve-

loppé. Les objectifs des expérimentations réalisées sont respectivement : l'évaluation des performances du service de découverte du réseau, l'évaluation des performances du temps de calcul du plan de déploiement initial et enfin, l'évaluation des performances de l'algorithme de déploiement et de reconfiguration sur un environnement réel.

4.3.1 Environnement cible des expérimentations

Les expérimentations présentées ici ont été menées sur deux types de machines (voir la figure 4.6.

Le premier type correspond aux 100 ordinateurs des salles de travaux pratiques à Télécom SudParis. Leur nom de domaine est `*.int-evry.fr`. Leur configuration matérielle et logicielle est la suivante :

- un processeur Intel(R) Xeon (R) 2.80GHz ;
- 4 Go de mémoire ;
- deux systèmes d'exploitation Linux Fedora 15 et Microsoft Windows 7 ;
- les ordinateurs sont reliés ensemble par un réseau filaire de type Ethernet.

Le deuxième type de machines correspond à dix machines Samsung de type tablette tactile équipées d'un processeur Intel(R) processor 800MHZ, 1 Go de mémoire et deux systèmes d'exploitation Microsoft Windows XP (Tablette PC Edition 2005) et Linux Ubuntu. Toutes les machines sont connectées à un réseau WiFi.

Le processus de déploiement commence toujours à partir d'un site initiateur de déploiement. Le site initiateur du processus de déploiement utilisé dans les expérimentations est un ordinateur portable équipé d'un processeur dual core Intel Pentium III Xeon 2.4 Ghz, 4 GO de mémoire et deux système d'exploitation Microsoft Windows XP et Linux Fedora 15.

Dans les expérimentations présentées par la suite, toutes les machines ont été équipées avec l'environnement d'exécution de j-ASD en utilisant le logiciel de bootstrap avant le lancement du processus de déploiement. L'environnement d'exécution de j-ASD dans un site cible de déploiement est constitué des éléments suivant :

- Le support de déploiement (Framework OSGi) qui consiste en un fichier .jar (org.eclipse.osgi_3.8.0.v20120529-1548) qui permet le lancement du Framework OSGi.
- Le bundle OSGi jasdCleint.jar, qui permet la création et l'exécution du client XMPP, d'un point de contrôle UPnP, d'un Device UPnP. Il permet aussi la création et l'exécution du système d'agents mobile dans le site cible de déploiement (agent de supervision locale et agents de déploiement). Enfin il permet d'assurer toutes les communications entre les agents.

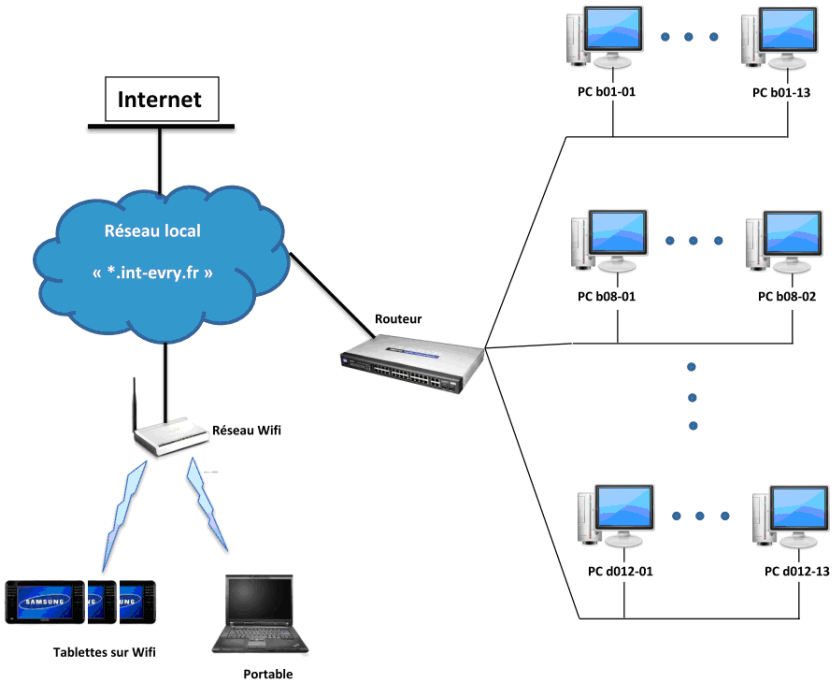

FIGURE 4.6 – Environnement des expérimentations

– Le bundle smack (smack.jar) qui consiste en une version OSGi de la librairie java smack du protocole XMPP. Le bundle smack permet d'assurer l'exécution des clients XMPP de notre service de découverte du réseau.
– Le fichier `awfullPolicy`, qui permet de définir les permissions nécessaires à la plate forme JavAct pour s'exécuter correctement dans un terminal.

Le logiciel de bootstrap est un programme java, chargé d'établir une connexion sécurisé (ssh) avec tous les sites cibles disponibles. Une fois la connexion établie, le bootstrap lance un script (dans chaque site cible connecté) qui permet de créer un répertoire dédié à l'intergiciel de déploiement j-ASD. Par la suite, il télécharge dans ce répertoire tous les fichiers de l'environnement de l'exécution de j-ASD. Dès le succès du téléchargement des fichiers, un autre script est lancé pour l'activation du Framework OSGi. Si le Framework est exécuté correctement, l'installation et l'activation des bundles `jasdCleint.jar` et `smack.jar` est réalisé et le processus de préparation de l'environnement de l'exécution est terminé.

4.3.2 Temps moyen de découverte du réseau

Dans la première série d'expérimentations, nous nous somme concentrés sur le temps moyen de découverte des sites cible de déploiement. Nous rappelons que l'objectif de notre intergiciel est de réaliser du déploiement logiciel à grande échelle.

Pour cela, volontairement nous avons concentré cette partie des expérimentations sur l'évaluation du module de découverte étendue de j-ASD qui se base sur le protocole XMPP. Les serveurs XMPP utilisés par notre service de découverte du réseau au courant des expérimentations sont respectivement : le serveur libre `swissjabber.eu` et `xmpp.jp`. Nous avons réalisé un ensemble de tests sur le réseau filaire, le réseau WiFi et une série de tests sur les deux réseaux (filaire et WiFi) en parallèle. Toutes les expérimentations ont été réalisées après le processus de préparation de l'environnement de l'exécution (le lancement du logiciel du bootstrap).

Les expérimentations consistent au lancement du service de découverte du réseau pendant des périodes de temps préalablement fixées, ensuite nous procédons au recensement du pourcentage des sites cibles détectés. Nous avons réalisé les expérimentations sur les périodes de durées (en seconde) suivantes : 1, 2, 4, 5, 8, 10, 12, 14, 15, 16, 17, 18, 19 et 20 secondes. Chaque expérimentation a été réalisée dix fois de suite et pour chaque période le résultat final est la moyenne des valeurs obtenues.

Les résultats obtenus sont présentés dans les figures 4.7, 4.8 et 4.9.

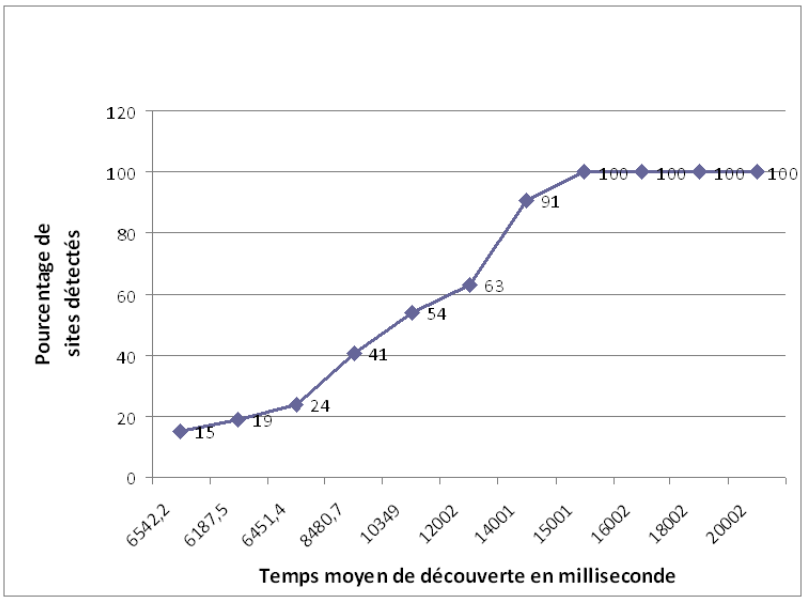

FIGURE 4.7 – Pourcentage et temps de découverte de 60 sites cibles dans le résau filaire

La figure 4.7 présente le pourcentage des sites cible détectés par rapport au temps de découverte pour un ensemble de 60 sites cible de déploiement connectés au réseau filaire. Les expérimentations ont montrées qu'un temps de découverte inférieur à 10349 millisecondes (10 secondes) permet de détecter au meilleur des cas 54% des sites cible de déploiement réellement disponibles. Autrement dit, sur

60 sites cibles, nous avons détecté uniquement 31 sites. Un temps de découverte de 14 secondes nous a permis de détecter dans certaines situations tous les sites cibles de déploiement, dans cette période nous avons pu détecter en moyenne 91 % de sites cibles réellement disponibles. Nous jugeons que ce pourcentage reste insuffisant et peut influencer négativement le succès d'un processus de déploiement. A partir d'un temps de découverte moyen de 15 secondes, tous les sites cibles (60 sites) de déploiement dans le réseau filaire ont été bien détectés avec succès. Le résultat reste le même (détection de tous les sites) pour un temps de découverte de 17, 18, 19 et 20 secondes.

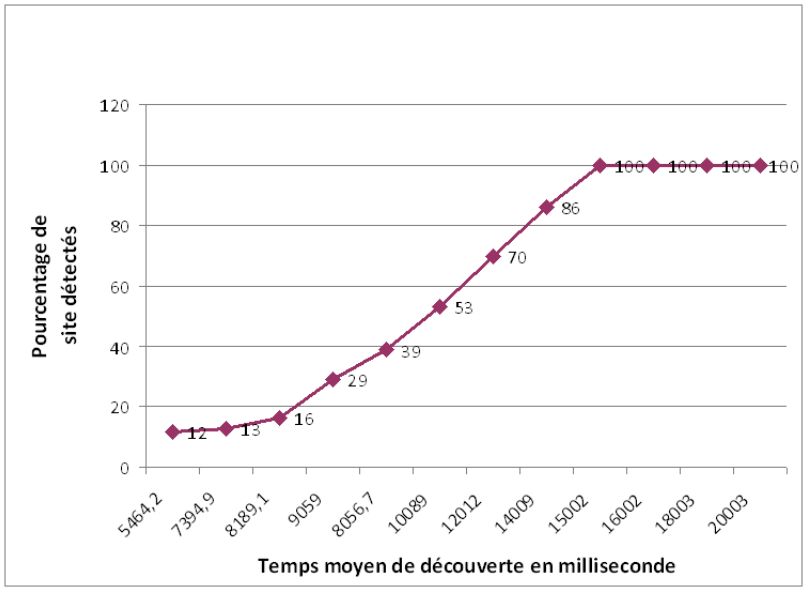

FIGURE 4.8 – Pourcentage et temps de découverte de 10 sites en WiFi

La figure 4.8 illustre le pourcentage des sites cibles détectés dans le réseau WiFi. Nous rappelons que nous avons réalisé les expérimentations sur 10 machines de type tablette Samsung connectées au réseau Wi-Fi. Le processus de découverte à été lancé à partir d'un ordinateur portable connecté aussi au réseau WiFi. Dans les expérimentations réalisées, un temps de découverte de 14 secondes permet de détecter uniquement 86 % des sites disponibles. Tandis qu'un temps de découverte de 15 secondes nous à permis de détecter avec succès les dix sites cibles connectés au réseau sans fil. Nous remarquons que le temps de découverte nécessaire pour la découverte des dix sites cibles connectés au réseau WiFi est le même que le temps de découverte de 60 sites connectés en mode filaire. Le temps de découverte dans un réseau sans fil (Wi-Fi dans notre cas) reste très important par rapport au temps de détection des sites cibles dans un réseau filaire. Cela est dû à la latence engendrée d'un côté par le routeur Wi-Fi et le serveur XMPP et d'un autre côté par les performances de la configuration des machines connectées au réseau Wi-Fi (un processeur Intel processor 800MHZ 1 Go de RAM). Des expérimentations supplémentaires sur

le réseau Wi-Fi doivent être réalisées avec une configuration matérielle similaire aux machines utilisées dans le réseau filaire pour pouvoir comparer correctement les résultats obtenus.

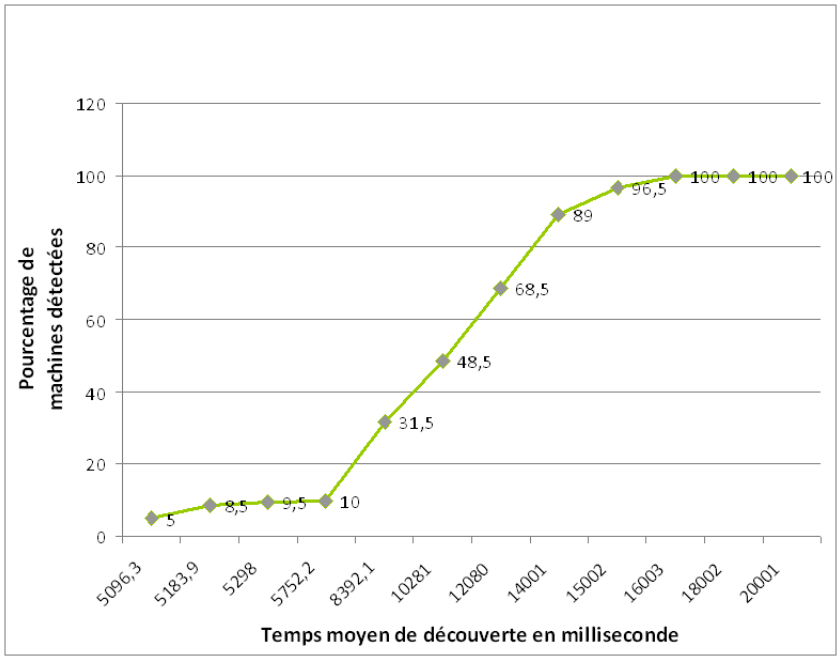

FIGURE 4.9 – Pourcentage et temps de découverte de 70 sites dans le réseau filaire et WiFi

La figure 4.9, expose les résultats des expérimentations dans un réseau mixte comportant dix sites connectés en Wi-Fi et soixante ordinateurs connectés au réseau filaire, sous la forme d'un pourcentage des sites détectés par rapport au temps moyen de découverte. Le site initiateur de déploiement à été connecté aux deux réseaux (filaire et Wi-Fi) en même temps. Un temps de découverte de 10 secondes permet de détecter en moyenne 35 sites cibles sur les 70 sites disponibles. Un temps de découverte de 15 secondes qui permet de détecter tous les sites cibles dans les deux premières séries d'expérimentation nous a permis de découvrir la présence de 96,5% des 70 sites cibles connectés au réseau Wi-Fi et filaire en même temps. Pour détecter tous les sites, le temps de découverte moyen est de 16 secondes. Le pourcentage de détection reste le même pour des temps de découverte de 17, 18, 19 20 secondes. Les expérimentations ont montrées aussi que le temps de découverte moyen nécessaire pour la détection d'une nouvelle connexion d'un site cible et / ou une déconnexion est de 15 secondes.

4.3.3 Temps de calcul du plan de déploiement initial

Après la détection des sites cibles et l'écriture des contraintes de déploiement, l'administrateur de déploiement peut alors lancer l'étape du calcul du plan de déploiement.

Nous rappelons que l'administrateur de déploiement déclare un ensemble de contraintes sous la forme d'un programme en utilisant le DSL j-ASD. Le programme est ensuite compilé et transformé afin de générer un programme de satisfaction de contraintes de bas niveau résolvable par le solveur de contraintes CHOCO. Les expérimentations présentées dans cette sous-section portent sur le temps nécessaire pour que j-ASD puisse trouver une solution consistante (qui ne viole aucune contrainte de déploiement). Cette dernière, une fois trouvée est le plan de déploiement initial.

Nous avons réalisé plusieurs expérimentations pour mesurer le temps moyen nécessaire pour le calcul d'un plan de déploiement initial pour plusieurs logiciels composés respectivement de 5, 8, 10, 15 et 20 composants.

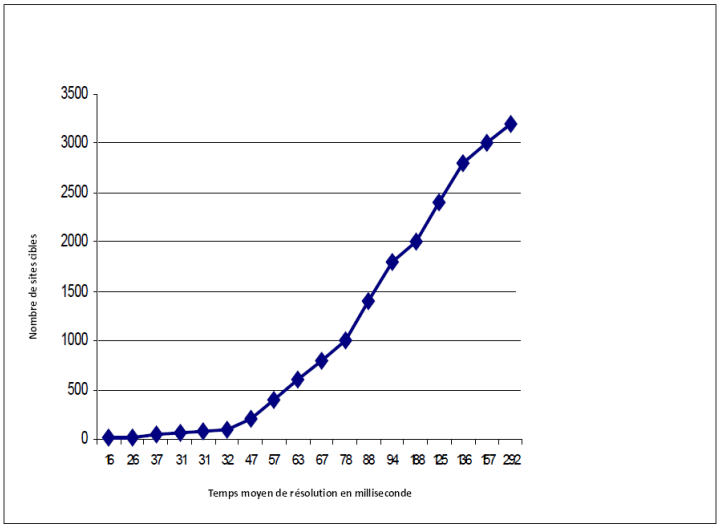

FIGURE 4.10 – Temps moyen pour calculer le plan de déploiement initial

Dans le chapitre 1, nous avons présenté deux scénarios de motivations. Le premier scénario consiste au déploiement d'une application constituée de huit composants (nœud de contexte COSMOS). La figure 4.10 montre le temps moyen nécessaire pour le calcul d'un plan de déploiement initial pour ce logiciel. Le temps nécessaire pour calculer un plan de déploiement de cette application dans dix sites cibles est de 16 millisecondes. Le temps nécessaire pour trouver un plan de déploiement initial pour le déploiement des huit composants de l'application sur 3200 machines en conservant les mêmes contraintes est de 292 millisecondes.

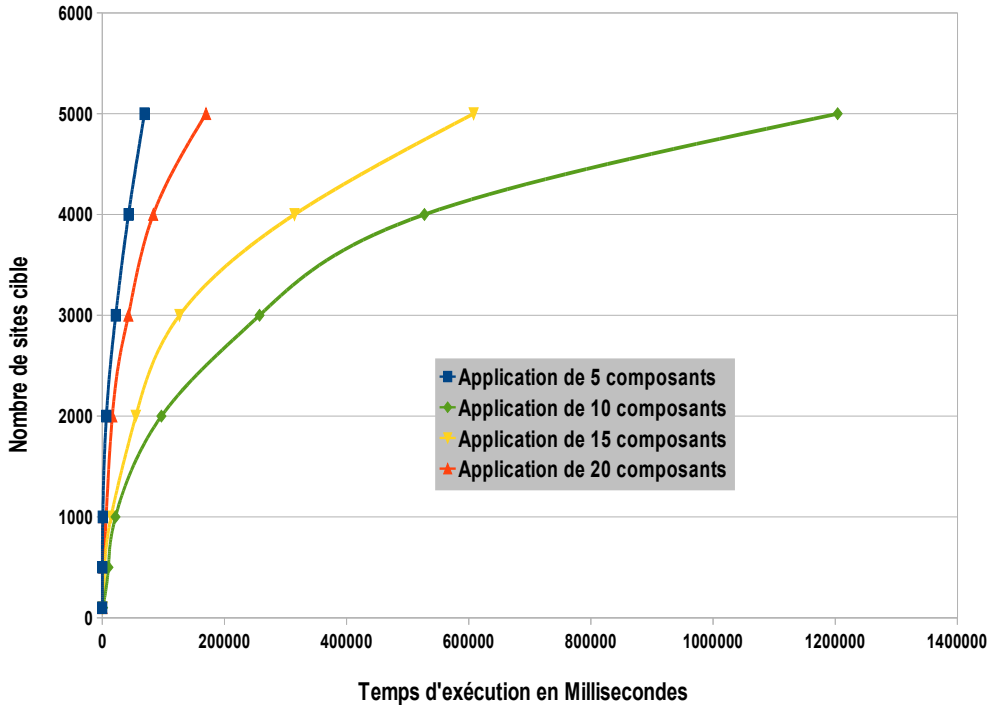

FIGURE 4.11 – Temps moyen pour calculer le plan de déploiement initial

La figure 4.11, illustre le temps nécessaire pour le calcul d'un plan de déploiement initial pour quatre applications comportant respectivement 5, 10, 15 et 20 composants. Les quatre applications disposent des mêmes contraintes de déploiement. Nous avons calculé des plans de déploiement permettant le déploiement des quatre applications sur 100, 500, 1000, 2000, 3000, 4000 et 5000 sites cibles. Le temps nécessaire pour calculer le plan de déploiement initial sur 100 machines est inférieur à une seconde pour tous les applications, tandis que le temps nécessaire pour un plan sur 5000 machines varie dans chaque situation. Globalement, le temps nécessaire pour le calcul du plan de déploiement des quatre logiciels sur 2000 machines est toujours inférieur à deux minutes. Enfin, le temps nécessaire pour le calcul du plan de déploiement initial de l'application formée de 20 composants sur 2000 machines cibles avec les mêmes contraintes est inférieur à une minute, tandis que le temps de calcul du plan de déploiement de la même application sur 5000 machines cibles est supérieur à vingt minutes.

Le temps de calcul dépend fortement du nombre de variables intermédiaires générées par le solveur de contraintes et les performances du solveur utilisé. En effet, l'augmentation du nombre des composants, des sites cibles et des contraintes de déploiement engendre la génération de beaucoup de variables intermédiaires, par

conséquent l'augmentation du nombre des branches de l'arbre de recherche et du temps nécessaire pour trouver une solution consistante. Nous rappelons ici, que notre objectif n'est pas de trouver la meilleure solution de déploiement (plan de déploiement optimale). La solution retournée est la première solution consistante trouvée qui ne viole aucune contrainte de déploiement.

4.3.4 Temps de déploiement

La troisième série des expérimentations, consiste à évaluer les performances de l'algorithme de déploiement. Nous rappelons que notre algorithme de déploiement commence après l'étape de résolution des contraintes et le calcul d'un plan de déploiement initial. L'algorithme prend en entrée le plan de déploiement et la liste de sites cibles correspondante.

L'algorithme démarre son exécution par une phase de vérification pour s'assurer de la disponibilité des sites choisis comme sites cibles de déploiement. Par la suite, la phase d'exécution du plan de déploiement est lancée. Cette étape constitue le cœur de l'algorithme de déploiement, dans laquelle le système d'agents mobiles est lancé et le plan de déploiement est exécuté. La dernière phase est l'étape de contrôle et de reconfiguration, dans laquelle l'intergiciel s'assure du bon déroulement du plan de déploiement et engage un processus de reconfiguration pour corriger toutes les situations de pannes et de déconnexions.

Dans cette série d'expérimentations, nous nous somme concentrés sur l'évaluation des performances de la phase de déploiement de notre algorithme. Pour cela, on calcule le temps nécessaire pour exécuter le plan de déploiement initial en faisant varier le nombre de sites cibles de déploiement et le nombre des composants qui forment l'application. Le temps nécessaire pour la vérification de la disponibilité des sites cibles est de l'ordre d'une seconde. Nous avons déployé quatre applications composées de 5, 8 et 10 bundles sur un ensemble de machines comportent des machines connectées à notre réseau filaire et dix machines de type tablettes connectées au réseau WiFI.
Dans le réseau filaire, chaque application a été déployée sur plusieurs sous-ensembles de sites qui comportent 5, 10, 20, 40 et 60 machines.

Chaque application a été déployée selon les deux modes de déploiement fournie par notre intergiciel (synchrone et asynchrone). Chaque processus de déploiement a été exécuté dix fois successivement et les résultats présentés sont la moyenne du temps global de l'installation.

Le temps de déploiement commence au moment de la création de l'agent de supervision globale et se termine au moment de la réception du dernier message du succès de l'installation. Le temps de l'installation comporte le temps de la création

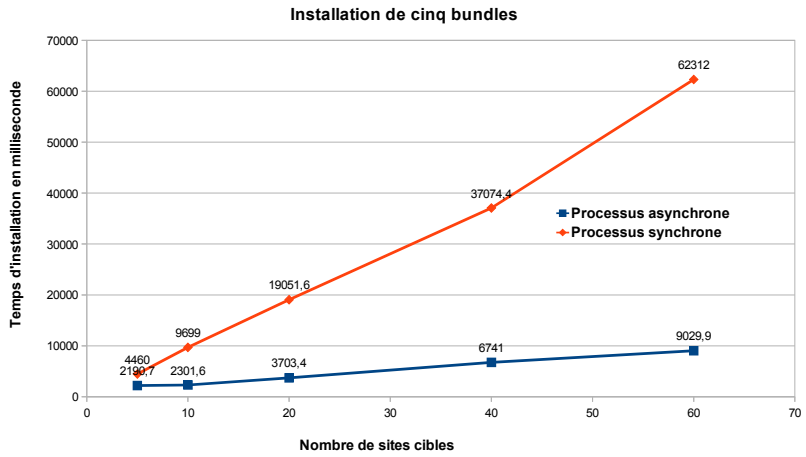

FIGURE 4.12 – Temps moyen de déploiement de cinq bundles

des agents mobiles, et le temps de l'installation d'une application avec toutes ses dépendances logicielles dans les sites cibles de déploiement.

Les figures 4.12, 4.13, 4.14, présentent respectivement les résultats obtenue lors du déploiement des applications en utilisant les deux modes (synchrone et asynchrone) de déploiement avec l'utilisation de deux agents de supervision locale. Les résultats obtenues nous ont permis de confirmer la pertinence du mode de déploiement asynchrone. Par exemple, le temps nécessaire pour installer les applications constituées de huit et dix bundles sur 60 machines en utilisant le processus synchrone est de 3 minutes environ, tandis que le temps nécessaire pour le déploiement des deux applications en utilisant le processus asynchrone est de 12205 millisecondes (12, 2 secondes) pour l'application de huit composants et 12229 (12,22 secondes) millisecondes pour l'application de dix composants. Les résultats obtenus sont très satisfaisante et prouve la faisabilité la pertinence de notre algorithme de déploiement.

Nous avons aussi comptabilisé le temps d'installation moyen dans des situations de défaillance en introduisant volontairement des situations de déconnexion de machines au moment du déploiement. En effet, au moment du déploiement nous arrêtons volontairement et d'une manière aléatoire un sous-ensemble de machines. Le comportement associé à cette situation est le suivant : après la détection d'une déconnexion par l'agent de supervision (globale ou locale), un ou plusieurs nouveaux sites cibles sont choisis pour remplacer les sites déconnectés (ou tombés en

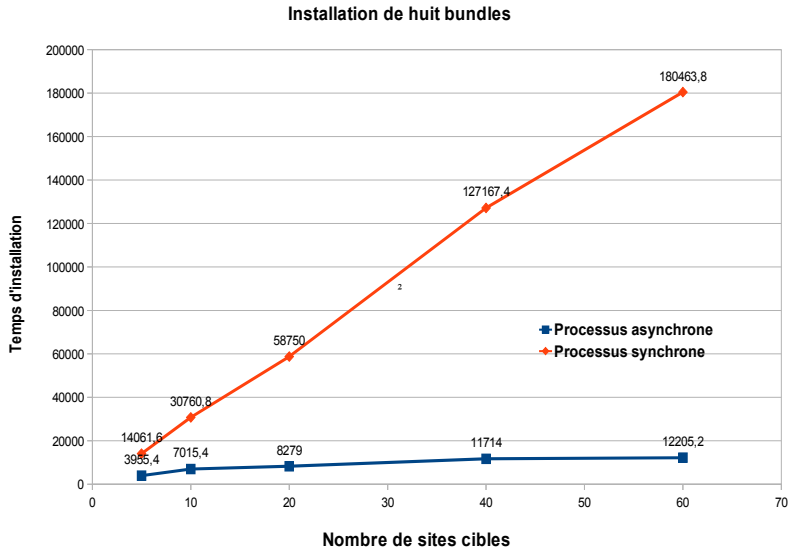

FIGURE 4.13 – Temps moyen de déploiement de huit bundles

panne), par la suite l'agent de supervision (globale ou locale) créé un ou plusieurs nouveaux agents de déploiement et envoie un ou plusieurs messages d'installation aux agents concernés.

Un autre type de défaillance que nous avons considéré consiste à la violation d'une contraintes de déploiement dans un site cible de déploiement, par exemple, le site dispose de moins de mémoire que prévu. Dans ce cas, le comportement est le suivant : l'agent de déploiement choisit un nouveau site cible de déploiement, informe son agent superviseur et réalise une migration vers le nouveau site. Une fois la migration réalisée, l'agent de déploiement notifie sa nouvelle position et le processus d'adaptation et terminé.

Les deux types de défaillance, ont été introduit au moment du déploiement initial. Dans notre expérimentation nous avons déployé l'application de références (application du scénario 1), sur un sous ensemble de 10 machines cibles. Au moment du déploiement, nous avons en premier lieu arrêté 2 machines cibles (soit 2% des machines cible) et en deuxième lieu nous avons violé une contraintes de déploiement dans un site cible.

Le temps nécessaire pour réaliser le déploiement initial de notre application et corriger les situations de défaillance est respectivement de 44936,6 et 51081 millisecondes. Soit 44.5 secondes pour corrigé une déconnexion de machine et 51 seconde pour réaliser une migration vers un nouveau site cible de déploiement. Notons qu'au moment de l'adaptation, les agents de supervisions utilisent des messages syn-

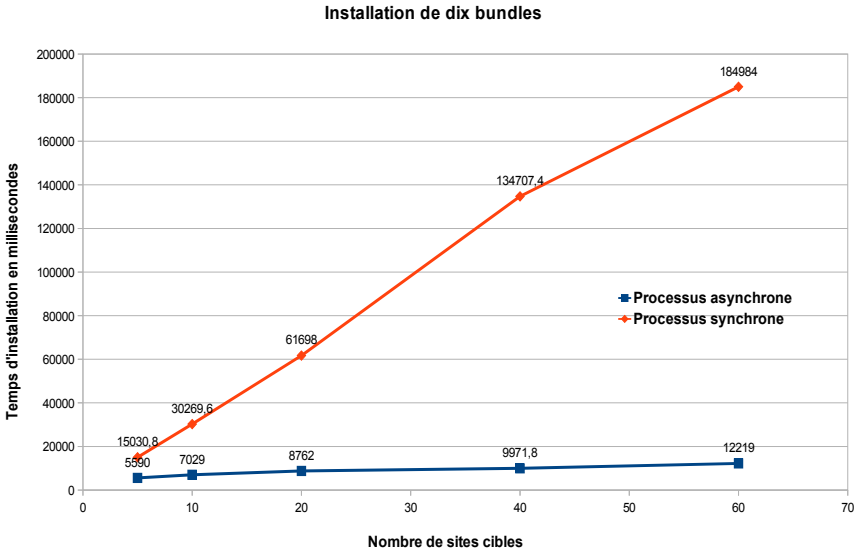

FIGURE 4.14 – Temps moyen de déploiement de dix bundles

chrones (pour éviter les messages perdus) jusqu'à la fin du processus d'adaptation. Nous remarquons, que le temps nécessaire pour réaliser une migration est légèrement supérieur au temps nécessaire pour corrigé une déconnexion. Les résultats obtenus restent très satisfaisants et donnent une idée sur le temps nécessaire pour réaliser une adaptation dynamique du processus de déploiement. Par ailleurs, des expérimentations avec un nombre plus conséquent de sites cibles permettraient de connaître d'une manière plus précise les limites de notre algorithme de déploiement.

4.4 Conclusion

Ce chapitre a présenté l'ensemble des études et des réalisations que nous avons faites pour implémenter, expérimenter et évaluer les performances de notre processus de déploiement. Nous avons d'abord présenté les détails de l'implémentation de notre prototype. Par la suite, nous avons rapporté les résultats d'évaluation des performances du système de découverte du réseau, du solveur de contraintes de déploiement et d'autre part les performances l'étape de déploiement du processus de déploiement.

Ces résultats nous ont permis de montrer d'une façon expérimentale la faisabilité et les capacités de notre intergiciel. Nous avons montré la pertinence de notre système de découverte du réseau et l'approche de génération automatique de plan

de déploiement. Cependant, une autre implémentation du système de découverte du réseau en utilisant d'autres technologies ainsi que l'utilisation d'un autre solveur de contraintes peut être pertinente afin de pouvoir comparer les résultats obtenus et améliorer le temps de déploiement.

Les performances de l'algorithme de déploiement, en particulier dans les phases de déploiement et de reconfiguration de déploiement ont montré la pertinence et l'originalité de notre approche. Par ailleurs, d'autres expérimentations sur le temps nécessaire pour l'adaptation du processus de déploiement sont nécessaires, d'une part l'augmentation de nombre de sites cibles utilisés, et d'autre part en intégrant d'autres types de terminaux comme les Smartphones et tablettes android. En effet, pour l'instant, il n'est pas possible de réaliser des opérations de migration d'un agent mobile à partir d'une machine virtuelle java standard vers une machine virtuelle java android ou l'inverse. Ce problème est purement technique, et sa résolution ne fait pas partie des objectifs de la thèse.

Chapitre 5

Conclusion et perspectives

Sommaire

5.1 Conclusion

Dans cette thèse, nous avons travaillé sur la problématique du déploiement autonomique de logiciels dans des environnements répartis ouverts. Cette problématique est d'actualité, surtout avec les avancées considérables réalisées tant au niveau des infrastructures réseaux, qu'au niveau des terminaux de déploiement (ordinateurs personnels, Smartphones, tablettes tactiles...).

Actuellement le déploiement d'applications réparties dans un environnement ouvert est réalisé principalement d'une manière manuelle ou semi-automatique, il nécessite généralement une intervention d'un opérateur humain (administrateur de déploiement) pour installer, configurer, mettre à jour, activer, désactiver, désinstaller, reconfigurer l'application et contrôler le processus de déploiement. Il existe un grand nombre d'outils, de procédures et d'articles traitant plusieurs aspects du processus de déploiement. Ces travaux ont pour objectif, de simplifier et d'automatiser certaines activités de déploiement et se basent essentiellement sur un modèle de contrôle centralisé pour contrôler le processus de déploiement. De plus, ils ne sont exploitables que sous certaines conditions comme la stabilité de l'architecture et l'absence de panne. Or dans les environnements répartis ouverts (ubiquitaires et P2P par exemple), ces différentes actions sont rendues difficiles par l'absence d'hypothèses sur la disponibilité du matériel.

L'objectif principal de ce travail de thèse est de produire un intergiciel pour le déploiement autonomique de logiciels à grande échelle. L'intergiciel a pour but, la

limitation des interventions humaines dans le processus de déploiement en offrant la possibilité de résoudre automatiquement les problèmes liés à l'instabilité et à l'ouverture de l'environnement. L'intergiciel en question est exploitable aussi bien dans des environnements répartis à grande échelle stable, comme les grilles de calcul que dans des environnements ouverts et instables comme les systèmes pervasifs et les systèmes P2P.

Pour réaliser cet intergiciel nous avons proposé une approche originale, basée sur les agents mobiles adaptables (AMA) pour superviser et exécuter le processus de déploiement. Grâce à leur autonomie, leurs proactivités et leurs facultés d'adaptation dynamique, les agents mobiles servent de support d'exécution et de supervision du processus de déploiement en permettant d'effectuer des adaptations et des reconfigurations dynamiques au moment de l'exécution sans aucune intervention de l'administrateur de déploiement.

Nous avons présenté dans le chapitre 3 notre approche pour le déploiement autonomique. L'intergiciel j-ASD, dédié au déploiement autonomique de logiciels en environnements répartis ouverts se base sur : (1) Un langage dédié (DSL) à la description des contraintes de déploiement. (2) Un service réseau pour découvrir d'une manière automatique les sites cibles de déploiement. (3) Un logiciel de bootstrap pour la préparation de l'environnement d'exécution. (4) Un solveur de contraintes pour la résolution des contraintes et la génération de plan de déploiement initial. (5) Un support de déploiement. (6) Un système d'agents mobiles adaptable pour l'exécution et la supervision des activités de déploiement. (7) Enfin, un algorithme de déploiement.

Nous estimons que j-ASD répond aux exigences et aux spécificités des environnements répartis ouverts. Les problèmes d'hétérogénéités matérielles sont résolus par l'utilisation du Framework OSGi comme support de déploiement. En effet OSGi, permet facilement le déploiement d'applications java composées d'un ou plusieurs bundles OSGi sur des équipements hétérogènes, comme les ordinateurs personnels, les smartphones, et les tablettes. Le service de découverte de réseau et le logiciel de bootstrap permettent respectivement la détection automatique des sites cibles et la gestion des problèmes d'administration multiples et de droits d'accès aux sites. Grâce au service de découverte de réseau, j-ASD dispose d'une vision globale sur la disponibilité des sites cibles à tout moment, ainsi il peut détecter les pannes et déconnexions d'un ou plusieurs sites cible de déploiement.

Le plan de déploiement initial est calculé à partir de la description des contraintes de déploiement déclarées par l'administrateur de déploiement en utilisant le langage dédié j-ASD DSL. Les contraintes sont automatiquement transformées vers un problème de satisfaction de contraintes qui est résolu par un solveur de contraintes adapté. La première solution consistante trouvée par le solveur est dynamiquement interprétée comme plan de déploiement initial par le système d'agents mobile.
Conformément à la propriété d'autonomie des agents logiciels, l'agent contrôle lui-

même ses propres évolutions et peut se déplacer de manière proactive sur le réseau, en transportant ses données, son code et son état d'exécution. L'utilisation d'agents nous a conduits à la fois à la décentralisation de contrôle du processus de déploiement. Nous considérons que, les agents constituent un outil privilégié pour l'adaptation du déploiement dans les environnements ouverts. L'algorithme de déploiement proposé comporte trois étapes, qui sont respectivement la vérification de la validité du plan de déploiement, l'exécution du plan du déploiement et enfin le contrôle et la reconfiguration du déploiement. Il permet aux agents mobiles de réaliser les activités de déploiement et de reconfiguration de façon autonome tout en gardant des interfaces d'échange avec l'administrateur du déploiement. Ces interfaces permettent à l'administrateur de consulter à tout moment l'état du processus de déploiement et d'intervenir pour corriger une situation si le système d'agents n'arrive pas à trouver une solution. La particularité de l'algorithme proposé est qu'il garantit la réalisation des opérations de reconfiguration de déploiement au moment du déploiement initial et après la réalisation du déploiement initial. De plus, il permet le passage à l'échelle grâce au système hiérarchique d'agents de supervision locale et globale.

Nous avons également implémenté et expérimenté notre solution, dans diverses configurations décrites au chapitre 4, afin de valider nos idées. Les expérimentations réalisées nous ont permis l'évaluation des performances du service de découverte du réseau, du plan de déploiement généré et l'algorithme de déploiement dans un environnement réel.

Le travail réalisé pendant cette thèse a été valorisé dans 3 publications : deux conférences internationales et un atelier francophone, toutes avec relecture et comité de sélection.

1. Vers un environnement de déploiement autonomique, présenté à Ubimob'2011. [Matougui 2011]
2. j-ASD : un middleware pour le déploiement logiciel autonomique, présenté à NOTERE/CFIP'12 [Matougui 2012b]
3. A middleware architecture for autonomic software deployment, présenté à ICSNC'12 [Matougui 2012a]

5.2 Perspectives

Le travail que nous avons réalisé dans cette thèse a permis, d'un côté de fournir une analyse relative à la problématique du déploiement autonomique de logiciels dans des environnements ouverts, et d'un autre côté de proposer une solution pour déployer des applications d'une manière autonomique en proposant des mécanismes adaptés pour résoudre automatiquement les problèmes liés à l'instabilité et à l'ouverture de l'environnement.

Cette thèse a permis de valider la pertinence et la faisabilité d'un environnement de déploiement autonomique, ce qui a servi de base à l'élaboration d'un projet ANR, INCOME [1], actuellement en cours. Une doctorante reprend les concepts proposés dans cette thèse afin de les étendre et de les appliquer au déploiement de systèmes répartis multi-échelle.

Ce travail de thèse ouvre plusieurs perspectives :

5.2.1 Contraintes et plan de déploiement

La première perspective de recherche concerne les contraintes et le plan de déploiement. Dans notre approche nous avons proposé un langage de description de contraintes de déploiement, présenté dans le chapitre 3, qui permet de décrire le logiciel à déployer ainsi que les contraintes d'installation. L'intégration de nouvelles contraintes comme les contraintes d'activation, de désactivation, de mise à jour et de désinstallation est nécessaire pour pouvoir couvrir toutes les activités de déploiement.

Les contraintes d'activation / désactivation peuvent être des contraintes de type relation de précédences, comme :

- **Start / Stop** Composant, pour l'activation / désactivation d'un composant.
- **Start / Stop** ComposantA **After** ComposantB, pour activer / arrêter le ComposantA après le ComposantB.
- **Start / Stop** ComposantC **Before** ComposantD, pour activer / arrêter le composantC avant le composantD.

Pour exprimer le fait d'activer / arrêter les composants Ci, Cj, ..., Ck avant le composant C, on peut utiliser plusieurs expressions du type activer / arrêter le composant Ci avant C. Notons qu'il faut prévoir des mécanismes internes au niveau du compilateur pour permettre la détection et le support de la transitivité entre les relations d'ordre d'activation / désactivation et la génération d'un plan d'activation / désactivation. Le plan d'activation / désactivation, peut être simplement un plan de messages d'activation / désactivation, qui vont être envoyés aux agents de supervision locale ou aux agents de déploiement.

Un travail plus approfondi sur la génération du plan de déploiement initial est nécessaire, pour pouvoir trouver le meilleur plan de déploiement. Cela peut se réaliser via l'exploitation des options et méthodes de résolution du solveur de contraintes CHOCO, ou en utilisant un autre solveur de contraintes pour pouvoir comparer les résultats obtenues.

1. Le projet INCOME (INfrastructure de gestion de COntexte Multi-Échelle pour l'Internet des Objets) est financé par l'ANR (Agence Nationale de la Recherche) dans le cadre du programme Infrastructures matérielles et logicielles pour la société numérique, édition 2011. http://anr-income.fr

Les contraintes de mise à jour peuvent être des contraintes de mise à jour dynamique (mise à jour sans arrêter l'exécution de l'application) ou bien des contraintes de mise à jour statique (mise à jour après avoir arrêté l'exécution de l'application). Les contraintes de mise à jour et de désinstallation peuvent être définies comme contraintes par défaut du système de déploiement fixé par l'administrateur de déploiement et ne nécessitant pas la génération d'un plan de mise à jour ou désinstallation. Elles comportent simplement un ensemble de contraintes qui doivent être respectées au moment de mise à jour et de désinstallation par le système d'agents mobiles.

5.2.2 Unité de déploiement

Nous avons proposé dans cette thèse une architecture logicielle, qui se base sur un support de déploiement existant pour la réalisation des activités de déploiement. Nous avons choisi le Framework OSGi comme support de déploiement de notre intergiciel, par conséquent l'unité physique de déploiement de notre intergiciel est un bundle OSGi.

Pour l'instant nous pouvons déployer uniquement des applications OSGi. L'expérimentation de nouvelles unités de déploiement comme les applications SCA [Open Service Oriented Architecture collaboration 2007] avec la plate-forme FraSCAti [Seinturier 2009, Seinturier 2012] comme support déploiement serait une extension naturelle de notre intergiciel. Cela permettrait le déploiement de nouvelles applications (pas forcément java) avec des niveaux de granularité différents.

5.2.3 Sécurité du processus de déploiement

Nous avons proposé un processus de déploiement dans lequel le déploiement de composants métiers est réalisé par des agents. Dans l'implémentation actuelle, les composants déployés dépendent du modèle de composants d'OSGi.

Du point de vue de la sécurité, le Framework OSGi dispose d'une couche de sécurité qui se base sur le **Java 2 Security Architecture specification** lequel permet de gérer et choisir le gestionnaire de sécurité, la politique de sécurité et les permissions des bundles installés. Il permet aussi de déterminer les fonctionnalités autorisées pour le bundles exécutés sur le Framework OSGi.

D'un autre côté, l'utilisation des agents mobiles présente quelques préoccupations liées à la sécurité du système. Dans un cadre général, un agent mobile est un programme autonome, se déplaçant sur le réseau avec son code et ses données et s'exécute sur une plate-forme d'accueil lui fournissant les ressources nécessaires à son exécution. De plus, ses déplacements et ses communications sont véhiculés par le réseau. Par conséquent, les entités qui peuvent subir des problèmes de sécurité

sont : l'agent mobile, la plate-forme d'accueil et les informations transmises par le réseau.

En effet, le système peut subir des attaques classiques sur le réseau comme l'écoute, le vol d'informations et l'interception de communications, ce qui peut toucher les communications entre les agents et peut porter atteinte aux agents eux-mêmes lors de leurs déplacements sur le réseau (le code des agents et leurs données étant sérialisés et transmis à la volée). La mise en place d'un système de chiffrement des messages, des données et les codes des agents est nécessaire pour assurer la sécurité du système de déploiement. En plus, le système peut aussi subir des attaques sur le système d'accueil des agents qui permet d'exécuter les agents en fournissant les ressources adaptées. Cette menace est limitée par la transformation du système d'accueil en bundle exécuté sur le Framework OSGi.

Des algorithmes d'adaptation dynamique du déploiement pour traiter les défaillances liées à la sécurité du système sont nécessaires pour améliorer le degré d'autonomie de notre système de déploiement.

5.2.4 Expérimentation

Les expérimentations actuelles du prototype, ont été réalisées sur un nombre assez limité de sites cibles de déploiement. Il serait intéressant d'évaluer les performances du service de découverte de réseau, du système de résolution des contraintes et de l'algorithme de déploiement avec un nombre très grand de sites cibles de déploiement avec l'intégration de nouveaux types de terminaux comme les IPhone, les tablettes android et les machines virtuelles dans le Cloud.

Dans le même contexte, des expérimentations qui permettent de répondre aux questions : jusqu'où peut-on aller en terme de reconfiguration et d'adaptation dynamique ? et jusqu'où peut-on aller en terme d'utilisation des agents pour la gestion du déploiement ? sont nécessaires pour déterminer d'une manière précise les limites de notre système de déploiement afin de pouvoir proposer de nouvelles politiques d'adaptation dynamique.

Enfin, la mise en place d'une évaluation basée sur la perception de l'utilisateur final est envisageable après la mise en place d'une interface graphique plus conséquente à l'intergiciel j-ASD.

Annexes

Annexe A

Technologies de découverte des sites cibles

Cette annexe regroupe une description des technologies candidates pour l'implémentation du mécanisme de découverte des sites cibles, ainsi que les discussions sur leur pertinence par rapport à nos besoins.

A.1 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) [UPnP Forum 2008] est une technique promue par le UPnP Forum, elle est constituée d'un ensemble de protocoles réseau destiné à être utilisé dans les réseaux locaux (spécialement dans les réseaux domestiques). Elle permet aux périphériques (ordinateur personnel, imprimante, points d'accès WiFi, les appareils mobiles) de découvrir d'une manière transparente la présence des autres équipements sur le réseau et d'établir des services fonctionnels comme le partage de données et les communications.

L'architecture d'UPnP permet la mise en réseau d'ordinateurs personnels, d'équipements réseaux et de périphériques sans fil. C'est une architecture distribuée basée sur les protocoles TCP, IP, UDP, HTTP et XML. UPnP permet à des clients appelés aussi des points de contrôle, de découvrir ou rechercher des devices (serveur UPnP sur les devices) dans un réseau local par système de diffusion. La recherche peut contenir des filtres tels qu'un type de dispositif (device) ou un type de service. UPnP ne se limite pas au problème de découverte de dispositifs, il fournit aussi les protocoles pour communiquer avec les dispositifs.

Les éléments de base dans UPnP sont, les équipements (**Devices**), les **services** et les points de contrôle (**contrôle points** ou PC).

Un Device UPnP est un élément qui contient des services et éventuellement d'autres Devices imbriqués. Un Device se comporte comme un serveur et il répond aux requêtes des points de contrôle.

Un service est la plus petite unité de contrôle dans un réseau UPnP, il propose des actions et son état est modélisé grâce à ses variables d'état. Il se compose d'une

table d'état qui modélise l'état d'un service par des variables d'état, un serveur de commande qui reçoit les demandes d'actions et qui exécute, met à jour la table et renvoi les réponses, et un serveur d'événements qui envoie des événements aux abonnés intéressés lorsque l'état du service change. Les actions sont des fonctions invocables. Les variables d'état sont des paramètres qui peuvent émettre des événements lorsque leur valeur change ou qui représentent des arguments de fonctions.

Le point de contrôle est un contrôleur capable de découvrir et de contrôler des Devices et d'utiliser leurs services. L'architecture d'un serveur est définie dans le fichier de description sous forme d'un fichier XML. Chaque service contient les cinq informations suivantes :

Le **serviceType** qui définit l'espace de nommage du service, le **serviceID** qui représente un nom unique du service, le **SCPDURL** qui représente l'URL pointant vers la description du service, le **conroleURL** qui représente l'URL avec laquelle communiquer pour les invocations d'actions et enfin le **eventSubURL** qui est une URL de souscription aux événements.

L'URL donnée par les serveurs dans les phases de recherche et de découverte pointe vers un fichier XML de description du serveur UPnP. Ce fichier contient des informations sur le constructeur de l'appareil ou du serveur UPnP tel que son nom, l'adresse de son site web, le nom et la version du dispositif et son numéro de série.

La récupération des fichiers de description des services fournit les informations sur les actions et les variables. En utilisant l'URL de contrôle (controlURL)on peut invoquer les actions et récupérer les valeurs des variables.

Un point de contrôle dans un réseau UPnP découvre et contrôle les autres devices. Une fois ces étapes effectuées, il peut récupérer la description du Device et la description des services, invoque les actions pour contrôler le service et souscrire aux événements d'un service.

La première étape du lancement d'un serveur UPnP est la phase d'adressage, elle permet à un nouveau dispositif d'obtenir une adresse et une configuration réseau. Pour connaitre l'existence d'un serveur UPnP deux moyens sont possible :

La recherche : les clients effectuent une méthode de recherche par diffusion (multicast ou broadcast). Les serveurs répondent alors seulement au client qui a effectué la recherche en indiquant les informations pour les contacter.

La découverte : les serveurs annoncent périodiquement leur présence par diffusion. Les clients découvrent alors les services et apprennent par cette annonce l'existence des dispositifs.

Lorsqu'un serveur UPnP est lancé, il annonce sa présence sur le réseau local par un ou plusieurs paquets multicast, qui sont définis par le SSDP (Simple Service Discovery Protocol) et GENA (General Event Notification Architecture). Cette

notification contient une URL qui pointe vers l'entité définie. Ce paquet est envoyé sur l'IP de multicast (239.255.255.250) sur le port 1900. Cette adresse et ce port sont réservés à UPnP par l'IANA (Internet Assigned Numbers Authority).

Ce message de présence comporte un temps maximal de validité, le temps au bout duquel l'existence du service ne sera plus garantie pour les points de contrôle. L'annonce sera réémise avant la fin de ce temps maximal, permettant aux nouveaux points de contrôle de connaitre l'existence du serveur. Lorsqu'un serveur UPnP s'arrête, il doit envoyer, dans la limite du possible, un ou plusieurs messages par diffusion qui indiquent aux points de contrôle que le serveur n'est plus disponible. Quand un point de contrôle découvre un serveur UPnP, il obtient une URL qui pointe vers un fichier de description du serveur. Dans ce fichier, il existe un champ qui correspond à l'URL de la page de présentation principale du dispositif.

A.2 Protocole SIP

SIP [Sparks 2007] (Session Initiation Protocol), est un protocole de signalisation défini par l'IETF (Internet Engineering Task Force), il est actuellement l'un des protocoles les plus utilisés pour la téléphonie par internet (VoIP). Il permet l'établissement, la modification et la terminaison des sessions multimédias telles que les communications téléphoniques par l'internet. Il permet aussi d'inviter des participants à des sessions déjà existantes, telles que les conférences en multi-diffusions. Le protocole SIP permet de prendre en charge d'une marnière transparente la transposition de noms et les services de redirection, ce qui sert de support à la mobilité personnelle. Les utilisateurs peuvent conserver un identifiant unique vu de l'extérieur, indépendamment de leur localisation dans le réseau.

SIP prend en charge cinq facettes de l'établissement et de la terminaison de communications multimédias, les cinq facettes sont :

- La localisation de l'utilisateur.
- La disponibilité de l'utilisateur.
- Les capacités de l'utilisateur.
- L'établissement de session.
- La gestion de session (le transfert et la terminaison de session, la modification des paramètres de sessions et l'invocation des services).

Le protocole SIP ne fournit pas des services, il fournit des primitives qui peuvent être utilisées pour mettre en œuvre différents services. Par exemple, SIP peut être utilisé pour initialiser une session qui utilise un autre protocole de contrôle de conférence. Parmi les primitives fournies par SIP on peut citer la méthode *INVITE* qui est utilisée pour demander l'établissement d'une session entre les agents utilisateurs, la méthode *ACK* qui est une méthode utilisée pour la confirmation d'établissement

de la session, *CANCEL* qui permet d'annuler un *INVITE* et BYE qui permet de terminer une session en cours.

SIP hérite de certaines fonctionnalités des protocoles HTTP et SMTP, il utilise par exemple HTTP pour naviguer sur le Web et SMTP pour la transmission des messages électroniques.

SIP s'appuie sur un modèle transactionnel (client/serveur) ; l'adressage utilise le concept d'URL SIP (Uniform Resources Locator) qui ressemble à une adresse électronique.

Chaque participant dans un réseau SIP est donc adressable par une URL SIP. Notons qu'il existe une version sécurisée de l'adressage noté URL SIPs.

SIP définit deux types d'entités (les clients et les serveurs), il définit un serveur proxy, un serveur de redirection, l'agent utilisateur et l'enregistreur. Le serveur proxy permet le traitement ou l'acheminement vers d'autres serveurs des requêtes de clients. Le serveur de redirection permet de traiter (accepter) des requêtes SIP, pour cela il traduit l'adresse SIP de destination en une ou plusieurs adresses réseau et les retourne au client.

L'agent utilisateur représente une application sur un équipement de l'utilisateur qui permet l'émission et la réception des requêtes SIP. L'enregistreur est un serveur qui accepte les requêtes SIP REGISTER. Le protocole SIP dispose d'une primitive d'enregistrement d'utilisateurs, pour cela l'utilisateur indique par une requête REGISTER émise à l'enregistreur, l'adresse où il est joignable. L'enregistreur par la suite met à jour la base de données de localisation.

Le protocole SIP comporte des avantages et des inconvénients, parmi les avantages on peut citer :

- SIP est un protocole ouvert dont les protocoles et les documents officiels sont détaillés et accessibles à tout le monde.
- SIP est un protocole normalisé par l'IETF.
- Le protocole SIP permet le passage à l'échelle et fonctionne en mode P2P, il est utilisé pour tout type de sessions multimédias (voix, vidéo,...).

Parmi les inconvénients de SIP on peut citer,

- Le protocole SIP comporte certaines latences dans la gestion de la présence et la messagerie instantanée.
- SIP ne bénéficie pas de l'effet réseau, par le fait qu'il n'est pas connu et utilisé par le grand public.

A.2.1 Protocole XMPP

XMPP [Saint-Andre 2009] est un protocole de messagerie extensible et de présence, il consiste en un ensemble de techniques ouvertes. Il est utilisé principalement pour la gestion de messagerie instantanée, la présence, le chat multi-parties, appels audio et vidéo, et la généralisation de routage de données XML. XMPP utilise le langage XML comme format de base pour l'échange d'informations, il fournit des moyens pour envoyer des petites pièces d'XML d'une entité à une autre entité en temps réel dans le réseau Jabber.

Ce protocole a été initialement développé dans le cadre de la communauté open-source Jabber pour fournir une plate-forme ouverte, sécurisée, et décentralisée pour les services de messagerie instantanée. XMPP fournit plusieurs services dont les principaux services sont :

- Le service de cryptage de connexion entre les clients et les serveurs (ou entre les serveurs).
- Le service d'authentification des entités prêtes pour la communication dans le réseau par les serveurs.
- Le service de présence, qui permet la détection des entités connectées et prêtes à communiquer dans le réseau.
- Le service de liste de contacts, qui permet d'enregistrer une liste de contacts.
- Les services d'envoi de message (d'une entité à une autre ou d'une entité à plusieurs autres entités).
- Le service de notification, qui permet la génération des notifications et assure sa délivrance pour plusieurs destinations.
- Le service de découverte, qui permet de savoir quelles sont les fonctionnalités supportées par une autre entité.
- Le service peer-to-peer media sessions, qui permet la négociation et le management d'une session avec une autre entité.

Les services fournis par XMPP sont utilisés dans plusieurs types d'applications telles que la messagerie instantanée, les applications de chat, les applications VoIP et les jeux vidéo. Le protocole XMPP est utilisé par d'autres types d'applications telles que les systèmes de contrôle, la géo-localisation et même des intergiciels ou le Cloud.

Le protocole XMPP utilise une architecture décentralisée de type client/serveur qui comporte quelques centaines de milliers de serveurs Jabber (ejabberd, Openfire,...) et plusieurs millions de clients Jabber (Adium, Gajim, Pigdgin,...).

Chaque entitée XMPP, dans le réseau, a besoin d'une adresse qui est appelée JabberID (JID). Elle constitue l'identifiant d'un utilisateur dans le réseau Jabber, est est composée d'un nom d'utilisateur unique, d'un nom de serveur et d'une ressource optionnelle. Un JID est représenté sous la forme suivante :

`utilisateur@serveur/ressources`, par exemple matougui@jabberfr.org avec matougui qui est le nom d'utilisateur et jabberfr.org qui est le serveur jabber utilisé.

Comme nous l'avons déjà cité, le protocole XMPP utilise un service de signalisation de présence ; ce service permet la signalisation de présence et/ou déconnexions lorsqu'un client Jabber se connecte ou se déconnecte à un serveur, ce dernier annonce automatiquement la connexion (la présence) ou la déconnexion de l'utilisateur à l'ensemble des membres de la liste des contacts. Il est à noter qu'il existe plusieurs types d'état de présence en ligne, déconnecté et absent. L'objectif du service de découverte du réseau est d'exploiter ces services de gestion de présence pour la réalisation d'une découverte de sites cibles de déploiement étendue. Nous présentons plus de détails dans le chapitre suivant.

Comme tous les protocoles XMPP, comporte des avantages et des inconvénients, parmi les avantages on peut citer :

- XMPP est un protocole ouvert ; en fait l'ensemble des protocoles et les documents officiels sont détaillés et accessibles à tous en téléchargement.
- XMMP est un protocole normalisé par l'IETF et son évolution continue par ses extensions avec la XMPP Standards Fondation.
- XMPP est un protocole décentralisé, en effet les services Jabber ne dépendent pas d'un seul point d'accès.
- Dans le protocole XMPP les communications entre le client et le serveur peuvent être chiffrées à la demande du client.
- XMPP est un protocole extensible et flexible, en effet il est possible de créer et documenter des extensions du protocole via les XEP, il est possible aussi d'utiliser le protocole dans plusieurs types d'applications y compris notre service de découverte du réseau qu'on expliquera dans la suite de cette section.
- L'implémentation du protocole XMPP est adoptée par des grands noms tels que Google, IBM, Sun et France Telecom.

Les inconvénients du protocole XMPP sont des limites techniques liées aux implémentations des serveurs et des clients du protocole comme le nombre de messages inutiles (70% des messages sont pour la présence). Pour notre cas d'utilisation du protocole XMPP cela est réduit par l'implémentation légère de nos propres clients XMPP.

A.3 Protocole SLP

Le Service Location Protocol [Guttman 1999] est un protocole de découverte de services qui permet aux ordinateurs et autres types dispositifs de trouver des services dans un réseau local sans aucune configuration préalable. Ce protocole est utilisé par les équipements pour annoncer des services dans un réseau local. Chaque service doit avoir une URL qui permet la localisation du service. Les applications

sont sous la forme de clients SLP qui ont besoin de trouver des serveurs dans un réseau local à l'échelle d'un réseau d'entreprise.

L'architecture du protocole SLP comporte trois principales entités,
- La première entité est le User Agent (UA), il représente un processus permettant la recherche de services.
- La deuxième entité est le Service Agent (SA), qui représente un processus qui permet d'annoncer un ou plusieurs services.
- Enfin, le Directory Agent (DA) est une entité utilisée par SLP pour la collecte des adresses des services et les informations en provenance des Service Agents et leurs mises en place dans la base de données et pour répondre aux demandes de services des User Agents.

Quand un nouveau service se connecte à un réseau, le SA contacte le DA pour annoncer son existence (Enregistrement de service). Quand un utilisateur a besoin d'un service, l'UA demande au DA les services disponibles dans le réseau. Après la réception de l'adresse et les caractéristiques du service souhaité, l'utilisateur peut enfin utiliser le service demandé.

Avant qu'un client (UA ou SA) ne soit capable de communiquer avec le DA, il doit découvrir l'existence de DA. Il existe trois méthodes différentes pour la découverte de DA, la méthode statique, la méthode active et la méthode passive. Dans la découverte statique les agents SLP obtiennent l'adresse du DA via DHCP. Le serveur DHCP fournie l'adresse du DA aux clients qui le demande. Dans la découverte active les UAs et SAs envoient les requêtes de services à l'adresse multicast du SLP (239.255.255.253). Un DA est à l'écoute sur cette adresse et pourra éventuellement recevoir une demande de service et répondre directement via unicast à l'agent demandeur.

En cas de découverte passive, l'AD envoie périodiquement des annonces de services. Les UAs et SAs apprennent l'adresse du DA par une annonce reçue et seront en mesure de contacter le DA.

Les services sont annoncés en utilisant une URL de service et une Template de service. L'URL du service contient l'adresse IP du service, le numéro de port et le chemin. La Template de service spécifie les attributs qui caractérisent le service et leurs valeurs par défaut.

A.4 Discussion

Dans cette Section, nous avons présenté quatre protocoles et technologies qui traitent la gestion de sessions multimédia et de signalisation de présence à grande échelle (cas du protocole XMPP et le protocole SIP), la découverte de services dans un réseau local (cas du protocole SLP) et UPnP qui est constituée d'un ensemble

de protocoles destinés à être utilisé dans un réseau local. UPnP permet aux périphériques connectés à un réseau local de découvrir d'une manière transparente la présence des autres équipements sur le réseau et d'établir des services fonctionnels.

Les deux premiers protocoles (XMPP, SIP) permettent l'établissement, la modification et la terminaison des sessions multimédia telles que les communications VoIP. En plus des mécanismes de gestion de sessions multimédia, ils permettent aussi la localisation de l'utilisateur en utilisant des méthodes de signalisation de présence des clients dans leurs réseaux respectifs (le réseau SIP et le réseau Jabber d'XMPP).

Les deux protocoles se basent sur une architecture de type client/serveur dans laquelle un client doit se connecter à un serveur pour pouvoir signaler sa présence aux autres clients et avoir accès aux différents supports et services de gestion de session multimédias. Le protocole SIP ne fournit pas des services mais il fournit des primitives qui peuvent être utilisées pour mettre en œuvre les services (la méthode INVITE et CANCEL par exemple). Par contre le protocole XMPP fournit des services tels que l'authentification et la présence.

Parmi les éléments qu'on a pu ressortir de cette étude de protocoles est qu'il faut distinguer entre trois cas de figures possibles. Le premier cas correspond au déploiement d'une application dans un ensemble de sites cibles connectés à un réseau local. Le deuxième cas de figure consiste au déploiement d'une application à grande échelle. Le troisième cas de figure est le déploiement d'un logiciel dans un environnement mutlti-échelle composé d'un ou plusieurs réseaux locaux et réseaux à grande échelle.

Bibliographie

[Agha 1986] Gul A. Agha. ACTORS : A model of concurrent computation in distributed systems. MIT Press, Cambridge, Massachusetts, 1986. (Cité en pages 66 et 94.)

[Alliot 2002] J.M. Alliot, T. Schiex, P. Brisset and F. Garcia. Intelligence artificielle & informatique théorique. Cépaduès-éd., 2002. (Cité en page 67.)

[APPLE 2009] APPLE. *Bonjour protocol specifications.*, 2009. (Cité en page 89.)

[Arcangeli 2001] J.-P. Arcangeli, C. Maurel and F. Migeon. *An API for high-level software engineering of distributed and mobile applications.* In Distributed Computing Systems, 2001. FTDCS 2001. Proceedings. The Eighth IEEE Workshop on Future Trends of, pages 155 –161, 2001. (Cité en page 113.)

[Baduel 2006] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel and Romain Quilici. *Programming, Composing, Deploying for the Grid.* In Omer F. Cunha Jose C. ; Rana, editeur, Grid Computing : Software Environments and Tools, pages 205 – 229. Springer, January 2006. (Cité en pages 35 et 53.)

[Bailey 2000] Edward C. Bailey. Maximum rpm. SAMS Publishing, 2000. (Cité en pages 34, 35, 47 et 74.)

[Bernard 2002] Guy Bernard and Leila Ismail. *Apport des agents mobiles à l 'exécution répartie.* Technique et Science Informatiques, vol. 21, no. 6, pages 771–796, 2002. (Cité en page 68.)

[Bradshaw 1997] Jeffrey M. Bradshaw, editeur. Software agents. MIT Press, Cambridge, MA, USA, 1997. (Cité en page 66.)

[Bruneton 2004] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma and Jean-Bernard Stefani. *An Open Component Model and Its Support in Java.* In CBSE, 2004. (Cité en page 17.)

[Burke 2006] Bill Burke and Richard Monson-Haefel. Enterprise javabeans 3.0 (5th edition). O'Reilly Media, Inc., Mai 2006. (Cité en pages 35 et 41.)

[Chess 1994] David Chess, Colin Harrison and Aaron Kershenbaum. *Mobile Agents : Are They a Good Idea ?* RC 19887, IBM, Yorktown Heights, New York, USA, 1994. (December 21, 1994 - Declassified March 16, 1995). (Cité en page 68.)

[CHOCO Team 2010] CHOCO Team. *choco : an Open Source Java Constraint Programming Library.* Research report 10-02-INFO, Ecole des Mines de Nantes, 2010. (Cité en page 109.)

[Conan 2007] Denis Conan, Romain Rouvoy and Lionel Seinturier. *Scalable Processing of Context Information with COSMOS.* In DAIS, pages 210–224, 2007. (Cité en page 17.)

[Cunin 2005] Pierre-Yves Cunin, Vincent Lestideau and Noëlle Merle. *ORYA : A Strategy Oriented Deployment Framework*. In Component Deployment, pages 177–180, 2005. (Cité en pages 35 et 55.)

[De Palma 2008] Noël De Palma, Sara Bouchenak, Fabienne Boyer, Daniel Hagimont, Sylvain Sicard and Christophe Taton. *Jade, un environnement d'administration autonome.* Technique et Science Informatiques, vol. 27, no. 9-10, pages 1225–1252, 2008. (Cité en pages 35 et 58.)

[Dearle 2004] Alan Dearle, Graham N. C. Kirby and Andrew J. McCarthy. *A Framework for Constraint-Based Deployment and Autonomic Management of Distributed Applications*. In ICAC, pages 300–301, 2004. (Cité en page 79.)

[Dearle 2007] Alan Dearle. *Software Deployment, Past, Present and Future*. In FOSE, pages 269–284, 2007. (Cité en page 30.)

[Dearle 2010] Alan Dearle, Graham N. C. Kirby and Andrew McCarthy. *A Framework for Constraint-Based Deployment and Autonomic Management of Distributed Applications*. CoRR, vol. abs/1006.4572, 2010. (Cité en pages 35, 60 et 79.)

[Dejan 1999] Milojivcicacute Dejan, Douglis Frederick and Wheeler Richard, editeurs. Mobility : processes, computers, and agents. New York, NY, USA, 1999. (Cité en page 67.)

[Ferber 1995] Jacques Ferber. Les systèmes multi-agents. vers une intelligence collective. InterEditions, 1995. (Cité en page 67.)

[Ferber 1999] Jacques Ferber. Multi-agent systems - an introduction to distributed artificial intelligence. Addison-Wesley-Longman, 1999. (Cité en page 66.)

[Filho 2000] Roberto Silveira Silva Filho, Roberto Silveira and Silva Filho. *Mobile Agents and Software Deployment*, 2000. (Cité en page 66.)

[Flissi 2008] Areski Flissi, Jérémy Dubus, Nicolas Dolet and Philippe Merle. *Deploying on the Grid with DeployWare*. In CCGRID, pages 177–184, 2008. (Cité en pages 21, 35, 48 et 75.)

[Fuggetta 1998] Alfonso Fuggetta, Gian Pietro Picco and Giovanni Vigna. *Understanding Code Mobility*. IEEE Transactions on Software Engineering, vol. 24, no. 5, pages 342–361, 1998. (Cité en page 68.)

[Guttman 1999] Erik Guttman. *Service Location Protocol : Automatic Discovery of IP Network Services*. IEEE Internet Computing, 1999. (Cité en pages 89 et 140.)

[Hall 1999] Richard S. Hall, Dennis Heimbigner and Alexander L. Wolf. *A cooperative approach to support software deployment using the software dock*. In Proceedings of the 21st international conference on Software engineering, ICSE '99, pages 174–183. ACM, 1999. (Cité en pages 30, 31, 32, 50 et 94.)

[Harrison 1995] Colin G. Harrison, Colin G. Harrison, David M. Chess, David M. Chess, Aaron Kershenbaum and Aaron Kershenbaum. *Mobile Agents : Are they a good idea ?*, 1995. (Cité en page 68.)

[Hashimi 2006] Y. Hashimi. Deploying .NET Applications : Learning MSBuild and ClickOnce. The expert's voice in .NET. Apress, 2006. (Cité en page 43.)

[Hewitt 1977] Carl E. Hewitt. *Viewing Controll Structures as Patterns of Passing Messages*. Journal of Artificial Intelligence, vol. 8, no. 3, pages 323–364, 1977. (Cité en page 66.)

[Horn 2001] Paul Horn. Autonomic computing˜ - IBM's Perspective on the State of Information Technology. 2001. (Cité en page 28.)

[Kephart 2003] Jeffrey O. Kephart and David M. Chess. *The Vision of Autonomic Computing*. Computer, vol. 36, no. 1, pages 41–50, 2003. (Cité en pages 28 et 29.)

[Leriche 2006] Sébastien Leriche. *Architectures à composants et agents pour la conception d'applications réparties adaptables*. Thèse de doctorat, Université Paul Sabatier, 2006. (Cité en page 69.)

[Leriche 2010] Sébastien Leriche and Jean-Paul Arcangeli. *Flexible architectures of adaptive agents : the agentϕ approach*. International journal of grid computing and multi agent systems (IJGCMAS), vol. 1, no. 1, pages 55–75, Janvier 2010. 8878. (Cité en page 69.)

[Lesser 1995] Victor R. Lesser and Les Gasser, editeurs. Proceedings of the first international conference on multiagent systems, june 12-14, 1995, san francisco, california, usa. The MIT Press, 1995. (Cité en page 67.)

[Louberry 2011] Christine Louberry, Philippe Roose and Marc Dalmau. *Kalimucho : Contextual Deployment for QoS Management*. In 11th IFIP WG 6.1 International Conference, DAIS 2011, Reykjavik, Iceland, June 2011, Proceedings, volume 6723 of *Lecture Notes in Computer Science*, pages pp.43–56, Reykjavik, Islande, Juin 2011. Springer. (Cité en pages 35 et 56.)

[Matougui 2011] Mohamed El Amine Matougui and Sébastien Leriche. *Vers un environnement de déploiement autonomique*. In Ubimob'11 : 7es journSées francophones MobilitSé et ubiquitSé, pages 57–62, Toulouse, France, 2011. IRIT. (Cité en page 129.)

[Matougui 2012a] Mohamed El Amine Matougui and Sébastien Leriche. *A middleware architecture for autonomic software deployment*. In ICSNC '12 : The Seventh International Conference on Systems and Networks Communications, pages 13–20, Lisbon, Portugal, 2012. XPS. 12619 12619. (Cité en page 129.)

[Matougui 2012b] Mohamed El Amine Matougui and Sébastien Leriche. *j-ASD : un middleware pour le déploiement logiciel autonomique*. In NOTERE/CFIP '12 : Conférence Internationale Nouvelles Technologies de la Répartition/-Colloque Francophone sur l'Ingénierie des Protocoles, page ., Anglet, France, 2012. Cépaduès. 12618 12618. (Cité en page 129.)

[Merle 2005] Noelle Merle. *Architecture pour les systèmes de déploiement logiciel à grande échelle : prise en compte des concepts d'entreprise et de stratégie*.

PhD thesis, Université Joseph-Fourier-Grenoble I, 2005. (Cité en pages 35 et 55.)

[Microsoft 1999] Microsoft. *Component Object Model (COM)*. `http://www.microsoft.com/com/default.asp`, Janvier 1999. (Cité en pages 35 et 43.)

[Object Management Group 2006] Object Management Group. *CORBA Component Model 4.0 Specification*. Specification Version 4.0, Object Management Group, April 2006. (Cité en pages 35 et 37.)

[Open Service Oriented Architecture collaboration 2007] Open Service Oriented Architecture collaboration. Power combination : Sca, osgi and spring introductory whitepaper. 2007. (Cité en page 131.)

[Quitadamo 2008] Raffaele Quitadamo, Giacomo Cabri and Letizia Leonardi. *Mobile JikesRVM : A framework to support transparent Java thread migration*. Sci. Comput. Program., vol. 70, no. 2-3, pages 221–240, 2008. (Cité en page 68.)

[Ribault 2010] Judicaël Ribault, Olivier Dalle, Denis Conan and Sébastien Leriche. *OSIF : a framework to instrument, validate, and analyze simulations*. In Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, SIMUTools '10, pages 56 :1–56 :9, ICST, Brussels, Belgium, Belgium, 2010. ICST. (Cité en page 21.)

[Rouvoy 2008] Romain Rouvoy, Denis Conan and Lionel Seinturier. *Software Architecture Patterns for a Context-Processing Middleware Framework*. IEEE Distributed Systems Online, vol. 9, no. 6, page 1, 2008. (Cité en page 17.)

[Saint-Andre 2009] Peter Saint-Andre, Kevin Smith and Remko Tronçon. XMPP : The Definitive Guide : Building Real-Time Applications with Jabber Technologies. O'Reilly Media, Inc., 2009. (Cité en pages 89 et 139.)

[Seinturier 2009] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni and Jean-Bernard Stefani. *Reconfigurable SCA Applications with the FraSCAti Platform*. In Proceedings of the 2009 IEEE International Conference on Services Computing, SCC '09, pages 268–275, Washington, DC, USA, 2009. IEEE Computer Society. (Cité en page 131.)

[Seinturier 2012] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni and Jean-Bernard Stefani. *A component-based middleware platform for reconfigurable service-oriented architectures*. Softw. Pract. Exper., vol. 42, no. 5, pages 559–583, Mai 2012. (Cité en page 131.)

[Sparks 2007] Robert Sparks. *SIP : Basics and Beyond*. Queue, pages 22–33, March 2007. (Cité en pages 89 et 137.)

[Specification 2006] OMG Available Specification. *Deployment and Configuration of Component-based Distributed Applications Specification*, 2006. (Cité en pages 35, 38 et 75.)

[Strembeck 2009] Mark Strembeck and Uwe Zdun. *An approach for the systematic development of domain-specific languages*. Software : Practice and Experience, vol. 39, no. 15, pages 1253–1292, 2009. (Cité en page 79.)

[Szyperski 2002] C. Szyperski. Component software, beyond object-oriented programming. 2nd Edition. Addison Wesley Professional, 2002. (Cité en page 34.)

[Tempich 2003] Christoph Tempich, Julien Tane, Steffen Staab, Marc Ehrig and Christoph Schmitz. *Towards Evaluation of Peer-to-Peer-based Distributed Information Management Systems.* pages 73–88, 2003. (Cité en page 114.)

[The OSGi Alliance 2009] The OSGi Alliance. *OSGi Service Platform Core Specification, Release 3. Version 4.2*, 2009. (Cité en pages 35 et 45.)

[Tolvanen 2010] Juha-Pekka Tolvanen and Steven Kelly. *Integrating models with domain-specific modeling languages.* In Proceedings of the 10th Workshop on Domain-Specific Modeling, DSM '10, pages 10 :1–10 :6, New York, NY, USA, 2010. ACM. (Cité en page 79.)

[UPnP Forum 2008] UPnP Forum. *UPnP Device Architecture*, 2008. (Cité en pages 89 et 135.)

[van der Hoek 1998] Andre van der Hoek, Richard S Hall, Antonio Carzaniga, Dennis Heimbigner and Alexander L Wolf. *Software deployment : Extending configuration management support into the field.* Crosstalk, The Journal of Defense Software Engineering, vol. 11, no. 2, 1998. (Cité en pages 13, 30, 31 et 32.)