



# An Ontology-based Approach to Semantically Deploy Services in Pervasive Environments

Hajer Chamekh and Frédéric Le Mouël  
ARES INRIA/CITI, INSA-Lyon,  
F-69621, FRANCE  
hajer.chamekh, frederic.le-mouel@insa-lyon.fr

**Abstract**—The development of highly dynamic and pervasive environments has led to a proliferation of services coming from different providers. These services are usually deployed independently of the context, platforms and devices present in the environment. We propose an ontology-based approach to take advantage of semantic to enable an optimized deployment of services in pervasive environments. We realize a deployment system which takes into account semantic description of services, semantic description of environment and semantic description of deployment itself to apply deployment strategies. Our semantic deployment is based on the Ontology Web Language for Services (OWL-S). We enrich the OWL-S with devices and platforms considerations.

**keywords:** services, components, deployment, semantic, OWL-S, ontology.

## I. INTRODUCTION

The development of highly dynamic and pervasive environments has led to a proliferation of services coming from different providers. These services are usually deployed independently of the context, platforms and devices present in the environment. This opportunity raises the challenge of deploying heterogeneous services non-previously considered in the environment. Also, deploying services in pervasive environments needs to be more autonomous and sensitive to context.

In this domain, one of the most challenging objectives to be achieved is to dynamically deploy services in pervasive environments. A number of research efforts have been conducted in this area [5], [2], [8]. However, most solutions didn't consider the context and the description of service and deployment. While dynamic deployment allows some degree of autonomy, semantic descriptions of services and environments allow user to be more sensitive to the environment's changes, leading to a higher level of autonomy and adaptation. Indeed, common semantic description resolves the problem of the heterogeneous services and devices. We propose a service-oriented architecture for semantic deployment of services taking into account semantic description of service, environment and deployment itself. We define a service as a software unit able to be published, discovered and reused. We apply semantic deployment strategies to deploy services. Our semantic deployment is based on the Ontology Web Language for Services (OWL-S) [14], which is one of the most prominent efforts for describing semantic Web Services. We apply OWL-S to describe services and devices.

We propose a dynamic deployment service which interacts with other services (service discovery, service matching...) in order to dynamically deploy services with semantic. This work is a part of the ongoing European integrated project: IST Amigo Ambient Intelligence for the Networked Home Environment [4].

The remainder of this paper is structured as follows. In section 2, we present a scenario illustrating the semantic deployment. Then, in section 3 we survey some research efforts in the area of deployment, and semantic deployment. In section 4, we present the semantic description of services with OWL-S. We describe our approach to semantic deployment in section 5, with its semantic description of service, environment and deployment. In Section 6, we present the implementation of our deployment system. Finally, we conclude with a summary of our contribution and future works in section 7.

## II. SCENARIO

Tom is in the airport, his flight has been delayed for three hours. He wants to take some notes but he has not a text edition service on his mobile phone. So he uses a deployment service that he has on his mobile phone to deploy the available text edition service. He gives a description of the service "text edition" and a description of the environment "airport waiting room". Unfortunately, there are not any suitable services semantically discovered in the environment. Tom decides to watch a movie, using a video service he has on his mobile phone, but due to its limited resources, he cannot use it. He gives a semantic description of service "watch movie" and a semantic description of environment "airport waiting room" to his deployment service. This deployment service has the ability to semantically discover the services and the devices available in his reach, to select the most appropriate service and the most appropriate device and to deploy the service in this device. After device owner permission, it deploys Tom's video service in Hans's laptop and displays the movie in the nearest screen to Tom. While Tom is enjoying the movie, some passengers come in the waiting room. He is notified about the existence of a "text edition" service. After service owner permission, the deployment service deploys it in Tom's mobile phone so Tom stops the movie and begins taking notes.

### III. RELATED WORKS

In this section, we survey various systems of deployment. We compare these systems according to different criteria we are interested in pervasive environments :

- Dynamic deployment.
- Semantic description of the deployment unit.
- Semantic description of deployment.
- Context description.
- Deployment strategies.

Fig 1 presents a comparative table of all the deployment systems.

Various deployment system were defined, the most prominent ones being .NET [1], Entreprise JavaBeans [3] and the CORBA Component Model [2]. They are based on component middlewares. However, They provide a static deployment, which do not manage description of deployment unit, deployment itself and context.

WSPeer [8] is a framework for deploying and invoking Web Services. It allows applications to expose themselves or parts of themselves as Web Services. This mechanism leaves control in the hands of the application which allows dynamic deployment. However, it doesn't consider semantic description of services, deployment and context. It neither apply deployment strategies. Whereas, [12] which describes an architecture for automatic deployment of component-based applications on computational grids apply a deployment strategy using a deployment plan.

COMCAS [5] allows description of context to which services are sensitive. It is based on an ontology description which provides an efficient support for dynamic description of context, interpretation rules, and adaptation policies. [6] proposes a schema for the just-in-time deployment of component-based applications which supports context-aware adaptation. This schema presents a deployment plan which specifies how to create component instances of the application to deploy, where to instantiate them, and how they are connected to each other according to context. Both of these systems provide solutions to the dynamic deployment of component and take into account the context. They also apply deployment strategies. However, they don't consider the semantic description of the deployment unit and the deployment itself.

[15] proposes an agent deployment model based on the CORBA Component Model. [20] proposes an approach for dynamic adaptation based on agents. They allow dynamic deployment and apply migration as deployment strategy. However, they don't take into account semantic description and context.

[16] defines and explores what is required in a software description language or schema in order to support software deployment. It combines software product knowledge with consumer site knowledge in order to get generic solutions to software deployment tasks. However, it doesn't manage context and doesn't apply deployment strategies.

[19] proposes a matchmaker to enhance UDDI by service capability matching. [21] proposes a peer to peer based

mobile environment consisting of stations providing semantic services and users with mobile devices which manage their owner's semantic profile. Both of these works propose dynamic deployment of services and take into account the semantic description of services. However, they don't consider the semantic description of deployment and they don't propose deployment strategies.

System	Deployment Unit	Semantic Description		Context	Strategy	Dynamic Deployment
		Unit	Deployment			
[1], [3], [2]	Component	-	-	-	-	-
[8] WSPeer	Service	-	-	-	-	+
[12]	Component	-	-	-	+	+
[15], [20] ESSMA	Agent	-	-	-	+	+
[6], [5] COMCAS	Component	-	-	+	+	+
[16]	Component	+	+	-	-	+
[19]	Service	+	+	-	-	+
[21]	Service	+	-	+	-	+

Fig. 1. Comparison of deployment systems

By analysing this table, we find that there is no system deployment which insures all the pervasive environment criteria previously specified. Thus, we propose a deployment system which insures all these criteria.

### IV. SEMANTIC DESCRIPTION: OWL-S

Semantic description of services results from the combination of the Semantic Web and Web Services. A number of research efforts has been conducted in order to bring semantics to Web Services [10], [9], [7]. These efforts aim at the semantic specification of Web Services towards automating Web Services discovery, invocation and execution monitoring. In this area, the Ontology Web Language for Services (OWL-S) is the most complete effort to describe services that can be expressed semantically. OWL-S is a Web Service ontology specified in OWL [13] for describing semantic Web Services. It is composed of three parts: the service profile, the service process and the service grounding, fig 2 presents the OWL-S service ontology.

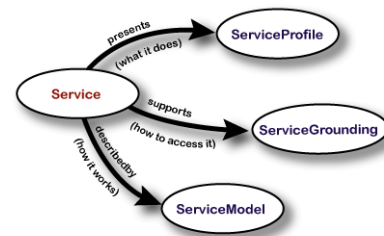


Fig. 2. Top level of the service ontology

- *Service Profile* gives a high-level description of a service and its provider. It is used for service publication and discovery. Service descriptions are constructed from a description of functional properties (i.e. inputs, outputs, preconditions, and effects -IOPEs), and non-functional properties (i.e. service name, text description).

- *Process Model* describes how the service works. Services can be described as a collection of atomic, simple or composite processes. Simple process is used as element of abstraction. Atomic process is a single, black-box process description with exposed IOPEs. Each atomic process is mapped to WSDL operation. Composite process is consisted of other simple, atomic or composite processes. These processes are constructed using different composition constructs, including: Sequence, If-then-else, Split, etc.
- *Service Grounding* provides a pragmatic binding between this concept space and the physical data/machine/port space, which facilitates the service execution. The OWL-S service grounding is based on WSDL.

The OWL-S is commonly applied to describe Web Services. In our work, we apply it to describe services and devices. A device contains the platform which executes the service.

## V. SEMANTIC DEPLOYMENT SYSTEM

In this part, we present the architecture of our Semantic Deployment System. Then, we propose a semantic deployment description of services and environments followed by a description of deployment strategies.

### A. Semantic Deployment Architecture

Each service has its own semantic description. In the same way, each execution platform has its own semantic description. The semantic description of service and the semantic description of platform are registered in the *ContextManager* which provides Semantic context description. Our main service is the *DynamicDeploymentService* which allows to deploy services by taking into account the semantic description of service and the semantic description of context. It interacts with other services to insure this functionality. Fig 3 presents the components of our system.

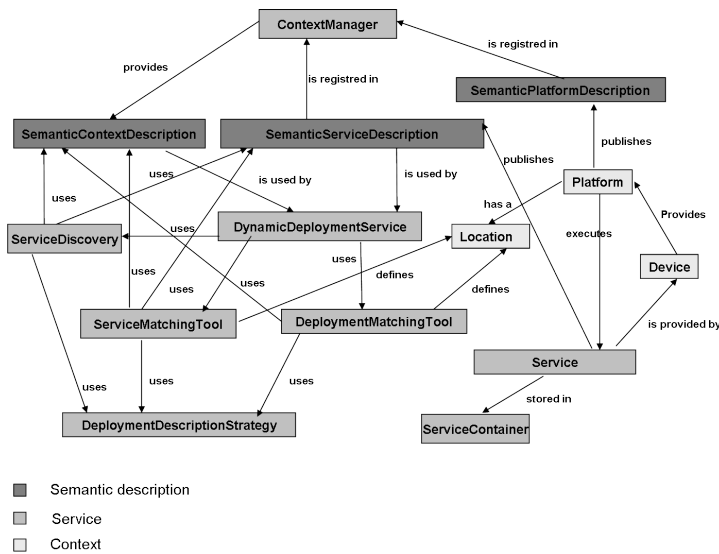


Fig. 3. Deployment System Architecture

- *ServiceDiscovery*: allows the selection of services providing descriptions semantically equivalent to the descriptions requested in the target user task and his environment.
- *ServiceMatchingTool*: allows to choose the service that matches the best with the service requested semantic description and the environment semantic description.
- *ServiceContainer*: stores current services executing on the current platform. This container can be filled locally by the current platform or remotely by other *DynamicDeploymentService*.
- *DeploymentDescriptionStrategy*: provides strategies of deployment in order to choose the service to deploy and the destination of the deployment.
- *DeploymentMatchingTool*: uses the *DeploymentDescriptionStrategy* to find the location where to deploy service using the semantic description of the environment.

### B. Semantic Deployment Description

1) *Semantic Description of Services*: The OWL-S profile and process models provide semantic frameworks where services can be discovered and invoked. Thus, we used the profile and process model to semantically describe services. An OWL-S ontology of the video service in our scenario is presented in fig 4. The video service presents a profile which

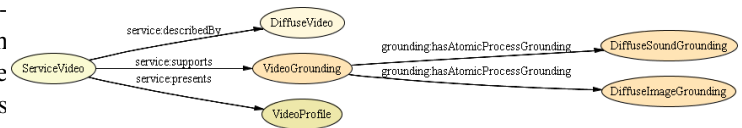


Fig. 4. Video Service Ontology with OWL-S

describes the serviceName, a textDescription and IOPEs. It is described by a process which contains a composite process "DiffuseVideo". This composite process is formed by two atomic processes "DiffuseSound" and "DiffuseImage". To semantically deploy services, we use first the service profile. If it is not sufficient to discover and match services, we use the process model. A part of the profile and process descriptions generated for the previous OWL-S ontology are presented in fig 5 and 6.

```

<!-- Profile description -->
<profile:Profile rdf:ID="VideoProfile">
  <service:presentedBy>
    <service:Service rdf:ID="ServiceVideo">
      <profile:serviceName rdf:datatype="...">diffuse video</profile:serviceName>
      <profile:textDescription rdf:datatype="...">show see watch</profile:textDescription>
      <profile:hasOutput rdf:ID="Stream">
      </profile:hasOutput>
      <profile:hasInput rdf:ID="VideoResource">
      </profile:hasInput>
    </service:Service>
  </service:presentedBy>
</profile:Profile>
    
```

Fig. 5. Video profile description

```

<!-- Process description -->
<service:describes rdf:resource="#ServiceVideo"/>
<process:CompositeProcess>
  <process:CompositeProcess rdf:ID="DiffuseVideo">
</process:CompositeProcess>
<process:process>
  <process:AtomicProcess rdf:ID="DiffuseSound"/>
</process:process>
<process:process>
  <process:AtomicProcess rdf:ID="DiffuseImage"/>
</process:process>
<process:Input rdf:ID="VideoResource">
  <process:parameterType rdf:datatype="...">DigitalResource</process:parameterType>
</process:Input>
<process:Output rdf:ID="Stream">
  <process:parameterType rdf:datatype="...">VideoStream</process:parameterType>
</process:Output>

```

Fig. 6. Video process description

2) *Semantic Description of Environment*: Our *DynamicDeploymentService* is in charge of dynamically downloading and/or uploading services to platforms of the environment. For uploading services, each platform do not migrate the services to predefined or specific platforms but migrates the services to the environment. For downloading services, each platform do not download services from a specific platform but from the environment. These functionalities are possible because of the use of our semantic description. We use The OWL-S profile to semantically describe devices. This description includes the description of platforms. Each device has a profile description. A part of the profile description generated for a laptop device used in our scenario is presented in fig 7.

```

<!-- Profile description -->
<profile:Profile rdf:ID="LaptopProfile">
<profile:serviceName rdf:datatype="...">laptopdevice</profile:serviceName>
  <service:presentedBy>
    <service:Service rdf:ID="Laptop">
      <service:presents rdf:resource="#LaptopProfile"/>
    </service:Service>
  </service:presentedBy>
<profile:textDescription rdf:datatype="...">airport device, Linux,OSGi</profile:textDescription>
</profile:Profile>

```

Fig. 7. Laptop profile description

### C. DeploymentDescriptionStrategy

As we have seen in the semantic deployment description, each service has its own semantic description and each device has its own semantic description. We add the semantic deployment description which can be instantiated in various ways that we have described in fig 8:

- **OptimizedDeployment**: allows deployment optimisation according to the resources constraints (CPU, memory, bandwidth, etc).
- **LocationTrackingDeployment**: allows migration of services to follow the user when he moves.
- **UserCustomizedDeployment**: the user defines his preferences in his profile. The service will be deployed according to this profile.

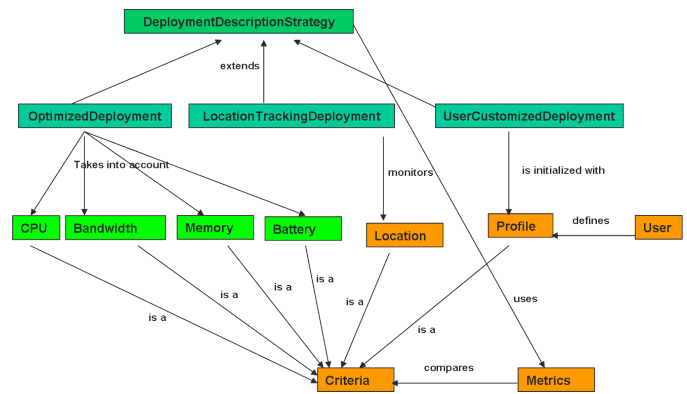


Fig. 8. Deployment Description Strategy

## VI. FIRST PROTOTYPE

### A. Tools

Protégé [11] is a platform that provides a suite of tools to construct domain models and knowledge-based applications with ontologies. The support for OWL in Protégé is very useful with the OWL plugin. OWL-S Editor [17] is implemented as a plugin to the Protégé OWL ontology editor. It allows to easy develop OWL-S services.

OWL-S API [18] is a Java library for working with OWL-S service description. It provides a high level of abstraction. It was designed to let programmers access and manipulate OWL-S service description easily. The OWL-S API provides a data model for services that reflects the structure of the OWL-S model. It also provides Java Interfaces and methods which were designed in conjunction with the classes and properties defined in the OWL-S ontologies.

### B. Implementation

OWL-S API allows us to manipulate this OWL-S description and generates many Java classes according to the OWL-S specification. In our case, we use the methods which allow to get the semantic description of services and devices. We use these methods to push (upload) and pull (download) services. Fig 9 presents our functions using OWL-S API.

The *DynamicDeploymentService* uses the semantic description of services and devices provided by the OWL-S API classes. Then it queries the *ServiceDiscovery* to discover the services according to these semantic descriptions. The *ServiceDiscovery* returns a list of services discovered in the environment. The *DynamicDeploymentService* also queries the *ServiceDiscovery* to search the local or global strategies. So the *ServiceDiscovery* returns first the local strategies. If there is no local strategies, it returns global ones. Then the *DynamicDeploymentService* queries the *ServiceMatchingTool* to choose a service from the provided list. The *ServiceMatchingTool* returns a service location according to the semantic descriptions of service and environment, the list of services and the strategy of deployment. Finally, The service is invoked from a distant *ServiceContainer* and stored in the local *ServiceContainer*. In case of uploading service, the *DynamicDeploymentService*

```

\\reading service description
Service service =
kb.readService("http://www..../VideoService.owl");

\\reading device description
Device device =
kb.readDevice("http://www..../VideoDevice.owl");

semanticDescriptionService =
service.getTextDescription();
...
semanticDescriptionDevice =
device.getTextDescription();
...

push(semanticDescriptionService,
semanticDescriptionDevice);
...
pull(semanticDescriptionService,
semanticDescriptionDevice);
...

```

Fig. 9. Semantic Deployment functions using OWL-S API

queries the *DeploymentMatchingTool* to choose the destination where to deploy the service. The service is invoked from the local *ServiceContainer* and stored in a distant *ServiceContainer*.

So to implement our scenario, we develop and use the following interface methods:

- *push(semantic description of service, semantic description of environment)* allows to upload a service according to its semantic description and the semantic description of the environment. According to our scenario, we use the push function in case of deploying a video service from Tom's mobile phone to the environment. For that, we instantiate *push("watch movie", "airport waiting room")*.
- *pull(semantic description of service, semantic description of environment)* allows to download a service according to its semantic description and the semantic description of the environment. According to our scenario, to deploy a text edition service from the environment to Tom's mobile phone, we use *pull("text edition", "my device")*
- *searchService(semantic description of service, semantic description of environment)* allows the discovery of services according to these semantic descriptions. In the scenario, we instantiate this function to search text edition services, so we use *searchService("text edition", "airport waiting room")*.
- *chooseService(semantic description of service, semantic description of environment, list of services, strategy)* allows to choose the service to deploy according to these descriptions. We instantiate *chooseService("text edition", "airport waiting room", "Microsoft office, Open office", "nearest to user")*.
- *chooseDestination(service reference, semantic description environment, deployment strategy)* returns the location where to deploy a service. In our scenario, Tom wants to deploy his video service in the environment, so we instantiate this function *chooseDestina-*

*tion("MediaPlayer", "airport waiting room", "nearest to user")*.

- *getService(service reference, URL)* allows to take the service from a distant device and stores it temporary in a buffer. The video service is taken from Tom's mobile phone using *getService("MediaPlayer", "http://...player")*.
- *storeService(service reference, URL)* stores the service in the local device. We instantiate *storeService("MediaPlayer", "http://...player")*.

Our deployment strategies are under development. If the user has local strategies, he can apply one of them. If there are no local strategies, the system searches global strategies in the environment to apply. For the moment, we use a static strategy, location strategy, so we choose the nearest device to the user to deploy a service. The methods related to deployment strategies need to be developed.

- *searchLocalStrategy(semantic description of environment)* returns deployment strategies in the local environment.
- *searchStrategy()*: If there is no local strategies, this method search deployment strategies on the global environment.
- *getStrategy(strategy URL)* returns the chosen strategy. If we have many strategies, the system chooses arbitrary.

## VII. CONCLUSION AND FUTURE WORKS

Our objectif is to semantically deploy services in pervasive environment. Most existing solutions to dynamic deployment poorly deal with semantic descriptions. We presented an architecture for the semantic deployment of services in pervasive environments. This architecture, based on a middleware distributed on each device, specially includes a Dynamic Deployment Service. This Dynamic Deployment Service takes into account the semantic description of services and the semantic description of environments to apply deployment strategies. For these semantic descriptions, we use the OWL-S service ontology that we enrich considering devices and platforms semantic. This solution offers great flexibility by enabling semantic discovery, matching, and deployment.

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

- Deployed services: after deploying services, two cases are possible. We leave the service in the execution platform or we destroy it after its use.
- Deployment strategies: 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 pervasive environments, interacting with malicious platforms is possible. Uploading (push) or downloading (pull) unknown services to platforms is risky since we do not know services providers neither services

behaviours. Taking into account the service semantic can also be a criteria to allow or not the deployment of this service.

## REFERENCES

- [1] Microsoft Corporation, An Introduction to Microsoft .NET. White Paper. 2001.
- [2] OMG.CORBA Components Version 3.0: An adopted Specification of the Object Management Group. 2002.
- [3] Sun Microsystems. Enterprise JavaBeans Specification 2.0. 2002.
- [4] Detailed Design of the Amigo Middleware Core: Service Specification, Interoperable Middleware Core. 2005.
- [5] Dhouha Ayed, Nabih Belhanafi, Chantal Taconet, and Guy Bernard. Deployment of Component-Based Applications on top of a Context-Aware Middleware. *In Proc. Mobile Computing Systems in Dynamic Environments Workshop, special session of the IASTED International Multi-Conference on Software Engineering*, 2005.
- [6] Dhouha Ayed, Chantal Taconet, and Guy Bernard. A Data Model for Context-aware Deployment of Component-based Applications onto Distributed Systems. *ECOOP Workshop on Component-oriented Approaches to Context-aware Computing, Oslo, Norway*, 2004.
- [7] Mark H. Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew V. McDermott, Sheila A. McIlraith, Srin Narayanan, Massimo Paolucci, Terry R. Payne, and Katia P. Sycara. DAML-S: Web Service Description for the Semantic Web. *International Semantic Web Conference*, pages 348–363, 2002.
- [8] Andrew Harrison and Dr Ian J.Taylor. Dynamic Web Service Deployment Using WSPeer. *In Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies.*, 2005.
- [9] Ian Horrocks. DAML+OIL: a Description Logic for the Semantic Web. *IEEE Data Eng. Bull.*, 2002.
- [10] Preeda Rajasekaran Amit P. Sheth Rohit Aggarwal John A. Miller, Kunal Verma and Kaarthik Sivashanmugan. WSDL-S: Adding Semantics to WSDL. *Technical Report*, 2001.
- [11] Holger Knublauch, Ray W. Ferguson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. *International Semantic Web Conference*, 2004.
- [12] Sébastien Lacour, Christian Pérez, and Thierry Priol. A Software Architecture for Automatic Deployment of CORBA Components Using Grid Technologies. *In Proceedings of the 1st Francophone Conference On Software Deployment and (Re)Configuration*, 2004.
- [13] Deborah L.McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. *W3C Recommendation*, 2004.
- [14] David Martin and all. OWL-S: Semantic Markup for Web Services. *white paper*, 2004.
- [15] Fabio Melo, Ricardo Choren, Renato Cerqueira, Carlos Lucena, and Marcelo Blois. Deploying Agents with the CORBA Component Model. *Component Deployment*, pages 234–247, 2004.
- [16] Alexandre L.Wolf Richard S.Hall, Dennis Heimbigner. Requirements for Software Deployment Languages and Schema. *Lecture Notes In Computer Science*, 1998.
- [17] Shahin Saadati and Grit Denker. An OWL-S Editor Tutorial Version 1.1.
- [18] Evren Sirin and Bijan Parsia. The OWL-S Java API. *Poster, In Third International Semantic Web Conference (ISWC2004), Hiroshima, Japan, November 2004*.
- [19] Tetsuo Hasegawa Massimo Paolucci Takahiro Kawamura, Jacques-Albert De Blasio and Katia Sycara. Public Deployment of Semantic Service Matchmaker with UDDI Business Registry. *Springer-Verlag Berlin Heidelberg*, 2004.
- [20] Giuseppe Valetto, Gail E. Kaiser, and Gaurav S. Kc. A Mobile Agent Approach to Process-Based Dynamic Adaptation of Complex Software Systems. *EWSP*, pages 102–116, 2001.
- [21] Andreas von Hessling, Thomas Kleemann, and Alex Sinner. Semantic User Profiles and their Applications in a Mobile Environment. *Artificial Intelligence in Mobile Systems*, 2004.