HAL
open science

A middleware architecture for autonomic software deployment

Mohamed El Amine Matougui, Sébastien Leriche

To cite this version:

Mohamed El Amine Matougui, Sébastien Leriche. A middleware architecture for autonomic software deployment. ICSNC '12 : The Seventh International Conference on Systems and Networks Communications, Nov 2012, Lisbon, Portugal. pp.13-20. hal-00755352

HAL Id: hal-00755352

https://hal.science/hal-00755352v1

Submitted on 21 Nov 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A middleware Architecture for Autonomic Software Deployment

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

***Abstract*—Autonomic software deployment in open networked environments such as mobile and ad hoc networks is an open issue. Some solutions to software deployment exist but they are usable only within static topologies of devices. We propose a middleware architecture providing a constraint-based language guiding the deployment process at a high level and an autonomous agent-based system for establishing and maintaining a software deployment according to a deployment plan. Constraints solver generates the deployment plan from the initial specification and a network discovery service is used to automatically detect the target hosts. This middleware architecture considers the challenges of deploying distributed software over mobile and ad hoc networks with minimal human oversight.**

***Keywords*-autonomic deployment; ubiquitous computing; mobile agents; middleware**

I. Introduction

Software deployment is defined as a complex process that includes a number of inter-related activities. Currently there is no consensus around deployment activities. The deployment life cycle includes all activities between the software release and the software removal from deployment sites [1]. A generic deployment process covers the installation of software into the execution environment and the activation of the software, it also contains some post installation activities such as deactivation, updating, monitoring, reconfiguring (adapting) and uninstalling of the software [2].

Several large-scale deployment platforms exist such as Software Dock [3], DeployWare [4], D&C [5], JADE [6] or more recently KALIMUCHO [7]. These deployment tools are beginning to reach their limits; they use techniques that do not suit the complexity of the issues encountered in ubiquitous infrastructures. Instead, they are only valid within fixed network topology and do not take into account neither QoS variations nor the machine or links failures characterizing these environments. In addition, users of these deployment tools are required to manage manually the deployment activities, which represent a very significant human intervention in the deployment process. Indeed, for large distributed component-based applications with many constraints and requirements, it is hard to accomplish the deployment process manually. Clearly, there is a need for new infrastructures and techniques that automate the deployment process and offer dynamic reconfiguration of software systems with a minimum of human intervention.

To address these issues, we propose in this paper a new middleware for autonomic software deployment that is composed of (1) a domain-specific constraint language and a constraint solver for expressing deployment constraints and planning how the software will be deployed onto the target hosts (calculating a deployment plan), (2) a network discovery service and a bootstrap for the discovery of the deployment target hosts, (3) a deployment support for executing the deployment activities, and (4) an adaptable mobile agent system that runs and supervises the deployment process.

This paper is organized as follows: Section 2 introduces a motivating example and discusses the need for autonomic software deployment and the requirements of ubiquitous and mobile or ad-hoc environments. Section 3 presents an overview of our approach for autonomic software deployment. In Section 4, we present the technologies involved in our prototype and some experimental results. In Section 5, we discuss some related work. Finally, in Section 6 we conclude the paper and give an overview of our future work.

II. Motivating example

In order to highlight specificities and problems encountered for software deployment in mobile and ad hoc networks infrastructures we present a deployment scenario, in which we deploy an activity as a monitoring application for a set of hosts (unknown at the design time) connected to a wireless network.

We consider a distributed software for providing statistical information on all hosts connected to the local area network (WiFi network). The context information in this experiment are respectively available memory size, OS type, processors usage and available disk space in each deployment target host.

At the beginning of the deployment process, the number of participants in this experiment is unpredictable; it can range from a dozen to a hundred of participants. Hosts involved in the deployment process are equipped with various hardware and software environments ranging from personal

computer and smartphone to ultra-mobile devices such as tablets or PDA.

Each host connected to the wireless network is a potential deployment target host. The software to deploy is composed of eight components, each component being a deployment unit. The `Display-Results` component, allows the display of the statistical information computed during this experiment. This component must run on a Linux platform, and requires 40MB of RAM and at least 20% of the CPU. The components `Average-Memory`, `Average-Disk` and `Average-CPU-Occupation` calculate respectively the average of the available memory, the average disk space available and the average occupation rate of processors.

The desired deployment plan for this application is as follows: one Linux host with enough memory and CPU will get all the components. Each of the other available host will get only the components (`OS-Type`, `RAM-Size`, `Disk-Size` and `CPU-Occupation`).

To run this plan, the deployment system must face into many problems and specificities. First the discovery of available hosts in the target environment by a network discovery service. We need this step because in this 'unpredictable topology' scenario, we do not know in advance (at the design time) the list of involved hosts of the deployment process. Second the multiple administrators problem. Indeed, the deployment target environment is a set of independent hosts where each host has its own administrator. Therefore, we must obtain the access rights on each host to be able to deploy any software.

Then, the deployment system must cover all deployment activities (from installing to uninstalling the software). The installation activity includes the software dependencies solving, transferring the components on the target sites and the physical installation of the components. At this step, the deployment system must provide a mechanism for dynamic reconfiguration of the deployment process to support the failure of hosts, the disconnections and the new connections of hosts. For example, if the selected host to have the `DisplayResults` component installed has a failure (or do not have enough RAM at the time of the deployment), the deployment system must dynamically find another host satisfying the constraints and go on with the deployment process. From this scenario we conclude that the deployment platform that can addresses the specificities and problems of these environments (P2P and ubiquitous) must (Rx are requirements):

- **R1**- Be able to detect, manage and access the target hosts with a minimum user interaction.
- **R2**- Be able to deal with hardware and software heterogeneity.
- **R3**- Provide a simple and intuitive language to describe the software dependencies, the software properties and the deployment constraints.
- **R4**- Be able to compute at least one deployment plan that satisfies the deployment constraints.
- **R5**- Perform the deployment activities with minimum human intervention.
- **R6**- Provide autonomic mechanisms to reconfigure the deployment process at runtime to handle variations of the topology and hosts or network failures, accordingly to the deployment constraints.
- **R7**- Be usable in large-scale infrastructures.

III. J-ASD OVERVIEW

In this section, we present our middleware platform called j-ASD for autonomic software deployment, which addresses the above Rx requirements. The proposed software architecture is shown in Figure 1.

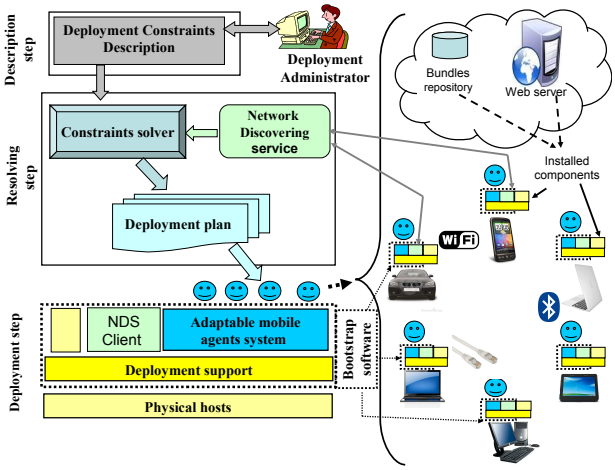

Figure 1. Architecture of j-ASD

The middleware architecture is composed of 5 different parts.

A *domain-specific constraint language* (DSL) for expressing the deployment constraints and some information on the software, a language parser and constraints solver are necessary for calculating a deployment plan.

A *network discovery service* to automatically detect target hosts in the network. By using this service, our deployment system should enable software deployment in open environments such as ubiquitous computing environments.

A *bootstrap software*, which prepares the execution environment in the target hosts of the deployment system. The bootstrap should resolve the multiple administrators problem, and install and activate all dependencies for our deployment system.

A *deployment support* equipped with a runtime environment able to run in heterogeneous infrastructures, and which can perform all or a part of the deployment activities.

Finally, we use an *adaptable mobile agent system* that performs and supervises the deployment process.

A. Language for Deployment Constraints

In order to automate the software deployment processes it is necessary to have some deployment knowledge about the software system. This knowledge, is called *description of the deployment constraints*. In the existing deployment platforms there are several formalisms to express this knowledge [3], they allow declaration of deployment constraints, software dependencies and software preferences. These methods include the use of ADL (architecture description language), the use of XML deployment descriptors (D&C and CORBA) and the use of a dedicated language (DSL) for describing deployment constraints.

Our approach is similar to the Deladas DSL [8], discussed in the related work (Section 5). We agree with the idea of an administrator, which describes a deployment goal in terms of available resources and constraints on their deployment. But we need much more expressiveness, particularly to deal with unpredictable aspects of the topology. For example, we want to express that one component should be deployed on each available host in the network. For this purpose, we have developed j-ASD DSL, a dedicated specific language with a simplified and intuitive syntax and grammar for describing deployment constraints.

By using the j-ASD DSL, the deployment administrator will be able to describe the software and the deployment constraints. The software is defined by a set of information about the software such as the software name, the software version, the software URL and the components, which form the software. The software is also defined by the software dependencies, the hardware and software constraints on the target hosts and finally the deployment constraints. Data types supported by j-ASD DSL are String and integer. The software can be composed of one or more components; each component is defined by the component name or ID, the component version, the component URL (the location of its implementation) and the component dependencies. Hardware and software constraints are respectively, operating system constraint `OsPref`, processor constraint `CPUPref`, memory constraint `RAMPref`, display constraint `HDPref` and network speed constraint `NetSpeedPref`. This list of constraints should be extended by adding other types of constraints and preferences like as the battery usage constraint for improving QoS offered by the j-ASD middleware.

As illustrated in Fig. 2, the deployment administrator describes a software named `ExtractFromScenario_1`, it includes two components called `ramSize` and `display`. In the same way, the component `ramSize` and `display` are defined by the component name, the component version and the component URL. The component can be located in a local repository or in a remote repository (in a web server for example). In this example, both components are located in an http server. The `hostConstraint` part is a high-level constraint specification of the display constraint on the target hosts. The `hostConstraint` name is Display-Constraint and expresses that in the deployment hosts:

```
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. DSL code sample for Scenario 1

- The processor constraint (`CPULoad`) in the deployment device must be less than 80%.
- The memory constraint (available memory) in the deployment device must be greater than or equal to 40 MB.
- The operating system constraint expresses that the operating system installed in the device must be Linux.

Finally, the deployment constraints are high-level constraint specifications, which express that the component `ramSize` will be deployed in each host available at the deployment time. The second constraint expresses that the component `display` will be deployed in one device that satisfies all constraints described in `Display-Constraint`.

B. Network discovery service

We use a discovery network service to allow end-users who do not necessarily know the deployment sites (devices) at the beginning of the deployment process to automatically detect these sites by invoking the service. In this context, we must distinguish two cases. The first case is the software deployment in local network (domestic network) or fixed networked devices in which the user wants to deploy the software across all (or in a subset) connected available sites in the network. The second case involves the software deployment in large-scale infrastructures like ubiquitous system and Grid. In both cases, the user does not necessarily know the deployment target host. Therefore, the deployment system must provide mechanisms to manage the network and allows the detection of each connected device in the network, then get the permissions (access rights) to each device by the bootstrap software. After the initial discovery of hosts, the network discovery service returns a host list to the deployment system. This list is used by the deployment system to produce a deployment plan.

In fixed network infrastructures, such as local area networks, several protocols such as Universal Plug-and-Play (UPnP) [9] and bonjour [10] have been successful solution.

However, these protocols do not operate effectively in large-scale environments. For these further specific discovery protocols have been created to specific domain like SLP [11], SIP [12] and XMPP [13]. For reducing interpretability issues, we have studied and chosen the UPnP and XMPP protocols to build our discovery network service. The UPnP protocols are used to deal with the first case of discovering (local network discovery) and a some parts of XMPP protocols are used to deal with the second case (large-scale network discovery) as discussed later in this section.

C. Bootstrap

As seen in the motivating example section, we need to deal with the multiple administrators problem. We do not want to bypass the principles of security in distributed systems, thus we must rely on each administrator to get the rights for running our deployment environment on each device. This could be achieved through a dedicated program voluntarily installed by the host administrator, and placed at its disposal through other ways. For example, it can be pushed on Bluetooth, or its url can be sent via e-mail, SMS or even embedded into a QR Code®. This very light bootstrap code is a script that asks the user the access rights to the host (such as permissions in a trusted architecture) and sets up the required runtime for the middleware.

D. Constraints solver

Once the user has completed the constraints description, the deployment system takes as inputs the constraints description program. The network discovery service is launched for detecting initial target hosts. The network discovery service returns a list of available hosts and some information about the context and resources of each one.

After syntax and lexical checking of the j-ASD DSL program, the constraints solver try to compute an initial deployment plan. For this, the program is compiled into a lower-level constraint satisfaction problem (CSP), which can be solved by an existing constraint solver like as JSolver [14] and Choco [15]. A CSP is expressed by declaring a set of variables whose values are drawn from a set of discrete domains, satisfying a set of given constraints. A lower-level constraint is simply a logical relation among several unknowns (or variables), each taking a value in a given domain. The constraints transformation is required because the wide gap between the level of abstraction used to model CSP programs in existing libraries and the abstractions used by a deployment administrator to express deployment constraints in j-ASD DSL. Our CSP problem is modeled by constructing a set of integer variables (location variables) and constraints on those variables. We model the CSP program as follow:

1) A finite set C of software components.
2) A set H of target devices (hosts).
3) A set of location variables (loc) such as:
   loc (Ci, Hj) = 1, if the software component Ci can be installed in the device Hj and loc (Ci, Hj) = 0, if the component Ci can not be installed in Hj.
4) A set P of host constraints or host preferences (e.g., CPULoad and the available memory).
5) A set of deployment constraints over the location variables (loc (Ci, Hj)).

The CSP problem that we must solve in order to build an initial deployment plan is the problem of component placement on the detected device in the network in accordance with the deployment constraints.

For example, the deployment constraints described in Fig. 2 are translated as follows:

The first constraint means that the component ramSize should be deployed in all available devices, which means formally:

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

The second constraint means that the component display should be deployed in one device that respecting all constraints defined in Display-Constraint. This means formally:

$display \in C$, $Hi \in H$, such as: **if** $((RAM \geqslant 40MB)$ and $(OSName = "Linux")$ and $(CPULoad < 80\%)$ **then** loc (display, Hi) = 1 **else** loc (display, Hj) = 0

By using this formal translation and other we automatically generate a CSP program. Then in the second step the generated CSP program is dynamically loaded into the solving tool. The solver is then invoked to resolve the generated CSP problem and returns the first solution found. The result of solving of the generated CSP program is a set of integer and boolean variables, which are mapped as a deployment plan by the mobiles agents system. The deployment plan determines where different components of the software will be installed and executed in the target environment. If the solver cannot find a consistent solution, the constraints solver will be restarted to try to find another deployment plan. If the constraints solver still unable to find a consistent solution, a failure is notified to the deployment administrator.

E. Deployment support

The deployment support should provide an execution environment and support services for components. It should allow installing, uninstalling, starting, stopping, and updating the components at runtime without restarting the entire system. It should also allow the system to deploy software on several heterogeneous devices like personal computer, laptop, PDA, tablet, smartphone, mobile phone, cars and ultra-mobile PC. Several frameworks and platforms such as OSGi [16] or D&C [17] provide a part or all desired functionalities (life cycle deployment activities). For our prototype, we choose the OSGi platform to deploy Java-based components, as discussed later.

F. Mobile Agent system

The use of mobile agents to perform an autonomic deployment process in large-scale infrastructures is not a new approach. Some works have used this technique for deploying software in static environment (for example [3]), but they have never used this technique in ubiquitous and P2P environments.

A software agent can be defined as a program that works on behalf of its owner [18]. It is an autonomous computing entity with private knowledge and behavior. A mobile agent is a software program able to move at runtime with its code, data, and computational state [19]. An adaptable mobile agent (AMA) [20] can change some of its operating and functional mechanisms at runtime. The agent itself controls mobility and adaptation. In order to fit wide-area networks, agents communicate in asynchronous mode. A Mobile Agent System is defined as a computational framework that implements the mobile agent paradigm [21]. This framework provides services and primitives that help in the implementation, communication, and migration of software agents.

We use an adaptable mobile agent system that runs and supervises the deployment process. For this purpose, we have created the deployment agents and the supervisor agents (global and local supervisor agents).

The Supervisor agents role is performing and controlling the deployment process, it can also reconfigure the deployed software in order to react to the environment changes (host or link failures for example) in which the software is installed. Our answer to scalability issues is to have two kinds of supervisor agents: the global supervisor agent (GSA) and the local supervisor agent (LSA). There is only one global supervisor agent in our system, running initially on the host where the deployment process has been started. This agent can decide itself to move if required by the changing environment. Local Supervisor Agent is deployed by the GSA on one host for each local sub-network (/24 sub-networks for scalability reasons). The LSA role is to create the deployment agents into each device in the sub-network for installing, uninstalling, activating, deactivating and updating the software and supervise the deployment process in the each target host in the sub-network. The LSA have the ability to migrate to another host without consulting the GSA in case of a local failure detection (network link failure for example). This gives the opportunity for a dynamic reconfiguration of the deployment process at runtime.

The global supervisor agent is created at the beginning of the deployment process. It performs the initial deployment plan calculated by the constraints solver. It controls the deployment process by creating local supervisor agents and then coordinating with. GSA exchange asynchronous messages with the LSA to know the status of deployment activities. Global supervisor agent provides some user interfaces to the deployment administrator. These interfaces allow users to check at any time the deployment process status and enables interventions in the deployment process at the request of the user.

A deployment agent is responsible for executing the deployment activities. Deployment agents are created and started in all target deployment devices, performing multiple operations such as: downloading the software packages (from a web server located in a cloud computing for example) into the target devices, resolving software dependencies, installing the software in the target hosts and notify the end of the install activity by an asynchronous message. The deployment agent allows starting the installed software at the request of the supervisor agents (local or global). It can also stop all or a part of deployed software and allows the software update and uninstall on the request of the supervisor agent or the end-user request by the GUI. After installing the software, each deployment agent checks locally the correctness of the installation process and sends a message of success or failure of the installation activity to the local supervisor agent. Once the message is received, the local supervisor agent notifies the information to the global supervisor agent by sending a successful or a failure installation message.

The GSA sends an activation message for all LSA once it has received all messages of successful installation. Then, the LSA behavior is creating and sending an activation message to the deployment agents in the sub-network. If GSA has not received any messages from an LSA, the GSA considers that there is a failure; it sends a deactivation message to each LSA and a suicide message to the LSA that does not respond. The next step is creating a new local supervisor agent in another chosen host to replace LSA that not responding then restarting all stopped LSA.

IV. Prototype

To validate our approach, we developed a fully working prototype. Deployment constraints are expressed with our own DSL, called j-ASD DSL. It has been designed with the Xtext [22] Eclipse plugin, giving us a ready to use environment (plugins) inside the Eclipse platform to drive our middleware. The plugin uses CHOCO [15], an open source Java library for solving constraint satisfaction problems (CSP), to build the initial deployment plan. The j-ASD DSL is compiled into a single lower-level constraint satisfaction problem (Choco program), which is then solved by Choco. The generated solution is a set of integer and Boolean variables, which are mapped as a deployment plan by the mobile agent system.

We have chosen the JavAct [23] framework as a mobile agent platform [24], the OSGi Framework as a deployment support, UPnP and XMPP to discover available hosts (local and large-scale form).

The OSGi [16] specification comprises a framework that provides an execution platform for Java-based components, called bundles. A bundle is the physical unit of deployment. Concretely, a bundle is a Java JAR file that contains a manifest and some combination of Java class files, native code, and any associated resources. OSGi allows installing, uninstalling, starting, stopping, and updating bundles at runtime without restarting the entire system and includes a generic mechanism for automatic dependencies management. Using OSGi as a deployment support allows us to deploy software on several heterogeneous infrastructures. The reuse of the deployment activities provided by the OSGi framework like installing and uninstalling bundles allows us to focus on other aspects of the deployment process. Currently, our deployment units are OSGi bundles. Deployment of other deployment units will be envisaged later. We use the OSGi Framework Equinox [25].

We have also developed mobile agents behaviors and algorithms for installing, starting, stopping, uninstalling and updating the deployment units. The current implementation allows the deployment of applications composed by one or more OSGi bundles on the OSGi runtime environments. The prototype allows us to create and send our specialized mobile agents to the targeted hosts in the network, to execute and supervise the deployment activities in the most installable environment like ubiquitous systems. These agents give us the possibility of reconfiguration and dynamic adaptation to the execution context of the deployed software. An example of adaptation is the decision to migrate to another target device if the current one does not have enough resources like memory or bandwidth.

In addition, we have built a network service for the discovery of the target hosts. We wanted some open sources/protocol technologies, to reduce interoperability issues. We found that UPnP [9], the XMPP [13] protocol, the SIP [12] protocol, or the SLP [11] protocol could be a basis for our discovery network service. At the end, we chose the UPnP [9] and XMPP [13] protocols with the Cyberlink [26] and Smack [27] open-source implementations integrated behind a lightweight facade design pattern. The XMPP and SIP protocols address multimedia session management and presence signalization. The idea is to use the UPnP technology for discovering target hosts in local networks (LAN), and the XMPP protocol for discovering the deployment hosts within a large-scale deployment (WAN).

At the time of writing, the prototype is functional. We are now working on the experimental evaluation and the validation of the deployment process in a real environment. We conducted several tests on wired and wireless network. Each experiment was performed five times to produce the time required to compute de deployment plan, the necessary time to detect the deployment hosts and the time needed to deploy the components into the target hosts.

The first experiment we have conducted concerns the time needed for computing the initial deployment plan. The performance data was obtained on a duel core Intel Pentium III Xeon 2.4 GHz laptop with 4GB RAM running Windows XP professional edition. The time needed for computing the initial deployment plan of 20 components into 200 hosts with 20 constraints is below 1 minute. We tried 20 components into 5000 hosts with 15 constraints; computing time can be up to 20 minutes. As shown in Fig. 3, the average time needed to for computing the initial deployment plan for the presented example in Section 2, for 10 and 3200 hosts is respectively 16 and 292 milliseconds.

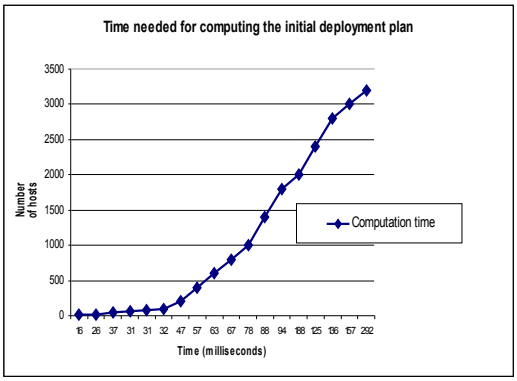

Figure 3. Time to compute the deployment plan results

The second experimentations series concerns the average time of discovering the devices and the average installation time. The install activity includes, the mobile agent creation, the bundles downloading in the target hosts, the bundles installation and the notification of the success of the bundle installation. This experimentation has been conducted on a fully connected 100 Mb/s Ethernet network of workstations (Intel(R) Xeon(R), 2.80GHz, 4029MB) and WiFi network of Samsung PC tablets (Intel(R) processor 800MHz, 0,99GB). The time needed to detect 60 devices is 15 seconds in the wired network and 16 seconds to detect 10 devices connected in the Wireless network.

As depicted in Fig. 4, the average installation time for the software presented in Section 2 (software composed from eight components) in 60 hosts is less than 13 seconds. For more details, you can download the full results of experiments (in French) from [28].

V. Related work

There are typically many constraints in the deployment of large-scale applications into distributed environments. For this purpose, software deployment process is given special attention both in academia and in industry and there is a large number of tools, procedures, techniques, and papers addressing different aspects of the software deployment process from different perspectives. In this section, we

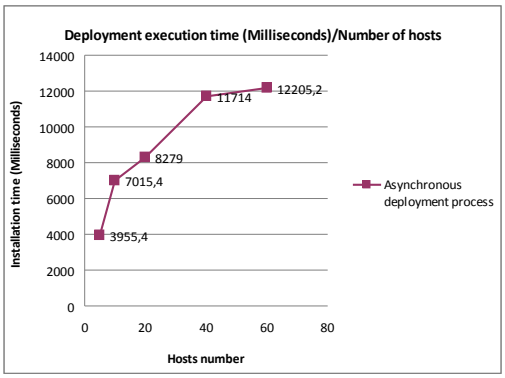

Figure 4. Installation time results

present some of the research literature related to software deployment.

Fractal Deployment Framework (FDF) [4] is a component based software framework to facilitate the deployment of distributed applications on networked systems. FDF is composed of a high-level deployment description language, a library of deployment components, and a set of end-user tools. The high level FDF deployment description language allows end-users to describe their deployment configurations (the list of software to deploy and the target hosts). The main limitation of this tool is the static and manual attributes of the deployment. Although the static deployment plan is eligible in stable environment like Grid, this deployment is not usable in an environment characterized by a dynamic network topology like ubiquitous environments. Another limitation is that at runtime this tool does not provide mechanisms for dynamic reconfiguration, which allows the treatment of the hosts and the network failures.

The Software Dock [3] is a research project, which provides a framework for software configuration and deployment. It has created a distributed, agent-based deployment framework, which supports cooperation among software producers themselves and between software producers and software consumers. The deployment framework uses client-server architecture in combination with an event system. However Software Dock, does not allow the description of the software architecture and propose a static centralized deployment process, which does not suit our requirements of dynamic adaptation and scalability.

R-OSGi [29] is a middleware platform that extends the standard OSGi specification to support distributed module management. R-OSGi provides a deployment tool to help developers to distribute an application by dragging and dropping between a visualization of the modules of the application and a representation of the distributed node available. The developer of R-OSGi application has full control on how the application is distributed. Creating a configuration and controlling the deployment process in the context of large-scale systems is a very complicated task, which represent for us a heavy human intervention in the deployment activities. In addition, R-OSGi is only intended to create static deployments and cannot be used within open environments that are characterized by dynamic network topology such as shown in our scenario.

A. Dearle et al. [8], [30] propose a middleware framework for deployment and subsequent autonomic management of component-based applications. The deployment constraints of distributed applications are specified using the Deladas (DEclarative LAnguage for Describing Autonomic Systems) language. An initial deployment goal is specified by using the Deladas language, and then in order to produce a concrete deployment of the application, the automatic deployment and management engine ADME attempts to generate a configuration that describes which components are deployed in which nodes by using a constraints solver. This approach has the similar motivations to our approach; in fact, one of the motivations is the costs reduction involved in human-managed system maintenance by the automatic generation of the deployment plan and with the mechanisms for reconfiguring the deployment at runtime. However, this centralized solution can not work in a change-prone environment (unpredictable topology), and needs a full restart of the deployment process for each error found at runtime. Our solution, involving decentralized decisions of deployment adaptation by mobile agents which allows to make light and local reconfiguration that are realistic (and scalable) in large scale and error-prone topologies.

VI. Conclusion

Our contribution in this paper can be summarized as follows. We described a scenario where other software deployment tools will fail; we discussed the need for autonomic software deployment in large-scale and change-prone infrastructures (Ubiquitous systems, P2P systems) and the requirements for such a platform. We presented j-ASD, our middleware dedicated to autonomic software deployment, and some elements of the prototype that validate our approach.

The j-ASD middleware addresses the requirements and specificities of those environments. (1) We have chosen the OSGi Framework as a deployment support to allow Java-based components deployment on several types of heterogeneous hardware. (2) The network discovery system and the bootstrap software allow the management and the access to the target devices with a minimum user interaction. (3) j-ASD DSL is an intuitive and a declarative way to specifies the deployment constraints (high-level constraints), the j-ASD DSL is compiled into a lower-level constraint satisfaction problem (CSP), which are resolved automatically by Choco solver. (4) The generated solution is dynamically mapped as an initial deployment plan by the mobile agents system. (5) Thanks to the characteristics of mobile agents

(autonomous behavior and ability to migrate), we can perform adaptations and dynamic reconfigurations at runtime without any user intervention.

We are currently pursuing our work on the evaluation of the last version of the j-ASD prototype toward large scale distributed systems and investigating on smarter algorithms to deal with the need for adaptation in more complex failure situations after the initial deployment.

REFERENCES

[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, and A. 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 *Proceedings of the 21st international conference on Software engineering*, ser. ICSE '99. ACM, 1999, pp. 174–183.

[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. Taton, S. Bouchenak, N. De Palma, D. Hagimont, and S. Sicard, "Self-sizing of clustered databases," in *WOWMOM '06*, 2006, pp. 506–512.

[7] 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*, vol. 6723, Jun 2011, pp. pp.43–56.

[8] 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.

[9] UPnP Forum, "UPnP Device Architecture, V 1.1," October 2008. [Online]. Available: http://www.upnp.org/

[10] APPLE, "Bonjour protocol specifications." 2009. [Online]. Available: http://developer.apple.com/networking/bonjour/specs.html

[11] E. Guttman, "Service location protocol: Automatic discovery of ip network services," *IEEE Internet Computing*, 1999.

[12] R. Sparks, "Sip: Basics and beyond," *Queue*, pp. 22–33, March 2007.

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

[14] A. H. W. Chun, "Constraint programming in java with jsolver," 1999.

[15] 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

[16] The OSGi Alliance, "OSGi service platform core specification, release 3. version 4.2," 2009.

[17] OMG, "Deployment and configuration adopted submission, document ptc/03-07-08 ed." Object Management Group, Tech. Rep., July 2003.

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

[19] 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.

[20] S. Leriche and J.-P. Arcangeli, "Flexible architectures of adaptive agents : the agentϕ approach," *International journal of grid computing and multi agent systems (IJGCMAS)*, pp. 55–75, 2010, 8878.

[21] R. S. S. Filho, "Mobile agents and software deployment," ICS280 Configuration Management and Runtime Change Final Paper, Information and Computer Science Department, University of California Irvine, Fall, Tech. Rep., 2000.

[22] "Xtext 2.3 Documentation," 2012. [Online]. Available: http://www.eclipse.org/Xtext/documentation/2.3.0/Documentation.pdf

[23] S. R. J.-P. Arcangeli, F. Migeon, "JAVACT : a Java middleware for mobile adaptive agents," 2011. [Online]. Available: http://www.javact.org

[24] 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.*, 2001.

[25] "Getting Started with Equinox," 2012. [Online]. Available: http://www.eclipse.org/equinox/

[26] "CyberLink for Java," 2012. [Online]. Available: http://www.cybergarage.org/twiki/bin/view/Main/CyberLinkForJava

[27] "Smack documentation." [Online]. Available: http://www.igniterealtime.org/projects/smack/

[28] Y. Wang, M. E. A. Matougui, and S. Leriche, "j-asd experiments," Institut Telecom ; Telecom SudParis, Tech. Rep., 2012. [Online]. Available: http://javact.org/JASD-rapport.pdf

[29] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-OSGi: distributed applications through software modularization," in *MIDDLEWARE2007*. Berlin, Heidelberg: Springer-Verlag, 2007.

[30] 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.