Mobile-Process-based Ubiquitous Computing Platform: A Blueprint

Holger Schmidt
Institute of Distributed Systems
Ulm University
Germany
holger.schmidt@uni-ulm.de

Rüdiger Kapitza
Dept. of Comp. Sciences
Informatik 4
University of Erlangen-Nürnberg
Germany
rrkapitz@cs.fau.de

Franz J. Hauck
Institute of Distributed Systems
Ulm University
Germany
franz.hauck@uni-ulm.de

ABSTRACT

Mobile objects and agents are used for implementing distributed applications. Both concepts allow efficient use of local resources, volatile network connectivity and more efficient communication due to appropriate migration, especially in dynamic ubiquitous environments. Mobile processes enable specifying the complete life cycle of complex mobile applications.

In this paper, we propose a mobile-process-based platform that turns ubiquitous application development into a manageable task. We advocate the use of Web services for implementing mobile processes in a heterogeneous environment. Based on previous work, we sketch a novel approach for realising mobile context-aware Web services, which implement process steps. We use a separation of state, functionality and implementation code, which enables a reduction, expansion and transformation of the service state and functionality during migration.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Services—*Web-based services*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Design

Keywords

Web Services, Mobile Processes, Mobility, Ubiquitous Computing, Middleware

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MAI'07, March 20, 2007, Lisbon, Portugal
Copyright 2007 ACM 978-1-59593-696-7/07/0003 ...$5.00.

1. INTRODUCTION

There is a trend to integrate intelligence and connectivity into small devices. This is enforced by the idea of ubiquitous computing [1] that envisions a future in which people are unnoticeably supported by small, surrounding devices. These devices are able to interact by creating dynamic ad-hoc networks as well as by using a static network infrastructure. The result is a highly dynamic and heterogeneous environment with big cooperation potential.

Designing and implementing distributed applications for such dynamic environments is a complex task. Concepts such as mobile objects and mobile agents have been a step into the right direction. However, these technologies are not sufficient for implementing complex scenarios, as system dynamics has to be handled by the application logic. We think managing ubiquitous computing scenarios among other things requires support for heterogeneity, mobility and context-awareness. Additionally, there is a need for dynamic code deployment, basic security and support for complex application development using custom code generation.

In this paper, we outline an infrastructure for supporting ubiquitous mobile applications using mobile process descriptions. This simplifies the creation of an application's abstract design of behaviour and interactions with other applications. We briefly sketch a possible code generation process that is able to map these descriptions to concrete implementations of mobile context-aware Web services. These Web services implement the mobile process. As Web services are already an established technology in many areas, we expect them to become accepted in ubiquitous computing as well. Thus, interoperability among ubiquitous devices based on Web services can be achieved, which is especially important within a highly dynamic and heterogeneous environment.

In contrast to previous work [2], we introduce mobile, context-aware Web services, which are able to migrate and to adapt their functionality, state and implementation code with respect to the current run-time environment. This is achieved by a separation of the service's functionality (aka interface), state and implementation code, which allows to reduce, expand and transform a service's state and functionality at runtime according to the environment's requirements. As code cannot be assumed to be locally available at any node within a dynamic environment, hints for a dynamic deployment infrastructure are given in this paper.

The remainder of the paper is structured as follows. In the

next section, we give basic background information on object migration, dynamic loading of code and Web services. Then, in Section 3 our concept of mobile processes based on Web services is presented. In Section 4, we apply our concept to an exemplary application. After discussing related work in Section 5, Section 6 concludes.

2. BACKGROUND

This section first gives basic background information on object migration, especially on our CORBA-based solution. Then, as object migration might require dynamic loading of locally non-existent code, we describe our approach for dynamic code loading. Last, basic information on Web services, which are the basis of our concept for building applications in a ubiquitous environment, is presented.

2.1 Object Migration

There exists a lot of work regarding object migration, in particular in the area of mobile agents (objects with an autonomous activity). However, as MOA [3], Mole [4] or Aglets [5], these rely on native Java serialisation and are therefore restricted to a homogeneous environment. There exist solutions for heterogeneous environments, but these do not rely on standards and therefore are not interoperable among each other and do not support the development process, such as Agent Factory [6] and the approach of Peter and Guyennet [7].

In previous work, we presented a platform- and language-independent service for object migration [8]. By building on the CORBA middleware [9] and the CORBA Life-Cycle Service (LCS) specification [10], interoperability among different LCS-capable implementations is achieved. The LCS specification defines interfaces of mobile objects and required entities.

For migration, a mobile object has to support a defined `LifeCycleObject` interface that declares a `move` method. The developer has to provide the actual implementation of this method as the LCS only specifies interfaces. For determining the target location during a migration, the developer can use the *factory finder*, which maintains a repository of *generic factories*. These factories enable the (remote) creation of objects at the migration target location.

In order to migrate a mobile object, the current state and the implementation code have to be transferred to the new location. In a heterogeneous environment, different programming languages may use different implementations. For this reason, only *implementation-independent* state has to be transferred, such as a list of key-value pairs for a hash table object. Otherwise, the target implementation might not be able to interpret the transferred state. Therefore, *value types* are used, which are special CORBA objects that are transferred by value and allow the specification of the implementation-independent state using the standard CORBA *interface definition language* (IDL). Additionally, this results in less error-prone implementations as value types are marshalled and demarshalled transparently and the developer does not have to write methods for externalisation and internalisation. As code cannot be assumed to be locally available in a dynamic environment, code loading is an essential part of our LCS implementation (cf. Section 2.2).

On the basis of the LCS implementation, we developed a context-aware LCS implementation [11]. There, we introduced a separation of state, functionality and implementation code for mobile objects. A concrete instance of such a context-aware mobile object has a specific state, functionality (interface), and implementation code. We call this configuration a *facet* of the mobile object, which is specified using standard IDL. Whenever such an object migrates, it is able to adapt to the current target location context by changing its available state, functionality and implementation code. This allows, for example, a migration from a fully-fledged facet on a powerful machine to a restricted facet on a resource-limited device. At the same time, the object can migrate from an object facet implemented in Java to a C++ object facet on the target location.

For enabling an adaptation of state to the current context, we introduce a separation of *active* and *passive* state. Active state is available within the current mobile object facet, whereas passive state is not. Passive state has to be stored as it might be used again within another facet. For storing passive state, we introduce a state store entity.

We have a working prototype implementation that enables context-aware migration from C++ to Java and vice versa.

2.2 Dynamic Loading of Code

Dynamic code loading is an essential part of object migration, especially in a dynamic environment without guarantee of local existence of required code. Therefore, we developed a dynamic loading service (DLS) for CORBA [12]. This service enables an ORB-independent dynamic loading of platform-dependent code on demand for arbitrary functionalities based on standardised compatibility requirements. The DLS follows the client/server paradigm and uses dedicated servers to host the program code and to offer specific information about available code.

Based on this work, we developed a decentralised type of the DLS (P2P-DLS) [13, 14]. This service allows any participating peer to offer and to obtain platform-specific code in a dynamic and heterogeneous environment. We propose a generic decentralised dynamic loading infrastructure that is independent from the peer-to-peer infrastructure in use. The peer-to-peer infrastructure only has to support keyword search. By building on our generic concept, we developed a JXTA-based [15] service for dynamic code loading [14].

For supporting dynamic code loading within our LCS implementation, we integrated the DLS, as well as the P2P-DLS, into the generic factory, which resides on the target location (cf. Section 2.1). There, the (P2P-) DLS is queried for appropriate location-dependent implementations. It is able to discover and to load code on demand for instantiating object-specific factories. However, as the (P2P-) DLS is a self-contained generic service concept for dynamic code loading, it can be used for other middleware infrastructures and applications as well.

2.3 Web Services and BPEL

Web services are a well-known XML-based application-to-application communication technology that is built upon standard internet protocols [16]. It follows the service-oriented architecture (SOA) approach; functionality is only provided by services [17]. These services are addressed using standard communication protocols, provide a standardised interface/description and can be composed of available services.

Web services are identified by uniform resource identifiers

(URI) and allow remote invocation of methods. The service interface and protocol bindings are specified using the Web services description language (WSDL) [18]. There, the interface is bound to a particular message protocol that is used for accessing the Web service. Common Web services use the simple object access protocol (SOAP) [19], a transport-protocol-independent XML application.

By building on XML, Web services are independent from platform and programming language, which enables their use in a heterogeneous environment. Dynamic environments are supported, as discovery and binding of Web services are handled at run-time. For supporting the discovery, universal description, discovery and integration (UDDI) [20] can be used. UDDI offers a generic interface to XML-metadata-based discovery of Web services for a specific domain.

According to the SOA notion, Web services can be composed of Web services. However, this composition can have a complex internal structure. For easing application development, the business process execution language (BPEL) [21] allows to describe Web service interaction (orchestration). The BPEL description is a standard-WSDL-compliant description of a Web service with an interface. This description can be executed by a BPEL engine that is able to execute the internal interactions with other Web services (implementation of a business process). For example, BPEL defines the order of Web service method invocations and the exception handling. The BPEL process is externally offered as standard Web service.

3. PLATFORM FOR MOBILE PROCESSES BASED ON WEB SERVICES

In this section, we sketch an approach for a mobile-process-based platform for supporting ubiquitous computing applications. Such a mobile process allows specifying the behaviour and interactions of an application that is able to dynamically change its execution context as well as to adapt itself to the current environment. Within our platform, the execution context is characterised by key-value pair metadata (e.g., location data, locally available resources and interfaces). Mobile processes are implemented as mobile context-aware Web services, which are introduced in the next subsection. Then, an approach to dynamic code deployment is presented, as migration requires such a mechanism (cf. Section 2.2). After this, we present mobile process descriptions on the basis of BPEL and an automatic code generation approach that is able to map the BPEL description to concrete mobile context-aware web services. As security is a crucial part of dynamic ubiquitous environments, we give information on security within our system.

3.1 Context-aware Web Service Migration

We propose a generic concept for context-aware Web service migration on the basis of context-aware object migration (cf. Section 2.1). For this purpose, we introduce stateful Web services that have a globally unique identifier (GUID) that does not change for the whole life time (cf. Figure 2, line 9). These Web services can be adapted to the current context by changing the interface, the state and the implementation code. We call the current triple of interface, state and implementation code the mobile Web service *facet* and specify several entities within our architecture by their service interfaces and behaviour:

- The *mobile Web service* has to implement our defined `MService` interface. This interface provides methods for migrating, copying and removing the Web service (methods `move`, `copy`, `remove`).
- A *manager service* is responsible for handling the migration on the source location. This service provides methods for initiating a migration. A mobile Web service only has to provide a reference to itself and metadata about the desired target location to the manager service, which is able to handle migration afterwards.
- Migration targets are represented by *factory services*. They enable the creation and deployment of Web services on a remote server. In case of locally non-existent code, the factory service is able to dynamically load code on demand (cf. Section 3.2). Additionally, the factory sets the appropriate active state to ensure stateful migration (cf. Section 2.1).
- The *factory finder service* is able to locate appropriate factories at target locations. It implements a repository of factory services for a specific domain and stores information about creatable service interfaces and the factory context (e.g., physical location). Thus, the functionality of the factory finder service is comparable to UDDI [20]. However, the factory finder service eases implementing user-tailored query processing and ordering of results (e.g., best-fitting results first). Nevertheless, we are able to substitute our factory finder service with a basic UDDI service.
- A *state store service* allows storing passive state (cf. Section 2.1). It provides methods for storing and retrieving state for a specific Web service that is identified by a GUID. This service is responsible for a specific set of mobile Web services. In principle, the state store service can be implemented as a mobile service, which enables to take along the complete state with the mobile Web service. This allows local adaptation without communication for state retrieving and storing.
- For retrieving information about the current context, we specify a *context service* that provides this data. The data is used by different components, e.g., the mobile Web service is able to monitor the context for changes which might trigger migration. Additionally, the factory service is able to use the context for providing up-to-date information to the factory finder service. For better interoperability, we would like to specify minimal context that should be provided. However, the concrete classification and specification is subject to future work.

Web service migration is realised according to Figure 1. For migrating a mobile Web service, a `move` method is called (1). Within this method, the `move` method of the manager service has to be invoked with parameters for specifying the desired target location using non-functional properties and the actual Web service that has to be migrated (2). These non-functional properties have either been passed to the mobile service or have to be determined within the mobile service's `move` method, which has to be implemented by the developer. The manager service first stores the active state for later use to the state store service (2.1). For

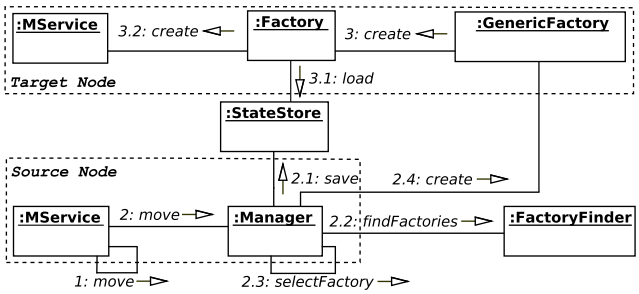

Figure 1: Context-aware Web service migration

introspecting the state of the Web service, a description of implementation-independent state has to be provided. For that purpose, we introduce a `states`-tag within the WSDL description (cf. Figure 2, line 10-12). Then, the factory finder service is queried for appropriate factory services (2.2). The manager service chooses the best-fitting factory service (2.3) and calls the creation method with the desired interface (2.4). The factory may load locally unavailable code by creating a Web-service-specific factory (3), which is able to create the desired Web service (3.2). For deploying the Web service, the state is first loaded from the state store service (3.1). In a last step, the Web service at the original location is removed.

We support different types of Web service migration:

- *Context-based*: Migration of a Web service caused by new requirements to the service context, for example, operation at another location with a specific IP address or on a specific device that has certain capabilities such as being a powerful machine. Additionally, a Web service might migrate onto a node because of there provided Web services for reducing communication costs.

- *Functionality-based*: Requirements to specific functionality can also cause migration. This allows implementing mobile Web services that are able to provide tailored functionality, e.g., a fully-fledged facet and a functionality-restricted facet. This enables an implementation of simple workflows [11]: The mobile Web service facets implement different roles, which can change during the workflow process.

These migration types are supported by our factory finder service. As already described, this service allows searching for migration targets using metadata information. For supporting both types of migration, the factory finder service has to offer the search for factories that provide specific context and for factories that enable the creation of facets with specific functionality.

For continuously addressing the mobile Web service, we introduce only an interface for integrating a location service, as the realisation is application-specific. On the client-side, Web service invocations are intercepted and forwarded to the current location using the location service interface. A simple implementation forwards calls to the particular migration target, which—in case of several migrations—results in a forwarding chain that could easily be broken by crashed nodes. Therefore, a central location service that is able to

```
1  <wsdl:definitions xmlns:wsdl="...">
2    <wsdl:types>...</wsdl:types>
3    <wsdl:portType name="Test">
4      <wsdl:operation name="getX">
5        ...
6      </wsdl:operation>
7    <wsdl:service name="TestService">
8      <wsdl:port>...</wsdl:port>
9      <asm:serviceID value="1329345329" />
10     <asm:states xmlns:asm="...">
11       <state>x</state>
12     </asm:states>
13   </wsdl:service>
14 </wsdl:definitions>
```

Figure 2: WSDL description with specification of implementation-independent state

manage current locations of a defined set of mobile Web services can be introduced as well. Web services initially have to register their contact address at the location service and identify themselves using their GUID. Whenever a Web service changes its location, it has to notify the location service about the new location. This concept can be enhanced by a peer-to-peer approach, in which information about current Web service locations is published using JXTA for example.

3.2 Dynamic Code Deployment

For supporting dynamic environments, code loading is an essential part of our concept. This enables a migration of Web services on machines where the service code has not even been known before. We propose an integration of our previously presented decentralised dynamic code loading service (cf. Section 2.2). For a seamless integration of code loading, factory services should be enabled to use our code loading service. These factory services are able to identify the required code by the interface name, load this code on demand and then deploy the Web service.

Additionally, we propose an inclusion of the factory service node's neighbourhood context. This implies that local dynamic code loaders get information about their neighbourhood context from the context service. This would enable an optimised code loading, for example, from a machine with the best network connection.

3.3 Mapping Mobile Processes on Mobile Web Services

We propose to use BPEL for describing self-adaptive mobile processes. As BPEL is represented by a Web service, we propose to implement the self-adaptive mobile process using our context-aware mobile Web services.

For an extended process description, we allow the annotation of BPEL. This enables the specification of non-functional properties for the invocation context for specific Web service calls, such as the desired invocation location. These annotations have to be evaluated at runtime and can result in a context-based migration of the mobile process.

Additionally, we introduce an extension of BPEL that allows interface-based migration without specific invocation interaction. This enables the transformation of a mobile process from one into another facet and can be used for realising mobile workflows. We propose to add specific tags

to the BPEL description. <migrate> enables the migration of the mobile process and <copy> allows to create a copy of the process with an own ID. <clone> allows to clone the process with the same ID. For specifying the target location, all tags can be annotated with non-functional properties as described before. Additionally, human interaction with the self-adaptive mobile process can be specified using BPEL4People [22].

For supporting the developer, we advocate the use of automatic code generation. There exist code engines that are able to automatically evaluate standard BPEL processes and generate the execution code (e.g., IBM WebSphere Process Server [23]). Although we currently do not have a prototype, we think it is possible to generate migration code for our extended BPEL processes. For supporting context, we introduce a *context decision service*, which is able to select the factory service with the best-fitting context for given criteria (e.g., required interface or location). This enables automatic code generation, as calls to this service are static and only context data, which is represented by key-value pairs within our system, is dynamic and can be handled as parameters. Thus, as processing the context is a static call and migration is handled by invoking the manager service, the complete code for migration can be generated automatically by a tool before runtime. This results in the generation of implementation skeletons of Web service facets, in which developers are able to integrate the real service logic. As we propose a generic concept, there has to be a code generation tool that is able to create tailored code for particular Web service containers.

3.4 Security Considerations

The presented concept has security issues that have to be considered. As mobile Web services have to rely on the platform, there has to be some trust relationship between the entities. First, the target selection process has to be secured. As context is provided by the nodes, the factory finder service and the factory service have to trust each other. Otherwise, malicious nodes can provide wrong context, which is then used for migration. Moreover, malicious nodes can simply interfere with the selection process and thus reduce performance. This can be solved using digital certificates in combination with encrypted communication (e.g., using the secure sockets layer).

Additionally, a deployment with dynamic code loading enables the injection of malicious code. This issue could be solved by introducing digital certificates for the loaded code portions or by using a sandbox.

Furthermore, the state store service has to be secured by rejecting malicious nodes that try to override passive state. Especially the save method has to be secured, e.g., using web-service-specific credentials. However, for restricting data access from unauthorised entities, the load method should be secured by credentials as well.

These mechanisms provide basic security. However, additional security is still subject to future work.

4. EXAMPLE APPLICATION

Our described concept should provide an architecture for the development of flexible and dynamic applications for ubiquitous environments. As a case study, we think of a mobile reporter application. There, mobile reporters are able to spontaneously initiate some kind of workflow: Reporters enter data into a local Web service, which migrates onto the machine of a (mobile) reviewer, who checks the data, and then migrates on the machine of a publisher. Additionally, reporters should be able to become reviewers after a specific number of accepted reports.

Such an application can be designed and implemented using our concept as described in Section 3. First, the self-adaptive mobile process has to be specified using our description language. Then, a code generation tool creates skeletons for the different Web service facets. The developer only has to implement the pure application logic for each facet, as the code for the context-aware migration decisions at runtime is automatically generated by the code generation tool based on the mobile process description, which may include human interactions as well (cf. Section 3.3). Then, the implementations have to be registered at our dynamic code loading infrastructure for being available within the whole system. Last, the self-adaptive mobile process can be deployed and be started.

5. RELATED WORK

Hammerschmidt and Linnemann developed a service for stateful migration of a Web service [2]. This work enables a continuous addressing of the migratable Web service during the whole lifetime using local stubs. However, as migration is based on native Java serialisation, a language-independent application is not possible. Moreover, the target location is specified statically and adaptation of the service state, interface and implementation code at run-time is not supported.

Regarding mobile processes, there is work on routes for mobile agents. These routes of mobile agents are called *itinerary*. For example *Erfurth and Rossak* developed a system for planning shortest routes [24]. However, itineraries are only a collection of predefined machines, on which specific tasks should be executed. This enables the implementation of simple processes only, as it lacks a dynamic concept on the basis of fine-granular non-functional properties.

The *DEMAC system* addresses support of mobile processes in ubiquitous environments [25]. It aims at the exchange and the distributed execution of processes by means of abstraction from the underlying transport protocols. A custom process description language was developed. However, DEMAC does not provide continuous support for application development. Additionally, there is no support for adaptive migration and no support for dynamic code loading.

Binder et al. developed an architecture for the creation of ad-hoc processes [26]. These processes are executed on the basis of transactional mobile agents. These agents include the process description as an XML file, which contains the calls that have to be invoked. However, the architecture is restricted to Java, as there is no concept for an abstract state description, and does not offer adaptability of mobile processes.

Ishikawa et al. describe a system for support of Web service integration for pervasive computing [27]. In this work, processes are described using BPEL. Additionally, there is a proprietary behaviour description for mobile agents that are able to take along Web services on other machines (migration). However, context is supported insufficiently, only one attribute of context is supported: available Web services at the target location. Thus, mobile workflows cannot be realised. Moreover, there is no support for adaptation and the

system is restricted to Java.

In contrast to a mobile workflow management system as for example proposed by Satoh [28], which transfers documents, our system enables the complete migration of services. This enables an adaptation of the application to the current context and to instantiate the application on devices, which are not aware of the application in advance. Additionally, a workflow management system specifies the document format (state), whereas we specify the interface for collecting data. This provides much more flexibility and generality.

6. CONCLUSION

In this paper, we presented a generic concept for supporting the application development within ubiquitous computing scenarios. For supporting developers, we propose the use of self-adaptive mobile processes for application design. We showed that these processes can be mapped on mobile Web services and sketched the use of an automatic code generation tool that supports the actual implementation of the mobile processes. Implementing mobile processes through Web services enables continuous addressing of the current process step during the whole life-time. This enables an external interaction: Other services are able to access the Web service to implement their functionality.

In the future, the proposed concept will result in an implementation of a generic middleware platform for supporting the development process of ubiquitous applications. We will implement the reporter application, as described in Section 4, to show the feasibility of our approach. We are planning to improve our concept by addressing resource-limited mobile devices, such as mobile phones. Therewith, we will examine the minimum requirements on our platform in ubiquitous scenarios.

7. REFERENCES

[1] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.

[2] B. C. Hammerschmidt and V. Linnemann. Migratable Web Services: Increasing Performance and Privacy in Service Oriented Architectures. In *IADIS Int. J. on Comp. Scien. and Info. Sys.*, volume 1, pages 42–56, 2006.

[3] D.S. Milojicic, W. LaForge, and D. Chauhan. Mobile Objects and Agents (MOA). In *4th USENIX Conf. on OO Tech. and Sys.*, pages 179–194, Santa Fe, New Mexico, 1998.

[4] M. Strasser, J. Baumann, and F. Hohl. Mole: A Java based mobile agent system. *2nd ECOOP Works. on Mob. Obj. Sys.*, 1997.

[5] D.B. Lange and M. Oshima. Programming and Deploying Java Mobile Agents Aglets, 1998.

[6] F.M.T. Brazier et al. Agent factory: generative migration of mobile agents in heterogeneous environments. In *ACM Symp. on Applied Comp.*, pages 101–106, Madrid, Spain, 2002.

[7] Y. Peter and H. Guyennet. Object mobility in large scale systems. *Cluster Comp.*, 3(2):177–185, 2000.

[8] R. Kapitza, H. Schmidt, and F. J. Hauck. Platform-Independent Object Migration in CORBA. In *OTM 2005*, LNCS 3760, pages 900–917, Oct 2005.

[9] Object Management Group (OMG). Common object request broker architecture: Core specification. OMG Doc. formal/02-12-02, 2002.

[10] Object Management Group (OMG). Life Cycle Service Specification. OMG Doc. formal/2002-09-01, 2002.

[11] R. Kapitza, H. Schmidt, G. Söldner, and F. J. Hauck. A framework for adaptive mobile objects in heterogeneous environments. In *OTM 2006*, volume 4276 of *LNCS*, pages 1739–1756, 2006.

[12] R. Kapitza and F. J. Hauck. DLS: a CORBA service for dynamic loading of code. In *OTM 2003*, Sicily, Italy, 2003.

[13] R. Kapitza, U. Bartlang, H. Schmidt, and F. J. Hauck. Dynamic integration of peer-to-peer services into a corba-compliant middleware. In *OTM 2006 Workshops*, volume 4277 of *LNCS*, pages 28–29, 2006.

[14] R. Kapitza, H. Schmidt, U. Bartlang, and F. J. Hauck. A generic infrastructure for decentralised dynamic loading of platform-specific code. In *7th Int. Conf. on Distrib. App. and Interop. Sys—DAIS'07*, 2007. Accepted for publication.

[15] L. Gong. JXTA: A network programming environment. *IEEE Internet Comp.*, 5(3):88–95, 2001.

[16] W3C. Web Services Architecture. http://www.w3.org/TR/ws-arch/, 2004.

[17] D. K. Barry. *Web Services and Service-Oriented Architectures*. Morgan Kaufmann, 2004.

[18] W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/wsdl20/, 2003.

[19] W3C. SOAP Version 1.2 Part 1: Messaging Framework. http://www.w3.org/TR/soap12-part1/, 2003.

[20] Organization for the Advancement of Structured Information Standards (OASIS). Introduction to UDDI: Important Features and Functional Concepts. Whitepaper, 2004.

[21] T. Andrews et al. Business Process Execution Language for Web Services - Version 1.1, 2003.

[22] Matthias Kloppmann et al. WS-BPEL Extension for People - BPEL4People, 2005.

[23] IBM. Websphere process server. http://www-306.ibm.com/software/integration/wps/, 2006.

[24] C. Erfurth and W. R. Rossak. Autonomous Itinerary Planning for Mobile Agents. In *3rd Symp. on Adapt. Ag. and Multi-Ag. Sys.—AAMAS*, 2003.

[25] C. P. Kunze, S. Zaplata, and W. Lamersdorf. Mobile Process Description and Execution. In *6th Int. Conf. on Distr. App. and Interop. Sys.—DAIS*, 2006.

[26] W. Binder, I. Constantinescu, B. Faltings, K. Haller, and C. Trker. A Multiagent System for the Reliable Execution of Automatically Composed Ad-hoc Processes. In *Auton. Ag. and Mult.-Ag. Sys.*, volume 12, pages 219–237, 2006.

[27] F. Ishikawa, N. Yoshioka, Y. Tahara, and S. Honiden. Mobile Agent System for Web Services Integration. In *Perv. Netw.*, 2004.

[28] I. Satoh. A document-centric component framework for document distributions. In *OTM 2006*, volume 4276 of *LNCS*, pages 1555–1575, 2006.