Department for Informatics
University of Fribourg (Switzerland)

UBIDEV
A Homogeneous Service Framework for Pervasive Computing Environments

THESIS

Submitted to the Faculty of Science of the University of Fribourg (Switzerland) in
conformity with the requirements for the degree of
Doctor scientiarum informaticarum

Submitted by

Sergio MAFFIOLETTI

of
Bergamo, Italy

Thesis No. 1528
Printed at Imprimerie St-Paul Fribourg
2006

Accepted by the Faculty of Science of the University of Fribourg (Switzerland) following the proposal of:

- Prof. Béat HIRSBRUNNER, Université de Fribourg, Switzerland, Thesis Director;
- Michèle COURANT, Université de Fribourg, Switzerland, Examinator;
- Prof. Peter KROPF, Université de Neuschâtel, Switzerland, Examinator;
- Prof. Vaidy SUNDERAM, Emory University, Atlanta GA, USA, Examinator;
- Prof. Stefano CERRI, LIRMM Université de Montpellier II, France, Examinator;

28 June 2006

The Thesis Director: The Dean:

Prof. Béat HIRSBRUNNER Marco CELIO

Ai miei genitori

I carried out this work at the Department of Informatics, University of Fribourg (Switzerland), as a member of the research group PAI. I am grateful to the leaders of this group, B. Hirsbrunner and M. Courant, for their helpful advice, discussions, support, and extreme openness to my work, for which I was granted complete freedom in all respects.

I would then like to thank my examinators, Peter Kropf, Vaidy Sunderam and Stefano Cerri for reading and commenting on my work.

During the years I spent in Fribourg, a number of other people have also brought their more personal contribution to this work. I would like to mention Simon Schubiger, Oliver Hitz, Oliver Buchel, Amine Tafat-Bouzid, Andrś Perez-Uribe, Patricio Lerena, Soraya Kouadri Mostéfaoui, Alessio Gaspar and Pierre Andry who were and are a great help, as colleagues and friends.

My thanks go to my parents for their love and encouragement, and for being always there. I express also my gratitude to my wife Ania and to my son Alessio fort their constant support and patience.

Encouragements and different forms of help, for which I am thankful, came from many friends and colleagues; it is impossible to list them all here.
This work has been supported by the Swiss National Science Foundation (grants 57279 and 65301).

Abstract

This dissertation studies the heterogeneity problem of pervasive computing system from the viewpoint of an infrastructure aiming to provide a service-oriented application model. From Distributed System passing through mobile computing, pervasive computing is presented as a step forward in ubiquitous availability of services and proliferation of interacting autonomous entities. To better understand the problems related to the heterogeneous and dynamic nature of pervasive computing environments, we need to analyze the structure of a pervasive computing system from its physical and service dimension. The physical dimension describes the physical environment together wit the technology infrastructure that characterizes the interactions and the relations within the environment; the service dimension represents the services (being them software or not) the environment is able to provide [Nor99]. To better separate the constrains and the functionalities of a pervasive computing system, this dissertation classifies it in terms of resources, context, classification, services, coordination and application. UBIDEV, as the key result of this dissertation, introduces a unified model helping the design and the implementation of applications for heterogeneous and dynamic environments. This model is composed of the following concepts:

- **Resource:** all elements of the environment that are manipulated by the application, they are the atomic abstraction unit of the model.
- **Context:** all information coming from the environment that is used by the application to adapts its behavior. Context contains resources and services and defines their role in the application.
- **Classification:** the environment is classified according to the application ontology in order to ground the generic conceptual model of the application to the specific environment. It defines the basic semantic level of interoperability.
- **Service:** the functionalities supported by the system; each service manipulates one or more resources. Applications are defined as a coordination and adaptation of services.
- **Coordination:** all aspects related to service composition and execution as well as the use of the contextual information are captured by the coordination concept.
- **Application Ontology:** represents the viewpoint of the application on the specific context; it defines the high level semantic of resources, services and context.

Applying the design paradigm proposed by UBIDEV, allows to describe applications according to a Service Oriented Architecture[Bie02], and to focus on application functionalities rather than their relations with the physical devices.

Keywords: pervasive computing, homogenous environment, service-oriented, heterogeneity problem, coordination model, context model, resource management, service management, application interfaces, ontology, semantic services, interaction logic, description logic.

Sommario

Questa dissertazione studia il problema della eterogeneità nei sistemi pervasivi proponendo una infrastruttura basata su un modello orientato ai servizi. I sistemi pervasivi sono presentati come un'evoluzione naturale dei sistemi distribuiti, passando attraverso mobile computing, grazie ad una disponibilità ubiqua di servizi (sempre, ovunque ed in qualunque modo) e ad una proliferazione di dispositivi (tra cui anche gli utilizzatori) in grado di interagire tra loro e con l'ambiente stesso. Al fine di meglio comprendere i problemi legati allintrinseca eterogeneità dei sistemi pervasivi, dobbiamo prima descrivere la struttura fondamentale di questi sistemi classificandoli attraverso la loro dimensione fisica e quella dei loro servizi. La dimensione fisica descrive l'ambiente fisico e tutti i dispositivi che fanno parte del contesto della applicazione. La dimensione dei servizi descrive le funzionalità (siano esse software o no) che l'ambiente è in grado di fornire [Nor99]. I sistemi pervasivi vengono così classificati attraverso una metrica più formale del tipo risorse, contesto, servizi, coordinazione ed applicazione. UBIDEV, come risultato di questa dissertazione, introduce un modello uniforme per la descrizione e lo sviluppo di applicazioni in ambienti dinamici ed eterogenei. Il modello è composto dai seguenti concetti di base:

- **Risorse:** gli elementi dell'ambiente fisico che fanno parte del modello dellapplicazione. Questi rappresentano l'unità di astrazione atomica di tutto il modello UBIDEV.
- **Contesto:** le informazioni sullo stato dell'ambiente che il sistema utilizza per adattare il comportamento dell'applicazione. Il contesto include informazioni legate alle risorse, ai servizi ed alle relazioni che li legano.
- **Classificazione:** l'ambiente viene classificato sulla base di una ontologia che rappresenta il punto di accordo a cui tutti i moduli di sistema fanno riferimento. Questa classificazione rappresenta il modello concettuale dell'applicazione che si riflette sull'intero ambiente. Si definisce così la semantica di base per tutto il sistema.
- **Servizi:** le funzionalità che il sistema è in grado di fornire; ogni servizio è descritto in termini di trasformazione di una o più risorse. Le applicazioni sono così definite in termini di cooperazione tra servizi autonomi.
- **Coordinazione:** tutti gli aspetti legati alla composizione ed alla esecuzione di servizi così come l'elaborazione dell'informazione contestuale.
- **Ontologia dell'Applicazione:** rappresenta il punto di vista dell'applicazione; definisce la semantica delle risorse, dei servizi e dell'informazione contestuale.

Applicando il paradigma proposto da UBIDEV, si possono descrivere applicazioni in accordo con un modello Service-oriented [Bie02] ed, al tempo stesso, ridurre l'applicazione stessa alle sue funzionalità di alto livello senza intervenire troppo su come queste funzionalità devono essere realizzate dalle singole componenti fisiche.

Contents

List of Figures

Introduction

All technologies have a lifecycle and, as they progress from birth, through adolescence, and on to maturity, their characteristics change. Today's computer technology is mature, it can be taken for granted, so its requirements and purposes are supposed to move from feature-driven to customer-driven.

Nowadays computers should be thought in terms of infrastructure: they should be silent, invisible, unobtrusive and user-centered. These are the starting assumptions to define pervasive computing, a vision of computer technology where computers disappear from sight and consciousness, letting users concentrate on their activities, their tasks. The goal is to "move from the current situation of complexity to one where technology serves human needs invisibly, unobtrusively"[Nor99][Wei91].

With each passing year, computers become faster and more powerful, not to mention smaller, cheaper, and more stylish. The last decade has been characterized by a growing relevance of the role of computer technology in our everyday life. We are experiencing an increasing level of integration of technology in our modern society.

That means that our society more and more rely on machines and, more precisely, on the services offered by this technology.

This phenomenon can be explained observing the direction that the technology is taking towards the user needs. It is maturing from all its aspects: improving from the electrical and electronics viewpoint; improving from the design viewpoint; improving from the application viewpoint and last but not least improving from the interface viewpoint. As a result, from one side, technology is more mature to satisfy the user needs and, on the other side, users are more keen to accept this level of integration.

In the early age of the computer science, technology was very limiting for any kind of user. Computers were room-size dinosaurs used only in very dedicated and specific application domains. Their size, cost and performance, as well as their interface was so limiting that they were used simultaneously by different specialized users (mainframe era). In those times the kind of interaction between human and computer was described by the many_to_one relation: many users for one shared machine.

Technology improvements in the field of electronic miniaturization (VLSI) and the appearing of the first decent GUI, has lead us to the Personal Computer era. Computers became a tool that every user could install in his office, house, room. The reduction of the prices as well as the improvement in the performance and interfaces facilitates a largely acceptance of the computer technology by almost every category of users. This period has been characterized by the one_to_one relation: each user with his own machine.

The relatively recent advent of the Internet technology and the subsequent improvement of distributed systems have promoted a larger acceptance and use of computer technology has a world wide network of computing machines that may cooperate with each other to accomplish a single user's requirement. The current period, called Internet era, is described by the relation one_to_many: one user makes a simultaneous use of many machines.

The actual situation is reinforcing modern visions of the level of coexistence and dependence between machines and human being. This is made possible by the shift of the perception users have about technology: we are moving towards a service-based perception. That means that, as in every other technologies already integrated in our everyday life, computers are more and more vanishing behind the services they offers to the users. The interaction patter is also changing toward a many_to_many paradigm where autonomous entities interact with each other on a peer level. This tendency is well described by the so called pervasive computing vision.

Pervasive computing is an attempt to break away from the current paradigm of desktop computing to provide computational services to a user when and where required. The user becomes one of the interacting partners in an environment where technology, services and context are glued together disappearing from the direct perception, vanishing in the background of our everyday life.

For computers to become entirely invisible, according to Norman[Nor99], two main things need to happen. First, the range of simple, inexpensive information appliances would have to increase significantly. There are still a number of important tasks we do every day that could be done with a computer.

And second, the infrastructure that allows such devices to communicate with each other seamlessly would need to be created.

At the moment, there are a great many wired and wireless technologies for connecting electronic devices, and an even greater number of protocols, formats, and encodings that make communication challenging. For example, a computer can communicate with a cell phone or with a camera, but the camera and the cell phone can't talk to each other. And none of these can communicate with the television set or the stereo.

If we consider this vision from the technological viewpoint, we may note that a new category of challenges is raised and old ones are put in a new light. The conception and the design of a pervasive computing system have to take into account a wider perspective than in classical desktop and distributed systems. The environment for example plays a relevant role in the definition, configuration and execution of a pervasive computing system.

This perspective can be summarized by what has been defined to be the heterogeneity problem for a pervasive computing system [Mat05]. By heterogeneous problem we mean all aspects that have to be considered when designing and implementing an application for a pervasive computing system. Typically it includes the configuration of a system in terms of physical components that belong to the system as well as the software components that run into the system.

As a result an application should be described in terms abstract enough to not to take into account the direct management of the environment that is not directly related to the application's functional behavior and, at the same time, should be able to configure and adapt itself to the specificity of the environment, including its changes.

During the presentation of this thesis we will argument that the most suited and powerful paradigm that can be used to describe an application for a pervasive computing system is to conceive the system as a homogeneous space where services and resources cooperate together to reach the application goal.

With this vision of ubiquitous information access, pervasive computing impacts computing devices and their deployment. In addition to conventional desktop and server computers, pervasive computing environments encompass many different devices of various size and capabilities, including PDA, cell phones, dedicated terminals and robots.

Pervasive computing also changes how people interact with the surrounding computing infrastructure. Differently from conventional computing environments, people focus on their activities and not on the computers. Furthermore, as already pointed out, these computers are often embedded within ordinary objects so they are not meant to be perceived directly. Second, tasks often span many devices, people, and places and task requirements may change frequently due to variations in the execution environment.

The key challenge for developers is to build applications that continuously adapt to such a highly dynamic environment and continue to function even if users move through the physical world and change interface devices, and if the surrounding network infrastructure provides only limited services. However, existing approaches to build distributed applications, including client/server or multitier computing, are ill-suited to meet these requirements.

The fundamental problem is that these approaches try to hide distribution and rely on technologies like RPC or distributed filesystem that enhance single-node programming methodologies to distributed systems. This model does not fit the requirements of pervasive computing systems where the dynamism and the heterogeneity of the environment need to be taken into account even at design level.

Focus of the Thesis

This thesis faces the heterogeneity problem of pervasive computing systems from resources, context and service management viewpoints to allow applications to be described in terms of services provided rather than their low level instantiation details. The aim is to explore the requirements and constrains of the design and of the implementation of a service-oriented application for pervasive computing environments.

This dissertation introduces a system architecture, called UbiDev: a collection of abstraction services for semantic-driven management of the underlying physical environment that provide a design methodology that allows to describe the behavior of the whole system in terms of coordinated homogeneous services.

The main contribution is that UbiDev takes an holistic approach in the management of the environment from the resources, services and context viewpoint. This approach allows applications to be described in terms of their functionalities while maintaining the degree of dependence they have with the physical environment.

Resources and services are considered to be the building blocks an infrastructure uses when providing a coordination model that would allow applications to abstract from the peculiarities of the underlying physical environment and, at the same time, to describe the entire system's behavior in terms of interactions between peer homogeneous entities.

As the physical environment plays a relevant role in influencing the overall system behavior, contextual information has been taken into account at the same level as resources and services.

By focusing on these requirements, our system architecture, in contrast to many previous systems, lets applications instead of users adapt to changes .

For that reason UbiDev focuses on the management and abstraction of resources, services and contextual information providing at coordination level a plethora of homogeneous entities (being them physical devices, users, autonomous robots or software services) and a set of semantic rich primitives to describe their interaction mechanism. What UBIDEV aims to demonstrate is the possibility to describe a pervasive computing application in terms of interacting services operating in a homogeneous and dynamic environment.

Contribution

The contributions of this thesis are:

1. Identify a design model for applications that would express all the fundamental abstractions of a generic pervasive computing environment.

2. Identify requirements to support the building and evolution of pervasive computing applications, resulting in a conceptual framework that both leverages the developers to explicitly deal with the heterogeneity of the underlying environment and take into account the peculiarities and the dependencies each software module may have.

3. Provide programming abstractions to facilitate the design of pervasive computing application: adapters to deal with resource management; capsules to encapsulate the notion of a running service together with its functional and physical dependencies; ontology and classification to abstract the underlying physical environment on the base of a semantic agreed by all the component of the application; a context infrastructure that allow to take into account the information gathered from the physical environment and to use it to steer the application behavior.

4. Provide a coordination model to supervise the service interaction and to provide high level primitives to express the behavior of an application in a Service-Oriented model.

5. Implement a functional architecture and a Proof-of-Concept application.

The value added by these contributions wit respect to advances in current practice is:

- by providing a design process for building applications, our research will give application designers a better understanding of a pervasive computing system as a whole and a methodology for using these abstractions.

- by helping to define a generic classification framework that could be used as a reference point for other middlewares.

- to supports the development of applications through the use of high-level programming abstractions. The goal is to provide an architectural framework to allow designers to prototype applications through a coordination model where all aspects of a pervasive computing system are descried and coordinated.

The architecture with its required set of supporting mechanisms will provide the necessary building blocks to allow others to implement a number of higher-level features for dealing with heterogeneity of resources, services, context and to describe the overall behavior of the system in terms of cooperating services.

On the application side, the provided abstractions will allow designers to build applications that were previously seen as difficult to build: context-awareness that scale along several dimensions, resource and service management that copes with heterogeneity using an agreed semantic, holistic coordination of resources in a service-oriented abstraction model.

Assumptions

Consistent with the hypothesis, the design of UbiDev focuses on meeting the requirements of pervasive computing from the resources, services and context heterogeneity viewpoint. However, it does not address all possible needs of pervasive applications. Hereafter we describe some of the topics that UbiDev does not take into account or takes for granted:

Network

Whit physical layer we intend to categorize all aspects related to the physicality of the environment and of resources that somehow could influence the behavior of the whole system.

Low level network issues like quality of network services, mobile networking support, disconnected operations, bandwidth-adaptive communication protocols, data consistency check, transcoding, energy sensitive memory management, are not directly taken into account by the service framework whose primary goal is to provide at higher abstraction levels (coordination and application) a service-oriented model that reflects the adaptive behavior of the whole system to the stimuli that the surrounding environment produces.

However, low level network issues could be addressed by the classification model. Our research focuses on resource and service management assuming that the underlying physical environment could be entirely classified through the system ontology.

Unconstrained Use of Contextual Information

UbiDev relay on the assumption that the whole environment is trusted and so all the contextual information gathered. No encryption or cryptography is used to protect data exchanged and stored. The infrastructure_to_device communication is always assumed to be trusted in order to ease the implementation phase of the conceptual framework.

Radio Frequency IDentification (RFID) is an automatic identification method, relying on storing and remotely retrieving data using devices called RFID tags or transponders. An RFID tag is a small object that can be attached to or incorporated into a product, animal, or person. RFID tags contain antennas to enable them to receive and respond to radio-frequency queries from an RFID transceiver. Passive tags require no internal power source, whereas active tags require a power source. The widespread deployment of RFID technology poses significant privacy risks not present with prior technology. These privacy issues arise because RFID tags enable tracking of items and people, possibly without their knowledge or consent.

A primary security concern surrounding RFID technology is the illicit tracking of RFID tags. Tags which are world-readable pose a risk to both personal location privacy and corporate/military security. More generally, privacy organizations have expressed concerns in the context of ongoing efforts to embed electronic product code (EPC) RFID tags in consumer products.

A second security concern is duplication, or ”cloning” of RFID tags. When tags do not contain built-in security features, an attacker may be able to scan the tag and ”clone” the data into a tag of her own. This is of particular concern when RFID tags are used in high-security applications, such as the proximity cards used to access secure facilities, or vehicle immobilizer anti-theft systems which use an RFID tag embedded in the vehicle key. It is also a problem when RFID is used for payment systems, such as contactless credit cards.

The security and privacy problem related to the use of RFID technology is just an example of all the implications and risks that should be taken into account when developing a context infrastructure. Identity of users as well as autonomous resources, trust of exchanged information, privacy of stored data are a must when thinking about a system that should really be integrated in our everyday life.

Security

Service frameworks are providing mechanisms by which users and applications can interact and control networked devices and services with unparalleled ease and convenience. Crucial to the successful adoption of such technologies however is security: can applications and users trust the services they find, and can the services trust the clients they serve? Authentication and certification mechanisms are required to ensure a suitable level of trust. Encryption and key distribution protocols are required to protect the exchange of sensitive information. Coupled with the notion of trust is control: the owner of a device might expect to be able to discover and control it and, furthermore, prevent others from maliciously tampering with the device. However, the same owner might also require that the system allows members of their family to control the device, perhaps only while they are present in the room or in a way that the owner approves of (e.g. not allowing a child to switch a television to an adult channel for instance). Although this level of social, role and contextual reasoning is almost certainly outside the scope of a service interaction protocol, such protocols should provide the core mechanisms to facilitate the construction of access control policies (e.g. access control lists, conflict resolution strategies and limitations on the adjustment of device and service state variables for particular clients).

Security is an issue that has not been taken into account in the conceptual framework nor in the implemented architecture. As will be described in next chapters, the modularity of the implemented architecture allows plugging a security infrastructure that would provide the required QoS.

Interfaces

[Sze96] describes four challenges that need to be met by human-computer interface technologies that could be easily applied to pervasive computing context. Interfaces need to be able to automatically adapt themselves to support the users current task. Interfaces need to be able to support multiple platforms. Interfaces should be tailorable to the users current needs. Interfaces should be able to handle both input and output using multiple mechanisms.

Technologies like MVC (Model View Controller) and PAC (Presentation, Abstraction, Control) [Cou87] are both agent based models, where an agent is defined to have state, possess an expertise, and is capable of initiating and reacting to events. [Cou]. An interface is built using hierarchies of agents. These agents represent an object in the application. In MVC, the model describes the semantics of the object, the view provides the (normally visual) representation of the object and the controller handles user input. In PAC, the abstraction describes the functional semantics of the object, the presentation handles the users interaction with the object, both input and output and the control handles communication between the presentation and the abstraction as well as between different PAC agents.

As the main focus of this thesis remain to tackle heterogeneity problem at resource and service level, the whole application interface support, has been defined and implemented as a classical MVC model (few adaptations have been made to further abstract the Viewer notion).

Thesis Outline

CHAPTER 1 presents the heterogeneity problem of pervasive computing environments at different levels of abstraction: resources, services, context, coordination, application and users. In doing so this chapter reviews the state of the art of relevant technologies and approaches; this includes a discussion on why existing support for building applications is not sufficient.

CHAPTER 2 introduces the UBIDEV reference model as a conceptual framework that supports the building and evolution of pervasive computing applications. It also presents the basic abstractions and how they address the heterogeneity problem.

CHAPTER 3 presents an implementation of the conceptual framework described in CHAPTER 2. It introduces the basic component programming abstraction that facilitates the building of pervasive computing applications. The architecture not only contains this implementation, but also includes a set of clear yet simple interfaces to plug new modules to enhance the functionalities of the middleware (an extended context infrastructure or a rule-based coordination model). The holistic management of the environment emerges at coordination level, where all the building blocks are described and coordinated in a service-oriented programming model.

CHAPTER 4 proposes a Proof of Concepts application that has been built with the UBIDEV implemented architecture. The application named Ubiquitous Message System, shows the UBIDEV programming model together with its main advantages in terms of abstraction of the underlying physical environment, semantic-based holistic coordination of resources, context and services.

CHAPTER 5 describes the related research activities that have been carried out within the PAI group during the whole period of this thesis. Their main contributions and their possible interactions and/or intersections with UBIDEV are also presented.

CHAPTER 6 presents the main related research works. Each of them is described and compared with UBIDEV using a common metric.

Finally, CHAPTER 7 contains a summary and conclusion with suggestions for future research.

Part I

The Heterogeneity Problem in Pervasive Computing

Chapter 1

Heterogeneity in Pervasive Computing

A distributed system consists of several computers that communicate over a network to coordinate the actions and processes of an application. Distributed systems have gained interest in recent years due to the proliferation of the Web and other Internet-based services. From distributed systems through mobile computing, pervasive computing could be considered as a step forward in ubiquitous availability of services interacting with autonomous entities and with the surrounding environment.

1.1 From Distributed System to Pervasive Computing

Well established techniques such as interprocess communication, remote invocation, naming services, cryptographic security, distributed file systems, replica services, and distributed transaction mechanisms provide the run-time infrastructure supporting today's network applications [CDK00]. The reference model is still the client-server architecture. However, applications for distributed systems rely more and more on middleware support through the use of software frameworks that provide higher level abstractions such as distributed shared objects and distributed communication, on services including secure communication, authentication and persistence and on models for describing the coordination of autonomous services.

What we are experiencing today is that distributed applications supports more and more mobile code, multimedia data stream, user and device mobility, spontaneous networking, service interoperability and context awareness [MS03]. Scalability, quality of service and robustness with respect to partial component failures is becoming a key issue because the size of a distributed system is growing in both the number of physical components involved and the number of services interacting. A shift towards large-scale systems has occurred in recent years: Internet with its protocols and the higher level World Wide Web are examples of the de-facto standard platforms for distributed applications. Here, the internet and its resources are viewed as the global environment in which computation takes place. Consequently, high-level protocols and standards such as XML, enter the focus of distributed systems research while low-level issues (such as operating systems peculiarities) become less important. The increasing number of computers connected to the Internet has also laid the foundation for new application domains such as grid computing [FK03] and per-to-peer computing [FK03]. Grid computing emphasizes the fact that the Internet can be viewed as a globally distributed computer with an enormous potential for computing power. In contrast to this, peer-to-peer computing emphasizes the immediate and unrestricted information exchange.

The rapid evolving of network and computer technology and the exponential growth of the services and information available on the Internet will bring us to the point where hundreds of millions of users have fast, pervasive access to a phenomenal amount of information, through desktop machines at work, school and home, through mobile phones, personal digital assistants, car dashboards, from anywhere and everywhere: the pervasive computing era [Wei91] [KG99].

This emerging scenario raises its own technical difficulties and requires specialized areas of research. Taking a broader view, there is the need to reexamine some fundamental assumptions about the structure of interactive systems and integrated environments. Today's conventional model of interaction and device communication has served us well up until now, but will have to evolve towards a different architecture, focused on multiple users in an interaction space rather than focusing on systems as a network of processors and devices. Moreover the cooperation of hundreds of thousands of resources and services will call for a coordination model that is scalable, flexible and able to take into account the granularity of the surrounding environment.

This chapter introduces the heterogeneity problem in pervasive computing structured by a conceptual model that captures the nature of a generic system, consisting of: resources, context, services, coordination, application and user. Starting from the infrastructure viewpoint, the chapter formalizes the basic functionalities and services that a pervasive computing environment should provide.

Then heterogeneity is discussed according to the classification defined by the conceptual model: for each abstraction concept, the main challenges heterogeneity rises are presented; then a survey on the state of the art of existing technologies and the way they cope with heterogeneity is described.

As a conclusion, the resulting scenario should already suggest the fundamental ideas that have guided the UbiDev model.

1.2 Pervasive Computing

To better understand what differentiate pervasive computing from classical distributes system we will first present a usage scenario. Today's mobile workforce is using mobile tools for communication and computing. To stay connected and keep up with their workflow while on the road, many workers have turned to smart phones, personal digital assistants (PDAs), notebook computers, and other portable devices that provide network and Internet connectivity. While on the road, users may want to access enterprise applications and databases. When they come back home, they will likely want to upload information from their portable devices into their primary workstations or onto the company network.

Pervasive computing is about connecting a wide variety of client devices (such as PDAs, cellular phones, automotive computers, home gateways, wearable computers, as well as traditional PCs) to a modern Web environment, and enabling interaction to occur via technology which is virtually invisible to the end user. These activities raise critical access and security concerns. Integration is a particular problem, because the portable devices operate on a variety of platforms and use a variety of communications protocols. Pervasive computing enables a broad range of end-user devices to access data and applications on servers, much like today's PC access to HTML sources across the Internet. Accesses to both personal and organizational information from different devices, products that are intuitive, and network access that is invisible to the end user are the basic requirements.

However, the new devices differ in their input, output, and processing capabilities; they differ in the content format and user interface paradigms. Most significantly, they also differ in the interaction patterns and their typical usage scenarios. Consequently, providing adequate services to such devices is more complex than providing services to uniformly configured PC-based HTML browsers.

The new pervasive computing paradigm enables a single user to use multiple clients connected to multiple servers in the network, and to do so in a consistent and natural way.

This model is enabled by bringing clients and servers in a uniform coordination environment where dynamism and heterogeneity of the underlying physical environment is somehow blurred by an infrastructure. This is a logical evolution from client/server that supported a client connected to a single server to network computing that supported a client connected to multiple servers in the network.

Another key aspect in pervasive computing is the surrounding environment. In traditional computing, the physical environment is regarded as irrelevant unless it becomes incompatible with the operation of the computer. The computer is logically self-contained and users can be viewed as simply source of input for process. Pervasive computing is ubiquitous, interconnected and dynamic which implies that it is embedded, distributed and mobile. The environment cannot be ignored; it must be factored into the conceptual model. The embedded aspect of pervasive computing suggests that sensors and actuators will play a significant role in many applications. This implies that there will be issues that exist beneath the physical layer of the computer, beyond the sensors and actuators.

Finally, the mobile nature of many pervasive computing systems ensures that the environment's presence determines the semantics of the system.

A Definition

We define a pervasive computing system as a physical space coordinated by a proactive software infrastructure that enhances the capabilities of applications to configure and adapt the physical and software dimension automatically and transparently with respect of the user. This definition underlies three main elements composing a pervasive computing system:

- **Physical environment** as the physical dimension of the system. This dimension describes all the physical entities belonging to the environment during the entire lifecycle of the system. As will be discussed in the next chapters, users are considered to be one of the interacting resources belonging to the environment.

- **Software infrastructure** that provides the interconnection framework for the element of the environment and a coordination model to allow these elements to cooperate on a common goal.

- **Application** that represents the functional aspect of the system. An application is structured in two parts: interaction and application logic [SVSF05]. Interaction defines the interchange between the elements of the environment and the system; the application logic handles the notion of services and their coordination.

Unlike traditional desktop applications with a graphical user interface, pervasive computing forces us to take a rather holistic view of a system.

In traditional GUIs, the interactive system is the desktop computer and a fixed set of input/output devices. The emphasis is on combining software components to provide services to the user.

pervasive computing on the contrary refers not only to software services but also to devices and how to describe their interactions.

What are the inherent features of a system that make it a pervasive computing system? According to [Wei91], pervasive computing is characterized by two main attributes:

- **ubiquity**: interaction with the system is available wherever the user needs it;
- **transparency**: the system is non-intrusive and is integrated into the everyday environment.

A refinement of such a characterization [SDA98], introduces the notion of user mobility dimension to reflect the freedom the user has to move about when interacting with the system: desktop computing, for example, allows no user mobility. Some systems allow more freedom of movement to the user when interacting with the system [WFG92]. When using a system based on a standalone portable device, user mobility is unconstrained. When the portable device relies on an infrastructure to provide services, the mobility of the user is usually constrained by the coverage zone of the infrastructure [FMS93]. For example, systems relying on the GPS service constrain mobility because GPS is not available indoors or may be obscured by buildings or tunnels.

Transparency applies to the system's interface and reflects the conscious efforts and attention the system requires of the user, either for operating it or for perceiving its output. Most interfaces today lack interaction transparency. To perform a task with the system, the user must consciously perceive, understand and manipulate an interface which is conceptually separate from the task being performed. The graphical user interface remains in the focus of the user throughout the interaction. A transparent interface disappears from the user's focus so that he/she can concentrate on the actual task at hand.

1.3 Infrastructure for Pervasive Computing

Requirements for a pervasive computing infrastructure are centered on a high-level conceptual model consisting of resources, users, context, services, coordination models and applications level interfaces.

Network level issues have to be taken into account when planning an effective infrastructure to allow seamless mobility of resources and users. The network as a whole may have broad coverage, but connection quality and services vary greatly from location to location. The connection may vary from wired or infrared in-room networks with great performance, to metropolitan cellular networks, to satellite networks with high latencies but tremendous coverage. A pervasive computing infrastructure should offer solutions for allowing resources to seamlessly roam both within network and across heterogeneous networks. The detection and setup of network connections should be transparent and automatic, as the selection of the best network in range.

Resource heterogeneity implies also differences in shape, capabilities, power and usability; an infrastructure must be able to recognize such diversities in order to adapt the services it provides and the services that it controls on behalf of an application. Resource is one of the key aspects in adaptability because, basically, they represent the endpoint used by the application to provide its functionalities. The role of the infrastructure is to provide to the application an easy way to describe the adaptation patterns without having to deal directly with resource management. That also means to take into account the role and even the intention that some resources may have in a given environment: take the example of an autonomous mobile robot; when it enters a new environment it should be able to negotiate with the application the services to use as well as provide information about its goal. That implies that the infrastructure should be able to manage and describe such resource to the application in a way that is compliant with the application knowledge.

Context represents the main shift from classical distributed systems because in pervasive computing the surrounding physical environment is explicitly taken into account by both the infrastructure and the application to adapt the behavior of the whole system to the caching that occurs during the lifecycle of the system. The increasing use of both active and passive RFID technology as information sensors allows systems to receive and transmit information to and from physical objects. That means that everyday objects could play an active role in a pervasive computing environment and, at the same time, an application could be tuned to a class or category of everyday objects [GBK99].

Services vary greatly as well: from home/office printer access, to local driving directions, to global services such as search engines and web access. Services tend to relay on a given hardware configuration for their execution; they have resource requirements that should be met to ensure their correct execution. Most of the time the coupling between software components and the hardware involved is so tight that the notion of a service embodies the two. The infrastructure should allow describing the fundamental interrelation between software components and resources while keeping their coupling in terms of functional dependencies. That should also be reflected at application design level.

A pervasive computing infrastructure should be highly available, cost effective, and sufficiently scalable to support millions of users. In general, computation, storage and complexity should be moved from the resources into the infrastructure, thus enabling powerful services, better overall cost performance, and small, light-weight, low-power, inexpensive mobile devices to increase functionality [BKA+98].

As a result, a pervasive computing system will be composed of different services and resources interacting with each other. A coordination model is required to formalize such interactions as well as the dependencies between the coordinated entities.

Finally an infrastructure should provide functionalities to allow application interfaces to express the adaptive and autonomous behavior of the whole system when interacting with the users and with other resources. The responsibility of the infrastructure with respect to applications includes supporting application requirements such as context awareness, adaptation, mobility, distribution and interoperability; facilitating the rapid development and deployment of software components; providing component discovery services; and providing scalability.

Adaptation

Adaptation is required in order to overcome the intrinsically dynamic nature of pervasive computing. Mobility of users, devices and software components can occur, leading to changes in the physical and virtual environments of these entities. Moreover, applications can be highly dynamic, with users requiring support for novel tasks and demanding the ability to change requirements on the fly. It should be the role of the infrastructure for pervasive computing to facilitate adaptation, which may involve adapting individual software components and/or reconfiguring bindings of components by adding, removing or substituting components. Adaptation may be done in an application-aware or application-transparent manner [Nob00] [GMGN04]. Dynamic adaptation can involve complex issues such as managing the adaptation of software components that are used simultaneously by applications with different (and possibly conflicting) requirements, and maintaining a consistent external view of a component that has behavior that evolves over time.

Adaptation is necessary when there is a significant mismatch between the supply and demand of a resource. The resource in question may be wireless network bandwidth, energy, computing cycles, memory, and so on. There are three common strategies for adaptation in pervasive computing: first, a client can guide applications in changing their behavior so that they use less of a scarce resource. This change usually reduces the user-perceived quality, or fidelity, of an application. Odyssey [FS99] [NSN$^+$97] is an example of a system that uses this strategy. Second, a client can ask the environment to guarantee a certain level of a resource. This is the approach typically used by reservation-based QoS systems [NCN98]. From the viewpoint of the client, this effectively increases the supply of a scarce resource to meet the client's demand. Third, a client can suggest a corrective action to the user. If the user acts on this suggestion, it is likely (but not certain) that resource supply will become adequate to meet the demand.

Smart spaces such as [JF04] and [Rek98] are examples of environments capable of accepting resource reservations. At the same time, uneven conditioning of environments suggests that a mobile client cannot rely solely on a reservation-based strategy - when the environment is uncooperative or resource-impoverished, the client may have no choice but to ask applications to reduce their fidelities. Corrective actions broaden the range of possibilities for adaptation by involving the user, and may be particularly useful when lowered fidelity is unacceptable.

Metacomputing Abstraction

Metacomputing environments [CS92] such as [MS99] and [KUB00] are component-based: the heterogeneous computing environment is aggregated and a concurrent programming platform emulated through a set of coordinated components. Software components define the functionalities (services) they provide and their interfaces. Through the composition and the coordination of such components the heterogeneous environment is aggregated within a concurrent programming platform. Component-based metacomputing frameworks utilize the concept of a Distributed Virtual Machine (DVM) [SGGB99] [BDF$^+$99] as an abstract boundary for a collection of active components. The DVM defines a name space and provides generic discovery mechanisms.

Metacomputers hide the existence of multiple computers and provides a single-system image to its users. Differently from a Network Operating System approach [TvS02] where a user is fully aware of the machines on which his job is executed, metacomputers dynamically and automatically allocate jobs to the machines of the system. The key concept behind these features is ”transparency”. Metacomputing, if conceived as a Distributed Operating System, supports several forms of transparency to achieve the goal of providing an abstraction of networked machines as a metacomputer. Harness [BDF$^+$99] is a good example of a system conceived to centralize the management of the underlying resources providing a uniform abstraction to the applications and users. In a pervasive computing system applications can greatly benefit from knowing some relevant functional details of the computational environment. That could allow them to configure themselves and adapt to every heterogeneous and dynamic aspect of an environment.

In pervasive computing an infrastructure is motivated by the need to federate the power of distributed resources into a single virtual compute space, which can be used to run applications whose requirements could not be met a priori but have to be considered according to the current configuration of the computational power of the compute space. This is very similar to what is happening in HPC [FK97], [FT89] where the effort is to provide a seamless access to computational resources collected into a virtual Grid [FK03].

In the following sections, heterogeneity is presented at the different levels that characterize a generic pervasive computing system with a focus on the role that an infrastructure should have in facing it.

1.4 Resources

Heterogeneity in computing systems is not meant to disappear in the future, but instead will increase as the range of computing devices increase. Devices in a pervasive computing environment will include sensors and actuators that mediate between physical and virtual environments; embedded devices in objects such as watches and shoes; home and office appliances such as videos, toasters and telephones; mobile devices, such as handheld organizers and notebooks; and traditional desktop machines. Heterogeneous devices will be required to interact seamlessly, despite wide differences in hardware and software capabilities. This will require an infrastructure that maintains knowledge of device characteristics and manages the integration of devices into a coherent system that enables arbitrary device interactions (for example, between a mobile phone and a desktop workstation).

If we consider devices used by the user to interact with the system, they can range from standard ones such as laptops, PDAs, and phones, to emerging ones such as those embedded in clothing and eyeglasses. The variety of available devices has several implications. One is the kind of input-output devices: textual and graphic input-output will not be the only forms of human-machine interaction. Audio, visual, and other sensory modes of communication will be prevalent. Another implication is the requirement that the environment must be prepared to adapt to the device currently used by the user. For example, if the user is requesting information and is currently driving, the retrieved data should be relayed to him with an audio message through the car radio. Three level of heterogeneity are discussed here:

- **Physical**: For a given cost and level of technology, weight, power, size and ergonomics represent a limitation with respect of computational resources such as processor speed, memory size, and disk capacity. While mobile elements will improve in absolute ability, they will always be resource-poor relative to static elements.

- **Communication**: some buildings may offer reliable, high-bandwidth wireless connectivity while others may only offer low-bandwidth connectivity. Outdoors, a mobile client may have to rely on a low-bandwidth wireless network with gaps in coverage. Over time, the synchronous model implicit in the use of RPC will become inadequate. What is required is a reliable transport layer that works with legacy servers, while hiding the effects of wireless losses and asymmetry that typically ruin TCP performance. Eventually, very wide-area distributed systems will have to be structured around an asynchronous model.
- **Power**: While battery and energy production technology will undoubtedly improve over time, the need to be sensitive to power consumption will not diminish. Concern for power consumption must span many levels of hardware and software to be fully effective.

Mobility

As users can be mobile and able to exploit the capabilities of several devices simultaneously, mechanisms will be required to enable the mobility and distribution of software. These mechanisms should be largely transparent to component developers, who should not be concerned with program and data migration or synchronization and coordination of distributed components. Deal with mobility goes beyond the current support for code migration provided by platforms such as the Java virtual machine (JVM) [LY99] as run-time migration in heterogeneous execution environments will be required. Similarly, the support for distribution will need to surpass the functionalities provided by platforms such as CORBA [gro02], which only offer transparency of distributed communication, and typically do not address mobility, synchronization or coordination.

Mobility introduces problems such as the maintenance of connections as devices move between areas of differing network connectivity, and the handling of network disconnections. While protocols for wireless networking handle some of the problems of mobility, such as routing and handovers, some problems cannot be solved at the network level, as they require knowledge of application semantics. It should be the role of the computing infrastructure to cooperate with applications in order to perform tasks related to device mobility, such as management of replicated data in cases of disconnection.

Consider remote control of a robot explorer on the surface of Mars. Since light takes many minutes to travel from earth to Mars, and emergencies of various kinds may arise on Mars, the robot must be capable of reacting on its own. At the same time, the exploration is to be directed live by a human controller on earth – a classic command and control problem. This example characterizes a distributed system where communication latency is a limitation and a synchronous design paradigm will not work.

Mobility increases the tension between autonomy and interdependence that is characteristic of all distributed systems. The relative resource poverty of mobile elements as well as their lower trust and robustness argues for reliance on static servers. But the need to cope with unreliable and low-performance networks, as well as the need to be sensitive to power consumption argues for self-reliance. Any approach to mobile computing must balance between these issues. This balance cannot be a static one in fact as the circumstances of a mobile client change; it must react and dynamically reassign the responsibilities of client and server. In other words, mobile clients must be adaptive.

At one extreme, adaptation may be considered to be entirely under the responsibility of individual applications. While this approach avoids the need for system support, it lacks a central arbitrator to resolve incompatible resource demands of different applications and to enforce limits on resource usage. It also makes applications more difficult to write and adaptation is even worse.

The other extreme of application-transparent adaptation places entire responsibility for adaptation on the system. This approach is attractive because it is backward compatible with existing applications: they continue to work when mobile without any modifications. The system provides the focal point for resource arbitration and control. The drawback of this approach is that there may be situations where the adaptation performed by the system is inadequate or even counterproductive because it operates without taking into account the application's perspective.

There is a clear need of dynamic networks where resources are allowed to appear and disappear; this behavior is most common in networks which incorporate wireless technologies such WLAN and Bluetooth. Many protocols like IPX [McL89] incorporate technologies to dynamically configure a network without central configurations servers. However many of these network protocols in the last years have been replaced by IP networks. The original IP architecture does not incorporate the notion of dynamism and automatic configuration.

One of the shortcoming of the current IP version 4 (IPv4) [Pro81] is the address space. The current 32-bit addressing used within IP is not allocated well, resulting in huge gaps of unused addresses. Also, the Internet is expanding at an exponential rate, with many more devices getting added to the network every day.

Another problem is the lack of security and authentication. IP does not encrypt packets. It is not possible to digitally sign a transmission. Today's approach is to use an additional add on software to encrypt packages transmitted.

IP version 6 (IPv6) [Pro97], provides 128-bit addressing (that's billions upon billions of addresses), compatibility with IPv4 addresses, security and authentication, quality of service (reserving bandwidth), plug-and-play for network device configuration, hierarchically structured routing and an ability to seamlessly integrate with the current IP during the transition stages.

An IPv6 address is represented as 8 16-bit numbers in hexadecimal, like

FEDC:BA98:0:0:0:BA98:7654:3210

Also, an IPv6 address can be dynamically built when a new device is plugged in, by sensing the network address, and then using the device Ethernet card's address (or whatever interfaces hardware address) to build an IPv6 address.

Thus the resources can reconfigure automatically when moving from place to place. This also means that an IPv6 address will change when a resource moves to another ISP (this is currently effective due to CIDR - classless InterDomain routing - which was invented to temporarily surmount the addressing and huge routing tables problem).

Physical Resources and the Surrounding Environment

In the MediaCup project [GBK99], active sensor tags are embedded into everyday objects to derive and provide services based on the situational context of users. For example, the context "meeting" can be inferred from the presence of many hot cups in a meeting room. However, this project assumes stationary access points that facilitate the cooperation between active artifacts and existing infrastructure services.

Many popular pervasive computing scenarios and research efforts are based on computer-augmentation of consumer appliances. In this kind of projects, everyday objects are enriched with tags so they can be referenced in computer applications without embedding computing capabilities. A well-suited tagging technology are RFID tags as they can be attached post hoc and in unobtrusive ways to everyday artifacts. Tagging though keeps computing away from artifacts and stays short of enabling artifacts as more autonomous components in pervasive computing environments.

Beyond tagging there are other approaches based on embedding computing capabilities in everyday artifacts. The Things That Think consortium at MIT Media Lab[1] have produced a wide range of computer-augmented artifacts. Also notable are computer-augmented toys with embedded sensors, processors and actuators, for example the Sony pseudo-dog AIBO[2], that goes more in the direction of autonomous robot technology.

All these approaches have been focused on local functionalities and have been thought for a given context. Interoperability with other environments as well as openness has not been yet taken into account.

Another important issue that an infrastructure should consider when integrating these augmented appliances is how to allow a system to configure, access and, eventually, adapt when a new resources enters the environment. In such a scenario the systems is willing to integrate the new appliance giving it an appropriate identity that characterizes its role within the environment; that means that services, contextual information and access rules are defined according to the given role. This is not just a matter of communication protocol but, also, a semantic challenge because the description of the new resource should be given according to the semantics of the hosting system. This is still one of the main challenges of pervasive computing as well as distributed systems.

1.5 Services

"A service is a self-contained, stateless function which accepts one or more requests and returns one or more responses through a well-defined, standard interface. Services can also perform discrete units of work such as editing and processing a transaction. Services should not depend on the state of other functions or processes. The technology used to provide the service, such as a programming language, does not form part of this definition."[3].

A Web Service is a programmable application logic accessible using standard Internet protocols. Web Services combine the best aspects of component-based development and the Web. Like components, Web Services represent black-box functionality that can be reused without worrying about how the service is implemented. Unlike current component technologies, Web Services are not accessed via object-model-specific protocols, such as DCOM, RMI, or IIOP. Instead, Web Services are accessed via ubiquitous Web protocols (ex: HTTP) and data formats (ex: XML).

[1]http://ttt.media.mit.edu/
[2]http://www.sony.net/Products/aibo/
[3]wikipedia definition: http://en.wikipedia.org/

The Web services architecture [WS-03] defines a service-oriented distributed computing model in which services interact by exchanging XML documents. The basic elements of the Web services architecture define the syntax for information exchange.

The W3C Web services Architecture working group provides the following definition [WS-03]:

- A **Web service** is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Service-Oriented Programming builds on Object Oriented Programming, with the assumption that problems can be modeled in terms of the services that an object provides or uses. Service-Oriented Programming prescribes that programming problems can be seen as independently deployable black boxes that communicate through contracts. The traditional client-server model often lacks well-defined public contracts that are independent of the client or server implementation. In Service-Oriented Programming, components publish and use services in a peer-to-peer manner: a client is not tied to a particular server; instead, service providers are interchangeable.

A Service-Oriented Architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity.

A service-oriented architecture defines a distributed system wherein agents, known as services, coordinate by sending messages. Quoting [WS-03] once again:

- **SOA** is a specific type of distributed system in which the agents are "services". Service is a software agent that performs some well-defined operation (i.e., "provides a service") and can be invoked outside of the context of a larger application. That is, while a service might be implemented by exposing a feature of a larger application the users of that server need be concerned only with the interface description of the service. "Services" have a network-addressable interface and communicate via standard protocols and data formats.

The first service-oriented architecture for many people in the past was with the use DCOM [dco] or Object Request Brokers (ORBs) of CORBA [gro02].

The technology of Web services is the most likely connection technology of service-oriented architectures. The Web services architecture has been broadly accepted as a means of structuring stateless interactions among distributed software services.

The figure 1.1 illustrates a basic service-oriented architecture. It shows a service consumer at the right sending a service request message to a service provider at the left. The service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. A service provider can also be a service consumer.

Semantic Web

Semantic Web is a vision of the next generation World Wide Web [BLHL01]. Research efforts in the Semantic Web are driven by the need for a new knowledge representation frame-

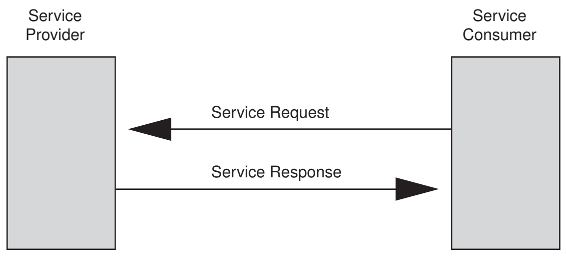

Figure 1.1: **Service Model**.

work to cope with the explosion of unstructured digital information on the existing Web. The present Semantic Web research focuses on the development of ontology languages and tools for constructing digital information that can be understood by computers [BLHL01]. A goal of the Semantic Web initiatives is to develop languages that are adequate for representing and reasoning about the semantics of information on the Web. The Web Ontology Language OWL is the latest standard proposed by the Web-Ontology Working Group. The OWL language builds on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data [OWLa].

The OWL language shares the root with its predecessor DAML+OIL [dam] (e.g., using RDF as the modeling language to define ontological vocabularies and using XML as the surface syntax for representing information [OWLb]).

OWL is a language for defining and instantiating ontologies. In such a context an ontology is a formal explicit description of concepts in a domain of discourse (or classes), properties of each class describing various features and attributes of the class, and restrictions on properties [OWLc]. OWL ontologies are usually placed on web servers as web documents, which can be referenced by other ontologies and downloaded by applications that use ontologies.

Services in Pervasive Computing

As pervasive computing environments are very heterogeneous due to a variety of computing devices and networks, a key element in these environments is service discovery. Service discovery protocols for pervasive computing allow clients to locate services even when facing mobility and heterogeneity.

The services and resources one wishes to locate are no longer constrained to local printers and scanners, but include online shopping services, publicly accessible web-cams, weather instruments and data collected by a set of sensors deployed in remote regions.

While today's service discovery protocols operate in small, local-area settings or in wide-areas, none work in both these environments.

Coping with the constraints of a generic pervasive computing system requires us to rethink client-server model. A fixed distinction between clients and servers will not hold anymore. The resource limitations of clients may require certain operations normally performed on clients to be performed sometimes on resource-rich servers.

Conversely, the need to cope with uncertain connectivity requires clients to sometimes emulate the functions of a server. These are, of course, short-term deviations from the classic client-server model for purposes of performance and availability. From the longer-term perspective of system administration and security, the roles of servers and clients remain unchanged.

Service Discovery

Discovery is a term used to describe the protocols and mechanisms by which a network connected device or software service becomes aware of the network to which it is connected and discovers which network services are available. For example, a PDA needs to discover the home network, and find a service that will provide PDA to PC synchronization capabilities and use that synchronization service. The issue of discovery of software components has been addressed in various research areas. In open distributed computing, discovery is supported by a type management repository, which maintains descriptions of service interface types, and a trader, which is aware of instances of service types.

Service discovery can be all pre-configured. This is the solution that techniques such as DHCP, DNS and LDAP provide for enterprise networks. For a relatively static system with infrequent addition of new devices or software services, this may be a viable approach. The configuration step can be done once during system installation, and manually maintained, usually by a skilled system administrator. For relatively static networks where central administration is needed or desirable, this sort of pre-configured service discovery may be appropriate.

However, within the networked home, new information appliances will be purchased and added to the network with some frequency. Mobile devices, such as cellular phones and PDAs, can enter and leave a home network quite frequently. In these situations, it is difficult to rely on manual configuration of the network services. So, we need service discovery in the home, mobile, and similar environments to be self-configuring.

There are many existing service discovery protocols that have varying degrees of self-configuration, and more are being announced all the time. For the most part, these protocols are incompatible. This is natural because the art has not yet achieved maturity. If different device and network service manufacturers adopt different service discovery protocols, then it will be difficult for devices built for one protocol to work with services provided by devices built FOR different protocols.

Let's examine some of the most important service discovery protocols.

Important Service Discovery Protocols

- **Salutation**: is an architecture for looking up, discovering, and accessing services and information. The Salutation architecture [Inc99] is being developed by an open industry consortium, called the Salutation Consortium; it defines abstractions for devices, applications, and services; a capabilities exchange protocol; a service request protocol; "personalities" (standardized protocols for common services); and APIs for information access and session management. The architecture consists of service brokers called Salutation Managers (SLMs). Services register their capabilities with an SLM, and clients query the SLM when they need a service. Salutation is independent on the network technology and may run over multiple infrastructures, such as over TCP/IP and IrDA. It is not limited to HTTP-over-UDP-over-IP. Moreover,

- **SLP**: Service Location Protocol (SLP) [Gut00] is a standard developed by a working group of the Internet Engineering Task Force (IETF). SLP addresses the problem of self configuring service discovery by applying existing Internet standards to the problem. SLP is designed to be a lightweight, decentralized protocol with minimal administration requirements. SLP offers a flexible and scalable architecture and the utilization of service templates make service browsing and human interaction possible. Since SLP is able to operate with or without a Discovery Agent, it is suitable for networks of different sizes, ranging from very small ad hoc connectivity to large enterprise networks. SLP also includes a leasing concept with a lifetime that defines how long a discovery agent will store a service registration. DHCP options to configure SLP are already defined. LDAP (Lightweight Directory Access Protocols) servers could be also used as a backend for SLP, i.e., DAs may use LDAP servers as their repository.

- **Universal Plug and Play**: (UPnP) is an initiative to bring easy to use, flexible, standards based connectivity to consumer networks, whether in the home, in a small business, or attached to the global Internet [Cor05]. From a technology perspective, UPnP is a suite of protocols and system services for device discovery and control in small and medium size IP networks. In UPnP environments the Simple Service Discovery Protocol (SSDP) [GCL$^+$99] runs on top of the IP layer to provide a means of discovering devices on the network. SSDP allows devices to advertise services using datagram sent using IP multicast. These advertisements contain a service type and a URL for the service being advertised. Clients interested in accessing services can either wait for announcements or can multicast a search request that forces all devices on the network to send service advertisements. Once a client has obtained an URL for a service, it can retrieve from the device an XML description of the service being offered. This description will include, among a host of other pieces of information, an additional URL that can be used to access a web page representing the user interface for the device and a list of state variables associated with the service. In more detail, in UPnP devices are modeled as objects with an associated state table (c.f. properties). This state table provides an external representation of the objects internal state. Clients can make changes to this state table that causes the associated object to invoke operations to achieve a corresponding change in its internal state. For example, an object representing a camera might have a state table that included a variable autofocus with possible values of on and off. Clients that wished to change the cameras autofocus mode would simply need to change the value of the autofocus variable. The architecture requires that objects generate events whenever their state changes and that clients register for these events in order to ensure all views on the device are consistent (for example, to ensure consistency between the cameras front panel and a remote application also controlling the camera). It is worth highlighting at this point that UPnP compatibility is defined in terms of the on-wire format used for messages and the architecture is OS and language independent. Many of the service discovery and interaction technologies operate in a similar way, and have a number of features in common with UPnP.

- **Bluetooth**: [AJF02] is a consortium developing a short-range wireless communication protocol. It is optimized for the highly dynamic nature of Bluetooth networks, and is a simple, efficient protocol that allows Bluetooth devices to discover services offered by or through other Bluetooth devices. The Bluetooth protocol stack contains the Service Discovery Protocol (SDP), which is used to locate services provided by or available via a Bluetooth device. SDP is described in the Bluetooth specification part E [blu].

It addresses service discovery specifically for this environment and thus focuses on discovering services, where it supports the following inquiries: search for services by service type; search for services by service attributes; and service browsing without a priori knowledge of the service characteristics. SDP does not include functionality for accessing services. Once services are discovered with SDP, they can be selected, accessed, and used by mechanisms out of the scope of SDP, for example by other service discovery protocols such as SLP and Salutation.

Limitations of Existing Technologies

Existing service discovery architectures have a few limitations which makes them unsuitable for wide deployment in the pervasive computing domain.

Some of these protocols like SLP and Salutation are deployed primarily within the enterprise or office environment; others like UPnP and Bluetooth were conceived for a more informal, casually connected environment, which could include networked vehicles and small offices as well as home networks. Consequently, a networking solution should be able to accommodate heterogeneity, both in terms of underlying connectivity, and in terms of the discovery infrastructure that is supported.

The infrastructure for pervasive computing must support diverse types of software component. It should be able to integrate software components, which may reside in fundamentally different environments (such as home or office computing environments), into compositions that can successfully interact and cooperate to achieve common tasks. As pervasive computing environments will be required to respond to novel tasks and situations, applications will increasingly be formed dynamically from available software components. This requires dynamic interoperability at the component level, in addition to interoperability that overcomes the heterogeneity of the environment and of components. Components will need to be capable of dynamically acquiring knowledge of each other's interfaces and behavior, in order to learn how to interact with previously unknown components.

Services are heterogeneous in nature. These services should be defined in terms of their functionalities and capabilities. The functionality and capability descriptions of these services should be used by the service clients to discover them. The existing service discovery infrastructures lack expressive languages, representations and tools that are good at representing a broad range of service descriptions and are good for reasoning about the functionality and the capabilities of the services.

In the existing service discovery infrastructures, it is impossible to find services which require a specific attribute value that can change based on the dynamic content of the environment. In addition, service functionality is described at the syntax level or object level. This makes difficult to apply approximate matching rules. For example, if the client is attempting to discover a black and white printer, an approximate matching rule would succeed on finding a color printer.

Services need to interact with clients and other services across environments and, probably, applications. Service descriptions and information need to be understood and agreed among various parties. In other words, well-defined common ontology must be present before any effective service discovery process can take place. Common ontology infrastructures are often either missing from or are not well represented in the existing service discovery architectures. Architectures like Service Location Protocol, Jini and Salutation do provide some sort of mechanisms to capture ontology among services. However, these mechanisms like Java class interfaces and ad-hoc data structures are difficult to be widely adapted by the industries to become standards. In the Universal Play and Plug (UPnP) architecture, service descriptions are represented in XML (eXtensible Markup Language), which provides a good base foundation for developing extensible and well-formed ontology infrastructure. However, a service description in UPnP does not play a role in the service discovery process.

These systems have made progress in various aspects of pervasive computing but are weak in supporting knowledge sharing and reasoning. A significant source of this weakness is their lack a common ontology with explicit semantic representation.

A key requirement for realizing pervasive computing systems is to give computer systems the ability to understand their situational conditions. To achieve this, it requires contextual information to be represented in ways that are adequate for machine processing and reasoning. At the same time resources and services needs to be described ad managed following a common semantics that should, also, be reflected at coordination level. The goal is to abstract the description of the application behavior from the management of the different instances that compose the environment.

Semantic Web languages are well suited for this purpose for the following reasons:

- Ontologies provide a means for independently developed systems to share resources, services and context knowledge,
- RDF and OWL are knowledge representation languages with rich expressive power that are adequate for modeling various types of contextual information, e.g., information associated with people, events, devices, places, time, and space.
- These knowledge representation languages are well suited to, also, describe resources and services.
- Because ontologies have explicit representations of semantics, they can be used by the available logic inference engines. Systems with the ability to reason about context can detect and resolve inconsistent knowledge.
- The Semantic Web languages can be used as meta-languages to define other special purpose languages such as communication languages for knowledge sharing, policy languages for privacy and security [KFJ03]. A key advantage of this approach is better interoperability. Tools for languages that share a common root of constructs can better interoperate than tools for languages that have diverse roots of constructs.

A common agreed standard ontology could be the “panacea” for most of the interoperability and openness issues raised in distributed systems and pervasive computing. But this is only an ideal scenario, quite difficult to realize given the current situation where lot of systems have developed their own communication protocols, description scheme and ontologies. Interoperability at such level is a very challenging and still open issue. Most of the approaches imply human intervention to solve, for example, ontologies mapping.

Despite the difficulties that the use of ontologies rise when considering interoperability between systems, it is a very efficient and interesting approach for describing a system as an isolate entity. All resources, services, contextual information as well as interaction and dependencies, when described with an agreed common semantic, could be better managed by the infrastructure resulting in a more clear separation between the application pure functional level and its specific system instantiation. This has been the approach that has inspired the UBIDEV model.

Development and Deployment

The number and diversity of software components that will be required in pervasive computing environments will necessitate methods for their rapid development and deployment.

Rapid development will be, in part, enabled by a feature rich infrastructure that lightens the need for application developers to be concerned with tasks such as adaptation, context gathering and management, resource discovery, distribution management and communication between distributed application components. Rapid development of specialized applications can be enhanced further by the development of special-purpose languages that enable applications to be specified at a very high level of abstraction. Infrastructural support for rapid application deployment can be achieved through the provision of execution environments into which applications can be placed without regard for configuration or adaptation. Rapid deployment of applications in distributed environments is already supported in a limited fashion by platforms such as the JVM, which can dynamically load and execute programs. However, these platforms do not yet meet the needs of heterogeneous environments, which require support for a broad range of component types, scalability, and dynamic configuration and adaptation of components.

1.6 Context

Invisibility of applications is accomplished by reducing input from users and replacing it with knowledge of context. The first definition of context-aware applications given by Schilit and Theimer [ST94]) restricted the definition from applications that are simply informed about context to applications that adapt themselves to context. Context-aware has become somewhat synonymous with other terms: adaptive [Bro96], reactive [CTB+95], responsive [EHC+93], situated [HNBR97], context-sensitive [RAH98] and environment-directed [FKS97].

“A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user task” [Dey00].

Context-aware software components exploit information such as the activities in which the user is engaged, proximity to other devices and services, location, time of day and weather conditions. Knowledge of context also is required to enable adaptation to changing environmental conditions, such as changing bandwidth and input and output devices, which can be brought about by mobility.

Adaptation requires a mobile client to sense changes in its environment, make inferences about the cause of these changes, and then react appropriately. These imply, for example, the ability to make global estimates based on local observations. To detect such changes, the client must rely on local observations; for example, it can measure quantities such as local signal strength, packet rate, average round-trip times, and dispersion in round-trip times.

The infrastructure for pervasive computing should support context awareness by facilitating the gathering of information from sources such as sensors and resource monitors; performing interpretation of data; carrying out dissemination of contextual information to interested parties in a scalable and timely fashion; and providing models for programming context-aware applications. A very challenging aspect is interpretation, which involves steps such as integration of data from different sources (for example, combining height and horizontal position into a three dimensional position); inference (for example, ”Bob is in the meeting room and Alice is in the meeting room, therefore a meeting between Bob and Alice is taking place"); prediction based on context history; resolution of inconsistencies between context data from different sources; and provision of estimates of the accuracy of contextual information.

The challenges associated with constructing context-aware applications for pervasive computing system and the importance of appropriate abstractions for gathering and reasoning about the context information has led an ontology-based shift in the research focus of the context awareness community. Ontology-based approach may improve over classical context modeling approaches by providing improved support for interoperation and sophisticated type of reasoning.

Number of interesting frameworks are investigating pervasive computing research, such as ContextToolkit [AKDS01], Cooltown [KBM+02], Intelligent Room [Coe98], OneWorld [GABW00], EventHeap [Joh03]. These systems use ad hoc representations of context knowledge, while others [RHC+02] similarly to UbiDev, explores the use of ontology to represent context knowledge.

Context Models

Many context models have been developed to support context-aware and adaptive systems and applications. These primarily address challenges as context representation, interpretation and dissemination.

The Sentient Computing project, for example, is concerned with supplying context information to applications, with particular focus on location information [ACH+01]. The location of mobile objects, such as people and equipment, is tracked by devices known as Bats (a successor to the earlier active badges), which communicate with base stations by ultrasound. Other context information is gathered by resource monitors, which track resources such as CPU, memory and bandwidth. Context is associated with a logical model of the physical world. In this model, real world entities are captured as objects that have types, names, capabilities and properties, including static and dynamic context. Additionally, applications can receive notifications of location-related events from a spatial monitoring service, which performs interpretation of location data and detects important events defined by containment rules. One of the drawbacks of the Sentient Computing framework is its focus on location. While the framework provides interpretation and event notification of location changes, its support for other types of context is limited to the ability to query the information via the proxy server. In order to support the rich context requirements of pervasive computing software, the means to apply context interpretation to arbitrary types of context are required.

Hewlett-Packard's Cooltown [KBM+02] project proposes a Web-based model of context. In this model, entities (people, places and things) have Web representations that can be retrieved using a URL. An entity's Web representation captures both static and dynamic aspects of context, including relationships with other entities and sets of services associated with the entity. One of the primary aims of the model is to enable adaptation of Web content according to user context. However, the potential uses of the framework are much broader. Location awareness is based around the concept of a space. Beacons wirelessly transmit URLs corresponding to spaces, enabling devices near the beacons to discover and access their local spaces. Spaces are accessed through portals, which are responsible for providing access control and a gateway to the space's services. A space manager performs tracking of the devices located within the space at any point in time and generation of dynamic Web

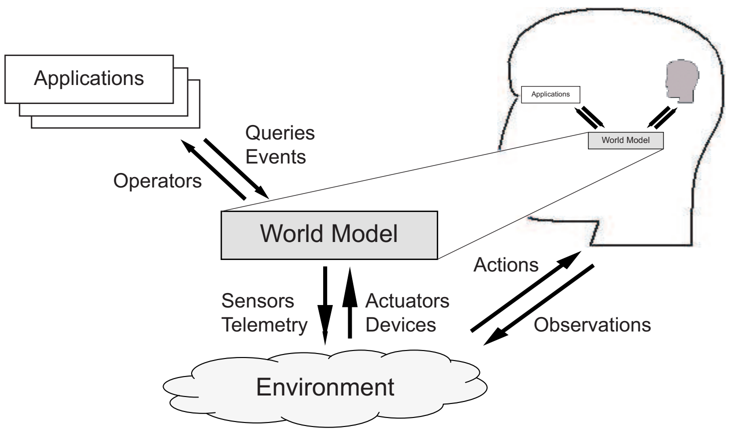

Figure 1.2: **Sentient Computing.** *Using sensors and resource status data to maintain a model of the world which is shared between users and applications.*

pages that reflect the current context. The Cooltown context model has several limitations. First, it does not address the means of specifying context, but instead allows arbitrary Web descriptions, which renders machine processing of context difficult. Additionally, interpretation of context and subscription to context events are outside the scope of the model.

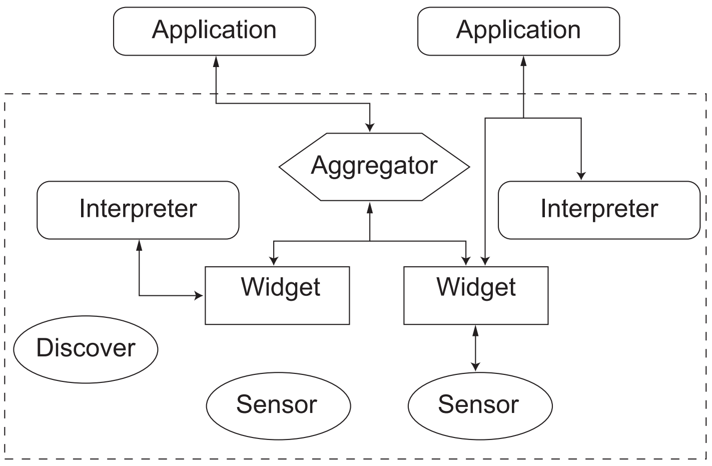

Figure 1.3: **Context Toolkit.** *An interaction example between applications and Widgets and between Interpreters and Aggregators.*

Unlike the Sentient Computing and Cooltown, the Context Toolkit project [Dey00] focuses on programming with context rather than context representation. The Context Toolkit has the aim of providing abstractions for separating the gathering and processing of context from the use of context. The toolkit comprises three types of component: context widgets, which acquire context data from sensors; interpreters, which perform processing of context data, such as abstracting high-level information about a person's location from raw location coordinates; and aggregators, which combine context data from multiple sources. None of the work carried out on context to date is adequate to satisfy the requirements of pervasive computing. The ideas of context modeling found in the Sentient Computing and Cooltown approaches, and those of context processing found in the Context Toolkit must be united into a scalable framework, and better programming models for context-aware applications, which support rich types of context-awareness and adaptation, must be created.

1.7 Coordination

Coordination languages, models, and systems aim finding solutions to the problem of managing the interaction among concurrent programs. Coordination has been defined as the study of the dynamic topologies of interactions among Interaction Machines, and the construction of protocols to realize such topologies that ensure wellbehaveness [Weg95].

Analogous to the way in which topology abstracts away the metric details of geometry and focuses on the invariant properties of shapes, coordination abstracts the details of interaction between processes, and focuses on the invariant properties of programs [Weg98]. As such, coordination focuses on program patterns that specifically deal with interaction.

Coordination models and languages are meant to close the conceptual gap between the cooperation model of an application and the low-level communication model used in its implementation.

As we move towards distributed, pervasive environments, the independence (mutual ignorance), partiality and unreliability of individual components is becoming a standard scenario. Modern network-based software is moving towards another model, in which communication connections are virtual and dynamic, rather than explicitly represented in configuration files, routing tables, and the like. Rather than creating explicit communication paths between individual components, each component can post information to a shared server (blackboard), and can subscribe to receive information that has been posted that matches a chosen pattern.

Anyway, the inability to deal with the coordination model in an explicit form increases the difficulty of developing pervasive computing applications that contain large numbers of autonomous interacting entities with nontrivial cooperation protocols.

In spite of the fact that the implementation of a complex protocol is often the most difficult and error prone part of an application development effort, the end result is typically not recognized as a commodity in its own right, because the protocol is only implicit in the behavior of the rest of the concurrent software. This makes maintenance and modification of the cooperation protocols of concurrent applications much more difficult than necessary, and their reuse next to impossible.

A number of software platforms and libraries are presently popular for easing the development of concurrent applications. Such systems, e.g., PVM, MPI, CORBA, etc., are sometimes called middleware. Coordination languages can be thought of as the linguistic counterpart of these platforms which offer middleware support for software composition. One of the best known coordination languages is Linda, which is based on the notion of a shared tuple space.

Tuple Space

The tuple space is a centrally managed space which contains all pieces of information that processes want to communicate. It is a well-established asynchronous communication model that is effectively a shared distributed memory spread across all participating hosts. It is the core abstraction of all Linda-like coordination models. A tuple is an ordered collection of values.

The tuple

$$(\mathbf{1}, '\mathbf{Bob}')$$

is a tuple of size 2, with an integer as the first element, and a string as the second. A template specifies what sort of tuple to retrieve.

A tuplespace is a virtually shared associative memory structure in that it is accessible to all processes, no matter what their physical location. It also makes no formal requirement of the actual distributive nature of the implementation.

All tuples in a tuplespace may be stored on a single node of the network, or they may be spread out over several nodes. Providing the tuplespace can be accessed from any node on the network it doesn't matter where the tuples are stored. A tuplespace is an associative structure because tuples are accessed based on their value and structure, not by address.

As described in [Gel85] tuplespaces are a generative communication method because when a tuple is added to a tuplespace by a process, A, then even if that process A dies the tuple remains in the tuplespace until it is requested by another process. Figure 1.4 shows the tuple space in action.

Tuple spaces provide temporal and spatial decoupling: hosts communicate through the space without being online at the same time or attached by an explicit binding, an ideal approach for mobile computing. Example tuple space implementations designed specifically for pervasive computing are described in hereafter.

- **Linda** [CG89] is a process coordination language where multiple processes interact by asynchronously entering and removing tokens from a single, globally shared tuple space. In the Linda programming model there are four operations: **out** for tuple creation, **in** for destructive tuple retrieval, **rd** for non-destructive retrieval and **eval** which is similar to out except that a new process is created to evaluate the arguments. Both kinds of retrieval select tuples via pattern matching, making it possible to operate on multiple tuples. Communication in Linda is decoupled both in time and space as sender and receiver processes do not need to be available at the same time or have knowledge of each other's location. This makes Linda highly appropriate for mobile platforms, where connectivity patterns typically vary.

  However, in an environment where communication links are unreliable and costly all operations being performed on a central space are infeasible.

- **Limbo** [DFWB98] is based upon the Linda tuple model architecture, but adds extensions for operation within a mobile environment. Limbo allows multiple tuples spaces to be created across hosts; tuples propagate between spaces using a bridging agent. Alternatively, an individual tuple space can span multiple hosts. The consistency between multiple tuple copies is maintained by a distributed tuple space protocol (that is implemented as a multicast group). The replication of tuple spaces across multiple hosts enables the tuple space to remain accessible during disconnection. Furthermore, QoS attributes can be added to tuples, including delivery deadline, so that mobile multimedia applications can be supported; this also allows the system to adapt itself to make best use of network connectivity. System agents monitor QoS and the propagation of tuples between tuple spaces. The monitoring agents watch characteristics such as connectivity, communication cost and power and inject tuples (representing

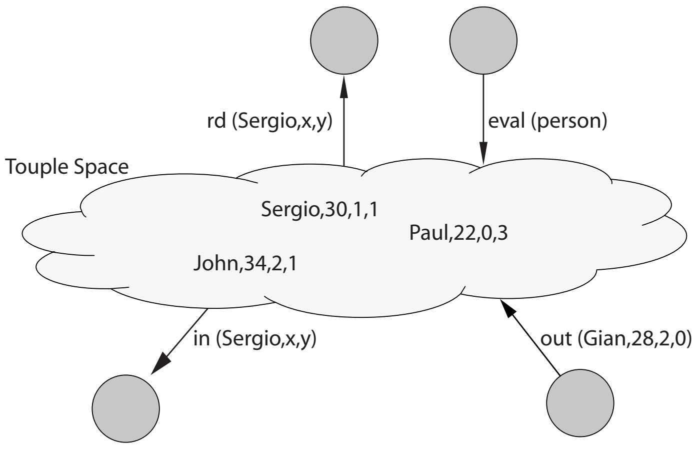

Figure 1.4: **Tuple Space abstraction.** *Tasks can generate asynchronous requests to read, take, write and eval tuples.*

the current system state) into a management tuple space. These are globally accessible, allowing remote hosts to query current QoS conditions. Filtering agents, a special type of bridging agent, then allow Limbo to adapt its behavior by performing transformations on the tuples being distributed. For example, a filtering agent can act between two spaces dealing with MPEG video frames and only transmit I-frames, or by performing color reduction on the I-frames if it detects a drop in bandwidth. Limbo's implementation of the tuple space paradigm means that disconnection, address migration and low bandwidth problems are solved. Furthermore, its use of monitoring and adaptation agents help address variable network conditions, and could feasibly be used to manage power consumption.

- **Linda In a Mobile Environment** (LIME) [MPR01] utilizes the concepts of the Linda coordination model and provides additional support to new types of distributed mobile applications. The underlying core is based upon a global virtual data structure (a tuple space whose content depends upon the connectivity of mobile hosts). This dynamically changing global context is accomplished by breaking up the Linda tuple space into many tuple spaces, each permanently associated to a mobile unit, and by introducing rules for transient sharing of the individual tuple spaces based on connectivity. The only way to access the global context from a mobile host is through an Interface Tuple Space (ITS); this contains the tuples that the host is willing to make available to other mobile units [PMR99]. Upon arrival of a new mobile unit, the tuples in its ITS are merged with those already within the global context and the result is accessible via the ITS. This abstraction provides the mobile application with the perception of a local tuple space contained within the federated space. LIME also allows the ability to react to changes in context, an important factor in mobile application design (for example, when a new member node arrives). The Linda model is extended by the Reaction concept. A reaction R(s, p) is defined by code fragment s, which specifies the behavior when a tuple matching the pattern p is detected within the tuple space. Like Limbo, the tuple space implementation is particularly suited to addressing problems of terminal mobility, and the reaction concept can be utilized to improve operation in varying network conditions and resource consumption.

Tuples spaces present powerful communication abstractions with simple programming interfaces. However, the application of the paradigm to real world systems is a complex task, requiring a thorough understanding of the process, and hence tuple spaces are not particularly well used compared to other paradigms. However, there is evidence that applications inherently matched to tuple spaces are simple to develop [MPR01]. Furthermore, commercial tuple space implementations (designed for the fixed network) including JavaSpaces [Wal98] and IBM's T-Spaces [WMLF98] [LMW99] are gaining prominence in the Java community.

1.8 Application

Applications for pervasive computing system are seen, from the infrastructure viewpoint, as the collection and the coordination of services provided under the umbrella of the user interfaces. Fundamental is to highlights the user-centered nature of pervasive computing applications. Concerns such as the user's intent, the user's context, the user's preferences, the user's abilities, the user's emotions, and the user's privacy are considered primary.

The user should be able to access any application or data using any device, within the limits of the capabilities of that device. Applications are considered primary, not the device from which the application is accessed. This calls for a separation between the logical data (user space) and the physical resources involved.

Applications need to be autonomous and invisible, by placing greater reliance on knowledge of context and reducing interactions with users. Moreover, applications must cope with highly dynamic environments in which resources, such as network connectivity and software services, frequently vary over time.

A programmer who built an interactive application needed to know about the specific devices and the details of their data structures and signals, in order to write code that used them appropriately. The code could be carefully tailored to the specific devices, to gain maximal efficiency and/or to take advantage of their special characteristics.

This arrangement works, but has some obvious shortcomings:

- Each new program had to have code to deal with the specifics of the devices.
- Each new device (or modification to an existing device) could require reprogramming of pre-existing applications.
- If a computer supports multiple processes, conflicts could arise when two processes communicate with the same device.

Indirection is one approach for decoupling programs from device interaction details. For example, the operating system provides device drivers, which are coded to deal with the specifics of the signals to and from the device, and which provide a higher level interface to programmers. Drivers can unify abstractions for different devices (for example, different physical pointing devices can provide the same form of two-dimensional coordinate information), or can provide multiple abstraction levels for a single physical device. Application programs can use libraries with program interfaces that provide higher level events and descriptions, while accessing lower level drivers provided by the operating system.

Another example is the distinction between software components and user interfaces. While software components are programming units that are dynamically composed to form complete applications, user interfaces are conceptual entities that are responsible for interaction with the user, and which may be distributed over multiple software components and devices.

An application is structured into two parts: interaction logic and application logic [SVSF05] [PPL^{+}03]. Interaction logic defines the interchange between a user and the system, regardless of the "context" of the user, including the device being used and the user's particular preferences. The application logic is organized into reusable software modules called services, which encapsulate a packaging of computation and data, perhaps to expose the information interface of physical artifacts.

Even for simpler objects, we are beginning to see a separation between the devices as viewed by a user and those designed into the computer system. For example, "tangible user interfaces" [FIB95] incorporate passive or semi-passive physical objects into computer systems as though they were virtual devices. Sensors such as cameras are used by programs that track these objects and model their behavior, and then provide a higher level interface to them.

A common approach is to separates three distinct conceptual elements:

- **Devices**: (sensors and actuators) and the signals they accept and produce
- **Phenomena**: a space of things and happenings relevant to a program
- **Observers**, which produce a particular interpretation of the phenomena using information from devices.

Application Level Adaptation

User interfaces for pervasive computing environments must be highly adaptable in order to respond to changes in the available input and output devices caused by mobility, to other changes in the context in which the application is used (for example, when the user switches from working at a desk to driving a car) and to novel application behaviors that are created in a dynamic fashion from available components. One of the challenges of providing user interface adaptation lies in ensuring that adaptation preserves a consistent view of an application. The user should have a uniform mental model of an application regardless of whether the user is interacting with a speech-based interface or a graphical one. Consistent adaptation is particularly challenging when the interaction paradigms are completely different. For example, interactions in gesture-based interfaces [GTV05] occur over a continuous period, whereas interactions in a mouse-driven interface occur at discrete points in time [KS03], which makes it difficult to map between the two forms of input. Another challenge lies in dynamically coordinating the use of heterogeneous collections of input and output devices to form a single user interface (for example, when a control interface for a video conferencing system is formed using a user's PDA as an input device and the videoconference screen as the output device).

User interfaces for pervasive computing must be carefully designed with several factors in mind. First, the ergonomics of the interface must be designed to keep the user's attention focused on the task at hand rather than on peripheral matters; that is, the interface should not be distracting [Nor99]. Second, the user interface should be rewarding and enjoyable to use. Third, the user interface must allow novel types of interaction that will become more common as computing tasks become increasingly ubiquitous, such as delegation of tasks and provision of guidance to software agents. Finally, user interfaces should be designed for ordinary people, rather than just for technologists.

User interfaces

Users in pervasive computing environments demand ubiquitous access to their computing applications, which creates a requirement for universally available user interfaces. Device heterogeneity introduces a further requirement for user interfaces that are highly adaptable. Moreover, the diminishing amount of user interaction with applications (brought about in part by the increasing ratio of applications to people) and the changing nature of the interactions (brought about by computing becoming situated in mobile and other novel situations) call for new types of user interfaces.

The need for universally available user interfaces creates a requirement for new methods of programming user interfaces that do not make assumptions about the available input and output devices. GUI interfaces that are designed for use with a screen, pointing device and keyboard will no longer be broadly useful in future computing environments, as devices with novel input and output mechanisms (for example, touch screens and gesture recognition) will become increasingly common. In order to provide scalable support for universal interfaces, it will be necessary for application programmers to write generic interfaces that allow the semantics of user interaction to be specified without reference to rendering or input modalities.

The main challenges that have to be faced at user interface level are:

- **Relocable input and output**: the interface should be able to steer information around an environment and serve as both input and output interface to computing. A far-field speech recognition system that allows speech based interaction with a moving display, or a moving projected keyboard that allows the user to type in commands would be examples of relocable interfaces that support both input and output.

- **Adaptation to user and environmental context**: as opposed to the typical fixed interfaces such as a monitor with a mouse and a keyboard, pervasive computing interfaces should free the user to move about and carry out other functions. For example, an interactive display should be oriented towards the user and be close enough to the user to facilitate interaction. An interface should also adapt to the characteristics of the environment in order to be useful. For example, a projected visual display should appear on an available surface in the environment that is of the appropriate size and orientation, has the appropriate color and texture to show the display with sufficient contrast, and is not occluded by other objects in the environment. Similarly an acoustic output should adapt to the presence of acoustic noise sources and objects blocking the acoustic signal. Thus, a steerable interface typically needs to be aware of, and reason about, the geometry and properties of the environment, and relate the user as well as input and output devices to a model of the environment.

- **Natural interaction**: interfaces should be natural and easy to use. The end result of ubiquitous access and adaptation to user and environmental context should be an interface that is intuitive and usable rather than something that distracts, confuses, or overwhelms the user [GSSS02].

Steerable interfaces like [PPL$^+$03] are examples of research results that go in such a directions.

1.9 Users

One of the basic shifts from today's systems to interactive workspaces is releasing the conventional assumption that each device is associated with a particular user who is logged onto it at a given moment [Car01]. Shared wall-based displays, for example, need to be usable by anyone at any moment, without going through a separate login or identification step [SGH98]. However, the sharing needs to respect individual information spaces. Users in pervasive computing environments can be mobile and have computing sessions distributed over a range of devices. The infrastructure's role with respect to users should be to maintain knowledge of their context and to manage tasks related to their mobility.

User Level Context

The infrastructure should maintain context data related to users, including their capabilities, preferences, current activities and active computing sessions. The uses of user-related context include allowing applications to provide adaptation to user requirements, and enabling the amount of input that applications require from users to be reduced. For example, with knowledge that a user is engaged in driving a car, an application can ensure interaction is carried out through a speech-based interface, rather than a screen based one, in order to enable the user to focus on the road. Information about users' computing sessions, including details about applications and the devices on which these applications reside, can be used to manage the application migration and adaptation that frequently must occur when a user is mobile.

User Level Mobility

User mobility between devices should be supported by enabling automated migration (or reinstantiation in a new location) of application components. The tasks of identifying the need for application migration and then carrying out the migration should ideally be performed by the computing infrastructure in a manner that makes the migration as transparent as possible to the applications concerned.

One of the key motivations for the generalizations in this architecture is the desire to support integrated applications with multiple users and multiple devices in an interaction structure that is many-to-many (one person may use several devices, several people may share one).

1.10 Summary

This chapter has presented the heterogeneity problem in pervasive computing focusing on the role that an infrastructure should take when providing support for it. In doing so, this chapter, has analyzed the state of the art of related technologies and approaches trying to underline their strength and what is still missing. The main problems have been categorized in terms of resources, services, context, coordination, application and user.

UBIDEV as an infrastructure for pervasive computing systems handles the heterogeneity problem from the following viewpoint:

1. **Resources:** the disparate and different nature of resources involved in pervasive computing environments make difficult for an infrastructure to provide a uniform way of describing, configuring and access them.

2. **Services:** in a Service-Oriented programming model, services need to abstract their heterogeneity at higher levels; they also need to express somehow their dependence and impact on the underlying physical resources.

3. **Context:** an infrastructure is supposed to provide a way of describing the model of the context as well as the contextual information contained. More than how to acquire such information the accent is on how to describe, structure and access it.

4. **Coordination:** it is required to define the basis for a coordination model built on top of Resources, Context and Service abstractions. An infrastructure should provide at application level a service-based coordination model where the behavior of the whole system is described in terms of homogeneous interacting entities.

5. **Semantics:** provide a way of describing the semantics of the whole system and ensuring that resources, services and contextual information are managed and coordinated accordingly. There is the need of providing mechanisms to allow an infrastructure to act upon a given environment in terms defined by an agreed ontology. In such a way the application is guaranteed to always have a coherent vision of the underlying environment.

Part II

UBIDEV

Chapter 2

The Model

This chapter presents the UBIDEV model, the key result of the research behind this dissertation. UBIDEV is an infrastructure that aims at supporting the development and execution of applications for pervasive computing system.

UBIDEV stands for **Ubiquitous Interacting Devices** because it is based on a model where an environment is characterized by the ubiquitous availability of services provided by abstract devices interacting with each other. As it will be presented during this chapter, users are considered to be part of such environment as devices (or resources) able to produce and perceive stimuli with their peer.

2.1 Introduction

We believe that extending the concepts from traditional distributed systems to pervasive computing system simplifies the management and the development of applications; but at the same time, we need to actualize these concepts to the new challenges that a generic pervasive computing system rises.

UBIDEV is a distributed middleware infrastructure that coordinates software entities, heterogeneous networked devices, users as well as physical entities contained in a physical space. UBIDEV exports services that could be composed together, it accesses and uses existing resources and context information to solve user tasks, and provides a framework to develop resource-aware, multi-device, and context-sensitive applications.

One of the main contributions of UBIDEV is not in the individual service it provides, but instead, in the interactions among these services. This interaction allows users and developers to abstract pervasive computing system as a single reactive and programmable entity instead of a collection of heterogeneous individual devices.

The first part of this research has been addressed to the analysis of a generic pervasive computing system. The peculiarity of these systems in terms of heterogeneity and dynamism forces applications to take into account both their physical and service dimension: building a pervasive computing system requires to analyze the computing infrastructure the environment should be equipped with, as well as how each single service should configure and adapt itself according to the topology of the resources composing the environment.

A middleware has been induced by the need of supporting applications concerned with resource and service management. While conventional middleware technology helps the development of distributed applications, it does not provide suitable support for dealing with the dynamic and heterogeneous nature of pervasive computing environments. Furthermore, the conventional middleware approach tends to hide the details of the underlying layers. On the contrary, pervasive computing applications can greatly benefit from knowing some relevant functional details of the computational environment.

Therefore what is required is a software infrastructure that defines a suitable coordination model to describe and manage the environment dynamism; that means to relieve the application from directly handling the heterogeneity of the underlying environment as well as from maintaining a model of the context. At the same time such a middleware should offer a level of fine-grained control over the environment to the applications that may need it. As part of this research we developed an adaptable and reconfigurable middleware prototype [MH02]. This approach as been influenced by the research of Georgia Tech [Abo99] [DAW98], Carnegie Mellon University [SG02], and Illinois [RHC$^+$02] on pervasive computing. This middleware allows bidirectional interaction between the physical entities and the application level.

UBIDEV is a generic model of a service framework with the ability to adapt the behavior of the whole system according to the changes that occur in the environments, for example, resource availability or user mobility. It is organized in a form of middleware that is compliant with the specifications of a Service Oriented Architecture: "problems can be modeled in terms of the services that an object provides or uses"[Bie02]. An application is then described in terms of composition of homogeneous autonomous services within a context structure. Semantics interoperability between each module is granted by an application ontology that emphasizes the application functional level that is considered to be invariant with respect to the dynamism of the environment. That ensures that the whole system is represented according to the perspective of the application viewpoint and not merely in terms of resources and services interaction.

The rest of this chapter presents the UBIDEV reference model and its building blocks. The next section introduces the relation that binds UBIDEV with the physical environment and how this relation determines the nature of a pervasive computing system. Then the reference model and the basic abstractions are presented. Then key notions of the model follow: the relation with the physical environment and the application ontology that represents the key element for granting the semantics interoperability level among all other elements. Resources, contexts, services and coordination are then presented with respect to the abstraction they represents and how they rely on the semantics stated by the application ontology. In the conclusion of the chapter we argument on why the proposed model answers the initial question.

2.2 A Bio-inspired approach

UBIDEV takes a bio-inspired approach in the relation with the physical environment: according to the theory proposed in [MV92], a living system is considered to have an organized structure able to maintain and regenerate its structure and its autonomy according to the variations of the environment (autopoiesis). In such a perspective UBIDEV completes a physical environment with the required functionalities and the interaction model that make a pervasive computing system an observing system based on perception, representation and action as depicted in figure 2.1.

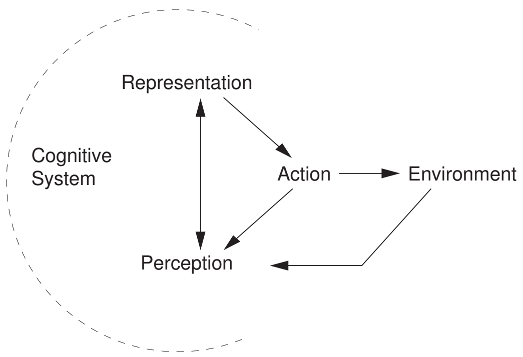

Figure 2.1: **A generic model of a cognitive system**. *The solid arrows indicate causal processes, and the thin lines indicate a coordination of acting, perceiving and representing.*

The interactions between each single autonomous entity and the environment occur in terms of reciprocal perturbations called stimuli. This kind of interaction describes also the relation between UBIDEV and the physical environment. In such a process, in fact, the structure of the environment declares and determines the structural changes that occur in UBIDEV and vice versa for the environment. The result is a mutual structural coupling (figure 2.2). In such a context the specificity of the physical space could influence the interaction mechanism. Moreover, as an autopoietic system, UBIDEV introduces a fundamental distinction between the organization (domain of control) and the structure (domain where entities exist) of the environment.

This framework allows modeling an environment in terms of resources, services that manipulate those resources, contexts as containers for resources and services and the composition of existing services according to contextual information. UBIDEV interacts with the environment on the base of stimuli that could be interpreted as either perceptual information from the physical space or explicit actions taken on the entities belonging to the environment. Those actions determine the nature and the sequence of the interactions that reflect the whole behavior of the system (figure 2.2). The resulting holistic management emerges from the building blocks UBIDEV defines: Resource, Context, Service, Coordination and the Application Ontology. This classification allows to separate the constraints that characterize all the pervasive computing systems and, at the same time, to define the basic structure of the model. As a result the design and the development of an application are driven by the functionalities supported by the system rather than by the configuration and by the use of the underlying technology infrastructure.

That can be summarized by the following motivating problem:

- *"How to define an abstraction of the physical environment that permits to describe applications in terms that are general enough to be not statically dependent of the specificity of a particular environment and, at the same time, to preserve the functional dependencies that the environment may have on the configuration and on the execution of an application."*

2.3 The Reference Model

UBIDEV is a lightweight infrastructure built around a reference model: Physical Entities, Re-

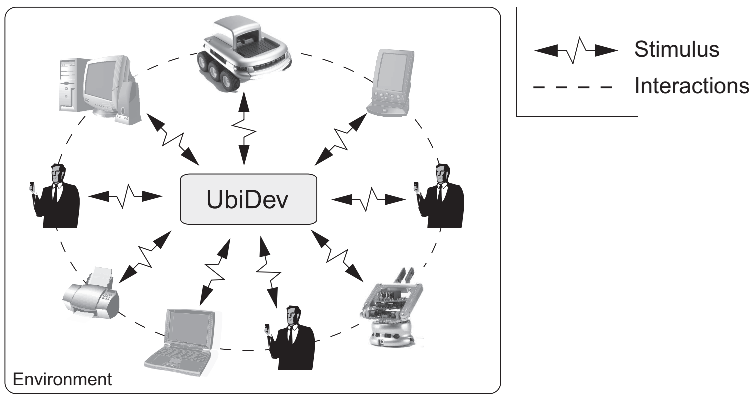

Figure 2.2: **UBIDEV perturbation of the environment**. *The environment is considered to be an autopoietic entity that, when perturbed, reacts and adapt its behavior accordingly.*

sources, Services, Context, Coordination and Application. Unlike most middleware as Jini that are constructed as monolithic black-boxes, UBIDEV is organized as a group of collaborating logical modules as depicted in figure 2.4. Layering cleanly separates abstraction from implementation and is thus consistent with sound software engineering. Layering is also encouraging standardization since it facilitates the creation of modular software components. Deciding how to decompose a complex system into layers or modules is nontrivial, and remains very much an art rather than a science. The two most widely-used guidelines for layering are the information hiding principle [Par72] and the end-to-end principle [SRC84].

All interactions between UBIDEV and the environment are described in terms of stimuli. Stimuli are classified into two main classes: perception stimuli as the contextual information perceived from the physical environment, and action stimuli as the explicit action that UBIDEV takes on each entity that result in the full interaction process that characterizes the behavior of the system. Such stimuli exist in the physical level of the system as they refer and rule the interactions between entities. At a more logical level, the application functional level, the perturbations of the environment are represented in term of input and output signal. The application, in fact, through UBIDEV perceives signals from the environment and produces output signals that are mapped, still through UBIDEV, to new perturbations at the physical level (figure 2.3).

This model could be used to describe interaction patterns of both of classical computing nature and those more structured of a pervasive computing environment.

Consider a well known and simple interaction scenario where a user types on a keyboard, the signal is interpreted by an application and the result is displayed on a screen. The user, the keyboard and the display are resources part of the physical dimension of the environment. The interactions between resources are categorized as perturbations produced within the environment in terms of both perception and action stimuli (the user interaction with the keyboard is more a mechanical interaction, but that produces a perturbation interpreted by the OS as input stimuli for the active application. The interdependence between the resources, the application and the action stimuli produced is described in a coordination pattern resulting in displaying at the screen what the user has just typed on the keyboard.

In a more pervasive-oriented scenario, a user interacts with an application using an input device (that could still be the keyboard) and prints out his document on the nearest printer.

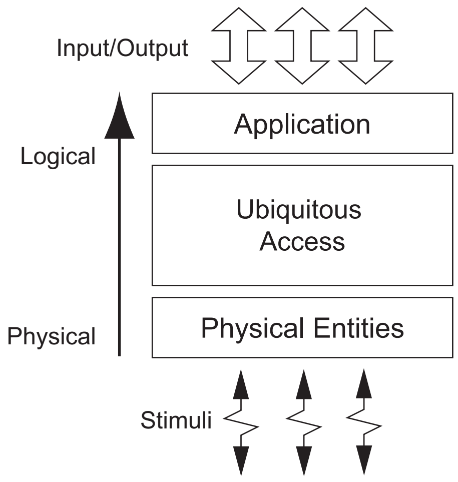

Figure 2.3: **UBIDEV stimuli model**. *Stimuli are classified in two main classes: 1) perception stimuli as the perturbation produced on the physical environment that could be perceived by the system. Input/Output stimuli are considered to be a subclass of perception stimuli. 2) Action stimuli as the explicit action that* UBIDEV *takes on each entity that result in the full interaction process that characterize the behavior of the system.*

In such a case, the scenario is enriched with contextual information like the position of the user, his proximity with other resources and, optionally, the user identity and access rights. Using the UBIDEV model to describe such scenario, the control unit within *Ubiquitous Access layer* allows the system to interpret the perturbation produced by the user as well as those perceived by some contextual information sensors. All information is interpreted by a coordination unit together with the application-interpreted input stimuli. The result is an action stimuli perturbing the user's nearest (and available) printer.

UBIDEV can then be considered as a unified management model for resources, services and context information. It proposes a context centric management of the environment: it is the application context that determines the semantics of the resources, of the services involved and of the contextual information. Consequently resource configuration, service instantiation, description and composition, context model as well as user task solving are based on this semantics. This allows pervasive computing applications to automatically reconfigure themselves according to context changes. As a result UBIDEV presents at application level a homogeneous coordination space seen as an unified mechanism for dynamic interaction of services [CA94] [AC93] as depicted in figure 2.4.

A key element in realizing this architecture is the use of an application ontology that undergrids the communication between entities and the representation of the environment. A generic knowledge representation system uses various terms with different domain specific definitions in order to describe the knowledge model. Instead of introducing its own semantics, UBIDEV identifies the internal representation of semantics and the relation to the environment as relations to context and resources. That leads to small topic-oriented ontology used to classify the whole environment in terms of resources, services and contextual information. An application can be described according to the conceptual model of the ontology that is independent of the specificity of the underlying environment. Once an application is instantiated in a specific environment, UBIDEV can ensure that the resource, the contextual information and the services are described and managed according to this model, thus shielding the application from directly dealing with it.

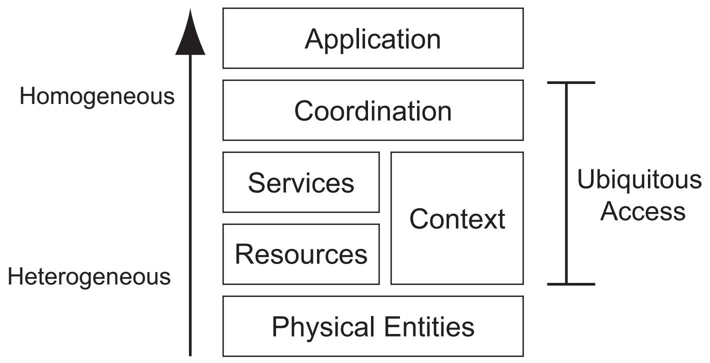

Figure 2.4: **UBIDEV reference model**. UbiDev *separates the coordination aspects from the resource and service management in order to hide at the application level the heterogeneity of the underlying environment.*

The *Physical Entities layer* represents all the entities belonging to the environment. This layer characterizes the physical dimension; is the ground level of description of a pervasive computing system. Each physical entity that has to be involved in the interaction process has to be modeled and represented inside UbiDev. That allows services to rely on such entities in exploiting their functionalities. In UbiDev the user is considered as an ultimate resource involved in the interaction process. As such he belongs to the lowest level of abstraction, in direct contact with the devices.

The *Ubiquitous Access layer* represents the core of UbiDev; it centralizes all functional aspects related to resource and service management. It is also responsible of creating and maintaining a model of the context for the application. It is composed of four modules according to the abstractions is supposed to provide:

- The *Resources layer.* Resources are considered to be all the entities that may be involved in the execution process of an application; they describe physical devices, software components and users. Due to the 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 *Services layer.* It is responsible of analyzing the relation between the service description and the physical devices. In exploiting such operation it classifies the system resources and, according with the service description, determines the suitable execution environment candidate to host the service; then it encapsulates the service and its execution environment into a self-contained homogeneous entity. That allows a high degree of flexibility in both service description and resource look-up and composition.

- The *Context layer.* It takes in charge the process of gathering and representing the context; it relies on dedicated services that sense the information from the physical environment. This information is then classified into context data type according to the application ontology. Context is organized as a container for resources and services;

it defines a namespace and the resource access. In order to extend or restrict the perception of the environment, UBIDEV allows some operations on contexts. A new, extended context can be constructed by the union of contexts and restricted by building a sub-context. Context determines also the role and the relation that resources may have with respect to the application; hence subcontext may be used to group resources that belong to the same set of roles. An example may be the subcontext ”meeting room” inside the context ”building”.

- The *Coordination layer.* It defines coordination space and interaction rules allowing applications to coordinate services as in any classical service-oriented model [Wal98], [Jav03]. The main difference from the classical coordination models is that through this layer an application can define behavioral laws that are attached to the context model and are applied to the resources involved in the interaction process. Its main goal is to solve complex queries coming from the *Application Layer* in terms of composition of basic services. Composition is the process of building complex composite services from primitive ones; thus enabling rapid and flexible creation of new services. In doing so, contextual knowledge and behavioral laws are analyzed in order to adapt the service composition policy, and hence the application behavior, to meet the system constrains.

The *Application layer* is a generalization of the view in a traditional Model Viewer Controller [KP88]. It works as a presentation module embodying some application dependent functionalities like the visualization producing the user tasks. Tasks are represented as a transformation from an input resource to a final output resource; this transformation is interpreted by the coordination module in terms of composition of abstract services. It provides tools for specifying the interaction logic, that will be more relevant at coordination level, and the application logic according to the shipped ontology. Application GUI modules, together with the resource specific access modules, are the only pieces of code that are executed on the specific resource according with the application requirements. UBIDEV does not define any particular client-side module; in the Ubiquitous Message System prototype, the Palm interface is realized by a simplified implementation of a remote framebuffer protocol [Ric03].

2.4 Physical Entities

The physical environment is perceived as an interaction space where different entities exists each time given. An application is then realized by the interaction among entities being them software modules, hardware devices, physical objects or humans. In such a perspective a pervasive computing system can be associated to the paradigm of an autopoietic system [MV92]. Fundamental characteristic of such a system is its autonomy and its operational closure. That means that the system is in a state of fully auto-reference, in which all actions taken are aimed to preserve its integrity with respect to the external and internal perturbations. This concept does not imply isolation of the system but it is more related to auto-behaviorism in which the actions of a complex system, realized by interconnected elements, have effect inside the system itself and inside the system dynamic.

These environments are considered to be physical spaces where different entities interact with each other for a common goal. Those entities vary from ordinary objects like tables and walls, to computing devices like sensors or computers, up to animated entities like people, animals and robots. All the entities that may have any relation with the structure and the behavior of the system, has to be included in the model of the physical environment. In such a way a pervasive computing application, seen as the functional part of a pervasive computing system, is realized by inspection and complex interactions of the entities composing the system. UBIDEV represents the software infrastructure that aims to govern and model these interactions. Those interactions are structured and governed on the base of stimuli that entities exchange with the infrastructure. The stimuli from the environment are the common denominators in the interaction process.

2.5 Application

A pervasive computing system is composed of a physical environment, of a software infrastructure that supplies services for configuring the environment and of an application that specifies the system functionalities. The Application layer embodies the description of the system functionalities expressed in terms of interfaces defined to generate the input queries, ontology to state the application conceptual model and the model of the context.

Interfaces

UBIDEV does not define a specific pre-defined GUI module for applications; nor does it define any client-side module that has to be loaded on the target resource. It only defines a presentation module that is used as a viewer in a classical MCV model [KP88]. Application developers have the freedom to design and code any GUI, assuming that they interface with UBIDEV through the presentation module. Due to the structure of the coordination space, the user input tasks are expressed in terms of requests to transform an input resource to an output resource in a specific context space.

Ontology

When two autonomous entities exchange messages they must have common interfaces and protocols, including a common message format. In addition, the parties must know or discover the semantics of the messages: the vocabulary of the messages, which includes the names and valid values of message elements. Essentially, the parties must have a shared schema for interpreting the messages. Semantic interoperability is the establishment of shared schema for interacting entities. To ensure that independent and autonomous entities can understand the semantics of the environment and of the other entities when interacting with each other, UBIDEV uses semantics web technologies [CFJ03] to attach semantics to various concepts in pervasive computing environments. The ontology allows different entities to have a common understanding of various terms and concepts and ease their interaction process. The ontology also defines the different kind of operation an entity may support to interact with one another. The goal is to use ontologies to support knowledge representation and communication interoperability in building pervasive computing systems.

Every UBIDEV application defines its own ontology specific for the pervasive computing system it aims to address. That leads to small ontologies that better tackle the specificity of the environment and allow applications to describe their conceptual model in functional terms more focused on the specific problem they aim to address.

The application conceptual model is represented as an Entity Relationship model [Che76]: it is defined as a collection of entities and of relationship among entities. An entity is an object that exists and is distinguishable from other objects. An entity set is a set of entities of the same type that share the same properties. A relationship is an association among several entities. It can take the mathematical form of a relation among $n \geq 2$ entities, each taken from entity sets

$$\{(e_1, e_2, ..., e_n,) \mid e_1 \in E_1, e_2 \in E_2, ..., e_n \in E_n\}$$

where $(e_1, e_2, ..., e_n)$ is a relationship.

Entities, that exists in the physical environment, assume a precise semantics when they are classified into different entity sets. There is a predicate associated with each entity set to test whether an entity belongs to it. An entity is represented by a set of attributes common to all members of the entity set it belongs. The interaction among different entities can, then, be structured according to the expected properties that the entity set guarantee.

The application conceptual model can be interpreted also from a linguistic viewpoint as a symbolic system [Har90]; a functional level of its own, with rule-full regularities that are independent of their specific physical realization. The meaning of the symbols comes from connecting the symbols to the world in the right way. Once one has the grounded set of elementary symbols provided by taxonomy of names (and the iconic representation that gives content to the names and allow them to pick up the objects they identify), the rest of the symbols can be generated by symbol composition alone, and they will inherit the intrinsic grounding of the elementary set. These elementary symbols called icons represent the sensor projections of the perceived physical entities.

Classification is the operation of identifying the elementary symbols of the model by testing whether an entity belongs to a specific entity set or not.

This approach that has already been considered in the COCA model and previously in the EMUDS [RCC98] project, is an attempt to face the symbol grounding problem [Har90] by defining instruments for identifing the iconic representation of the real world in a specific environment.

Context model

In order to extend or restrict the perception of the environment, UBIDEV allows an application to define a structural model of the context. It is a descriptive model that defines the structure of the physical environment in terms of logical spaces. The context module takes this description to create an organizational structure that will be used by coordination module to store resources and services and to determine the role of each entity. That allows application to restrict resource spaces to have only resources exhibiting certain properties within a scope. Users could be described within the system with extended properties such as roles, identities and intentions, giving the possibility to the system to take such information into account when adapting individual services to the requests perceived within a given context and by a given user.

2.6 Resources

Resources are considered to be all the entities in an environment that could be manipulated by an application. Hence a resource may range from ordinary desktop computers, trough mobile phones, measurement systems, robots and users. In UBIDEV the user is considered as an ultimate resource involved in the interaction process. As such he belongs to the lowest level of abstraction, in direct contact with the devices.

This classification is compliant with the one proposed in other projects [KBM+02]; UBIDEV just separates the notion of place from things and people because, as we will see later, places are captured in the context abstraction.

The main question the resource abstraction has been conceived to address is:

- *"How to identify and discriminate resources belonging to the environment allowing an application to access them according to their semantics?"*

In UBIDEV resources are considered to be atomic elements; hence the necessity to model the environment in terms of manipulation of resources. Resources are mapped in the model using an adapter that provides the resource independence abstraction incorporating the software modules that the rest of the system uses to accomplish I/O operations. Resources are then associated with semantic-rich tags coming from the application ontology. This process, called classification, describes the whole physical dimension in terms known by the application. Finally a dedicated protocol allows services to access resources addressing concepts of the same application ontology.

Adapters

An adapter is considered as a virtual representer of a resource inside the UBIDEV system; it defines basic standard access mechanisms to exchange data between the platform and the resource. The major issue is to make all resources look the same at least from the access viewpoint. Through the adapter is possible to define uniform access functions calls for accessing resources without having to deal with the details of the peculiar resource commands. All adapters have the same interface; in this way it becomes quite easy to plug in a new adapter, provided it conforms to the interface. An adapter contains the details of a particular resource, so each resource corresponds to a specific adapter. The approach is similar to Unix where all resources (physical) are mapped in terms of an entity called device (/dev) [Tan97]. This kind of abstraction allows to hide at upper layers the management of the specificity of each resource as the device driver approach permits a uniform access to different resources [Tan97]. Adapter defines simple yet complete interfaces:

- **open**: request to gain access to the resource and open a dedicated channel for communication. As it will be presented in next chapter, reservation is granted by a lock mechanism.

- **close**: when a resource go "out-of-reach" or spontaneously leaves the environment, the adapter is requested to garbage-collect the representation of the resource as well as its communication channel.

- **read**: request to read data from the resource. No type is specified for the data since the interface is meant to be valid for all resources. Typing or casting the data is up to the requester.
- **write**: request to write data to the resource. Similarly to read(), write takes raw data to be sent to the resource. Is up to the adapter to structure the data to suite the format of the resource it is associated to.

Adapter is the first step towards the definition of an abstraction of the underlying physical dimension. It is particularly suitable for integrating handheld devices because it embodies the specific resource access information.

Classification

The main problem related to the management of resources is the role that each resource may have in a given context. This is referred as the semantics and is reflected on all action that could be taken on a resource, typically description, discovery and access. UBIDEV relies on the COCA[Sch02] principle that it is the application context that should determine the meaning of resources. In doing so it relies on the ontology stated by the application and by classifiers. Classifiers [Sch02] are services that given a resource and an ontology, output concepts of that ontology. This mechanism replaces the classical scheme where resources supply their description directly.

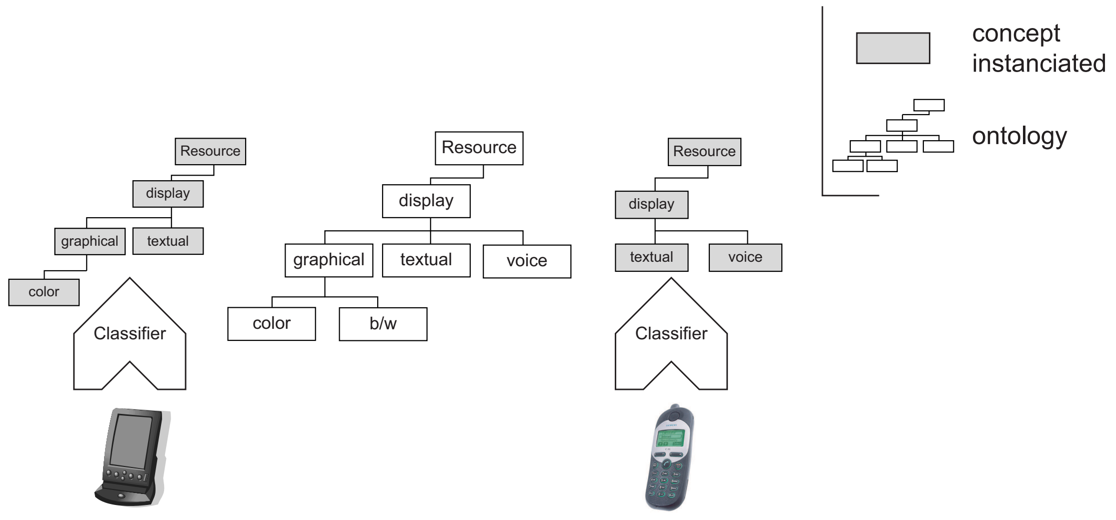

Figure 2.5: **UbiDev Classification Process**. *An example of classification of physical resources. Classifiers are seen as the knowledge required to identify the relation between the icons in the symbol system and the physical elements.*

A classifier accesses resources through adapters, associates one or more concepts it knows with a resource in a given context, and tags the resource as an instance of that concept. Classifications of resources are stored and used as a cache when an instance of a concept is requested by services. 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, or name. Thanks to the *classifier* UBIDEV decouples the high-level concepts (abstractions) from the instances implemented by a context. The concept ″nearest printer″ [Kam00] 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).

Figure 2.5 shows an example of classification for the Ubiquitous Message System prototype. According to the application ontology, a PDA and a portable phone are classified respectively as instances of concepts ″display″, ″textual″, ″graphical″ and ″color″ the former and ″display″, ″textual″ and ″voice″ the latter. A service that manipulates textual display will find these resources semantically equivalent; the service may access both of them transparently addressing the concept ″textual″. The related adapters will do the rest resulting for the PDA to receive the text as a new Memo note and, for the portable phone to receive the text as a SMS.

UBIDEV protocol

UBIDEV defines an access protocol to allow the access to resources through the ″addressing by concept″. Services manipulate resources through such a protocol that grants the resource location transparency. In such a way a service, for example, can access a printer resource by addressing the ″printer″ concept given that the printer has been classified as an instance of that concept. The UBIDEV protocol provides a dynamic configuration mechanism for services in a local area network. It is not a global resolution system for Internet resources; rather it is intended to serve local networks with shared resources. Services are modeled as clients that need to access resources in a specific dedicated environment.

The structure of the UBIDEV protocol is similar to HTTP thanks to its simple and yet clear syntax. The protocol scheme-specific information following the ″ubidev://″ URL scheme identifier provides information necessary to access the resource. The form of an ubidev: URL is as follows:

- *ubidev: URL = ″ubidev://″ <concept> ″?″ <request_type> ″:″ <param>″*

where <concept> and <param> are one of the terms of the application ontology. A request message from a client to a server includes, within the first line of that message, the <request_type> to be applied to the resource. The <request_type> token indicates the method to be performed on the resource identified by the concept. Allowed methods are:

- **SND**: requests that the enclosed entity (<param>) be stored under the supplied <concept>. This method allows creating a transfer stream between two resources. A typical example could be ″ubidev://display?SND:message″ that results in sending the content of the resource ″message″ to the resource ″display″.

- **RCV**: means retrieve whatever information (in the form of an entity) is identified by the <concept>. If the <concept> refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless the text is to be the output of the process.

This protocol allows two level of indirection for services accessing resources: firstly the protocol handler maps the concepts in the request to specific resource references and, secondly, it creates a bind with the corresponding adapters allowing services to transfer data transparently. The linear correspondence with the adapter interfaces and the consequent transaction mechanism justify the limited methods implemented in the protocol. In such a way a service could not even be aware of how the request is exploited on the resource, as long as it is guaranteed that the resource respect the expected semantics.

That implies a considerable level of abstraction because, for example a service may send data to a concept like ″display″ not being aware of how the display is reached and how the data is structured. In the Ubiquitous Message System prototype, presented in chapter 4, two instances of the same service address a ″textual″ display concept resulting, for one to send an email to an email client and, for the other to send an SMS to a portable phone.

UBIDEV Resource Abstraction in Pervasive Computing

UBIDEV proposes a model for resource management based on internal representation and classification according to the system ontology. Adapters, classification and access protocol are the basic abstraction for resources configuration, description and access:

- **Configuration:** every resource corresponds to a dedicated adapter that encapsulate the resource specific access information. Once the adapter exists and is correctly instantiated the resource has a virtual representer in the system.
- **Description:** resource description is implicitly embodied in the classifiers while the resource semantics is expressed by the application ontology. Services make a semantic selection of resources according to the classification. Each concept of the ontology represents a set of specific characteristics that resources are granted to supply.
- **Access:** the UBIDEV protocol as been defined as an instance of ″addressing by concept″. Services access resources by formulating requests in terms of concepts they know.

2.7 Context

A context-aware middleware allows applications to reason on contextual information gathered from the physical environment. Moreover, a holistic management of pervasive computing systems imposes the possibility to structure the environment according to a specific model of the context.

The context support in the UBIDEV model as been conceived to address the following question:

- *How to model and represent the physical environment and the related contextual information in order to allow applications to manipulate and infer semantic-rich knowledge?″*

Context represents an active container for resources, services and contextual information. According to this definition, the environment can be represented in terms of a hierarchical logical structure similar to a n-tier tree called context-tree. This approach is similar to the RAUM location model [BZD02]. This structure represents the organizational shape of the physical environment in terms of logical spaces interconnected with each other; the application can model the environment through this abstraction according to its model of the context. The context-tree is characterized by:

- **context-root:** represents the main container; is the most generic form of logical space. All other sub-context inherit from it.
- **context-node:** is a logical organizational context. It may contain different entities and it has only one parent. It is used to aggregate resources in groups that share similar characteristics or roles (i.e. all the resources that a user is manipulating).
- **context-leaves:** leaves are directly associated to resources. Context-leaves are used to identify resources and to deduce their role inside the context structure.
- **entity ubiquity:** each node can dynamically change its parent. This functionality has been introduces to tackle resource mobility. This principle applies to both nodes and leaves. An entity (both leaves and nodes) can be tied to several contexts. However, in our model we take for granted the Pauli principle; that means that an entity can be active at the most in one context at a time (it can have only one direct parent). Apart from this environment, is can be at most ”virtually” or ”sensorial” present in the other environments.

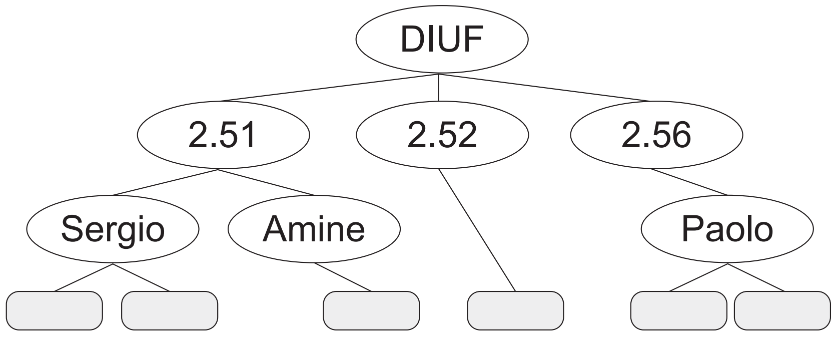

Figure 2.6: **Context-Tree.** *An example of the context model used in Ubiquitous Message System prototype.*

Example of the use of such organizational structure is the context-tree for the Ubiquitous Message System (UMS) prototype as depicted in figure 2.6. The whole environment corresponds to the physical space of the Department of Informatics University of Fribourg (DIUF) and is represented by the context-root. Each office is represented by the context-nodes at the second level. The third level represents the people inside the building. Finally leaves represent all the resources that can be manipulated by the UMS application. This structure is flexible and dynamic so it can monitor user and resource mobility quite efficiently.

The role of the context module is to maintain the representation of the context-tree according to the actual configuration of the physical environment. Upper layer modules, typically the coordination module, can profit of this structure to have a consistent representation of the environment according to the application model of the context.

The context infrastructure consists of a number of components called context provider services that sense the information form the physical environment. These include sensors that track location of entities, the condition within an environment (e.g. light, temperature and position) and other external conditions. The nature of such services is related to the availability of physical devices in the environment each time given. These services, in fact, are defined as classical UBIDEV services.

Context in UBIDEV is also considered to be active because, together with representing the logical structure of the physical environment, each context entity of the context-tree supplies an access to the contextual information gathered from the portion of the environment it represents.

As in any other context-aware system, UBIDEV allows to sense the information from the physical environment and to represent it in a sort of knowledge base at application level. Instead of defining and implementing its own model for representing the contextual information, UBIDEV defines a set of specifications and the suitable abstraction to allow applications to manipulate and infer on such information according to a well defined model of the context. In such a way, different context-aware approaches could be implemented as a context module for UBIDEV without affecting the way the application manipulates the knowledge created on it. Context knowledge is constructed in three phases:

- **Gathering:** is the process of acquiring contextual information provided from the physical environment provided by software and hardware sensors.
- **Classification and store:** in order to be used by a context-aware application, this context information needs to be structured according to the application conceptual model of the context. The Classification phase structures this context information according to the application conceptual model (the ontology). The resulting context knowledge is then stored and accessed by the coordination module.
- **Access:** is an interface to the context knowledge as in a classical database system. The model of the context is expressed in terms of concepts of the application ontology. Access guarantee to coordination module the up-to-date values associated to each concept stored.

This is similar to what Chen denoted in [CK00] by sensing the contextual information, and reasoning about contextual knowledge. For each of these steps UBIDEV defines a computational module that ensures the upper layer module, the coordination, to manipulate contextual knowledge according to the application conceptual model, the application ontology.

Using this approach, the application can retrieve the specific contextual data in a way that is decoupled from the service used for acquiring the data [MH03].

These phases allow implementing any gathering service and store system as long as the classification is done according to the application ontology. In such a way the semantic consistency of the contextual information is granted because the application can infer on the resulting knowledge base only through the same ”addressing by concept” defined for resource access.

2.8 Services

In a dynamic environment, resources not only exist, they are created, deleted, and manipulated. In a Service Oriented Architecture [Bie02], service is the highest level of abstraction defined; it should capture the application functionalities and, at the same time, it should be expressed in terms of manipulation of the underlying resources. A coordination space based on service oriented architecture should be defined in terms of composition of homogeneous entities that interact with each other through simple interfaces. These requirements lead the definition of a UBIDEV service as an atomic action that transforms an input resource yielding a new resource as output. Services manipulate resources through the UBIDEV protocol that allows them to address concepts, that represents the semantic capabilities that the resources are expected to supply. For this reason 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) as depicted in figure 2.7.

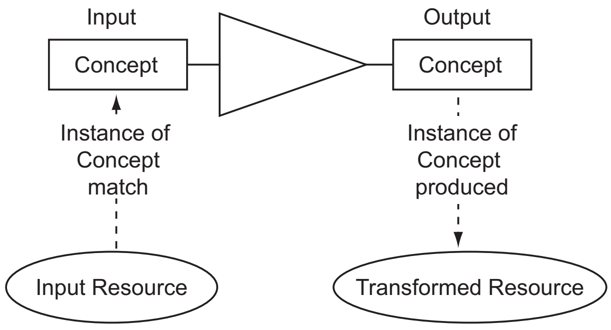

Figure 2.7: **UBIDEV Service**. *Services are described as a transformation from an input resource yielding an output one. Their description and access method relies on the application ontology.*

The main question the service support has been conceived to address is:

- *"How to create and control the execution of services according to the actual configuration of the underlying environment?"*

As for web service technology[TBGC01], UBIDEV services need to be described in order to allow the system to inspect and create them. Differently from classical distributed systems, the service support does not rely on the Client/Broker/Server architecture. The centralized management of the services allows UBIDEV to have a full control and a complete knowledge of the topology of the service dimension of a pervasive computing system. Service description is analyzed by the service module in order to identify the service dependencies and to expose the service interface to coordination.

The service module relies on two main abstractions that capture the dynamic nature of the underlying environment and, at the same time, present services to the upper layer as homogeneous autonomous entities. Those abstractions are 1) the service dependencies that the system has to analyze in order to build a suitable execution environment and 2) the environment itself that embodies the service heterogeneity exporting only its functional interfaces.

EXecution Environment

We argue that operating systems and middleware for pervasive computing system must take into account the dependencies between software components as well as the dependencies between software and hardware components. Finding a suitable representation of such dependencies would allow implementing services that can configure themselves and adapt to every heterogeneity and dynamic environment.

To address the problem from the service management viewpoint, UBIDEV explores the service dependencies in terms of *EXecution Environment (EXE)*: requirements for loading a service into a runtime system. As long as UBIDEV knows the requirements for installing and running each service, it can automate the installation and configuration of new components. It can improve application performance by analyzing the dynamic state of system resources, analyzing the characteristics of each service module and configuring each of them in the most efficient and suitable way. It can also adapt the configuration policy to the contextual information the environment provides, resulting in a fully context-sensitive system. Requirements usually are expressed as dependencies on both persistent and dynamic resources.

Service's EXecution Environment (EXE) must specify any special requirement needed to load, configure and execute the service. It is included in the service description together with the input and output concepts. Even the EXE is expressed in terms of the application ontology since the semantics of resources, hence their capabilities, is captured in the classification phase. A service manager might use these information to determine where, how and when to execute the service.

The analysis of the inter-component dependencies, expressed in terms of relation among services and between services and the context, can help to automate and improve the configuration process. UBIDEV can scan the EXecution Environment to ensure that all concepts required for the execution of a particular service are met before the service is instantiated.

This can also prevent many problems that are common in existing systems where detection of the lack of a particular resource happens only after a service is running.

capsule

A UBIDEV service is encapsulated in a specific dedicated environment that fulfills its requirements according to the service's EXecution Environment. This environment contains all resource access information the service requires for execution.

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 coordination all heterogeneous aspects related to the execution of the service it embodies. A *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 organizational unit to encapsulate a service computing environment within the system architecture, just as a process is an organizational unit for the components of a running application [Tan97].

UBIDEV also enables the reconfiguration of services that are already running. In fact, when the service EXE is no longer met, it can freeze a service's state (including its runtime dependencies) by suspending the execution of the corresponding capsule. Every change that occurs in the environment leads to a reclassification of the resources; UBIDEV, then, analyzes the services EXecution Environment in order to verify whether they could be met according to the actual classification of the environment or not. If so, UBIDEV can resume the service's execution by using the frozen runtime dependencies as the new additional EXecution Environment for loading the service. Although UBIDEV does not guarantee safe reconfiguration by itself, it provides a valuable framework for developers to implements safe reconfiguration easily and uniformly.

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.

The main advantage of facing the heterogeneity problem with the capsule abstraction is at coordination level, where the system is seen as an unified mechanism for dynamic communication, coordination, and sharing of homogeneous objects (in distributed systems, JavaSpace [Jav03] is a good representer of such a mechanism).

UBIDEV Service Abstraction in Pervasive Computing

Service management in pervasive computing implies the support for configuration, description, discovery, composition and execution. UBIDEV addresses these issues as follow:

- **Configuration:** the centralized management of services allows UBIDEV to analyze the service dependencies before creating it. When a service is instantiated, its execution is granted by the fulfillment of its EXE.
- **Description:** service description help the service module to analyze the service's dependencies and to register the service interface. The interface is expressed in terms of input and output concepts.
- **Discovery:** in UBIDEV there is no need of a fully functional look-up service because the discovery of services is needed only at coordination level when is required to build the composite service that solve the user query. Each subcontext holds a minimal knowledge about the services instantiated in it. The centralized management of service allows the UBIDEV to have a full knowledge of the service dimension of the system.
- **Composition:** the service composition is a responsibility of the coordination module. It analyzes the service interface and, according to its compositional policy it determines the sequence of the service invocation. The description of service interface in terms of input and output concepts, ensure that the composition is done according to the same semantics level.
- **Execution:** Service execution is monitored by the service module that ensure the consistency of the service's EXE through capsule freezing and centralized service invocation.

2.9 Coordination

The need to aggregate or combine small services into larger services is core to service oriented architecture and is the main contribution of the coordination module. In many cases, a single service will act as a front-end to many small services. There are a variety of methods for combining services, including:

- **Pipe and Filters**: Direct the output of one service into the input of another service. (Topology = point-to-point)

- **Orchestrations**: Utilize a high-level scripting language to control the sequence and flow of service execution. (Topology = hub & spoke)

In UBIDEV coordination takes place as a context driven composition of existing services. The coordination module acts as a control unit of the whole system. It centralizes the control of the underlying environment and, together with the semantic interoperability ensured by the lower layer modules, presents at application level a homogeneous coordination space.

Coordination has been defined to address the following motivating problem:

- *"How to steer the execution of an application in terms of selection and composition of services, resources and contextual information according to the changing in the environment?"*

A coordination model is intended to support the specificity of pervasive computing systems and to provide at application level a simple yet complete programming environment that reflects the underlying abstraction model. The role of the coordination is to identify the functional components in terms of their interfaces, to define the relationships among those components, and, according to the structure of the context, to establish a set of constrains to affect the desired properties of the overall system.

Its essential characteristics are:

- Genericity that is obtained through a high level abstraction based on the notion of entity and rules.
- A capacity to handle the dynamism of the underlying environments and the context sensitivity of applications, thanks to the explicit notion of environment;
- A homogeneous management of the contextual dynamism of components by the formalism of role and rules attached to the notion of environment.

The Coordination module proposes a model of the context structured as an organizational domain composed by autonomous Entities (coordinable and coordinative), Media and Laws [Sch01] [ACH98]; the coordination laws ruling the actions of coordinable entities define the semantics of the coordination mechanism allowed in the coordination model. 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, 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 behavioral rules [TCH03]. These rules are applied 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 to drive the service composition strategy and hence the application behavior.

Coordination defines a language for compose and analyze rules and contextual information in terms of:

- **Structure** of the environment in terms of context and rules to define the role of each sub-context (these rules if applied to leaves are interpreted to be role specification for the resources).
- **Interface** to access context knowledge and services (through capsule).
- **Services** that exists as rules belonging to a specific context. The description of these service rules are expressed in terms of input and output concepts. In such a way Coordination is the process of finding the composition of rules that may satisfy the context environment and may accomplish the user query.

The Coordination module determines in which context the query should be solved and applies the solving algorithm taking into account all rules that govern the specific context (that implies also all the rules that are inherited from the context's parent). The result is expressed in terms of composition of rules (services are considered to be part of these rules) that could transform the input concept into the expected output one. Service invocation mechanism is then controlled by the service module that ensures that each service is correctly executed inside its proper execution environment.

2.10 Summary

In this chapter the reference model of UBIDEV has been presented along with its base structure composed of five building blocks. The holistic management of the pervasive computing environment results in the possibility UBIDEV gives to describe the behavior of the system as well as the structure of the application model of the context in a homogeneous coordination space. To close the circle, the initial question is answered by UBIDEV as:

- *"How to define an abstraction of the physical environment that permits to applications from one hand to be described in terms general enough to be not statically dependent of the specificity of a particular environment and, on the other hand, to preserve the functional dependencies that the environment may have on the application configuration and execution."*
  Applications describe their dependencies in terms of concepts that are mapped to concrete resources in a specific context. The coordination of services hence the whole application behavior, is determined on the base of the implicit semantics stated by the classification phase through the application ontology. Ontology allows modeling any kind of dependency which can be used during deployment to ensure proper configuration. The domain specific approach of UBIDEV together with subcontext construction ensures that an application only see relevant resources. Because resource and service selection is based on semantics and not on predefined interfaces, a resource with required semantics is guaranteed to be found if available.

To conclude UBIDEV is a study model for providing a service framework in heterogeneous and dynamic environments like those of pervasive computing. It has been validated with different prototypes as the UMS that will be presented in chapter 4. Still its architecture is far from being complete and, hence to be used as a referring tool for building robust and stable pervasive computing system. The accent of this project is on resource and service management, but the integration of other existing systems that may augment the support for context and coordination as well as for dynamic interfaces, will make UBIDEV a complete functional middleware for modeling pervasive computing system in a flexible and structured way.

Chapter 3

The Architecture

In this chapter the implementation details of the UBIDEV architecture will be presented. Starting from the reference model presented in the previous chapter, we have instantiated the UBIDEV building blocks into a functional architecture that exploits services for managing pervasive computing system through a coordination-based approach. The architecture has been implemented as a collection of collaborating modules each of them embedding the management of one of the building blocks.

3.1 Introduction

In the following sections, the functional architecture will be presented with respect to the reference model. The implementation of the UBIDEV model into a functional architecture had become necessary prroving the overall conceptual model not only at design but also at implementation level was raised.

The architecture introduces 5 modules each of them implementing the functionalities of the UBIDEV building blocks: resource, context, service, coordination and the application; it introduces also a functional module that embeds the classification phase of the environment according to the application ontology.

When trying to give a physical shape to a purely theoretical model like the one of UBIDEV, we had to make some assumptions to restrain the overall model to the constrains and the requirements of a functional system.

As an example, the classification process as been implemented with a certain level of pre-defined knowledge (as given belief); in such a way the system is able to classify ground elements on the base of such knowledge.

Another assumption is related to the definition of the adapter: adapters have already some pre-defined knowledge that is used to access the related devices. Such knowledge is separated from the classifiers and, in a certain way, is used implicitly by the classifiers themselves. This has been done purely for implementation reasons; to ease the adapter and classifier modules.

Despite such implementation assumptions, the resulting architecture still demostrates the fundamental abstractions and approach of the reference model.

3.2 Implemented Architecture

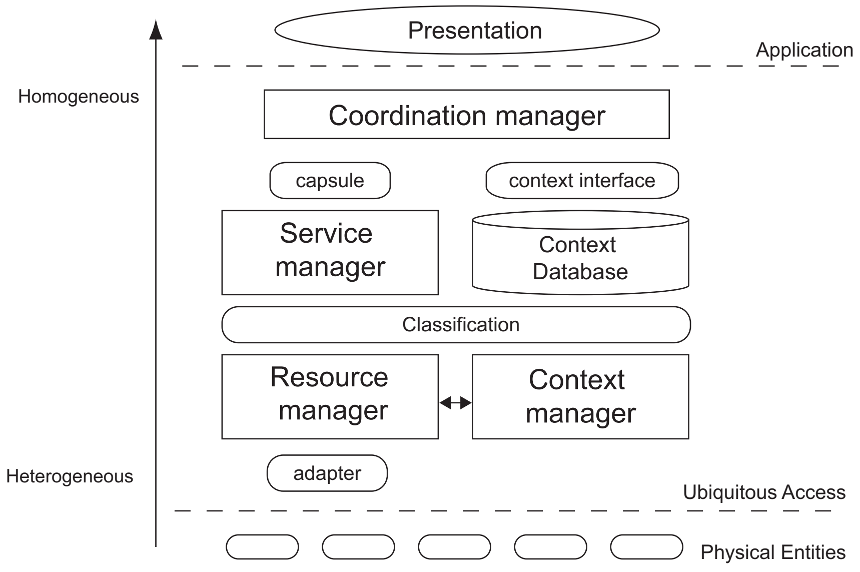

Figure 3.1: **UBIDEV implemented architecture.**

Figure 3.1 shows the actual implementation of UBIDEV based on the reference model presented in the previous chapter. The model building blocks are reflected in the implementation in two ways:

- in the programming model because they represent the abstractions that are used to configure a pervasive computing system;
- in the implementation of the Ubiquitous Access Layer because every implemented module takes in charge of the management of the corresponding building block abstraction: the *resource manager*, the *context manager*, the *service manager* and the *coordination manager*. The classification phase separates the physical environment management level from the system management level.

As a result there is a direct correspondence between the modeling phase of a pervasive computing system and its implementation. At the technical level UBIDEV leverages Java technology to implement the on-the-fly discovery, selection, run-time (as opposed to design time) composition of services and on the execution of such composition; the composition represents the user tasks. UBIDEV relies on XSB [SSW94], a Prolog engine, to implements the semantic interoperability level; thus service description and context representation and the application ontology are expressed in terms of Prolog facts.

The main novelty of this implementation resides in the context-centered resource and service management and in the way the corresponding modules (*resource manager*, *context manager* and *service manager*) collaborate to define the abstraction of the underlying environment at the coordination and the application level. Even if the presence of the Java virtual machine, as an existing abstraction layer, reduces the level of control that UBIDEV may have on the execution process of the application, the management of resources and the control of the service instances in a distributed environment makes UBIDEV appear as a Distributed Operating System for pervasive computing scenarios.

The following subsections present the different module of the architecture and the mechanism that governs the inter-module interactions.

Modules

This section presents the role of each of the modules depicted in figure 3.1 with respect to the abstractions of the reference model.

- The *Resource Manager* handles the communication with the physical entities through standardized interfaces (*adapters* in figure 3.1). It implements the methods for resource reservation and a loosely coupled lock-and-transaction mechanism for resource access. It is also responsible of maintaining a consistent representation of the physical environment for the other modules in terms of resource federation.
- The *Context Manager* embodies the context-awareness infrastructure. It is organized as a knowledge base system sensing the contextual information from the physical environment and structuring it in context types. The *context database* stores these context types in a collection of Prolog facts. When *coordination manager* solves user tasks, it requires the evaluation of a subset of such facts through *context interfaces*. The *context manager* is also responsible of maintaining the logical organization of the physical environment. Through the same interface, *coordination manager* can infer information about the environment in terms of concepts representing the contextual information related to the physical environment (i.e. the location of a user or a resource with respect of the logical organization of the environment).
- The *Classification* module defines the abstraction of the underlying physical environment according to the application conceptual model. In doing so it states the mapping of the resources and the contextual information into concepts taken from the application ontology expressed as a collection of Prolog facts. In this way the whole system is described according to a common ontology that reflects the application conceptual model. Upper modules access resources and contextual information formulating complex Prolog-like questions based on these concepts. That ensures the semantic interoperability between resources inside the same pervasive computing system.
- The *Service Manager* controls the lifecycle of each single service. It ensures to the *coordination manager* the abstraction of a generic homogeneous running service that can be invoked through uniform and simple execution interfaces (*capsule* in figure 3.1). Capsule works as service access points for the *coordination manager*. This is one of the fundamental abstractions that help to separate the application functional level from the control of the specific environment.
- The *Coordination Manager*'s main goal is to interpret the user tasks, defined at application level, in terms of composition of existing services. It accesses these services only through the I/O interfaces defined by the capsules. It can also access contextual information through context interfaces. The composition algorithm is affected by the analysis of the contextual knowledge provided by the context interface, by the availability of the services inside a specific context environment and by the analysis of the

behavioral rules of each sub-context. In such a way the *coordination manager* adapts the service composition policy and hence the application behavior, to the specific contextual configuration.

- The *Application layer* embodies the GUI modules that represents the system interface. UBIDEV application framework allows defining the shared ontology, the model of the context as well as the behavioral rules attached to each context module. The system lacks of a suitable component that allows the upload of the GUI modules on the required devices. The responsibility of the distribution of these GUI modules is shared between adapters and off-line upload. Moreover it turned out to be quite difficult to clearly draw a border between generic application modules that could be realized as middleware services and application specific ones. For such reasons the implementation of the *presentation* module has been omitted in the architecture. One instance has been realized for the Ubiquitous Message System prototype as will be described in next chapter.

As a result, developing a pervasive computing system in UBIDEV means:

- define the common ontology that will be used to ensure the semantic interoperability level;
- define the base services that will compose the minimal functionalities of the system;
- define the model of the context and
- define the behavioral rules that would be applied to each context module.

In addition the system will have to be completed with:

- the pool of classifiers compatible with the chosen ontology,
- the pool of adapters for the expected resources and
- the GUI modules that will compose the system interface.

Module Interaction

UBIDEV modules are distributed objects and require communication services to support remote interactions. The current implementation uses a push-based informative bus technology, an event notification mechanism and RMI-based APIs [SM98]. It is possible to port UBIDEV to other communication architecture like CORBA [gro02] or SOAP [GHMF01] for a more uniform interaction mechanism.

- The *informative bus* paradigm [OPSS93] [LCBB00] models the communication between the physical environment and the *resource manager* and *context manager*. The UBIDEV informative bus is a collection of Java-based libraries that allows different distributed entities to publish and retrieve data. It is organized in a form of parallel channels each of them working as a transport mechanism for different classes of events. Resource authentication and context sensing services generate context events that are published on the informative bus; *resource manager* and *context manager* are subscribers of such a bus and are notified whenever an event they have subscribed for is fired. This action allows both *resource manager* and *context manager* to react in real-time to the changes in the environment. The implementation does not take into account all the specification of an informative bus [OPSS93] because of the simplified scenario of UBIDEV in which *resource manager* and *context manager* are not considered to be mobile entities that may change during the lifecycle of the system. The main advantage of such technology is the possibility to decouple the publishers form the subscribers allowing dynamic registration of new services. For example, whenever a new instance of a context sensing service is created, it automatically registers to the bus as a publisher. That allows *resource manager* and *context manager* to subscribe for a class of events and not to well known event sources. The classes of events that are generated on the bus are statically defined and every context sensing service must adhere to their specifications. Like tuple spaces, the information bus helps with coordinating loosely coupled services [OPSS93]. Unlike tuple spaces, it is based on a publish/subscribe paradigm and does not retain sent messages in storage. While its design is nominally object-based, data exchanged through the bus is self describing and separate from service objects, comparable to the separation of data in the form of tuples and functionality in the form of components in UBIDEV.

- The Java event notification mechanism addresses the interaction between the other modules of the architecture. The core of UBIDEV is centralized in the same address space. Event notification allows modules to interact with each other asynchronously; it is also convenient for decoupling information suppliers from information consumers; a crashing supplier can be automatically replaced without disrupting the system. UBIDEV does not implement a dedicated event manager as in [RHC$^+$02]; it relies on the Java event mechanism defining proper event sources, event types and event subscribers. It supports the following type of notification:

  - The *resource manager* notifies to classifier a resource change event. The *resource manager* is responsible of monitoring and maintaining the resource federation. When it receives a context change event; it analyzes if the change concerns a resource (for example a new resource enter the environment or an existing one leaves). Classifier will then classify the new resource producing a classification table, or it will remove the classification table associated to the resource disappeared.
  - *resource manager* notifies to *service manager* resource change events. *Service manager* react checking service descriptions from the service description pool to determine whether a service can be instantiated on the newly classified resource or not. In doing so it queries the classifier for the classification table of the new resource. This request is done through an API in order to ensure that the classification process is correctly terminated on the new resource before the *service manager* request.
  - *context manager* notifies to *coordination manager* of a context change event every time context database is updated. Database update happen when *context manager* is notified of a context change event. *Context manager* then extracts the data context information and classifies it through classifiers. Then it updates the context database and notifies the context change event to *coordination manager*.

3.3 Resource Manager

The principal role of the *resource manager* is to support the functionalities related to the management of UBIDEV resources in terms of description, configuration and access. In doing so it implements the concepts defined in the reference model: adapter, classification and the access protocol. During the execution of the system, it is also responsible of the reservation of resources for service instances. Reservation does not imply exclusive allocation. It's a soft-allocation scheme used to monitor the resource utilization and to control resource reservation. Whenever a service require an access to a specific resource, *resource manager* reserves and locks the resource for the operation requested by the authorized service; locking ensure that a service has an exclusive access to the resource only during the time required by the single operation.

The services offered by the *resource manager* together with the physical entity that exists in the environment make the concept of a UBIDEV resource. A resource, hence, is the composition of a physical entity together with its adapter, the result of the classification process and the protocol handler as illustrated in figure 3.2. This combination allows realizing the stimuli interaction process between UBIDEV and the physical entities in the environment.

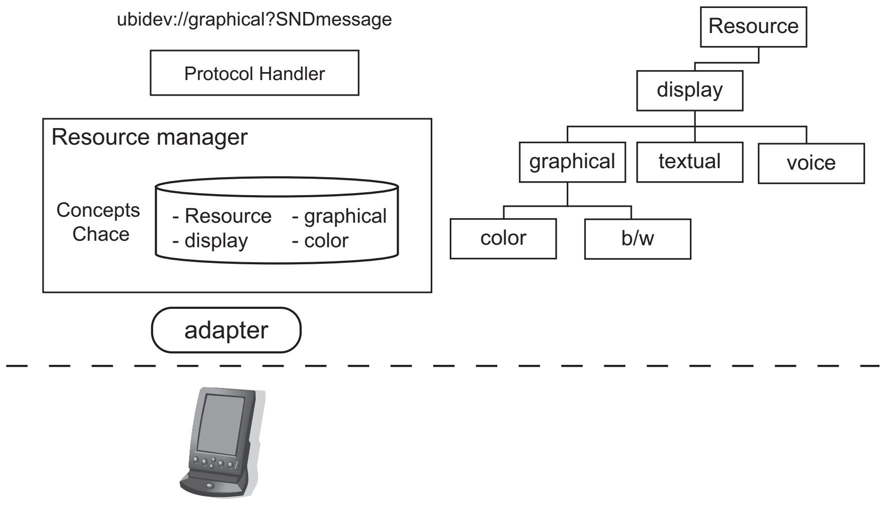

Figure 3.2: **UBIDEV Resource Abstraction**. *The virtual representation of a physical entity inside* UBIDEV *is composed of an adapter containing the specific resource access information, of the result of the classification process that allow to abstract the resource according to the application ontology and of the access protocol that grant the consistence of the mapping between concepts and resources.*

Adapter

An adapter works as a device driver [Tan01] to allow services, through *resource manager*, to access a physical entity. In the implemented architecture, adapters are defined as Java libraries that are instantiated at runtime whenever the corresponding physical entity is detected into the environment. *resource manager* through a process of pre-classification, determines which adapter should be associated to which entity. The role of an adapter is threefold:

- It contains the specific resource access information that allows exchanging data to and from the entity. The modularity of the implementation ensures that a new adapter (a more suitable one, for example) can be loaded dynamically without affecting the execution of the system. In the Ubiquitous Message System prototype, for example, the portable phone entity is associated with an adapter the uses diuf.sms library [diu] that allows sending SMS messages. A possible extension of the system may replace such an adapter with another one supporting MMS capabilities.
- It allows classifiers to access information of the entity and hence to classify it according to the application ontology. This is ensured by a simple interface that adapters are supposed to implement. Different adapters for the same resource may lead to different classification. For example the portable phone adapter predefined for the Ubiquitous Message System prototype lead to a classification in terms of display, textual and voice. The extended adapter allows classifier to inspect the resource for the MMS compatibility, leading to another classification of the phone as display, graphical, textual and voice. This information will be used by *service manager* to instantiate a service that allows sending images to the portable phone.
- It defines a uniform API for *resource manager* to exchange data with the entity. Even if this level of access is transparent to services (because they access resources only through the ubidev protocol), a set of generic interfaces has been defined in order to maintain the system modular and more flexible.

The actual implemented architecture allows to associate to an entity one and only one adapter each time given. Adapter composition is possible but not implemented. Through adapter composition it would be possible to define abstraction of more complex resources (up to an entire system for example). Despite of the powerfulness of this approach, we believe that it would lead to a less flexible system that would require a different approach in the management of the dynamism of the resources.

Classification

The classification process has been implemented in a separate module for merging the need of classifying both the resources belonging to the environment and the contextual information related to such resources. Classification module takes in charge the operation of creating a table of semantic-rich concepts for each resource or contextual information. In doing so it relies on a pool of COCA classifiers that takes as input an instance of a UBIDEV resource and produces as output a list of the concepts they know. The result of the classification process is stored in a cache that other modules will use to both access resources and infer contextual information. The way modules access this cache of concepts is specified by the COCA interfaces [Sch02]. The actual implementation of the COCA model is based on the Java language resulting in a simplified integration process of the COCA classifiers into UBIDEV.

One way of defining a classifier for a resource is to wrap an existing classification scheme and embed it into the classifier class; in such a way the most part of resources could be classified without having to write any particular resource dependent analysis. For example it does not make much sense to redefine the semantics of the concept ”Wav document” when there are media players available that can easily decide if a file is in a wav format or something else. Thus building a classifier for the concept ”Wav document” means simply invoking a media player and looking at its exit code. This leads on the one hand to direct reuse of semantics and knowledge, encoded and available in computer programs today and on the other hand it helps constructing new semantic out of the existing pool of programs.

The modularity of the COCA specification allows deploying a pool of different classifiers that could be re-used in different applications. Similarly to the rest of the system, in fact, the level of standardization they are supposed to meet is at the ontology level. As an example, the classifiers used in the Ubiquitous Message System prototype for document classification (ascii, pdf, ps, bmp, gif,), have been reused for another UBIDEV prototype called Document Classifiers [pai]; the adaptation of a classifier from one system to another one is on the mapping from the ontology of the former to the ontology of the latter. Another example of sharing ontology is brought by the need on inter-application communication; in both cases the COCA model proposes three approaches [Sch02]:

- **Explicit declaration of concept equivalence**: Meta ontology may be used to explicitly state equivalence of concepts in different ontology. This meta-ontology will require careful maintenance, likely done by humans.
- **Proving concept equivalence**: If two concepts in different ontology are defined by the same classifier they are considered equivalent allowing inference of further properties through each ontology relations.
- **Inferring concept equivalence**: Concept equivalence may also be automatically inferred by observing classifiers by so-called meta-classifiers. That is, the automated version of building meta-ontology. Concepts may be considered equivalent as long as different classifiers consistently output the same concepts for a set of resources under observation.

Protocol Handler

In multithreaded and multiprocessing environments, additional care must be taken with regard to reliability and consistency because two threads accessing the same object concurrently, for example, may leave the system in an inconsistent state or cause its failure. In UBIDEV concurrent access to resources is managed by the *resource manager* with the help of the adapter abstraction and a lock-and-transaction policy. Requests to a resource access is captured inside the syntax of the ubidev protocol defined to allow any client (service) to request an access to a resource inside its execution environment.

The ubidev protocol handler interprets the requests and converts them into specific resource access requests for the *resource manager*. *Resource manager* uses, then, a lock system to guarantee the exclusive access to the resource adapter through the whole time necessary to elaborate the request.

There is an instance of a protocol handler inside every service execution environment; the local protocol handler forwards the requests to the resource manager that guarantee the exclusive access to the resources as depicted in figure 3.3.

According to the specification of the protocol, each client requires resources sequentially; that implies that they could handle to wait until the current resource access request is executed. This approach may not look like an optimal one since it supposes that the order of the resource access request is always sequential.

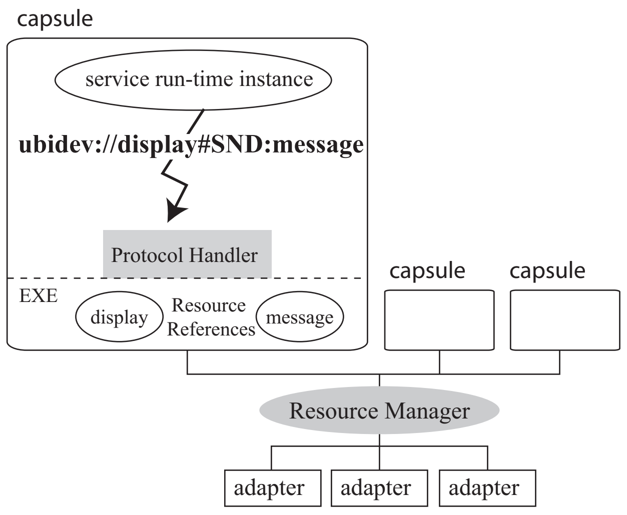

Figure 3.3: **UBIDEV protocol handler**. *The consistence of the mapping between concepts and resources reference is evaluated inside the service execution environment by the protocol handler.*

In the execution of a UBIDEV service, location transparency is achieved by packaging all resource access requests into the access protocol. All resource access requests are intercepted by the dedicated protocol handler that is responsible of maintains the consistency of the request according to the availability of the resources inside the service's execution environment. In this way a service is never aware of its specific location as long as its prerequisites are met by the underlying environment.

3.4 Service Manager

Service manager controls the execution process of services. According to the Service Management module, it is responsible of:

- Analyzing the service pre-requirements to determine whether a service is entitled to be instantiated or not;
- Initialize a generic capsule: an execution environment that will guest the service run-time modules and the resource references as specified in the service EXE description;
- Together with *resource manager* instantiate a service inside its dedicated capsule,
- Ensure that the service invocation requests are fulfilled and
- Monitor the service execution process according to the changes that occurs in the context.

The granularity of the distribution of the system is the service; user tasks are translated and solved invoking multiple services each of them instantiated by the system on a specific execution environment. That makes the pervasive computing system looks like a parallel distributed system. If during the execution of the system new resources become available, the resource manager notifies *service manager* of the reclassification of the environment; *service manager* is designed to utilize these new concepts by possibly reassigning resources to services.

Figure 3.4 shows a simplified version of a state-transaction diagram describing the control of the *service manager* over a service. *Service manager* may react at two classes of events:

- a resource change event fired by *resource manager* or *context manager* and
- an input resource change event fired by *coordination manager*.

The former happens when something changes in the physical environment (or in its logical representation); for example a resource that leaves the environment or a new resource that joins. *Service manager* extracts from the event the information about the modified resource and checks whether the associated classification may influence the pool of services or not. Two scenario are considered here:

- When a resource leaves, for any reason, the environment. That means that all the services associated to such resource are not entitled anymore to be ready. in such a case, *service manager* stops the execution of the service, dumps the associated capsule state and tries to meet the service pre-requirements from the updated classification pool.
- When a new resource enter the environment. This produces a new classification process; hence the *service manager* checks the classification pool to find out concepts that may meet the pre-requirement of one service from the service pool. If so, it instantiate the service and encapsulate it into a new instance of a capsule.

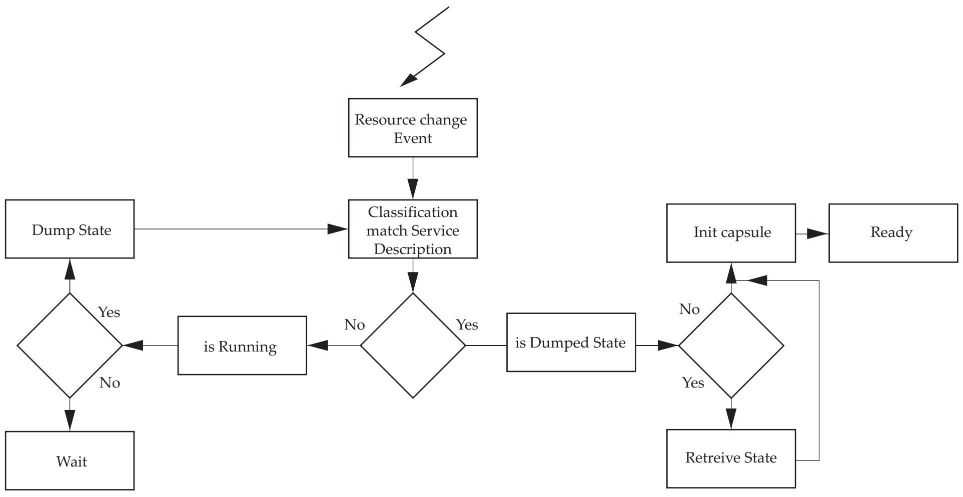

Figure 3.4: **UBIDEV Service diagram**.

Service Description and EXE

UBIDEV services are described using an XML-like proprietary description format. Further extension of the system will include the standardization of service description to WSDL

[CCMW01]. That will also simplify the opening UBIDEV to web-services. Service description includes information about the service name, its interfaces in terms of input and output concepts and the prerequirements.

Service manager parses the service prerequirements in order to determine whether the current service is entitled to be instantiated or not. If so it issues a resource reservation request to *resource manager* that replies with the corresponding resource access reference. Resource access reference together with the service run-time module and an instance of the ubidev protocol handler are encapsulated into a new capsule. In the development of a UBIDEV service, developer must ensure the correspondence between the concepts required in the service description and the effective ubidev protocol invocation format.

Capsule

UBIDEV services are able to act upon stateful resources providing access to, and manipulating a set of logical stateful resources based on messages sent and received. That means the a UBIDEV service executes against dynamic state, i.e., state for which the service is responsible between message exchanges with its requester. UBIDEV service is stateless because it delegates the control of its internal state as well as the state of the associated resources to the capsule. In this way service migration is made easier because the capsule allows to dump its complete state, and because the representation of the state does not change from service to service. However, the consistency of the use of the service state has to be ensured by the service implementation. Whenever a service prerequirement is no longer met, *service manager* stops the execution of the corresponding capsule and tries to reconfigure it. UBIDEV supports completely transparent capsule reconfiguration. After a reconfiguration process, a service inside a capsule continues to interact with its environment regardless of the changes. The capsule state is divided in two sub-states: the service state which can be migrated and the user state that is related to the model of the context and may not be migrated. The service state contains the program code and a representation of the service data. The user state contains user dependent data that are organized according to the model of the context. For this reason it is not necessary for this state to migrate because the service can access it as a contextual information provided by its execution environment.

This approach is robust and efficient because the consistence of the mapping between concepts and resources is always granted by the middleware; so services do not have to take into account resource specific information that are not classified according to the application ontology.

Context Sensing Services

The context sensing services lifecycle is controlled by *service manager*. For sake of uniformity, they are instantiated into capsule whose interface is NULL; in such a way the *coordination manager* won't take them into account when solving user input queries.

3.5 Context Manager

The context infrastructure is twofold:

- Provides the contextual information in a form of knowledge base as in a classical context-aware system [CK03], [BZD02].
- Represents the logical structure of the physical environment as context-tree.

It has been developed around XSB, a Prolog engine. The contextual model of the environment as well as the contextual information is represented as a collection of Prolog facts that are evaluated at run-time for the *coordination manager*.

Context-awareness

Figure 3.1 illustrates how context management is implemented in a form of knowledge base abstraction for the coordination and application modules. The context-aware infrastructure works as a classical database system providing interfaces (context interface in figure 3.1 for clients, a query interpreting system that is also responsible of decomposing input queries into more low-level ones, and a synthetizer engine that assemble the results for the clients. It is divided in four modules each of them implementing one of the required functionalities of a knowledge base system:

- **context gathering:**
  - acquire data from context sensing services
  - register to context event class on the informative bus
  - extract data from notified events
  - request to classify data in context types
  - send classified results and data to Context Knowledge Base engine
- **context classification:**
  - classification of context information into context types. It relies on the Classification module that returns a list of concepts and their relation for the requested data.
  - classification allows decoupling the conceptual structure of the contextual information from the sources used to acquire data. The location information, for example, could be classified with a more abstract position concept allowing *coordination manager* to infer information about the location of a specific resource without having to know where and how these data are obtained. This level of abstraction is very important because it allows describing the behavior of a system in generic terms. The instantiation of these terms is evaluated only at run-time, when the physical environment characterizes the whole system.
  - context types are specified at ontology level and the classifiers grant the compatibility of the data representation with the associated context type.
- **context database:** the knowledge engine stores the result of the classification as well as the context data values, in a form of Prolog facts.
- **context interface:** it handles the interfaces to the knowledge base engine as in a classical database system. Queries are formulated still using Prolog; XSB takes in charge the interpretation phase.

Context Model

The context-tree is defined at application level as a sequence of Prolog facts. The context engine interprets these assertions and creates an instance of the context-tree interface for the *coordination manager*. Context-tree contains also the behavioral rules applied by the *coordination manager* when solving user tasks. Once the instance of the context-tree interface is created, *context manager* is responsible of maintaining the consistence of its structure with respects of the changes in the physical environment. The context change events are interpreted and modify the context-tree. Context engine extracts the concepts modification and builds the new prolog facts; XSB, then, re-evaluates the whole pool of assertions in order to update the context knowledge. Experience in the Ubiquitous Message System prototype has shown poor performance in the maintenance of the contextual knowledge base also because the pool of facts tends to grow when the context-tree has a complex structure.

3.6 Coordination Manager

Coordination manager takes in charge of managing the dependencies in the interaction process of the application-level user task solving.

Coordination manager manages the dependencies between services and contextual information when, at application level, a user task request is defined. These dependencies influence the service composition that is at the hart of the whole coordination process. To better describe these dependencies, we need to identify and separates the computation and the coordination aspects of a generic pervasive computing system.

Coordination manager has been implemented as an active orchestrator of the interactions (coordination) between services (computation) within a structured context space. Differently from classical implementations of coordination spaces, no explicit coordination is defined at development level; *coordination manager* reacts to the application level requests expressed in terms of user tasks. Such tasks are characterized by an application service expressed in terms of resource transformation. The current implementation of the *coordination manager* uses a first order logic to represent the terminological knowledge of an application domain in a structured and formally well-understood way; more precisely it is used to describe behavioral and contextual rules as well as the application ontology (see an example of system ontology in figure 3.5). Such rules are expressed as Prolog facts that are evaluated (all or just a part of them) when solving the user task. First activity of the *coordination manager* is to determine, according to the configuration of the context and according to the user task, which are the rules that have to be evaluated. The result of the evaluation determines the run-time dependencies and constrains that *coordination manager* takes into account in the service composition process.

Thus, the resulting composite service that is in charge to accomplish the user task, is influenced by: the active part of the context where the request has been generated, the social rules defined for such context, the availability of resources and services, the social laws related to the context, services and resources. The core of *coordination manager* is based on XSB Prolog engine. *Coordination manager* can not be directly programmed but only instructed by a richer set of Prolog facts. By analyzing both contextual and behavioral rules, *coordination manager* has a full knowledge of the status of the environment and of its role.

Service interaction is achieved by requesting service invocation to *service manager* inside

```
@ums:ont {
 // ontology
 ums:display[rdfs:subClassOf -> rdfs:Resource]

 ums:textual[rdfs:subClassOf -> ums:display]

 ums:graphical[rdfs:subClassOf -> ums:display]
 ums:bw[rdfs:subClassOf -> ums:graphical]
 ums:color[rdfs:subClassOf -> ums:graphical]

 ums:voice[rdfs:subClassOf -> ums:display]

 ums:Document[rdfs:subClassOf -> ums:Resource]

 ums:text[rdfs:subClassOf -> ums:Document]
 ums:html[rdfs:subClassOf -> ums:text]
 ums:ascii[rdfs:subClassOf -> ums:text]

 ums:sound[rdfs:subClassOf -> ums:Document]
 ums:wav[rdfs:subClassOf -> ums:sound]
 ums:adpcm[rdfs:subClassOf -> ums:sound]

 ums:image[rdfs:subClassOf -> ums:Document]
 ums:gif[rdfs:subClassOf -> ums:image]
 ums:bmp[rdfs:subClassOf -> ums:image]

 ums:http_net[rdfs:subClassOf -> ums:Resource]
 }
```

Figure 3.5: **System Ontology.** *An extracted of the OWL ontology used for the Ubiquitous Message System prototype.*

a common context space. In this way *coordination manager* is not aware of where exactly each capsule physically is, but it only has knowledge of the relative position inside the context structure. The *resource manager* analyzes resource dependencies to determine its position in the context. Service composition means invoking two services chained on a common resource (the output resource of the first is the input of the latter). Service sequencing is specified by behavioral rules.

1. *Coordination manager* analyses the user task to determine the context dependencies.

2. It evaluates context-rules and context information whose pattern matches user task; in such a way it can determine the context in which the user task should be solved. User's identity and related roles are evaluated in this phase.

3. It then evaluate service/capsule pool interface to find a path from user task's input resource to the required output resource among the services that matches the just evaluated environment (i.e. an instance of a service could be related to a resource the user is not entitled to use; this is specified by social rules that limit the role of the resource within the environment).

4. If a path could be found, then *coordination manager*, through the *service manager*, chains the input resource to the first service of the path; that produces the service invocation and the consequent resources transformation. *Coordination manager* re-evaluates the environment according to the perturbation produced by each service invocation. In such a way the system can also tackle the run-time modification of the environment that may occur during service invocation. *Coordination manager* ensures that the produced output resources are chained to the next service in the found path.

5. Output resource is then produced as either a new resource belonging to the environment or as a modified existing one.

6. User perceive these actions in terms of perturbations produced to their surrounding environment (like displaying a message on their personal phone or print the document to the nearest printer.

The Coordination model allows to define rules for every context that may also restrain the algorithm of resource and service management (i.e. if a service has to be related to a user-context only, *service manager* does not propagate the requests to the upper contexts, but just freeze the capsule until the service EXE is still available). These rules are specified by the Context itself; the structure of these rules is outside of the contribution of this project because is one of the key results of the XCM model [TCH04], still to be integrated into the UBIDEV middleware. Further details may be found in chapter 5.

3.7 Summary

This chapter has presented an implementation of the UBIDEV reference model. The goal of the architecture is to reflect all building blocks of the model and maintain their interdependence and role. Differently from the reference model, some implementation assumptions have to be made.

The interest of this research project is not on the deployment of a fully functional system but, instead, on the analysis of resource and service management for pervasive computing system. The implementation of UBIDEV, hence, has been driven by the need of proofing the architectural concepts defined in the model. Some aspects like security, scalability and efficiency, were not taken into account as fundamental issues that the system should address. That does not imply that UBIDEV does not scale on large systems, but its actual implemented architecture does not emphasize these aspects. Another important aspect related to the implementation phase is that UBIDEV is not intended to be a fully open distributed system; it focuses on how to model the environment in order to allow applications to configure and to adapt to the changes that occur in such environment. For this reason aspects related to inter-platform and inter-environment interoperations are not directly addressed by the model.

The centralized management of resource and services is a mechanism easy to implement and use. The potential scalability bottleneck introduced by the inherent single point of failure of such approach will be addresses within future works, where a replica of the core services will be introduced.

Chapter 4

A Validating Example: Ubiquitous Message System

Ubiquitous messaging is the ability to connect people through anywhere-anytime messaging via any kind of device and communication media. Access transparency means that users do not distinguish between different communication platforms. They expect a single messaging application, along with their messages transferred from cell phones to pagers to PCs. How do these attributes apply else where in pervasive computing? Consider an application for a tourist: It will be expected that the traveler will be able to make itinerary adjustments from a phone, a PDA or a public kiosk in realtime, contacting a live person as required. Location information will be an assumed part of the transaction (as well as, perhaps, a record of the day that might indicate a recommendation on where to get a quick meal). The tourist will even be comforted by the ability to share experiences in detail in realtime, providing blow-by-blow updates to his or her best friends so he or she can create a shared, emotional accent for the new adventures.

4.1 Ubiquitous Message System

Ubiquitous Message System (UMS) has been thought for a scenario where 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 media (i.e. telephone) it is not a trivial task to set up and configure different communication channels because there is virtually no interoperability between messaging context infrastructure and a coordination model. 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 worsened by the absence of information that would allow a sender to determine the best way to reach his or her intended recipient.

Therefore, there is a need for a unified, multipath system for messaging that enables the connection from sender to recipient, allowing each of them to use the tools they prefer. In a pervasive world, a wire-line-based device like a computer or desk phone should be able to detect and route incoming messages based on the proximity of the person who receive 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.

Ubiquitous Message System is a simplified implementation of commercial unified messaging systems like Lotus UMS for Domino [Lot00] and GeneralMagic Portico [Mag01]. It has been developed to show the added value of an infrastructure like UBIDEV in terms of application design.

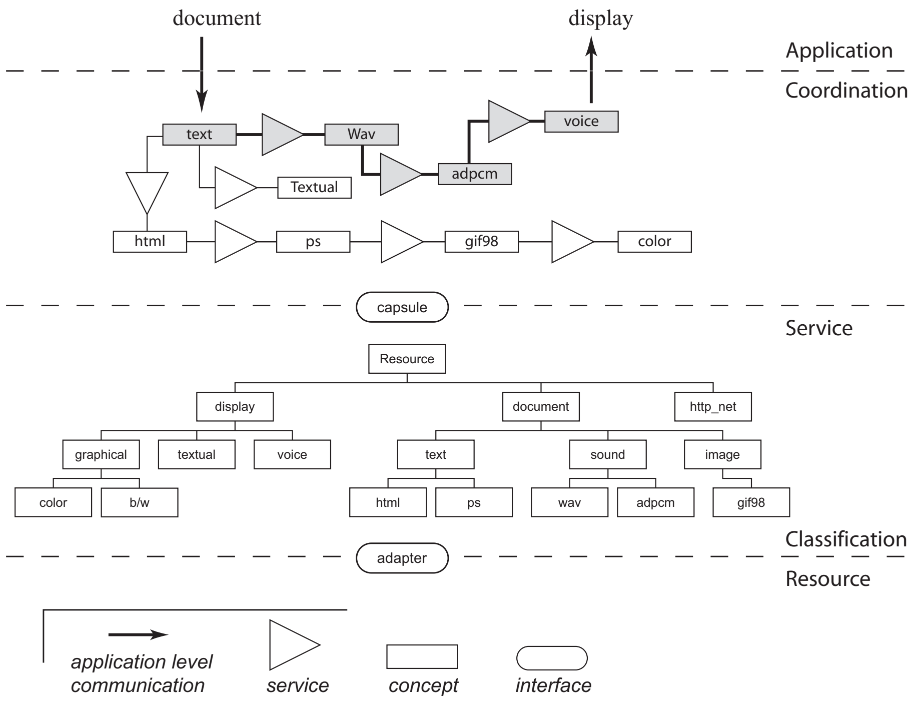

Figure 4.1: **Ubiquitous Message System**. *A composite service can be realized by the coordination manager by composing existing services according to the classification of the underlying environment. Coordination manager solves user tasks by finding the path from the input resource generating the output resource. This path represents a service invocation chain. Service manager takes in charge the service execution process.*

Ubiquitous Message System has been configured to run into the DIUF department. It provides functionalities to exchange messages between different interface devices relaying on different media. The application context is organized as a hierarchy of sub-contexts modeling the physical space and the users. The model of the context used describes one training room (2.56) and two user labs (2.51 and 2.52) inside the DIUF department, as well as three potential users. An extract of the DAML description is depicted in figure 4.2. A portable phone, a mailer, a PDA or the terminal where the user just logged in represents an example of a user context. UMS allows users to exchange messages through a virtual service called *document_to_display* without concerning them about how this service is realized in every user context. Figure 4.1 shows an example of the realization of the abstract service *document_to_display* made by *coordination manager* according to the application ontology and to the availability of resources and services (chained together in a service graph).

Resource management in this prototype means to keep a consistent view of the physical dimension and to control the resource association, allocation and lock for the running services. In the current implementation, "plug" and "unplug" a resource is done manually: there is a dedicated interface to modify the federation. Dynamic control of the resource fed-

```
@ums:ont {
 // ontology
 ums:diuf[rdfs:subClassOf -> rdfs:Context]

 ums:2.51[rdfs:Property]
       ums:2.51[rdfs:subClassOf -> ums:diuf]
       ums:2.51[rdfs:Container]
              ums:2.51[rdfs:li]
                     ums:sergio
              ums:2.51[/rdfs:li]
       ums:2.51[/rdfs:Container]
 ums:2.51[/rdfs:Property]

 ums:2.52[rdfs:Property]
       ums:2.52[rdfs:subClassOf -> ums:diuf]
       ums:2.52[rdfs:Container]
              ums:2.52[rdfs:li]
                     ums:amine
              ums:2.52[/rdfs:li]
       ums:2.52[/rdfs:Container]
 ums:2.52[/rdfs:Property]

ums:2.56[rdfs:Property]
       ums:2.56[rdfs:subClassOf -> ums:diuf]
       ums:2.56[rdfs:Container]
              ums:2.56[rdfs:li]
                     ums:paolo
              ums:2.56[/rdfs:li]
       ums:2.56[/rdfs:Container]
 ums:2.56[/rdfs:Property]

 ums:sergio[rdfs:subClassOf->Resource]
 ums:sergio[rdfs:Property]
       ums:sergio[rdfs:List]
              ums:sergio[rdfs:li]
                     ums:palm[rdfs:Property]
                            ums:palm[id="af7438"]
                     ums:palm[/rdfs:Property]
              ums:sergio[/rdfs:li]
       ums:sergio[/rdfs:List]
 ums:sergio[/rdfs:Property]
 }
```

Figure 4.2: **Context-Tree** *An example of the context model used in Ubiquitous Message System prototype.*

eration is activated as soon as one or more context services are instantiated (like when a IrDA port is plugged); the *resource manager* automatically registers to every new context service; actually three context services are implemented: IrDA sensing, 802.11 wireless device sensing and web client sensing.

Service manager reacts to every resource change event fired by the *resource manager* matching the service description pool with the actual configuration of the resources. It verifies that all running services are consistent with their resource requirements and checks if a new service could be instantiated.

The modifications that occur at resource and service level are reflected also at the coordination level, where the *coordination manager* solves user tasks adapting on the fly the service composition and invocation according to the actual service availability. The context rules defined to steer the service composition policy are re-evaluated at run-time for every user task. In the first version of the prototype the only context rules defined were those to allow the *coordination manager* to combine services by their matching interfaces. That lead the *coordination manager* to solve the user tasks with a ”Shortest-Path-First”[Wir85] service composition algorithm. A peculiar property of the user resource is the preferred language that is used by *coordination manager* to determine, for example, which version of the *text_to_speech* service invoke when trying t reach a *voice, display*. thanks to the relatively easy integration of web services, has been possible to instantiate different versions of the *text_to_speech* service with multi-language text translation support.

Developing Ubiquitous Message System

The actual implementation of theUbiquitous Message System allows 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 that reflects the application cognitive model (figure 4.4).

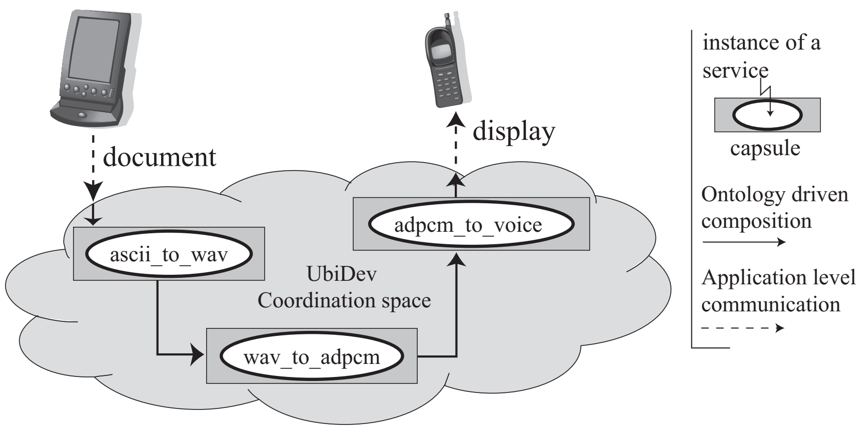

Figure 4.3: **Ubiquitous Message System.** *The application can compose capsules in a more structured entity, a composite high level service (document_to_display).*

- Code ontology using DAML-OIL (an extract is shown in figure3.5).
- Code the COCA *classifiers* that allow UBIDEV to tag resources with concepts of the *ontology*. Classifiers embed the knowledge required to tag a specific resource with one or more concepts of the ontology. the classification is based on both the analysis of the specific resource (i.e. extract all the information from a generic HTTP request that a web client sends when trying to contact the web-client sensing service) and the reuse of existing tools (i.e. all MIME types are "freely" classified by the MIMEType java class). The classifiers are strongly dependent on the application ontology since they supply the icons upon which the symbol system will be constructed [Har90]. For example a portable phone will be classified as instance of *voice, b/w, display* and *Resource*.
- Code the services and their description. Services will be automatically instantiated according to the actual configuration of the resources federation. The *sound_to_voice* service, for example, requires a *voice, display*; so when the *classifier* tags a resource as instance of *voice* and *display* concepts, the *service manager* will create a capsule containing the instantiated service and its EXE represented by the access reference of the tagged resource. The binding 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 context rules. These rules are expressed in Prolog predicates. Presently we have defined simple rules to verify weather a resource can be used by a user; thus the *coordination manager* has only to verify whether a service is entitled to access a resource on behalf of a specific user.
- Code the interfaces that will produce the input queries. The only query generated by the actual implementation of UMS is *document_to_display*. The *coordination 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 (figure4.3).

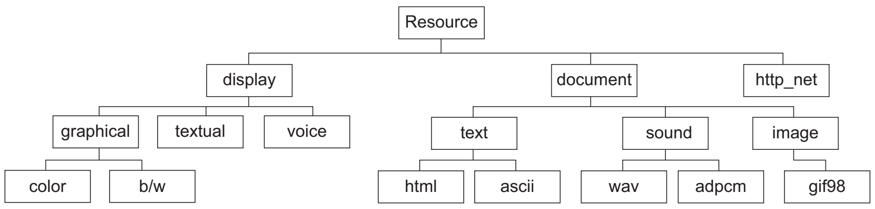

Figure 4.4: **Ubiquitous Message System.** *the ontology used in the prototype. "is_a" relation is used to relate concepts.*

Other important aspects related to the deployment of a Ubiquitous Message System like complex context adaptation and user interface, have been taken into account in the design phase but not yet in the implementation.

Configuration of the Environment

Ubiquitous Message System implements a bootstrap sequence that interprets a configuration file and starts the kernel services accordingly. The configuration file contains basic information about the UBIDEV core, such as the location of the services description, the name of the interface of the services, the location of the description of the ontology, of the model of the context and of the interfaces for the coordination module. UBIDEV makes the assumption that a pervasive computing system has a starting state that has been already configured. Currently, the administrator has to manually provide information about the initial state of the system. This approach has been followed only in the prototype for seek of simplicity; while it drastically reduce the flexibility and the autonomy of the system, it allows UBIDEV to skip the 0-level phase, where the system is unaware of it surrounding environment.

Starting the Ubiquitous Message System

We configure this application so it automatically starts the monitor interface with the predefined list of the devices contained in the environment. During the boot process, UBIDEV interprets the model of the context specification, then it evaluates the ontology and, finally, it generates the first resource change event. This boot-strap event let UBIDEV begin its working loop starting from the classification of the existing resources. The current implementation allows to manually specifying new resources included into the environment.

A user entering the room with his PDA, points the PDA towards the IrDA port and gets authenticated. New instances of UBIDEV services are instantiated using the newly classified resource as execution environment (specifically *ascii_to_graphical* and *gif98_to_color* services). When the user wants to send a text to another user, he just types the text on the PDA through a very simple interface specifying the text and the user name. Through the IrDA port, the text is transferred to the *text_to_wav* and to the *wav_to_adpcm* services reaching the fixed phone near the destination user who receive a phone call spelling the text typed.

4.2 Summary

We have introduced the design and the first implementation stage of a unified user-to-user communication system based on the functionalities offered by the UBIDEV middleware layer for messaging. This work illustrates the feasibility of developing of such a middleware layer to allow autonomous and adaptative integration of existing messaging services. The encapsulation mechanism of a UBIDEV service simplifies the integration of external web services; that makes quite easy to extend the system with a richer set of services as for the case of the *text_to_speech* service. The analysis of the model of the context as well as the evaluation of contextual rules that steer the service composition policy, allow UMS to be classified as a context-aware messaging system.

Compared to a messaging service offered by a collection of existing messaging systems and connecting gateways, our approach does not suffer from common denominator restrictions and frees the users from having to deal with differences between the individual underlying messaging systems.

Part III

Discussions

Chapter 5

Connected Projects at PAI group

This chapter presents other research projects that have been developed within the PAI group[1] and have a strong connection with the UBIDEV project. Some of them like the COCA model and Focale have strongly influenced the reference model; others like XCM and CB-SeC have been taken into account at integration level. The UBIDEV architecture has been implemented to allow the integration of these projects, in particular the context infrastructure and the structured coordination model, to form a fully functional infrastructure for pervasive computing environments.

5.1 WELCOME

The WELCOME[2] project [wel04] aims at constituting a conceptual and methodological know-how related to the design of Intelligent Networks with an immersive interaction strategy. The considered networks include Internet information and communication services on the one hand, and the various range of devices gathered under the umbrella of the pervasive computing domain on the other hand. The project has been structured around two poles: (1) Network Infrastructure, where the PAI group tackles with different levels of software and artificial entity management, and (2) Immersion where the group addresses the opening of networks onto users and their physical environment.

Orthogonally with these two thematic poles, WELCOME revealed to be organized along a second axis characterizing the system dynamicity, which is of course associated with an increasing problem complexity. The origin of that axis corresponds to Interconnection, that is, the component assembling operation producing functional initialization, while beyond that origin begins the area of adaptation, that is, the process allowing interconnected components to redefine dynamically their functionality regarding individual or collective objectives. Two kinds of adaptation processes may be distinguished: those purely adaptive, acting only at the functional level (functional adaptation), and the evolution processes, allowing a structural renewal of a system through new component generation and assembling (structural adaptation).

[1]www.diuf.unifr.ch/pai

[2]Originally, an acronym from We Enable Large-scale Computing, Operating, Mining and Exchange. SNF grants 2000.53930.98 and 2000-057279.99.

During the different phases of the WELCOME project, a number of subprojects were developed to meet the requirements of the two individual thematic poles, while UBIDEV takes place between the two poles.

5.2 COCA

The COCA model faces the software configuration issue in a heterogeneous and dynamic environment, by transforming heterogeneous resources through a process called classification into a semantic enriched conceptual representation. This enables high-level manipulation and configuration by pervasive computing applications. The COCA model introduces seven elements helping in the design and implementation of systems for heterogeneous and dynamic environments. These elements are:

- Resources as the root abstraction of COCA and the atomic unit that can be handled.
- Contexts are the containers resources live in. They define resource access and naming.
- Classifiers are an approach to face the symbol grounding problem and provide the basic level of semantic abstraction by associating concepts with resources.
- Concepts are the semantic abstractions grounded by classifiers and used to construct ontologies.
- Ontologies consist of interrelated concepts and provide higher level semantics together with an inference mechanism.
- Relations are used to represent higher level semantics in ontologies.
- Actions capture the dynamic aspects of the model by transforming resources from one concept to another.

Applying COCA allows automatic configuration of software items and addressing by concept, which in turn enables the construction of fault tolerant systems. COCA not only helps in system design, it also accommodates important programming paradigms such as object oriented, functional, and logic programming. Autonomous agent systems and the entity relationship model have also a mapping to COCA resulting in a wide coverage of important concepts currently found in computer science.

The key to the implementation of such systems is the ability for multiple versions of services to interact in a meaningful manner. Therefore a software configuration technique has to be found to provide this interaction. Because of the changing nature of the application domain, services must be found on other criteria than names. COCA optimizes the selection of resources for a user request depending on user preferences, cost and other parameters.

COCA resides in the core of UBIDEV as they have been conceived in parallel and in synergy [SMTH00]. COCA provides the mechanism for classification and addressing by concept that represents one of the steps UBIDEV takes in abstracting the underlying physical environment for the coordination module.

5.3 XCM/UCM

XCM is a generic coordination model intended aimed to supply abstractions and mechanisms for the effective support of communication, synchronization, and cooperation of distributed applications [TCH05].

As a coordination model, XCM comes within [ACH98]'s approach, and the vision of coordination proposed by [MC94], while prolonging an experience of coordination platform development previously carried on [Sch01]. Within this approach, XCM adds on a theoretical component inspired by autopoiesis i.e. the modeling of living systems elaborated in [VM80]. The interest of this heritage is double: 1) it allows profiting from the specificity of the physical space for modeling mechanisms like the construction and the maintenance of organism frontiers; 2) it introduces a fundamental distinction between organization (domain of control expression) and structure (domain of entity existence).

The model is organized in terms of entities, environment, ports and social laws. Entities represent an organizational abstraction of the coordinable elements. Every entity, except the universe, exists as a component of another entity called its environment. Environment defines the structure and the dynamism of the entities it contains. The social laws associated to each environment determine the interactions between entities embodied; they rule out the assembling and disassembling of an entity with the other components. Finally a port is a special type of entity dedicated to communication between entities. The coupling between an entity and a port is obtained through a special type of composition called interface, which is specified by the social laws that exists in the environment that embodies them.

UCM (Ubiquitous Coordination Model), as instantiation of XCM is a coordination model for pervasive computing environments. It instantiate the generic components of the XCM into a functional system related to the coordination module of UBIDEV:

- **Entity / agent**: the generic concepts in XCM naturally correspond to the basic notion of capsule provided as interface by the service layer in UBIDEV. Capsules are mapped to XCM atomic entities.

- **Environment**: it is an execution environment, like the atomic entity's one corresponding to capsules, or the user context.

- **Port**: it is a capsule input/output port in UBIDEV. It captures the service interface.

- **Social laws**: they are matching rules specifying how to combine capsules in UBIDEV, in order to form higher level structures through service composition.

For the Ubiquitous Message System example considered in chapter 4, this would then give us three UCM entities: ascii_to_wav, wav_to_voice, voice_to_phone, with the ports: ascii_in, wav_in and out_wav, voice_in and out_voice, and out_phone. The environment of these entities is the user's context, and its social rules specify that entity composition is obtained through coupling of same type in and out ports. An example is given in figure 5.1, were the broadcast functionality is obtained at coordination level by bridging the communication ports of different entities. It's up to *coordination manager* to apply the policy for connecting the ports as specified by the UCM specification language.

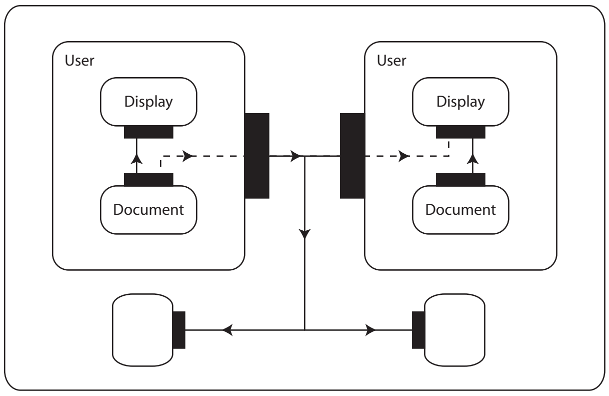

Figure 5.1: **UMS.** *Broadcast service as model in UCM.*

5.4 CB-SeC

CB-SeC (Context-Based Service Composition) is a generic context-based service discovery and composition framework. The approach adopted in this framework is the deployment of intelligent agents, which collectively gather the users context, in order to provide better user- tailored composite services. Agents have the advantage of being able to assist users, to discover and compose services. Agents are also mobile and are able to deploy functions on the fly. This helps with fast and autonomous adaptation to context changes. The context awareness considers the user context, the computing context, the time context, and the context history. The main characteristics of the CB-SeC framework are flexibility and adaptability. Flexibility is provided by the layered architecture, allowing tasks and functionalities to be distributed on the different layers of the model. Each layer has certain functions to fulfill, and keeps the processing details in its own layer. Other layers receive only the results they need. Adaptability is achieved by the context awareness mechanism embedded in the system.

The context parameters considered represent only a slice of all possible elements that make up a full definition of context, but nonetheless potentially useful for CB-SeC service discovery and composition framework.

1. **User context**: role, identity, location, preferences, social situation, permission profile, etc.

2. **Computing context**: network connectivity and nearby resources (printers, displays, and workstations), etc.

3. **Time**: time of a day, week, month, etc.

4. **Context history**: more importantly, when the user, computing, and physical contexts are recorded across a time span, we obtain a context history, which is also useful when discovering and composing service.

The CB-SeC underlines, more than UBIDEV, the relevance of the integration of contextual information in the service discovery, selection and composition processes. UBIDEV relay

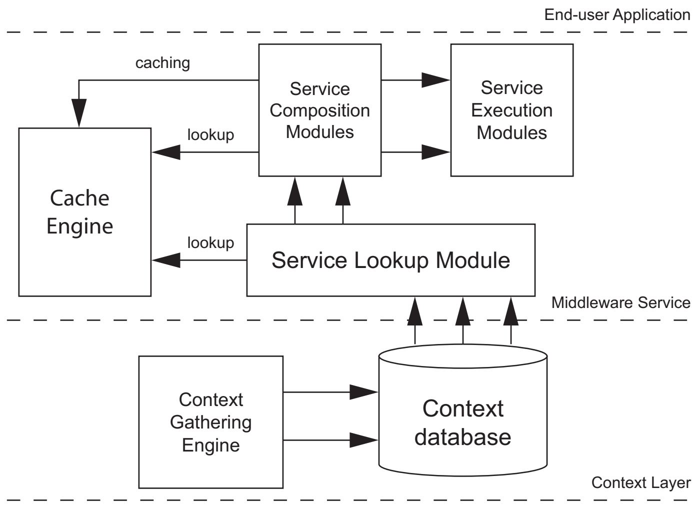

Figure 5.2: **CB-SeC.** *Three layer architecture.*

on the CB-SeC approach in structuring the context-aware infrastructure; the contextual information gathering and store phases are similar to those defined in the UBIDEV reference model since they have been conceived to interoperate with each other.

5.5 Focale

The FOCALE project aims providing interfacing schemes and concepts supporting a full-dynamical interaction mechanism with a priori unknown applications running onto the network [PC99].

The idea consists in externalizing the application interface, into a pluggable and adjustable device called virtual instrument (VI), inspired by the physical instrument metaphor.

At one side, it captures and emits information fluxes from and towards the application. At the other side, it is typically connected with input/output devices, which are directly accessible to the user, like graphical displays, audio devices, or any kind of sensor- motor device coupled to the user's body.

The user customizes a virtual instrument in order to choose which objects he wants to interact with, what part of these objects he wants to observe or change, when the interconnection must be effective, etc.

The model works under minimal assumptions about the application except an object-oriented design, and the delivering of a declarative interface. Virtual instruments implementation uses a mirroring technique in order to generate minimal disturbance on the application behavior. The model has been confronted to several applications, like object following in an intelligent building, or remote cooperation, and validated through several implementations, namely related to the artificial biology application field.

The computing approach, which is based on the paradigms of autonomous agent and situated intelligence, marks a deep renewal in the way of considering interaction. This approach shifts Human Computer Interaction (HCI) from the client-server paradigm to the ecosystemic partnership paradigm [PC00].

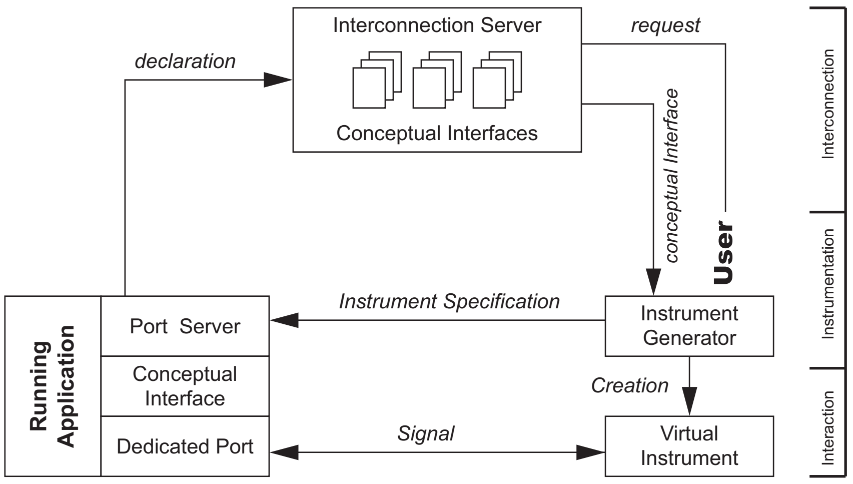

Figure 5.3: **Focale** *defines three functional levels: (1) interconnection, where the interconnection server is used, (2) instrumentation, aiming at the user interface building and adaptation, and (3) interaction, where the final exchanges between user and application actually take place.*

Indeed, in order to interact with an evolutive computing system, new sort of interaction tools allowing observing and to control this system, but also to discover and to explore it, are necessary.

This project is situated at the application level of UBIDEV. Focale could work as the interface module for any UBIDEV application; the VI would contain the modules that allow a user to manipulate the application by generating the input stimuli that would perturbate the system state inducing a reaction. The result of such a reaction could be sent over the VI or could perturbate another resource in the environment.

The interaction dedicated components of the application are respectively the declarative interface (conceptual interface), and two kinds of operational interfaces: the port server, in charge of the meta-interaction with the interconnection server and the instrument generator, and the dedicated ports, modeling the information flux handling with the user-dedicated virtual instruments.

Chapter 6

Related Works

This chapter analyses approaches of existing middlewares both academic and commercial that have been used in different pervasive computing projects. The presented middlewares have been selected because of their peculiar approach in facing heterogeneity at different level of the pervasive computing system. they will be described and compared with UBIDEV. Middlewares that sustain service-oriented approach, typically, aim to facilitate the rapid creation and deployment of services, while also offering dynamic service discovery, including the ability for clients to learn the capabilities of services. Some platforms also address issues of scalability and adaptation. Another approach of pervasive computing middlewares is the decoupling of users from devices and viewing applications as entities that perform tasks on behalf of users.

6.1 Gaia

Gaia OS [RHC$^+$02] is a meta-operating system that aims at supporting the development and execution of portable applications for active spaces. Gaia is a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space. Gaia exports services to query and utilize existing resources, to access and use current context, and provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive, and mobile applications. The system is built as a distributed object system. The Gaia Kernel, the Gaia Application Framework, and the Applications Active Space form the building blocks of the whole architecture. An Event Manager is a mechanism to expose changes of the environment through a publish subscriber model. The context infrastructure consists of a number of components, called context providers that provide information about the current context. Gaia implements a bootstrap protocol that interprets a configuration file (Lua script) and starts the kernel services. The configuration file contains information about the Gaia Kernel services, such as the name of the service, the name of the interface of the service, the Gaia node or nodes that will host the service, the service instantiation policy (i.e., instantiate the service in all Gaia nodes or only in the first available Gaia node), and start parameters.

This approach works fine when the topology of the active space is well known a priori. Fixed categorization of resources implies that the relation between hardware and software level is, in a sort of way, pre-defined. That makes reuse of existing resources as well as the introduction of new ones a difficult task.

Gaia makes use of ontology at different levels [MRMC03], as for the classification of the context and the semantic service discovery. UBIDEV shares the same philosophy in the use of ontology to augment the action and reaction of a Ubiquitous Computing system. Differently from Gaia, UBIDEV relies on the assumption that it is the application that determines how a specific environment should be configured and managed in terms of resources, services and context.

6.2 Aura

The Aura project [SG02] is about distraction-free pervasive computing. It supports mobile users inside a computing environment by maximizing the use of available resources inside the environment and by minimizing the user distraction and focus on user attention. Its goal is to provide each user with an invisible halo of computing and information services that persists regardless the location. It defines the notion of personal aura which can be considered as a service proxy for the mobile user it represents.

Aura proposes a programming model for task-based computing [WG00]. In this model, tasks are viewed as compositions of services. Both tasks and services have explicit representations. Services are described by virtual service types, which define functional, state and configuration interfaces and dependencies upon other services. Virtual service types can be related through inheritance, and can also be composed to form new virtual services. Tasks are toplevel compositions of services that are specified as flows that decompose tasks into steps of subtasks or primitives (actions carried out by services). Tasks are instantiated by a protocol that is responsible for gathering information about available services, selecting suitable services to carry out tasks and binding them together, and, finally, performing configuration and initialization of services. A coordination protocol manages the plugging and unplugging of services in response to resource changes. Tasks are also managed by a third protocol responsible for task migration, obtaining consistent snapshots of task state, and managing replication and consistency.

Aura shares lot of similarities with the UBIDEV project; both of them, in fact, take a holistic approach in the model of the environment. The definition of abstract services and the idea of mapping them into concrete instances are similar to the service classification model in UBIDEV. Also the Task Manager, as the coordination manager in UBIDEV, acts as a control unit for resources, services and context. Differently from UBIDEV, abstract services are mapped into concrete instances by analyzing service description. In Aura the whole process relies on the assumption that suppliers of a given service type share a vocabulary of tags and the corresponding interpretation.

Aura however lacks in defining at Prism level a proper model of the context. In UBIDEV the coordination manager has a complete and consistent model of the environment in terms of resources, services, context and their dependences. This approach allows a more easy writing of context dependent rules that may drive the application behavior.

6.3 Jini and JavaSpace

Jini [Jin03] is a distributed system based on Java. It offers a service model based on three components: an infrastructure for federating services in a distributed environment, a programming model for distributed services, and a set of system services, including a lookup service used by clients to locate required services and to dynamically access them through the use of Java RMI stubs.

However, Jini does not address the management of component-based applications and inter-component dependence. It only provides static look-up (exact matching) of services and does not consider the run-time resource constraints for small clients. Also, the large memory requirements imposed by Jini makes it not viable for most mobile devices. In addition, Jini announces service using UDP multicast by default, which may be suitable only in LAN-based application, but may not be applicable for large-scale deployment such as the Internet.

In the Jini architecture, service functionalities and capabilities are described in terms of Java object interface types. Service capability matching is processed in the object-level and syntax-level only. For instance, the generic Jini Lookup and other discovery protocols allow a service client to find a printing service that supports color printing, but the protocols are not powerful enough to find a geographically closest printing service that has the shortest print queue. The protocols do exact semantic matching while finding a service. Thus they lack the power to give a ”close match” even if it was available.

Jini is independent of the platform and operating system to run on. Most important, Jini uses Java Remote Method Invocation (Java RMI) protocols to move program code around the network. This introduces the possibility to move device drivers to client applications, which is its main advantage over the nonJava based service discovery concepts. On the other hand, the fact that Jini is tightly tied to the programming language Java makes it dependent on the programming environment. It also requires its devices to run a JVM, which consumes memory and processing power. This can be a hard requirement for large device drivers and might not be fulfilled in embedded systems. Due to the dynamic nature of ad hoc networks, Jini employs the concept of leasing. Each time a device joins the network and its services become available on the network, it registers itself only for a certain period of time, called a lease. This is especially useful for very dynamic ad hoc network scenarios.

An important part of the Jini system is formed by a coordination model for generative communication called JavaSpace. A JavaSpace is a Linda-like coordination system that stores tuples representing a typed set of references to Java objects. Multiple JavaSpaces may coexist in a single Jini system. Jini does provide a specialized look-up service that allows clients to look up registered services using an attribute-based search facility. Each service has an associated service identifier, which is a globally unique 128-bit value generated by the lookup service. A service uses this identifier to register a service item at the lookup service. A client can provide a template tuple when looking for specific tuple instances. The lookup service will select only those tuples that match the template.

Jini implicitly forces the use of ad-hoc ontology by ensuring that client queries contain interfaces. The lack of a well-defined ontology for service descriptions could result in false matching. These protocols do not solve the problem of making service discovery more flexible and powerful

In the existing Jini architecture, ontologies are captured in the level of Java object interface types. UBIDEV has not been conceived to be an Open Distributed System as Jini so it does not really address service description and discovery from a fully open scenario. For this reason, using an ad-hoc representation of the environment through application ontology is not considered to be a limitation of the system. Moreover UBIDEV allows addressing resources and context making complex semantic queries that allows the application to be described in terms of its functional aspects instead of its possible interactions.

The scenario addressed by Jini technology [Wal99], spontaneous networking and service delivery, is very close to a pervasive computing system where services should be realized and adapted on the fly. Since Jini is entirely Java based, the environment is homogeneous as a precondition and only a few additional abstractions are necessary. As a result, Jini has a very flat structure mainly consisting of services and some support libraries providing remote method invocation (RMI), security, leasing, transactions and distributed events. Jini does not distinguish between physical devices and services implemented as software only. Nevertheless, the term "Jini-enabled device" comes close to a virtual representer running one service. Service discovery and lookup takes place through a set of protocols in conjunction with a *lookup service*. Jini automatically discovers services as they appear on the net and registers them with the lookup service. Unlike classifiers that provide a semantic based lookup through addressing by concept, Jini limits lookup to Java interfaces and attributes provided by the service. Service composition lies entirely in the hands of the application or is done interactively by the user. The Rio [Mic01] architecture provides a higher level programming model based on Jini for application development with a focus on quality of service, dynamic deployment and fault tolerance.

6.4 Semantic Service Discovery

Semantic Service Discovery project [AJF02] follows within the research activities of the UMBC eBiquity Research Group on the second generation Semantic Web. SSD enhances the Bluetooth SDP matching mechanism to use semantic information associated with services in the process of service look-up and selection. This includes priorities, expected values of service attributes, and some index of a match's closeness. To support this matching mechanism and allow more efficient service discovery, SSD introduces a service ontology described in a semantic language and a Prolog-based reasoning engine that uses the ontology. Traditionally, the assumption is that sophisticated matching mechanisms impose heavy computational and memory burdens that only server class systems can handle.

Describing services ontologically is superior to UUID-based descriptions because it provides a structure for reasoning about and deriving knowledge from the given descriptions. Ontology also describes relationships between different entities in the system more clearly. Further, the ontology can facilitate inexact matching by specifying rules and constraints on attributes –for example, by imposing a priority on each option. SSD uses the semantically rich DAML+OIL Darpa Agent Markup Language and Ontology Inference Layer –to describe the ontology. Using DAML+OIL as a description language offers two main advantages. First, it's easier to use the syntax and rules of an existing language than to create a new one. Second, because DAML+OIL is becoming a standard for use in the semantic Web, services developed for the Internet using this language can easily be installed on Bluetooth-enabled devices and disseminated in Bluetooth networks.

The key to correct functioning of SSD reasoning engine is a knowledge base with sufficient and complete information about service instances. Large amounts of factual data must be loaded into the knowledge base before the program can use them. XSB, then, extracts the relationships that DAML describes and uses them to arrive at a solution to a given service discovery query. The engine first performs pattern matching to determine whether it can answer the query directly. Upon failure, it evaluates other possible solutions, which match the requested query attribute values within an error range based on the specified closeness index. SSD is a project that aims to face the semantic interoperability issue in a world wide web scale using an ontology-based approach. Similarly to other existing projects, SSD focus on the technology that would allow a semantic matching mechanism to work. It proposes a common ontology that would be shared among all applications in order to allow services to be located and selected. It does not provide any mechanism for ontology decomposition or mapping between sub-ontologies. This project has been very inspiring for UBIDEV since their reasoning engine based on XSB.

6.5 one.world

One.world [Gri04] architecture is based on three main abstractions: Tasks that represent computations, tuples that represent persistent data, and environments that provides structure and control.

Tasks execute code using multiple threads, and have their own, private state. Tuples are immutable records and are strongly typed in that both their fields and the tuples themselves are typed [CG89]. Tasks can request to be notified when (specific) tuples are added or written. Environments provide structure for computations and data by encapsulating tasks, tuples, and other environments. Tasks execute within an environment and, by default, access only tuples in the same environment. But, in order to enable remote interaction, tasks can explicitly request to access tuples in other environments. Besides the obvious create and destroy operations, the basic operations on environments are move, which moves an environment and all its contents either within a local hierarchy or to a different location, and open, which moves all contents of an environment into the enclosing environment and deletes the emptied environment.

Such environments can be used to provide functionality similar to that of distributed virtual machines [BDF$^+$99] within one.world architecture by enforcing a uniform policy across all nodes within an organization and by providing a single point of administrative control.

As for tuple spaces, one.world uses templates to describe tasks, tuples, environments, as well as external resources, and, if several resources match a specified template, any of the matching resources can be selected. Generally, a resource matches a template if its type is a subtype of the template and if all fields specified by the template match the corresponding fields of the resource.

Project one.world [GABW00] promotes a new application structure designed to cope with frequent changes in pervasive computing environment. Solar could be a complementary system used by one.world applications to detect the contextual changes

University of Washington's One.World Project provides an integrated framework for building pervasive applications. One.World allows dynamic decomposition of applications into components and it separates the functionalities and data. We adopted the same approach on the separation of the functionalities and data. However, our facet is a Java-based component as we believe Java programming language is most platform-independent and is more portable and less complex in terms of engineering effort. Similar to the Ninja project, One.world did not address code mobility. A client-server model is adopted for obtaining Web services.

6.6 Ninja

Berkeley Ninja [GvBB$^+$01] is a framework for dynamically composable wide-area services based on strongly typed reusable components. Ninja follows a dataflow-computing model. A group of dispersed services are identified and chained to form a path based on some resource demands. Client users are then able to obtain the required service by flowing data through the path. Ninja does not fully address code mobility. Mobile code is used only to instantiate the path. Also, all its reusable service components are not able to migrate. On the contrary, our design enables the dynamic loading of codes to client devices without moving client data for remote processing, unless the client is unable to handle large computations locally.

Although Ninja addresses the broad range of distributed Internet services, some of the problems to solve are very similar to those found in pervasive computing. Specifically, Ninja has to deal with issues such as heterogeneity, service description, naming and service composition. Ninja uses a layered approach as well to leverage the inherent heterogeneity. The basic building blocks of Ninja are *units* which are devices and sensors, interacting with the physical environment and users, *active proxies* which act as an adapter between units and services and *services* which run on *bases* that are clusters of workstations. Active proxies combine the roles of adapters and virtual representers. Active proxies are different insofar that usually multiple units are handled by one proxy and therefore do not exactly mirror the physical environment in the logical environment as virtual representers do. Service description and discovery is handled by a distributed directory and lack the semantic introduced by classifiers. Because Ninja services run on powerful clusters of workstations, resource requirements are a minor issue. In consequence, Ninja limits service description to their programming interface. For service development, Ninja provides design patterns and distributed data structures as well as a security infrastructure. A powerful service composition technique called *path* allows explicit and implicit (automatic) building of high-level services through combination of basic services. The current Ninja architecture has been designed to address many of the requirements of pervasive computing, many other issues remain beyond their scopes, including context-awareness and user and user-interface issues.

6.7 E-speak

The e-speak platform [Lab00] aims at the developments, deployment, management and intelligent interaction of e-services. As for Ninja, the environment is the Internet but similar problems are found in an interactive environment too. E-speak does not represent the physical dimension of the environment and deals with heterogeneity through a common protocol to all e-services and an XML-based service description. Service description is well separated from the service contract (the programming interface) and provides a similar degree of abstraction as ontologies thanks to a per-service vocabulary. Search recipes that contain a resource description, a lookup method, predicates and multiple-match policy information provide a powerful mechanism for resource lookup going further than addressing by concept. In addition, e-speak provides discovery of services by a registration process, negotiation to narrow down the set of service matching a request, introspection of e-services and mediation of service access enabling monitoring and dynamic reconfiguration and service composition. The main difference between UBIDEV and e-speak beside the physical dimension is, that e-speak does not include the service requirements in the service description. This implies that an appropriate execution environment is always available for an e-service given and limits therefore the possible actions the runtime system can take during service execution.

Chapter 7

Conclusions

In this dissertation, we have explored the impact that heterogeneity may have on pervasive computing systems.

Heterogeneity Problem

Pervasive computing presents an attractive vision for the future of distributed computing, where devices are ubiquitous and seamlessly coordinated in order to help people in accomplishing their tasks.

However, existing approaches to building distributed applications fall short in realizing this vision. The problem is that they try to hide heterogeneity at all levels and rely on technologies, such as remote procedure call packages, that extend single-node programming methodologies to distributed systems.

Applications built on top of these technologies tend to be structured like single-node applications and assume an execution environment where resources and services are homogeneous and continuously available.

As a result, users must explicitly reconfigure their devices and applications every time the execution environment changes, which is tedious at best and antithetical to the vision of pervasive computing at worst.

How the Problem Has Been Addressed

To face the heterogeneity problem in pervasive computing we have introduced an architecture for supporting the designing, building, execution of applications that relay on semantic-based and application-centered management of resources, services and context.

Following this approach, system support allows a certain level of visibility of the heterogeneity of the underlying environment and, at the same time, allows applications to explicitly provide their own semantics of the environment.

The overall abstraction is realized at the coordination level where the system is described in terms of autonomous and uniform interacting services.

That way, applications can see resources, services and contextual changes in a uniform way and then adapt to it instead of forcing users to constantly reconfigure their systems.

We have presented UBIDEV, a system architecture for pervasive computing, that embodies this approach to building pervasive applications. The architecture supports a simple design process for building applications, starting from the definition of the system ontology, the pool of classifiers that will tag resources and contextual information as instances of the concepts composing the ontology, simple rules to express the interrelation between concepts of the ontology that will be used by the coordination control unit to solve the user's tasks.

By taking a bio-inspired approach, the reference model as been defined as an autopoietic system where the stimuli abstractions describe the interactions that happen within an environment.

UBIDEV builds on five foundation services:

- Resources are virtualized within the environment through a representer granting a uniform access interface.
- Resources, services and contextual information are classified and addressed according to a system ontology to allow a uniform abstraction of the whole environment.
- A coordination unit is responsible of solving user's tasks through composition of homogeneous services.
- At coordination level capsules, context model as well as contextual information are described according to the agreed ontology.
- Environments host applications, store persistent data, and, through nesting, facilitate the composition of applications and services.

On top of these services, UBIDEV provides a set of system services that address common application needs, including discovery and access resources.

Main contributions

This dissertation has made the following contributions:

- **Resources**: the notion of representer as a virtualization of a physical resource, allows a system to have a full representation of the physical environment in terms of involved resources. In such a perspective, resources are seen as interacting entities that can both produce and receive stimuli from the control systems and from other entities. Classification of resources according to the system ontology allow a uniform description and configuration as well as a simple access protocol based on "addressing by concepts".
- **Services**: UBIDEV introduces the notion of service prerequirements as a collection of system properties that have to be met for ensuring the executability of a given service. By matching such prerequirements with the current available resources, UBIDEV builds a proper execution environment for each service. Capsules are instantiated to frame the notion of a running service together with its execution environment. Capsules exports at coordination level only service interfaces.
- **Coordination**: is a control unit where all information from resources, context, services, users and application are orchestrated. The coordination module is in charge of solving user's tasks by composing existing services; in doing so it analyses contextual information to determine which services to invoke and under which conditions.

- **Holistic Management**: in UbiDev there is the explicit notion of system as a physical environment, a service infrastructure and an application. Resources, services and context are managed as part of a common space where the same semantic is used. Users are described as resources belonging to the environment; they can interact directly with other resources and could be virtualized within the system by their role, identity and intentions.

Main differences from Other Approaches

In chapter 6 we have presented different projects that address the same class of problems UbiDev does with similar approaches. The main differences from such approaches are:

- In UbiDev the semantics of the environment is left to the application by a system reference ontology. Then the classification of the environment is done according to such an ontology. That means that at application level, resources, services and contextual information are described and configured in a uniform way that application expects.

- UbiDev focuses on the holistic management of resources, services and contextual information rather than describing, configuring and addressing them separately.

- UbiDev allows a system to be described in terms of interacting homogeneous entities while exposing at coordination level the properties of the environment that may influence the whole system.

Future Works

UbiDev has been the first concrete research project targeting pervasive computing scenarios within the PAI group. Under the umbrella of the WELCOME SNF project, UbiDev tried to profit from the experience of past researches and to position itself as a glue for all new coming projects defined in similar research areas.

The implemented architecture and the Proof of Concepts have to be considered as development exercises aimed to demonstrate the feasibility of the whole approach.

UbiDev can be considered as a starting point for further exploring research projects in similar and complementary areas. For such reasons UbiDev leaves many open directions:

- Application interfaces as well as base services needs to be deployed and migrate over a wide spectrum of different devices; that implies support of code migration at the infrastructure level. Migration needs to be visible to applications rather than being transparent, so that applications know their execution context and can adapt to it. There is the need to integrate persistent storage, so that applications can continuously provide access to peoples information. Must be easy to control, so that migration logic can be effectively factored from the rest of the application functionality and devices are protected against malicious applications roaming around the network. Code migration needs to perform well enough to match peoples movements in the physical world.

- Interoperability between systems is still an open issue. The use of ontology to describe the taxonomy and the semantics of a system frames the interoperability at ontology mapping level. Ontology mapping has been widely studied in semantic web. Most ontology mapping tools developed seeks to find a one-to-one corresponding mapping between concepts in two ontologies [SLD05] [DMDH02]. These mapping tools can be classified into two types: source-based and instance-based. Source-based mapping tools compare the similarity of the concepts based on the properties of the concepts and the structure of the ontology defined in the source ontologies. Examples of source based mapping tools are PROMPT and Chimaera [MFRW00]. Instance-based ontology mapping tools compare the similarity of the concepts based on the source ontologies and their data instances. Examples of instance-based ontology mapping tools are FCA-Merge and GLUE [CFJ04].

- Another issue related to the use of ontology is the intrinsic difficulty of defining the ontology that would describe all possible events that would occur within a system. Experience has shown that the system ontology is too critical to describe from scratch. For such a reason, ontology-templates have been defined especially in semantic web area. COCA classifiers would help the integration of sub-ontologies thanks to the real-time classification of the environment. Dedicated classifiers could be defined to ensure the coherence of the description of all resources according to a given sub-set of the system ontology.

- A fully functional infrastructure requires a minimum support for security and trust; something that has been neglected in the UBIDEV project. What is required at infrastructure level is a trust management support for user authentication, access control, and delegation; assign security credentials to individuals; allow entities to modify access rights of other entities by delegating or deferring their access rights and provide resource access control on the base of system policies and users identities and credentials [KFJ01].

- At coordination level it would be interesting to further investigate the impact and the potentialities of a fine grained control unit able to take into account environmental issues like contextual situations, user's virtual identities and roles as well as resources access rights. The integration of XCM coordination model is a step in such direction.

Bibliography

[Abo99] G. D. Abowd. Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment. *IBM Systems Journal*, 38:50–53, 1999.

[AC93] G. Agha and C. J. Callsen. ActorSpace: An Open Distributed Programming Paradigm. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, May 1993. Also published as a Special Issue of SIGPLAN Notices vol. 28, No. 7, pp 23-32, July, 1993.

[ACH98] F. Arbab, P. Ciancarini, and C. Hankin. Coordination Languages for Parallel Programming. *Parallel Computing*, 24(7):989–1004, 1998.

[ACH$^+$01] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper. Implementing a Sentient Computing System. *Computer*, 34(8):50–56, 2001.

[AJF02] S. Avancha, A. Joshi, and T. Finin. Enhanced Service Discovery in Bluetooth. *IEEE Computer*, 35(6):96–99, June 2002.

[AKDS01] G. D. Abowd A. K. Dey and D. Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction Journal, Special Issue on Context-Aware Computing, 16(1)*, 2001.

[BDF$^+$99] M. Beck, J. Dongarra, G. Fagg, G. Geist, P. Gray, J. Kohl, M. Migliardi, K. Moore, T. Moore, P. Papadopoulous, S. Scott, and V. Sunderam. HARNESS: A Next Generation Distributed Virtual Machine. *Future Generation Computer Systems*, 15(5-6):571–582, 1999.

[Bie02] G. Bieber. Introduction to Service-Oriented Programming. `http://www.openwings.org/download.html`, October 2002. Motorola ISD.

[BKA$^+$98] E. Brewer, R. H. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. D. Gribble, T. Hodes, G. Nguyen, V. N. Padmanabhan, M. Stemm, S. Seshan, and T. Henderson. A Network Architecture for Heterogeneous Mobile Computing. *IEEE Personal Communications Magazine*, 5(5):8–24, October 1998.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[blu] Bluetooth Specification. `https://www.bluetooth.org/foundry/adopters/document/Bluetooth_Core_Specification_v1.2`. version 1.2.

[Bro96] M. G. Brown. Supporting User Mobility. In *Proceedings of the IFIP Conference on Mobile Communications (IFIP'96)*, Canberra, Australia, September 1996.

[BZD02] 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.

[CA94] C. J. Callsen and G. A. Agha. Open Heterogeneous Computing in Actorspace. *Journal of Parallel and Distributed Computing*, 21(3):289–300, 1994.

[Car01] J. M. Carroll, editor. *Human-Computer Interaction in the New Millennium*. Addison-Wesley, 2001.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, March 2001. W3C Note.

[CDK00] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 3rd edition, 2000.

[CFJ03] H. Chen, T. Finin, and A. Joshi. Semantic Web in a Pervasive Context-Aware Architecture. In *Workshop on Artificial Intelligence in Mobile Systems, the Fifth Annual Conference on Ubiquitous Computing*, Seattle, 12-15 October 2003.

[CFJ04] H. Chen, T. Finin, and A. Joshi. Semantic Web in in the Context Broker Architecture. In *Proceedings of the Second Annual IEEE International Conference on Pervasive Computer and Communications*, pages 277–287, Washington, DC, USA, March 2004. IEEE Computer Society.

[CG89] N. Carriero and D. Gelernter. Linda in Context. *Communication of the ACM*, 32(4):444–458, April 1989.

[Che76] P. Chen. The Entity/Relationship Model: Toward a Unified View of Data. *ACM Transaction on Database Systems*, 1(1):9–36, March 1976.

[CK00] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR 2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[CK03] G. Chen and D. Kotz. Context-Sensitive Resource Discovery. In *First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 243–252, Fort Worth, Texas, 23-26 March 2003.

[Coe98] M. H. Coen. Design Principles for Intelligent Environments. In *Fifteenth National Conference on Artificial Intelligence*, pages 547–554, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[Cor05] Microsoft Corporation. Universal Plug and Play: Device Architecture Version 1.0. http://www.upnp.org/, May 2005.

[Cou] J. Coutaz, L. Nigay, and D. Salber. Agent-Based Architecture Modelling for Interactive Systems in: D. Benyon and P. Palanque (eds). Critical Issues in User Interface System Engineering, pages 191-209. Springer-Verlag. 1995.

[Cou87] J. Coutaz. PAC: An Object Oriented Model for Implementing User Interfaces. *ACM SIGCHI Bulletin*, 19(2):37–41, 1987.

[CS92] C. Catlett and L. Smarr. MetaComputing. *Communications of the ACM*, 35(6):44–52, 1992.

[CTB+95] J.R. Cooperstock, K. Tanikoshi, G. Beirne, T. Narine, and W. Buxton. Evolution of a Reactive Environment. In *Proceedings of the 1995 ACM Conference on Human Factors in Computing Systems (CHI '95)*, pages 170–177, Denver, CO, May 1995.

[dam] D. Connolly, F. van Harmelen, I. Horrocks, D. McGuinness, P. F. Patel-Schneider and L. Stein. DAML+OIL (March 2001) Reference Description, December 2001. W3C Note.

[DAW98] A. K. Dey, G. D. Abowd, and A. Wood. CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services. *Intelligent User Interfaces*, 11:47–54, 1998.

[dco] DCoM.

[Dey00] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, 2000.

[DFWB98] N. Davies, A. Friday, S. P. Wade, and G. S. Blair. An Asynchronous Distributed Systems Platform for Heterogeneous Environments. In *Proceedings of the 8th ACM SIGOPS European Workshop: Support for Composing Distributed Applications*, pages 66–73, Sintra, Portugal, 7-10 September 1998.

[diu] Diuf development library. `http://www.unifr.ch/diuf/pai/`.

[DMDH02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to Map Between Ontologies on the Semantic Web. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 662–673, New York, NY, USA, 2002. ACM Press.

[EHC+93] S. Elrod, G. Hall, R. Costanza, M. Dixon, and J. des Rivieres. Responsive Office Environments. *Communications of the ACM*, 36(77):84–85, July 1993.

[FIB95] G. Fitzmaurice, H. Ishii, and W. Buxton. Bricks: Laying the Foundations for Graspable User Interfaces. In *Proceedings of Human Factors in Computing Systems (CHI95)*, pages 442–449, Denver, Colorado, May 1995.

[FK97] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[FK03] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2nd edition, 2003.

[FKS97] S. Fickas, G. Kortuem, and Z. Segall. Software Organization for Dynamic and Adaptable Wearable Systems. In *Proceedings of the 1st International Symposium on Wearable Computers (ISWC'97)*, pages 56–63, Cambridge, MA, October 1997.

[FMS93] S. Feiner, B. MacIntyre, and D. Seligmann. Knowledge-Based Augmented Reality. *Communications of the ACM*, 36(7):52–62, 1993.

[FS99] J. Flinn and M. Satyanarayanan. Energy-aware Adaptation for Mobile Applications. In *Proceedings of the 17th ACM Symposium on Operating Systems and Principles*, Kiawah Island, December 1999.

[FT89] I. Foster and S. Taylor. *Strand: New Concepts in Parallel Programming*. Prentice-Hall, Englewood Cliffs, 1989.

[GABW00] R. Grimm, T. Anderson, B. Bershad, and David Wetherall. A System Architecture for Pervasive Computing. In *9th ACM SIGOPS European Workshop*, pages 177–182,, Kolding, Denmark, September 2000.

[GBK99] H. Gellersen, M. Beigl, and Holger Krull. The MediaCup: Awareness Technology Embedded in an Everyday Object. *Handheld and Ubiqutious Computing*, 1707:308–310, 1999.

[GCL$^+$99] Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright. *Simple Service Discovery Protocol*, June 1999. IETF Internet Draft.

[Gel85] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[GHMF01] M. Gudgin, M. Hadley, J. Moreau, and H. Frystyk. SOAP Version 1.2 - Part 1: Messaging Framework, 2001.

[GMGN04] X. Gu, A. Messer, I. Greenberg, and D. Milojicic K. Nahrstedt. Adaptive Offloading for Pervasive Computing. *IEEE Pervasive Computing*, 3(3):66–73, July-September 2004.

[Gri04] R. Grimm. One.world: Experiences with a Pervasive Computing Architecture. *IEEE Pervasive Computing*, 3(3):22–30, July-September 2004.

[gro02] OMG group. Common Object Request Broker Architecture: Core Specification. `http://www.omg.org/cgi-bin/doc?formal/02-11-01`, November 2002. Version 3.0.

[GSSS02] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing, special issue on "Integrated Pervasive Computing Environments"*, 21(2):22–31, April-June 2002.

[GTV05] M. Gutierrez, D. Thalmann, and F. Vexo. Semantic Virtual Environments with Adaptive Multimodal Interfaces. In *11th International Multimedia Modelling Conference (MMM'05)*, pages 277–283, Melbourne, Australia, January 2005.

[Gut00] E. Guttman. *Service Location Protocol Modifications for IPv6*, January 2000. IETF Internet Draft.

[GvBB+01] S. D. Gribble, M. W. Rob von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Zha. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Computer Networks*, 35(4):473–497, 2001.

[Har90] S. Harnad. The Symbol Grounding Problem. *Physica D*, 42:335–346, 1990.

[HNBR97] R. Hull, P. Neaves, and J. Bedford-Roberts. Towards Situated Computing. In *Proceedings of the 1st International Symposium on Wearable Computers (ISWC'97)*, pages 146–153, Cambridge, MA, October 1997.

[Inc99] The Salutation Consortium Inc. Salutation Architecture Specification (Part 1). http://www.salutation.org, 1999. Version 2.1.

[Jav03] JavaSpaces Service Specification. `http://wwws.sun.com/softwarejini/specs/`, June 2003.

[JF04] B. Johanson and A. Fox. Extending Tuplespaces for Coordination in Interactive Workspaces. *Journal of Systems and Software archive*, 69(3):243 – 266, January 2004. Special issue: Ubiquitous Computing.

[Jin03] Jini Architecture Specification. `http://wwws.sun.com/software/jini/specs/`, June 2003.

[Joh03] B. E. Johanson. *Application Coordination Infrastructure for Ubiquitous Computing Rooms*. PhD thesis, Stanford University, 2003.

[Kam00] A. Kaminsky. Jini Print Service Design. `http://print.jini.org/`, February 2000.

[KBM+02] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. People, Places, Things: Web Presence for the Real World. *Mobile Networks and Applications*, 7(5):365 – 376, October 2002. Kluwer Academic Publishers.

[KFJ01] L. Kagal, T. Finin, and A. Joshi. Trust-Based Security in Pervasive Computing Environments. *Computer*, 34(12):154–157, 2001.

[KFJ03] L. Kagal, T. Finin, and A. Joshi. A Policy Language for a Pervasive Computing Environment. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 63–75, Washington, DC, USA, 2003. IEEE Computer Society.

[KG99] D. Kotz and R. Gray. Mobile Code: the Future of the Internet. In *Third International Conference on Autonomous Agents*, Seattle, 1999.

[KP88] G. E. Krasner and S. T. Pope. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[KS03] C. Kray and M. Strohbach. Gesture-based Interface Reconfiguration. In *Workshop "AI in mobile systems" (AIMS 2003) at Ubicomp'03*, Seattle, WA, USA, October 2003.

[KUB00] P. Kropf, H. Unger, and G. Babin. WOS: an Internet Computing Environment. In *Workshop on Ubiquitous Computing, IEEE International Conference on Parallel Architecture and Compilation Techniques*, pages 14–22, Philadelphia, PA, October 2000.

[Lab00] HP Labs. E-speak Project. `www.hp.com/go/espeak`, 2000.

[LCBB00] C. Liebig, M. Cilia, M. Betz, and A. P. Buchmann. A Publish/Subscribe CORBA Persistent State Service Prototype. In *Middleware '00: IFIP/ACM International Conference on Distributed systems platforms*, pages 231–255. Springer-Verlag New York, Inc., 2000.

[LMW99] T. Lehman, S. McLaughry, and P. Wyckoff. TSpaces: The Next Wave. In *In Proceedings of the 32nd Hawaii International Conference on System Sciences (HICSS-32)*, January 1999.

[Lot00] Lotus. Unified Messaging Strategy and Mobile Services for Domino. Data sheet, 2000.

[LY99] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Sun Microsystems Inc., 1999.

[Mag01] General Magic. Portico. `http://www.genmagic.com/portico`, 2001.

[Mat05] F. Mattern. *Ubiquitous Computing: Scenarios from an informatised world*, pages 145–163. E-Merging Media - Communication and the Media Economy of the Future. Springer-Verlag, 2005.

[MC94] T.W. Malone and K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[McL89] L. McLaughlin. *A Standard for the Transmission of 802.2 Packets over IPX Networks*, November 1989. RFC: 1132.

[MFRW00] D. McGuiness, R. Fikes, J. Rice, and S. Wilder. An Environment for Merging and Testing Large Ontologies. In *Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, pages 483–493, Breckenridge, CO, USA, April 2000.

[MH02] 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.

[MH03] S. Kouadri M. and B. Hirsbrunner. Towards a Context Based Service Composition Framework. In *1st International Conference in Web Services, ICWS'03*, pages 42–45, Las Vegas, Nevada, USA, 23-26 June 2003.

[Mic01] Sun Microsystems. Rio Architecture Overview. `http://www.sun.com/software/jini/whitepapers/rio_architecture_overview.pdf`, March 2001.

[MPR01] A. Murphy, G. Picco, and G. C. Roman. Lime: A Middleware for Physical and Logical Mobility. In *Proceeding of the 21st International Conference on Distributed Computing Systems (ICDCS)*, pages 524–533, April 2001.

[MRMC03] R. E. McGrath, A. Ranganathan, M. D. Mickunas, and R. H. Campbell. Investigations of Semantic Interoperability in Ubiquitous Computing Environments. In *15th IASTED International Conference on Parallel And Distributed Computing And Systems (PDCS 2003)*, Seattle Marina del Rey, CA, USA, 3-5 November 2003.

[MS99] M. Migliardi and V. Sunderam. The Harness Metacomputing Framework. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio (TX), USA, 22-24 March 1999.

[MS03] F. Mattern and P. Sturm. From Distributed Systems to Ubiquitous Computing – The State of the Art, Trends, and Prospects of Future Networked Systems. In K. Irmscher and K. Fahnrich, editors, *Proceeding KIVS 2003*, pages 3–25, Springer-Verlag, February 2003.

[MV92] H. R. Maturana and F. J. Varella. *The Tree of Knowledge - the Biological Root of Human Understanding*. Addison-Wesley, Shambhala USA, 1992.

[NCN98] K. Nahrsted, H. Chu, and S. Narayan. QoS-aware Resource Management for Distributed Multimedia Application. *Journal on High-Speed Networking*, 7(3), 1998.

[Nob00] B. Noble. System Support for Mobile, Adaptive Applications. *IEEE Personal Communications*, 7(1), February 2000.

[Nor99] D. A. Normann. *The Invisible Computer*. MIT Pres, 1999.

[NSN$^+$97] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile Application-Aware Adaptation for Mobility. In *In Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France, October 1997.

[OPSS93] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus - an Architecture for Extensible Distributed Systems. In B. Liskov, editor, *Proceedings of the 14th Symposium on Operating Systems Principles*, pages 58–68, Asheville, NC, USA, December 1993. ACM Press.

[OWLa] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, 2003.

[OWLb] F. van Harmelen and J. Hendler and I. Horrocks and D. L. McGuinness and P. F. Patel-Schneider and L. Stein. Owl Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, 2002.

[OWLc] M. K. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide. http://www.w3.org/TR/owl-guide/, 2003.

[pai] Pervasive and Artificial Intelligence Research Group. `http://www.unifr.ch/diuf/pai/`.

[Par72] D.L. Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12), December 1972.

[PC99] S. Le Peutrec and M. Courant. Instruments Pour la Vie Artificielle. In *Proceedings of the Eleventh French-speaking Congress of Human Computer Interaction*, pages 37–40, Montpellier, France, November 1999.

[PC00] S. Le Peutrec and M. Courant. Revisiting HCI for Networking Computers: A First Breakthrough for Artificial Biology. In *Proceedings of the International ICSC Symposium on Biologically Inspired Systems (BIS 2000)*, pages 37–40, Wollongong, Australia, December 2000.

[PMR99] G. Picco, A. Murphy, and G. Roman. Lime: Linda Meets Mobility. In *Proceeding of the 21st International Conference on Software Engineering (ICSE '99)*, pages 368–377, Los Angeles, CA, USA, May 1999.

[PPL$^+$03] G. Pingali, C. Pinhanez, A. Levas, R. Kjeldsen, M. Podlaseck, H. Chen, and N. Sukaviriya. Steerable Interfaces for Pervasive Computing Spaces. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 315–322, Washington, DC, USA, March 2003. IEEE Computer Society.

[Pro81] Defense Advanced Research Projects Agency Internet Program. *Internet Protocol*, September 1981. RFC: 791.

[Pro97] Defense Advanced Research Projects Agency Internet Program. *Basic Socket Interface Extensions for IPv6*, April 1997. RFC: 2133.

[RAH98] J. Rekimoto, Y. Ayatsuka, and K. Hayashi. Augment-able Reality: Situated Communication Through Physical and Digital Spaces. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers (ISWC'98)*, pages 68–75, Pittsburgh, PA, October 1998.

[RCC98] A. Robert, F. Chantemargue, and M. Courant. Emuds: Grounding Agents in EMud Artificial Worlds. In *Proceedings of the First International Conference on Virtual Worlds, VW'98*, Paris, France, 1-3 July 1998.

[Rek98] J. Rekimoto. A Multiple Device Approach for Supporting Whiteboard-Based Interactions. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, pages 344–351, Los Angeles, CA USA, April 1998.

[RHC$^+$02] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, Oct-Dec 2002.

[Ric03] T. Richardson. The RFB Protocol. `http://www.realvnc.com/docs/rfbproto.pdf`, August 2003. Version 3.7, RealVNC Ltd.

[Sch01] M. Schumacher. *Objective Coordination in Multi-Agent Systems Engineering*. Springer Verlag, LNAI 2039, 2001. Also published as PhD Thesis, Department of Computer Science, University of Fribourg (CH).

[Sch02] 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 and B. Hirsbrunner. A Model for Software Configuration in Ubiquitous Computing Environments. In Pervasive 2002, International Conference on Pervasive Computing. 26-28 August 2002, Zurich.

[SDA98] D. Salber, A. K. Dey, and G. D. Abowd. Ubiquitous Computing: Defining an HCI Research Agenda for an Emerging Interaction Paradigm. Technical Report GIT-GVU-98-01, Gerogia Institute of Technology, February 1998.

[SG02] J. Pedro 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*, pages 29–43, 25-31 August 2002.

[SGGB99] E. G. Sirer, R. Grimm, A. J. Gregory, and B. N. Bershad. Design and Implementation of a Distributed Virtual Machine for Networked Computers. In *Symposium on Operating Systems Principles*, pages 202–216, 1999.

[SGH98] Norbert A. Streitz, Jörg Geißler, and Torsten Holmer. Roomware for Cooperative Buildings: Integrated Design of Architectural Spaces and Information Spaces. *Lecture Notes in Computer Science*, 1370:4–21, 1998.

[SLD05] W. Shen, X. Li, and A. Doan. Constraint-based Entity Matching. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 862–867, Pittsburgh, PA, 2005.

[SM98] Inc. Sun Microsystems. Java Remote Method Invocation Specification, Revision 1.50, JDK 1.2, October 1998.

[SMTH00] S. Schubiger, S. Maffioletti, A. Tafat, and B. Hirbrunner. Providing Service in a Changing Ubiquitous Computing Environment. In *Workshop on Infrastructure for Smart Devices - How to Make Ubiquity an Actuality, HUC2K, Bristol, UK*, September 2000.

[SRC84] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4), November 1984.

[SSW94] K. Sagonas, T. Swift, and D. S. Warren. XSB as an Efficient Deductive Database Engine. *ACM SIGMOD Record*, 23(2):442–453, June 1994.

[ST94] B.N. Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):22–32, September-October 1994.

[SVSF05] P. Sauter, G. Vogler, G. Specht, and T. Flor. A Model-View-Controller Extension for Pervasive Multi-Client User Interfaces. *Personal Ubiquitous Computing*, 9(2):100–107, 2005.

[Sze96] P. Szekely. Retrospective and Challenges for Model-Based Interface Development. In F. Bodart and J. Vanderdonckt, editors, *Design, Specification and Verification of Interactive Systems '96*, pages 1–27, Wien, 1996. Springer-Verlag.

[Tan97] A. S. Tanenbaum. *Operating Systems: Design and Implementation*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1997. 2nd edition.

[Tan01] A. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 2001. 2nd edition.

[TBGC01] 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)*, pages 447–461, Stanford University, California, USA, August 2001.

[TCH03] 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.

[TCH04] A. Tafat, M. Courant, and B. Hirsbrunner. A Generic Coordination Model for Pervasive Computing Based on Semantic Web Languages. In *9th International Conference on applications of natural languages to information Systems, ICANLIS'04*, Manchester, June 2004. Appeared in Lectures Notes in Computer Science, Springer, F. Meziane, E. Mtais, eds., 2004 vol. 3136. pp. 265-275.

[TCH05] A. Tafat, M. Courant, and B. Hirsbrunner. Implicit Environment-based Coordination in Pervasive Computing. In *20th ACM Symposium on Applied Computing, SAC'05*, pages 457 – 461, Santa Fe, New Mexico, USA, March 2005.

[TvS02] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentince Hall, Upper Saddle River, New Jersey, 2002.

[VM80] F. Varela and H. Maturana. Autopoiesis and Cognition: The Realization of the Living. *Boston Studies in the Philosophy of Science*, 1980.

[Wal98] J. Waldo. JavaSpaces Specification 1.0. Technical report, SUN Microsystems Inc, March 1998.

[Wal99] J. Waldo. The Jini Architecture for Network-Centric Computing. *Communications of the ACM*, 42(7):76–82, 1999.

[Weg95] P. Wegner. Interaction as a Basis for Empirical Computer Science. *ACM Comput. Surv.*, 27(1):45–48, 1995.

[Weg98] Peter Wegner. Interactive Foundations of Computing. *Theor. Comput. Sci.*, 192(2):315–351, 1998.

[Wei91] M. Weiser. The Computer for the 21st Century. *Scientific America*, 265(3):94–104, September 1991. reprinted in IEEE Pervasive Computing, pp. 19-25 Jan.-Mar. 2002.

[wel04] Welcome Project Leaflet. `http://diuf.unifr.ch/pai/research/welcome`, 2000 - 2004.

[WFG92] R. Want, A. Hopper V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.

[WG00] Z. Wang and D. Garlan. Task Driven Computing. Technical Report CMU-CS-00-154, Carnegie Mellon University, May 2000.

[Wir85] N. Wirth. *Algorithms and Data Structures.* Prentice Hall, 1985.

[WMLF98] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T-Spaces. *IBM Systems Journal archive*, 37(3):454 – 474, 1998.

[WS-03] *The W3C Web Services Architecture working group*. `http://www.w3.org/TR/2003/WD-ws-arch-20030808/`, August 2003. public draft.

CURRICULUM VITAE

Sergio Maffioletti
Swiss National Supercomputing Centre CSCS
Galleria 2 - Via Cantonale
CH-6928 Manno (Switzerland)

Personal information

Birth:	11th September 1971
Place of birth:	Bergamo, Italy
Nationality:	Italian
Marital status:	married
URL:	http://diuf.unifr.ch/people/maffiole
Email:	sergio.maffioletti@unifr.ch

Languages:	Italian:	mother tongue
	French:	spoken and written
	English:	spoken and written

Academic Credential

- PhD Candidate in Computer Science
  University of Fribourg (CH)
  Dissertation: “UbiDev: a Homogeneous Service Framework for Pervasive Computing Environments”.
  Supervisors: Prof. Béat Hirsbrunner and Michèle Courant
  02/2000 – 06/2006

- Master Degree of Science, Computer Science
  Thesis subject: "Intelligent Agents: Message Scheduler and Conversation".
  Supervisor Prof. S.A.Cerri.
  Milan University
  Milan (Italy)
  1998

- Maturity degree in Computer Science
  Paleocapa ITIS
  Bergamo (Italy)
  1990

Professional Experience

•**Technical Assistant**

Microsoft Italia spa. Milan (Italy) 03/1994 – 12/94.

- Technical support for all Microsoft products.
- Advanced support to Windows and NT OS.

•**Technical Assistant**

Microsoft Italia spa. Milan (Italy) 06/1995 – 10/95.

- Technical support for Microsoft products.
- Beta testing and support on Microsoft Windows 9.

•**Technical Assistant & Developer**

Communal Library. Capriate commune. Bergamo (Italy). Summer 1996

- Consulting: configuration, administration and maintenance of library network system (based on Windows).

•**Technical Assistant & Developer**

Communal Library. Brembate commune. Bergamo (Italy) 1997 - 1998

- Consulting: configuration, administration and maintenance of library network system (based on Windows). Support and development of communication utilities for bibliography classification (basically done in C).

•**Software educational**

Filippin Institute. Bassano del Grappa (Italy). Summer 1999

- Organized and lead professional course: "Java: programming language for Internet applications" for secondary school's teachers.

•**Software educational**

Communal Library. Brembate commune. Summer 1999

- Organized and lead course: "Internet and neighbours". Open course.

•**Software Developer**

Milan University. FPL laboratory. Milan (Italy). 02/1999 – 10/1999

- Developed agent-based applications in the context of the European project LarFLaST.
- Java based agent system dealing with Web information retrieval and representation.

•**Software Developer**

SoftSolution srl. Bergamo (Italy). 10/1999 – 02/2000

- Implemented, documented and maintained applications for E-market.
- Java-based packages for communication and configuration utilities.
- Java-based request broker (as ORB for Corba) for interoperability among heterogeneous communication systems.
- Java-based informational bus.

• **Scientist in Grid Computing**

Swiss National Supercomputing Centre 06/2004 – Present.

Publications

- S. Maffioletti, S.K. Mostéfaoui and B. Hirsbrunner . **Automatic Resource and Service Management for Ubiquitous Computing Environments.** to Appear in Middleware Support for Pervaisve Computing Workshop (at PerCom '04), PerWare '04, Orlando, Florida USA, 14 March 2004.

- S. Le Peutrec, M. Courant, S. Maffioletti and B. Hirsbrunner. **Adaptable Interfaces for Augmented Virtual Reality.** Proceedings ot the First Research Workshop on Augmented Virtual Reality, AVIR'03, Geneva, Switzerland, 18 - 19 September 2003. pp. 29-30.

- S. Maffioletti, S.K. Mostéfaoui and B. Hirsbrunner. **Automatic Resource and Service Management for Ubiquitous Computing Environments.** Technical Report no.03-17, Department of Informatics, University of Fribourg, September 2003

- M. Courant, S. Le Peutrec, S. Maffioletti and B. Hirsbrunner. **Architecture for a full-dynamical Interaction in Pervasive Computing.** Proceedings of the 10th International Conference on Human - Computer Interaction, HCI03, Crete, Greece, 22 - 27 June 2003.
  Appeared in **Human-Computer Interaction Theory and Practice**, C. Stephanidis and J. Jacko (Eds.), Laurence Erlbaum Associates, 2003 vol.2 no.Part II. pp. 38-42.

- S. Maffioletti and B. Hirsbrunner. **Towards a Homogeneous Coordination Space for Ubiquitous Interacting Entities.** Technical Report no.03-02, Department of Informatics, University of Fribourg, February 2003.

- S. Schubiger, S. Maffioletti and B. Hirsbrunner. **Making Sense in Heterogeneous and Dynamic Environments.** Personal and Ubiquitous Computing, 2003. Submitted.

- S. Maffioletti and B. Hirsbrunner. **UbiDev: A Homogeneous Environment for Ubiquitous Interactive Devices.** Short paper in Pervasive 2002 - Proceedings of International Conference on Pervasive Computing, ETHZ, Zurich, Switzerland, 26 - 28 August 2002. pp. 28-40.

- S. Maffioletti. **Requirements for an Ubiquitous Computing Infrastructure.**
  Proceedings ot the Cultura Narodov Prichernomoria Journal, Simferopol, Ukraine, September 2001. Appeared in Cultura Narodov Prichernomoria Journal vol.3 no.ISSN 1562-0808. pp. 35-40.

- S. Maffioletti, S. Schubiger and B. Hirsbrunner. **Towards a Homogeneous Environment for Ubiquitous Interacting Devices.** Technical Report no.01-21, Department of Informatics, University of Fribourg, July 2001.

- S. Schubiger, S. Maffioletti, A. Tafat-Bouzid and B. Hirsbrunner. **Providing Service in a Changing Ubiquitous Computing Environment.** Proceedings of the Workshop on Ubiquitous Computing Enabling the Networked Society, ICSI, HUC'00, Bristol, UK, 27 September 2000. Proceedings of the Workshop on Infrastructure for Smart Devices.

- S. Maffioletti. **UbiDev: A Middleware for Ubiquitous Computing.** Department of Informatics, University of Fribourg, PhD thesis, started 2000-. in progress.