Automatic negotiated integration of services in pervasive environments

Noha Ibrahim, Frédéric Le Mouël, Stéphane Frénot

▶ To cite this version:

HAL Id: inria-00394952

https://inria.hal.science/inria-00394952

Submitted on 12 Jun 2009

Automatic negotiated integration of services in pervasive environments

Noha Ibrahim, Frédéric Le Mouël and Stéphane Frénot

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

Abstract

The development of many highly dynamic environments, like pervasive environments, modified the behavior of users and consequently, their expectations from systems and applications used in these environments. Thus, a user equipped with a laptop can connect to different places, and, each time, like to use his applications integrating the functionalities offered by the physically or logically close environment.

In this article, we offer the architecture of a system of automatic negotiated integration of services. This integration introduces the elements of decision of the integration (negotiation, automatisation) and an interface `Integrable` *making possible the application of different techniques of integration (parameterization, downloading/uploading - deployment, composition, weaving).*

Key words: distributed systems, pervasive environment, integration, components / services, negotiation, automatisation.

1. Introduction

The development of many highly dynamic environments, like pervasive environment [19], modifies the attitudes of applications and their way to react with them. Devices tend to be more and more tiny and even invisible. These devices are either mobile and/or embedded in almost any type of imaginable objects such appliances and various consumer goods [18]. In these environments, applications would like to, whenever it is possible/needed integrate the services offered by the local environment. In particular, if no single service can satisfy the functionality required by the application, there should be a possibility to combine existing services together in order to fulfill the request. Because of the pervasive nature of the environment, this integration should better be automatic leaving to the application the opportunity to focus on using services rather than analysing how or when to integrate them. This trend has triggered a considerable number of research efforts on the integration of services. Nowadays, services are not capable of integrating automatically in existing systems, without the need of complicated processes; hence, the objective is to create an environment with automatic, smart and customizable integration of services. This one introduces the elements of decision of the integration (automatisation, negotiation) and an interface `Integrable` making possible the application of different techniques of integration (parameterization, downloading-uploading/deployment, composition, weaving).

In section 2, we begin by examining closely-related fields of integrating services in a pervasive environment. In section 3, several scenarii illustrate how the integration of services facilitates everyday life. In section 4, we define the integration such as we conceive it, we list the properties it respects and we introduce the architecture of our system. Section 5 concludes and introduces future works.

2. Related Work

Three major domains of object oriented programming lean over the concept of integration: the Component-Based Software Engineering (CBSE) [7], the Aspect-Oriented Programming (AOP) [9] and the Service-Oriented Programming (SOP) [1]. Each of these domains has several definitions and techniques of integration according to the different existent platforms. Pervasive environments become more and more a target domain for this type of programming and the integration of services takes a new sense. Different types of models based on components as EJBs [13], CORBA Component Model [22], Fractal [2] and Web services [8] allow the interaction between distant components. The integration of components in these different models is often reduced to the deployment and/or the

parameterization of these components. However, these integrations are not perfectly adapted to pervasive environments and they do not take into account the change of context at the time of execution or the deployment of components. In these models, the definition of new components is rather difficult during execution, so the integration of components is often predefined beforehand. The development of pervasive environments throws a certain number of new challenges for component programming based, especially concerning taking into account mobility, context awareness and adaptability. Molène and AeDEn [12] projects offer an approach consisting of an adaptive distribution of applications allowing to use the resources of the environment dynamically to palliate the insufficiency of the resources of the mobile. AURA [6] project proposes a model of programming based on task. In this model, tasks are seen as being a composition of several components. AURA interprets the physical context of the user and can thus discover and compose components to fulfill a task.

Aspect-Oriented programming allows to establish transverse concerns (aspects) independent ones of the others and to combine them (the weaving) later to produce final application. AspectJ [10], Fac [14] and [5] are models based on aspect, applying the weaving of aspect as method of integration. Recent works were fulfilled on adaptation seen as an aspect in pervasive environments [16]. By using the aspects of AspectJ, the system modularizes three essential faces of adaptation in pervasive environments: management of the devices present in context, management of their contents, as well as the adaptation of devices to the change of context.

In the terminology of Service-Oriented programming, the integration of service is often reduced to a composition of service. Nowadays, researches aim at developing an architecture which allows the composition of service by using a logical reasoning given by the languages of description of service as DAML [20], Universal Description, Discovery and Integration (UDDI) and Web Service Description Language (WSDL) [21]. These languages define standard ways for service discovery, description and invocation (message passing). SWORD [15] is a developer toolkit for building composite web service. It does not deploy the emerging service description standards such as WSDL and DAML-S, instead, it uses rule-based plan generation, it specifies the web services by using Entity-Relation model. Many current service composition platforms have been designed with the inherent assumption that the services are resident in the fixed network infrastructure and running on a relatively stable platform. Few have tried to consider alternate design approaches of service composition systems for pervasive environments. A distributed broker-based service composition protocol for pervasive environments [4] proposes a model adapted for pervasive computing, but it focuses only

on the composition aspect of integration. For each composite request, the protocol elects a Broker from within a set of nodes. The request source delegates the responsibility of composition (i.e. discovery, integration and execution) to the elected broker. The main protocol, based on the composition and the integration, is seen to be a part of the protocol of composition. Scooby [17] a middleware for service composition in pervasive computing, proposes a system which provides a solution based on the use of binding variables utilising late and lazy dynamic binding, along with the supporting service composition language in which users can formally specify their policies based on event notification messaging system.

3. Scenarii

To specify our contribution compared to related works, we offer different scenarii corresponding to situations of common life. These scenarii put the emphasis on the importance of physical and/or logical context as well as on variations affecting this context, the processes of negotiation between services and various methods of integration.

3.1. Scenario 1: acquisition and personalization of services

Max, an architect, wants to accomplish a video of one of his models. Equipped with his PDA, he enters a studio of production of video clip. The studio has a camera and movie maker software. The PDA integrates the software of the video camera automatically as soon as Max enters the studio. Max begins taking the shots he needs of his model. He can command the camera by using a familiar interface installed on his PDA. He doesn't need to know how the camera works! The PDA integrates the driver of the camera and weaves it with the interface software that Max always uses. Then, Max uses the movie maker software that the PDA integrated to create his video clip. The PDA composes the movie maker software with the driver of the camera proposing then a new service that enables Max to take shots and create the sequence of the video clip. This new service, initially not offered by environment is now available on Max's PDA. It has a life of one week during which Max can use it in every visit to the studio. Once it outdates its length, this service will be expire and be uninstalled automatically from the PDA.

3.2. Scenario 2: broadcasting of individual services

Julia logs on to a cyber game, virtual space, through her laptop from her work. She logs on very often during her breaks to play her favorite car racing game with the users

present in the virtual space. The uniqueness of this game is that every user can provide a personalized part of his virtual space (his virtual home, his virtual car) to other players. Julia offers a special service of a set of virtual tracks (mountains, snow, beach). Julia can also use services offered by other players such as sets of virtual race cars compatible with each track. the virtual environment is therefore totally formed of the composition of the individual environments of the players. Julia must attend a meeting and stops playing. She disconnects from the virtual space and leaves to the virtual space the task to arrange and take in charge her unexpected departure. The private services which were downloaded from Julia's laptop are taken away of the space and the environment must find alternatives to continue offering the players the public services of Julia.

3.3. Scenario 3: automatic and negotiated discovery of services

Pierre, a PhD student, enters the university library, equipped with his numerical badge, Dbag. Automatically a process of identification is established between Dbag and the environment of the library. Dbag authenticates itself into the environment. Pierre needs to attend an audio-visual course displayed in the library. The Dbag negotiates with the environment the availabilities. The environment proposes an adequate media service after analysing the high bandwidth (384Kbits/s) present at the moment. The Dbag integrates the service and notifies Pierre that he can begin to watch the course. Suddenly, the bandwidth drops and the integrated service can no longer support the media course. Again a negotiation is established between the Dbag and the environment which searches for other alternatives to continue the service rather than stopping it. The environment proposes to the Dbag another service that enables Pierre to watch the media in lower quality preserving the sound quality of the media and agrees with the Dbag to notify it when the previous service is available so it can switch back to it.

4. Integration

To answer these scenarii's needs, we give in this section the definition of integration as well as its properties as it applies to our research. We will explain, later, the architecture of our system. Service, the core of our system is defined as the instantiation of a specific model of component. Then we discuss how the system architecture applies to the scenarii defined in section 3. Finally, the Middleware layer we offer, is represented with its essential services: service of decision, service of negotiation and service of integration.

4.1. Definition

Integration is the process of incorporating a service or a set of services so that they can work together and provide a new service [3]. Integration must be independent from any software technics. We consider that services are gathered in the middleware layer. The middleware layer is situated between the application layer and the system layer. It resides on every host, terminal and devices 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 management of the hardware, the network and connectivity.

4.2. Properties

We will be particularly interested in four properties for the integration [11]. The three first ones apply in a general way to integration, and the last comes from needs expressed in pervasive environments:

Genericity: Integration must be used by all sorts of applications requiring the integration of services within its own services, or desiring to upload its services in any environment.

Reversibility: pervasive environment is an always changeable, extremely dynamic environment. Integration must be reversible to allow deleting services previously integrated if these are not accessible any more.

Lifetime cycle: The integration must be set for a certain time. It must have a life cycle to represent its existence and its evolution in time. A service resulting from integration, can therefore last a certain time span and disappear once attained its cycle.

Context-awareness: integration must fit to the variations of context. These variations can be hardware as well as software.

4.3. System architecture

First of all we begin by modeling context and service, and then we represent the essential services present in our Middleware layer as well as their interactions. Finally, the different techniques of integration are defined.

4.3.1. Requirements : Context Model. Context plays a very important role in integration in pervasive environments. We define context by three characteristics: its contents, its attachment and its range. The content of a context is all the services present at a certain moment in the context. Context must be tied to an entity and must have a certain range. The entity, to which a context can be tied, can be a physical entity as a room or a logical entity as a service. The range of context can also be physical as a geographical distance, or logical as though to apply to a class of ser-

vices.

Service model we consider a service to be an instantiation of a component. As shown in the figure 1, a component includes a dependency manifest described by an Architecture Description Language (ADL), classes and static context. It also publishes interfaces, requested and provided ones and special interface of integration. `Integrable` interface provides the possibility of inserting a service easily in group of services according to the most adapted context. During the instantiation, Manifest is instantiated in bindings between services in Middleware. Classes are instantiated in objects and the present static context in the component becomes dynamic at the time of instantiation.
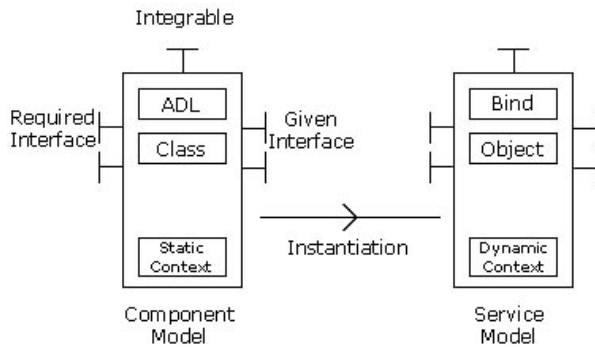

Figure 1. Component and service model

4.3.2. Middleware Layer. The middleware (Figure 2) contains essential services of integration of our system within the `IntegServ service`. A detailed presentation of these services is given below. The System Layer communicates with the `discovery service` in the Middleware. The system informs this service of any change in the context and the latter brings them back to the `context manager service`. The `service binder` allows services uploaded and downloaded in environment to register, and also allows taking away the services which are not available any more in environment.

4.3.3. Interactions between services. The `IntegServ service` of the Middleware is itself the integration of several essential services which interact together. The integration process is as follows. First of all, the `decision service` interacts with the context manager to know the state of the context. The `context manager service` informs the `decision service` of the state of context and the `decision service` can work out a strat-
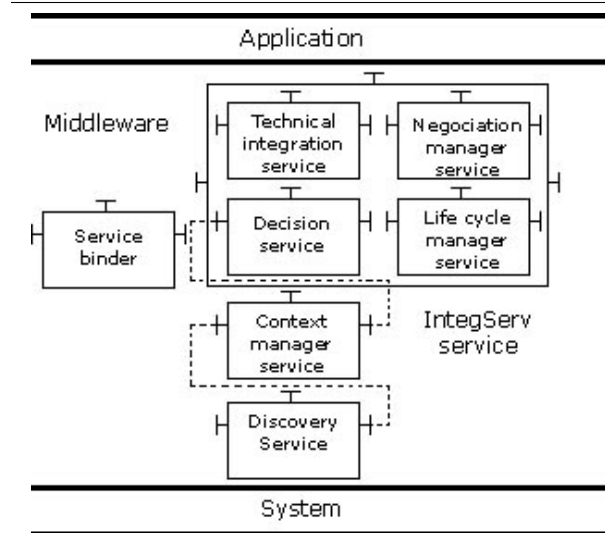

Figure 2. Middleware architecture

egy of integration with the present services. If more than one strategy is found, the `decision service` evaluates all strategies and proposes the best one for execution. Once the strategy defined, the `decision service` interacts with the `technical integration service` which carries out the orders of the latter. If a change in context occurs, the `context manager service` alerts the `decision service` which interacts with the `life cycle manager` and the `negotiation manager service` to adapt integration to the changes of context.

4.3.4. Manager services of integration.
Decision service : The capacity of decision of our architecture is provided by the `decision service`. This service can take decisions and adapt automatically to the variations of context. The `decision service` uses strategies to decide the proper integration to apply. These strategies should be made based on the up to date information due to the pervasive nature of the environment. The `decision service` proposes also other types of integration. Semi-automatic integration is jointly decided by the context and the `decision service`. The upon request integration is performed at the request of one of the services to the `decision service`. The to be confirmed integration can not be realized unless the `decision service` confirms integration to the services. The `decision service` decides also which technology of integration is to be used and which are the services to be integrated. It chooses these services by analyzing their capacities to form a new service. The `decision service` is reactive, it can react to the variations of context and adapt to them. It is also proactive, it can initiate a decision of integration independently of any variations in context.

Negotiation service : Negotiation is provided by the `negotiation service`. This service implements a `Negotiable` interface provided to other services so that they can come to an agreement on terms and conditions of their integration. It occurs when certain services requested for integration are not available, or when context changes and requires a re-analysis of integration. It offers an alternative to the primary integration decided before. Negotiation is made according to contracts.

Life cycle manager : `Life cycle manager` is in charge of the life time of the integration. This service interacts directly with the `decision service`. The integration initiated by the `decision service` can have, from its creation, a life time, known and managed by the `life cycle manager`. Once this time expires, the `life cycle manager` informs the decision service which disintegrates this integration, and tries to react if necessary with the context. The change of context can also have impact on the life cycle of integration. In fact, if, for instance one of the services of integration leaves the context, the `context manager service` informs the `decision service` about it. The `decision service` interacts with the `life cycle manager` to find a solution.

4.3.5. Methods of integration. The `technical integration service` is the service which allows applying different methods of integration. It is part of the basic services and carries out the orders of the `decision service`. These different methods can be applied one before the other and/or combined. They all have a set of services as target.

Parametrization The parameterization of a service concerns its context. It allows services to integrate with a new context, or to fit to their context if this last one changes. As shown in figure 3, only the context part is changed during this technique of integration.
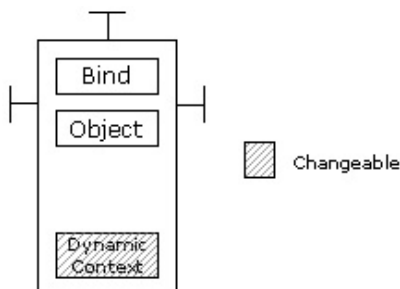

Figure 3. Parametrization of services

Downloading/Uploading - Deployment Downloading/Uploading - deployment integrates a service by downloading it or uploading it in a context. As shown figure 4 only bind part is changed in this integration.
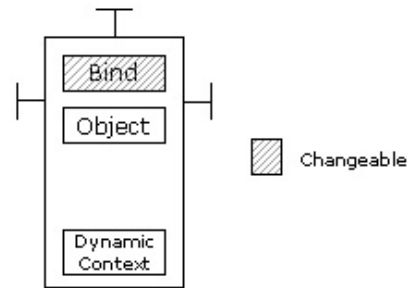

Figure 4. Downloading/Uploading - Deployment

Composition The composition of a set of services is a technology of integrating the services by connecting their interfaces. The result of the composition of several services is a service ready to be used by other services. As shown in figure 5, the composed services can belong to different platforms; the important thing is that they all belong to the same context.
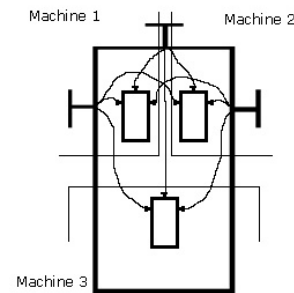

Figure 5. Composition of services

Weaving The weaving of a set of services allows spreading code in a transverse way in objects. As shown figure 6, a service is weaved on a set of two services but only the objects of services are concerned.

4.3.6. Applying scenarii to the system. In this section we will give a brief description of how to apply the scenarii we defined in section 3 on our architecture defined above. We will focus on the context, aspects of negotiation and the strategies to choose the appropriate method of integration.

Scenario 1 In this scenario, the context is modeled as follows: the contents of the context are the services present in the studio of production, this means the camera driver, the movie maker software and the user display interface on the PDA. This context is physically attached to the stu-
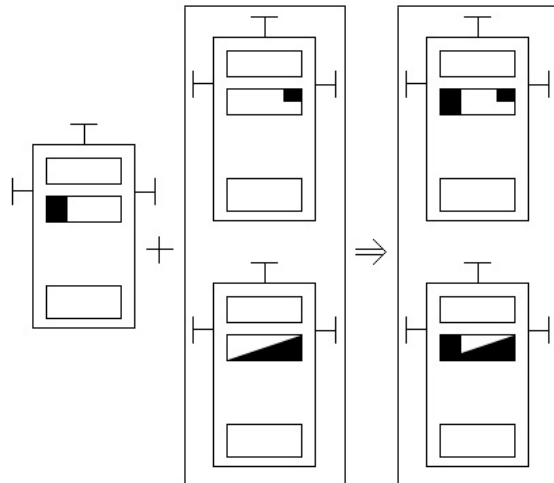
Figure 6. Weaving of services

dio of production and the range of the context is the border of the studio. Two methods of integration are used, the weaving and the composition of services. The `decision service` upon the request of the PDA analyses the two services to integrate. On one hand there is the camera, on the other hand the interface command on the PDA, the `decision service` decides to weave the interface command with the camera so Max can use it. For the other integration, the one which integrates the movie maker with the camera the `decision service` decides to compose them after analysing the output of the camera service and the input of the movie maker and finding that they match. This composition has, for instance, a life time of one week and is controlled by the `life cycle manager`. In this scenario the aspect of negotiation is not very important due to the fact that all needed services are available.

Scenario 2 Here the context is attached to a virtual entity and is composed of the services offered by users present in this virtual space. To take part of the context a service must log in the virtual space and to quit the context it must log out. When Julia decides to log out, her service, a set of virtual tracks, is no longer available. The `context manager service` informs the `decision service` of this departure and the `decision service` must apply its strategy to integrate other services to cover up the departure of Julia's service. The `decision service` must at first interact with the `life cycle manager` of the no longer available integration. Then, the `decision service` decides to integrate services from other players, Bob and Joe, to replace Julia's service. Bob offers a virtual platform providing the preliminary plans of race tracks around the world and Joe offers the environments and sceneries for these tracks. The combination of these 2 services will make up for Julia's service absence. The

`decision service` chooses to integrate the services offered by Bob and Jo, because by combining them the new resulting service replaces Julia's service.

Scenario 3 The context is formed by the set of services present in the library. The context is attached to a physical entity as in scenario 1. This scenario shows exactly how the process of negotiation works. If a service is no longer available due to variations in context, the `decision service` must cope with this problem by negotiating with the other available services. In the scenario, the `decision service` of the environment negotiates with the Dbag to find the appropriate service to integrate according to the Dbag's preferences. Once they reach an agreement, a contract may be established and stored in the `negotiation service`. The contract provides the Dbag a service when the band-width is low and another one when it is high.

5. Implementation

We begin the implementation of a prototype, by the definition of the interface of integration `Integrable`. This interface constitutes the first block of the development of the integration system offered in our article.

Interface `Integrable` provides three methods allowing to manage the integration in a service, of a service or group of services: method `Integrate`, method `Unintegrate` and method `GetIntegratedServices`.

```
public interface Integrable {

void integrate(Collection serviceSet)
throws IntegrationException;

void unintegrate(Collection serviceSet)
throws UnIntegrationException;

Collection getIntegratedServices();
}
```

Method `Integrate` allows integrating a set of services into the common service. The technical integration service implements this method by parameterization, downloading-uploading, composition or weaving. In case the integration is not possible an exception `IntegrationException` is generated. This case of error can appear if for instance we undertake a weaving between objects unweavable.

Method `Unintegrate` allows to cancel integration of a group of services beforehand integrated. It guarantees the reversibility of integration. In case the service to desintegrate is being used in the context or is not available, an exception `UnIntegrationException` is raised.

Method `GetIntegratedServices` returns all the services having already been integrated into the common service.

After the definition of this interface, we are interested in defining our different services introduced in figure 2 in form of bundles OSGi, and in defining other necessary interfaces for the total process of integration as: `Negotiable` interface and `LifeCycle` interface.

6. Conclusion and perspectives

In this article, we introduce a middleware architecture of automatic integration of services. We focused on three main points. First, we defined a context and a model of service. Then, we conceived an integration service, providing an interface `Integrable`, and itself resulting of the integration of different services (technical integration service, negotiation service, decision service, context manager and life cycle manager). Finally, we defined the different techniques of integration (parameterization, downloading/uploading - deployment, composition and weaving). In the future, we aim, at finishing the development of our system under OSGi and publishing our services as UPnP devices. We are also working on adding a semantic description of our services so as to enrich the services of negotiation and decision.

References

[1] G. Bieber and J. Carpenter. *Introduction to Service-Oriented Programming*. OpenWings, Apr. 2001. White paper.

[2] E. Bruneton. *Developing with Fractal*. The ObjectWeb Consortium, France Telecom (R&D), Mar. 2004. version 1.0.3.

[3] D. Chakraborty. *Service Discovery and composition in Pervasive Environments*. PhD thesis, University of Maryland, 2004.

[4] D. Chakraborty, Y. Yesha, and A. Joshi. A Distributed Service Composition Protocol for pervasive Environment. In *University of Maryland, Baltimore County (UMBC) eBiquity*, Mar. 2004.

[5] R. Douence, T. Fritz, N. Loriant, J.-M. Menaud, M. Ségura-Devillechaise, and M. Südholt. An expressive aspect language for system applications with Arachne. In *4th International Conference on Aspect-Oriented Software Development (AOSD'05)*, Mar. 2005.

[6] D. Garlan and B. Schmerl. Component-Based Software Engineering in Pervasive Computing Environments. In *4th International Conference on Software Engineering (ICSE) Workshop*, May 2001.

[7] G. T. Heineman and W. T. Councill. *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, June 2001.

[8] W. Iverson. *Real Web services*. O'Reilly, Oct. 2004.

[9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *European Conference on Object-Oriented Programming (ECOOP)*, pages 220–242, June 1997.

[10] R. Laddad. *AspectJ in Action: practical Aspect-Oriented Programing*. Manning publications, July 2003.

[11] F. Le Mouël. *Environnement adaptatif d'éxécution distribuée d'applications dans un contexte mobile (Adaptive environment for distribution of applications in a mobile context)*. PhD thesis, Université de Rennes 1, Rennes, France, Dec. 2003.

[12] F. Le Mouël, F. André, and M.-T. Segarra. AeDEn: An Adaptive Framework for Dynamic Distribution over Mobile Environments. *Annales des Télécommunications*, 57(11-12):1124–1148, Nov.-Dec. 2002.

[13] R. Monson-Haefel. *Entreprise JavaBeans*. O'Reilly & Associates, Mar. 2000.

[14] N. Pessemier, L. Seinturier, and L. Duchien. Components, ADL and AOP: Towards a common approach. In *Workshop ECOOP Reflection, AOP and Meta-Data for software Evolution (RAM-SE04)*, June 2004.

[15] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *11th World Wide Web Conference*, 2002. Honolulu, USA.

[16] A. Rashid and G. Kortuem. Adaptation as an Aspect in Pervasive Computing. at Object-Oriented Programming, Systems, Languages & Applications (OOPSLA) '04 Workshop, Oct. 2004.

[17] J. Robinson, I. Wakeman, and T. Owen. Scooby: Middleware for Service Composition in Pervasive Computing. In *2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing*, Toronto, Canada, Oct. 2004.

[18] D. Saha and A. Mukherjee. Pervasive Computing: A paradigm for the 21st century. *IEEE Computer Society*, Mar. 2003.

[19] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communication*, Aug. 2001.

[20] M. Sheshagiri, M. des Jardins, and T. Finin. A planner for composing services described in DAML-S. In *International Conference on Automated Planning and Scheduling (ICAPS) 2003 Workshop on planning for web services*, July 2003.

[21] A. E. Walsh. *UDDI, SOAP and WSDL: the web services specification Reference book*. Pearson Education, Apr. 2002.

[22] R. Zahavi. *Entreprise Application Integration with Corba Component and Web-Based solutions*. Johne Wiley & sons, Nov. 1999.