Semantic Deployment of Services in Pervasive Environments

Frédéric Le Mouël, Noha Ibrahim, Yvan Royon, Stéphane Frénot

▶ **To cite this version:**

HAL Id: inria-00394958
https://inria.hal.science/inria-00394958

Submitted on 12 Jun 2009

Semantic Deployment of Services
in Pervasive Environments

Frédéric Le Mouël, Noha Ibrahim, Yvan Royon, and Stéphane Frénot

INRIA ARES – CITI Lab. – INSA Lyon
21 Avenue Jean Capelle
F-69621 Villeurbanne Cedex, France
{frederic.le-mouel, noha.ibrahim, yvan.royon,
stephane.frenot}@insa-lyon.fr

Abstract. Pervasive devices, such as mobile phones or personal gateways, allows more and more to execute personal services and not only legacy services. This opportunity raises the challenge of deploying heterogeneous services non-previously considered in the environment.
In this paper, we present an architecture for the semantic deployment of services in a pervasive environment. This architecture, based on a middleware distributed on each device, specially includes a *Deployment Service* interacting discovery and interoperability middleware services. This *Deployment Service* takes into account the semantic description of services and the semantic description of deployment itself to apply a semantically and timely local deployment strategy.

1 Introduction

Pervasive devices, such as mobile phones or personal gateways, are fast-expanding and become part of our every day life. More and more, they allow to execute personal services and not only legacy services. This opportunity raises the challenge of deploying heterogeneous services non-previously considered in the environment.

In this article, we present an architecture for the semantic deployment of services in a pervasive environment. This architecture relies on a middleware distributed on each device. This middleware provides a *Deployment Service* interacting with discovery and interoperability middleware services. This *Deployment Service* takes into account the semantic description of services and the semantic description of deployment itself to apply a semantic deployment strategy.

This work is part of the ongoing European integrated project: IST Amigo – Ambient Intelligence for the Networked Home Environment [1].

First, section 2 presents a scenario illustrating the semantic deployment needed. Then, we detail in section 3 our core architecture, with its semantic description of deployment and its semantic strategy of deployment. In section 4, we give a short overview of related works and, in section 5, we finally conclude and discuss open research issues.

2 Deployment Scenario

For the annual meeting of their company, a group of businessmen arrives at a central conference building. When they come in equipped with their PDAs and laptops, the whole set of company's communication services is deployed in each meeting rooms previously booked by the company. This set is composed of the company's security service, the company's mailer service and the company's VoIP service. First, for their coordination meeting, they choose a great meeting room. When they come into the room, PDAs and laptops of speakers receive input ant output services controlling the display devices of the room. After one hour talking, one of the attending businessman decides to give a detailed presentation and to conduct a talk about a specific topic. Then, a sub-group decides to move to a nearby smallest conference room. When this sub-group comes into the small meeting room, input ant output services on PDAs and laptops of speakers do not migrate but are changed for appropriate ones provided by the new room. To keep connected with the other room, two additional services using company's communication services are deployed in both rooms: the shared data storage service and the collaborative blackboard service. After one working day, businessmen leave the conference building with, on their laptops, the company's services and without building's input ant output services.

3 Semantic Deployment

The previous scenario shows the need to introduce semantic for the description of different services and for the description of the deployment of these services.

In the first section, we define the concept of semantic and the semantic level that our approach will reach. Then we present a middleware architecture for the semantic deployment of services. In sections 3.3, 3.4 and 3.5, we detail an ontology for the semantic description of services, an ontology for the semantic description of the deployment and the semantic strategy of deployment.

3.1 Semantic Levels

It is often necessary to establish a relation between two or n services. Formulas 1 and 2 show for instance the matching relation between services S_1 and S_2. In a basic level, this equivalence relation can be based on an interface matching (method's signatures accordance). In the semantic level, this equivalence relation is based on service's concept matching. This concept matching consists in ontology graph matching.

$$S_1 \leftrightarrow_{matching} S_2 \quad if \quad Interface_{S_1} \leftrightarrow_{method} Interface_{S_2} \tag{1}$$

$$S_1 \leftrightarrow_{matching} S_2 \quad if \quad Concept_{S_1} \leftrightarrow_{ontology} Concept_{S_2} \tag{2}$$

As shown in formula 3, we consider in our approach the deployment relation between services S_i and platforms P_j with deployment ontology matching.

$$S_i \leftrightarrow_{deployment} P_j \quad if \quad Concept_{S_i} \leftrightarrow_{ontology} Concept_{P_j} \tag{3}$$

3.2 Middleware Architecture

Usually, the deployment of services is statically decided in an unchanging way before running an application. The innovation of our approach is that it provides a dynamic deployment that goes along with the dynamic nature of the environment and a semantic deployment that takes into account the nature of services/platforms and the nature of the context present at a time being.

We consider that services are gathered in an Amigo middleware layer. This middleware layer resides on every hosts present in the environment. The application layer uses the services of the middleware and the services of the middleware rest on the system layer which provides the hardware and connectivity management.

The semantic deployment functionality is provided by the *Dynamic Service Deployment* service (cf. figure 1). Our service is in charge of dynamically downloading or uploading services to platforms of the environment. It interacts with other high-level services of the middleware such as the *Enhanced service Discovery* that enables a semantic discovery and the *Service Matching Tool* that allows communication through heterogeneous services. These high-level services relies on more classical services, the *Discovery Service* and the *Interoperability Service* for the discovery and the protocol transformation. Communication interfaces of these services are defined by a *Service Applications Programming Interface* family (SAPI), which is a set of possible interfaces such as Amigo interfaces, standardized interfaces (UPnP, etc) or legacy interfaces.
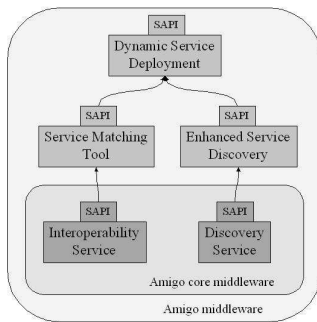
Fig. 1. *Dynamic Service Deployment* service in the Amigo middleware
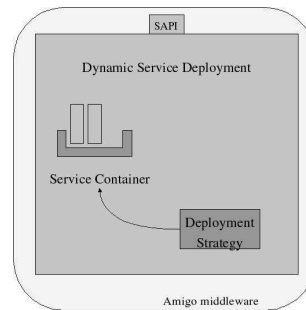
Fig. 2. Internal architecture of *Dynamic Service Deployment* service

The figure 2 describes the internal architecture of the *Dynamic Service Deployment* service:

– *Service Container*: the *Service Container* stores current services executing on the current platform. This container can be filled locally by the current platform or remotely by other *Dynamic Service Deployment* services.

- *Deployment Strategy*: the *Deployment Strategy* is in charge of deciding of the target of the deployment (which services have to be deployed ?), the destination of the deployment (where services have to be deployed ?) and the duration of the deployment (how long do these services have to stay there ?).

3.3 Semantic Description of Services

The first step to semantically deploy services is to have a description of services.

The ontology [2] chosen for our service description is given below. It is based on standard device property descriptions and tries to be generic and to cover also implicit information left out of those classifications.

Figure 3 is a part of this ontology and illustrates that a service publishes its description and can be provided by different kind of devices:
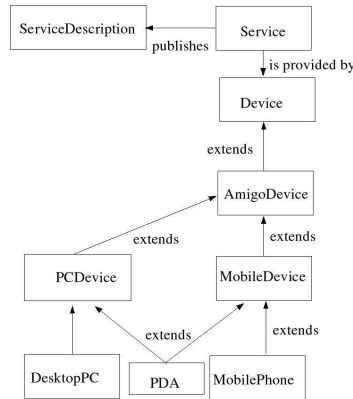

Fig. 3. Amigo service/device relationship ontology

Our service ontology is independent of any service model and implementations and can be applied to EJBs [3], CORBA Components [4], Fractal components [5], OSGi^{TM} bundles/services [6] or Web services [7].

Our service description is also independent of any description languages and can be written in different standard description languages such as WSDL [8], UDDI/XML [9] or OWL [10].

3.4 Semantic Description of Deployment

The second step for the semantic deployment of services is to define the description of the deployment itself (cf. figure 4).

As we have seen in the previous section, each service has its own semantic description. In the same way, each execution platform has its own semantic
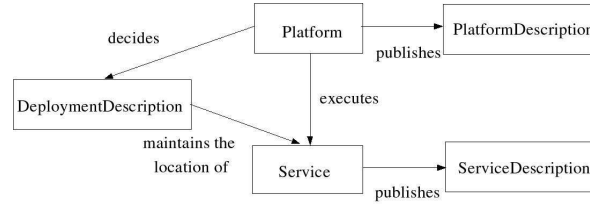
Fig. 4. Amigo platform/service ontology

description. The important description that we add is the semantic deployment description. This description is attached to the platform and links the different services to the platforms they are running.

This semantic deployment description can be instantiate in various different ways that we have also described in an ontology:
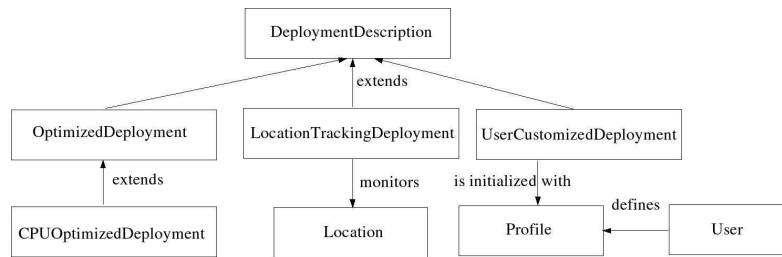
Fig. 5. Deployment description ontology

For instance, figure 5 shows that the deployment can be optimized according to resources constraints (such as CPU, memory, etc), or can apply migrations to follow a user or can be customized by the user.

3.5 Semantic Strategy of Deployment

The last step of our semantic deployment is done by the semantic deployment strategy. This strategy receives all services and platforms descriptions changes from the *Enhanced Service Discovery*. It is then in charge of modifying/updating the semantic deployment description. These modifications/updates consist in (re)deploying some services. So the crucial choices are which services to redeploy, where to redeploy and how long do we redeploy.

Because Amigo applications are running in highly dynamic environments, we choose this solution:

- **Local targets**: each Amigo platform only deploys services that are locally currently running. These services are the ones stored in the *Service Container*.

- **Context destination**: each Amigo platform do not migrate the services to a predefined or a specific other platform but migrates the services to the environment. This functionality is possible because of the use of our semantic description. For example, we do not specify any more a migration to an url `http://ares-lemouel-1:10000/` but to an `OSGiPlatform`.
- **Timely-limited duration**: the deployment of a service is stamped with a duration. During this period, the service can no more be redeployed. This guarantee prevents boomerang and domino effects we can observe in distributed systems.

4 Related Work

Effective and universal service access requires deployment of services on a highly mobile and dynamic environment, the pervasive environment. We present three groups of deployment works: static, dynamic and semantic deployment approaches.

Self-Deployment and Self-Configuration of Pervasive Network Services [11] presents a policy-based approach to achieve self-deployment and self-configuration of pervasive systems and services. In this approach, applications are decomposed into a set of interconnected components. The application components and how they interact are described using a work-flow and it can be expressed into Autonomia self-deployment policies. With each component, the autonomic attributes required to self-configure and self-manage the operations of the components in Autonomic Component Architecture (ACA) are specified. The Autonomia policy for self-deployment is expressed in IF-condition-THEN format. Self-Deployment service is to automate deployment of applications.

FORMAware [12] is a framework that combines a novel component-based programming model enhanced by a reflective design. In addition, FORMAware provides a set of tools and services for automating software development and adaptation (i.e. support for the generation, assembly, deployment and dynamic reconfiguration processes). Deployment relies on the notion of delegate components which are ambassadors of composites running on remote machines. Each composite uses a factory for creating delegates which then allow them to execute visitor operations to remotely run, kill, plug and unplug components.

Autonomic service deployment in networks [13] introduces a two-phase mechanism that achieves an efficient and flexible deployment of services in networks: a macro-level deployment phase that operates in a hierarchically distributed manner to query and collect capabilities of nodes in the network, and then execute the deployment itself; and a micro-level deployment phase that refines the actual installation of a service according to the specific capabilities of each element comprising the network node.

The pervasive nature of the environment requires a dynamic strategy of deployment of services. Many works focus on ways to dynamically deploy services.

A Generic Active Service Deployment Protocol [14] proposes an architectural framework for the dynamic deployment and control of active services in a heterogeneous network environment. This approach particularly defines the central

primitives and fundamental functionalities that a generic service interface should support and presents the Active Service Deployment Protocol (ASDP) that allows to accommodate and integrate different active service loading methods, irrespective of the service paradigm.

Dynamic Web Service Deployment Using WSPeer [15] describes how WSPeer, a framework for deploying and invoking Web services, allows application code to host and invoke Web services. The mechanisms described differ from usual Web service deployment scenarios in which a service is deployed into a container framework which manages the service environment. Instead WSPeer leaves control in the hands of the application. This allows Web services to be created and deployed that can respond dynamically to application requirements. WSPeer allows an application to maintain control over service life-cycle, state and invocation by adding service capabilities to the application rather than requiring the application to be deployed into a container environment. This makes WSPeer an ideal platform for extending the possibilities of Web service deployment and invocation. In particular it enables Web services to be used in P2P contexts.

If deployment of services in pervasive environments is answered enough, semantic deployment is not. Usually semantic is attached to the description of services as in Web Services [7] and not to the deployment strategy.

Requirements for Software Deployment Languages and Schema [16] describes software systems and components in a complete and rigorous manner. This description is required for the creation of a general infrastructure to support the software deployment life cycle. Such a software description definition must include ways to describe system assert constraints, dependency constraints, artifacts, configurations, and specialized deployment activities. Combining such a definition with a semantic description of consumer sites makes it possible to create general solutions to the various deployment tasks. The main purpose is to create a standard, rigorous schema for describing software systems and consumers sites and to create a software deployment framework to utilize these descriptions. The approach taken by the Software Dock is to develop a distributed framework where various agents are used to interpret semantic deployment information and then to automate the software deployment life cycle tasks.

5   Conclusion and Discussion

In this paper, we present an architecture for the semantic deployment of services in a pervasive environment. This architecture, based on a middleware distributed on each device, specially includes a *Deployment Service* interacting with discovery and interoperability middleware services. This *Deployment Service* takes into account the semantic description of services and the semantic description of deployment itself to apply a semantically and timely local deployment strategy.

Our approach can still be improved and several research issues can be discussed:

 – **Semantic requirement**: introducing semantic description for each part of our system (service, platform, etc) increases the development difficulties.

Some services do not require to have such high semantic description level, but can require a lower one such as LDAP filters on service properties. Intermediate or optional solutions have to be explored.

– **Local strategy versus global strategy**: having local deployment strategies allows a great autonomy of each platforms but increases a lot services administration. Having a global view of services currently deployed is often very useful but introduces consistency problems. Intermediate strategies have also to be explored.

– **Security**: in very changing environments, interacting with malicious platforms is possible. Allowing to push and pull services is so very dangerous. Taking into account the service semantic can also be a criteria to allow or not the deployment of this service.

References

1. Georgantas, N., ed.: Detailed Design of the Amigo Middleware Core: Service Specification, Interoperable Middleware Core. Delivrable D3.1b, Amigo project (2005)
2. Kalaoja, J., ed.: Detailed Design of the Amigo Middleware Core Service Modelling for Composability. Delivrable D3.1a, Amigo project (2005)
3. Monson-Haefel, R.: Entreprise JavaBeans. O'Reilly & Associates (2000)
4. Zahavi, R.: Entreprise Application Integration with Corba Component and Web-Based solutions. Johne Wiley & sons (1999)
5. Bruneton, E.: Developing with Fractal. The ObjectWeb Consortium, France Telecom (R&D). (2004) version 1.0.3.
6. OSGIalliance: About the OSGI service platform. Technical report, OSGI alliance (2004) revision 3.0.
7. Iverson, W.: Real Web Services. O'Reilly (2004)
8. Walsh, A.E.: UDDI, SOAP and WSDL: The Web Services Specification Reference Book. Pearson Education (2002)
9. UDDIForum: Universal Description, Discovery and Integration (2005)
10. Members, W.: OWL Web Ontology Language. Technical report, W3C (2004)
11. Chen, H., Hariri, S., Kim, B., Zhang, Y., Yousif, M.: Self-deployment and self-configuration of pervasive network services. IEEE/ACS International Conference on Pervasive Services (ICPS'04) (2004)
12. Moreira, R.S., Blair, G.S.: Supporting adaptable distributed systems with FOR-MAware. In: 4th International Workshop on Distributed Auto-adaptive and Reconfigurable Systems. (2004) Edinburgh, Scotland.
13. Haas, R., Droz, P., Stiller, B.: Autonomic service deployment in networks. IBM Systems Journal **42**(1) (2003)
14. Sifalakis, M., Schmid, S., Chart, T., Hutchison, D.: A generic active service deployment protocol. In: 2nd International Workshop on Active Network Technologies and Applications (IECE'03), Osaka,Japan (2003) 100–111
15. Harrison, A., Taylor, I.: Dynamic Web Service Deployment Using WSPeer. In: 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies, Louisiana State University (2005) 11–16
16. Hall, R.S., Heimbigner, D., Wolf, A.L.: Requirements for software deployment languages and schema. Lecture Notes in Computer Science **1439** (1998)