j-ASD : un middleware pour le déploiement logiciel autonomique

Mohamed El Amine Matougui, Sébastien Leriche

j-ASD : Un middleware pour le déploiement logiciel autonomique

Mohammed El Amine Matougui & Sébastien Leriche
Institut Telecom ; Telecom SudParis
UMR 5157 CNRS SAMOVAR,
F-91011 Evry Cedex, France
Email : {mohammed_el_amine.matougui, sebastien.leriche}@it-sudparis.eu

***Résumé*—Dans cet article nous nous intéressons à la problématique du déploiement autonomique de logiciel dans des infrastructures réparties à grande échelle à topologie variable, tel que les systèmes ubiquitaires et les systèmes pair-à-pair. Nous proposons j-ASD, un intergiciel pour l'automatisation du processus de déploiement. Le middleware est basé sur une spécification des contraintes de déploiement, qui sera résolue par la suite par un solveur de contraintes et un système d'agents mobiles adaptable pour l'exécution du plan de déploiement, la supervision et l'adaptation dynamique du processus de déploiement lors de l'exécution.**

Mots-clés

Intergiciel, déploiement autonomique, informatique ubiquitaire, agents mobiles.

Abstract

In this paper, we address the problem of autonomic software deployment in large-scale, distributed and heterogeneous infrastructures (ubiquitous and P2P systems). We propose j-ASD, a middleware for automating the deployment process. The middleware is based on, deployment constraints specifications, which are then resolved by a constraint solver to produce a context-dependent deployment plan ; and, a mobile-agent based approach for executing the produced deployment plan and subsequent runtime adaptations.

keywords

Middleware, autonomic deployment ; ubiquitous computing ; mobile agents

I. Introduction

Le déploiement de logiciel est un processus complexe qui comprend toutes les activités entre la production du logiciel et sa désinstallation des sites de déploiement [1]. Un cycle de vie de déploiement générique comprend l'installation, la désinstallation, l'activation, la désactivation, la mise à jour et la reconfiguration du logiciel [2].

Dans cet article nous nous intéressons au déploiement de logiciels dans des infrastructures réparties à grande échelle qui peuvent être dynamiques telle que les systèmes ubiquitaires, les systèmes P2P. Ces systèmes sont caractérisés par la mobilité des sites et des changements fréquents de la topologie du réseau. Ils sont aussi caractérisés par un grand nombre de machines hétérogènes dotées d'environnements matériels et logiciels différents. Plusieurs outils de déploiement à grande échelle existent, tels que software Dock [3], DeployWare [4], D&C [5] et plus récemment KALIMUCHO [6]. Ces outils sont généralement utilisables dans des topologies réseau fixe et ne prennent pas en compte les situations de déconnexions et les défaillances de machines (ou des liens du réseau) qui caractérisent les environnements ouverts.

Dans [7], nous avons proposé nos premières idées sur l'architecture de notre middleware de déploiement et deux scénarios de déploiement où les outils de déploiement logiciel actuels ne sont pas pertinents. Le premier scénario décrit le déploiement d'une application dans un environnement ouvert avec une topologie imprévisible et des machines qui ne sont pas connues au début du processus de déploiement. L'exemple consiste à exploiter le maximum d'ordinateurs et de téléphones portables connectés en wifi dans une salle de conférences et d'y déployer dynamiquement une application qui montre en temps réel des statistiques sur les machines disponibles. 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 à grande échelle. La première spécificité, consiste à la nécessité de l'utilisation d'un service de découverte de réseau. En effet, dans les systèmes ouverts, l'administrateur de déploiement ne connaît pas forcément à l'avance les hôtes cible de déploiement que nous devons détecter afin de pouvoir déployer notre logiciel. La seconde spécificité consiste à 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 logiciel. La dernière spécificité, consiste à prévoir des outils de reconfiguration dynamique et d'autoadaptation 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 qui peut répondre aux spécificités et problèmes de ces environnements doit être capable de :

1) détecter, gérer et accéder aux sites cibles d'une manière automatique.
2) gérer les hétérogénéités logicielles et matérielles des sites détectés.
3) fournir un moyen pour la déclaration des dépendances logicielles, les préférences matérielles et les contraintes de déploiement.
4) calculer un plan de déploiement qui satisfasse les contraintes de déploiement d'une manière automatique.
5) d'exécuter les activités de déploiement avec un minimum d'intervention humaine.
6) d'offrir des mécanismes d'adaptation automatique au moment de l'exécution pour prendre en charge les situations de panne de machines et des déconnexions.
7) s'exécuter dans des topologies dynamiques à grande échelle.

Ce papier est organisé comme suit : la section 2 présente l'architecture de notre middleware de déploiement autonomique de logiciel, j-ASD. La section 3, présente les différentes technologies employées dans notre prototype ainsi que l'état d'avancement de son implémentation. Dans la section 4, nous présentons quelques travaux connexes. Enfin, la section 5 conclue l'article et présente un aperçu de nos travaux futurs.

II. Architecture de j-ASD

Dans cette section, nous présentons l'architecture de notre middleware pour le déploiement autonomique de logiciel qui répond aux exigences présentées dans l'introduction. L'architecture proposée est illustrée dans la Figure 1. Le middleware est composé des éléments logiciels suivants :

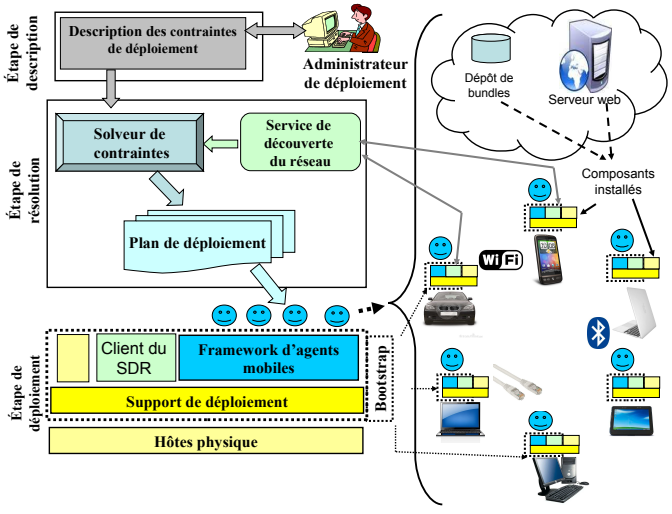

Figure 1. Architecture de j-ASD

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. Le DSL comporte aussi un parseur et un solveur de contraintes qui permet de calculer un plan de déploiement initial.

Un service de découverte du réseau pour permettre la détection automatique des hôtes cibles de déploiement dans le réseau. ouverts tels que les systèmes ubiquitaires sera possible. Un bootstrap qui permet la préparation de l'environnement d'exécution dans les sites cibles de déploiement. Il a pour but la résolution des problèmes d'administration multiples et des droits d'accès aux sites cibles de déploiement. dépendances logicielles de notre système de déploiement. Un support de déploiement équipé d'un environnement d'exécution, capable de s'exécuter dans des infrastructures matérielles hétérogènes et qui permet d'exécuter tout ou une partie des activités de déploiement.

L'intergiciel est doté d'une interface graphique qui permet aux administrateurs de déploiement à tout moment de vérifier l'état du processus de déploiement et d'intervenir dans une ou plusieurs activités de déploiement.

Enfin, un système d'agents mobiles adaptable prend en charge l'exécution et la supervision du processus de déploiement.

A. Langage de description des contraintes de déploiement

Configurer et déployer des logiciels à grande échelle est une tâche très complexe. La complexité est due à la multitude de composants qui forment le logiciel et l'hétérogénéité et le nombre important des sites cibles de déploiement. Afin d'automatiser le processus de déploiement de logiciels, il est nécessaire d'avoir un certain nombre d'informations sous la forme d'une description des contraintes de déploiement. Le but de cette description est de fournir des informations suffisantes sur le logiciel pour être en mesure de le déployer d'une manière autonomique. 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 XML (D& C et CORBA) et les langages dédiés (DSL).

Notre approche consiste à l'utilisation d'un langage dédié pour la description des contraintes de déploiement. Cette approche est similaire à l'approche utiliser dans [8] [9] discuté dans la section 4. L'idée est qu'un administrateur de déploiement écrit les contraintes de déploiement en matière de ressources disponibles en intégrant des mécanismes qui nous offrent beaucoup plus d'expressivité par rapport au langage proposé dans [8], notamment pour traiter les aspects imprévisibles de la topologie (aspects non traités dans [9]). Par exemple, nous voulons exprimer qu'un composant doit être déployé sur tous les hôtes disponibles dans le réseau. Pour cela, nous avons développé j-ASD DSL, un langage de description de contraintes déploiement. j-ASD DSL est un langage déclaratif doté d'une grammaire simplifiée

et intuitive réaliser sous la forme d'un plugin Eclipse en utilisant Xtext [1]. Grâce a j-ASD DSL, l'administrateur de déploiement est en mesure de décrire le logiciel à déployer ainsi que les contraintes de déploiement.
Un logiciel est défini par son nom (identifiant), sa version, son URL et les composants qui le forment. Le logiciel est défini aussi par ses dépendances logicielles, ses préférences matérielles et ses contraintes de déploiement. Les types de données supportées par j-ASD DSL sont : chaîne de caractères (String) et entier (Integer). Un logiciel peut être composé par un ou plusieurs composants, chaque composant est défini par le nom du composant, la version du composant, la localisation de son implémentation (URL du composant) et les dépendances logicielles du composant.
Les contraintes matérielles et logicielles sont, les contraintes sur le système d'exploitation `OsPref`, les contraintes sur le processeur `CPUPref`, les contraintes sur la mémoire disponible `RAMPref`, les contraintes d'affichage `HDPref`, et les contraintes sur la vitesse du réseau `NetSpeedPref`. Cette liste n'est pas exhaustive et peut-être étendue par la suite par d'autres types de contraintes comme la contrainte sur l'utilisation de la batterie `PowerPref` par exemple.

décrit le logiciel à déployer, les composants qui forment le logiciel, les services fournis et requis du logiciel, les dépendances du logiciel, les préférences matérielles et les contraintes de déploiement.

Le programme montré ci-dessous (Fig. 2) décrit un logiciel nommé ExtractFromScenario_1, il comporte deux composants (ramSize et display). Les composant ramSize et display sont définis par leurs noms, leurs versions et leurs URL respectives. Les composants ramSize et display sont récupérés à partir d'un serveur http.

```
Software {
  Name=ExtractFromScenario_1
  Version=1
  Components=ramSize display
}
Component {
  Name=ramSize
  Version=1
  Url="http://x.fr/RAM-Size.jar"
}
Component {
  Name=display
  Version=1
  Url="http://x.fr/Display.jar"
}
```

```
HostConstraint {
  Name=Display-Constraint
  CPULoad < 80%
  RAM >= 40 MB
  OSNameContains "Linux"
}
Deployment {
  ramSize @ all
  display @ 1 with Display-
  Constraint
}
```

FIGURE 2. Exemple de programme écrit avec le DSL j-ASD

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 :

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

- La charge du processeur (CPULoad) doit être inférieur à 80%.
- Le composant à besoin d'un minimum de 40 Mo de mémoire.
- Le système d'exploitation doit être un système Linux.

Finalement, 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).

B. Service de découverte du réseau

Nous proposons un service de découverte du réseau qui permet de détecter dynamiquement l'ensemble des sites disponibles pour servir de cibles de déploiement. Ce service permet à la fois la découverte du réseau local dans le cas où l'utilisateur souhaite déployer son logiciel sur l'ensemble (ou un sous-ensemble) des sites disponibles connectés au réseau local ; une découverte étendue (à grande échelle) dans le cas où l'utilisateur souhaite déployer une application à grande échelle ; ou une découverte multi-échelle (découverte mixte du réseau local et à grande échelle).

Pour la réalisation de notre système de découverte nous avons choisi de combiner les protocoles UPnP [10] et XMPP [11]. Le protocole UPnP est utilisé pour la découverte du réseau local, tandis que le protocole XMPP est utilisé pour la découverte à grande échelle. L'idée est que le système de déploiement (le bootstrap), 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 d'un réseau local. Le point de contrôle global, permet la détection d'une manière totalement transparente les connexions/déconnexions des Device UPnP client dans le réseau et fourni à notre système de déploiement un ensemble d'informations (nom de la machine, adresse IP) qui permettant par la suite le calcule d'un plan de déploiement initial.
Dans le cas de la découverte à grande échelle le système de déploiement installe et exécute un client XMPP sur chaque site cible de déploiement et un client spécialisé (client central) dans le site initiateur de déploiement. Tous les clients XMPP seront connectés d'une manière automatique à un serveur XMPP. Une fois qu'un client est connecté au serveur, il envoie d'une manière périodique le couple d'information (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 exploitons les fonctionnalités de gestion de présence offertes par le protocole XMPP. Ces deux modules de découverte du réseau permet de détecter toutes les situations de nouvelles connexions, de pannes des machines et de déconnexions. Cela permet à notre middleware de déploiement de réaliser un déploiement de logiciel même dans les environnements les plus instables

comme les systèmes P2P et ubiquitaire. De plus, la vision globale sur l'état des sites cibles de déploiement ainsi que la détection de défaillances ou des déconnexions représente un atout qui permettra à notre système de déploiement de réaliser des opérations de reconfigurations dynamiques au moment de l'exécution.

C. Bootstrap

Comme nous l'avons vu précédemment, notre système de déploiement doit traiter les problèmes d'administrations multiples et des droits d'accès aux sites cible de déploiement. Nous ne voulons pas contourner les principes de sécurités des systèmes distribués, par conséquent nous comptons que sur les administrateurs des sites pour obtenir les droits d'installation et d'exécution de notre environnement de déploiement dans leurs hôtes. Cela pourrait être réalisé grâce à un programme dédié (le bootstrap) installé volontairement par l'administrateur du site et mis à sa disposition par d'autres 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 est 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 middleware. Il contient en particulier les éléments clients du système de découverte du réseau décrit précédemment.

D. Solveur de contraintes de déploiement

Une fois que la description des contraintes de déploiement est fournit sous la forme d'un programme j-ASD DSL, le service de découverte du réseau est lancé pour détecter les sites cibles de déploiement. Ce service retourne une liste de sites cibles potentiels, qui sera la liste initiale des sites cibles de déploiement. Cette liste est passée en entrée du programme j-ASD DSL, ce qui permet de générer un problème de satisfaction de contraintes résolvable par un solveur de contraintes. Un problème de satisfaction de contraintes (CSP) est exprimé par la déclaration d'un ensemble de variables (aux sens inconnus) dont les valeurs sont tirées à partir d'un domaine de valeurs et un ensemble de contraintes sur les variables. Les contraintes sont simplement des relations logiques entre plusieurs variables. La résolution d'un CSP revient à trouver un ensemble consistent de valeurs pour les variables et qui satisfassent toutes les contraintes sur les variables. Nous avons choisi Choco [12] pour notre prototype.

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 généré par le compilateur est construit à partir d'un ensemble de variables entières (variables de localisation) et un ensemble de contraintes sur ces variables. Nous modélisons le programme CSP avec les éléments suivants :
Un ensemble fini C de composants logiciels. Par exemple, dans l'exemple présenté précédemment (Fig. 2) l'ensemble C = {display, ramSize}.
Un ensemble de sites cibles de déploiement H détectés par le service de découverte du réseau.
Un ensemble de variables de localisation (Loc), qui modélisent 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 le niveau de charge du processeur 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.

Les contraintes de déploiement déclarées dans le programme j-ASD DSL seront traduites par le compilateur en des contraintes sur les variables de localisation (loc (Ci, Hj)) comme montré dans l'exemple ci-dessous (Fig. 3).

```
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 hosts with Constr2;
}
```

FIGURE 3. Second exemple de programme écrit avec le DSL j-ASD

La première contrainte signifie que le composant C1 doit être déployé dans tous les sites cibles de déploiement, cela signifie formellement :
$c1 \in C, \forall Hj \in H$, loc (c1, Hj) = 1.
La deuxième contrainte signifie que le composant c2 doit être déployé dans un sous-ensemble de trois sites. La troisième contrainte signifie que le composant c3 doit être déployé dans tous les sites qui satisfassent les contraintes définies dans Constr1. Autrement dit, le composant c3, doit être installé dans tous les sites qui comportent au minimum 150 Mo de mémoire disponible et équipés d'un système d'exploitation de type Windows.

Formellement :

$c3 \in C, \forall Hj \in H$, **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.

Le solveur retourne la première solution valide du CSP si il en existe une. Elle se présente sous la forme d'un ensemble de variables, qui sera directement interprêtée comme un plan de déploiement initial par le système d'agents mobiles. Si il n'y a pas de solution, un message d'erreur est renvoyé à l'utilisateur qui devra revoir ses contraintes pour réussir un déploiement. Il est possible que le solveur de contraintes arrive à trouver plusieurs solutions consistantes et 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, pour l'instant notre objectif est de trouver une première solution consistante qui satisfasse les contraintes de déploiement. Le travail sur la sélection de la meilleure solution est une perspective intéressante de ce travail.

E. Support de déploiement

Le support de déploiement fourni 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 logiciel sur des hôtes hétérogènes et à ressources réduites tel que les ordinateurs portables, les tablettes tactiles, les Smartphones, les voitures et les PC-Ultra mobile. Nous ne visons pas dans ce travail des environnements plus petits que ceux décrits.

Plusieurs plates-formes de déploiement existantes comme OSGi et D&C fournissent tout ou une partie des fonctionnalités souhaitées. Pour la réalisation de notre prototype, nous avons choisi la plate-forme OSGi [2] comme support de déploiement. Les motivations de notre choix de plate-forme et d'unité de déploiement sont discutées dans la section suivante.

F. Système d'agents mobiles

L'utilisation des agents mobiles pour l'automatisation du processus de déploiement n'est pas une nouvelle approche. Il existe plusieurs travaux dans la littérature qui ont utilisé cette technique pour déployer des logiciels dans des infrastructures réseau statique (voir par exemple [3]). Généralement, les solutions proposées ne sont pas adaptées au déploiement de logiciel dans des environnements ouverts tels que les systèmes ubiquitaires.

Un agent logiciel peut être défini comme un programme autonome caractérisé par des données et un comportement privé [13]. Un agent mobile est un logiciel capable de se déplacer au moment de l'exécution avec son code et ses données [14]. Un agent mobile adaptable (AMA) [15] est une entité logicielle autonome capable de communiquer et de se déplacer, disposant d'un comportement privé et d'une capacité d'adaptation au contexte de l'exécution. Les agents logiciels communiquent généralement par envois de message en mode asynchrone. Un système d'agents mobiles est un Framework qui implémente le paradigme des agents mobiles [16], il fournit des primitives et des services qui permettent l'implémentation, les communications et la migration des agents logiciels.

Nous utilisons un système d'agents mobiles adaptables pour exécuter et superviser le processus de déploiement. Pour cela, nous avons créé des agents de supervision (global et local) et des agents de déploiement. Les agents de supervision 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, déconnexion, par exemple). Afin d'assurer le passage à l'échelle nous proposons deux types d'agents de supervisions, des agents de supervision locaux (LSA) et un agent de supervision global (GSA).

Un agent de supervision local est déployé par l'agent de supervision global sur un site cible d'un sous réseau local (un sous réseau de classe C pour des raisons de passage à l'échelle). L'agent de supervision local a pour rôle la création des agents de déploiement dans chaque site cible du sous-réseau pour installer, activer, désactiver et mettre à jour le logiciel à la demande de l'agent de supervision global. Il a aussi pour rôle la supervision du processus de déploiement et la réalisation des opérations de reconfiguration dynamique dans le sous-réseau.

L'agent de supervision global est créé au début du processus de déploiement. Il a pour rôle l'exécution et la supervision du plan de déploiement initial calculer par le solveur de contraintes. Il n'y a qu'un seul agent de supervision global dans notre système créé et exécuté initialement dans le site initiateur du processus de déploiement. Il contrôle le processus de déploiement par la coordination avec les agents de supervision locaux. Pour cela, l'agent de supervision local échange des messages asynchrones avec les LSA pour connaître l'état des activités de déploiement dans chaque sous-réseau. L'agent de supervision global fourni aussi des interfaces d'échange à l'administrateur de déploiement qui lui permet de vérifier et d'intervenir à tout moment dans l'une des activités du processus de déploiement. Cet agent peut aussi prendre d'une manière autonome des décisions de migrations vers d'autres sites pour réagir aux variations de qualité de service et aux pannes de machines et de déconnexion.

L'agent de déploiement est chargé d'exécuter les activités de déploiement dans chaque site cible de déploiement.

2. http://www.osgi.org

Il est créé et exécuté dans un site cible selon le plan de déploiement. 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 composants dans le site cible et la notification de la fin de l'activité d'installation à l'agent de supervision par un message asynchrone. L'agent de déploiement permet aussi l'activation, la désactivation et la désinstallation des composants de l'application à déployer à la demande de l'administrateur de déploiement ou l'agent de supervision.

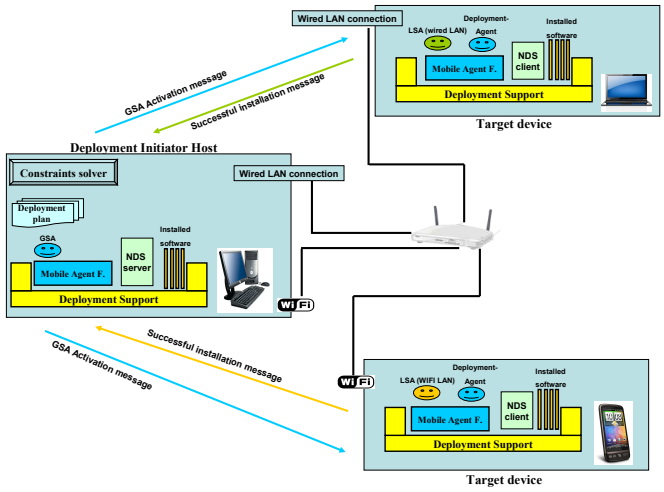

FIGURE 4. Agents de supervision

Après l'installation d'un (ou plusieurs) composant, l'agent de déploiement effectue une opération de vérification locale de l'échec/succès de l'activité d'installation et envoie un message de notification à l'agent de supervision local selon la situation (message de succès ou d'échec de l'installation). Une fois le message est reçu, l'agent de supervision local traite localement les messages reçus et notifie par la suite l'information de l'échec/succès de l'installation au GSA par l'envoi d'un message de succès ou d'échec d'installation dans le sous-réseau.
Le GSA envoie un message d'activation à tous les agents de supervision locaux une fois qu'il a reçu tous messages de succès de l'installation comme le montre la figure 4. Le comportement des agents de supervision locaux sera la création et l'envoi des messages d'activation à tous les agents de déploiement déployés dans le sous-réseau. Si ls GSA ne reçoit aucun message (succès/échec de l'installation par exemple) de la part d'un agent de supervision local (à cause d'une déconnexion ou panne de machine par exemple) dans un intervalle de temps fini, il procède à une procédure de reconfiguration, qui consiste à la désactivation de tous les agents de supervision locaux et l'envoi d'un message d'auto destruction (suicide) à l'agent qui ne répond pas. Ce dernier, une fois qu'il a reçu le message de suicide, envoie un message d'auto destruction aux agents de déploiement créés et termine sa propre exécution. En parallèle l'agent de supervision global 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 LSAs désactivés après la réception du message de succès de l'installation de la part du nouveau LSA.

Il est à noter que l'agent de supervision local peut migrer vers un autre site cible de déploiement dans le sous-réseau sans consulter l'agent de supervision global afin de corriger une défaillance détectée (l'agent dispose de moins de mémoire par exemple). Cela offre à notre middleware la possibilité de réaliser des opérations de reconfiguration dynamique et autonomique au moment de l'exécution. Le LSA informe par la suite l'agent de supervision global de ça nouvelle position après la correction de la situation.supervision local 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 afin de trouver une solution à cette défaillance. Dans le cas où aucune solution n'est trouvée, le GSA mettra fin au processus de déploiement et demande à l'administrateur de déploiement de réduire l'ensemble des contraintes proposées.

III. PROTOTYPE

Pour valider notre approche, nous avons développé un prototype entièrement fonctionnel. 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 implémenter sous la forme d'un plugin Eclipse en utilisant le plugin Eclipse Xtext. Le plugin développé utilise la librairie Java open source CHOCO [12] pour la résolution des contraintes de déploiement afin de calculer un plan initial de déploiement. Comme nous l'avons expliqué précédemment, le programme j-ASD DSL sera compilé sous la forme d'un problème de satisfaction de contraintes de bas niveau (programme Choco) qui sera résolu par la suite par le solveur de Choco. La solution trouvée par le solveur de Choco est un ensemble de variables entières et booléennes qui seront interprétés comme un plan de déploiement par le système d'agents mobiles. Nous avons choisi le Framework JavAct [3] comme plate-forme d'agents mobiles [17] , le Framework OSGi comme support de déploiement et les protocoles UPnP et XMPP pour la réalisation de notre service de découverte du réseau.
OSGi est une spécification qui définit une plate-forme d'exécution de composants Java appelés Bundles. Un bundle est l'unité physique de déploiement. Concrètement, un bundle est un fichier Java (archive JAR) qui contient un manifest et une combinaison de fichier de classes Java et des ressources. OSGi permet l'installation, la désinstallation, l'activation, la désactivation et la mise à jour de bundles au moment de l'exécution et inclus

3. http://www.javact.org

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 hétérogènes. 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. déploiement d'autres types d'unité de déploiement est envisagé plus tard.

Nous avons aussi développé le système d'agents mobiles ainsi que les algorithmes 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 cibles de déploiement pour exécuter et superviser les activités de déploiement dans les environnements les plus instables. L'utilisation des agents mobiles nous offre des possibilités de reconfigurations et d'adaptations dynamiques au contexte d'exécution du logiciel déployé. 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.

Nous avons proposé et implémenté un service de découverte du réseau pour la détection automatique des sites cibles. Pour cela nous avons choisi les protocoles XMPP et UPnP avec leurs implémentations Java open source respectives Smack [4] et Cyberlink [5].

Au moment de la rédaction de cet article, le prototype est fonctionnel et une première évaluation grandeur nature a été réalisée [6]. Par exemple, le temps nécessaire pour calculer le plan de déploiement initial de 20 composants sur 200 sites cibles de déploiement avec 10 contraintes de déploiement est inférieur à une minute. Les données de performances sont calculées sur un ordinateur portable équipé d'un processeur duel core Intel Pentium III Xeon 2.4 Ghz et 4 GO de mémoire. Le temps de calcul du plan de déploiement de 20 composants sur 5000 hôtes cible avec 15 contraintes de déploiement peut aller jusqu'à 20 minutes. Lors de l'ajout de plus de contraintes nous avons eu des temps de calcul de moins de 5 minutes dans certaines situations.

IV. Travaux Connexes

Il existe un grand nombre de travaux académiques et industriels décrivant des d'outils, des procédures autour de plusieurs aspects du processus de déploiement. Mais ceux-ci sont presque toujours destinés à des topologies de machines figées et/ou connues au moment du déploiement et donc peu pertinentes pour notre contexte. Cette section présente quelques travaux de recherche relative au déploiement de logiciel.

4. http://www.igniterealtime.org/projects/smack
5. http://www.cybergarage.org/twiki/bin/view/Main/CyberLinkForJava
6. Rapport d'évaluation : http://Javact.org/JASD-rapport.pdf

Fractal Deployment Framework (FDF) [4], est un outil générique qui permet de faciliter le déploiement d'applications distribuées. Il est composé d'un langage de description de déploiement, une librairie de composants et un ensemble d'outils et d'interfaces utilisateur. Le langage de description de déploiement pour décrire les configurations de déploiement. Les interfaces utilisateur permettent le chargement, l'exécution et la gestion des configurations. L'unité de déploiement est une archive qui contient les binaires du logiciel et un descripteur de déploiement. La principale limitation de cet outil est l'attribut statique du déploiement, bien qu'un plan de déploiement statique soit admissible dans des environnements relativement stables comme les grilles de calcul, ce type d'outil n'est pas utilisable dans des environnements caractérisés par une topologie dynamique du réseau comme les environnements ubiquitaires. Une autre limitation de FDF, est qu'il ne fournit pas de mécanismes de reconfiguration dynamique qui permettent la prise en compte des situations de pannes de machines par exemple.

Software Dock [3], est un projet de recherche, qui fournit un Framework pour la configuration et le déploiement de logiciels. Software dock, utilise un système d'événements et des agents mobiles pour contrôler les activités de déploiement comme l'installation et l'activation. Le cycle de vie de déploiement dans software dock incluse, l'installation, l'activation, la désactivation, la mise à jour, la désinstallation et la reconfiguration. Le système de déploiement utilise une architecture client/serveur en combinaison avec un système de gestion des événements. Un serveur appelé release dock est installé au niveau du site du producteur. Un client appelé field dock est installé sur chaque site des consommateurs de logiciels et qui joue le rôle d'une interface pour le release dock. Cependant, software dock ne permet pas la description de l'architecture du logiciel et les contraintes de déploiement. Software dock, propose aussi un processus de déploiement centralisé et statique qui ne permet pas de répondre à nos besoins de reconfiguration dynamique et de déploiement sur des environnements ouverts.

R-OSGi [18] est un middleware qui étend les standards de la spécification OSGi pour supporter la gestion des modules distribués. Il permet à des hôtes équipés du Framework OSGi d'interagir d'une manière distribuée. Au moment du déploiement, R-OSGi peut être utilisé pour exécuter une application comme une application distribuée en indiquant simplement où les différents modules doivent être déployés. Le développeur d'une application R-OSGi dispose d'un contrôle total sur la façon dont l'application est distribuée. Le contrôle manuel du processus de déploiement ainsi que sa configuration dans un environnement réparti à grande échelle est une tâche très complexe et représente pour nous une intervention humaine très importante dans le processus de déploiement. De plus, R-OSGi est uniquement destiné à créer des déploiements statiques de logiciels qui ne peuvent pas être utilisés dans le cadre

des environnements répartis à grande échelle ouverts tel que les systèmes ubiquitaires et P2P.

A. Dearle et al propose dans [9] un middleware pour le déploiement et la gestion des applications constituées d'un ou plusieurs composants appelés Cingal-bundle. Les contraintes de déploiement sont spécifiées avec le langage Deladas. L'administrateur de déploiement spécifie un objectif de déploiement initial, ensuite le système de déploiement tente de générer une configuration qui décrit la manière de déploiement des composants de l'application. Après un déploiement initial, le système de déploiement vérifie la satisfaction de l'objectif initial et re-déploie l'application si nécessaire. 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. Toutefois, le middleware proposé n'est pas utilisable dans des environnements dotés d'une topologie imprévisible, en plus l'utilisateur de cet outil est obligé à redémarrer tout le processus de déploiement dans le cas d'une déconnexion une défaillance par exemple. Notre solution, comporte des décisions d'adaptations décentralisées prises par les agents mobiles et permet de faire des reconfigurations réalisables et très légères au niveau local dans des environnements répartis ouverts.

V. Conclusion

Dans cet article nous avons présenté notre middleware j-ASD dédié au déploiement autonomique de logiciels en environnement ubiquitaire ainsi que certains éléments du prototype qui permet la validation de notre approche. La solution proposée est composée d'un langage dédié (DSL) à la description des contraintes de déploiement, un service réseau pour découvrir d'une manière automatique les sites cibles de déploiement, un logiciel de bootstrap pour la préparation de l'environnement d'exécution, un solveur de contraintes pour le calcul d'un plan déploiement initial et un système d'agents mobiles adaptables pour l'exécution et la supervision des activités de déploiement. Nous estimons que j-ASD peut répondre aux exigences et aux spécificités des environnements répartis ouverts dotés d'une topologie dynamique du réseau. Nous avons utilisé le Framework OSGi comme support de déploiement pour permettre le déploiement d'applications basées sur des bundles OSGi sur plusieurs types d'équipement hétérogènes (PDA, téléphones mobiles, Smartphones, tablettes et ordinateur). Le service de découverte de 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. La solution générée par le solveur de contraintes est dynamiquement interprêtée comme plan de déploiement initial par le système d'agents mobiles. Les caractéristiques des 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 poursuivons actuellement nos travaux sur les tests et les évaluations de la dernière version du prototype j-ASD sur des environnements répartis à grande échelle. En parallèle nous expérimentons des algorithmes de déploiement plus intelligents pour faire face aux besoins d'adaptations dans des situations plus complexes après la réalisation du déploiement initial.

Références

[1] A. Dearle, "Software deployment, past, present and future," in *FOSE*, 2007, pp. 269–284.

[2] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. van der Hoek, A. L. Wolf, A. V. Der, E. L. Wolf, and E. L. Wolf, "A characterization framework for software deployment technologies," Dept. of Computer Science, University of Colorado, Tech. Rep., 1998.

[3] R. S. Hall, D. Heimbigner, and A. L. Wolf, "A cooperative approach to support software deployment using the software dock," in *ICSE '99*, 1999.

[4] A. Flissi, J. Dubus, N. Dolet, and P. Merle, "Deploying on the grid with deployware," in *CCGRID*, 2008, pp. 177–184.

[5] O. M. Group, "Corba component model 4.0 specification," Object Management Group, Specification Version 4.0, April 2006. [Online]. Available : http://www.omg.org/docs/formal/06-04-01.pdf

[6] C. Louberry, P. Roose, and M. Dalmau, "Kalimucho : Contextual Deployment for QoS Management," in *11th IFIP WG 6.1 International Conference, DAIS 2011, Reykjavik, Iceland, June 2011, Proceedings*, ser. Lecture Notes in Computer Science, vol. 6723. Springer, Jun. 2011, pp. pp.43–56.

[7] M. E. A. Matougui and S. Leriche, "Vers un environnement de déploiement autonomique," in *Ubimob'2011 : Septièmes journées francophones Mobilité et Ubiquité*, 2011, pp. 57–62.

[8] A. Dearle, G. N. C. Kirby, and A. J. McCarthy, "A framework for constraint-based deployment and autonomic management of distributed applications," in *ICAC*, 2004.

[9] A. Dearle, G. N. C. Kirby, and A. McCarthy, "A framework for constraint-based deployment and autonomic management of distributed applications," *CoRR*, vol. abs/1006.4572, 2010.

[10] U. Forum, "Upnp device architecture," 2008. [Online]. Available : http://www.upnp.org/

[11] P. Saint-Andre, K. Smith, and R. Tronçon, *XMPP : The Definitive Guide : Building Real-Time Applications with Jabber Technologies*. O'Reilly Media, Inc., 2009.

[12] C. Team, "choco : an open source java constraint programming library," Ecole des Mines de Nantes, Research report, 2010. [Online]. Available : http://www.emn.fr/z-info/choco-solver/pdf/choco-presentation.pdf

[13] J. M. Bradshaw, Ed., *Software agents*. Cambridge, MA, USA : MIT Press, 1997.

[14] C. G. Harrison, C. G. Harrison, D. M. Chess, D. M. Chess, A. Kershenbaum, and A. Kershenbaum, "Mobile agents : Are they a good idea ?" 1995.

[15] S. Leriche and J.-P. Arcangeli, "Flexible architectures of adaptive agents : the agentϕ approach," *International journal of grid computing and multi agent systems (IJGCMAS)*, vol. 1, no. 1, pp. 55–75, Jan. 2010, 8878.

[16] R. S. S. Filho, R. Silveira, and S. Filho, "Mobile agents and software deployment," 2000.

[17] 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*, 2001.

[18] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-osgi : distributed applications through software modularization," in *Proceedings of the 8th ACM/IFIP/USENIX international conference on Middleware*, ser. MIDDLEWARE2007. Berlin, Heidelberg : Springer-Verlag, 2007.