Communication Networks and Distributed Systems Modeling and Simulation Conference, CNDS'04, San Diego, California, USA, 18 - 21 January 2004.
Sponsored by: The Society for Modeling and Simulation International (SCS)

An Holistic Approach for Pervasive Computing Environments

Sergio Maffioletti, Soraya Kouadris M. and Béat Hirsbrunner

DIUF – Department of Informatics
University of Fribourg, Switzerland
{Sergio.Maffioletti, Soraya.Kouadrimostefaoui, Beat.Hirsbrunner}@unifr.ch
http://www.unifr.ch/diuf/pai/
Swiss National Science Foundation Project N. 2000-065301

Abstract

Pervasive computing is about building applications to bring computation into the real, physical world. The high degree of dynamism and heterogeneity of the resources involved in such applications makes service adaptation and interoperability a difficult task. This paper presents UBIDEV, a service framework that faces the heterogeneity problem by hiding at the application level the dynamism of the underlying environment. We describe the UBIDEV architecture focusing on the description and the management of services and resources. We also describe how this approach decreases the complexity of the design and development of service-oriented applications. A prototype implementation of a unified messaging system is presented as a validation of the architectural design.

Keywords: Ubiquitous Computing, Resource Management, Service Management, Ontology, Context, Coordination.

1. Introduction

Pervasive environments are inherently composed of heterogeneous and dynamic resources. Applications are characterised by a large number of services that have to configure the environment and the interaction with other services in order to carry out their execution. Moreover services perceive their environment, usually with the help of sensors, interpret this information and hence derive their model of context, based on their perception. Application uses this context information to adapt its behaviour to accommodate user needs and activities. Hence coordinating services in a continuously changing search space becomes a necessary but challenging task.

Consider a scenario in which members of the same department want to exchange messages with each other (point to point), with group of colleagues (multicast) or with everybody (broadcast). Without the presence of a unified communication medium (i.e. telephone) it is a non trivial task to set up communication channels because there is virtually no interoperability between messaging media. As a result, individuals must check messages in several different media using the client specific to each of them. The difficulty of dealing with several message pipelines is complicated by an absence of information allowing a sender to determine the best way to reach his or her intended recipient. In an ideal world, a wire-line-based device like a computer or a desk phone should be able to detect and route incoming messages based on the proximity of the person who receives messages on that device. Just walking away from a wire-line device should cause important messages to be routed to the recipient's portable phone, pager or PDA.

In a scenario characterized by a high degree of heterogeneity of the underlying resources, it is not possible for applications to rely on common base functionalities offered by a Network OS. What is required is a software infrastructure that defines a suitable coordination model to describe and manage the dynamism of these services and, at the same time, to relieve the application from directly handling the heterogeneity of the underlying environment. Incorporating context into the infrastructure facilitates the adaptation of the computing environment to the needs of users.

This paper presents how UBIDEV [13] faces heterogeneity from both the physical and the service dimension with a focus on how resources influence the service management. UBIDEV hides resources heterogeneity and can adapt dynamically to changes in the environment while maintaining the integrity of the overall system. It proposes a context-centric resource and service management: it is the application context that determines the semantics of resources and services involved; consequently resource access, service instantiation, service composition and client query solving are based on this semantics. That allows applications to automatically reconfigure themselves according to context changes.

The rest of this paper is organised as follows: Section 2 briefly introduces the architectural model of UBIDEV. Section 3 presents the functionalities of UBIDEV. Section 4 illustrates an application example. Finally conclu-

sions and future perspectives are given in Section 5.

2. System overview

UBIDEV is a lightweight system for pervasive computing environments conceived with the aim of supporting an application to configure and adapt itself to the underlying environment. In doing so it separates the coordination aspects from the resource and service management aspects in order to hide at the application level the heterogeneity of the underlying environment. As a result an application can be described in terms of composition[1] of homogeneous services as in classical coordination-based distributed systems [8].

A key element in realising this architecture is the use of an application ontology that undergrid the communication and representation. Knowledge representation systems use various terms with different domain specific definitions in order to describe the knowledge model. Instead of introducing its own semantics, UBIDEV relies on the ontology defined by the application to determine the internal representation (concepts) of semantics and the relation to the environment (context and resources). That led to small topic-oriented ontology used to classify the whole environment as well as the contextual information. Figure 1 represents the actual implementation of UBIDEV based on such a model.

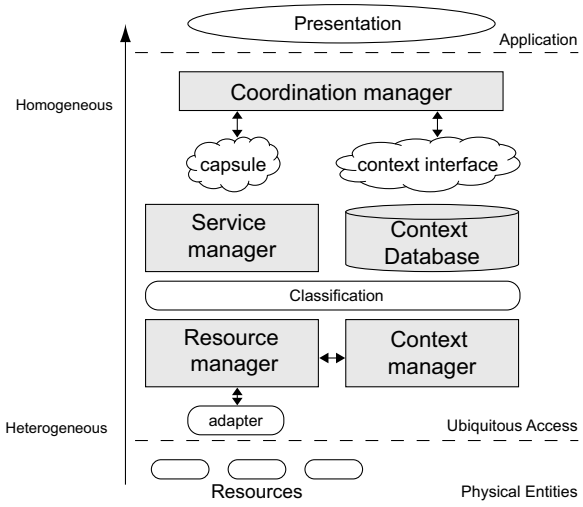

Figure 1: UBIDEV **Architecture:** UBIDEV *is built around a classical layered reference model: Physical Entities, Service and Coordination (the Ubiquitous Access Layer) and Application.*

The *Physical Entities layer* represents the resources belonging to the environment. Resources describe all the entities that may be involved in the execution process of an application; they describe physical devices, software components and users[2]. Due to the dynamic changes in the availability of physical devices, network bandwidth, connectivity and user location, the system has to classify those resources monitoring constantly their changes. In such a way the upper layers have always a consistent description of the physical dimension.

The *Ubiquitous Access layer* represents the core part of UBIDEV; it centralises all functional aspects related to resource and service management; it is also responsible of creating and maintaining a model of the context of the application. It is composed of four main modules as depicted in figure1: The Resource Manager handles the communication with the underlying resources through standardized interfaces called *adapters*. It also supplies the corresponding classification that identifies the resource capabilities; in doing so, it relies on the *Classifier* [18] to create an abstraction of the resources in terms of high level concepts taken from an ontology. Service Manager is responsible for instantiating and monitoring services. It analyses the resource classification in order to bind resources and services according to the service requirements. The resulting *capsule* (figure 1) hides at upper level the heterogeneity nature of the embodied resources and service. Context awareness is achieved by the Context Manager that is responsible for gathering, processing and representing contextual information. It also classifies this information into context types. Coordination Manager is responsible for decomposing complex queries coming from the *Application Layer* in terms of composition of basic services. In doing so it relies on the contextual information provided by the *context interface* in order to adapt the service composition policy, hence the application behaviour, to the specific context configurations. Next section will give more details about functional aspects related to each module.

The *Application layer* is a generalisation of the view in a traditional Model Viewer Controller [10]; it works as a presentation module embodying some application dependent functionalities like visualisation that produces the input queries. Input queries are represented as a transformation from an input resource to a final output resource; this transformation is interpreted by the *coodination manager* in terms of composition of abstract services. Application GUI modules, together with the adapter are the only piece of code that is executed on the specific resource according with the application requirements. UBIDEV does not define any particular client-side module; in the UMS example, the Palm interface is realized by a simplified implementation of a remote framebuffer protocol.

[1] Composition is the process of building complex composite services from primitive ones; thus enabling rapid and flexible creation of new services.

[2] In our model the user is considered as an ultimate resource involved in the interaction process. As such it could correspond to the lowest level, which is in direct contact with the devices.

3. UbiDev: the Ubiquitous Access Layer

This section presents the functional aspects of the four modules composing the Ubiquitous Access layer. The focus is on how resource and service management are faced, and what is the resulting abstraction at the coordination level.

3.1. Resource Manager

Resources have a tight coupling between hardware and software, on one hand because of the limitations of the devices themselves and on the other hand due to dependencies of software on specific hardware features like pen-input or temperature measurements. Therefore configuring and describing these resources is an important issue.

Existing infrastructures tend to address heterogeneity problem basically from the physical dimension viewpoint, like in Rendezvous [22] where the focus is on the autoconfiguration aspect, or like in Jini where the underlying hardware is abstracted by the JVM. Other systems face heterogeneity from the service dimension viewpoint like in Corba where the presence of a service broker facilitates the service lookup task. In both cases they have to address a common ground problem: how to define an abstraction of the underlying resources composing the application environment. This problem can be divided in two basic stages:

- Configuration: how to configure a resource when it joins an application environment. A typical configuration problem is to decide which policy to use for assigning it an IP address.
- Description: how to describe the functional characteristics of a resource inside a system, in order to allow services to query them.

3.1.1. Resource Configuration

UBIDEV focuses resource configuration issue on the federation management aspect. The actual implementation relies on classical and semi-static configuration technologies such as DHCP and DNS. Even if such technologies fall short in solving the autoconfiguration problem in a highly dynamic and heterogeneous environment, they are quite easy to set-up and their services are flexible enough for prototyping. Further integration of more structured configuration systems at the bottom of UBIDEV, will allow more flexible and dynamic way to address resource configuration. Upon these technologies UBIDEV defines a federation manager that controls resources belonging to a federation each given time. Its role is to associate to every resource an *adapter*. An *adapter* is considered as a virtual representer of a resource inside a UBIDEV environment; it defines uniform access mechanisms for exchanging data in a seamless way. *Adapters* are the first step towards the definition of an abstraction of the underlying physical dimension. The *adapter* approach is particularly suitable for integrating handheld devices because it embodies the specific resource access information.

3.1.2. Resource Description

A common approach for resource access and selection is standardising interfaces. Several architectures rely on directory services combined with attribute based query services for resource selection and access protocols (like in Corba or Jini). Other approaches like INS [2] use simple description languages based on attributes and values for names. Applications use this language to describe what they are looking for (their intent). INS uses a late binding mechanism to maintain a mapping between service description and its network location. In both cases resources are responsible of announcing their attributes to the rest of the community.

UBIDEV shifts the focus of resource description from a resource centric to a context centric. In UBIDEV resources are classified relying on a set of abstract concepts collected in ontology. The ontology reflects the shared conceptual model of the resource, which includes what a resource is capable of doing (e.g. the functional interface of the service), the terms describing the resource (e.g., the data types for describing the service) and the meanings of the terms (e.g., what they stand for).

In UBIDEV the meaning of these concepts is implicitly given by *Classification* (figure 1). Classifiers [18] are services that given a resource and an ontology, output concepts of that ontology. This basically means that a classifier associates one or more concepts it knows with a resource in a context and tags the resource as an instance of that concept. Classifications of resources are stored and used as a cache when a service requests an instance of a concept. The process of requesting an instance of a concept is called "addressing by concept", because the instance is referred by a concept instead of specific resource identification, such as memory address, name or URL. Thanks to the COCA *classifier*, UBIDEV decouples the high-level concepts (abstractions) from the instances implemented by a context. The concept "nearest printer" [9] for instance may be used no matter how a context supplies the corresponding implementation. In such a way a user moving around different environments will not have to reconfigure her printing application. This means that an application may express its resource requirements in terms of concepts instead of addressing specific resources directly (i.e. by an URL).

Similar approach has been considered for semantic service discovery like in [21] or in [3] where a XSB-based engine is used to interpret DAML queries.

The main advantage of this approach in facing resource management problem is that resources selection

is based on their semantics

3.2. Service Manager

A service oriented paradigm [5] describes an application in terms of what functionalities (services) it can exploit instead of which objects are involved.

In Jini for example, services are based on well known interfaces written in the Java language; whether a service is implemented in hardware or software is not a concern. The client only knows that it is dealing with an implementation of a specific known interface.

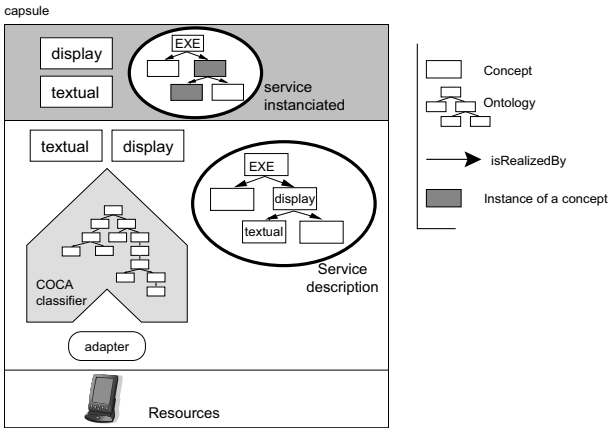

Figure 2: **The Service layer**. *It manages the service's instance operation. Classifier tags the PDA resource with concepts "textual" and "display" taken from the ontology it knows. Service Manager analyses the service's requirements (EXE) in order to look-up the suitable environment in which the service may be hosted. The capsule is created embodying the service module and the related resource access information.*

UBIDEV links hardware and software in terms of the dependence that resources have on the service (figure 2).

In UBIDEV a service is defined as an atomic action that transforms an input resource yielding a new resource as output. Additionally, a service is constraint by (1) the concept it accepts as argument (input concept) and (2) the concept produced as a result of the action (output concept).

To execute such actions, resources that match the service's semantics requirement have to be found (like printer for a print service). This requirement represents the service EXecution Environment (EXE) and is expressed by the service itself in terms of concepts the system should provide. The service EXE contains all resource access information the service requires to complete its execution. This EXE is created by the *service manager* during the instantiation of the service. It contains all resource access information the service requires to complete its execution (figure 2). A run-time instance of a service inside its environment is represented by a homogeneous entity called *capsule*. A *capsule* is homogeneous in the sense that it hides to the *coodination manager* all heterogeneous aspects related with the execution of the service it embodies. *Capsule* exports to the upper layer only the service interface in terms of input and output concepts. In this way a *capsule* represents a new organisational unit to encapsulate a service's computing environment within the system architecture, just as a process is an organisational unit for the components of a running application.

In systems like [16] the infrastructure is aware of the service requirements and discovers resources that match these specifications. However these requirements are expressed in terms of resource capabilities and basically are defined for compatibility reasons (Memory = 20Mb, OS = Solaris).

One way to obtain a service for the UBIDEV architecture is to wrap an existing application. For instance Emacs, Microsoft Word, PalmOS memo-pad and the QTopia text-editor can each be wrapped to become suppliers of a text editing service. Such wrappers map abstract service descriptions into application-specific settings. A similar approach has been taken into account in [19] where the infrastructure defines a name space for services; several service suppliers provide instances of the abstract services that tasks are composed of. Differently from UBIDEV, the classification of the services is done manually and is embedded into the service description.

The main advantage of facing the heterogeneity problem with the capsule abstraction is that UBIDEV can present at application level a homogeneous coordination space seen as an unified mechanism for dynamic communication, coordination, and sharing of objects [8].

3.3. Context Manager

The operating of the context-awareness approach in UBIDEV enables application to obtain and use different kinds of context. The whole process consists of sensors and dedicated services that sense various contexts, of a classification and a reasoning module that infer context knowledge and of interfaces that allow application to query and use of this knowledge. The use of ontology to model context is useful for checking the validity of contextual information because of the unified class typing. It also makes it easier to specify the behaviour of context-aware applications since the description relies on these context types.

There are different types of context that can be used by the application: physical, environmental, personal, social, application, etc. The structure of the context types [15] is defined by the application ontology. Each type of context corresponds to a class in the ontology.

This is similar to what Chen et al. denote in [6] by sensing the contextual information, the acquisition and sharing of contextual information, and reasoning about

contextual knowledge. For each of these steps we have associated a computational module, respectively, the *context manager*, the *context database* and the *context interface* (figure 1).

The *context manager* is responsible for gathering and processing context information and for providing this information to the *context database*. This contextual information is acquired through dedicated services that are instantiated according to the resource availability. The *context manager* register to these services that notify, through an event mechanism, the acquisition of context data.

In order to be used by a context-aware application, this context information needs to be structured according to a specific conceptual model that should reflect the application model of the context. The Classification phase takes the raw data of the contextual information and structure them according to the application ontology. That ensures that the resulting context knowledge reflects the conceptual model of the application. The pool of classifiers tags context data with one or more concepts of the application ontology. The result is passed to the *context database* to be stored as context knowledge.

The *context database* is acting as a repository; it receives the specifications of the knowledge from the *Classification* and stores them in a collection of Prolog predicates. This knowledge will be queried by the *coodination manager* to instantiate the contextual rules that drives the service composition policy. *Context database* acts as a knowledge base in the sense that it can analyses queries expressed in terms of Prolog questions. It solves these queries by making complex automated reasoning on the predicates it knows. The actual implementation of the system makes use of a Logic Programming system engine called XSB [17] to implement the reasoning system.

The *context interface* allows *coodination manager* to query the *context database* in order to instantiate the contextual rules that drive the service composition policy.

Using this approach, the *coordination manager* can retrieve the specific contextual data in a way that is decoupled from the service used for acquiring the data [15].

3.4. Coordination Management

The *coodination manager* acts as a control unit of the whole system. It centralizes the control of the underlying environment by presenting at application level an homogeneous coordination space. It proposes a model of the context structured as an organisational domain composed by autonomous entities. An entity is defined by its structure obtained by a recursive composition of entities. The application model of the context can be described using this hierarchical organization: an entity hierarchy represents the structure of the context. Each entity can be mapped to concrete context space like a building or a room in the case of the Ubiquitous Message System prototype in section 4, or can represent a logical context like the user space. The building blocks of this structure are atomic entities that are in direct correspondence with the underlying resources. Entities can interact with each other through communication endpoints that define a set of actions. In this model a service is described by an active connection between the input and the output entity. The main difference from the classical coordination models is that it defines context-dependent behavioural rules [20]. These rules are applied by the *coodination manager* to the communication endpoints: a set of rules may determine whether two entities have to be interconnected with each other or not. Rules describe the actions that should be taken in different contexts. A rule consists of a condition that, when satisfied, leads to a set of actions. Actions are interpreted by *coodination manager* to drive the service composition strategy and hence the application behaviour.

A concrete example of how actions are composed and controlled is given in the following section.

4. Ubiquitous Message System

Ubiquitous Message System (UMS) is a simplified implementation of commercial unified messaging systems like Lotus UMS for Domino [11] and General Magic Portico [14]. It has been developed to show the added value of an infrastructure like UBIDEV in terms of application design.

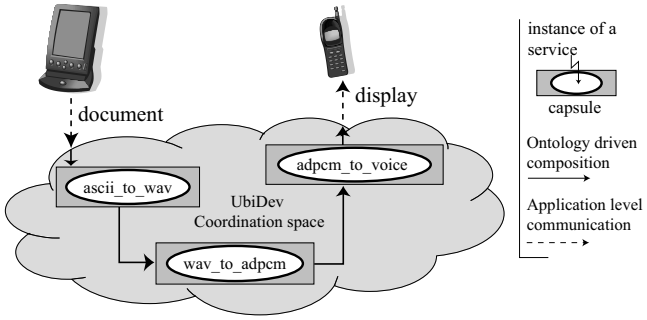

Figure 3: **Ubiquitous Message System.** *The application can compose capsules in a more structured entity (message_to_display). Social rules applied to such entity controls the interactions of the embodied services.*

UMS allows users to exchange messages through a virtual service called *message_to_display* without concerning them how this service is realised in every user context (figure 3). The scenario is characterised by an application context that represents the whole environment, and different user contexts for each federated user. Portable phone, mailer, PDA or the terminal where the user just logged-in represents an example of a user context.

Resource management in UMS is based on an event driven mechanism that allows the federation manager to

inform its subscribers about all the changes occurring in the federation. The dynamic control of the federation is based on context services instantiated according to the availability of specific resources (like when a IrDA port is plugged or when the Ethernet interface is brought up). Federation manager registers to these services in order to acquire information about the resources (a PDA is authenticated through the IrDA port or by querying the DHCP server on the Ethernet interface).

Service manager reacts to the federation event changes adapting the behaviour of the application in terms of service instance execution. In doing so the *coodination manager* controls the execution of the capsules embodying the service and the resource reference. Once a *service manager* is notified of a resource that left the environment, it stops the execution of the corresponding capsules and tries to re-initialise the execution of the service inside a new capsule according to the actual resources classification. These modifications are reflected also at the coordination level, where the *coodination manager* solves user queries adapting on the fly the service composition and invocation according to the actual service availability.

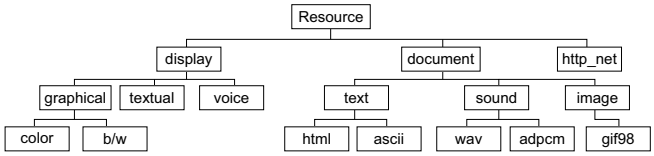

Figure 4: **Ubiquitous Message System.** *the application ontology is used by the system to model the environment.*

The actual implementation of the Ubiquitous Message System allow the use of PalmOS-based PDA, Zaurus PDA, speakers, any kind of personal phones, fixed phones, fax machines, X terminals or mail clients.

Coding UMS in UBIDEV requires to:

- Define the application ontology. An example is shown in figure 4.
- Code the COCA *classifiers* that allow UBIDEV to tag resources with concepts of the *ontology*. For example a portable phone will be classified as instance of *voice, b/w, display* and *Resource*.
- Code the services and their specification. Services will be automatically instantiated according with the actual configuration of the resources federation. The *sound_to_voice* service, for example, requires a *voice display* so when the COCA *classifier* tag a resource as instance of *voice* and *display* concepts, *service manager* will create a *capsule* containing the instantiated service and its EXE represented by the access reference of the tagged resource. The bind between a service and a resource is transparent to the service because of the uniform access granted by the *adapter* and the access protocol. That means a service like *ascii_to_textual* that requires a *textual display* can be equally instantiated on a PDA, on a portable phone or on an X term.
- Code the interfaces that will produce the input queries. The only query generated by the actual implementation of UMS is *document_to_display*. The *coodination manager* will solve this query inside each user context by composing available instances of services; that means, for example, it can compose *ascii_to_wav, wav_to_adpcm* and *adpcm_to_voice* services in order to reach a user context composed of her personal phone (figure 3).

An ongoing work of a UBIDEV compliant coordination models [7], will permit to have a structured composition policy with the introduction of contextual social rules. A typical example of a social rule can be the hierarchical preference of multiple instances of the same service on the basis of user preferences or contextual scenarios (i.e. sms over phone call if the user is in a conference room). Another example is the broadcast service that is realised by the *coodination manager* combining the *document_to_display* service in every user sub-context.

5. Conclusion and Perspectives

This paper has presented how UBIDEV eases the construction and coordination of highly dynamic distributed services without forcing the application to deal with the heterogeneity of the underlying environment.

UBIDEV is not focus on the openness issue but on the model of the environment itself, in particular on the resource and service management. For this reason the model does not directly address aspects related to inter-platform and inter-environment interoperations. Some functional aspect of the inter-platform communication are taken into account but they are out of the scope of this paper (a more exhaustive explanation can be found in [12]).

A prototype of a unified messaging system has been presented in order to underlying the advantages of this system especially for what concerns the application design. This system is far to be completed; further works in the specification and implementation of the architecture will focus on the federation management adapted to the context hierarchy, and on the full integration of the rule-based coordination model [7] . Another important extension is the integration of a structured location model as presented in [4], that will permit to better track the resource location and hence adapt service execution and composition on the base of context spatial information. By the time of writing this paper several projects are under development [1] as testbed for the system.

6. References

[1] Pervasive and artificial intelligence research group. http:// www.unifr.ch/ diuf/ pai/ research/ .

[2] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *17th ACM SOSP*, Kiawah Island, SC, Dec 1999.

[3] S. Avancha, A. Joshi, and T. Finin. Enhanced service discovery in bluetooth. *IEEE Computer*, 35(6):96–99, June 2002.

[4] M. Beigl, T. Zimmer, and C. Decker. A location model for communicating and processing of context. *Personal and Ubiquitous Computing*, 6(5-6):341–357, 2002.

[5] G. Bieber. Introduction to service-oriented programming. http:// www.openwings.org/ download/ specs/ ServiceOrientedIntroduction.pdf. Motorola ISD.

[6] G. Chen and D. Kotz. A survey of context-aware mobile computing, research. Technical Report TR 2000-381, Dartmouth Computer Science, 2000.

[7] M. Courant, A. Tafat, and B. Hirsbrunner. A generic approach of coordination for ubiquitous computing. Technical Report 03-1, Department of Informatics, University of Fribourg, January 2003.

[8] Javaspaces service specification. http:// www.sun.com/ jini/ spec/ js1_1.pdf, Mars 2001.

[9] A. Kaminsky. Jini print service design. http:// www.jini.org/ exchange/ users/ jpgwg/ JiniPrintService/ design20000215/ index.htm, February 2000.

[10] G. E. Krasner and S. T. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system, 1988. ParcPlace Systems, Inc., Mountain View.

[11] Lotus. Unified messaging strategy and mobile services for domino. http:// www.lotus.com/ home.nsf/ welcome/ mobile , 2000.

[12] S. Maffioletti. *UbiDev: A Homogeneous Coordination pace for Ubiquitous Computing Environments*. PhD thesis, University of Fribourg, Department of Computer Science, 2004. to appear.

[13] S. Maffioletti and B. Hirsbrunner. Ubidev: an homogeneous environment for ubiquitous interactive devices. In *Short Paper in Pervasive 2002 - International Conference on Pervasive Computing*, Zurich, Switzerland, August 2002.

[14] Gerenal Magic. Portico. http:// www.genmagic.com /portico, 2001.

[15] S. K. Mostéfaoui and B. Hirsbrunner. Towards a context based service composition framework. In *2003 International Conference in Web Services, ICWS'03,*, pages 42–45, Las Vegas, Nevada, USA, 2003.

[16] M. Romn, C. K. Hess, R. Cerqueira, A. Ranganathan, Roy H. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, pages 74–83, Oct-Dec 2002.

[17] K. Sagonas, T. Swift, and D. S. Warren. Xsb as an efficient deductive database engine. In *SIGMOD. ACM*, 1994.

[18] S. Schubiger. *Automatic Software Configuration*. PhD thesis, Department of Computer Science, University of Fribourg (CH), October 2002. No. 1393. A short version appeared in: S. Schubiger, B. Hirsbrunner. A Model for Software Configuration in Ubiquitous Computing Environments. In Pervasive 2002, International Conference on Pervasive Computing. 26-28 August 2002, Zurich.

[19] J. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, 2002.

[20] A. Tafat, M. Courant, and B. Hirsbrunner. A coordination model for ubiquitous computing. In *3rd WSEAS Int. Conf. on Multimedia, Internet and Video Technologies (ICOMIV 2003)*, Rethymna, Crete Island, Greece, October 13-15 2003.

[21] D. Trastour, C. Bartolini, and J. Gonzalez-Castillo. A semantic web approach to service description for matchmaking services. In *International Semantic Web Working Symposium (SWWS)*, 2001.

[22] Zeroconf and Rendezvous project. Zero configuration networking (zeroconf). http:// www.zeroconf.org, Sep 25 2002.