UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CRISTIANO ANDRÉ DA COSTA

Continuum:
A Context-aware Service-based Software Infrastructure for Ubiquitous Computing

Thesis presented in partial fulfillment of the requirements for the degree of Doctor of Computer Science

Prof. Dr. Cláudio Fernando Resin Geyer
Adviser

Porto Alegre, November 2008.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Costa, Cristiano André da

Continuum: A Context-aware Service-based Software Infrastructure for Ubiquitous Computing / Cristiano André da Costa – Porto Alegre: Programa de Pós-Graduação em Computação, 2008.

170 f.:il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2008. Orientador: Cláudio Fernando Resin Geyer.

1.Sensibilidade ao Contexto. 2.Computação Ubíqua. 3.Computação Pervasiva. 4.Infra-estrutura de Software. 5.Middleware. 6. Ontologia. I. Geyer, Cláudio Fernando Resin. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Aldo Bolten Lucion
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenadora do PPGC: Prof[a]. Luciana Porcher Nedel
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

ACKNOWLEDGEMENTS

My most sincere thanks are due to my adviser Cláudio Geyer, who accepted me under his supervision as a PhD student, and for his support during almost three and half years in the development of this work.

I would like to express my deepest thanks to my great friend Adenauer Yamin, for helping me all through this work, from the choosing of the thesis' theme to the various revisions and suggestions in the final text. Without his support, this work would not be the same.

I would like to thank in particular Luciano Cavalheiro, my colleague at UFRGS, for his contribution in many discussions, project directions, and articles.

I would also like to express my thanks to the colleagues that developed the prototypes and conducted the experiments presented in this work, particularly, Felipe Kellermann, Óliver Pessutto, and Rodolfo Stoffel. Thank you for your dedication.

I would like to express my gratitude to my dear friend Andrei Cunha, who is my English (and French) instructor, assisting me in the proofing of this text (and also in obtaining the proficiency certification in French). Many thanks for his patience in reading all the material produced and undoubtedly improving the quality of the text.

Special thanks are due to my beloved wife Adriana, who supported me throughout the writing of this work. I thank you for standing by me during all the moments we missed together because I was working on the thesis.

I would like to thank Unisinos for the institutional support. I would also like to thank the Institute of Informatics at UFRGS, which provided an excellent environment for research and science development. I thank all the professors and staff of this institution.

This work would not have been possible without the help of many researchers that read the papers produced, gave suggestions, and devoted their time in the improvement of the research. Furthermore, I would like to express my thanks to the organizations that support academic events and journals worldwide, such as SBC, ACM, and IEEE.

Thanks to Apple Inc., for developing outstanding hardware and software. Carrying out this work using Mac computers was a great and enjoyable experience.

I would like to dedicate this work to the excellent professors that I had during my academic journey so far, at ETFPEL, UCPEL, and UFRGS. Without their commitment and competence, I would never have achieved this goal.

TABLE OF CONTENTS

LIST OF ABREVIATTIONS AND ACRONYMS

AJAX	Asynchronous Java Script + XML
ANSI	American National Standards Institute
AOP	Aspect Oriented Programming
API	Application Program Interface
APPELO	*Ambiente de Programação Paralela em Lógica* (Parallel Logical Programming Environment)
ATM	Automatic Teller Machine
AVU	*Ambiente Virtual do Usuário* (Virtual User Environment)
CAC	CoApp Configuration
CAI	CoApp Implementation
CAR	CoApp Resources
CD	Compact Disc
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
CLOPn	*Sistemas Escaláveis de Alto Desempenho para Programação Lógica com Restrições* (Hi-performance Scalable Systems for Constraint Logical Programming)
CML	Context Modeling Language
CMU	Carnegie Mellon University
CNPq	*Conselho Nacional de Desenvolvimento Científico e Tecnológico* (National Counsel of Technological and Scientific Development)
CoApp	Continuum Application
CoBrA	Context Broker Architecture
CoDSA	Continuum Distributed Service Architecture
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets

DBMS	Database Management System
DFS	Distributed File System
DHCP	Dynamic Host Configuration Protocol
DL	Description Logic
DNS	Domain Name System
DOM	Document Object Model
DSLP	Distributed Scheduler for Logic Programming
DUMMBO	Dynamic Ubiquitous Mobile Meeting Board
EPA	*Espaço Pervasivo de Arquivos* (Pervasive File Space)
E-R	Entity-Relationship
EXEHDA	Execution Environment for Highly Distributed Applications
FOAF	Friend of a Friend
GaiaOS	Gaia Operating System
GeneAl	Grid Approach for Genetic Sequence Alignment
GPS	Global Positioning System
GRANLOG	Granularity Analyzer for Logic Programming
GUI	Graphical User Interface
HCI	Human-computer interaction
Holo	Holoparadigm
HP	Hewlett-Packard
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IBM	International Business Machines
ID	Identification
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
II	*Instituto de Informática*
INCITS	International Committee for Information Technology Standards
IoS	Internet of Services
IP	Internet Protocol
ISAM	*Infra-estrutura de Suporte às Aplicações Móveis* (Mobile Applications Support Infrastructure)
ISAMpe	ISAM Pervasive Environment

ISO	International Organization for Standardization
Java ME	Java Micro Edition
Java SE	Java Standard Edition
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
KB	Knowledge Base
LIME	Linda in Mobile Environment
MultiS	Multi-Sensor Context Server
MVC	Model-View-Controller
NSF	National Science Foundation
OOP	Object Oriented Programming
Opera	Or Parallel Prolog
ORB	Object Request Broker
ORM	Object-Role Modeling
OS	Operating System
OWL	Ontology Web Language
P2P	Peer-to-peer
PACE	Pervasive Autonomic Context-aware Environments
PC	Personal Computer
PDA	Personal Digital Assistant
PerDiS	Pervasive Discovery Service
PHD	Portable Help Desk
PloSys	Parallel Logic System
QoS	Quality of Service
RDB	Relational Database
RDF	Resource Description Framework
RFID	Radio Frequency Identification Badge
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RuleML	Rule Markup Language
SaaS	Software as a Service
SDK	Software Development Kit
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture

SOAP	Simple Object Access Protocol
SOC	Service-Oriented Computing
SOCAM	Service-Oriented Context-Aware Middleware
SOUPA	Standard Ontology for Ubiquitous and Pervasive Applications
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SWRL	Semantic Web Rule Language
TCP	Transfer Control Protocol
TTL	Time To Live
Ubicomp	Ubiquitous Computing
UCE	Ubiquitous Computing Environment
UCPel	*Universidade Católica de Pelotas* (Catholic University of Pelotas)
UDP	User Datagram Protocol
UFRGS	*Universidade Federal do Rio Grande do Sul* (Federal University of Rio Grande do Sul)
UFRJ	*Universidade Federal do Rio de Janeiro* (Federal University of Rio de Janeiro)
UI	User Interface
UIUC	University of Illinois at Urbana-Champaign
UML	Unified Modeling Language
UNISINOS	*Universidade do Vale do Rio dos Sinos* (University of Valley of Rio dos Sinos)
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
W3C	World Wide Web Consortium
Web	World Wide Web
Wi-Fi	Wireless Fidelity
WSDL	Web Service Description Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The present work is a proposal of a context-aware software infrastructure for ubiquitous computing (ubicomp) named Continuum. The ubicomp area, also called pervasive computing, presupposes a strong integration with the real world, with focus on the user and on keeping high transparency. For the development of applications in this scenario, we need an adequate software infrastructure. The infrastructure designed in this work is based on service-oriented architecture (SOA), making use of framework and middleware, and employing a redefinition of follow-me semantics. In this redefined vision, users can go anywhere carrying the data and application they want, which they can use in a seamlessly integrated fashion with the real world. The specific focus of our work is context awareness: the perception of characteristics related to users and surroundings. We consider the resources available in the environment and keep a history of context data. Furthermore, we propose the representation of context to promote reasoning and knowledge sharing, using ontology. In this way, context is represented in a considerably expressive, formal approach, different from many solutions that exist today, which still use ad hoc representations models. Our work is then at the intersection of these three main areas: software infrastructures for ubicomp, context awareness, and ontologies. In the development of this thesis, we also survey the field of ubiquitous computing, suggesting a general architectural model to deal with its fundamental challenges. Based on the established requirements for this model, we propose a set of services for Continuum. The services are designed considering the previous works developed by our research group, namely ISAM (*Infra-estrutura de Suporte às Aplicações Móveis* – Mobile Applications Support Infrastructure), and particularly the middleware EXEHDA (Execution Environment for Highly Distributed Applications). We further extend these projects, by adding aspects to them that had not been considered at the time of their development. Particularly, we improve context awareness support, proposing an ontology for the formalization of context information. We have conducted some analysis, using case study methodology, to evaluate the main propositions of our work. Based on these assessments, we present lessons learned and draw the conclusion of our work. As a result, Continuum is a software infrastructure that addresses many aspects of ubiquitous computing, seamlessly integrating many different challenges.

Keywords: context awareness, ubiquitous computing, pervasive computing, software infrastructure, middleware, ontology.

Continuum: Uma Infra-estrutura de Software Sensível ao Contexto e Baseada em Serviços para a Computação Ubíqua

RESUMO

Este trabalho apresenta uma proposta de infra-estrutura de software sensível ao contexto para a computação ubíqua (ubicomp) denominada Continuum. A área de ubicomp, também chamada de computação pervasiva, pressupõe uma forte integração com o mundo real, com foco no usuário e na manutenção de alta transparência. Para o desenvolvimento de aplicativos nesse cenário, é necessária uma infra-estrutura de software adequada. A infra-estrutura projetada é baseada no padrão da arquitetura orientada a serviços (*service-oriented architecture* ou SOA), fazendo uso de framework e middleware, e empregando uma redefinição da semântica siga-me. Nessa visão redefinida, os usuários podem ir para qualquer lugar carregando os dados e os aplicativos que desejam, os quais podem ser usados de forma imperceptível e integrada com o mundo real (*seamless integration*). O foco particular desse trabalho é sensibilidade ao contexto: a percepção de características relacionadas aos usuários e ao entorno. No trabalho são considerados os recursos disponíveis no ambiente e é mantida a história dos dados de contexto. Além disso, é proposta a representação do contexto para promover raciocínio e compartilhamento de conhecimento, empregando uma ontologia. Dessa forma, contexto é representado de uma maneira formal e bastante expressiva, diferente de muitas soluções existentes hoje em dia que ainda usam modelos de representação *ad hoc*. Esta tese está então na interseção destas três áreas principais: infra-estrutura de software para ubicomp, sensibilidade ao contexto e ontologias. No desenvolvimento desta tese, também examina-se o campo da computação ubíqua, e sugere-se um modelo de arquitetura geral que enfrente esses desafios fundamentais. Baseado nos requisitos estabelecidos para esse modelo, propõe-se um conjunto de serviços para o Continuum. Os serviços são projetados considerando o trabalho previamente desenvolvido pelo nosso grupo de pesquisa, mais especificamente o projeto ISAM, e particularmente o middleware EXEHDA. A proposta estende esses projetos, adicionando aspectos que não haviam sido considerados no momento do seu desenvolvimento. Particularmente, o suporte a sensibilidade de contexto é melhorado com a proposta de uma ontologia para a formalização da informação de contexto. Algumas análises, usando a metodologia de estudo de caso, foram conduzidas para apreciar as principais proposições da tese. Baseado nessas avaliações, foram apresentadas algumas lições aprendidas e traçada a conclusão do trabalho. Como resultado, Continuum é uma infra-estrutura de software que endereça muitos aspectos da computação ubíqua, integrando imperceptivelmente diferentes desafios.

Palavras-Chave: sensibilidade ao contexto, computação ubíqua, computação pervasiva, infra-estrutura de software, middleware, ontologia.

1 INTRODUCTION

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it" (WEISER, 1991). Mark Weiser's statement from his classic and visionary article about computation for the 21st Century summarizes what is expected from pervasive or ubiquitous computing (ubicomp): user access to the computational environment, everywhere and at all times, by means of any device. The difficulty lies in how to develop applications that will continually adapt to the environment and remain working, as people move or change devices (GRIMM et al., 2004).

The more traditional mobility goal of providing computation any time and anywhere is considered a reactive approach to information access. However, it represents a proactive step toward ubicomp. This is what we mean by the expression *all the time, everywhere* (SAHA and MUKHERJEE, 2003). For this purpose, we need a new class of software. The development of this field, however, is still hindered by the limited number of frameworks and tools available (ROMÁN et al., 2002).

Ubiquitous applications need middleware to interface between many different devices and end-user applications (SAHA and MUKHERJEE, 2003). The aim is to hide environment complexity, by isolating applications from the explicit management of protocols, distributed memory access, data replication, communication faults, etc. Middleware can also solve heterogeneity problems related to architectures, operating systems, network technologies, and even programming languages, promoting their interoperation. On the other hand, a framework is an environment, composed of APIs, user interfaces, and tools, which simplifies software development and management in a specific domain (BERNSTEIN, 1996). We can use frameworks to develop middleware and to build software that runs on that middleware.

This middleware must allow users to access their computational environment (applications and data) at any time and place. One possible solution is to apply *follow-me semantics* (AUGUSTIN et al., 2004; YAMIN et al., 2003). The idea behind this concept is that applications and data go along with users, providing a virtual environment and adapting to the current context. This adaptation is fundamental to ubiquitous computing vision, and involves the perception of the context (*context awareness*) and the proper adjustment of the system based on this perceived information (*context management*).

In this perspective, the defended thesis is that *the use of a software infrastructure specifically aimed at ubicomp can reduce the distance between Weiser's vision and the current distributed computing scenario.* To accomplish this goal, our work focuses on

the proposal of a service-based software infrastructure for ubiquitous computing, employing framework and middleware.

The main focus of our work is on context awareness, so that the environment encompasses the characteristics that the users need, enhancing the real world. To achieve this goal, we propose a redefinition of the follow-me semantics concept: users can go anywhere carrying the data and application they want, which they can use in a seamlessly integrated fashion with the real world. This notion differs from the original one in two aspects: first, there is no idea of virtual user environment but rather the idea of using the actual environment. Second, the user's session is not sustained for all applications and data; instead, we propose that users choose which applications and data they want to carry with them. We believe that with this new approach, we break with the idea of replicating the user desktop session in every scenario and increase the applicability of the solution to more general situations.

Our proposal differs from other works, such as Aura (GARLAN et al., 2002; SOUSA et al., 2006), Gaia (RÓMAN and CAMPBELL, 2000; RÓMAN et al., 2002), and One.World (GRIMM et al., 2004), to the extent that we have a more general focus on enhancing the real world. Aura concentrates on users, and specifically on their attention. Gaia adopts an opposite view, and emphasizes smart spaces, i.e., particular physical environments with embedded services and devices. One.World, in its turn, aims mainly at providing a set of libraries and services for building applications.

In this thesis, we propose a software infrastructure, not specifically aimed at the user nor at specific environments, but rather at a global view, unhindered by a local or personal scope. Differently from One.World, we investigate and propose innovative solutions to deal with context. Also, we have distinctive assumptions related to infrastructure, software development model, and to the set of services that should be available to the user.

The work presented here makes use of ISAM (*Infra-estrutura de Suporte às Aplicações Móveis* – Mobile Applications Support Infrastructure) (AUGUSTIN et al., 2004), especially of EXEHDA (Execution Environment for Highly Distributed Applications) middleware (YAMIN, 2004). We propose an evolution of the ISAM project, modifying EXEHDA middleware to better support context awareness issues involved. EXEHDA focus is on execution support and the provision of an infrastructure for the development of pervasive applications. Perhaps the main changes are consequence of the difference in the execution model by the redefinition of follow-me semantics.

This chapter establishes the background of the thesis, presenting motivation, goal, and description of the research problem. At the end, the organization of the text as a whole is presented.

1.1 Background and History

Various projects in which we have been involved since the past decade have had an influence on the contents of this thesis. In this section, the main projects are highlighted and their historical evolution explained in chronological order.

The OPERA project was started in the Laboratory of Génie Informatique in Grenoble, at University Joseph Fourier, and its development extended to the Informatics Institute (II) at UFRGS in the late 1980s. OPERA is a parallel Prolog implementation intended to increase the speed of logic programming execution towards the OR parallelism exploitation. Besides, another OPERA aim is to offer an alternative to simplify the programming of parallel architectures.

At the beginning of the 90's, some researchers of our group at UFRGS were involved in the development of a distributed operating system to foster heterogeneity among different computers, named HetNOS (BARCELLOS et al., 1994). The aim of this project is to simplify distributed programming, by providing a layer over native operating systems. With the development of HetNOS, our research team gained experience on middleware development and, more specifically, on addressing heterogeneity issues in distributed systems.

At the same time, OPERA gave rise to many other projects: exploitation of AND parallelism has been added (YAMIN, 1994; WERNER, 1994); a new version of OPERA targeting exploitation of OR parallelism in shared memory architectures has also been built (named PloSys – Parallel Logic System) (MOREL et al., 1996); and an environment for the automatic analysis of granularity in logic programming, GRANLOG – Granularity Analyzer for Logic Programming (BARBOSA, 1996), has been developed.

Among OPERA's spin-offs we should highlight DSLP – Distributed Scheduler for Logic Programming (COSTA, 1998), with which the author of this thesis has been directly involved. DSLP is a model of hierarchical scheduling for the exploitation of AND/OR parallelism in logic programming (COSTA and GEYER, 1998). This model uses granularity information from GRANLOG.

Later, OPERA activities were encompassed by a multi-institutional project named APPELO (*Ambiente de Programação Paralela em Lógica* - Parallel Logical Programming Environment) (GEYER et al., 1999). APPELO has involved researchers of various Universities: UFRGS, UFRJ, UCPel, University of Porto (Portugal), and New Mexico State University (NM, USA).

With the end of APPELO funding, researchers gave continuity to the project through CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico* - National Counsel of Technological and Scientific Development) and NSF (National Science Foundation / USA) support. The new project was named CLoPn (*Sistemas Escaláveis de Alto Desempenho para Programação Lógica com Restrições* - Hi-performance Scalable Systems for Constraint Logical Programming) (COSTA et al., 1999). The aim of CLoPn was to develop a program environment that provided a high level declarative language to the user, towards parallelism exploitation and programmable hardware, promoting cooperation between Brazilian research groups and the New Mexico State University research group partner.

From this vast experience in the exploitation of logical programming parallelism, we started a new research investigation related to multi-paradigm models. This investigation gave birth to Holoparadigm (BARBOSA, 2002), a distributed programming model. We found that, due to its inherent characteristics, Holoparadigm was particularly suitable for use in mobile environments (YAMIN et al., 2003) and Grid computing-based systems (BARBOSA et al., 2004; BARBOSA et al., 2005).

In order to support the distributed execution of such mobile applications with adaptive behavior, and considering pervasive computing, we conceived Project ISAM (Infra-estrutura de Suporte às Aplicações Móveis - Mobile Applications Support Infrastructure). The programming model used in ISAM was Holoparadigm.

In the scope of ISAM, ISAMadapt (AUGUSTIN, 2004) specifies the abstractions to express, in development time, context adaptation in mobile and distributed applications targeting pervasive computing. Among others, ISAMadapt defines abstractions to adapters in applications entities; adaptation policies, which guide decision-making in the middleware; and also, context elements, which direct adaptive mechanisms of the execution environment.

The execution of applications in ISAM is managed by a middleware named EXEHDA (YAMIN, 2004). EXEHDA proposes an architecture of mechanisms for the coordination, communication, and adaptation, targeting application execution in pervasive computing. The middleware provides a reactive and active behavior in the management of application entities, comprising scalability and cooperating support, based on definitions made in design- and execution-time.

EXEHDA generated various subprojects developed at UFRGS. Among the main projects, we can highlight: PRIMUS – objects execution and distribution support in pervasive computing (SILVA, 2003); DIMI – dissemination strategy optimization of information originated from resource monitoring (MORAES, 2005); PerDiS – resource discovery service in pervasive computing (SCHAEFFER, 2005); MultiS – a multi-sensor context server for ubiquitous computing (FEHLBERG, 2007); EPA – use of application-aware adaptation in pervasive file access (FRAINER, 2008). The present work describes a more evolved construct, based on ISAM and EXEHDA, of a software infrastructure for ubiquitous computing.

1.2 The Problem

The complexity of software development increases according to the functionalities we want to provide to the user. With the advent of ubiquitous computing, we need software that is able to run using innumerous and assorted network-connected devices, seamlessly integrate with the real world, to keep the focus on the users, and to disappear into the environment, as if it was invisible. To these features we should also add the characteristics that a particular software should provide. Imagine trying to solve a real world problem in a specific domain, and also having to include all the features of ubicomp. To draw a parallel in the history of Computing Science, we can compare it to Lotus 1-2-3 in the beginning of the PC era. The spreadsheet was completely written in assembly language, which involved the development of various complex routines, such as floating-pointing and fixed-pointing math (KAPOR, 2007). At that time, assembly was chosen because the requirements were small memory usage and the fastest speed possible (KAPOR, 2007). This choice introduced an overhead in software development, since many libraries and routines had to be implemented.

We are nowadays living the beginning of the Ubicomp era. Although we have had a huge evolution in languages and tools for software development since the advent of the PC, when we focus on the ubiquitous computing requirements, we are at the first steps. We need middleware and framework to smooth the progress of software

implementation in this scenario. It is still difficult to find a software infrastructure that has all the necessary characteristics of ubiquitous computing; besides, the tendency today is providing middleware or frameworks for specific issues. In spite of this tendency, we think that a general infrastructure model for software may help to develop pervasive middleware or frameworks. Our thesis proposes that, in order to fulfill Weiser's vision, future ubiquitous infrastructures should seamlessly integrate many different challenges.

An additional problem in the ubicomp scenario is how the infrastructure can let users access their data and applications wherever they go and however they move. The promise of "at all times, everywhere" that came with the idea of ubicomp is difficult to be obtained. Another related problem is how to use these data and applications in a seamlessly integrated fashion with the real world. These issues involve the addressing of mobility, heterogeneity, and scalability among other concerns.

Besides these general problems, we face specific questions related to context awareness. Regardless of the recent attention given to this area, context-aware software is still running on labs, rather than the real world every-day situation (HENRICKSEN and INDULSKA, 2006). The main reasons for that, according to Henricksen and Indulska (2006), are the overhead imposed by application development, social barriers related to privacy and usability, and an unclear understanding of the context-aware possibilities.

Particularly, systems today tend to use ad hoc data structures for representing context and specific communication mechanisms; there is no standardization format and protocols, which leads to weak interoperation (BALDAUF et al., 2007). Furthermore, many projects employ informal models to represent context, making programming more cumbersome, and also affecting the providing of reasoning and knowledge sharing among systems (HASELOFF, 2005).

Some of the major drawbacks in context-aware systems available today are the lack of security and privacy consideration and the absence of a discovery mechanism to find sensors available in the surroundings (BALDAUF et al., 2007). In the specific case of ISAM, besides these shortcomings, there is no storing of historical context data, which limits the establishment of trends and the prediction of future context values. Also, ISAM does not have a policy for placement of contextual data.

To further illustrate the problems we are trying to address, we present in the next subsection some scenarios in which the lack of software infrastructure for ubicomp, and proper addressing of context-aware issues, greatly hinders the development of applications. Also, we try to show conditions in which our proposal could bring benefits.

1.2.1 Sample Scenarios

Let us consider a University Campus. A Campus is an attractive environment to the development of ubiquitous applications due to people familiarity to it and because of its easy access (GARLAN et al., 2002). Suppose that in this Campus we have a ubiquitous application named e-Campus (Electronic Context-Aware Mobile Pervasive Ubiquitous System). This application could offer services to the Campus community classified in three areas: academic, administrative, and logistic. Services should be available all over

the campus area. They could be accessed by various different types of devices, spread around the campus, such as cell phones, digital displays, desktops, notebooks, PDAs, etc. A person physically present at the University could benefit from a variety of services provided by e-Campus:

- *Schedule Book*: it comprises an administrative service. Its main function is to provide the management of the person's agenda everywhere. It also reminds the person of appointments and anticipates actions needed in the next appointments, for example: confirming participation in events, preparing the environment for meetings, informing the location and means of access, preparing material that will be used in the next appointment, etc.;
- *UbiCourse*: an academic service that keeps the digital environment related to a specific course. It allows a course participant, either a professor or a student, to access all the digital information related to it (not only the current edition, but also all the past editions). When a student is in class time, verifiable by the Schedule Book, and at the classroom, UbiCourse shows in every device, an interface to access the course. For instance, the presentations that will be used on that day, the past presentations and exercises which the teacher chooses to disclose, the list of pending activities, the particular student notes, the public annotations, the calendar of the course, the data which classmates choose to disclose, the history of the course over the time, the classmates present in that specific class, etc. In the class we could have a digital whiteboard, and other devices on the desks and spread around the room, which facilitate the interaction with the service;
- *Digital Transport*: a logistic service related to mobility and transport. It shows the location of rooms and buildings, presents the time someone needs to go from one place to another, finds people and events, suggests routes, indicates free parking space in parking lots and special parking spots, features bus timetables and delays, etc.;
- *Service Finder*: a logistic feature to locate services available in a specific place. The system should be proactive and suggest services according to people's needs, location, and preferences. Example of services: printing, visualization, ATM, toilets, etc.

The ideas listed here will be analyzed in order to select case studies in the process of thesis development. It is not our aim to exhaust, in this subsection, all the possible services and applications that could be available in a campus; nor is it to present all the services that could be developed using concepts of the current work.

1.3 Thesis Goals

The general goal of this thesis is to propose a service-based software infrastructure that facilitates the development of ubiquitous computing applications. In the process of proposing this software infrastructure, we make use of ISAM and EXEHDA.

The specific focus of this thesis is *context awareness*. Our major concern is to deal with context awareness issues in the design of a software infrastructure for ubicomp. Differently from ISAM, the objective here is not to keep the users' virtual environment,

but rather to enhance the real world making use of software architecture, encompassing the characteristics that the user needs.

To attain these goals, we need to address some specific concerns:

1) To survey the field of ubiquitous computing, revising the main concepts and the state of the art;
2) To define a general architectural model that supports ubiquitous computing's fundamental challenges, establishing the base for the proposition of our infrastructure;
3) To create a set of services for the development of ubiquitous applications. We propose the use of service-oriented architecture (SOA) and web services for building context-aware services, contributing to the increase of interoperability and standardization;
4) To use a representation for context to promote reasoning and knowledge sharing. We employ ontology instead of the markup scheme model originally used by EXEHDA, giving meaning to context;
5) To make use of the resources available in each and every environment, obtained dynamically. We propose the utilization of resource discovery in the process of finding sensors;
6) To keep a history of context data for each entity in the environment. We propose means of storing context data, suggesting a database of historical context with querying capabilities. We also deal with the distribution and placement of context information;
7) To develop a prototype of the architecture for service support, employing the use of web services;
8) To build an experimental evaluation, based on case studies to describe and explain the proposed ideas in the context-aware subsystem.

As a result, we propose an evolution of ISAM, and consequently of EXEHDA middleware, considering these context-aware aspects. The original proposal (YAMIN, 2004) focuses only on partial context awareness: obtaining raw information, distributing it, and converting it into an abstract context element guided by an XML (eXtensible Markup Language) description. This work substantially improves the original project with many features not predicted at that time.

Due to the time, space, and scope limitations for the development of this work, it is *not* our goal to tackle certain issues:

- To detail all the services and features that could be available in the infrastructure; rather, we give a view of which services could be available and a general set of primitives;
- To specify the Continuum framework. We focus mainly in the middleware, although the proposed comprehensive architecture presents issues that should be covered by this framework;
- To fully implement the Continuum software infrastructure;

- To validate and test all the propositions presented here. Instead, we intend to provide a case study to demonstrate the main characteristics designed.

The thesis shows throughout its chapters these ideas and developments.

1.4 Project Name

We chose Continuum as the name of our project. The New Oxford American Dictionary's definition of the term is "a continuous sequence in which adjacent elements are not perceptibly different from each other, although the extremes are quite distinct." We settled on this name, because we wanted to represent the idea that from the point-of-view of individuals, the software infrastructure should be almost imperceptible in their daily activities, even if its use brings about some real world enhancements and changes in the environment, aiming to create a *continuum* between reality and its improvements.

We chose the Möbius strip (Figure 1.1) as the logo of our project because it represents a continuous curve.

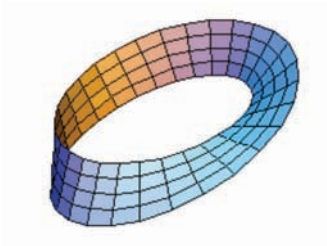

Figure 1.1: Continuum logo – the Möbius strip

1.5 Thesis Outline

The text of the thesis is structured around two main parts. In the first part, located after the introductory chapters, we define the basis around which the work evolves. The first part is a horizontal thread, in which the software infrastructure is defined and detailed, and is composed of chapters 3 and 4. We consider this as a horizontal thread, because it deals with the general goal of the thesis, and establishes the support for the development of the work's focal point.

Built over this foundation, we present then the second part, where the focus is on context awareness. This comprises the vertical thread, in which the context-aware architecture of Continuum is revealed and assessed, and is developed in chapters 5 and 6. We consider this as a vertical thread, because it deals with the specific focus of our thesis, an in-depth study. After this second part, we go on to the prototype implementation and analysis of results, followed by the conclusion of the work. Figure 1.2 summarizes this structure, presenting its relation with the organization of the text.

The text is organized as follows. Chapter 2 is an overview of the ubiquitous computing field and of the motivations behind our present work. It presents the fundamental concepts of ubiquitous computing, and then its evolution. Essential areas to the field are emphasized and issues that are unique or still open are discussed.

In chapter 3 we propose a comprehensive architectural model for ubiquitous computing. This comprises the main requirements for the development of a software infrastructure for ubiquitous computing. It starts with the discussion of limitations in the use of traditional programming models, and then goes on to the proposition of the architectural model to address these limitations. It is also in this chapter that we investigate work related to ubiquitous software infrastructure.

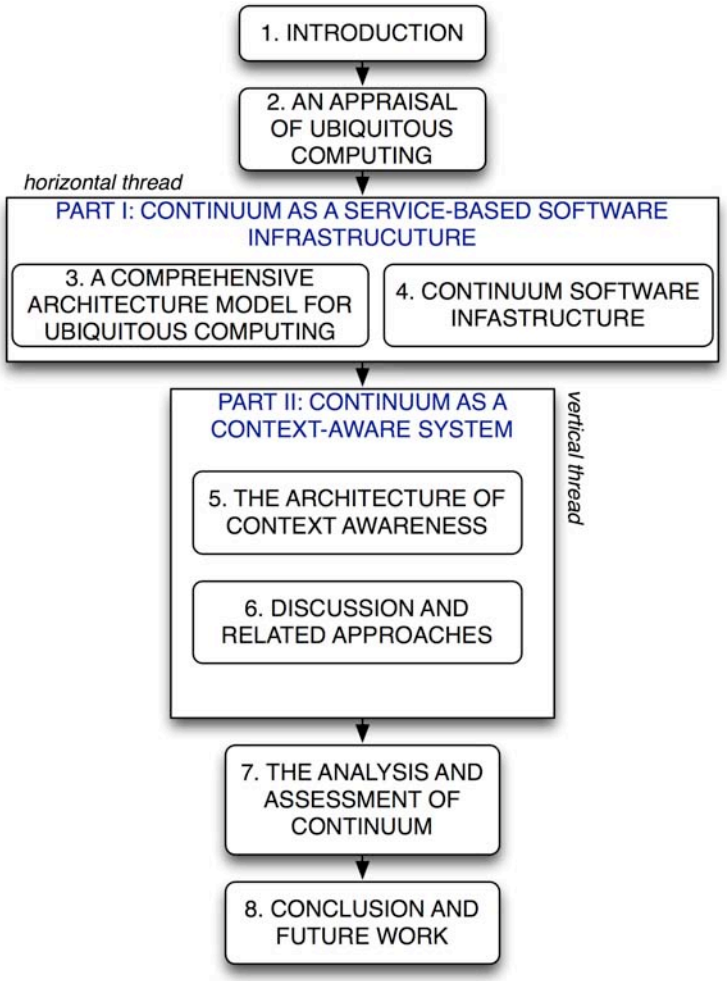

Figure 1.2: Text structure and organization

Chapter 4 describes Continuum. It proposes a software infrastructure for ubiquitous computing, based on the requirements discussed in the previous chapter. We present the physical organization of devices and also all the layers and services of the proposed infrastructure.

The design of context awareness in Continuum is detailed in chapter 5. This includes the description of the context model we employ, as well as its representation and means of utilization. The ontology proposed is described, along with the methodology employed to obtain it. We also present the subsystem and services related to context awareness.

In chapter 6 we reflect on the experience of designing context awareness in this infrastructure and discuss some points that were only partially addressed. The design

principles are described, as well as the main projects related to context awareness. Also, we discuss points in common and differences between our experience and the state of the art in context awareness.

Chapter 7 presents the assessment of Continuum along with the analysis of results. We describe the method used for validation, i.e. case study, and present the experimental evaluation of the main propositions in the project. Three broad case studies are conducted. The first one assesses the proposition of the distributed service architecture. In the second case study, we analyze the formal representation of context and its ability to infer and represent context information. The third case study, on the other hand, evaluates the possibilities in terms of available tools and standards for the implementation of the context awareness subsystem.

Finally in chapter 8, we present the conclusion of this thesis, summarizing the main contributions, and suggesting future work.

2 AN APPRAISAL OF UBIQUITOUS COMPUTING

In this chapter we briefly review essential concepts of the area, its evolution, and propose challenges that must be addressed in the field.[1] We should begin by defining *ubiquitous computing* (also called *ubicomp*). Mark Weiser created this term, so he is considered one of the area's fathers. He presents computer ubiquity as the idea of integrating computers *seamlessly*, invisibly enhancing the real world. Weiser (1991) formulates a "new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the *background*." Computers will vanish as a consequence of human psychology: when people use things without consciously thinking about them, they focus beyond. This is a phenomenon defined by some philosophers and psychologists (WEISER, 1991): people cease to be aware of something when they use it sufficiently well and frequently. Philosopher Heidegger calls this phenomenon *ready-to-hand*[2] and Edmund Husserl calls it *the horizon*.[3]

Heidegger makes a phenomenological analysis of the way people deal with the world. According to him, our first behavior toward entities such as tools, devices, and systems within the world is one of *use*. These entities, viewed from their aspect of use, are called ready-to-hand. In "Being and Time"[4], Heidegger (1996) affirms that:

> The peculiarity of what is proximally ready-to-hand is that, in its readiness-to-hand, it must, as it were, withdraw in order to be ready-to-hand quite authentically. That with which our everyday dealings proximally dwell is not the tools themselves. On the contrary, that with which we concern ourselves primarily is the work.

Edmund Husserl was the first to propose the horizon concept (KEEN, 1975). Husserl was a philosopher, one of the founders of phenomenology, and a mathematician. The concept refers to human experience as a background that turns

---

[1] To further facilitate the understanding of the area, we added at the end of this volume a glossary with the definition of the main terms used in ubiquitous computing.

[2] *Vorhandenheit* in the original.

[3] *Horizont* in the original.

[4] The book was first published in 1927 with the title *Sein und Zeit*. For this text the translation to English published in 1996 was used.

experiences possible. Horizon points to a network of known meanings focusing not much on physical things, but on an ordered pattern which we formulate implicitly in our act of being (KEEN, 1975).

To achieve the physical integration of computers into the world, as a background, we must apply some conceptual changes. In this perspective, Weiser also defines *embodied virtuality* in opposition to virtual reality, as computers cannot be limited to their devices and software installed. Moreover, it is inadequate to consider the Internet or distributed file systems access as an example of *seamless integration.* Weiser points out that the power of ubiquitous computing does not stem from the capacity of a particular device, but rather from the interaction of all devices.

Besides computer interaction, *scale* and *location* are two important topics highlighted by Weiser. There will be many computers per room, in different sizes, with different user's interfaces, and suitable for specific jobs. Computers must also know where they are and use this information to adapt to the environment. *Adaptation* is then currently one of the crucial concerns in ubicomp.

Analyzing Weiser's vision, Saha and Mukherjee (2003) state that, in spite of significant hardware developments, computers are still machines that run programs in virtual environments and not yet a "portal into an application-data space." Want et al. (2002) agree that many hardware components are now ready for ubiquitous computing, as a consequence of many improvements since Weiser's seminal article, including wireless networks, high-performance low-powered processors, enhancements in displays, high-capacity, and low-powered storage devices.

To achieve ubiquitous computing, we need advances in physical integration and in spontaneous interoperation as defined by Kindberg and Fox (2002). Integration between devices and the physical world is crucial. There should be system boundaries and specifications for the scope of the environment, but these should not be a constraint to interoperation. As components move among devices and environments, they must change both identity and functionality in order to interoperate.

We must understand and support everyday practices of people to reach Weiser's vision, offering different forms of interactive experiences through heterogeneous devices connected via integrated network components (ABOWD et al., 2000).

2.1 Defining Pervasive Computing

The origin of the term *pervasive computing* is frequently associated with IBM. This is perhaps because it was used as the main subject of a whole issue of the IBM System Journal (Vol. 38, No. 4, 1999). This issue defines pervasive computing as a change in the way we view computers and their use. Computers are everywhere and are used not as distinct machines but rather as "sophisticated, computerized, networked machines" (HOFFNAGLE, 1999), i.e. parts of larger devices.

A group of IBM researchers (BANAVAR et al., 2000) defined three characteristics associated to pervasive computing:

> First, it concerns the way people view mobile computing devices, and use them within their environments to perform tasks. Second, it concerns the way applications are created and deployed to enable such tasks to be performed. Third, it concerns the environment and how it enhanced by the emergence and ubiquity of new information and functionality.

As a consequence of these characteristics, authors maintain that a new application model is needed in pervasive computing. Devices must be a portal into application and data space; applications are means for performing tasks; and the environment is the physical world with the user's information (BANAVAR et al., 2000).

Grimm et al. (2004) affirms that pervasive computing suggests a "computing infrastructure that seamlessly and ubiquitously aids users in accomplishing their tasks and that renders the actual computing devices and technology largely invisible." This vision creates the need for smart devices in the real world. Devices must coordinate with each other, in order to accomplish user's tasks. The difficulty lies in designing, building, and deploying applications in these circumstances (GRIMM et al., 2004).

Another definition (SATYANARAYANAN et al., 2001) suggests that the essence of pervasive computing is "the creation of environments saturated with computing and communication capability, yet gracefully integrated with human users". Satyanarayanan also asserts that pervasive computing and ubiquitous computing are basically different terms used to describe the same concept. The main difference between both concepts is that pervasive computing is a bottom-up vision that emerged from the widespread exploitation of computing services, while ubiquitous computing is a top-down approach were these services are used in a transparent manner and integrated with the environment (ROBINSON et al., 2005). In this text, we adhere to this vision, although many authors nowadays treat both terms as synonyms.

2.2 Evolution

The advent of Personal Computers (PCs) in the mid 1970s, besides making computers popular, brought them closer to people and represented a first step in the direction of ubiquitous computing (SAHA and MUKHERJEE, 2003). However, making the computer personal is a technological misplacement in Weiser's vision. The computer remains the focus of attention, and is thus isolated from the overall situation (WEISER, 1993).

Distributed computing is generally considered a major step in ubicomp evolution. The need to exchange information and communication stimulated the development of computer networks. Distributed systems benefited from this already existing infrastructure, acting as a set of interconnected computers using communications links in different media and topology. In such systems, processing entities exchange information using message passing through a variety of protocols to perform an execution of distributed tasks. To accomplish this distributed computing, more research was and is still needed in many areas.

Satyanarayanan (2001) emphasizes some areas important to pervasive computing foundation in the spectrum of distributed systems:

- *Remote communication*: techniques such as message passing, remote procedure call (RPC) and group communications are common possibilities for interaction in distributed systems. More recently, with the wide dissemination of object-oriented programming, approaches like remote method invocation (RMI) and code mobility are being widely used. RMI integrates RPC with the object-oriented paradigm. Code Mobility is a different method that makes it possible to create a dynamic change of location in which objects are executed (FUGGETTA et al., 1998). This is an interesting concept to apply to ubiquitous computing;
- *Fault Tolerance*: the aim is to make computing systems more reliable in handling faults. Faults are defects at the lowest level and may cause errors. An error, in effect may lead to a failure deviating the system from its correct specification (GÄRTNER, 1999). An important measure in this field is the *dependability* of a system, which is the ability to avoid service failures that are more frequent and more severe than acceptable (AVIŽIENIS et al., 2004);
- *High availability*: availability is concerned with the readiness for correct services (AVIŽIENIS et al., 2004). High availability requires mechanisms such as data replication and recovery. By data replication we mean maintaining multiple copies of data in different machines. This increases availability by allowing access to data even when some computers are unavailable. Optimistic replication increases availability and is better suited for mobile computing (SAITO and SHAPIRO, 2005);
- *Remote information access*: distributed file system (DFS) and databases are a common information repository. They allow users of distributed computers to share data across the network in a transparent manner (LEVY and SILBERSCHATZ, 1990). An important aspect of DFS is user mobility. In the system users can log and use their files from any machine. It is up to the system to locate the data and to transport them to the client machine;
- *Security*: an important issue in distributed systems is how to ensure authenticity, authority, integrity, confidentiality, and non-repudiation. Security mechanisms such as cryptography and secure protocols are used. Privacy and trust are major concerns and these increase with ubiquitous computing. Another concern is that users must trust infrastructure and the exchange of information. It is important to define how much information or how many resources should be disclosed to others (ROBINSON et al., 2005). The application of trust management systems and models is necessary.

Another important step in evolution is the World Wide Web (hereafter referred to simply as the Web). With the Web, information and communication have become nearly ubiquitous. The simple mechanism used for linking resources is a good way of integrating distributed information and a potential starting point for pervasive computing, even though the Web does not integrate the physical world (SAHA and MUKHERJEE, 2003).

The final step in evolution is mobile computing. This arises from advances in two areas: *wireless networking* and *portable devices*. With these devices the user can access

information anywhere, regardless of their physical location or mobility (JING et al., 1999). The difference from traditional computing is that computing services go with people and become more present, providing expanded capabilities. Combined with network access, those services transform computing "into an activity that can be carried" (LYYTINE and YOO 2002).

Limited resources such as wireless bandwidth, battery life, computational power, screen size, etc. are typical mobile constrains. On the other hand, software does not change significantly as we move. To address these problems, *adaptation* is needed. This consists in reacting to changes and creating a dynamic balance between available resources and applications needs.

Moreover, Jing et al (1999) discusses two other major research aspects of mobile computing, apart from adaptation: *extended client-server model*[5] and *mobile data access*. The former consists of dynamically partitioning responsibilities between client and server, while the latter covers issues related to remote data access, cache consistency, and ways of structuring data.

An additional contribution from mobile computing in this evolution, emphasized by Satyanarayanan (2001), is *location sensitivity*. Research in this field proposes algorithms and techniques for sensing physical location. Certain systems also provide *location-aware* behavior.

The integrated possibilities brought about by the development of PC, distributed systems, the Web, as well as mobile computing, set the stage for ubiquitous computing to evolve. The main issues involved in achieving ubiquity will be described in the next section.

2.3 Ubiquitous Computing Challenges

To achieve ubiquitous computing, as proposed by Weiser, some challenges must be addressed. A number of previous studies enumerate issues unique or still open in pervasive computing (BANAVAR et al., 2000; GRIMM et al., 2001; KINDBERG and FOX, 2002; NIEMELÄ and LATVAKOSKI, 2004; SAHA and MUKHERJEE, 2003; WANT and PERING, 2005).[6] In this section the key challenges are presented and discussed. The study presented here was published in IEEE Pervasive Computing (COSTA et al., 2008). Table 2.1 summarizes the main challenges and presents their aliases, areas in which they gain focus and central motivations to address these in the scope of ubicomp.

Heterogeneity is a concern derived from distributed systems. Applications must be able to run in different kinds of devices, with assorted operating systems, and user's interfaces. Software must mask differences in infrastructure to the user and manage the required conversions from one environment to another. As a result, we must address protocol mismatches. In this scenario, it is impossible to recreate device-specific

[5] In the context of ubiquitous computing referred as *cyber foraging*.

[6] See Appendix B for a related work analysis of ubiquitous computing challenges.

software. Consequently, application logic must be created only once with a device-independent approach.

Table 2.1: Ubiquitous computing challenges

Issue	Alias	Focus Area	Motive
Heterogeneity		Distributed systems	- Variety and difference - Different types of devices, networks, systems, and environments
Scalability	Localized Scalability[7]	Distributed systems	- Large scale - Increase in the number of resources and users
Dependability and Security	Fault Tolerance[8]	Mission-critical and Distributed Systems	- Avoiding failures that are more frequent and more severe than acceptable - Providing availability, confidentiality, reliability, safety, integrity, and maintainability
Privacy and Trust		Internet and Mobile computing	- Protecting against bad use of personal data - Defining the trustworthiness of interacting components
Spontaneous Interoperation	Volatility	Mobile computing	- Allowing interaction with a set of components that can change both identity and functionality - Permitting association and interaction
Mobility	Follow-me applications	Mobile computing	- Application and data access anywhere and any time - The user environment goes along
Context awareness	Perception	Mobile and Ubiquitous computing	- Perceiving user's state and surroundings - Inferring context information
Context management[9]	Smartness, Masking uneven condition, Adaptability	Mobile and Ubiquitous computing	- Modifying the behavior of the system based on the perceived context information - Adapting
Transparent User Interaction	Human-computer interaction[10]	Ubiquitous computing	- Merging user interface with the real world - Allowing user focus on tasks with minimal distraction
Invisibility	Ubiquity, Pervasively	Ubiquitous computing	- Allowing users focus on task, not tools - Making computers disappear in the background

Another related issue inherited from distributed systems is scalability. In pervasive computing, a large number of users, devices, applications, and communications are expected on a scale never established before. Furthermore, it would be impractical to explicitly distribute and install applications. We must avoid centralized solutions for

[7] This term was used by Satyanarayanan (2001) and means that physical distance is a significant issue in pervasive computing and that we must consider the important role played by local interactions.

[8] Actually this term is more restrictive. Recently the community is converging to use the more general word *dependability*.

[9] Some authors consider context management as a part of context awareness.

[10] This term is used in a more general sense.

better scalability and bottleneck prevention. Moreover, distant interactions must be reduced to a minimum.

Sometimes, the system cannot execute according to functional specifications. Additionally, there might arise problems related to misspecifications. These situations lead to failures. A failure is defined as a transition from a correct service to an incorrect service (AVIŽIENIS et al., 2004). A correct service is obtained when the system implements the desired function. Incorrect service should be detected and execution restored to a correct state. Avoiding failures that are more frequent and more severe than acceptable leads to *dependability*, a concept that integrates the attributes of availability, reliability, safety, integrity, and maintainability. The term pervasive dependability has been used to refer to these needs in the scope of ubicomp (FETZER and HÖGSTEDT, 2002).

Security is a concept strictly related to the dependability of a system. A system is considered secure if there are measures to assure availability, integrity, and confidentiality. There are many mechanisms to provide security in distributed systems that could also be used in ubicomp. However, these actions must be lightweight, to preserve both the spontaneity of interactions and the limitations of some devices, in the provision of security for resources and user data (COULOURIS et al., 2005).

The *privacy* of that user data is a noteworthy matter. As ubicomp becomes more a part of everyday life, almost invisible devices will collect user information, including personal data, without even being noticed. Guarantee the ways in which such information could be used or passed on will be extremely difficult. Another associated challenge is *trust*. In a very heterogeneous and dynamic scenario, the trustiness of interacting components should be evaluated. Since there is no fixed infrastructure and neither a specific domain, we must use a trust management system to measure what should be disclosed to other components (ROBINSON et al., 2005).

Bringing together varied components available in several devices, as well as making communication and understanding among these possible, is a challenge identified as *spontaneous interoperation*. A component interoperates spontaneously if it “interacts with a set of communication components that can change both identity and functionality over time as its circumstances change” (KINDBERG and FOX, 2002). We need this spontaneity because of the volatile nature of ubicomp. The components are in movement and interacting with a constantly changing set of services.

Another challenge named *mobility* provides access to applications and data wherever users go and however they move (AUGUSTIN et al., 2002). This is because, in portable devices, such as PDAs and notebooks, the environment goes along with the user. However, physical mobility (of equipments or the users) is not the only option. Moving components such as applications, data and services (logical mobility) is also desirable. Nowadays, many of these components are attached to a specific device, so that the user cannot carry them along. Applications should move from one device to another, and data access should be maintained (follow-me applications) (AUGUSTIN et al., 2002).

Mobile computing has also introduced the idea of *context awareness*, i.e., inferring context to supply information or services to the user when the availability of services is limited or intermittent (DEY, 2001). The concept is broader in ubicomp than in mobile computing, as devices must sense changes and software should act proactively. Context

is defined as any information that can be used to describe the situation of entities (persons, places, or objects) (DEY, 2001). It is generally acquired using embedded computers or sensors. However, most devices today cannot sense their environment, and neither can the software react to these changes.

Since it is possible to perceive context, it is necessary to use this information and act proactively. *Context management* is action in response to sensing. Based on sensed data, the system makes decisions such as configuring services according to environmental change or keeping memory of past environments to restart services when users reenter in those (LYYTINEN and YOO, 2002). Management can also expand the capacity of devices by using available resources in the current context.

Human-computer interaction (HCI) design is also a significant subject. With ubicomp there will be many ways of interacting with users. On top of that, as computers become ‘smarter’, the intensity and quality of human-computer interaction is bound to increase (SAHA and MUKHERJEE, 2003). Focus on user interface evolved from software design, but it acquired a different meaning since the emergence of mobile computing and new modes of interaction. Merging user data with the real environment is another condition for HCI development in ubicomp. This redirects the attention *to transparent user interaction.* The idea is to preserve human attention, avoiding information saturation (SIEWIOREK, 2002). Users must be able to focus on the task without distractions from the system.

The last issue is directly related to ubicomp itself. *Invisibility* is about keeping user focus on the task, not on the tool (WEISER, 1994). To fulfill this vision, software must satisfy user intent, by helping, not obstructing it. Software should learn with users and, in some cases, let they change their preferences, interacting “almost at a subconscious level” (SATYANARAYANAN, 2001).

To address these challenges, a software infrastructure is needed. The next chapter focuses on this subject. It starts with a discussion on why traditional models do not fit pervasive computing. A general architectural model for ubiquitous computing is then presented. Finally, some current projects are discussed.

PART I:
CONTINUUM AS A SERVICE-BASED SOFTWARE INFRASTRUCTURE FOR UBIQUITOUS COMPUTING

3 A COMPREHENSIVE ARCHITECTURAL MODEL FOR UBIQUITOUS COMPUTING

Ubiquitous applications need a middleware to interface between many different devices (desktops, notebooks, PDAs, wireless equipment, etc.) and end-user applications (SAHA and MUKHERJEE, 2003). The aim is to hide environment complexity isolating applications from explicit management of protocols, distributed memory access, data replication, communications faults, etc. A middleware can also solve heterogeneity problems related to architectures, operating systems, network technologies, and even programming languages, promoting the interoperation of them. On the other hand, a framework is an environment, composed of APIs (Application Program Interfaces), user interfaces and tools, that simplifies software development and management in a specific domain (BERNSTEIN, 1996). A framework is used to build software that runs on a middleware. The middleware itself can be developed using existing frameworks.

In this chapter we propose a comprehensive architectural model targeting ubicomp that uses framework and middleware. This model considers all the challenges we believe significant in the ubiquitous computing field. The focus of this comprehensive architectural model is on highlighting numerous requirements necessary to ubiquitous computing that should be covered by a software infrastructure. Before presenting this model, we argue why traditional development models do not fit ubiquitous computing. An abridged version of this chapter appeared in IEEE Pervasive Computing (COSTA et al., 2008).

3.1 Implementing Ubiquitous Applications

A great effort is dedicated today to the development of distributed systems. Many languages and frameworks have been in use to implement such systems. The Object Oriented Paradigm (OOP) is the dominant programming model utilized. Distributed objects are accordingly becoming more common. Despite the use and dissemination of this model, some authors affirm that this is not sufficient to ubiquitous computing and a new programming framework is required. Some of the major reasons for that are the challenges presented in the previous chapter: the traditional programming models does not usually address all topics discussed.

In this text, traditional programming models are considered the techniques currently used when implementing software. In general, these models are applied for the

development of distributed software and based in OOP. In the core of these models are programming languages such as Java, C++, and C#.

There are three central limitations, in the traditional programming models presently in use, to implement distributed systems that affect pervasive computing (GRIMM et al., 2004):

- *Distribution is transparent*: communications mechanisms employed such as distributed objects, RMI, and DFS hide physical location to developers. This transparency simplifies programming, since both local and remote resources can be used practically in the same manner, but that makes context awareness and management more difficult;
- *Components integration via interfaces*: All objects export an interface of methods to be used by other components. This facilitates composition among objects but presupposes a tight coupling, complicating the addition of new behaviors. Usually the interface is considered quite stable;
- *Object abstraction*: objects encapsulate code and data. Keeping data inside objects makes data sharing more difficult. Also, data is usually stored without proper format definition.

In addition to these limitations, traditional computing development models usually are based on static assumptions: architectures, applications, data, operating systems, etc. Moreover, in general, all resources that are used must be known a priori. To make matters more complicated application interfaces are commonly developed integrated with the program logic. As a result, it is not easy to create pervasive and seamlessly integrated applications using only traditional models and OOP.

An important shortcoming of traditional models is the lack of support for changes in the system. Frequently, manual intervention is required to address these changes. Saha and Mukherjee (2003) defend that adaptation to the environment is one of the chief characteristics that differentiate ubiquitous computing from traditional computing.

Although there is need to address many features in traditional computing to conceive ubiquitous applications, we must consider support for legacy applications, popular operating systems, use of existing data, and users' knowledge on how to use current software and systems (KINDBERG and FOX, 2002). Because of that, it is normal that architectures for the development and execution of ubiquitous applications be based on traditional models with extended functionalities. Typically, a software infrastructure is built, creating layers of abstraction for ordinary hardware, operating systems, and traditional programming models, adding a set of new services to address general limitations.

3.2 Architectural Model

A comprehensive architectural model targeting ubiquitous computing is presented. This model considers all the challenges explained before (in section 2.3). Nevertheless, this model should widen the spectrum of commonly used languages and systems. In effect, current methods for remote communication, fault tolerance, high availability, remote information access, and security could be inherited. The focus of this general architectural model is on highlighting numerous requirements necessary to ubiquitous

computing that should be covered by a software infrastructure. This model could also be useful in classifying proposals and suggesting needed features.

Figure 3.1 presents the comprehensive infrastructure model we propose, including each of the issues highlighted before and the corresponding characteristics that should be available to address it. The structure is then divided considering the application life cycle (*design time*, *load time*, and *runtime*) (as in BANAVAR et al., 2000). Design time is when the application is conceived, extended, or maintained. At load time, applications are loaded to specific devices. At runtime, applications are executed and utilized by the user.

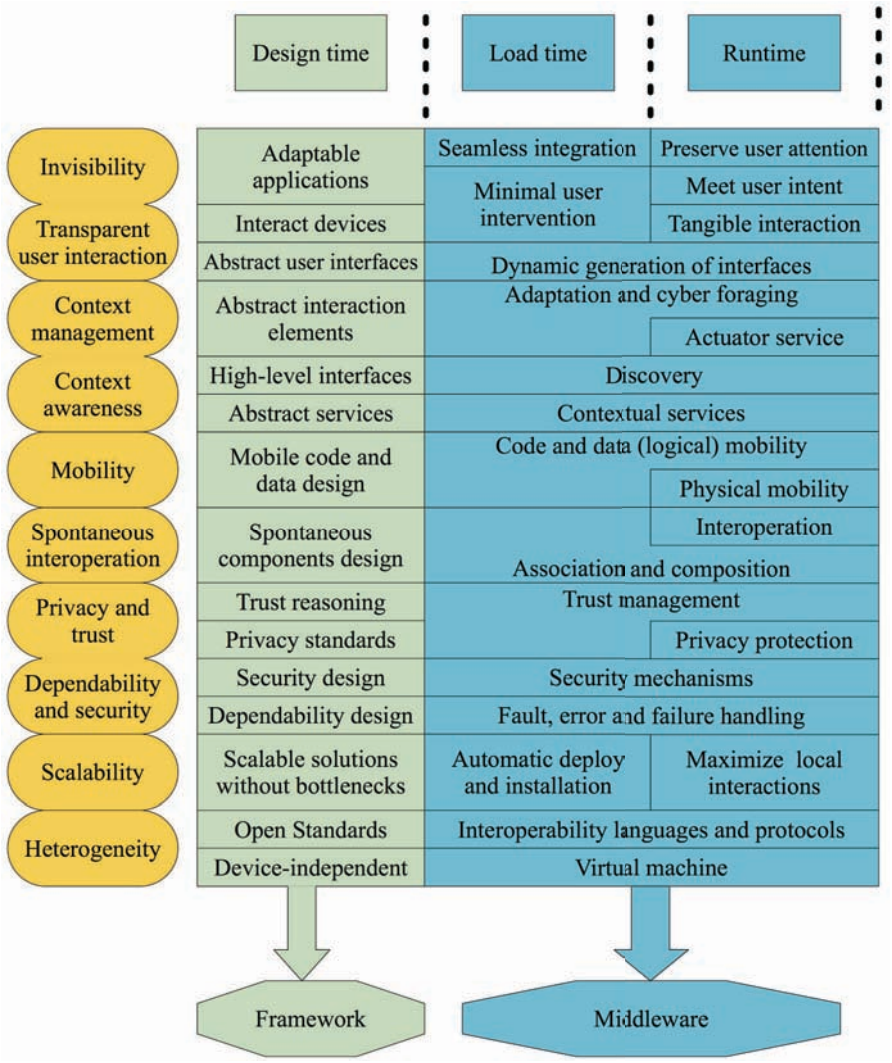

Figure 3.1: Comprehensive architectural model for ubiquitous computing

Each row presents a challenge (in an oval box at the leftmost side of the figure) and, on its right side, we can see the essential characteristics to be addressed at design time, load time, and runtime, respectively. Some challenges, such as *dependability and security* and *privacy and trust*, are more closely related to each other than others (these are represented without a separating horizontal line). In this situation, we can define dependability as the ability to deliver services that we can justifiably trust. Moreover, to attain privacy protection, collected personal data should be secure. Close dependence

also involves *context awareness* / *context management* and *transparent user interaction* / *invisibility*, making it difficult to draw an exact borderline.

The order of the issues does not imply a layered model, in which each tier depends on the services provided by the other. Services appear bottom-up, low-level services first. At the bottom, we introduce the challenges already tackled in distributed systems. The figure also shows issues more related with mobile computing in the middle, and the challenges that arise with ubicomp at the top.

A framework can provide the abstractions needed to ubiquitous computing at design time. The design time column shows all the characteristics of this stage. The same applies to load time and runtime. However, to provide the characteristics required in these stages, we suggest the use of middleware.

3.3 Infrastructure Characteristics

In the next subsections is a more detailed discussion of each row of the general architectural model. Since the challenges were described before (section 2.3), the focus will be on the characteristics proposed to address each one of those.

3.3.1 Heterogeneity

There are several levels of heterogeneity both in hardware (networks, devices, screen sizes, power capability, etc.) and in software (languages, component models, structures, etc.). To facilitate the bridging between heterogeneous systems, we should use open standards, with published interfaces and standardized communications mechanisms, allowing easier system extension or re-implementation.

Also, frameworks for device-independent projects can make it possible for different hardware, even from diverse vendors, to use the same source-code, sometimes with little alteration. Thus, we can keep the developed application almost unmodified, limiting change to device-drivers or to the framework itself.

The current solution to heterogeneity is to use middleware with a common and integrated Application Programming Interface (API), and a unified binary format. This binary file should run on a virtual machine, like Java, which would be available on all platforms. However, depending on device capability, we cannot always employ the same virtual machine, run the same binary code, or expect that the set of available features remain unchanged. For instance, Java has different virtual machines for mobile devices and PCs. Nevertheless, the use of virtual machine reduces the cost of heterogeneity because fewer changes are needed compared to languages that generate specific machine codes.

Finally, we need to focus on the interoperability of components, the “ability to understand the exchanged information and to provide something new originating from the exchanged information” (NIEMELÄ and LATVAKOSKI, 2004). Interoperability languages, such as XML, are commonly used, making it possible to represent data in a standard and structured form, more portable between applications. In other cases, software converts source data into an expected format. This conversion is transparent to the user, but differences may occur between the source and destination versions.

Besides, protocols that can negotiate services and resources between applications and devices must be available, allowing integration during load and execution.

3.3.2 Scalability

To address the problem of scalability, we need to develop software that considers the abundance of users, interactions, components and devices, avoiding centralized solutions and bottlenecks. The management and loading of applications should be automatically done at load time. Besides, whenever a new application is available, it should be automatically deployed and installed, since manual distribution and installation of software for each device would be impractical.

During execution time, interaction with distant resources should be reduced. This idea, *localized scalability* (SATYANARAYANAN, 2001) should be a goal of ubicomp, even if it disagrees with the current guideline of network transparency, in which local and remote resources are accessed with identical operations, their physical location notwithstanding. We should consider the location of resources and give priority to local interactions over distant ones.

3.3.3 Dependability and Security

Among the attributes encompassed by dependability, in the scope of ubicomp, we need to maximize reliability, availability, and safety. It is also vital to minimize the cost of maintainability and the effort to preserve integrity. In terms of security, we have to deal directly with the attribute of confidentiality, but also with availability and integrity.

During the development of applications, verification could diagnosis and remove faults. Verification is the process of checking if the system adheres to certain characteristics. Causing faults should otherwise be diagnosed, corrected, and the verification process should be repeated (AVIŽIENIS et al., 2004). Testing is a widely used type of dynamic verification.

Failure detection and recovery strategies used today (such as checkpointing, compensation, isolation, or reconfiguration) could be applied to ubicomp as well. However, there are some concerns to address in ubicomp, because requirements are different from those of traditional computing, since applications execute in environments, and there is always a context involved. Also, devices are means of access to applications, but some failures in devices may not be specified in application or middleware. Besides devices and applications failure, we should also consider network and services failure (CHETAN et al., 2005).

We ought to differentiate failures from changes in the system, i.e., situations requiring detection and recovery mechanisms from those where adaptation takes place. In order to have an adaptable system, we need to specify which types of changes will cause adjustments, even though we cannot predict all kinds of possible situations. Sometimes, some unpredicted changes occur. The system may also generate unspecified results. These are examples of failures, in which no adaptation mechanism is possible. In these cases, we must detect and recover the failures. We also should not consider disconnections as failures, but rather as part of the system specifications, treating them with adaptation mechanisms.

There are some approaches that can be used in ubiquitous system to increase dependability (CHETAN et al., 2005):

- *Using a surrogate*: when a failure of an application is detected, one common technique is to restart this application with the last state saved in a stable store. Sometimes however, the application has failed because of a device problem. In this case, a surrogate device can be used to run the restarted application. Other possibility is that this device could not run the same application, and in that case an equivalent one can be used;
- *Alternate notification mechanisms*: if the system detects that a communication with the user has failed, other communication interfaces can be used. Various ways of interacting with the user provide redundancy;
- *Handling Errors in sensing and inferring context*: to avoid errors in the perception of context, multiple sensors and/or algorithms to infer context can be used. A complementary approach is to permit users to indicate errors observed by the system;
- *N-versions approach*: the idea is to use redundant modules with different implementations to execute the same task. A software arbitrator is then used to determines the correct answer and provide the result.

The security design of a ubiquitous system must consider some aspects (DOURISH et al., 2004). First, it should be user-centered, i.e., consider usability. Users can circumvent security mechanisms that are discordant to common practices (BARDRAM, 2005). Second, security depends on the context and because of that mechanisms should be near the activity in which it makes sense. Third, the design must be made in a way that users understand and manage the employed solutions. Only thus can the user choose the suitable mechanism according to the security needed in each action and context.

The mechanisms to deal with security in the perspective of ubicomp must also consider three characteristics. They should be scalable to devices with limited resources, expect lack of knowledge, and allow dynamicity of mobility (ROBINSON et al., 2005). For instance, user authentication for each and every device through login and password would not be feasible. We need other methods; for example, the system could exploit biometric information, or authenticate based on the location of people.

3.3.4 Privacy and Trust

Directly related with the security concerns are the aspects of privacy and trust. They are treated separately from the previous issue because of their magnitude in ubicomp. Although privacy is typically a subject of legislation, technology should be applied in this new scenario of ubiquity due to the risks of the user exposing too much personal information to an environment, sometimes even unaware of the surveillance. On top of that, an increase is expected in the amount and accuracy of data collected. Furthermore, the protection of privacy is particularly difficult in ubiquitous system because of location-sensitivity. The context-aware mechanism of sensing the exact user location could be exploited for tracking purposes. With this mechanism, it is possible to infer the movement of the users and their activities, associating it with their personal information.

During design, we can apply privacy standards. These standards are enforced by jurisdiction and market, and consist of a group of procedures that should be observed in the collection of data (ROBINSON et al., 2005). During the execution phase, we can employ protection mechanisms to realize these standards. For instance, data could be accumulated anonymously or deleted after a period of time.

A trust management can establish the trust in the relationship among components to the exchange of information and resources access. The difficulty lies in precisely defining the trustworthiness of an interacting entity and grant permissions based on that decision. In some cases, there is little or even no evidence available about an entity and, as in our daily trust decisions, it is more of a subjective notion. Apart from being subjective, trust has other characteristics (CAHILL et al., 2003): non-symmetry (two interacting components can have different trust in each other), situation-specific (dependent of context), dynamic (increase or decrease in time), and it is inherently associated with risk (no reason to trust if there is no risk involved). Because of these, there should be trust reasoning support. This reasoning analysis is made based on available information and considering the various aspects of trust. In this case, solutions for uncertainty should also be present.

3.3.5 Spontaneous Interoperation

The first step is the design of spontaneous components, i.e., entities that support a frequent change in the communicating partners and that can easily interact with others. To accomplish this design, we need not a fixed, but a dynamic environment, with assorted infrastructure and partners. The availability of a framework can facilitate the development of spontaneous components and provide a generic interface, which will be combined to specific entities during execution. Ideally, we should employ a uniform description language for the specification of components, and build them independently of context (NIEMELÄ and LATVAKOSKI, 2004).

During execution, components associate with each other. Association is the logical relationship established between components that allow interactions; we call these interactions interoperation (COULOURIS et al., 2005). When assessing association, three points are important (COULOURIS et al., 2005; KINDBERG and FOX, 2002): *scale* – efficiently choosing components to associate in a scenario with various possible partners; *scope* – defining the extent to which components must be considered and including all possible partners; *boundary principle* – considering the physical limits (or other criteria) when defining the scope of association. We can also use discovery services (in this architectures, a context awareness characteristic) as a part of the association solution.

Interoperation depends on the communication models employed. In ubicomp, we tend to use models based on event systems or tuple spaces, due to the asynchronous nature of the former, or the ease of development and inherent persistence of the latter. Occasionally, both models are used in the same middleware. Conversely, we can apply other forms of communication such as message passing, remote invocation, or agent systems.

Composition is a special case of association, in which external components control inner ones, since all interoperation passes through the former, redirecting or modifying the association. Composition facilitates adaptation and mobility. Each device can have a

specific component nesting all others and making all the required changes to their specific interfaces and capabilities. When a component migrates from one device to another, it enters in the specific device components and continues to issue the same set of operations. The adaptation process is up to the outer component of each device, as is the redirection of messages or events arriving after an inner component has migrated.

3.3.6 Mobility

In ubicomp, users changes devices frequently, but user applications and data must always be available. This means that the environment should migrate from one device to another. Besides, migration also helps reducing communication costs or preventing disconnection.

To support code migration during load- and runtime, components must be designed with mobile technology. We can obtain this by using languages and systems compatible with code mobility (FUGGETTA et al., 1998). During execution, middleware has to deal with the mobile component and manage migration. To achieve this, the middleware should be aware of the network, and not treat it in a transparent manner.

We must also address data mobility. We cannot always employ remote data access, due to the possibility of disconnection or deficiency of resources. In these situations, data could be moved or copied to different locations, provided attention is given to data coherence and synchronization. Also, conversion between different formats, for specific applications, or hardware, may be necessary.

Besides code and data mobility support, also known as logical mobility, we need to consider physical mobility. As people move, the devices in use will change their network addresses. This is because they will be communicating with different access points and being assigned to different IP addresses. The DHCP provides this dynamic acquisition of addresses, allowing devices to maintain service access, regardless of location. However, it might be difficult for other components to interoperate with those devices, because the IP routing mechanism is based on fixed locations, and may lose packets when addresses change. Besides, their updating on the DNS is slow, due to extensive use of cache.

To support physical mobility, we can employ a location management strategy. Conceptually, this strategy consists of two operations (ADELSTEIN et al., 2005): *search* – operation invoked by a node that needs to communicate with a mobile device; and *update* or *registration* – operation performed by the mobile node to inform its current location. Another crucial concern is ensuring that a mobile node remains connected while moving from one scope to another. This is known as *handoff*, and involves the following steps (ADELSTEIN et al., 2005): deciding when to change to a new scope, selecting it, acquiring resources, and rerouting packets to the new location.

3.3.7 Context Awareness

To be ubiquitous, middleware must use relevant information and services available in the surroundings. Discovery is the component that detects services and devices in the current context, while sensors infer the significant information that can be used by the context manager to reason about actions to take. The addition of context awareness

characteristics to middleware increases the usability of devices and allows better user interaction (LOKE, 2006).

We need framework support to assist the implementation of context-aware applications. Two characteristics are fundamental in this (DEY, 2001): a set of abstract services that programmers can employ in the building of their components, and high-level interfaces that hide specific devices or sensors details from the user.

During execution, we must store and share context data generated by sensors. We suggest a uniform data representation to improve data access anywhere and from any application. For a truly ubiquitous system, instead of just representing data, we also need some form of knowledge representation. *Ontologies* could be used to explicit semantic representation. One possible model, among various ongoing solutions, is SOUPA – *Standard Ontology for Ubiquitous and Pervasive Applications* (CHEN et al., 2004), that is a shared ontology specifically designed for ubicomp.

To manage this contextual information, middleware must provide four categories of contextual services (ADELSTEIN et al., 2005; DEY et al., 2001): *context subscription and delivery* – a service that can notify a component in the occurrence of some event; *context query* – a mechanism to find suitable information or service; *context transformation* – the conversion of low-level data into high-level information; *context synthesis* – the aggregation of context information to generate a more precise or detailed context. Besides these services, we also need dynamic resource discovery, which are detailed in the next subsection.

3.3.7.1 Discovery

Dynamic resource discovery is a mechanism to dynamically locate and enumerate resources, available in the environment or matching certain requirements (ZHU et al., 2005). A resource could be a service, application, device or any other component. Requirements are sets of specifications or characteristics to which the needed resource must comply.

Many resource discovery systems exist today with different purposes and design.[11] However, when applied to ubicomp, these existing approaches have some limitations, such as interoperability, integration to user, and scalability (FRIDAY, 2004; ZHU et al., 2005). We desire a system with no need for manual or static configuration, which can find required resources in every environment at any time.

Besides this dynamicity, we must avoid centralized solutions. The solution could be using multiple resource providers, in a distributed fashion, or peer-to-peer (hereafter referred to simply as P2P) approaches. In this last solution, there are direct communications among nodes without the intermediation of centralized servers. An important characteristic of P2P is self-organization, i.e. the capacity of dealing with failures, variable quantities of nodes, and network variations (ANDROUTSELLIS-THEOTOKIS and SPINELLIS, 2004). Although P2P is suitable for ubiquitous

[11] A survey of resource discovery systems can be found in Vanthournout et al. (2005) and in Edwards (2006).

computing, existing systems are limited to file sharing and not general enough for resource discovery (VANTHOURNOUT et al., 2005).

Below are some of the most important characteristics of resource discovery in the ubicomp area (FRIDAY et al., 2004):

- *Location-awareness*: pervasive applications execute in the physical world and because of that, the pinpointing of resources is essential to discovery mechanisms. This mechanism should locate resources near to the user, in the same context. A resource that matches many of the requirements but is distant is useless to the system;
- *Temporal elements*: to aid the discovery of services, it is possible to associate usage profile with resources. With the history and preferences of the resources, usability can be improved. This can be achieved by finding the most suitable resource in each context for that specific user;
- *Resources states*: the discovery resource system must deal with the dynamic states of resources. Besides meeting specifications and location, states are central requirements. The representation of states, as well as their dynamic changes, must be covered;
- *Security and control*: Many resources need authentication and controlled use. Resource discovery mechanisms should exploit resources without user intervention, to maintain invisibility, but with certain constraints. Whenever possible, it must prevent malicious actions and automate authentication.

3.3.8 Context Management

By detecting context, we can affect system behavior. This change can be made by adapting the system to the new conditions or augmenting the available resources to compensate for the lack of some feature. Another possibility is changing the context by the use of actuators, i.e., software-controlled devices that affect the real world. An actuator can activate a device; alter a physical condition, such as temperature or luminosity; or execute a logical action (load code, alter parameterization, move components, etc.). To support this management, we need abstract interaction elements in design time. These elements can also be used during execution, according to context.

The two most important characteristics of this issue are adaptation and cyber foraging. They will be described in the next two subsections respectively.

3.3.8.1 Adaptation

Adaptability is a central concept in pervasive computing. Adaptation consists in adjusting aspects of applications to changes in operating environments.

Charles Darwin originally formulated the concept of adaptation in the context of natural selection.[12] It is defined as a process that makes species better at surviving. Piaget, in his developmental theory, states that knowledge development was a

[12] A detailed description can be found in The Origin of Species by Charles Darwin available on-line at <http://www.gutenberg.org/etext/2009>.

biological process, and consists of an adaptation by an organism to an environment, as previous asserted by Darwin. Piaget defined adaptation as a process of assimilation and accommodation (PIAGET, 1971).

Based on Piaget's model, Costa and Dimuro (2005) applied the concept of adaptation to describe how machines adjust to environments: "Adaptation is the process of self-regulated adjustment of internal and external operations of the computing machine to the possibilities and constraints determined by the environment."

This adaptation concept involves assimilation and accommodation, as proposed by Piaget. The former is the processes of applying currently available operations to internal and external objects, while the latter is the ability to adjust this set of operations to make them applicable in those objects (COSTA and DIMURO, 2005).

Satyanarayanan (1996) defines three strategies for adaptation. In the laissez-faire approach, the individual applications are responsible for adapting. There is no system support. On the other hand, the system could be totally responsible for its own adaptation. This approach, called application-transparent, permits existing applications to continue working in a mobile environment without modifications. The intermediate approach is called application-aware adaptation. This means collaboration between the system and the applications. Applications are free to decide how to best adapt, while maintaining system ability to enforce resource allocation decisions and monitor resources.

Application-aware adaptation is probably the best-suited strategy for pervasive computing. It mixes programming with automatic adaptation from the system. Adaptation at programming level could be more easily achieved with Aspect Oriented Programming (AOP). The idea in AOP is specifying separately the concerns (properties) of a system and leaving its composition to the environment. This facilitates programming, since scattered concerns can be treated together as aspects and not hierarchically as with OOP (ELRAD et al., 2001). Different aspects can be attached to or detached from components, facilitating adaptability. In some AOP systems, there are even some constructors for dealing with unexpected changes (PACE and CAMPO, 2001).

The most common use of adaptation is in resource-aware applications, when there is a significant difference between resources presented in the environment and those needed (AUGUSTIN et al., 2002). These resources could be, among others, network bandwidth, energy, storing space, or computing power. There are some approaches to resource adaptation: fidelity reduction, quality of service (QoS) systems, or the suggestion of corrective actions (SATYANARAYANAN, 2001). The first method consists in changing the application to a minimal use of limited resources. The second keeps a certain resource at a satisfactory level. The last one relies on user intervention to make the desired resources available.

Adaptation is important to other kinds of applications besides resource-aware ones. This gives rise to three other types of applications (AUGUSTIN et al., 2002):

- *Location-aware applications* need to consider physical location. This is not only important in resource discovery, but also when adapting. Location-dependent actions could be made. Location is a key point in determining the

context of an application. Therefore, this category can be considered as a subset of the next one;

- *Context-aware applications* use sensors or monitors to infer state and better choose an adaptation strategy. These states describe information related to the capabilities and preferences of the user, location, devices, and the environment in general;
- *Situation-aware applications*[13] use the most general form of adaptation. These applications, perceive other near applications and their context of usage. Adaptation takes place depending on usage context and user preferences. Situation-aware applications are different from the previous approach, because adaptation decision is made externally to applications.

3.3.8.2 Cyber foraging

A special case of adaptation is cyber foraging. Mobile devices usually have limited capabilities, such as processor power, memory, and battery life. With those constraints, it is sometimes difficult to satisfy the user's computational needs. To minimize this problem, we can use near machines as computing and data-staging servers, thus augmenting capability (SATYANARAYANAN, 2001). Cyber foraging means sharing or dividing code or data among servers and mobile devices, which middleware can automatically do, during load and execution time. Alternatively, it could be user-initiated – for instance, when anticipating changes in connectivity or exchange of device.

Servers used to augment capabilities of mobile devices are sometime called *surrogates* (GARLAN, 2002). These surrogates may employ encryption algorithms in stored data. Thus, the users of these servers cannot access information saved there.

3.3.9 Transparent User Interaction

We should design device-neutral applications, i.e., we should not start with the presentation and then build up the programming logic from that (BANAVAR and BERSTEIN, 2002). To accomplish this, during design time we can define abstract user interfaces and predict different types of interaction, so that the decision of which interface to use can be postponed to execution-time. Another option is to dynamically generate the interfaces during execution, based on the abstract definitions, specific devices features, and contextual information. This option requires less effort during design, and tends to consume more processor power and communication latency during execution. However, it facilitates the use of contextual data.

The generation of interfaces suited to each specific device is one of the characteristics towards achieving transparent user interaction. These interfaces must consider the most natural form of interaction for those specific devices, and also contextual information and user behavior (preferences, history needs, etc.) (CANNY,

---

[13] This is actually another way of naming pervasive applications. In the context of this work, it mixes context-awareness and invisibility issues. For instance, capturing user intent is required.

2006; NYLANDER et al., 2005). For example, speech recognition is one of the best interfaces for cell phones because they have small screens and tiny buttons and are optimized for voice communication (CANNY, 2006).

A broader concept would not focus only on the human-computer interface of devices, but rather on designing the physical interaction itself. This idea leads to tangible interaction and its use in the scope of ubicomp (HOLMQUIST et al., 2004). The proposal is to create a richer interaction experience, by coupling digital information with physical artifacts, using the human body as an interface and combining real objects and devices with computers in interactive spaces (HORNECKER, 2005). The challenge consists in creating interfaces seamlessly integrated with the real world, and considering social, personal, and emotional human experience (ROSS and KEYSON, 2007). Finally, to achieve a proper transparency, people should be able to focus on their task intuitively, and to get minimally involved with system issues.

3.3.10 Invisibility

The first step towards an invisible system is to design adaptable applications. We need framework support that eases this development, following the goals of disappearing computing and of keeping the user focus on the task. At runtime, we require uninterrupted use, with minimal user intervention. For instance, disconnection periods could occur in mobile devices. Actually, the system must mask this disconnection, by keeping services uninterrupted, and still satisfy the user's needs, maybe with some degradation.

An important characteristic towards invisibility is seamless integration, i.e., the transparent association and cooperation of various components. The idea of components that interoperate with each other seamlessly requires much effort from the middleware and careful development of each system element, considering many aspects presented on the other layers of the architecture proposed. Banavar et al. (2002) propose a task-based model that links the abstract interaction to the application logic. This model facilitates integration, since tasks are highly abstract, and can be used at load- and runtime to compose with other applications, services, and capabilities available in the pervasive environment. This can bring the notion of a *task-aware system* (SOUSA et al., 2006).

To be invisible during runtime, a system must act unobtrusively, meeting the user's expectations. It also needs minimal human intervention. Saha and Mukherjee (2003) affirm that "humans can intervene to tune smart environments when they fail to meet user expectations automatically." The system should not only respond to actions initiated by users, but also anticipate users' needs, in a non-intrusive way, by capturing their intent. Preserving user attention is another characteristic that has to be considered. Users are the most important resource in a system, (GARLAN et al., 2002) and keeping their focused on the task can foster invisibility. Invisibility is the most difficult characteristic to be obtained in a ubicomp system.

Invisibility is the most difficult issue to be obtained in a ubiquitous computing system. We are still far from reaching a truly invisible system that fulfils Weiser's vision. Some authors are even skeptical about reaching this feature and propose some solutions near to our reality today, such as engaging computing (ROGERS, 2006). In

this, instead of making the surroundings proactive and smart, the goal is in engaging people more actively in their actions by consciously acting upon the environment.

3.4 Related Architectures and Systems

In this section we present an individual analysis of projects targeting pervasive computing. It is not our aim here to give a complete description of all proposed software infrastructures, as a large amount of work in ubicomp area is still under development at present. We focused our examination on selected infrastructures that are particularly relevant to our work, because of their broader approach. Some commercial initiatives were not described, due to the lack of information and published papers. As a consequence, only academic projects are investigated.

For each of the projects presented here, we give a general structure description detailing objectives, implementation, architecture, and existing applications. At the end of this section, we describe the ISAM project. Based on this last project our work is build.

3.4.1 Aura

Aura proposes infrastructure and services specifically designed for pervasive computing. The project focuses on the user's attention, which is considered a scare resource. It tries to minimize users' distractions by adapting to context and to theirs needs (GARLAN et al., 2002). The main motivation behind this is the fact that human attention is the most limited resource in computing and not hardware resources, such as processor speed, main memory, network bandwidth, and disk capacity.

There are two important concepts in Aura Project: proactivity and self-tuning. The first deals with anticipation requests from the users and other system layers. *Prism* (SOUSA et al., 2006) is a system component that maintains the representation of user intent and provides this proactivity at a high-level. Self-tuning means adapting the system based on demands made by the user and on the system layers. This adaptation is obtained by adjusting performance and resource usage.

Aura project is not a complete solution, but rather a set of services and applications built using basic components. Most of the services run on top of operating systems such as Linux and Windows. Parts of Aura project can be used on the Carnegie Mellon University (CMU) campus by their community. Currently, there are some C and Java API to access these services.

The architecture (Figure 3.2) uses some components created prior to Aura, such as *Coda* and *Odyssey*. Coda (LEE et al., 1999) is a distributed file system that supports nomadic, disconnected, and adaptive file access. Odyssey (NOBLE, 2000) is responsible for resource monitoring and adaptation in a file system level. Another basic component of Aura architecture is Spectra. It is responsible for remote execution and uses context to decide how to optimally execute a remote call (GARLAN et al., 2002).

Some service layers where added to the basic components (GARLAN et al., 2002). A *Wireless bandwidth advisor* was created to estimate future available network throughput. This can be used to make decisions about the best server or place to access the network. There is also a service for people location. Based on signal strength and

access point information, the *People locator* can physically locate users. Another service created to amplify the capabilities of resource-limited clients, such as PDAs, is *Cyber foraging*. This service is responsible for cyber foraging and can be used for computing and data-staging servers.

Prism is atop the multilayered structure of Aura. It even executes above running applications and captures and manages user intent. Prism creates a task layer to explicitly represent user intent. At this level, users specify their activities and goals and it is up to the system to map this into available capabilities (SOUSA et al., 2006). Services suppliers are provided to offer these capabilities. Besides this task layer, there are two other components in Prism infrastructure: the environment management and the environment layer. The first addresses resource monitoring and adaptation, while the latter includes the applications and devices that can be used.

Some applications were developed using Aura infrastructure (GARLAN et al., 2002). Portable Help Desk (PHD) is an application that mixes physical location with scheduling and personal information. It displays the CMU campus map and points out the location of people and resources on it. PHD uses both a GUI and an audio interface. Another application is Idealink. It is a virtual collaboration environment that provides a shared distributed whiteboard. There are graphic and text tools to interact with the whiteboard. Idealink also supports multiple simultaneous sessions.

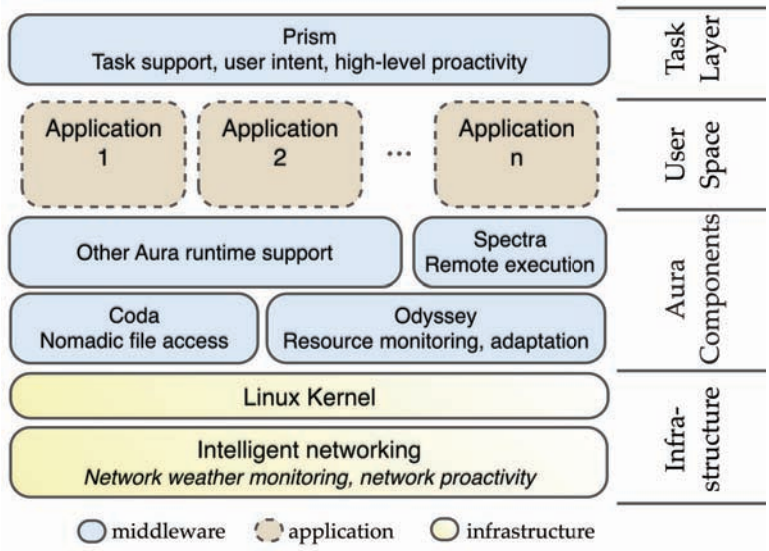

Figure 3.2: Aura architecture (GARLAN et al., 2002)

3.4.2 Gaia

Gaia is a distributed middleware infrastructure for *active spaces*, i.e., physical environments, with ubiquitous computing devices, aware of their resources and conditions (RÓMAN and CAMPBELL, 2000). The project, in development at University of Illinois at Urbana-Champaign (UIUC), focuses on the support of applications execution in these active spaces. To accomplish this, a metaoperating system is proposed, called *GaiaOS* that manages software and devices. Gaia was created primarily as an attempt to achieve an optimal functionality of the integrated services rather than their individual capabilities (RÓMAN et al., 2002).

The idea of a metaoperating system brings ubicomp the same features an ordinary operating system provides a personal computer, simplifying management and application development. In Gaia, these should be user-centric, resource-aware, multi-device, context-sensitive, and mobile applications (RÓMAN et al., 2002). To achieve this, common operating system services are provided together with new features targeting pervasive computing, such as context and location awareness.

There is a prototype developed in a specific room at UIUC.[14] The room is equipped with various devices, such as touch-screen displays, a projector, plasma displays, wireless networks, and badge detectors. Gaia was implemented using CORBA for distributed object interaction. Some extensions were also implemented to deal with soft state, dynamic resource detection, and fault tolerance (RÓMAN et al., 2002). To program in Gaia, a high-level scripting language named LuaOrb is used, which is a binding between Lua and CORBA, COM and Java.

Gaia architecture is organized in three layers (Figure 3.3). From bottom to top, the layers are: the kernel, the application framework, and the active space applications (RÓMAN et al., 2002). The component management core and a set of basic services form GaiaOS kernel. The core is responsible for dealing with components and applications, performing tasks such as dynamic loading, unloading, transferring, creating and destroying applications. There are five basic services provided by the Gaia kernel: the *space repository* stores information about hardware and software; the *event manager* is a mechanism used to expose changes in components state; the *context file system* is a file system that uses application-defined properties and context; the *presence service* detects and maintains information about components, devices, and software; and, finally, the *context service* queries and registers context information.

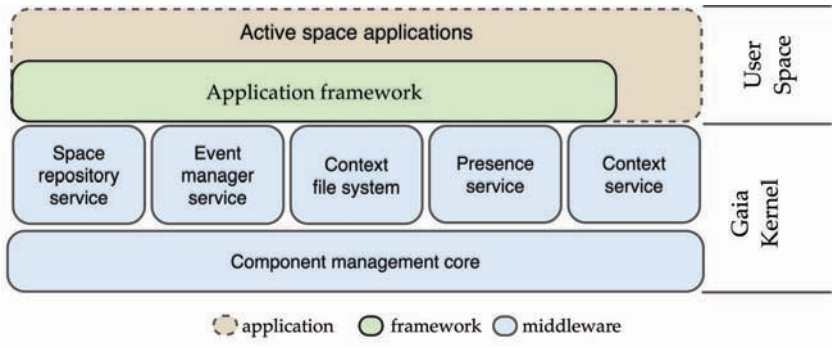

Figure 3.3: Gaia architecture (RÓMAN et al., 2002)

The application framework allows applications targeting active spaces. Another objective is to facilitate the adaptation of traditional applications to Gaia. The framework is composed by an infrastructure, a mapping mechanism, customization policies, and Model-View-Controller (MVC) extensions. The main components to implement any application are defined in the *infrastructure*: the model (application logic), the presentation, the input sensor, the controller (which maps input sensor events into requests for the model), and the coordinator (which manages all previous components). The *mapping mechanism* customizes a generic application to a specific

[14] More information can be obtained at <http://gaia.cs.uiuc.edu/>.

environment, considering component requirements. Applications should rely on *customization policies* to address issues such as mobility and adaptation. The traditional MVC model is then reused and extended. Active space applications execute atop this framework.

A *Presentation Manager Application* (RÓMAN et al., 2002) was developed using Gaia. The application manages slide presentations in various displays and with many different input devices. The user can easily move or duplicate presentations through many different devices and input sensors. For instance, the user may control a presentation in different output devices, such as plasma displays or projectors, with a PDA.

3.4.3 One.World

Project One.World (GRIMM et al., 2004) is a complete solution for developing adaptable applications. The focus of One.World is "to provide an integrated and comprehensive framework for building pervasive applications" (GRIMM et al., 2004).

The main motivation behind the project is based on the assumption that distribution must be explicit for the development of pervasive software. In this manner, applications can detect changes and adapt to them. The system is based on three requirements (GRIMM, 2004): embracing contextual change, encouraging ad hoc composition, and recognizing sharing as default.

One.World executes on top of existing operating systems, such as Linux and Windows, and is mainly coded in Java. Besides Java, it uses some native libraries and the Berkley DB to implement tuple storage. There is a fully functional prototype of the system for free download and use.[15] Currently, the prototype lacks support for transactions and for automatic distribution of code among devices (GRIMM et al., 2004).

The architecture of One.World (Figure 3.4) is based on three layers: foundation services, system services and library support. There are four foundation services: the *virtual machine*, to provide heterogeneity to the system; *asynchronous events*, to deal with all communications in the system; *tuples*, allowing data sharing and creating a common data model; *environments*, which are mechanisms for composing, isolating and storing applications.

The system services are created on top of the foundation services. *Migration* provides mobility, offering the possibility to move or copy environments with all their contents (applications, data and nested environments). After migration, the system can use the service *discovery* to find local or remote resources. Another system service is *tuple storage*. Also related to tuples is the service query engine, which lets the system search data. Tuples become accessible in environments through the system service *structured I/O*. This service provides the main operations to manipulate data. *Remote events passing* is the final system service. It is responsible for forwarding events to remote services.

---

[15] One.world can be found at <http://www.cs.nyu.edu/rgrimm/one.world>.

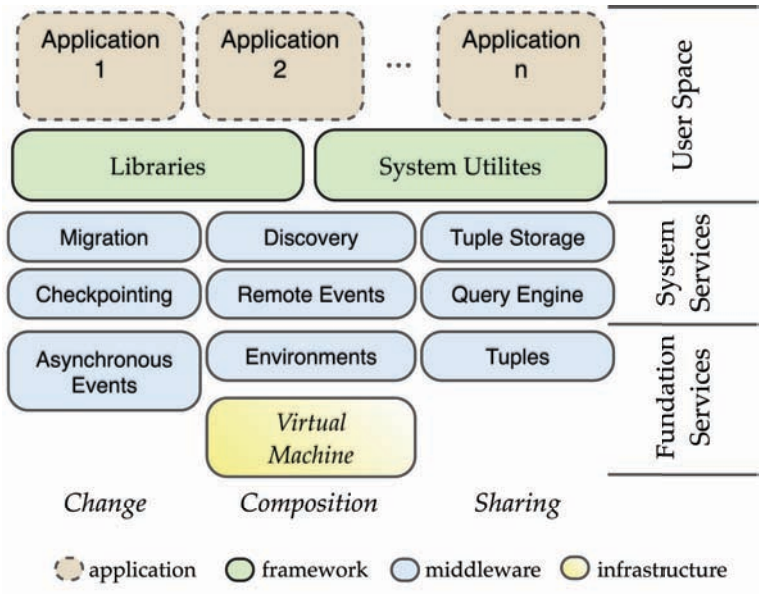

Figure 3.4: One.World architecture (GRIMM et al., 2004)

User-level libraries offer additional support in the One.world architecture. They include the logic/operation pattern, support for the development of user interfaces and for the timed execution of events (GRIMM et al., 2004). The logic/operation pattern splits computations that can fail (called *operations*) than other computations (named *logic*). Operations can be used for failure detection and for further recovery.

There are some applications built atop One.World system. Emcee is the user and applications manager. It allows the management of users and their applications, checkpointing and mobility among devices. Another application is Chat, which is a messaging system that supports audio and text. Perhaps the most interesting developed application is Labscape (ARNSTEIN et al., 2002). It automates a real biology laboratory, making experimental data follow researchers as they move. Labscape also stores all data in a central repository and can collect experimental data using *radio frequency identification badges* (RFID) and barcode scanners.

3.4.4 ISAM

ISAM is a Brazilian acronym for *Infra-estrutura de Suporte às Aplicações Móveis* (Mobile Applications Support Infrastructure), developed by researchers from Federal University of Rio Grande do Sul (UFRGS). The project aims at integrating the concepts of context-awareness, grid, and mobile computing (YAMIN et al., 2003). The idea behind ISAM is to build a pervasive computing infrastructure, integrating a programming language and middleware to support its execution.

Differently from other proposals, ISAM focuses on application development rather than on the environment and services. Because of that, the project encompasses a model, a language, and a runtime support to build and execute pervasive applications. There is a prototype available, built mainly in Java, with some modules in C.[16] The

[16] ISAM can be downloaded at < http://www.inf.ufrgs.br/~isam>.

prototype is fully functional and bundled to a Linux live CD, in order to facilitate its use. Programmers can develop applications utilizing Java.

To further facilitate the development of pervasive applications, ISAMadapt (AUGUSTIN et al., 2004) was defined as a framework for a programming language. It provides some means for expressing dynamic adaptation and context-awareness in design time. ISAMadapt uses some concepts of a multiparadigm model named Holoparadigm (hereafter simply referred as Holo) (BARBOSA et al., 2005). In Holo, a logic blackboard, called *history*, implements the coordination mechanism, and a new programming entity, called *being*, organizes several encapsulated levels of beings and histories (multi-domains). These "beings" are the main Holo abstractions. They represent the logical or physical components of the system that is modeled.

The architecture of ISAM (Figure 3.5) is organized in three components: the infrastructure layer, the intermediate layer, and the superior layer. The infrastructure consists of the network, the operating system, and the Java Virtual Machine (JVM). Currently, ISAM programs are developed in Java source code, then compiled, and executed in the JVM.

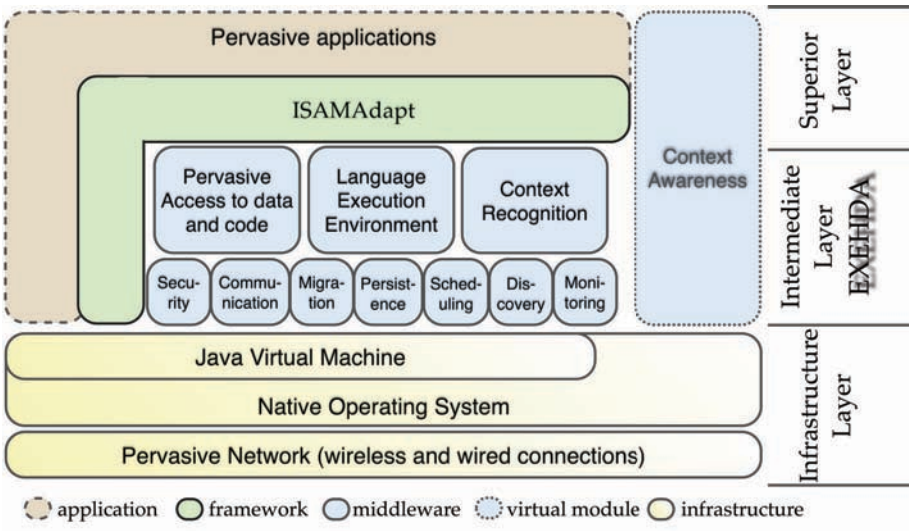

Figure 3.5: ISAM architecture (YAMIN, 2004)

The intermediate layer is the Execution Environment for Highly Distributed Applications (EXEHDA). Designed as middleware, it consists of a collection of services, such as naming, communication, migration, replication, interoperability, location, and monitoring. On top of these basic services, EXEHDA executes the User Virtual Environment, a container for user applications and sessions; the Scheduler, for migration and remote execution of objects; and the Context Server, for context-aware adaptive behavior. The superior layer has the ISAMadapt and the distributed mobile applications. Context awareness is represented as a virtual module, since it is present in the conception of all other ISAM components (YAMIN, 2004).

Applications run on the ISAM pervasive environment (ISAMpe), which uses cellular hierarchy. Each *cell* has a specific host, called *base*, responsible for communications among cells. Devices belonging to the same cell can directly

communicate with each other and are identified as *nodes*. The hierarchy allows a cell to recursively contain other cells.

Applications were implemented using ISAM. WalkEd (AUGUSTIN et al., 2004) is a text editor that may be used both on desktops and mobile wireless devices. It follows the user and adapts to the environment. For instance, when the user switches from the desktop to a PDA, WalkEd migrates to the PDA and changes its presentation to the interface available. Another interesting application is GeneAl: A Grid Approach for Genetic Sequence Alignment (SCHAFFER FILHO et al., 2005). GeneAl's objective is to find the best N alignment among biosequences spread in distributed databases. It uses ISAM's dynamic discovery services to find databases during execution. Another characteristic of adaptive execution is the dynamic selection of works.

4 CONTINUUM SOFTWARE INFRASTRUCTURE

In this chapter, we propose Continuum software infrastructure, which is an evolution of project ISAM (section 3.4.4) based partially on the requirements offered by the comprehensive architecture model (sections 3.2 and 3.3; COSTA et al., 2008), and partially on context awareness considerations. The latter, including the Context Awareness subsystem and services, will be detailed in the next chapter.

4.1 Continuum as an Evolution of ISAM and EXEHDA

Continuum (COSTA et al., 2007) is proposed as a service-based software infrastructure for ubiquitous computing, integrating frameworks and middleware. The main focus of our work is context awareness, so that the environment encompasses the characteristics that the user needs, enhancing the real world. The project is based in a redefined view of follow-me semantics: users can go anywhere carrying the data and application they want, which they can use in a seamlessly integrated fashion with the real world.

The project is based on ISAM, but also considers the challenges requirement by the comprehensive model (section 3.2). Differently from ISAM, Continuum is not tied to a particular language; neither does it propose new changes for existing languages. That is due to the fact that programmers usually prefer to use languages they are already familiar with, allowing them to use legacy codes. Additionally, we want to support the context awareness without excessively burdening the programmer and the software development process. We propose the use of a framework, instead of a language, to achieve these goals. The Continuum framework should maintain all the characteristics we consider important to support the design time development (see Figure 3.1). Furthermore, the framework could inherit some important characteristics from ISAMadapt, in an independent language approach.

Another difference from ISAM is in the redefined view of follow-me semantics. In ISAM, we are most focused on a *desktop vision*, in which users' applications and data, available in their desktop, can be accessed everywhere, with assorted devices, providing the idea of a virtual environment (named AVU – virtual user environment – in ISAM). In Continuum, the idea is to provide services to help the users carry the software components that they want. Also, we do not intend to provide a virtual environment, but rather to use those components to enhance the real world in a seamless way.

ISAM is based on a vision that is more pervasive than ubiquitous, considering the differences that were significant at the time of the project development. Because of that,

the higher level services proposed in the comprehensive architecture model (related to invisibility and transparent user interaction) are not tackled in project ISAM.

Incorporated in Continuum are various services that had already been available in EXEHDA; others have been redefined; and new ones have been proposed. As the focus of Continuum is on context awareness, we propose the rebuilding of the EXEHDA context recognition and adaptation subsystem. The original project (YAMIN, 2004) focuses only on partial context awareness, i.e. the obtaining of raw information, its distribution, and the conversion of raw information into abstract context elements, guided by an XML description. Another important element in ISAM is informality in the treatment of context. A more detailed description of context awareness in Continuum will be given in the next chapter.

4.2 Software Architecture

Figure 4.1 illustrates the proposed development process using Continuum. The left side of the figure shows the environment and the support during design time. It comprises an Integrated Development Environment (IDE) for the implementation of ubiquitous applications. The IDE encompasses a language API, a set of development tools (compiler, editor, linker, debugger, etc.), the Continuum framework, and other application frameworks as needed. In this environment, we can build the ubiquitous application source code. The right side of the figure presents, in general terms, the components required during execution: the application binaries, the Continuum framework and middleware, and the platform necessary to execution (network, computer, operating system and additional running support). We identify this runtime environment as Continuum software architecture.

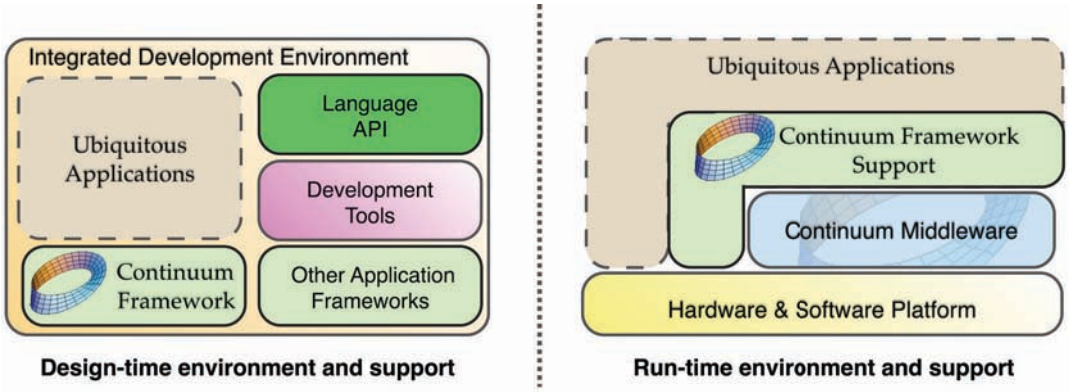

Figure 4.1: Continuum development process

The proposition for Continuum software architecture is presented in Figure 4.2. The architecture is divided in layers: foundation, middleware (subsystems and pluggable services), and user space. The *foundation* comprises the execution and support environment, including the network, the operating system, and the language runtime support. For instance, if the application is developed in Java, this language support includes the Java Virtual Machine (JVM). The *middleware* is divided in *subsystems*, which are further divided into services. The subsystem layer is conceptual: not an element in itself, but rather a group of related services.

The *pluggable services* constitute the core of Continuum middleware and supply the main functionalities during execution. As the name implies, these services can be loaded on demand. Finally, the *user space* layer contains user applications and the Continuum framework support. Applications can use the foundation layer directly and also interact with the middleware.

The *framework* incorporates Execution Profiler support, which helps the user choose the service needed for each application. It also assists the user in selecting which services will be available in each node. The Execution Profiler parameterizes the deployment process during load time. This reconfiguration process is needed each time a node is bootstrapped.

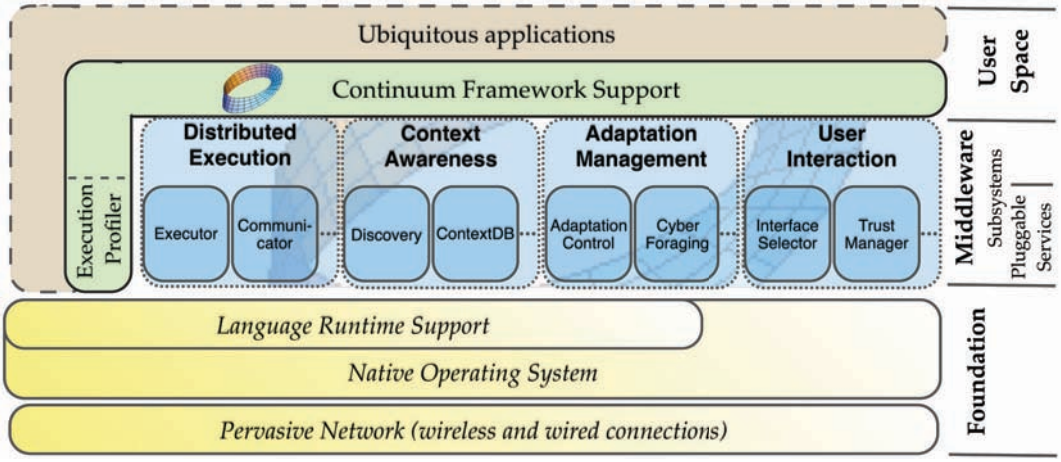

Figure 4.2: Continuum software architecture

During execution, applications may also need to use services on demand. This is done by a service in the *Distributed Execution* subsystem, which is responsible for the distributed processing support and communication in Continuum. In this component, applications are managed, services are deployed on demand, and then copied or migrated among nodes. Furthermore, this subsystem keeps the physical organization of the environment, by storing attributes related to the management of the infrastructure, i.e. resources, users, and applications.

The *Context Awareness* subsystem groups the services that deal with a variety of contextual information, in an independent application manner. The subsystem also considers user preferences (requirements that vary from user to user and over time). The Context Awareness subsystem is also in charge of storing context, along with points in time at which these data have been created, and distributing / localizing them.

Another subsystem is *Adaptation Management*. Not only does it target at the adaptation process itself, but also at the management of the adaptation process, which includes agility aspects and the maintenance of system stability (SILVA et al., 2008). On one side, we have to address the delay between the perception of a new context state and the execution of actions to adapt the system to this new environment condition; this process demands agility. On the other hand, the execution of adaptation actions has a computational cost and competes with the application itself. In an extreme case, adaptation actions can be very frequent, leading the system to a state of instability, in

which the majority of resources is consumed by the execution of these adaptation actions. This requires stability maintenance in the environment.

Finally, we have the *User Interaction* subsystem. Services in the subsystem are in charge of reinforcing invisibility issues, giving special consideration to user attention and intent. The main features of this subsystem are to provide ubiquitous access to files, to deal with trust and privacy, to supply the choice of an interface, and to help with invisibility issues, more specifically to ensure user attention, to meet user intent, and to cause minimal user intervention. In the latter functionality, interfaces suitable to each type of device or environment could be selected. To accomplish this, during design time we can define abstract user interfaces and predict different types of interaction, with the aid of the framework, so that the decision of which interface to use can be postponed to execution-time.

As already pointed out, these subsystems are only a conceptual organization; in practice, Continuum uses a service-based organization, which selects services on demand, depending on what functionalities the applications need. These pluggable services add an adaptive behavior, which is important due to the high heterogeneity of the many different resources. In addition, Continuum proposes the use of Service-Oriented Computing - SOC (PAPAZOGLOU and GEORGAKOPOULOS, 2003). In SOC, the service layer follows the service-oriented architecture (SOA). The purpose of SOA is to support critical applications, which require the management and deployment of services and applications; it is also targeted at providing support for open services (PAPAZOGLOU and GEORGAKOPOULOS, 2003). The application of SOC on the web is obtained by the use of web services. SOC, SOA, and web services create a general interface, which makes interaction easier in Continuum; in a more ad hoc approach, those elements enable many applications to make effortless use of its services.

Besides being selected on demand, the services are context adaptive, i.e., the infrastructure is able to use the implementation that is better tuned to each device. Furthermore, we reduce resource consumption by selecting only services that are actually necessary. Such scheme is possible because services are defined by their semantics and interface, instead of a specific implementation. Moreover, it is easy to add other services, since we make use of SOA architecture. In section 4.6 we present the services proposed for Continuum, organized by subsystems.

4.3 Modeling the Physical World in Continuum

We propose a model to abstract the entities that will be used in applications developed with Continuum. As our focus is on context awareness, we choose a model that considers the three entities that can be distinguished when dealing with context (according to DEY et al., 2001): places, people, and things. Since the proposed model is an abstraction of the world, there is no need to model everything, but rather represent only the entities of interest.

We name the physical abstraction of the involved entities in a given Continuum application as a *CoDimension*. From an infrastructural viewpoint, a CoDimension is the physical organization of a set of devices, in which resources and services are managed by the software infrastructure. This is based on the previous ISAM pervasive

environment – ISAMpe (YAMIN et al., 2003; YAMIN, 2004). As in the original conception, cells form the topology, which are the union of several mobile and stationary physical resources in the network infrastructure. Differently from the original concept, our proposal is broader and incorporates the representation of entities not considered at that time. Also, we include the concept of composition, which was a requirement of the comprehensive model (section 3.3.5).

A CoDimension is organized in *CoCells*, each one representing a place. The degree of abstraction of a cell can vary according to the application being developed. For instance, we can represent as a cell a city, a room, a building, an institution, or the entire earth. A CoCell could encompass other CoCells, benefiting from the composition.

People transit among cells. In Continuum, they are identified individually as *CoPerson*. They can be physically present in one cell at a specific time. Things, on the other hand, can be stationary (always in the same cell) or mobile. We call a thing in Continuum a *CoNode*. Usually, we are particularly interested in representing CoNodes that are computers, sensors, devices, or other electronic components.

There are special kinds of CoNodes in Continuum. The CoCell's internal organization might contain a base node (named *CoBase*) and component nodes, denoted as *CoNodes* for stationary devices and *CoMobis* for mobile ones (the term device will hereafter be used to refer to a generic host in the system that could be a CoBase, a CoNode, or a CoMobi).

A CoBase is in charge of all the basic Continuum services for a specific CoCell or a CoCell and the inner cells, although some services could be distributed among other devices for scalability issues. This base node executes the middleware and makes pluggable services available to other nodes.

CoNodes represent nodes that execute users' applications. They are processing units presented in the infrastructure, which execute the Continuum middleware. CoNodes use pluggable services as needed. CoMobi, on the other hand, represents a special kind of CoNode that is mobile, typically wirelessly connected and with a more restricted capacity, generally in terms of network latency, processing speed, available memory, and power supply.

The cell's external organization is based on peer-to-peer association among cells. Accordingly, we make use of a super-peer organization (ANDROUTSELLIS-THEOTOKIS and SPINELLIS, 2004). Generally speaking, the idea is to have only one dimension in Continuum and embrace all the cells in a hierarchical super-peer organization (GARCÉS-ERICE et al., 2003; BISCHOFS et al., 2004).

A super-peer organization is very similar to the idea of P2P, except that not every node in the system is a peer. Only super-peers act as a peer would, in a traditional P2P system; other nodes act as clients and are connected to a single super-peer only (YANG and GARGIA-MOLINA, 2003). Compared to traditional P2P systems, super-peers introduce advantages from both the centralized client-server model, such as efficiency, and from distributed search, such as autonomy, load balancing, and robustness (BISCHOFS et al., 2004). Each CoCell in our work is associated with a super-peer, normally its own CoBase, and the other devices in the cell act as clients.

There is a natural hierarchical relation among entities. This is true both for inter-cells (among CoCells) and for intra-cells (among nodes belonging to a cell). For

instance, a house is situated in a neighborhood, which is located in a city, associated with a state belonging to a country. Considering a house as a cell, we could have rooms with appliances and equipments. Besides being natural, hierarchical organization also allows a general improvement in scalability (GARCÉS-ERICE et al., 2003). Another advantage of this organization is to employ a hierarchical look-up service to assign and locate resources in the super-peer network, instead of the most common distributed "flat" strategy used by many existing P2P middlewares (BISCHOFS et al., 2004).

There are three types of a relationship that a device can establish with the cell (this is loosely based on BISCHOFS et al., 2004):

- **Aggregation**: describes a close cooperation between a device and a cell. The device constitutes a part of the cell. A device connected with this relationship can move among cells;
- **Composition**: describes a device that is a constituent part of a cell. The cell could not exist without it. This type of device cannot be moved and it is always attached to the same cell. For instance, CoBase nodes have always this type of connection with the cell where they are located;
- **Association**: describes a loosely coupled relation, in which, from an association relationship, no hierarchy is clearly observed.

When considering cells, the only possible relationship is composition, since a place can always be inside another place. The most general cell is in the dimension. People, on the other hand, are always associated to cells.

To illustrate our proposal, we present a sample dimension in Figure 4.3. We developed a notation, partially inspired by UML (Uniform Modeling Language), to facilitate the visual representation of entities and their relationship. A cloud represents a CoDimension and an empty oval a CoCell. Drawing one entity inside another represents composition among those types of entities. The same is true for the relationship between a CoPerson, which is represented by an oval with an actor inside, and a CoCell. Since nodes could have different kinds of relationships, there are three distinct representations for those. We used in this case, the same equivalent in UML to represent aggregation, composition, and association among classes.[17] There is also a special notation for CoBase, CoNode, and CoMobi, as seen in the figure.

In our example, the outermost cell represents a city. Inside it there are only two cells represented: a specific neighborhood and a workplace. Observe that there is no need to model all the neighborhoods in a city, but only those of special interest for our applications. Additional cells can be modeled during the execution of an application. They can be added to any level of the hierarchy. For instance, if we want to model the neighborhood in which our work is located, it can be inserted inside the CoCell that represents the city and as the external cell of work.

Observe that in the notation, for simplification and better visualization purposes, some relationships are not shown: among cells, because the only possibility is composition; between cells and a CoDimension, since the relationship is always composition; concerning a CoPerson and a CoCell, as persons are always associated to

[17] More about UML can be found in FOWLER (2005).

cells; and, between a CoBase and the CoCell in which it is located, because, in this case, the relationship is always composition. Other relationships are always presented in the notation. For instance, the figure shows the aggregation between a CoNode and the "Work" CoCell. Another consideration about the notation is that when a CoBase is not represented inside a CoCell, the next outer cell with a CoBase takes over.

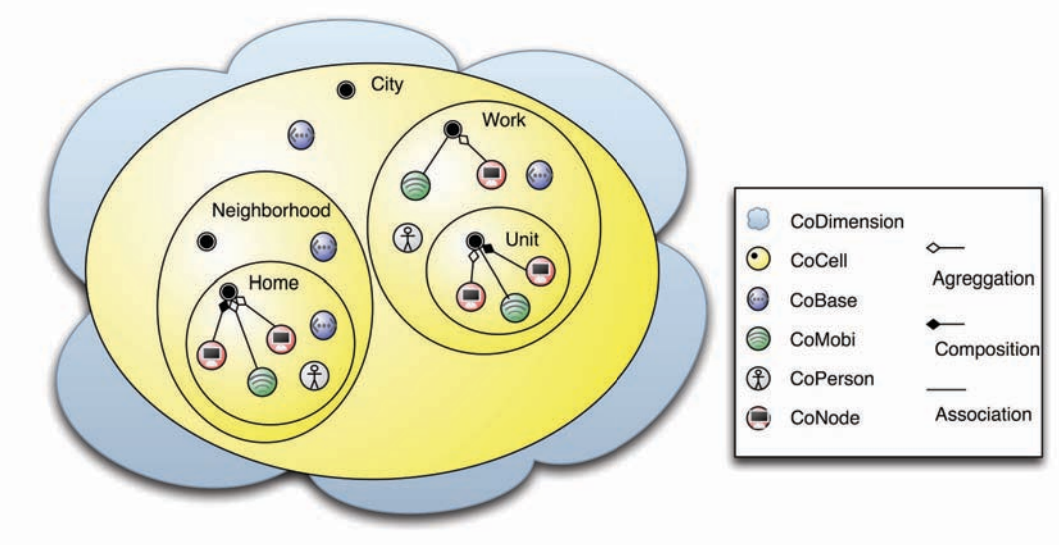

Figure 4.3: A sample CoDimension

The physical organization proposed is in agreement with the scalability requirements of the comprehensive architecture (section 3.3.2). We have circumvented centralized solutions by proposing a cellular organization, using hierarchical super-peer model. This means that the finest granularity at which management needs to be considered is a cell. To further avoid bottlenecks in our proposal, we foresee the possibility of CoBase replication or specialization, distributing services among different nodes in the same CoCell.[18] The physical organization of a CoDimension also improves local interactions over distant ones. This is due to the fact that nodes report to one of the cell's CoBase and preferably establish communications with nodes in the same cell. Finally, the employment of hierarchical structure aims at large-scale utilization.

So far, a limitation of ISAM has been the support of mobile devices, especially those with hardware constrains, such as cell phones or PDAs. This is because, in ISAM, EXEHDA middleware must run in each integrating node of the physical topology. The solution was to use a more restricted version of EXEHDA in these cases. This approach still presents some issues:

- The need to port the middleware to each and every device. Even using Java, this might involve various changes in the code.[19] Besides, a fully compatible JVM may not be available;

[18] This feature must be detailed in a future work.

[19] The EXEHDA middleware was only ported to the Sharp Zaurus PDA. This involved the reimplementation of EXEHDA considering the CDC profile of the particular J2ME specification for the device.

- The device memory used to store middleware, virtual machine, and the user application;
- Other devices constraints, such as processor speed, energy consumption, and network latency.

In Continuum, we consider these mobile devices with hardware constraints as a special case of CoMobi named *CoGadget*. In this thesis, we define gadget as a device that has a practical and specific purpose in daily life. Common examples are smart phones and PDAs. A CoGadget is a gadget supported by the Continuum that has a infrastructure layer different from the one shown in Figure 4.2; it also uses the middleware in a more ad hoc manner. Because of that, the only possible relationship between a CoGadget and a CoCell is association. Figure 4.4 illustrates a CoDimension with some CoGadgets. As seen in the figure, we propose a special notation for the visual representation of gadgets. Since there is only one type of relationship, we just have to draw a CoGadget inside a cell, meaning that this device is associated with it.

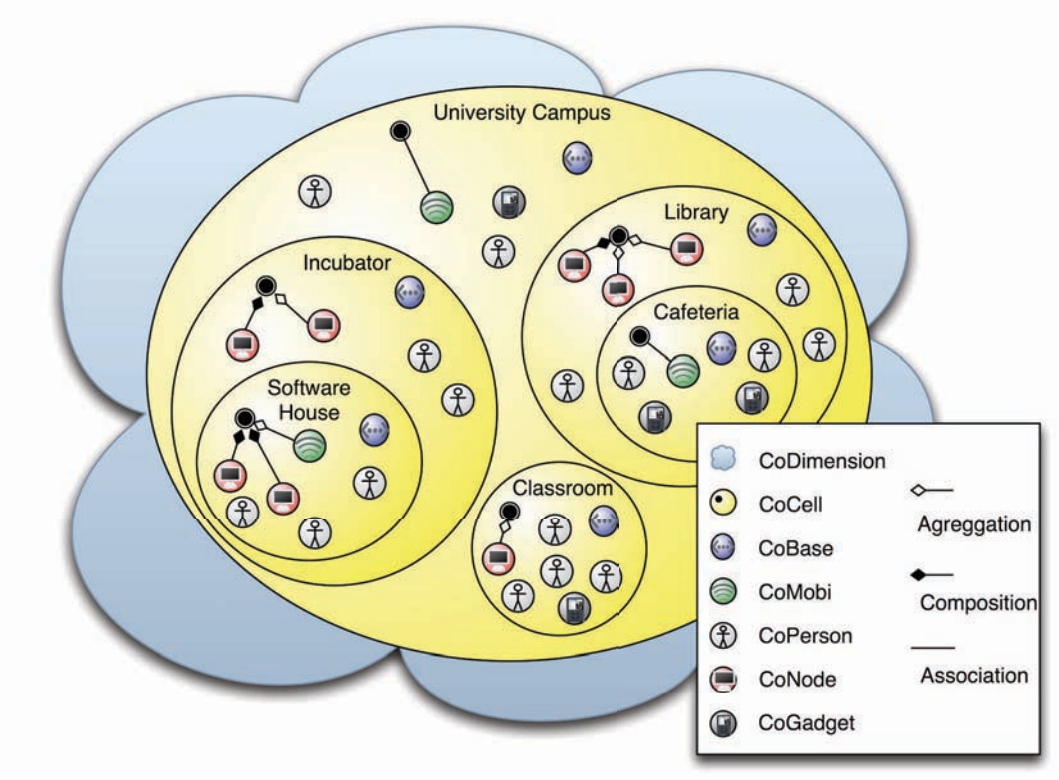

Figure 4.4: A Continuum dimension with some gadgets

The basic abstractions of Continuum are summarized in Table 4.1. The table shows each abstraction, its visual notation, the entity that it represents, and a brief description. In Table 4.2, we go over the possible relationships in our notation, presenting their name; notation; entities that they could relate to, pointing out those that are not shown in the notation; and their descriptions. The next section describes the infrastructure layer of Continuum and highlights the different vision used by CoGadgets.

Table 4.1: Continuum basic abstractions

Name	Notation	Entity	Description
CoDimension		Place	A physical abstraction of the real world embracing all modeled entities.
CoCell		Place	Represents a place and comprises nodes in that physical location. CoCells are nested forming a hierarchy.
CoNode		Thing	A device executing users' applications and making use of Continuum services. Typically a stationary device.
CoBase		Thing	A device in charge of managing a specific CoCell, or a group of nested cells, and responsible for interaction with other cells.
CoMobi		Thing	A device with mobile capacity executing users' applications and making use of Continuum services. Usually, wirelessly connected and with a more restricted capacity, generally in terms of network latency, processing speed, memory available, and power supply.
CoGadget		Thing	A device that accesses users' applications in a more ad hoc manner. In general, a special purpose device such as a smart phone or a PDA.
CoPerson		Person	A user registered in the infrastructure that is physically present in one CoCell.

Table 4.2: Continuum Relationships

Name	Notation	Relationships	Description
Aggregation		CoNode ▶ CoCell CoMobi ▶ CoCell	A tightly coupled relation between entities.
Composition		CoCell ▷ CoDimension CoCell ▷ CoCell CoBase ▷ CoCell CoNode ▶ CoCell CoMobi ▶ CoCell	An entity that is a constituent part of another entity.
Association		CoPerson ▷ CoCell CoGadget ▷ CoCell CoMobi ▶ CoCell CoNode ▶ CoCell	A loosely coupled relation between entities.

Symbols: ▶ always shown in the notation ▷ not shown in the notation (default relationship)

4.4 Infrastructure Layer

The infrastructure layer is the physical layer of system execution. It comprises the hardware, with its network interconnection, a native operating system, and the language runtime support. In the present version, the JVM is used as the runtime support for the execution of some services. This restriction is due to the fact that we have many inherited services from EXEHDA middleware, which has been developed using Java. This limitation does not imply that applications should be built using Java, because the service interface is based on web services.

Continuum was designed as independently as possible from the native operating system. This choice causes some drawbacks in our model, since we cannot obtain the full potential of each device; even if we follow the requirements of the comprehensive

architecture. This is because there is an important trade off between portability and a more thorough use of device capabilities. As a project decision, we preferred to sacrifice the latter to promote the former.

We call *pervasive network* the wired or wireless infrastructure available for the interconnection of devices. The term, as previously employed by Yamin (2004), highlights the mobile nature of devices and how they can be used with varied protocols, adapters, network speed, and throughput. Sometimes, the pervasive network may not be available at all, and this fact is dealt with in the above layers of the Continuum software infrastructure.

Although the use of the JVM clearly improves portability, it can be inconvenient in some platforms. First, it may require the installation of additional code that can further reduce the total memory available. Second, we currently have specific versions of JVM for different devices. For instance, Java Micro Edition (Java ME) is targeted at mobile and embedded devices. It is conceptually different from Java Standard Edition (Java SE). Third, JVM compatibility varies, especially in portable devices: some still support Personal Java, which is a platform for Java mobile development superseded by Java ME. A notable omission is the support for Windows Mobile based devices, since Sun does not currently provide binaries for Java. Even in Java ME, we have two different implementations: Connected Limited Device Configuration (CLDC) for resource-constrained wireless phones and communicator-type devices, and Connected Device Configuration (CDC) for more powerful devices.

Because of these JVM limitations, as well as the previously discussed lack of full mobile device support of ISAM, we decided to use a different infrastructure in CoGadgets. We propose the access of Continuum via a web browser using Web 2.0 concepts. This term was coined by Tim O'Reilly in 2005, and describes "a quickly growing set of web-based applications" (SCHROTH, 2007). O'Reilly (2005) described Web 2.0 in terms of seven characteristics: the web as a platform, exploitation of the collective intelligence of web users, ownership of data, end of the software release cycle, use of lightweight programming models that allow for loosely coupled systems, software not limited to the single device / PC platform, and richer user experience.

Web 2.0 systems should provide simple interfaces, be scalable, and produce results that are sensitive to context (LIN, 2007). Among the technologies used in Web 2.0, we chose to employ AJAX, which is an acronym for Asynchronous Java Script + XML. AJAX is one of the key paradigms used in the development of Web 2.0 applications (SCHROTH and JANNER, 2007) and a combination of some other technologies: XHTML, CSS, and the Document Object Model for presentation management; XMLHttpRequest for asynchronous data retrieval; and JavaScript for programming.

AJAX, due to its asynchronous nature, lessens the problem of network latency and gives a better system response than that of the traditional RPC. Its use in cell phones and other gadgets has become a tendency. As an example, the iPhone[20] programming model is based on AJAX, over a browser, and makes use of a Software Development Kit (SDK) provided by Apple for granting access to many of the device's features, such as e-mails, calls, and Google Maps.

---

[20] More details in <http://developer.apple.com/iphone/>.

Recently, the convergence of concepts from Web 2.0 and SOA has become a topic of discussion (HOWERTON, 2007; SCHROTH and JANNER, 2007; SCHROTH, 2007). These concepts have been considered complementary, and the use of both together brings the vision of an Internet of Services (IoS). Schroth (2007) defines IoS as a "global platform allowing both end-users and businesses to seek, combine, customize, use and publish interoperable resources." Not only does the author emphasize the need to use principles from SOA and Web 2.0, he also advocates the need to employ contextual computing and the concept of Semantic Web.[21] Our proposal aligns itself with this convergence, and reinforces the decision to collectively use all this technology in the scope of ubicomp.

The infrastructure layer was proposed according to heterogeneity requirements already presented in section 3.3.1. We used a virtual machine (JVM) to implement some services, but programmers could employ any language they want to develop applications. To circumvent JVM limitations, further improving heterogeneity, we proposed the use of Web 2.0 technology. This choice goes along with the device-independent approach suggested before, and also with the use of open standards. In the next section, we present some other characteristics that improve heterogeneity: the use of SOA for pluggable services and the employment of XML as an interoperability language.

4.5 Pluggable Services

Each pluggable service in Continuum is defined as a web service, which is the most common implementation for SOA because it uses XML for data and employs platform-neutral communications (HOWERTON, 2007). The idea of obtaining functionalities as network-delivered services corresponds to a model named Software as a Service (SaaS). Anerousis and Mohindra (2006) have defended the use of SaaS for ubicomp environments and state that the most significant challenges in this field are how to handle periodic disconnections and how to address differences in devices. They propose the use of adaptive services to solve the latter challenge and the caching of data on devices, enabling offline operations to tackle the former problem. This vision adheres with our proposition.

In Continuum, pluggable services are accessible in the infrastructure from the CoBase. Each cell has at least one CoBase that provides the services. For scalability issues, it is possible to have more than one CoBase in each cell. CoNodes and CoMobi have remote access to services (from one CoBase in the cell) by default. However, they can run some services locally to improve autonomy, allowing offline operations in case of disconnection. When this occurs, the middleware is placed in charge of further synchronizing modifications with the CoBase of the CoCell to which it belongs.

It is important to reinforce that CoNode, CoMobi and CoBase run the middleware. The selection of services in these nodes can be made statically (in the middleware bootstrapping) or dynamically, as needed by the nodes, in a deployment that is on-demand, on the fly. The Execution Profiler (described in section 4.7.1) controls the

[21] More about Semantic Web can be found in <http://www.w3.org/2001/sw/>.

static selection of services, and guides the middleware bootstrap. During middleware execution, the Adaptation Manager (described in section 4.6.2) does the dynamic selection of services.

CoGadgets do not run the middleware, and access all pluggable services remotely via a web interface, using AJAX, in a browser. In these cases, services are adapted accordingly to device capabilities, exposing only a subset of features, making it possible to use Continuum in small form factor devices.

4.5.1 Distributed Architecture for Service Support

This section describes the Continuum Distributed Service Architecture (CoDSA), which is a SOA that uses web services for communication. The entities involved in CoDSA are:

- **CoService**: a basic service in Continuum. A CoService offers well-defined functions;
- **CoProvider**: a node in the environment that offers CoServices;
- **CoConsumer**: each node, either a CoNode or a CoMobi, that uses CoServices in the environment;
- **CoDirectory**: a node in each CoCell that lists the available CoServices. This node runs the pluggable service *ServiceManager*.

Each node in Continuum, with the exception of CoGadgets, may act as a CoProvider. In addition, CoGadgets cannot be considered as CoConsumers; they are indeed a special kind of device that indirectly uses CoServices through a web interface. Figure 4.5 illustrates the CoDSA, with the entities described above, and also presents two possibilities in the architecture: CoService migration and replication.

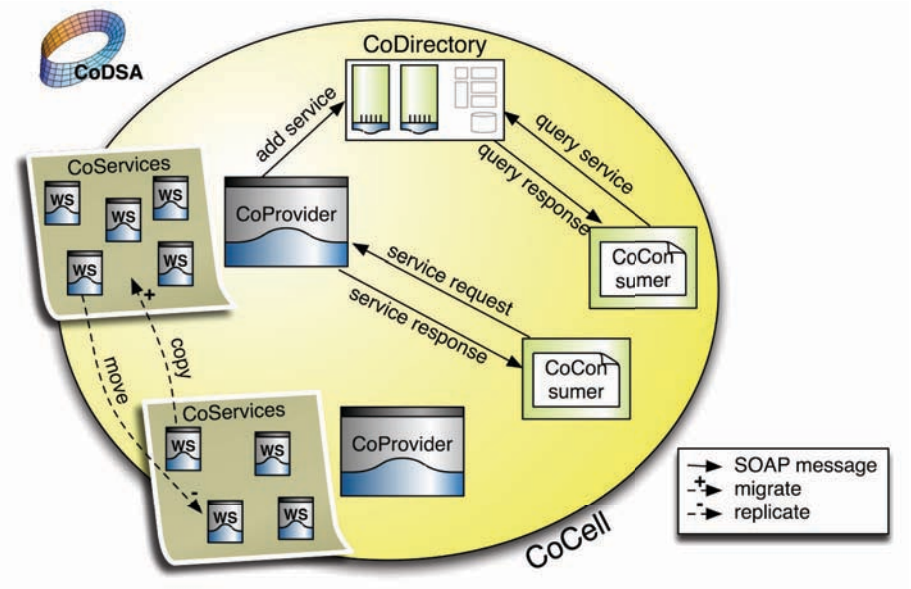

Figure 4.5: Continuum Distributed Service Architecture

Migration allows a CoService to change its location from a CoProvider to another. This occurs in the scope of a CoCell, and is used to improve the system performance, reducing communication costs and delays. Currently, web service architecture does not support this feature. However, there are some proposals to address migration in this scope (HAO et al., 2006). The main concern is how to decide when a CoService should migrate and to which location. This decision must consider hysteresis and the costs involved. In our proposal, the Distributed Execution subsystem (section 4.6.1) evaluates these aspects and makes the decision.

Another option in CoDSA is replication. CoServices can be replicated among CoProviders in the scope of a CoCell. This can improve system reliability of CoServices. Whenever a node changes its location or disappears in a CoDimension, which is common in mobile environments, the CoService it provides becomes unavailable. If there is a replica, it can be discovered from the CoDirectory. Besides this discovery and registration feature, we also need a mechanism for the replication and synchronization of web services. Some methods have been proposed to address these mechanisms (JUSZCZYK et al., 2006; MOSER et al., 2006). The two main problems with replication are the synchronization of CoService copies and the decision whether a replica of a CoService should be made. Moreover, communication costs of the dynamic copy must be considered. Similarly to what happens during migration, it is the Distributed Execution subsystem that is in charge of the main operation, as well as of dealing with the other problems pointed out here.

CoDSA allows the dynamic selection of CoServices, which are chosen according to the functionalities needed for each node of the system. We can obtain an adaptive behavior in Continuum by replacing or reconfiguring the CoServices that a CoConsumer employs. During software design, the Execution Profiler provides support for the selection of services needed by each node in the system. During execution, it is the Adaptation Management that takes this decision.

The interaction among nodes in the CoDSA, as illustrated in Figure 4.5, uses SOAP messages.[22] SOAP, an acronym for Simple Object Access Protocol, is a lightweight communication protocol based on XML that allows the accessing of web services. The protocol is platform and language independent. As the name implies, it is very simple and also extensible. SOAP enables asynchronous client-sever communications and can make use of a wide range of protocols, including HTTP, SMTP, TCP, and UDP.

Table 4.3: Messages exchanged in CoDSA

Message	From	To	Purpose
Query Service	CoConsumer	CoDirectory	Looking up for a CoService
Query Response	CoDirectory	CoConsumer	Informing the location of a CoProvider
Add Service	CoProvider	CoDirectory	Publishing a CoService in the CoDirectory
Remove Service	CoProvider	CoDirectory	Deleting a CoService from the CoDirectory
Update Service	CoProvider	CoDirectory	Updating information about a CoService
Service Request	CoConsumer	CoProvider	Demanding a specific CoService
Service Reply	CoProvider	CoConsumer	Responding to a requested CoService

[22] More details about SOAP in <http://www.w3.org/2000/xp/Group/>.

The SOAP messages used in the CoDSA are generally described in Table 4.3. It shows each message along with the origin node, the destination node, and its purpose.

4.5.2 Proposed Services

The Continuum functionalities are presented in terms of services. The next section presents an overview of the pluggable services proposed. We must emphasize that the services presented here, as well as their functionalities, do not aim at exhausting all the possibilities, neither do they cover all the topics discussed in the comprehensive model. Rather, our goal is to offer an idea of what features might be available in the software infrastructure, and how they could be provided. In some services, we propose the integration of other works, developed in the perspective of the ISAM project.

There are two types of services in Continuum:

I. derived from EXEHDA, incorporating new or redesigned features;

II. new services, created specifically for Continuum.

The detailed description of the services of the first type can be found in Yamin (2004). Special attention is given here to new or redesigned features. The second type of services is fully described.

The functionalities of the pluggable services are presented in terms of its interface, as an overall service description. We present the operations belonging to a web service using a UML class diagram (FOWLER, 2005) for visually modeling key portions of the Web Services Description Language (WSDL).[23] WSDL is commonly used for service representation. It is a W3C recommendation, defined as an XML schema for representing the description of a service. Our work is based on WSDL 2.0, which is the version currently in use.

WSDL 2.0 (hereafter simply referred to as WSDL) describes the functionalities of a web service in two parts: an abstract part describing messages exchanged through a type system and a concrete part that defines details such as transport, location, and implementation (CHINNICI et al., 2007). The abstract description of a web service comprises three characteristics: messages, operation, and interface (formerly known as PortTypes in 1.0). The concrete part comprises binding, services, and endpoints. Figure 4.6 summarizes the conceptual WSDL model. For a full description of the WSDL model, refer to Chinnici et al. (2007).

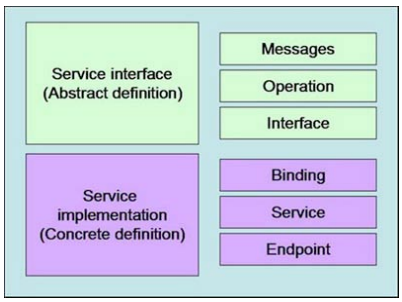

Figure 4.6: WSDL conceptual model (DHESIASEELAN, 2007)

[23] More about WSDL can be found in < http://www.w3.org/2002/ws/desc/>.

When describing Continuum services, we are particularly interested in modeling the property interface of the abstract part of the WSDL conceptual model. Therefore, only a subset of a WSDL is showed in the UML diagram of the proposed services. Figure 4.7 shows the generic class diagram for this work. In the following subsections we describe the Continuum services, classified by their type (I or II), and organized in subsystems.

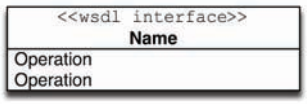

Figure 4.7: Generic UML representation of a WSDL interface

4.6 Subsystems

The concept of subsystem is used to represent an aggregation of services with a common aim and purpose. In practice, no additional task is provided by the subsystem itself, but rather it comprises a set of services that are used to achieve some specific goal of the infrastructure. Some services in the system are used by more than one subsystem, while others are specific.

Continuum subsystems are presented in Figure 4.8. We describe all Continuum subsystems with the exception of Context Awareness, which will be presented in the next chapter (in section 5.3). Each subsystem can be personalized according to the services that should be available. With the aid of the Execution Profiler, during bootstrap, services needed in each node of the system are loaded.

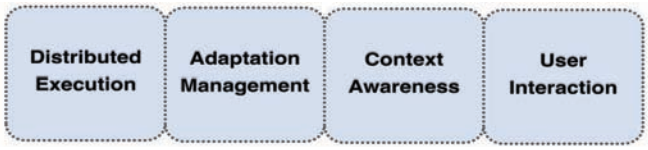

Figure 4.8: Continuum subsystems

4.6.1 Distributed Execution

The Distributed Execution subsystem is in charge of Continuum distributed processing and communication. Figure 4.9 shows the services that are part of the subsystem. Three services (Executor, CIB, and Co-Space) are redesigned from previously existent EXEHDA components. The others are new additions to the software infrastructure. In the next subsections each one of these services is described.

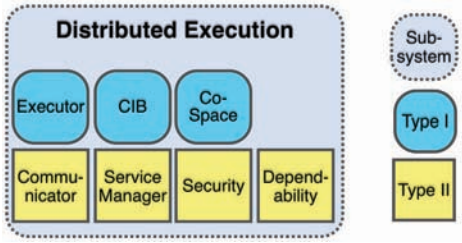

Figure 4.9: Distributed Execution subsystem

4.6.1.1 Executor

Executor is a service for management and distribution of applications in Continuum, derived from an ISAM service with the same name. It is developed regardless of communication model. Through this service, it is possible to handle application-related operations. This service should be present in every device that runs Continuum applications (this does not include CoGadgets, which are executed in an ad hoc approach via web browser). Executor acts as a thin runtime interface layer for any application that uses Continuum software infrastructure. The operations of the service are listed in Figure 4.10. The new operations appear in bold.

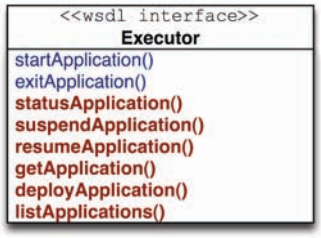

Figure 4.10: Executor service interface

We call each piece of software registered in Continuum a *CoApp* (detailed in the work KELLERMANN, 2008). A CoApp is a predefined format for the encapsulation of Continuum applications. It includes a configuration section, which has all the metadata related to the application, such as different versions, dependences, and XML codification (schema XSD). Another layer of a CoApp contains the resources. This layer is optional and may include databases, images, and internationalization data, among other resources. Finally, there is also the implementation layer. Not only does this layer include the application code, but also all its dependences, such as the necessary runtimes and libraries. The interface of a CoApp is a web service and all its definition follows web service standards, such as communication, localization, and management protocols.

In Executor, we have some methods to manage CoApps. The operations `startApplication` and `exitApplication` are responsible for the initialization and finalization, respectively, of Continuum Applications. There is a specific operation, named `statusApplication`, to return the state of a CoApp along with its reference and other related data. Other methods deal with application states: `suspendApplication` and `resumeApplication`. These functions are in charge of suspending and resuming the execution of a CoApp in the Continuum infrastructure.

It is also possible to obtain a CoApp object from a given identifier. This is accomplished by the `getApplication` method. To install an application, we use the operation `deployApplication` and to list all Applications locally available in the current node we use `listApplications`.

4.6.1.2 CIB (Cell Information Base)

The Cell Information Base (CIB) is in charge of keeping the distributed infrastructure of Continuum. By way of this service, we can add and remove entities from a CoCell. This service is based on a previously existing functionality, with the

same name, from EXEHDA. The service's main operations are listed in Figure 4.11, with the new ones in bold.

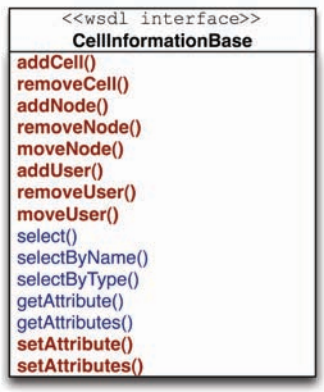

Figure 4.11: CIB service interface

The operations `addCell` and `removeCell` are used to create and delete cells in Continuum. In the former, we must pass as an argument which cell it will compose. If omitted, it will be the top-level cell, directly inside the CoDimension, embracing all other cells, assuming that they exist. The latter is used to remove a cell. When a cell is removed, all inner entities and cells are also removed. Operations `addNode` and `removeNode` work closely. The main difference between the two is that the target here is a node. When using the first method, we have to pass as an argument the type of node we are creating (node, mobile, or gadget), the cell in which we want to add it, and the relationship that it will establish with the cell (association, composition, or aggregation). With the second operation, we can eliminate a node. Only nodes associated and aggregated to a cell can be deleted. To remove a composed node, we have to eliminate the cell.

The `moveNode` operation provides mobility to nodes. This method can be used with associated and aggregated nodes. As an argument, we inform the destination cell and the type of relationship that the node will establish with it. The user operations are analogous to the nodes operations: `addUser`, `removeUser`, and `moveUser`. The only difference is that there is no need to specify the relationship, since users are always associated to a CoCell. The move operations of the CIB service provide physical mobility to Continuum.

The select operations (`select`, `selectByName`, and `selectByType`) are inherited from EXEHDA and used to find resources in the CIB. The same occurs with `getAttribute` and `getAttributes`, which are used to obtain the features of nodes. The first fetches one specific resource property, while the latter obtains a list of features. We add to Continuum two more operations to facilitate the modification node attributes, named `setAttribute` and `setAttributes`. These can be used to change modifiable features of a node, such as its name, IP address, etc.

4.6.1.3 Communicator

The Communicator service provides an event system for Continuum, based on the publish-subscriber model. This service replaces the former EXEHDA service named

dispatcher, which was based on the request-reply interactions of the client-server model. We made this change according to the spontaneous interoperation requirements of the comprehensive architecture model (section 3.3.5). Furthermore, this service is highly used by the Context Awareness subsystem. The operations we propose for the service are presented in Figure 4.12.

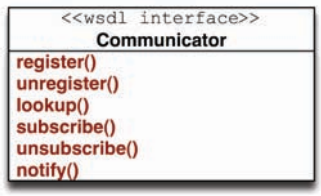

Figure 4.12: Communicator service interface

The idea of an event system is that a component can react to change occurring in another component, in an asynchronous way. To achieve this, we propose six operations. The operation `register` (and `unregister`) is used by a component that wants to publish (unpublish) some particular event. An event in the system has attributes that can specify details of its occurrence, such as time, name, etc. If a component wants to be informed of occurrences of some event, it can `subscribe` to it. Eventually, when this component does not want to receive any further notification, it can `unsubscribe` the event.

If a component wants to find some specific event, it can query the Communicator service, with a `lookup` operation, for available events. Finally, the `notify` operation is used by a registered component to send an event to a component that has previously subscribed to it. The event is delivered to all the subscribers asynchronously.

4.6.1.4 CoSpace

The CoSpace service, based on the previous EXEHDA service named CCManager, provides a tuple space for Continuum. It is also in agreement with the interoperation characteristics of the general model (section 3.3.5). The idea of providing a tuple space, besides an event system, is to add synchronous operations, which are easier to program, to the infrastructure. Another reason is the persistence offered by a tuple. The operations proposed are shown in Figure 4.13. The new operations appear in bold.

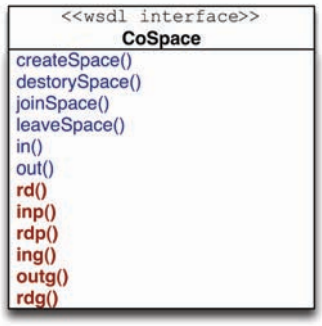

Figure 4.13: CoSpace service interface

In the service, there are methods to create and destroy a tuple space (`createSpace` and `destroySpace`) and to enter or depart from a space (`joinSpace` and `leaveSpace`). The operations proposed to manage tuples in the space were based on LIME, a middleware for mobile environments (MURPHY et al., 2006):

- `in`: obtains a tuple that matches a given template, and removes it from the tuple space. If no tuple matches, it waits until one shows up (synchronous);
- `out`: inserts a tuple in the tuple space;
- `rd`: retrieves a copy of a tuple that matches a given template. The tuple remains in the space. As with method `in`, this operation is also synchronous;
- `inp`: same as `in`, but asynchronous. If no tuple matches the given template, it returns `NULL`;
- `rdp`: an asynchronous retrieve of a tuple. Same as `rd`, but if no tuple matches the template, it returns `NULL`;
- `ing`: retrieves all tuples that match a given template, removing them from the space;
- `outg`: writes a set of tuples in the space;
- `rdg`: retrieves a copy of all tuples that match a given template. The tuples remain on the space.

4.6.1.5 Service Manager

The Service Manager, as the name implies, deals with all pluggable services in Continuum. Not only does it handle predefined Continuum services, it also allows any CoApp to be managed as a service in the software infrastructure. This feature enables the "pluggable" feature of Continuum services. It also collaborates to the extensibility of the platform, facilitating the use of legacy code.

Service Manager replaces the service WORB of EXEHDA, although they have quite different functionalities. WORB provides communication that is similar to RMI (Remote Method Invocation), without the need of maintaining a synchronous connection. This is not possible anymore in Continuum. Instead, through Service Manager, we can register a software component as a web service, and access its operations using SOAP messages. Another option is using the events operations available in the Communicator service. The Service Manager interface is shown in Figure 4.14.

Figure 4.14: Service Manager interface

Service Manager provides the functionalities of a CoDirectory (see section 4.5.1). `registerService` is responsible for registering a service in the directory. In a complementary way, `unregisterService` deletes a service from the directory. If modifications are made in a service, it can be updated in the directory using `updateService`. We use `lookupService` to query services available in the directory. The method, by default, only looks for services in the current CoCell. A parameter can define that the query should be passed on to the outer cell and so on, searching in the entire hierarchy. In this case, the query returns either when it finds the service or when, at the end of the hierarchy, it has not found anything.

Services can be moved (`moveService`) or copied (`copyService`) among nodes. In the former case, when a service is moved to another cell, it is registered in the CoDirectory of the target CoCell and removed from the current directory. In the latter operation, an event is associated with the original service, and all modifications in it generate notifications to the copy. No modifications are allowed in the copied version, only in the original service. These notifications are accomplished using the Communicator service. Another possible use of the Communicator service, in the scope of ServiceManager, is the possibility to create events for a notification when a specific service gets registered in the directory.

4.6.1.6 Security

The Security service offers some operations to help users and applications to deal with security in the ubicomp software infrastructure. When addressing this issue, we tried to follow the characteristics determined in the comprehensive model (section 3.3.3). Besides security mechanisms being manageable and understandable by users, due to the spontaneous interoperability of ubicomp, they are mainly an end-user concern (DOURISH et al., 2004). With these characteristics in mind, we proposed the Security service interface (Figure 4.15). The new operations appear in bold.

Figure 4.15: Security service interface

The public key management (`addPublicKey`, `removePublicKey`, and `getPublicKey`) is used to provide end-to-end private communication. Components, using security service, can easily share public keys. Internally, components can use their private keys to encrypt or decrypt exchanged data, or to form a secure association among devices. The methods for encryption and decryption are available through the

framework, i.e. not via web service. We chose this strategy, because the private key must be kept local to the node and not exchanged via the network, in order to further increase the security of the mechanism. In spite of this approach, not all security issues have been solved: there remains the possibility of breaking into the node and obtaining the private key.

The idea of also providing a secure channel, via `createSecureChannel` and `removeSecureChannel`, is to offer a reliable channel. It could be used, for instance, to perform a direct validation of exchanged public keys between two communicating components. This secure channel is established using a symmetric key, instead of the asymmetric public-key method used in the previous operations. Because the cryptography method is symmetric, it can be employed by portable devices with resource-limited capabilities (COULOURIS et al., 2005).

To make the establishment of a security channel more reliable, one possibility is to pass an argument in the `createSecureChannel` method, signalizing that there is no need to create the key, since the two communicating devices already share it. Thus, no key is transmitted over the network, provided that both devices have the key. It can be hard-coded in the device, or exchanged by a physically constrained channel, such as infrared, laser, audio, or a barcode/camera method (for more information on the latter, refer to KATO and TAN, 2007).

The communication using a secure channel takes place via `sendS` and `receiveS`, classical methods for exchanging messages. Communicator and other Continuum services employ these methods so that reliable communication is available throughout the infrastructure.

The operation `authenticate` provides localized authentication in Continuum. Only persons physically present in one CoCell can use the services available there. The proper functioning of the operation depends on obtaining the pinpoint of the authenticating person. The operation uses the context awareness service in order to access this information.

The last operations are inherited from other preexisting EXEHDA services. The `grant`, `drop`, and `renew` methods are used to deal with access control. Using these operations, it is possible to make resources available to applications and users that are located externally from the CoCell these resources belong to. On the other hand, the `log` and `trace` functions are used for logging purposes, normally applied to the debugging phase of application development.

4.6.1.7 Dependability

The Dependability service offers some operations that could be used to avoid failures when they are more frequent or more severe than acceptable. Basically, two sets of operations are provided: the first simplifies the recovery of failures, by using a checkpoint mechanism; the second helps the establishment of redundancy for fault tolerance purposes. The operations proposed are summarized in Figure 4.16. Other mechanisms available in Continuum could also be used to improve dependability: the secure channel operations of the previously presented Security service could provide an alternative communication channel; the Cyber Foraging subsystem (section 4.6.2.2)

could be used to mask some uneven conditions; and the replication mechanism for services in the Service Manager could be employed to improve availability.

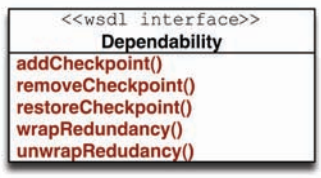

Figure 4.16: Dependability service interface

Checkpointing is used to capture the execution state of a specific CoCell. The state is saved in a tuple using the CoSpace service. This feature was inspired by a similar functionality from One.World (GRIMM et al., 2004). The operation `addCheckpoint` creates a snapshot of the environment state of a CoCell and saves it. The complementary operation `removeCheckpoint` deletes the saved state. Finally, the `restoreCheckpoint` operation reverts the current execution environment to a previously saved one. As stated in Grimm et al. (2004), this method "simplifies the task of gracefully resuming an application after it has been dormant or after a failure."

The `wrapRedundancy` operation is used to include some redundancy into a specific service. Here are some options available:

- *Hardware redundancy*: adding additional nodes that will execute the same service;
- *Design diversity* (AVIŽIENIS et al., 2004): adding software components that execute the same function, possibly made with a separate design and implementation;
- *Time redundancy* (AVIŽIENIS, 1998): specifying a number of times to execute a service, repeating the computation;
- Any combination of the previous possibilities.

The operation returns a new service that incorporates the chosen redundancies. When using this service, all the redundant copies are executed a specific number of times. If the execution involves returning a result, the final result of all replicas is compared and, if it is the same, returned; otherwise, it could signalize a failure or the value returned by the majority, according to a parameter passed to the `wrapRedundancy` method. The `unwrapRedundancy` operation does the opposite, i.e. it eliminates some (or all) redundancy of a specific service.

4.6.2 Adaptation Management

The Adaptation Management subsystem is responsible for context management in the Continuum software infrastructure. It is composed of three services (Figure 4.17). Adaptation control is an evolution of a previous EXEHDA service and has been developed as a doctoral thesis at UFRGS (SILVA, 2008). The two other services are new propositions made in the scope of Continuum and based on the previously presented comprehensive architecture.

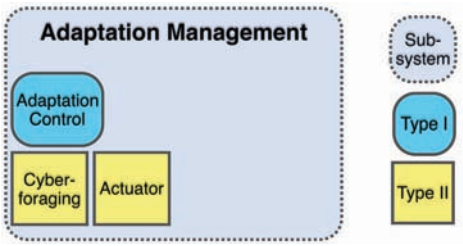

Figure 4.17: Adaptation Management subsystem

4.6.2.1 Adaptation Control

The Adaptation Control is the service in charge of adaptation in the Continuum infrastructure. The service uses the ACTUS architecture, which provides a general solution for adaptation control in ubicomp (SILVA, 2008). Based on definitions by ACTUS, the presented service acts as an application shaper (SILVA, 2008), which represents an interface of a specific binding solution to the proposal. The planned service interface is presented in Figure 4.18. Although this module uses ACTUS, the set of operations proposed are new.

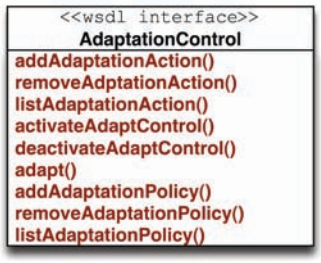

Figure 4.18: Adaptation Control service interface.

The first three operations manage adaptation actions, which are, as defined in ACTUS, behaviors that are associated with a specific state of some related context elements (SILVA, 2008). The `addAdaptationAction` operation inserts actions related to a specific component. In a complementary fashion, `removeAdaptationAction` deletes some action associated with a component. The `listAdaptionAction` method presents all actions related to a specific component.

The `activateAdaptControl` and `deactivateAdaptControl` functions rule the start / stop of ACTUS. Once active, ACTUS issues adaptation actions according to changes in the context. ACTUS chooses the order and timing of each adaptation. Also, the architecture considers various metrics in this decision process, such as agility of adaptation, stability of the system, and quality of service (SILVA, 2008). We can also force an execution of ACTUS's Adaptation Control module by calling the `adapt` function, which can possibly generate some adaptation action.

The last set of instructions is related to policies from the perspective of ACTUS. In the proposal, a policy defines criteria for assessing adaptations suggested by ACTUS (SILVA, 2008). The `addAdaptationPolicy` and `removeAdptationPolicy` operations insert and delete, respectively, policies related to a specific component. On

the other hand, `listAdaptationPolicy` returns all policies related to a given component.

4.6.2.2 Cyber Foraging

The Cyber Foraging service is based on the idea of Project Aura to amplify the capability of devices with low processing power and limited storing capacity (GARLAN et al., 2002). The service allows the apportioning of near machines as surrogates, which can act as computing or data-staging servers (for more detail, refer to section 3.3.8.2). The main operations proposed are listed in Figure 4.19.

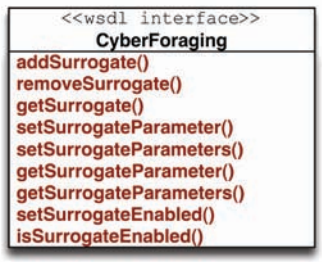

Figure 4.19: Cyber Foraging service interface

The `addSurrogate` operation is used to register a node as a surrogate in a specific CoCell. It can be subsequently deleted by the complementary `removeSurrogate` method. When a node, typically a mobile one, needs a computing or data-staging server, it may obtain it using the operation `getSurrogate`. In this, the caller must specify if it needs a computing or data-staging server (or both). If none is available, the return is `NULL`.

Once a surrogate server is returned, it can be configured by `setSurrogateParameter` (which alters only one option) or `setSurrogateParameters` (which alters a group of options). Complementary get / set operations (`getSurrogateParameters` and `SetSurrogateParameter`) are also available.

To activate / deactivate a surrogate, we use the `setSurrogateEnabled` method. The last function `isSurrogateEnabled` returns the current status of a particular surrogate server.

4.6.2.3 Actuator

This service is intended for controlling actuators. It is inspired by the monitor service (section 5.3.1), which is aimed at the sensor control. The idea of the API is similar, but specifically targeted at actuators. The interface is presented in Figure 4.20.

The function `getName` is used to acquire the specific name of an actuator. Its reference can be obtained via `getActuator`. There are also specific methods to handle their parameters (`getActuatorParameter`, `getActuatorParameters`, `setActuatorParameter, setActuatorParameters`), to activate/ deactivate an actuator (`setActuatorEnabled`), and to verify if an actuator is enabled (`isActuatorEnabled`). To obtain the current value of an actuator, we use the

`probeActuator` function. The last method, `installActuator`, is used to add a new actuator to the system.

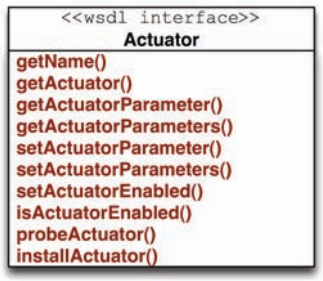

Figure 4.20: Actuator service interface

4.6.3 User Interaction

The User Interaction subsystem deals with the high level layers of the comprehensive architectural model, namely invisibility and transparent User Interaction. This subsystem is composed of four services, one of which has been inherited from EXEHDA (Persistence service). Figure 4.21 presents the main services of the User Interaction subsystem. Although the persistence service is derived from EXEHDA, it has been improved by graduate research done at UFRGS (for more details, refer to FRAINER, 2008).

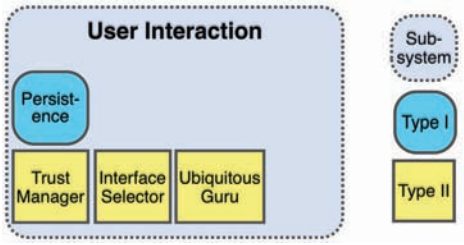

Figure 4.21: User Interaction subsystem

4.6.3.1 Persistence

This service provides ubiquitous access to files in the Continuum infrastructure and is detailed in Frainer (2008). The Persistence service integrates distributed file system concepts with application-aware adaptation. The interface is defined in Figure 4.22, with the new operations appearing in bold.

The primitives `open`, `create`, `close`, `read`, and `write` are, at first view, the traditional file system operations available in every OS. However, they deal with pervasive files (FRAINER, 2007). Firstly, a pervasive file is a file that could have many copies spread among various devices, in a way that increases availability. Secondly, it could have different versions, to facilitate adaptation. Finally, it can be composed of a range of smaller files, making it easier to deal with larger files. The Persistence service automatically manages these pervasive files. The metadata, managed with `getMetaData` and `setMetaData`, consists of high-level pairs – {value, attributed} –, which can be used to guide adaptation, increase availability, and specify conversion

possibilities. The normal low-level OS attributes are obtained and altered using `setAttribute` and `getAttribute`, respectively.

Figure 4.22: Persistence service interface

The `prefetch` is used to increase file access performance, by creating new copies of files that have a high probability of being used. The function can be called directly, but the `open` function also uses it indirectly. The metadata of the pervasive file are used to help with this operation.

The `addVersionChoiceHeuristic` and `removeVersionChoiceHeuristic` functions are used to manage the decision heuristics that will be used to open a file. This heuristics is used to guide the decision as to which version should be employed. The user can register (and unregister) different versions, using `addAlternateVersion` (and `removeAlternateVersion`). Another possibility is the dynamic creation of versions. These are obtained by using conversion plug-ins (`addConversionPlugin` and `removeConversionPlugin`), which specify how a file can be transformed from one type into another. The last pair of operations (`addTransferPriority` and `removeTransferPriority`) is used to define the priority of files in the Continuum environment.

4.6.3.2 Trust Manager

The Trust Manager service deals with trust and privacy in the Continuum environment. We choose to keep this service in the User Interaction subsystem rather than in the Distributed Execution one, because, normally, the goal of a trust manager "is to provide a computational version of the human notion of trust" (SEIGNEUR, 2005), i.e., the concept is more related to users and their interactions than to processing and communication. The Trust Manager interface is shown in Figure 4.23.

The first operation (`anonymizeData`) is related to privacy. With this function, a pseudonym is given to a set of data that are associated with a user. This can be employed to manage the identities of users in the infrastructure, in such a way that, if a data is intercepted, the real person cannot be identified. To insure this behavior, after the use of `anonymizeData`, only the pseudonym returned should be used. Cas (2005) affirms that this technique is not sufficient to guarantee privacy to the user, although it

is considered appropriate to at least avoid the necessity of user consent in every act of giving personal data. Nevertheless, we believe that making data anonymous helps in the direction of achieving privacy.

Figure 4.23: Trust Manager service interface

The next set of operations is more related with the provision of trust. According to Seigneur (2005) to take this decision, two modules are needed: one that dynamically evaluates the trustworthiness of an interaction based on evidences, and another that assesses the involved risk, helping in choosing the action to be taken. The function `evaluateTrustworthiness` is related to the first module, while `evaluateRisk` is associated with the second one. A decision-making algorithm, which takes into account evidences stored in a database, fulfills these evaluations. Thus, components can use this result to decide whether an interaction should or should not be made.

The remaining functions related to these two modules (`setTrustworthiness`, `getTrustworthiness`, `setRisk`, and `getRisk`) are connected to the management of an evidence database. Using these functions makes it possible to add or obtain recommendations and observations connected to trust and risk information, respectively.

For further detail on this service, we believe that additional studies have to be conducted. For supplementary information on the subject, refer to Cas (2005), Cahill et al. (2003), Robinson et al. (2005), Seigneur (2005), and Surie et al. (2007).

4.6.3.3 Interface Selector

The Interface Selector service helps in maintaining and creating a user interface and the means of interaction for devices in the software infrastructure. The service stores interfaces in an independent design approach. Those have to be converted to a user interface (UI) of a specific device. The idea is that the framework layer addresses this conversion. The service interface is presented in Figure 4.24.

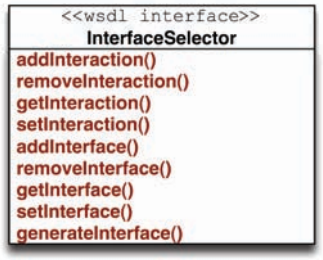

Figure 4.24: Interface Selector service

The first three operations are related to the specification of interaction between the user and the components of the infrastructure. Interactions are actions that services present to users, and are described independently of device, service, and user interface type (as in NYLANDER et al., 2005). In the article, Nylander and her colleagues propose an XML compliant language to encode interaction. We suggest the adoption of such language in Continuum. In the language, a group of eight interaction acts are possible (NYLANDER et al., 2005): *input* to the system, *output* to the user, *select* from a set of choices, *modify* information stored, *create* new objects, *destroy* existing objects, *start* session with a component, and *stop* an interaction with a component.

Using `addInteraction` operation makes it possible to store an interaction specification for a component. The complementary `removeInteraction` deletes a specification from the infrastructure. The `getInteraction` and `setInteraction` functions are used to, respectively, obtain and alter the behavior of the interaction in the specification.

The last set of operations deals with the control of presentation information. It is possible to specify (`addInterface`) or delete (`removeInterface`) the UI for a specific component (also following the work of NYLANDER et al., 2005). As in the case of interaction, the interface is platform independent and should be converted to a specific environment (device, language, API, etc.). As in the previous set of functions, `getInterface` and `setInterface` are related to the obtaining and changing of the stored presentation. The final operation, `generateInterface`, creates a generic presentation, based on the interaction specified for any given component. If no presentation information is specified for a component, this function can be used to generate a user interface with the default settings. We believe that providing a specific presentation offers more control to the user, and probably a better experience.

To tackle the specific aspects of tangible interaction, more work has to be done. There are some proposals with initial results that could be considered (HORNECKER, 2005; ROSS and KEYSON, 2007; STRINGER et al., 2005).

4.6.3.4 Ubiquitous Guru

The service called Ubiquitous Guru is in charge of dealing with invisible issues in the Continuum infrastructure. This service addresses user attention, user intent, and seamless integration. The interface proposed for Ubiquitous Guru is presented in Figure 4.25.

Figure 4.25: Ubiquitous Guru service interface

The first three functions manage the user's preferences. They are used in this service for obtaining user attention and intent, besides being used in adaptation and context awareness. The `addUserPreference` and `removeUserPreference` are respectively used to include and delete user's preferences in the infrastructure. The `listUserPreference` is used to present all preferences associated with a specific user. Preferences have the format of a named pair. The first is the context for the preference to apply, and the second, a scoring expression ranging from 0 to 1, whose value is directly proportional to its level of desirability, as defined in Henricksen and Indulska (2006). Not only do these authors provide a score, they also propose the use of four special values instead: *veto*, indicating that this choice should not be selected; *obligation*, denoting that this choice should always be selected; *indifference*, representing an absence of preference; and *undefined*, signalizing an error condition (HENRICKSEN and INDULSKA, 2006).

The next set of functions deal with tasks. This is in conformity with the task-aware system proposed for seamless integration in the invisibility challenge description of the comprehensive architecture (section 3.3.10). Operation `addTask` tells the infrastructure what the user wants to do, `removeTask` deletes a specific item from the To Do list, `listTask` presents the complete list, `orderTask` permits changing the order of pending elements, and `nextTask` returns what is next on the To Do list, deleting it. The use of tasks in Continuum is based on the work of Sousa et al. (2006) on the project Aura. In Aura, tasks define explicit representation of user's activities in a high-level format, independent from the mechanisms that accomplish these activities. Based on the user's preferences, the system can choose the best available mechanism to carry out the desired task. This is obtained in Continuum by the use of `suggestAction`. Tasks are defined by the specification of the service needed and of the related preferences, as in Sousa et al. (2006). In the original proposal, a vocabulary is defined for the possible services. We propose the aggregation of this vocabulary in the ontology designed for modeling the context. Further works should address this topic.

The last group of primitives handles user feedback. They are used to better tune the suggesting of actions. Sometimes, simply exposing the preferences may not lead to a good matching between a task and a mechanism. By using feedback, users can give information that could modify future choices in the system. The feedback takes the same format as user preferences. The only difference is that it is always associated with a suggested action. The `addUserFeedback` and `removeUserFeedback` functions insert / delete feedbacks in Continuum, while `listUserFeedback` shows all feedbacks given by a specific user.

4.7 Framework

The Continuum framework is aimed at addressing design time abstractions needed for the implementation of ubiquitous software. Our proposal suggests that this layer deal with essential characteristics presented by the comprehensive model, incorporating some features from ISAMadapt (AUGUSTIN, 2004). This framework must be further detailed in a future work. Here, we present some general considerations for the development of this layer.

Continuum framework is intended to help the development of ubiquitous applications using middleware services. It is also its aim to simplify the use of the underling middleware services. There are some elements that integrate this framework:

- **Application Programming Interface (API)**: it is composed of a set of services that uses the API provided by the middleware layer. The framework API specializes the interface and simplifies its use, providing some private services not available from the middleware. In the framework, some information is maintained in order to preserve an environment among different calls for middleware services. Another characteristic of the framework API is to provide a work model (as defined in BERNSTEIN, 1996). In this model, the framework does not have to offer all the services available in the middleware, but rather only those that are significant to a specific application or environment;
- **User Interface (UI)**: it provides a look and feel adapted to the platform being used for ubiquitous application design. This UI can help the development of applications, with icons, layers, and appearance that are suited for this design task;
- **Tools**: they represent a set of generic applications to simplify the use of the framework. It may be composed of editors, help, debuggers, etc. The tools in Continuum framework are aimed at programmers. One of the available tools is the Execution Profiler, which assists in the parameterization and deployment of services in the infrastructure.

Based on these elements, and also on the design time characteristics of the comprehensive architecture, we propose in Table 4.4, some features for the Continuum framework. The table summarizes each feature, along with its corresponding characteristics, and the framework element that should be used to attain this objective.

In the design of Continuum framework, we can employ the language abstractions proposed for adaptation in the perspective of ISAMadapt. We list here these abstractions, along with the framework element that can address it:[24]

- Context element definition: ISAMadapt proposes a UI for the definition of templates to guide the execution of context services. In addition, the proposal uses environment variables to help with context identification. These features may be included in the abstract interaction element characteristics of the Continuum framework;
- Expressing adaptive behavior: in ISAMadapt, the command `onContext` is defined for this task. It can be incorporated in the API of the adaptable application characteristics;
- Adaptive being and method: it comprises a component or a function, in ISAMadapt, whose code is defined by context. This feature can be used both in the definition of the abstract services and in the adaptable application characteristics;

[24] For a detailed description of ISAMadapt language abstractions, refer to AUGUSTIN (2004).

- Adaptation commands: a group of adaptation methods is described in ISAMadapt that might be added to various frameworks API. Among the proposed commands, we highlight *move* and *clone*, which may be included in mobile code and data design;
- Adaptation policies: they define a set of guidelines that direct the adaptation decision in ISAMadapt. They could be employed in the adaptable applications characteristics of the framework.

Table 4.4: Proposed features for Continuum framework

Design time characteristics	Framework element	Main functionality
Device-independent	API	Exposing the same set of interfaces for different hardware and drivers
Open standards	API	Employing published and standardized interfaces for service access, especially for communication
Scalable solutions without bottlenecks	Tools	Assisting with distribution, replication, and caching to improve the performance of frequently used resources
Verification	Tools	Providing diagnostics and tests, facilitating fault detection and removal during design time
Security design	API	Presenting an interface to help the security project at the design stage, enforcing ubicomp aspects
Privacy standards	Tools	Reinforcing privacy by presenting procedures that should be observed in the collection of data, including specific jurisdictions
Trust reasoning	Tools	Evaluating the trust based on available information at design and suggesting recommendations and observations to be considered during run time
Spontaneous component design	Tools / UI	Offering a description language and user interface for the development of spontaneous components
Mobile code and data design	API	Supporting the project of mobile code and data
Abstract services	API	Presenting a set of generic services for the development of software, helping with the implementation of context-aware applications
High-level interfaces	API	Generalizing the access to sensors and actuators to ease their use in ubicomp software
Abstract interaction elements	API / UI	Helping specifying elements that will be used in the adaptation during execution
Abstract user interfaces	UI	Designing user interfaces suited to different devices
Interact devices	API	Improving the way the infrastructure addresses the interaction style of different devices
Adaptable Applications	API	Easing the development of ubicomp applications, providing ways of defining adaptable code

In the next subsection, we concentrate on the description of the Execution Profiler framework tool. The further development of the Continuum framework is listed at the end of this work as an intended future work.

4.7.1 Execution Profiler

Execution Profiler is a framework tool intended to configure a CoDimension in Continuum. In this tool, the user can define the environment in which their ubicomp

applications will execute, selecting a group of settings and parameterizing the deployment process.

The deployment setting is divided in three categories: hardware, software, and user settings.

The *hardware* category defines device organization and role in the Continuum infrastructure. In this, CoCells are defined, with their CoNode, CoBase, CoMobi, and CoGadget. The types of relationship (aggregation, composition, and association) are also specified here. This specification is not static, but rather it is the initial organization available after system bootstrap.

The *software* part configures which services will be initially available in the system and at which location. We also inform if the service should be preloaded or loaded on-demand, according to infrastructure needs. Later, services can be moved, copied, or reconfigured dynamically. In this category, the components that will be used are also determined with its parameters.

The final configuration is related to *user settings.* In this, the initial registered CoPersons are defined with their identifiers. Also, we set privacy, security policies, permissions, and roles of users in the Continuum infrastructure. Dynamically, more users can be added to the system.

The set of all the configurations with the hardware, software, and user setting categories is called an *execution profile*. In the definition of this profile, we should consider the checklist of questions described in Hansen et al. (2006). Briefly, this checklist defines a group of issues, for each category, that should be observed during real-world deployment (HANSEN et al., 2006):

- Hardware: cost (equipment, realization, special devices), security (location, risks, protection), power (requirements, recharging, consumption), network (connections, infrastructure, latency), space (size, location, convenience), and safety issues (contingency plans, dependability, interference);
- Software: transmission (transference, scalability, update processes), debugging (fault detection, error detection, failure detection), security (risks, confidential information), integration (third-party components, communication), performance (speed, scalability, requirements), fault tolerance (recovery, state storing, notification), and heterogeneity (communication, difference);
- User setting: usability (quantity of users, interface, ease of use), learning (support, helping, tutorials), politics (control, access), privacy (dealing of personal information), and adaptation (resistance, changes).

Most of the issues listed here are supported by Continuum services. It is important that when defining the execution profile, these issues are considered. The idea is that the Execution Profiler tool could reinforce this checklist, making deployment an easier process. For this purpose, we suggest the presentation of the checklist to the users and their manual check of each issue, as a way to help with deployment planning.

Figure 4.26 summarizes the Execution Profiler tool, presenting the three categories deployment settings, the checklist, and the final result, which is the execution profile. This profile is then used to accomplish the initial deployment in the infrastructure.

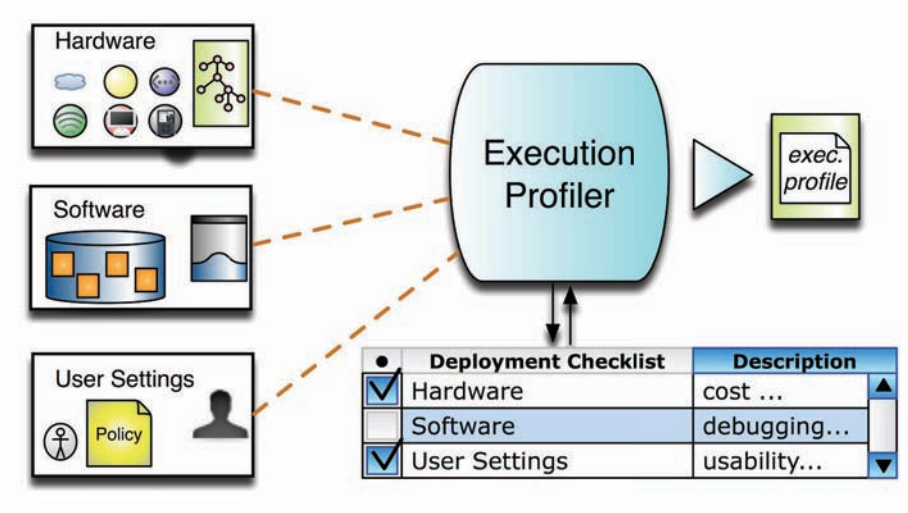

Figure 4.26: Execution Profiler organization

PART II:
CONTINUUM AS A CONTEXT-AWARE SYSTEM

5 THE ARCHITECTURE OF CONTEXT AWARENESS

In this chapter we describe the design of context awareness in Continuum software infrastructure, addressing the specific focus of the current thesis. The proposed architecture is built on the foundation established in the previous chapter, in terms of general infrastructure, models, and services. We start by giving an overview of the architecture, followed by a proposal for a context model. Our context model is based on an ontology, which is fully detailed in the sequence. Lastly, we discuss the Context Awareness subsystem and its services.

5.1 Overview

The major concern in this thesis is designing the Continuum architecture of context awareness. Our current proposal addresses many limitations of ISAM and EXEHDA middleware:

- The focus is mainly on context gathering, rather then on its modeling;
- The context is represented in a markup scheme, in an ad hoc manner. This choice is not very expressive and does not meet most context awareness requirements (STRANG and LINNHOFF-POPIEN, 2004);
- The user is forced to manually define context profiles for each application and every desired action;
- The representation of context is mixed with its use;
- The quality of context information is not considered;
- The context is considered in the previous follow-me vision, which replicates the user desktop session.

With these limitations in mind, we propose the context awareness architecture of Continuum, which has many advantages over the previous project:

- Considering a variety of contextual information;
- Providing a formal representation for context information in an application-independent manner (including the possibility of detecting implicit context);
- Considering user preferences;
- Storing a historical context data;
- Providing a distributed architecture for context storage and use;

- Making available a discovery service to find sensors;
- Easing the interoperability with a common communication mechanism based on web services.

To attain its objective, Continuum redefines two EXEHDA services. The discovery service is used to dynamically locate context components. Moreover, the monitoring service offers interaction with context sensors and the transformation of local data. It also provides notifications about relevant changes.

Several new services are proposed in the Context Awareness subsystem:

- *Processor*: responsible for filtering, i.e., eliminating errors from gathered data and making some low-level derivations of context. It transforms sensed data into a more high-level representation and also provides uniform representation for upper layers, thus hiding the details of sensors. In this service, the explicit context is obtained;
- *Aggregator*: aggregates contextual information related to real world entities. It incorporates user preferences and addresses some quality characteristics of gathered context. It also obtains the context related to entities;
- *Contextdb*: manages the storing and representation of context information. It stores generated data along with points in time when these data were created. Also, it stores all available context sensors according to the type of information they provide. This service enables the user to infers the context, reasoning over the database;
- *Context Action*: obtains information from sensors. This service can also be employed to subscribe to sensor changes, using event-driven communication (via the Communicator service). When a modification occurs in a sensor, the context action service informs its subscribers.

The first step when dealing with context is defining the model we will utilize for its representation. The next subsection covers this aspect.

5.2 Context Model

In order to represent, store, and utilize context data, we need first to define a context model. For this, we must have a machine-processable structure of the real world context. Among the various context model approaches (to know them refer to section 6.1.3) we chose an ontology-based model.

Ontology is an explicit formal definition of a common vocabulary and its relations to a specific domain of knowledge or discourse (GRUBER, 2007). Developing ontology may include several steps (NOY and MCGUINNESS, 2001):

- Creating *classes* to describe concepts in the specific domain;
- Defining a *class-subclass* hierarchy, arranging the concepts in a taxonomy;
- Characterizing the *slots* (also called roles or properties), i.e. the properties of concepts, such as features and attributes, and the allowed values for these;

- Describing the *facets* (also called role restrictions), which are the definitions of features that a slot can take, such as number of values (cardinality), value type, and range;
- Generating individual instances for each class, filling in the slot values.

The use of ontology enables structuring information and relationships, forming a *knowledge representation.* This description is the base for a Semantic Web (BERNERS-LEE et al., 2001): an extension of the human-targeted web that brings meaning to contents in a software-understandable way.

There are several languages for expressing ontology in the context of a Semantic Web. Among those, we highlight the OWL Web Ontology Language (MCGUINNESS and HARMELEN, 2004). OWL is a W3C standard for processing the content of information needed by an application. This standard adds to a stack of other W3C recommendations related to the Semantic Web: XML, XML Schema, RDF (Resource Description Framework), and RDF Schema. According to McGuinness and Harmelen (2004), OWL adds to this stack of standards additional vocabulary for expressing properties and classes. The choice for OWL seems natural, since it is a W3C recommendation. Besides, OWL promises high expressiveness, guaranteeing completeness and decidability, and is adopted by a vast range of tools (AGOSTINI et al., 2006).

We chose this model because ontologies are the most promising approach for context representation, thanks to its high expressiveness and reasoning techniques (BALDAUF et al., 2007; STRANG and LINNHOFF-POPIEN, 2004). In the next subsection, we will detail the ontology proposed for Continuum.

5.2.1 Context Representation

To define the ontology for context representation, we followed the methodology proposed by Noy and McGuinness (2001). We started by defining the domain and scope of the ontology we intend to propose for Continuum. The planned ontology will be used for modeling entities that are involved in a ubiquitous application. It will be used by a software infrastructure for ubicomp and must be general enough to allow modeling a vast range of applications. We should model all entities used by the software infrastructure, including places, people, and things. Additionally, we must consider the abstractions proposed for modeling the real world in Continuum (section 4.3). We can represent the contextual information in the environment and use it for context awareness and management (including adaptation), via the designed ontology.

To help defining the scope of the ontology, Noy and McGuinness (2001) suggest listing *competency questions*, i.e. issues and subjects which should be answered by a Knowledge Base (KB) built from the designed ontology. Appendix C presents the list of competency questions proposed for Continuum context representation.

As the second step of the methodology, we should consider reusing existing ontologies. In this process we consider some libraries of public and reusable ontologies. After our study, we chose to employ some existing ontologies in Continuum:

- DAML-Time: an ontology for representing temporal content and properties (HOBBS and PUSTEJOVSKY, 2003). It includes the representation of time

instants, time intervals, “before” relations, interval relations, linking time with events (provided that there is a specific ontology for event representation), and calendar units;

- SOUPA Space: an ontology for representing special relations, mapped from geo-referenced coordinates (CHEN et al., 2004). This ontology includes the description of geographical space, region, location coordinates, and GPS (Global Positioning System) information;
- SOUPA Event: an ontology designed to support event activities, including occurrences and schedules (CHEN et al., 2004). It provides the representation of an event along with its temporal and spatial information, using the ontology formerly presented;
- REI Policy: an ontology created for representing policies, including constructions for rights, prohibitions, obligations, and dispensations (KAGAL et al., 2003). In this ontology, we define actions and conditions, which are restrictions imposed on actions.

The following methodology stage consists in listing import terms in the perspective of the proposed ontology. We start the list with the terms used in section 4.3, where we proposed a representation of the real world in Continuum. In the sequence, we listed terms that were important to our work, such as time, location, and event. The complete list of terms, with their definitions, is presented in Appendix D.

Next, we defined the classes and their hierarchy in the proposed ontology. Figure 5.1 shows the result. In this representation, we used the same notation as Gu et al. (2004). The notation is very similar to the UML class diagram, except that classes are represented by ovals instead of rectangles. As in the original diagram, a hollow triangle shape on the supertype class end of the line represents a generalization (“is a” relationship). We employed the top-down approach (USCHOLD and GRUNINGER apud NOY and MCGUINNESS, 2001) to develop the class hierarchy: starting from the most general concepts in the ontology, and successively specializing them. We used this approach because we had a systematic top-down view of the domain we are modeling.

In the figure, we follow the representation proposed for modeling the physical world in Continuum (according to section 4.3), adding classes, which are needed for context description purposes. The hierarchy headed by the *Entity* class represents all distinguished entities in the real world: people, things, and places. We named all the classes using singular nouns, such as *Thing* and *Place*. Also, instead of creating a class named Person, to represent people, we chose to name this class *Being*, allowing for future extension, and then representing other kinds of living organisms beside persons, such as animals and plants.

The main Continuum abstractions, related to entities, are present in the figure: *CoPerson*, as a subclass of Being; *CoDimension* and *CoCell*, as subclasses of Place; *CoNode*, *CoBase*, *CoMobi*, and *CoGadget* as subclasses of Thing. We also added the class *Device*, as a superclass of all abstractions for things used in Continuum, to generically represent any kind of node. Finally, we added two classes for representing specific kinds of CoNodes: *Sensor* and *Actuator*. They represent information related to these particular devices.

Another group of classes in our proposed ontology does the mapping for the already-existent ontology suggested for reuse: *Event* class maps for the SOUPA Event ontology; *Policy* class corresponds to the main class in the Rei Policy ontology; *Timestamp* is the equivalent of the head class of the DAML-time ontology; and *Space* class is the same as the SOUPA Space class. Because of their equivalences, these previously mentioned classes are represented with an equivalence symbol (≡) in the figure.

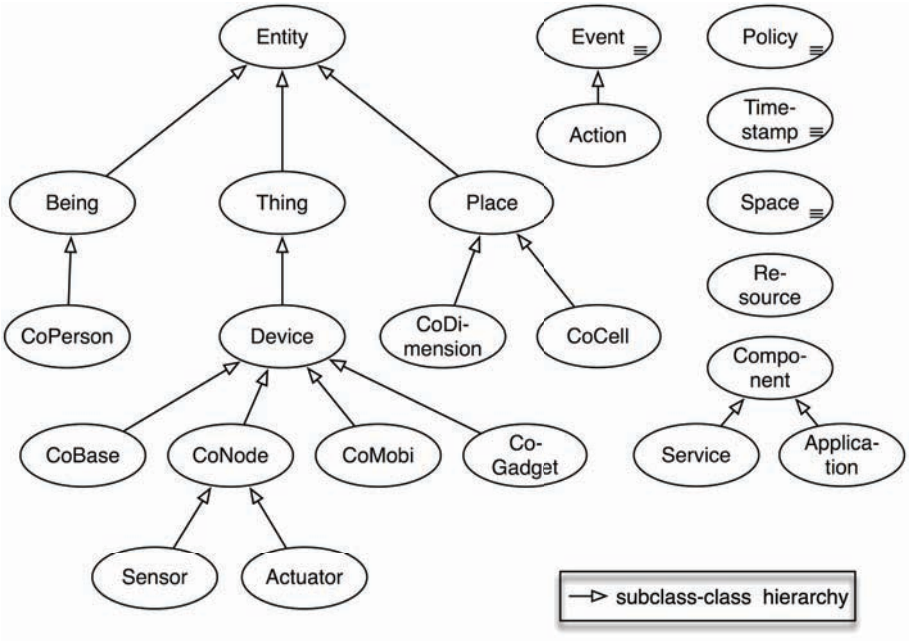

Figure 5.1: Class hierarchy of Continuum ontology

We further specialize the Event class, proposing the *Action* class. In our proposal, an Action is a particular type of event performed by Beings. The remaining classes describe some characteristics used by the middleware: *Resource* expresses the attributes of Devices, such as available memory, processor speed, and so on; *Component* typifies all software components, i.e. pieces of software registered in Continuum, which can be a *Service* or a plain software component, named *Application*.

The definition of the properties (or slots) of the classes is the next step in the applied methodology. We employ the same representation used before (from GU et al., 2004). The additional graphical symbol used for property is a filled triangle shape on the contained class end of the line. Also, we used a curved line for this representation, instead of the straight one used for generalizations. Figure 5.2 presents the class hierarchy with the main properties. Observe that not all properties are represented in the diagram, only those that indicate a relationship between classes. Other intrinsic and extrinsic properties were postponed to the implementation phase.

The main relationships between entities are presented in the figure: *performs*, indicating that a Being carries out an Action; *isAt*, denoting the Place of a Being or a Thing; *location*, signalizing the specific pinpoint of an Entity in a Space; *uses*, meaning that a CoPerson is utilizing a device; *aggregatedTo, associatedTo, and composed*, specifying which type of relation (aggregation, composition, or association) a Device has with a CoCell; *contains*, representing the Resources offered by a Device; *provides*,

associating software components with Devices; and finally, *emcompasses*, representing the composition among CoCells and for the outermost CoCells, their composition in a CoDimension.

The other relationships presented in the figure are related to Actions and Events. All Actions in our ontology should *follow* a Policy, to guide their behavior. Also, as an Actions is a specific type of Event, it always *occurs* with an associated Timestamp.

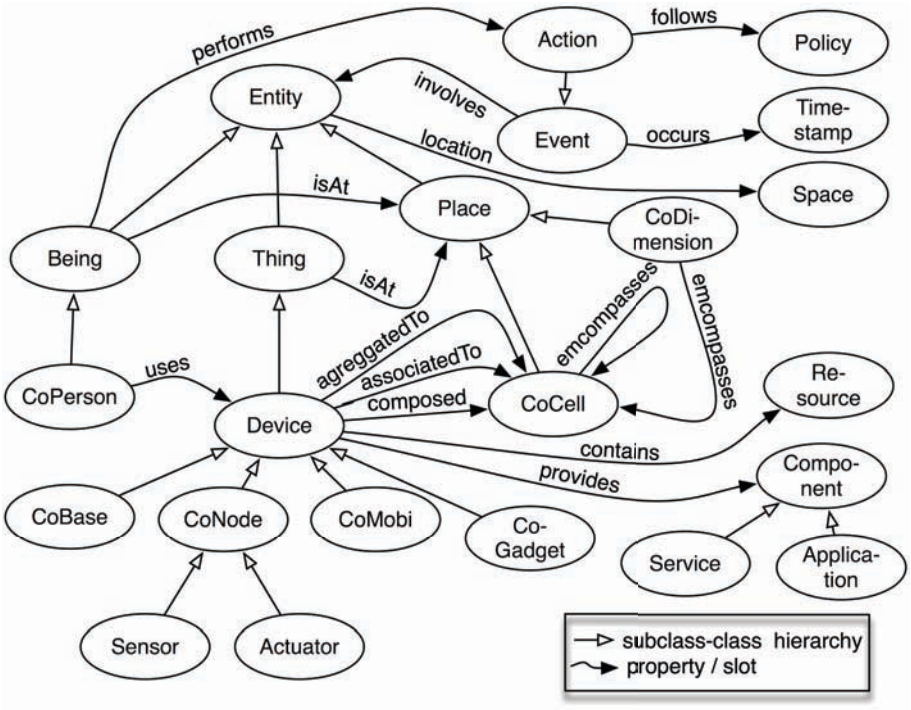

Figure 5.2: Continuum ontology with relationships

The last two steps proposed in Noy and McGuinness (2001), respectively to define the facets of the slots and to create the instances, are also delayed to the implementation phase. In the next subsection we analyze how the context is stored.

5.2.2 Context Storage

In Continuum, we propose the storage of context in a relational database. We choose this strategy, because databases are a very efficient way of finding information and their relationship. Additionally, it has been proved that we must use database for ontology storage to overcome various problems associated with the access of structured data (LUBYTE, 2007). For instance, the SQL (Structured Query Language) interface is an adequate way of querying for information.

In our work we propose the conversion of the Continuum ontology into a relational database. Some current works address the opposite problem: how to convert a relational database into an ontology (XU et al., 2006; LUBYTE, 2007) or how to use an ontology to make the integration of heterogeneous databases (DOU and LEPENDU, 2006). One of the exceptions is the work of Sugumaran and Storey (2006), which proposes the use

of ontology for database design. We follow some steps proposed in their work to achieve our context storage.

The first step in this process is transforming our proposed ontology into an entity-relationship (E-R) Model. An E-R model abstracts the real world into a group of objects, named entities, and their relationships. This is close to the idea of classes and relationships in an ontology model. Entities are created based on the classes identified in the Continuum ontology. The relationships among those classes become relationships among the entities. Once the E-R model is obtained, the second step is to convert it into a real database. There are well-defined rules to map this model into a relational database (SUGUMAN and STOREY, 2006). The design of the E-R model for Continuum is postponed until the implementation phase, since it must include all the attributes, which are the properties with their facets in the ontology.

Since we already have a method for storing ontology in a database, we can use SQL for the management of data in Continuum. SQL is a powerful and standard solution for data retrieval in a relational database. Their use in ubiquitous computing for dealing with context is a natural and straightforward one.

The next challenge in the use of context storage is how to establish a location and the distribution of the database. For this purpose, we suggest that each CoCell in the infrastructure keeps a context database. In our proposal, the context related to each cell is stored inside it. This includes all the entities that are presented in that cell (people, things, and other cells) and also all the remnant information related to it. The database is not just a snapshot of the current context situation. Since there is timestamp data, it keeps all the historical context of the cell. When a person or a device moves from one CoCell to another, the data is updated to register that this entity is not presented anymore in the source CoCell and it is stored in the database of the destination place.

Context information is, by default, never deleted. This introduces the overhead of storing a large database and also maintaining it. Furthermore, for availability purposes a single copy per cell can introduce some problems. We believe that current replication solutions and the growing size in storage devices, along with its decreasing in price, may help to address this drawbacks.

The Contextdb service, as already pointed out, is responsible for storing context. This service introduces a front-end to help finding context information. Whenever a query is made, it starts by searching in the current Cell's database and, if the data is not found, it propagates the query to the cells composed in it. In the case of another failure in finding the context, the propagation is then sent to its outer cell. Since the locality is always related to context, this seems to be a straightforward way of searching for context data. The idea is that all queries in Continuum follow the hierarchy imposed by the CoSpace organization.

Figure 5.3 shows a sample CoDimension of a University Campus, and presents the steps followed during a context search. In the example, a query ("Where is Lucas?") is made in the "Informatics Institute" CoCell to find a CoPerson, whose property name is equal to "Lucas." The first search is carried out in the current cell. Since the CoPerson is not physically present there, it propagates the search, initially in its composed cells ("Lab. 12" CoCell in the sample), then in the outer cell ("University Campus" CoCell). Since the CoPerson is also not present there, the outer CoCell propagates the search, following its inner hierarchy. In the end, "Lucas" is found at the "Cafeteria" CoCell,

inside the “Humanities Faculty” CoCell. Instead of querying a context information in the present, we could have questioned an historical information, for instance “Where was Lucas yesterday?” To satisfy this specific query, the infrastructure should search all CoCells presented in the sample CoDimension.

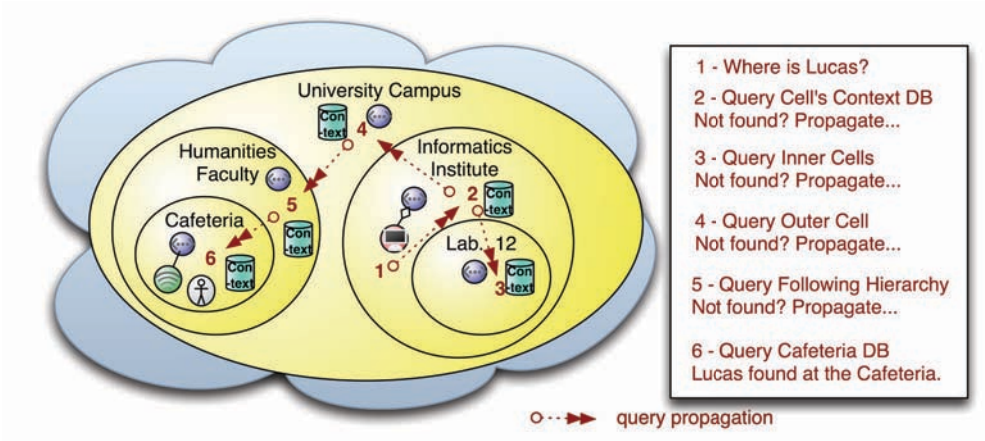

Figure 5.3: Sample of a context search

5.2.3 Context Utilization

There are several ways to use context in Continuum. To illustrate this use and to increase the understanding of the interaction among various context-related services, we present here the main workflows related to context. We used the UML activity diagram (FOWLER, 2005) to attain this objective. We believe that this notation is adequate for our purpose because we want to present the flow of work and interaction among the various services related to Context. Also, it represents a high level vision, which is also the aim of this section.

The first workflow related to context occurs when we have a change in some monitored condition, i.e., when a sensor detects some alteration. The activity diagram that represents this action is shown in Figure 5.4. The detection is taken by the Monitor service, which interacts directly with sensors and fetches the context change from it. This low-level context is transferred to the Processor service, which filters it, ensuring quality attributes and keeping only the relevant information, and passing it on to the Aggregator service. This service, based on the representation used, transforms this context. Then, context is associated with entities in the system and the considered user preferences. According to preferences, context can be shaped in a specific format. In the sequence, the Context Action service verifies if there is any subscriber to this context information; if so, changes are sent to the subscriber. Concurrently with the execution of this service, the storing of context data in the Contextdb is performed.

The next workflow starts in the context action service, when a subscription to a particular context is requested. Figure 5.5 shows the activity diagram. The following step is then querying the database to see if there is a sensor in it that could provide the information required by the subscription. If there is no sensor stored, the Discovery service is activated to try to find a sensor that can capture the context needed. If the sensor is found, either in the database or by the use of a discovery mechanism, the subscription is associated with the sensor in the context action service. When we

discover a new sensor, there is the additional possibility of not finding the desired context. In this case, the information that a context cannot be obtained is returned. In contrast, when a sensor is found, it is always stored in the database for future use. A situation that is not represented in the workflow may occur when a sensor fails or ceases to be available. This condition is detected by the Monitor service, which uses the Contextdb to modify the status of a sensor. Also, subscriptions to this sensor are either cancelled or transferred to another equivalent unit. When a sensor becomes unavailable, future subscriptions cannot obtain it directly from the database, but rather have to carry a discovery operation to find a new one or to check if the sensor became functional again.

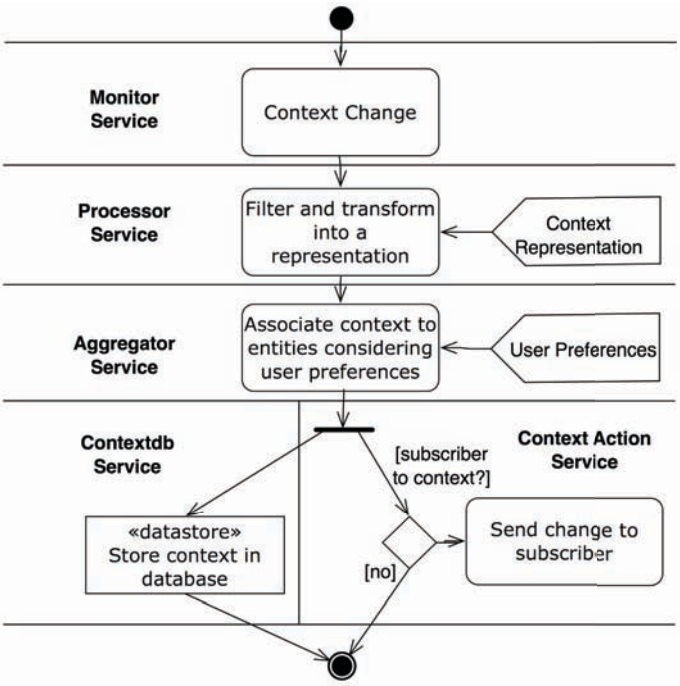

Figure 5.4: Activity diagram of context change

Another possible workflow is context search (activity diagram in Figure 5.6). The search starts by trying to find a context, reasoning over the database. If the inference is successful, Contextdb returns the context directly. If not, and if the desired context is not satisfied with a specific resource, it tries to locate a sensor in the database that could meet the request. If it is found, a probe is done into the sensor (presented in the next activity diagram) and the context is returned; if not, a discovery is issued and in the case that a sensor is found, it is stored in the database and a probe is started. If the sensor is not available, the information that a context cannot be found is returned.

Sometimes, the context we are trying to find is itself a resource, and not obtainable by a sensor (e.g. a particular device). In this case, if the resource is not found in the database, instead of searching for a sensor, Continuum tries to directly discover the resource itself. If it is found, it is stored in the database and the resource returned. If not, the information signalizing that the context cannot be found is then sent back. The chief difference between this workflow and the previous one is the synchronous nature of the former in contrast with the asynchronicity of the latter.

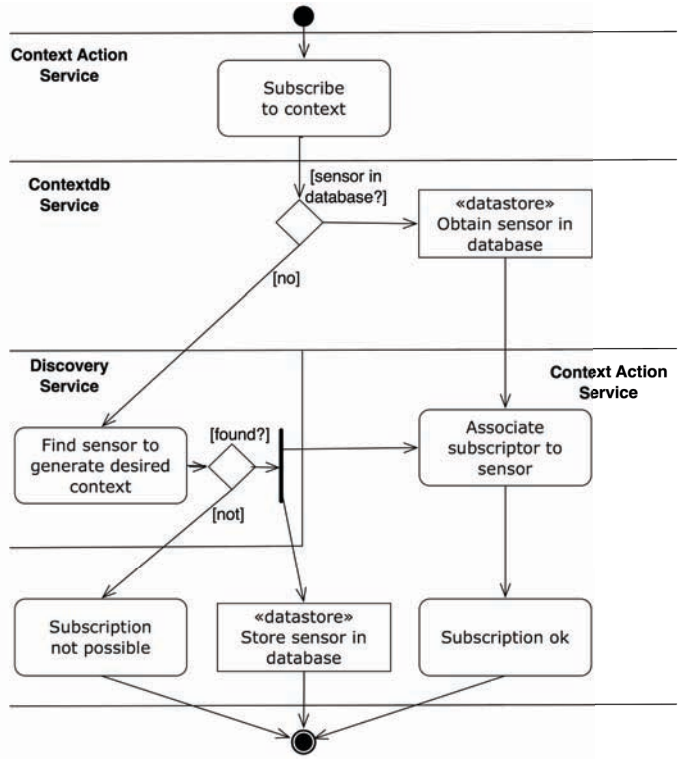

Figure 5.5: Activity diagram of context subscription

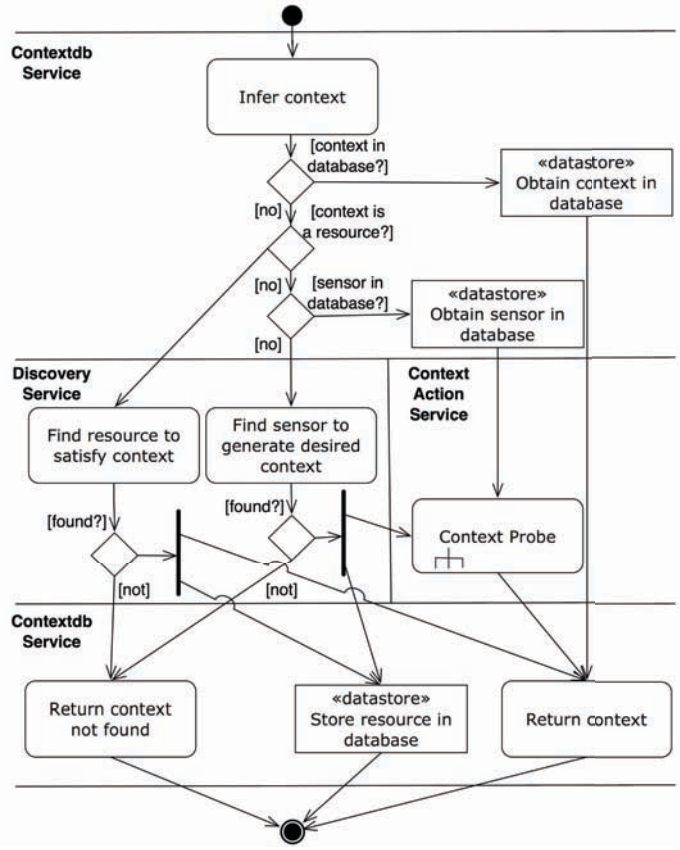

Figure 5.6: Activity diagram of context search

The last activity diagram (Figure 5.7) shows a context probe workflow. This operation consists of a direct synchronous verification of context information in a sensor. It starts in the Context Action service, where information from the sensor is obtained by instructing monitor service to issue a probe in the sensor. After the sensor has returned the information, it follows the same steps as the first workflow: filtering and transformation (Processor service), association to entities and incorporation of user preferences (Aggregator service), and the concurrent actions of storing in the database (Contextdb service) and returning context (Context Action service).

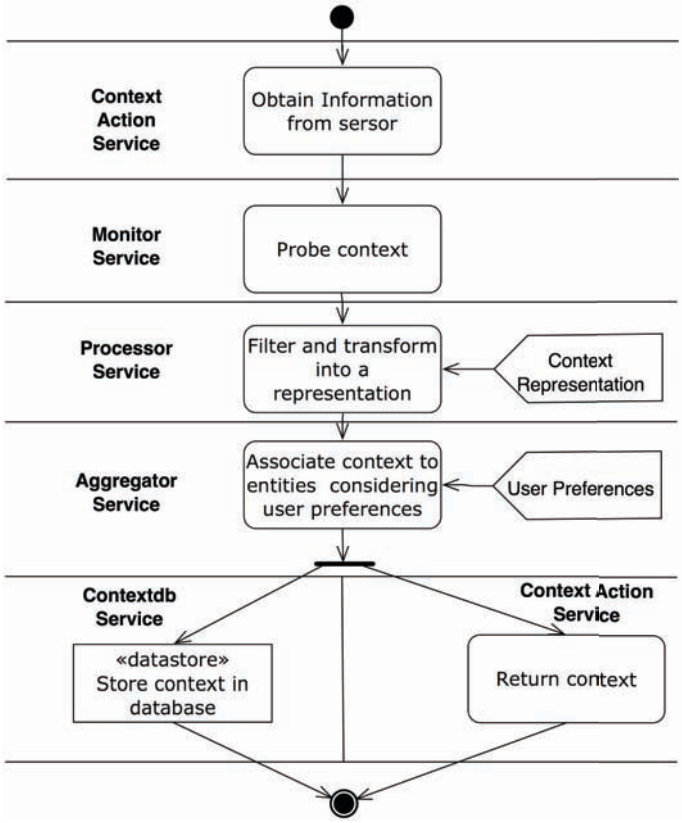

Figure 5.7: Activity diagram of context probe

In the next section, we further describe the context services, presenting each individual service, along with the description of its interface.

5.3 Context Awareness Subsystem

The Context Awareness subsystem congregates all services related to context in the Continuum infrastructure, as shown in Figure 5.8. There are two services based on previous EXEHDA components: Monitor and Discovery. These components have been further improved in graduation research projects done at UFRGS (refer to FEHLBERG, 2007 and SCHAFFER FILHO, 2005, respectively). The other services have been created specifically for Continuum, considering the various aspects needed to address context awareness (as defined in the comprehensive architecture in chapter 3).

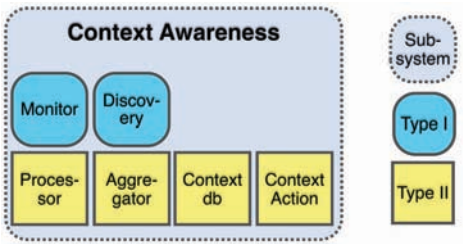

Figure 5.8: Context Awareness subsystem

5.3.1 Monitor

The Monitor service is in charge of interacting directly with sensors and extracting raw data. It is based on two previously EXEHDA services, namely Collector and Monitor (YAMIN, 2004). In Continuum, we choose to integrate both services because the differences between them were very subtle and, in the new architecture proposed for context awareness, no visible gain is obtained by splitting those services. Additionally, we considered the improvements made by a graduation research done at UFRGS, named MultiS (Multi-Sensor Context Server) (FEHLBERG, 2007). The interface proposed for the Monitor service is presented in Figure 5.9. The new operations are presented in bold.

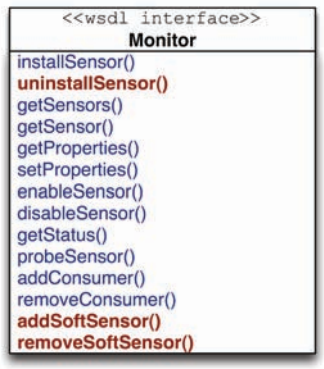

Figure 5.9: Monitor service interface

The first function, `installSensor`, is used to add and initially configure new physical sensors to the infrastructure. After the installation, configuration can be altered by the `setProperties` operation, or checked with `getProperties`. Later, if we want to remove an installed sensor, we can employ `unistallSensor`. To list information related to a specific sensor, we use `getSensor`. If we want information for a group of sensors that adhere to certain characteristics, the `getSensors` function can be employed instead. To activate / deactivate sensors, we use `enableSensor` and `disableSensor`, respectively. Sometimes, we just need to check if a sensor is enabled. This action is accomplished using `getStatus`.

To obtain the raw data, the first alternative offered is `probeSensor`. This function returns the values sensed at that particular time through synchronous communication. Asynchronous interaction is also possible by the use of consumers. A consumer subscribes to a particular sensor, and all new information is notified via a callback. The

`addConsumer` and `removeConsumer` operations deal with the addition / deletion of consumers to the infrastructure. This asynchronous communication is implemented using the Communicator service previously presented (section 4.6.1.3).

The last set of functions is used to create (`addSoftSensor`) or delete (`removeSoftSensor`) software sensors. These are special kind of sensors that are implemented in software. They are also named virtual sensors (INDULSKA and SUTTON, 2003). They are created as Continuum software components and gather conditions in the system that can be obtained without special hardware, such as available memory, processor load, and so on. A parameter to these functions defines the software component that acts as a sensor. These operations are conceptually the same as the function pair to install / uninstall sensors, the only difference is that they manage software sensors. After the addition of software sensors, the same group of operations that manage normal hardware sensors can be employed to both types of components.

5.3.2 Discovery

The Discovery service is responsible for resource detection in Continuum infrastructure. A resource in Continuum is either a device represented in the modeling of the physical world in the infrastructure, such as CoNode, CoMobi, etc., or special purpose devices, i.e. printers, scanners, monitors, among others.

This service is based on a previous graduation research project done at UFRGS, named PerDiS (Pervasive Discovery Service) (SCHAEFFER, 2005). The work improves the preceding EXEHDA Discovery service. PerDiS deals with generic resources, which are specified using XML. In Continuum, we extend the use of PerDis to find sensors. In EXEHDA, sensors were added manually to the middleware. The interface proposed for the Discovery service is presented in Figure 5.10.

Figure 5.10: Discovery service interface

The `registerResource` and `removeResource` operations manage the addition / deletion of resources in Continuum. As a parameter, these functions receive the resource description in XML, according to the specification defined in Schaeffer (2005). Resources in Continuum, as in PerDiS and EXEHDA, can have an associated lease, so that they periodically renew their register. This is accomplished using `renewResource`. A lease is a time limit established between the component that registers the resource and the Discovery service. Before this time expires, the component must renew its lease with the service; if it fails to renew, the service removes the resource.

Differently from PerDiS and EXEHDA, there is no obligation to use lease for all resources. More stable resources could not use this feature, reducing the number of

messages in the system. Thus, we reduce the utilization of network bandwidth and energy, even if we lessen the timeliness. If at some point a component becomes unavailable, it can be detectable and then a discovery operation can be launched again to find an equivalent or approximate resource. This detection occurs by using a timeout mechanism. After a limited number of retransmissions, the infrastructure assumes there has been a failure, even though it cannot distinguish between a network and a process failure. Moreover, this failure model does not guarantee that the message sent will be received.

To discover a resource, we use `findResource`. As a parameter to this function, we specify the search criteria. The format for these criteria was defined in Schaeffer (2005), based on state of the art proposals for service discovery, and combining characteristics of expressiveness and interoperability. Furthermore, this operation enables the user to find resources by their name (as in a white pages service) or by some attribute (as in a yellow pages service). The search criteria can also be used to asynchronously find resources that adhere to it. For this purpose, we can use `addListener`. This operation adds a listener for a particular type of resource using the Communicator service. The listener can be deleted using `removeListener`. Whenever a resource is found, either synchronously or asynchronously, an identifier is returned to the caller. This means that the resource was found, but does not imply in any allocation or reservation (YAMIN, 2004; SCHAEFFER, 2005).

The allocation of a resource is accomplished by calling `requestResource` (previously named `allocResource` and presented in EXEHDA's ResourceBroker service). This additional step informs the Discovery service that a certain component is trying to use a resource. Depending on the type of use, shared or exclusive, the access can be granted or denied. When a resource is allocated exclusively to a component, others cannot use it. To all mutual exclusive access, a time limit is associated; when it expires, the user of the resource is notified and its access revoked (YAMIN, 2004).

If a resource is being shared by more than one component, this access must be canceled before the Discovery service grants exclusive access. However, there is no problem in granting additional shared access when a resource is already being jointly used. On the other hand, the exclusive access is only possible for resources that can cause race conditions or other related synchronization problems. This is specified when a resource is added to the Discovery service.

Discovery and Monitor are considered low-level services in the Continuum infrastructure. Programmers should use the Contextdb and Context Action services instead, to respectively find context information, and obtain or subscribe to sensor data. These services then employ Discovery and Monitor as needed. Another consideration, specifically related to Discovery, is that in the original planning of PerDis, user preferences could be used in the discovery process. We moved the consideration of user preferences to another service (Aggregator), in order to benefit from their use in various resources simultaneously.

5.3.3 Processor

This service is responsible for converting low-level sensed data into a high-level context representation. This is a requirement called *context transformation*, as already

defined in section 3.3.7. It converts the data received by sensors to a format that can be stored in the database, using the ontology proposed for context representation. In this service, it is also possible to obtain explicit context directly from sensors. The service interface is presented in Figure 5.11.

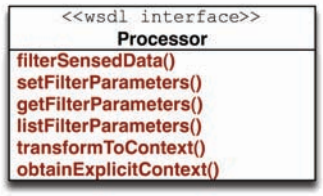

Figure 5.11: Processor service interface

The first four operations control the filtering process of sensed data. The `filterSensedData` function takes the data received from sensors and returns a filtered new version. This process involves modifying the source data according to parameters defined by `setFilterParameters`. To obtain a specific parameter, we use `getFilterParameters` and, to obtain all configured parameters, we employ `listFilterParameters`.

There are five possible parameters for the filtering operation: *noiseReduction*, which defines a function to be applied in the sensed data to reduce interference; *validRange*, which specifies an interval for the sensed data; *errorRange*, which defines an interval for the sensed data to be considered as erroneous; *ttl* (time to live), which characterizes a time limit for the validity of sensed data; and *accuracy*, which describes the deviation degree from the correct solution for the sensed data (in case of hardware sensors, it is normally specified by the manufacturer). As observed, some parameters alter the final sensed value, while others aggregate properties to the sensed data. These properties can be used as quality metadata. More than one parameter can be added to a filtering operation; if there are conflicting parameters, an error is returned.

To generate a context from the raw data, we use `transformToContext`. This function produces a context in the format proposed by the ontology definition. If the sensed data has been previously filtered, it also incorporates the added properties and considers as input the value already modified by this filtering process.

The last function, `obtainExplicitContext`, returns the explicit context. This consists of the context returned directly by the sensor, after the filtering process. We added this function, because we believe that it could be useful for applications requiring to consider the data acquired by a specific sensor, without additional inference, aggregation, or transformation.

5.3.4 Aggregator

The Aggregator service combines context information from various sources in a more detailed and accurate context (*context synthesis*, according to the requirement defined in section 3.3.7). In this process, the service considers user preferences. The Aggregator also provides context generated by the combination of multiple sources. The interface proposed for this service is in Figure 5.12.

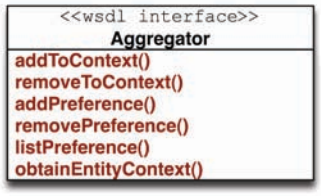

Figure 5.12: Aggregator service interface

The `addToContext` and `removeToContext` operations respectively insert and delete context associated with a particular entity. By using the first operation, we can merge context data obtained from different sources. The second operation removes sensed data from previously generated context for a specific entity. With the combination of context from different sources, we may obtain a better context result.

The way in which data will be combined to form the final aggregated context is configured by using `addPreference` and `removePreference`. While the former adds a new preference to the service, the latter removes it. In case of conflicts, an error is returned. To obtain a set of preferences related to a particular aggregation, we can use the `listPreference` function.

There are seven possible preferences we can specify to obtain the aggregated context. Some establish equivalence properties, i.e., settings that guide how the infrastructure will accomplish the aggregation. Other preferences define corrections to be applied during aggregation. The preferences are configured according to different parameters: *entiEqu* – contexts are considered equivalent if they belong to the same specified entity or entities; *posiEqu* – contexts are aggregated if they are at the same location or location range; *timeEqu* – aggregation is realized only if it occurs at the same time or within the same time interval; *condEqu* – the equivalence is only tackled if some user specified condition, passed as an additional parameter, is observed; *propCor* – in the aggregation process corrections are made according to filter parameters (validRange, errorRange, and ttl) ; *acurCor* – the filter parameter accuracy is considered in the aggregation process following some preferred configuration; and *udefCor* – additional corrections are made to the aggregated value according to user-defined settings.

The last function (`obtainEntityContext`) returns context related to a specific entity. According to the definition of Indulska and Sutton (2003), this operation acts as a logical sensor, combining information from different sources, either physical or virtual.

5.3.5 Contextdb

Contextdb is a service in charge of managing the context database. It has the traditional SQL data manipulation language. It also introduces an additional function for inferring context, reasoning over the database. Contextdb covers the category of context service called *context query* (as defined in section 3.3.7), and its interface is presented in Figure 5.13.

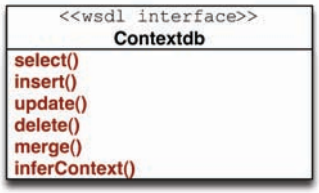

Figure 5.13: Contextdb service interface

The `select` operation is used to retrieve data from a table or group of tables in a database. The function `insert` adds new rows to tables, while `delete` removes them. To modify values from rows, we employ `update`. On the other hand, `merge` is used to combine data from multiple tables. Due to the fact that SQL queries and data manipulate operations are widely known, we will not further describe their use. The idea is to employ the ANSI/ISO standard SQL specification (INCITS/ISO/IEC 9075).

The last function is `inferContext`, which is employed for reasoning over the database, extracting information not explicitly stored in the ontology model. The operation receives query requirements and applies inference rules to the stored data. There are two types of inference rules used (LOPES, 2008): consistence rules, which verify data consistency, avoiding incoherent queries; and extension rules, which infer knowledge from data modeled by the ontology. The second rule allows to semantically extend the ontology.

The detailed model for inference in an ontology database has been developed by LOPES (2008) in his graduate research project at UCPEL. This project still made use of EXEHDA, since Continuum had not yet been made available at the time. Later, in the work PESSUTTO (2008a) we extended Lopes's model, adapting its implemented prototype to the Continuum project. This prototype was then used to obtain of the results presented in section 7.3.

5.3.6 Context Action

The Context Action service is used as a high-level interface to manipulate context information. Using this service, we can obtain context from sensors and perceive context changes in Continuum infrastructure. Nevertheless, this service does not fetch context from the database, so all information is obtained from sensors. Besides, it uses other services according to the issued action. The service is also in charge of *context subscription and delivery* (according to the requirement defined in section 3.3.7). The Context Action interface is shown in Figure 5.14.

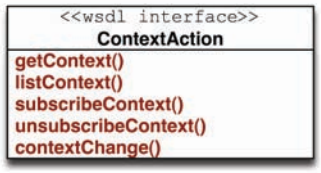

Figure 5.14: Context Action service interface

The `getContext` function returns context from sensors. This is the operation used in the context probe activity diagram (previously shown in Figure 5.7). As seen in the

diagram, not only does the function get the information from sensors, but it also carries all the subsequent operations: processing, aggregating, and storing the context in the database. The `listContext` operation tries to find all context data related to a particular entity (passed as an argument). Note that it does not include the information stored in the database, but rather the data that can be fetched from physical, logical, or virtual sensors. New context data gathered throughout this process is always stored in the database for future use.

The `subscribeContext` and `unsubscribeContext` operations deal with the subscription of context information, using the Communicator service (described in section 4.6.1.3). The former provides a way of subscribing to information from sensors, while the latter undoes this operation. When subscribing, it may use three kinds of sensors: physical and virtual ones, managed by the Monitor service; and logical ones, generated by the Aggregator service. This operation is illustrated in the context subscription activity diagram (formerly shown in Figure 5.5). Whenever context data is asynchronously generated, a callback to `contextChange` occurs (as previously demonstrated in context change activity diagram – Figure 5.4).

The aim in providing operations for dealing with data obtained directly from sensors, without considering the database, is three-fold: first, it has to do with the fact that sometimes we want to consider only real-time explicit data, without historical or inferred context; second, if we cannot find a context in the database, we still can try this option, as previously demonstrated in the activity context search diagram (Figure 5.6); third, it is a possible method for coordinating an operation that involves various services: Monitor, Discovery, Processor, Aggregator, and Contextdb.

6 CONTEXT AWARENESS: DISCUSSION AND RELATED APPROACHES

We present in this chapter a discussion of the state of the art in context-aware systems and reflect on the experience of designing Continuum-related architecture, as described in the previous section. We start by presenting the main design characteristics of context-aware solutions, proposing a multi-tiered model. In the sequence, we describe some of the most representative projects related to the area. Finally, we assess the Continuum Context Awareness subsystem, by comparing it to the discussed projects, highlighting their main features and limitations.

6.1 Context-aware design principles

A context-aware system usually has a set of services to deal with context (contextual services, as previously defined in section 3.3.7). Based on these services and on previous work on the subject (ADELSTEIN et al., 2005; AILISTO et al., 2002; DEY et al., 2001; BALDAUF et al. 2007), we propose a multi-tiered model for context-aware systems (Figure 6.1). Analyzing each tier, we can discuss the main design principles related to context awareness.

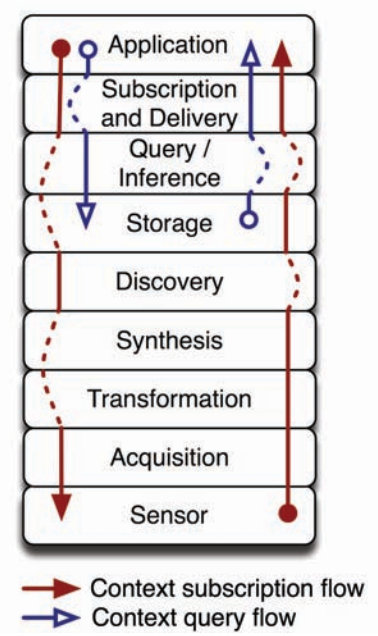

Figure 6.1: Multi-tiered model for context-aware systems

Figure 6.1 presents the main components found in context-aware systems. Various systems have been recently proposed, which differ in architecture, scope, aim, and also in the name they use for the tiers. Additionally, some systems do not include all the context services presented in our model, while others integrate some tiers into fewer components.

The bottom-most tier (*sensor*) consists of a collection of sensors that gather the raw data. These sensors are coordinated, parameterized, and controlled by the *acquisition* layer. The next tier carries out the *transformation* of data obtained from sensors into higher-level information. The following layer is responsible for *synthesis*, i.e. aggregating the context information, generating a more detailed context. Then, the model presents a *discovery* layer that is employed in the finding of sensors and other resources. The sixth tier represents a *storage* that accumulates the context data. Normally, there is inference atop of this data (or at least straightforward queries) which is accomplished by the subsequent tier, named *query* / *inference*. Next, the *subscription* and *delivery* layer offers event-communication mechanisms for notification of context events. And finally, the top-most tier represents the context-aware *application*.

The model shows two common flows of interaction among the tiers. The first one, represented by a line with a filled shape on both ends, represents the context subscription action and the subsequent occurrence of the event. Note that some tiers are not used (represented by a dotted line) in the top-down path, only in the return of the gathered context data. Furthermore, there are tiers that are not used in this flow. The second showed flow, denoted by a line with a hollow shape on both ends, indicates a query on the database. This is usually used to obtain historical data or, in some systems, to apply a reasoning engine to the available context information.

It is a noteworthy matter that these two flows are not the only possibilities in a context-aware system; however, they represent the two most common operations related to the subject. The next subsections detail the main design principles involved in each tier.

6.1.1 Sensor

There are three types of sensors that can be used in a context-aware system (INDULSKA and SUTTON, 2003): physical (or hardware) sensors, virtual (or software) sensors, and logical sensors, indicating that data can be provided either by a physical or a virtual sensor. A virtual sensor obtains data from a software component, while a physical sensor fetches data using a hardware device. A logical sensor, on the other hand, combines assorted sensors with additional information obtained from a database or other sources (BALDAUF et al., 2007).

Individual nodes, usually physical sensors, can be combined in a more complex arrangement, generating a sensor network, which mingles relatively simple sensors with real-time, low-level manipulation and analysis (CHONG and KUMAR, 2003). Usually, the dynamicity of a sensor network is very high, due to the possibility of adding and removing nodes from the network during execution.

Sensors can also be classified according to the nature of the obtained information. The main types of context elements obtained by sensors are location (positioning of an entity), motion (direction, speed, etc.), activity (progress of an action, consumed

resources, etc.), physical conditions (human vital signs, temperature, luminosity, etc.), emotional state, reachability (getting through to an entity), and surroundings (what is in some environment) (HASELOFF, 2005).

6.1.2 Acquisition

The main purpose of this tier is to isolate the top layers of the system from the complexity of gathering data. Some systems do not present acquisition as a separate layer, reducing the possibility of reusing sensors, and jointly handling both the obtaining of data and its use or representation.

In the design of the acquisition tier we must deal with the following concerns related to sensors (DEY, 2000): installation, configuration, ways of communication, and type of sensed data. Some sensors offer an API to facilitate interaction, while others do not, so that the designer has to determine how to operate them.

One interesting solution for acquisition is the *context widget* used in Context Toolkit (DEY, 2000), which applies the concept of widgets used in GUIs. When used for this purpose, they are employed to hide the specifics of input devices from the programmer, causing minimal impact on applications. Analogously, context widgets provide the same benefits: hiding the complexity of actual sensors, abstracting context information, and supplying reusable and customizable context acquisition (DEY et al., 2001).

6.1.3 Transformation

This tier, also called context interpretation, transforms the information received from the acquisition layer into a machine processable format. The main concern in this process is which context model to use in the representation of context.

There are various data-structure modeling approaches employed in the representation of context (STRANG and LINNHOFF-POPIEN, 2004; BALDAUF et al., 2007): key-value models – using pairs of values to represent context; markup-scheme models – applying a hierarchical data structure with markup tags, such as XML; graphical models – utilizing a visual representation, for instance UML; object-oriented models – exploiting OOP techniques in context representation; logic-based models – employing a logical definition, defining context as facts, expressions, and rules; ontology-based models – applying ontology for denoting context. Among all the approaches, the latter, based on ontologies, is the most expressive, and also the solution that better satisfies the requirements of context modeling (STRANG and LINNHOFF-POPIEN, 2004).

According to the modeling solution employed for context representation, a particular mapping strategy should be applied to convert the sensed data into context information. After that, context information may be directly stored, or sent to the synthesis tier, depending on the system design.

Another design feature related to transformation is whether the system allows the filtering of sensed data. This data is prone to sensing errors and noise, and the processor tier can consider quality metadata, such as certainty and freshness estimates, to address these flaws (HENRICKSEN and INDULSKA, 2006).

6.1.4 Synthesis

Synthesis or aggregation is the process of composing individual context information related to a specific entity. The module identifies different sources of information and combines contexts to produce a result that is more precise and easier to use.

The main design principle in this tier is related to the process of aggregation: what the rules are, or what the domain is, for combining context information (PARK and LEE, 2005). A simple alternative is to combine data that refer to the same context element, for instance, to employ some entity ID for matching them. When the layer receives various sources of context information with the same entity ID, it can put them together. Another way of aggregating data is to combine all sensed data that are associated with a common location and timestamp. A third method is to employ a user-defined combination, i.e. the user determines the parameters to be applied in the synthesis process. Finally, another possibility is to carry out the aggregation, using generic processing steps (CHEN and KOTZ, 2002a). For instance, the Solar system uses this method with a graph-based abstraction, in which data flows though a graph of operators and, in the end, becomes an aggregated context (CHEN and KOTZ, 2002b).

An additional devised characteristic of the synthesis layer is the possibility of considering the quality of context, in order to detect ambiguity or unknowns. This is especially useful when context information generates conflicting values, in the composing of context representation (HENRICKSEN and INDULSKA, 2006). To tackle this matter, one possible solution is to consider preferences in the synthesis process.

6.1.5 Discovery

This tier is responsible for dynamic search and finding resources at run time. When addressing context awareness, the resources we are particularly interested in are sensors. Some systems offer services or components that accomplish this process, while others need to employ built-in sensors or to rely on a pre-configuration. In this case, failures and the addition of new sensors need to be manually tackled.

One characteristic of the discovery mechanism is the method used for detecting the dynamic availability of a resource. It is common to employ leases or some pooling mechanism. Another design principle is the method that is used to look up resources. Normally, components can be queried using white pages (search by the component name) or yellow pages (search by a specific attribute) (BALDAULF et al., 2007).

There are various other characteristics related to the discovery layer. This includes the topology used, transport mechanisms for message exchange, scope, search facilities, and security mechanisms (EDWARDS, 2006). We will not describe these aspects here, because they are not directly related to context awareness. For more details on these other design choices, refer to Edwards (2006).

6.1.6 Storage

The storage tier keeps the context information. Some systems maintain historical context data that, according to Baldaulf et. al. (2007), “may be used to establish trends and predict future context values”. The main design principle is related to distribution and placement. The solutions vary from centralized to totally distributed storage. Trade-

off alternatives are also possible, such as employing hierarchical or partially distributed solutions. The concept of localized scalability (SATYANARAYANAN, 2001) may also be applied here.

Another design characteristic is related to the storage method. Probably the best solution is to employ a database. This has the advantage of persistence and also of using SQL to manage data. Additional options include the possibility of maintaining context information in volatile memory or using files with a system-specific structure. Other systems store context data in tuple space, which improves persistence and synchronization.

6.1.7 Query / Inference

This layer comprises the management of context information. It can vary from simple mechanisms for querying the data, to powerful reasoning, including the inference of deduced context. This option constitutes the first design characteristic of the tier: presence or absence of an inference engine and a KB. This characteristic has an influence on the type of generated context. The tier could infer only explicit context, or it could also infer implicit context, i.e. context generated from reasoning.

If an inference engine is available, the characteristics are associated with the interpreter used: syntax of inference rules, how these rules are stored, capabilities of the interpreter, etc. Some systems use popular semantic web toolkits, such as Jena (CARROLL et al., 2004).

Another characteristic of the tier is the query language employed. It can vary from a system-specific language to the use of a standard query language, such as SQL. Some systems use other options for consulting context information and for defining inference rules. For instance, Gaia employs the first-order logic as a means to specify high-level rules and queries (RANGANATHAN and CAMPBELL, 2001).

Finally, there are security and privacy issues that should be observed in the access of sensed context data. This is significant, since context may include sensitive information on people. The mechanisms available in systems vary from simple (users have ownership of context information) to more sophisticated access control (policy language to control context use). Another solution used is authentication, for proofing the identity of users (BAULDAUF et. al, 2007). Some systems also have particular solutions that address this issue. For example, Gaia has a secure tracking system that further protects location privacy (ROMÁN et al., 2002).

6.1.8 Subscription and Delivery

Normally associated with context-aware system, is the capability of subscribing to specific context change. Typically, this operation is based on a publish-subscriber model. Because of that, we chose subscription and delivery for the tier’s name. This layer gives asynchronous communication to the context-aware system, while the former tier generally constitutes a synchronous operation.

When subscribing to a context change, some systems allow the specification of certain conditions to be observed for the occurrence of an event. Furthermore, particular attributes that are of interest may be indicated. Another design issue is whether it is

possible to subscribe to changes related to specific entities or only to individual sensors. This defines the subscription element: a sensor or an entity-related context.

The subscription and delivery tier benefits from the existence of a discovery tier in the system, because this makes the finding of more appropriate and up-to-date context sources more flexible.

6.1.9 Application

The last tier is represented by the context-aware application. The main design principle in this layer is connected to the way programs make use of context. This is strictly related to the context management method, and particularly to the adaptation strategy.

There are three general strategies for adaptation, as already defined in section 3.3.8.1: laissez-faire, application-aware, and application-transparent. In the latter, since the adaptation is only a system responsibility, the application involvement with context is non-existent. In the former, the application is responsible for all management of context, without system support. The intermediary approach seems to be a good trade-off, in which context is managed by the application, with some system support. This is the tendency in the context-aware systems, although some of those systems do not directly address the adaptation process itself; only context-aware issues are covered, while context-management characteristics, such as adaptation support, are not directly tackled.

In the application-aware strategy, since applications manage context information, they should use an API to interact with the context subsystems. The set of operations offered to the application is another design principle that varies from system to system and depends on the tiers and features that are implemented. Normally, jointly developed with the context-aware systems, some applications or case studies are proposed, in order to validate or demonstrate the functionality of their projects.

6.2 Related Context-aware Systems

Here we present some of the most representative context-aware architectures recently developed. Due to the large amount of ongoing proposals, we will not give a complete description and evaluation of all the systems. Besides, there are several articles that cover a comprehensive examination and comparison of various proposals (BALDAUF et al., 2007; CHEN and KOTZ, 2000; DEY, 2000; HASELOFF, 2005; STRANG and LINNHOFF-POPIEN, 2004).

Therefore, we restrict our analysis only to three significant projects in the field that are particularly relevant to our thesis. The chief criteria of choice was the impact generated by the articles that describe these projects and the number of citations they produced.[25] As an additional restriction, we considered only works that propose general solutions and which were published in the past ten years. Furthermore, we give only a brief description of each system and present references for supplementary information.

---

[25] This information was obtained at <http://citeseer.ist.psu.edu/>.

6.2.1 Context Toolkit

Context Toolkit (DEY et al., 2001; SALBER et al., 1999) is a programming framework for the development of context-aware applications that has been developed at Georgia Institute of Technology. The main objective is to provide a reusable solution for context manipulation, improving the development and deployment of interactive context-aware software. In the toolkit, a distributed architecture is proposed, using P2P communication, and five abstractions are used: *widgets*, *interpreters*, *aggregators*, *services*, and *discoverers*.

Widgets are derived from the GUI concept. As already pointed out in section 6.1.2, they act as an acquisition tier. Furthermore, the subscription of context data is also addressed by this component, as well as by a mechanism for synchronous communication. Interpreters, on the other hand, are responsible for the transformation of low-level sensed data into higher-level context information, possibly combining various sources of data in this process. Other abstractions, named aggregators, gather context information related to an entity, thus making it easier for applications to use them.

The service abstractions are in charge of executing behaviors in the environment. They are incorporated in widgets, and change the environment by using actuators. Finally, discoverers are centralized mechanisms employed to dynamically find components and to maintain a registry of capabilities available in the system. Among the components discovered are widgets, interpreters, aggregators, and services.

A public release of Context Toolkit, developed in Java, is available.[26] In spite of using Java, the prototype was developed employing programming language-independence mechanisms, so that it could easily apply components developed in other languages (DEY et al., 2001). For instance, there are widgets and applications written in C++, Frontier, Visual Basic, and Python.

Some applications were developed in the Context Toolkit to demonstrate its functionality. The first, called In/Out Board, shows the status of people in a building, if they are inside or outside it. Additionally, it presents the day and time when a person has last entered or left the building. Another interesting application is named DUMMBO (Dynamic Ubiquitous Mobile Meeting Board), which provides a digitizing whiteboard to support the capture of spontaneous meetings and the access to captured data (SALBER et al., 1999).

The project makes it possible to develop context-aware applications and offers a tool that can easily be employed for gathering context. Many aspects of acquisition, transformation, synthesis, discovery, and subscription/delivery are tackled. However, there is no formal model for context; they are represented as a set of widget attributes. In addition, context acquisition is not separated from its representation. Further limitations include: lack of support for quality characteristics of context data, no reasoning over context information, and simplicity of the synthesis process, since it includes only data type conversions.

---

[26] More information can be found at <http://www.cc.gatech.edu/fce/contexttoolkit/>.

6.2.2 Solar

Solar (CHEN and KOTZ, 2002a; CHEN and KOTZ, 2002b; CHEN, 2004) is a middleware for the development of context-aware applications that collects data from heterogeneous sources, aggregates this data to obtain high-level context, and carries the dissemination of this context across network nodes. It has been developed at Dartmouth College, and the main objective of the project is to employ a graph-based abstraction, named operator graph, for context aggregation and dissemination.

The operator graph is an interconnected tree of operators to collect and aggregate desired context. Each operator is an object, which subscribes and processes one or more information sources and produces an event stream. These sources could be either physical or virtual sensors, while the output is a sequence of events produced by the operator. Typically, operators are filters, transformers, or more sophisticated aggregators, and, as in the case of sources, they can also be subscribed to (CHEN and KOTZ, 2002b).

In the project, a *Solar system* consists of a centralized *Star* and several interconnected *Planets*. The former processes subscription requests from applications and deploys operators into Planets, as needed, while the latter are execution platforms for Solar objects, such as operators (CHEN, 2004). Planets are the central Solar abstraction. They manage all subscriptions for each operator physically residing in them. Sources and applications run outside the Solar system. They access Solar services and functionalities via a small library, which allows sources to publish events, applications to send requests to Stars and manage their subscriptions, and to receive Solar events (CHEN, 2004).

There is a prototype implemented in Java, in which events are modeled as objects, and event transmissions as object serializations.[27] In the future, the authors intend to use XML or attribute-based representations for events, as most of them have a simple structure (CHEN and KOTZ, 2002a). Some applications have been developed using Solar. A SmartReminder exploits context information to remind users of appointments. A meeting detector is another developed application, which routes incoming phone calls to voice-mail when a user is in a meeting. Meetings are detected using sensors of pressure and motion effects on chairs.

Solar is mainly focused on context acquisition, transformation, and synthesis, allowing a flexible and extensible combination of operators to accomplish these processes. However, it uses an informal representation for context. In addition, Solar does not consider the quality of context. Another drawback is the fact that Solar does not deal with historical context data. Furthermore, context gathering requires direct involvement from the application, since it must manage sensors, operators, and requests.

6.2.3 Framework for Context-aware Pervasive Computing Applications

The Framework for Context-aware Pervasive Computing Applications (HENRICKSEN and INDULSKA, 2006; HENRICKSEN and INDULSKA, 2004;

---

[27] More information can be obtained at <http://www.cs.dartmouth.edu/~solar/>.

HENRICKSEN, 2003) has been developed at University of Queensland, as part of the PACE (Pervasive Autonomic Context-aware Environments) Project.[28] The proposal is a solution for supporting software engineering challenges in the development of context-aware software. The main objective is to provide a conceptual framework and a software infrastructure that together address context modeling techniques, a preference model for requirement representation, and two programming models.

In the framework, context is modeled using a graphical approach named Context Modeling Language (CML). This is an extension of the Object-Role Modeling (ORM) and was chosen due to its high expressiveness and formality; also, ORM can easily be mapped to relational databases (HENRICKSEN and INDULSKA, 2006). In CML, we can describe types of information (*facts*), classification (*sensed*, *static*, *profiled*, or *derived*), quality metadata, and *dependences* among different types (HENRICKSEN, 2003).

Context is then stored in a relation database, which can be further accessed throughout a query mechanism. Additionally, a *situation abstraction* is provided, based on predicate logic, to facilitate the implementation of context-aware applications. This abstraction is used to define context conditions in terms of facts, and they may also be combined to develop complex situations (HENRICKSEN and INDULSKA, 2004).

In order to support the adaptation process, a *preference model* is offered. The model is based on situation abstraction and represents each situation by a named pair, with a *scope* and a *scoring* expression (HENRICKSEN and INDULSKA, 2006). While the former describes the context to which the preference is related, the latter assigns a grade to the choice (numerical value ranging from zero to one). The framework also proposes two programming models: *branching*, an application logic which helps in the decision process related to context information; and *triggering*, an asynchronous mechanism that activates actions based on context changes (HENRICKSEN and INDULSKA, 2006).

A software infrastructure is proposed with six loosely coupled layers (HENRICKSEN, 2003): *context gathering*, in charge of acquiring, transforming, and synthesizing context data; *context reception*, responsible for providing a bi-directional mapping between gathering and upper layers; *context management*, to store the context models and their representation; *query*, used to provide an interface for querying context information; *adaptation*, employed in the management of situations, preferences, and trigger definitions, and also in the evaluation of those on behalf of the *application*, which constitutes the upper system layer.

There is a prototype of the software infrastructure implemented using Java and PostgreSQL. To validate it, a case study was carried out to evaluate the framework ability to support and develop context-aware software. The application created for this case study consists of a recommendation tool of communication channels. The tool suggests forms of contact with people, based on context and preferences.

The proposed framework provides a powerful software engineering methodology for context-aware software development. However, the offered context model hinders the representation of complex contexts. Besides, although in the proposal the authors

---

[28] More information about the project can be reached at <http://www.itee.uq.edu.au/~pace/>.

foresee an aggregation process, no method has been described so far. Furthermore, data filtering can only be applied when related to the same context element. Neither does the project provide a discovery mechanism, nor a way of reasoning over the relational database. Nevertheless, the contribution of the proposal in terms of context management, and more specifically, related to adaptation, is promising and innovative.

6.3 Comparison of Approaches

We compare the design principles of the context-aware systems described in the previously section with Continuum (Table 6.1). The evaluation was based on the multi-tiered model proposed in the beginning of the chapter. Each design principle is considered in the scope of the tier it is related to. Moreover, for each system a symbol indicates if the characteristics are fully-supported, partially-supported, not-supported, or not-applicable. For some issues, the comparison consists in defining the way in which some design principle is tackled in the system, rather than its support or availability.

Examining the comparison, we can observe that none of the systems considers networks of sensors. This is probably because the main focus is on individual sensors rather than on a group of them. Another aspect related to acquisition, is that Context Toolkit and Solar do not separate the gathering process from the use of context data. The lack of a loose coupling affects the tolerance of component failures and disconnections.

When assessing the transformation tier, we can perceive that different context models are used. Among all systems discussed, only Continuum employs an ontology, which is considered the most expressive model for context representation (BALDAUF et al., 2007). Besides, filters are only fully considered in Solar and Continuum. In the synthesis process, Context Toolkit and Solar do not consider quality issues. Furthermore, the aggregation process differs. We highlight Solar and Continuum, which provide a more flexible approach.

The discovery layer is also different, depending on the system. The Framework for Development of Context-aware Pervasive Computing Applications does not provide a discovery tier, while the others, with the exception of Continuum, only locate resources by their name. Analyzing the issues related to storage, we conclude that among the systems, which provide this layer, the employment of database is the method of choice. Besides, the tendency is to use a centralized solution, which is only avoided by Continuum, as a way of reducing scalability-related problems.

In the assessment of the query tier, we conclude that only Continuum provides reasoning capabilities, while all the proposals employ some query language for context obtaining. The support of inference and machine learning capabilities leverage the support of implicit context and proactive behaviors (HENRICKSEN and INDUSLKA, 2006). Related to security and privacy, each system has some solution to address these issues. However, we believe that no sufficient security and privacy is offered by any of the projects. Further studies have to be conducted in the field, in order to improve this design principle.

All considered projects have subscription and delivery capabilities. The only difference rests in the element that can be subscribed. Finally, when assessing the

upper-tier, we realized that all solutions employ an application-aware strategy. Nevertheless, Context Toolkit and Solar do not directly address adaptation, only questions related to context awareness.

Table 6.1: Context-aware systems comparison

Context-aware tier - Design principle	**Context Toolkit**	**Solar**	**Framework for Context-aware P.C. Applications**	**Continuum**
Sensor				
- Physical sensor	✓	✓	✓	✓
- Virtual sensor	✓	✓	✓	✓
- Logical sensor	✓	✓	✓	✓
- Network of sensors	×	×	×	×
Acquisition	✓	✓	✓	✓
- Separate from use	×	×	✓	✓
Transformation	✓	✓	✓	✓
- Context model	key-value	key-value	graphical	ontology
- Filtering capability	×	✓	●	✓
Synthesis	●	✓	●	✓
- Aggregation process	entity	various	facts	various
- Quality considering	×	×	✓	✓
Discovery	✓	✓	×	✓
- Leasing/pooling	✓	✓	×	✓
- White pages	✓	✓	×	✓
- Yellow pages	×	×	×	✓
Storage	✓	×	✓	✓
- Placement	centralized	-	centralized	hierarchical
- Method	database	-	database	database
Query/Inference	✓	✓	✓	✓
- Inference engine	×	×	×	✓
- Query language	✓	✓	✓	✓
- Security and Privacy	ownership	policies / authorization	policies	policies / anonymized data
Subscription	✓	✓	✓	✓
- Conditions	✓	✓	✓	✓
- Attributes	✓	✓	✓	✓
- Subscription element	widget	sensor / operator	situation	sensor / entity
Application				
- Adaptation strategy	application-aware	application-aware	application-aware	application-aware
- Adaptation support	×	×	✓	✓

Symbols: ✓ fully-supported ● partially-supported × not-supported - not-applicable

Many context-aware systems are still using ad hoc data structures to represent context, and do not fulfill various design principles associated with context awareness, such as the use of a discovery service and the application of reasoning techniques. In addition, neither of the context-aware systems presented constitute a complete solution for ubiquitous computing, addressing all the challenges imposed by this novel field.

7 THE ANALYSIS AND ASSESSMENT OF CONTINUUM

In this chapter, we present the developed prototypes and assess the main innovative propositions in the Continuum project. The method used for validation is based on experimental evaluation, i.e., we propose case studies to assess the basic ideas of the thesis. Some experimental environments were created and used as the base for the research. It is our aim to concentrate on the assessment of the main contributions of this work, namely the distributed service architecture, the ontology representation and storage, and the context awareness subsystem.

7.1 Methodology

The methodology used for the assessment of this thesis and the main proposed concepts is based on case study. The idea behind this approach is to choose a single instance, also called an event or a *case*, in which an in-depth and over-time examination is prepared (FLYVBJERG, 2006). The cases should be selective, focusing on the main issues that are important to the subject being analyzed; besides, the choice of the case to be studied must increase the extent of what can be learned, in the time interval available for the completion of the work (TELLIS, 1997).

As the first step in the methodology, TELLIS (1997) suggests that we define a case study protocol, which may include a chain of sections:

- An overview of the case study, including its objective, purpose, and unit of analysis (i.e., what the case is);
- The specific questions, i.e. the “how” and “why” inquiries, we must keep in mind and try to answer during the study;
- The sources of information, in our case, the experimental environment created to assess the proposition; and,
- The outline of the final report, i.e. the format in which the interpretation of the results is presented.

In this thesis, we followed the protocol proposed by TELLIS (1997), dividing each case study according to his proposal: objective, experimental environment, research questions, and experiments and analysis of results.

Regarding experimental environments, we employed some off-the-shelf tools, in order to accelerate the obtaining of results. Moreover, we chose to apply open standards and widely accepted protocols whenever possible. What is more, when deciding on

programming languages, we considered the ease of programming, productivity gains, and rapid prototyping.

We decided to employ this methodology in order to abridge the time in obtaining results for our research. Furthermore, besides being considered a scientific method, case studies are deemed as acceptable, in terms of perception of the facts involving the object of study. They also fulfill the three main ideas of the qualitative method: describing, understanding, and explaining (TELLIS, 1997).

7.2 Case Study 1: Distributed Service Architecture

In this case study, we discuss the proposition of CoDSA, the Continuum distributed service architecture, based on SOA and web services. To accomplish this goal, we have modeled and implemented some services (proposed in section 4.6.1) in the distributed execution subsystem, more specifically Executor and Service Manager. When doing these experiments, we followed the methodology proposed in the previous section.

7.2.1 Objective

The unit of analysis in this case study is the architecture of CoDSA using web services. Our purpose is to demonstrate the proposed ideas related to service and application modeling and the management of web services, supporting replication, migration, and deployment. The goals derived from these intentions are:

- An assessment of the architecture of distributed services based on SOA and web services;
- A demonstration of the possibility of implementing deployment, replication, and migration of services;
- An evaluation of the main drawbacks in using SOA and web services.

7.2.2 Research Questions

In this case study, we have tried to answer the following research questions:

- How can distributed services be implemented, using SOA and web services, and what are the most important new possibilities for ubicomp, created by the application of this concept?
- How could the replication, the migration, and the deployment of web services be implemented?
- What are the main limitations in the proposed architecture and how can we tackle these drawbacks?

Based on these questions, we have defined our experimental environment, conducted the development of a prototype, and carried out the analysis of results.

7.2.3 Experimental Environment

We have created a prototype to evaluate this case study. The services were first defined using WSDL. In doing so, heterogeneity was ensured and the language

employed for the implementation was not significant for interoperability purposes. We chose Python[29] as the language for implementing our services. Our choice was based on various characteristics of the language, such as: platform independence, open source license, and ease of programming. Perhaps the main motive to employ Python was the fact that it is considered one of the best languages for quick development and initial prototyping, since the code is usually shorter and faster to write, due to its high expressiveness and set of libraries.

The use of web services in Python is provided by various libraries[30], which support SOAP, WSDL, and other related protocols. Among those, we have chosen SOAPpy. Another library employed in our implementation was ElementTree[31]. This toolkit helps the management of XML files in Python, providing a container object to store hierarchical data structures in memory.

In addition to implementing the services, we have had to detail the CoApp format (as defined in 4.6.1.1). A CoApp was defined as an XML document with an XSD specification. Along with these files, a CoApp may also contain various others, such as bytecodes, images, and databases. Due to the need of bundling together many different files, a CoApp is encapsulated using the ZIP data compression and archival format.

As the communication protocol, we used the standard TCP/IP stack with HTTP in the transport layer. For discovery purposes, we employed the Multicast-DNS[32] (mDNS) protocol. By using this protocol, it is possible to apply standard Domain Name Service (DNS) management in small networks without the need of a DNS server. This protocol was utilized for CoNodes finding their associated CoBase.

7.2.4 Experiments and Analysis of Results

The complete description of these experiments is detailed in Kellermann (2008). Here, we briefly discuss their main steps and draw our main conclusions. Our first experiment started with the definition of the basic functional requirements of CoDSA, which essentially are the capacity of installation, replication, and migration of services. Currently, SOA does not support these requirements, so we have needed to complement the present features of web services. To accomplish that, we have decided to define an abstraction for representing the instantiation of mobile and distributed services.

We need this abstraction because the current web services standard, which is modeled using WSDL, does not cover all the requirements of a pluggable service in Continuum. To draw a parallel, consider the affirmation in Papazoglou (2008), which states that an XML scheme alone could not define a web service, requiring an additional standard, i.e. WSDL. The abstraction for representing applications in Continuum is called CoApp.

---

[29] More about Python can be accessed at <http://www.python.org/>.

[30] As can be seen in <http://pywebsvcs.sourceforge.net/>.

[31] ElementTree can be found at <http://effbot.org/zone/element-index.htm>.

[32] Multicast DNS information can be accessed at <http://www.multicastdns.org/>.

A CoApp is an application or a component in the form of a web service that can be installed, migrated, or copied. It represents an application in the Continuum infrastructure. The outer interface of a CoApp is always a web service for interoperability purposes. A CoApp is divided in three sections (KELLERMANN, 2008):

- CoApp Configuration (CAC): describes the application and its interface;
- CoApp Resources (CAR): details the specific application resources, such as images and internationalization definitions;
- CoApp Implementation (CAI): contains one or more implementations using a specific runtime.

In Figure 7.1 we present the simplified conceptual vision of a CoApp. The figure presents the CoApp container, divided in the three sections (CAC, CAR, and CAI). The lines at the lower part of the representation correspond to the services that the CoApp exports.

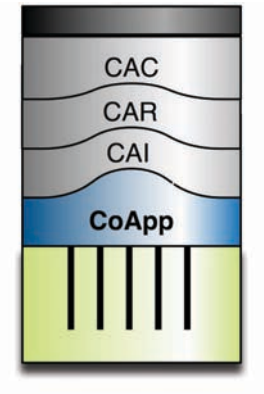

Figure 7.1: CoApp conceptual vision

A CoApp consists of a data format that represents a group of related files compressed and archived, in which there is a sufficient amount of information to infer the type of application, different versions, and requirements, such as the execution runtime needed, the location of resources, and the external service interface. It is our aim that the CoApp format be as simple and as small as possible.

An application implemented in any language along with its resources and exported interface (in WSDL format) is wrapped together with the XML definition of the CoApp. The CAC constitutes this XML definition; the CAR aggregates the resources; and, the CAI is the sum of all the implementation codes in any language. The transformation process of a preexisting application in a CoApp is straightforward and involves two steps. The first step is to describe the application, possibly with a WSDL interface, in a standard CoApp XML definition, containing the three sections described before. An XSD is also provided to validate the XML document created. This process is very simple, but we intend to automate it in the future. The second step is to create a single archive that has the binary codes, XML definitions, and resources, among others files. We have used the standard ZIP format in this process with base64 codification.

The next step was to create some CoApps using the proposed definition. Figure 7.2 shows a sample CoApp description of an application, named Calendar, implemented in

Python. In the CAC section, we define the elements *Name*, *Version*, *Author*, and *Description*. Additional elements are supported, such as *License* or *Copyright*. The element *Service* defines the service interface, using WSDL. The last element present in the CAC section of the sample CoApp is *Requirements.* Requirements of the "Runtime" type apply during execution. "Component", a generic type of requirement used for stating CoApp dependencies, is not shown in the example. In the CAR section, all the resources employed by the CoApp are listed. According to the runtime requirements fulfilled in the CAC section, the associated runtime in the CAI section is chosen. In the example, there are two possibilities in terms of runtime: Python and Lua. Details about the CoApp description and a group of other developed applications can be obtained in Kellerman (2008).

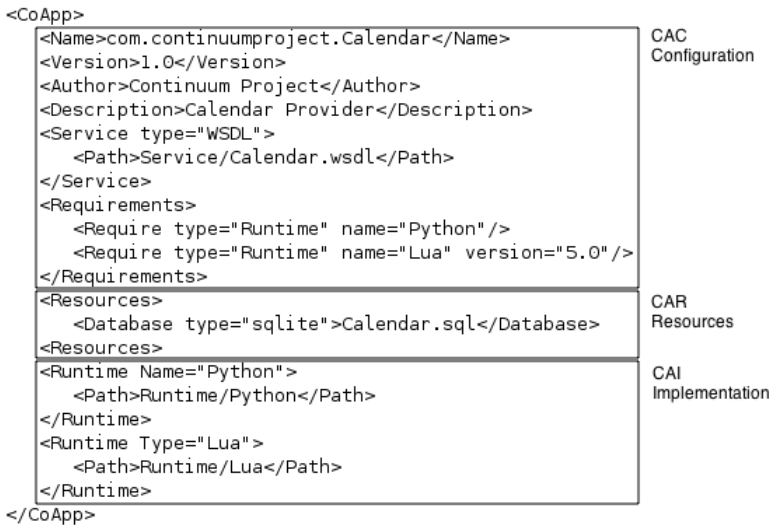

Figure 7.2: Calendar CoApp description

Since we are able to create CoApps, the next step would be to manage them in our distributed architecture. Therefore, we continued our experiment by implementing the Executor and the Service Manager. These services were implemented using the same interface proposed in sections 4.6.1.1 and 4.6.1.5, respectively. We described the WSDL in detail[33], with the design of various data types, and provided a python implementation.

To execute applications in Continuum, we start by installing them (using the *deployApplication* method of the Executor service), and then they can be managed (started, stopped, suspended, and so on) with the Executor interface. An application that has only been deployed is not considered as a Continuum pluggable service, but rather as a stand-alone application. To transform this application into a service, we must call the *registerService* method of the Service Manager. After that, we can treat a CoApp as a pluggable service, moving and replicating it as we wish.

Replication consists in copying one CoService from one CoNode to another in the Continuum infrastructure. The operations offered by this service continue to be

[33] The WSDL of Executor and Service Manager services can be reached at <http://www.continuumproject.com/wsdl>.

accessible in the origin, in addition to the new availability at the destination. In the developed prototype, this operation involves some steps. First of all, a CoConsumer asks for replication calling copyService. This CoConsumer can be any node, probably (but not necessarily) the node that contains the service being replicated. After that, Service Manager verifies if it has the CoApp content; if not, the service calls the Executor's *getApplication* method in the origin node, thus obtaining the desired content. The next step is to issue *deployApplication* in the Executor of the destination CoNode, sending the CoApp content to it. Subsequently, when the destination node accepts the CoApp, the Service Manager must be updated. This is accomplished by calling *registerService* in the CoDirectory that is associated with the destination CoCell. When a node searches for a specific service, it receives a *CoNodeLocation*, which contains the reference of both CoProviders (origin and destination).

Migration is very similar to replication, with a few additional steps. The difference in this operation is that we move the service from origin to destination. After the conclusion of this operation, the service is no longer available from the origin CoNode. If a node tries to access this service in its former location, it receives a message that indicates that the service is not present. As a consequence, it must call *lookupService* from the Service Manager to obtain the new address. To minimize the occurrence of this situation, whenever a service is migrated, the Service Manager notifies the new location to the CoConsumers that are currently using the moved service, sending its new CoServiceReference. To perform this notification, the Service Manager has to call the Executor's *serviceReference* method. Another additional step is the unregistering of the CoService in the CoDirectory associated with the origin CoCell, which is done by the *unregisterService* operation of the Service Manager.

Activities in web services tend to be coordinated, using standards such as BPEL (Business Process Execution Language), sometimes referred as web services orchestration in the literature (PAPAZOGLOU, 2008). This coordination reinforces the fact that services and components should be reutilized. As a consequence, the most complex operations proposed (replication and migration) are defined as low-level, in a coordinated approach. We believe that in this way, new features could be added to the model without any adaptation in the provided infrastructure.

Some cases (or events) were proposed as the next step of the experiment, using the developed prototype. In this thesis, we show four of those cases: application deployment, register of a CoApp as a CoService, replication of a CoService, and migration of a CoService.

In Figure 7.3 we present the deployment process of a CoApp in the infrastructure, containing one desktop and two mobile devices, highlighting four possible steps. The *deployApplication* method is employed, which generates the message *deployApplicationRequest* (step 1). This message contains a whole CoApp. As a return (step 2), the node produces a CoApp descriptor.

An alternative way of deployment is presented as the next steps of Figure 7.3. A CoNode calls the *getApplication* operation of the Executor service (step 3), using a *getApplicationRequest* message and passing a CoApp descriptor. The response is a *getApplicationResponse* message (step 4), which contains the application in its CoApp format. This received application is then installed in the local CoNode.

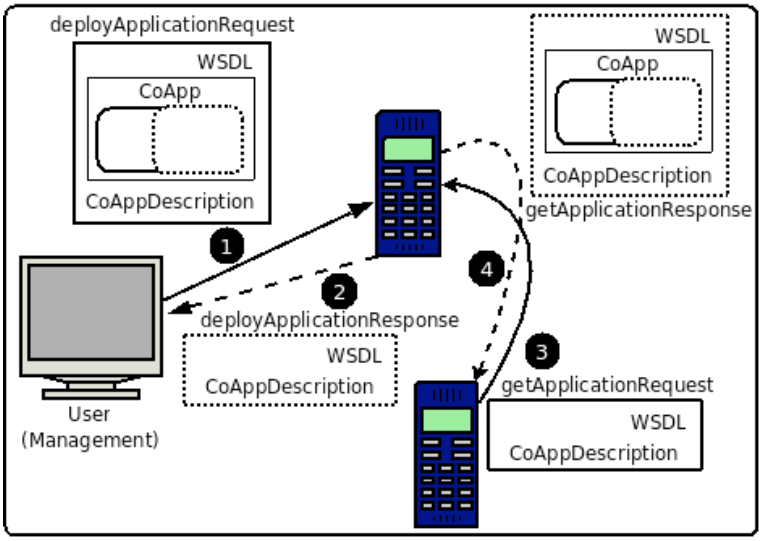

Figure 7.3: Deploying applications in Continuum

The second case, illustrating the transformation of a CoApp into a CoService, is presented in Figure 7.4. It considers the same infrastructure employed in the former event. Whenever a CoApp is registered as a service, its interface is disclosed, so that other CoNodes can make use of its operations as a pluggable service. The first call is to the *registerService* operation of the Service Manager, which uses the *registerApplicationRequest* message (step 1), containing a CoApp. As a return (step 2), we obtain a *CoServiceReference* composite message. Not only does this message contain the functional and semantic description of a service, but also its physical location (represented by *CoNodeLocation*).

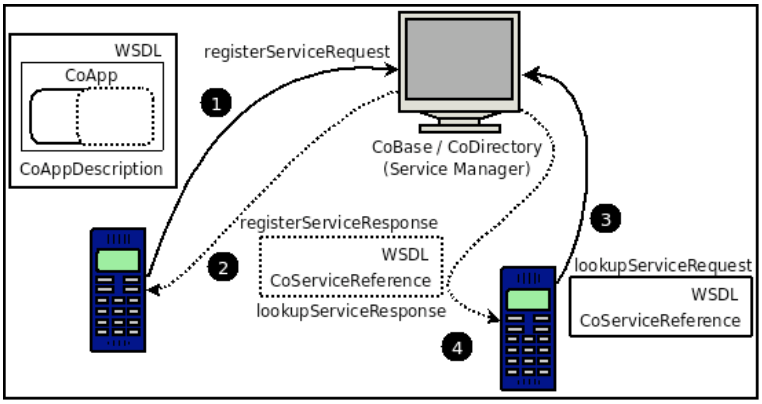

Figure 7.4: Applications becoming services in Continuum

In Figure 7.4 we also present the process of finding a service, accomplished by calling the *lookupService* method of the Service Manager. A *lookupServiceRequest* message is generated (step 3), containing *CoServiceReference.* As a result, a *lookupServiceResponse* message is produced (step 4), which also contains the same *CoServiceReference.* The difference between both messages is that in the first one only the location of the service is relevant whereas in the second a list of references could be possibly obtained.

Figure 7.5 presents the event of replicating a service. Differently from the previous cases, this diagram does not present the WSDL messages, but rather only some symbolic names. This is a simplification of the real process, which also involves additional steps. The process starts when a node calls *copyService* (step 1), passing a *CoServiceReference* (*myService* in the figure) and the *CoNodeLocation* to which the node will be copied (*destination* in the figure).

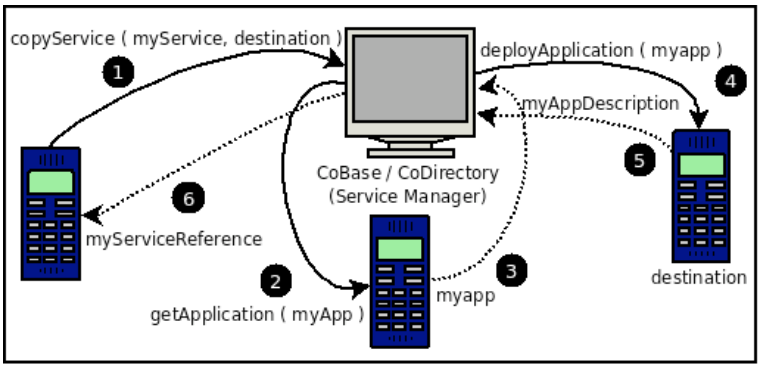

Figure 7.5: Service replication in Continuum

In the case developed, the CoDirectory did not have a copy of the CoApp that implements the desired service (*myService*). Therefore, the CoApp must obtain the application from its origin using the *getApplication* method (steps 2 and 3). If the CoApp is available in the CoDirectory, these steps could be omitted. After that, the service must be deployed in the destination CoNode (step 4). The destination nodes accept the installation of the CoApp as a trust relationship has been previously created between this CoNode and the CoDirectory (the aspects of security and trust are out of the scope of the present work). The Node returns a *deployApplicationResponse* message (step 5), which contains the descriptor of the CoApp. An additional step, omitted in the diagram, is the registering of the new location in the CoDirectory itself. The final step (6) is the message returned to the CoApp that started the operation, containing the reference of the newly instantiated service.

Finally, in Figure 7.6 we show the last case, which illustrates the service migration. This case has also been simplified and has similar steps (from 1 to 6). The only difference is the called method, which is *moveService* instead of *copyService*. The additional steps are related to the movement of the service from origin to destination (step 7) and the update of a CoNode that is using the service being migrated (step 8). This last step consists of updating the references, in nodes that are using the service, with those of the new location, so that the subsequent accesses employ the new address (illustrated in step 9).

From the cases developed, we have learned some lessons. The main advantage of the proposed model is that it does not create a completely new technology, but rather, while developing the distributed service architecture for Continuum, we are extending the current web services standards adding a new data abstraction and introducing a set of service orchestrations. By doing so, not only are we inheriting technological aspects of the SOC, but we are also inheriting an existing knowledge, terminology, and understanding on the field. Another important point is that we are reusing existing

applications with as little change as possible, making them available in the Continuum infrastructure.

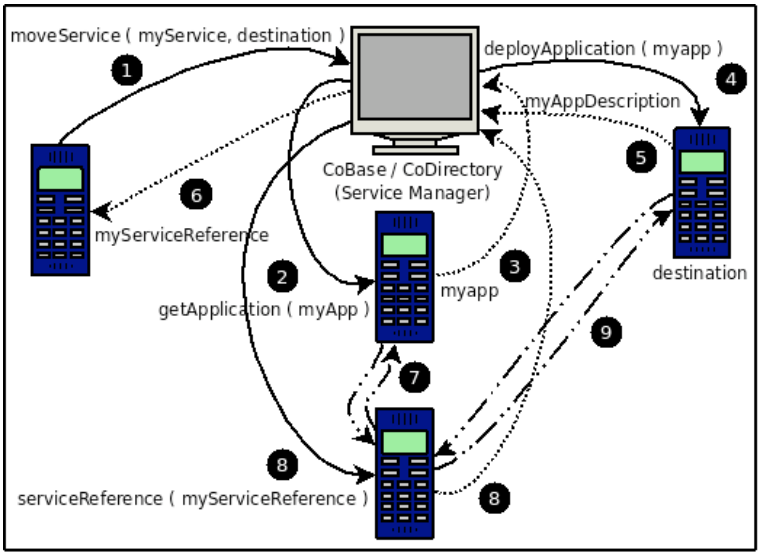

Figure 7.6: Service migration in Continuum

In the cases developed we also detected some limitations. First, all applications should have a CoApp description. Currently this is a manual process, which introduces some overhead in porting legacy code. Second, the execution of existent applications is only offered for software that uses runtimes, such as those developed for virtual machines. Programs compiled to a specific platform, are currently not supported. We tested or prototype with applications implemented in Python and in Java. We do not see this as a very strong limitation, since the use of virtual machines has been defended as a solution to improve heterogeneity in ubicomp (COSTA et al., 2008).

Another limitation concerning our current implementation is that we had limited the discovery and availability of services only to one CoDirectory. This occurs since we did not yet developed a mechanism for CoDirectory orchestration. Furthermore, the programmer is entirely responsible for all decisions regarding when and where to migrate (or to copy) a service. In the future, we can add a specific algorithm to help in this decision.

Although we observed some limitations, we believe that the use of SOA in ubicomp is a promising opportunity. It clearly fosters heterogeneity and integration among software implemented in different languages. Also, concerning our model, we believe that it helps the deployment and availability of code due to its pluggable feature. Another strength is related to the possibility of specifying alternative codes in the CoApp, possibly using distinctive runtimes. This feature adds a certain degree of adaptability, since code can be select according to the runtime available in the destination node.

We believe that the use of web services is an appropriate solution for ubicomp, as it improves the standardization of formats and protocols for describing services and their communication mechanisms. Moreover, we trust that the development of an architecture independent of runtime, platform or language incorporate advantages already obtained by SOA and web services standards. Among those advantages, we

highlight the interoperability among heterogeneous environments, the decoupling of the architecture from the hardware and low-level software infrastructure, and the independence from any type of proprietary technology, device or manufacturer.

7.3 Case Study 2: Ontology Representation and Inference

This section describes a case study related to the second part of the work, i.e. Continuum as a Context-aware system. We are particularly interested, in this analysis, in the formal representation of context and its capability of representing and inferring knowledge. As in the previous case study, we follow the methodology described in section 7.1.

7.3.1 Objective

The unit of analysis in this case study is the formal representation of context. Our purpose is to demonstrate the proposed ideas related to context representation, keeping historical context data, and supporting reasoning. The goals derived from those intentions are:

- An assessment of context representation using an ontology;
- Establishing the possibility of storing a history of context data with querying capabilities;
- An evaluation of the inference capability and possibility of obtaining context implicitly.

7.3.2 Research Questions

Based on the objectives proposed for this case study, we pose the following research questions (each one related to an objective previously presented):

1) How is context represented using an ontology, and what are the main strengths / weaknesses in representing context using this technique?
2) How is it possible to store historical context data, and what are the query capabilities?
3) What potential, in terms of knowledge, does the proposed context representation provide, and how can we obtain implicit context?

In the next sections we answer these questions, showing how it was achieved using the experimental environment created.

7.3.3 Experimental Environment

The main tool used in this experimental environment is Protégé[34]. This software constitutes an open-source platform for the development of knowledge-based frameworks and to edit ontologies. Protégé has been created at Stanford University,

[34] The tool can be downloaded at <http://protege.stanford.edu/>.

more specifically at the Stanford Center for Biomedical Informatics Research. It comprises a powerful tool for the representation, management, and visualization of ontologies, using the most common representation formats, including RDF, OWL, and XML Schema. Besides, we can also use the software for the definition of a KB, assigning values to the class properties, defining its facets, and creating the instances.

Besides the modeling of ontologies and knowledge bases, Protégé supports the specification of rules. These rules are codified using SWRL (Semantic Web Rule Language), a proposal for a Semantic Web rules-language, which combines OWL and the Rule Markup Language (RuleML). SWRL was submitted as a W3C proposal (HORROCKS et al., 2004), not yet homologated as we were writing this work. The rules in SWRL have the form of an implication between an antecedent and a consequent, in which the first is normally referred to as *body* and the last as *head*. The interpretation of an implication is *whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.* Both head and body have one or more *atoms*; if a body is omitted, it must be satisfied by every interpretation; if a head is empty, it should not be satisfied by every interpretation (HORROCKS et al., 2004).

To generate new knowledge, we use an inference machine. The execution occurs according to the description of Figure 7.7. In the Figure, the input consists of the rules and facts, while the generated output constitutes the new facts.

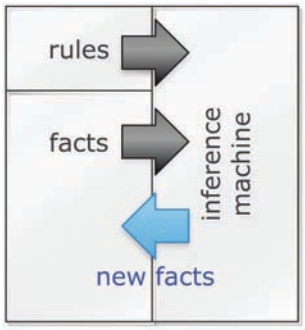

Figure 7.7: Execution of an inference machine

Besides inferring new knowledge, an inference machine can also be used for consistency verification, ensuring that an ontology does not have any contradictory facts, and classification, creating the complete class hierarchy by computing the relations between modeled classes, among other possibilities.

To execute inferences in Protégé, we can employ a rule engine, such as Jess[35]. In rule-based programming, we can apply a group of rules to the stored data, so that an algorithm combines both, i.e., so that it reasons over the knowledge, by applying a set of declarative rules. The algorithm chosen by Jess is Rete (FORGY, 82) and is commonly utilized in the development of expert systems.

Another tool for reasoning available in Protégé is Pellet[36], which is an open source OWL reasoner based on tableaux algorithms for expressive description logics (DL)

[35] Jess can be found at <http://www.jessrules.com/>.

[36] Pellet information can be reached at <http://pellet.owldl.com/>.

(SIRIN et al., 2007). Pellet supports OWL DL, which is an OWL sublanguage that provides the maximum expressivity possible, while sustaining computational completeness and decidability (SMITH et al., 2004).

The difference between Pellet and Jess is the distinctive approach for answering queries: while the former tries to satisfy queries as they are posed, the latter evolves the KB from an initial stage in order to satisfy queries at a smallest feasible response time (DELIAS, 2007). Another difference between both reasoners, according to DELIAS (2007), is that Pellet has minor initialization time but is slower during query executions. This is because whenever we change parameters in Pellet, the queries must be executed from scratch, although the sequential execution of the same query is executed instantaneously.

All queries in Protégé are carried out by SPARQL (SPARQL Protocol and RDF Query Language), a W3C recommendation (PRUD'HOMMEAUX and SEABORNE, 2008). SPARQL is a query language specifically aimed at RDF, which can be used to represent queries across various data sources. The main capability of this language is to query graph patterns, supporting value testing and constraining, and returning result sets or RDF graphs as the output.

7.3.4 Experiments and Analysis of Results

We started by modeling the ontology in Protégé. First, we represented the classes (described in section 5.2.1) and created the properties that define relations among classes (previously presented in Figure 5.2). Next, we defined the other intrinsic and extrinsic properties, such as data-types (which map to primitive types) and instance-types (which map to objects). Subsequently, we added facets, i.e. restrictions imposed to the slots. In this step, we defined the cardinality (the number of values a slot can have), the slot-value type (primitive or instance of classes), the slot domain (classes to which a slot is attached), and the range of a slot (allowed classes for instance slots). A complete description of these phases can be found in PESSUTO (2008a). To exemplify this modeling, Figure 7.8 shows the Protégé class editor window for the creation of a CoBase.

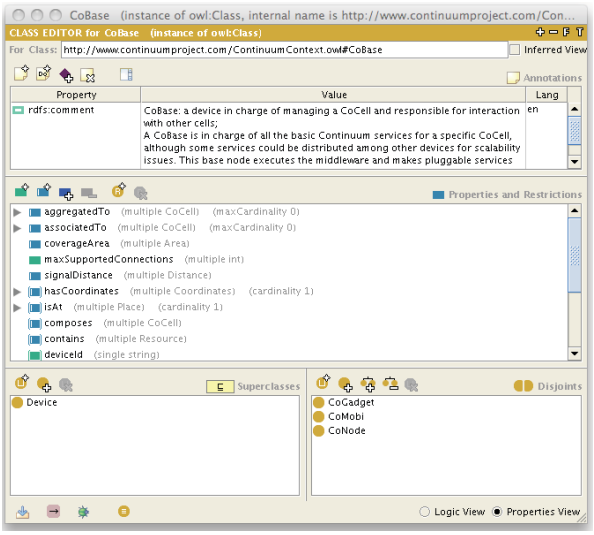

Figure 7.8: CoBase class modeling in Protégé

After modeling the ontology, we created a sample KB to accomplish this case study. We have chosen to represent information to characterize the sample scenario for the hypothetical applications *Digital Transport* and *Service Finder* (listed as samples scenarios in section 1.2.1). For this purpose, we modeled the CoDimension presented in Figure 7.9. This modeling consists of the definition of the experiment scenario, involving the selection of classes, the creation of individual instances, and the supplying of the slot values. As an example, we show in the Figure 7.10 the definition of a CoCell that represents the Library located in the University Campus.

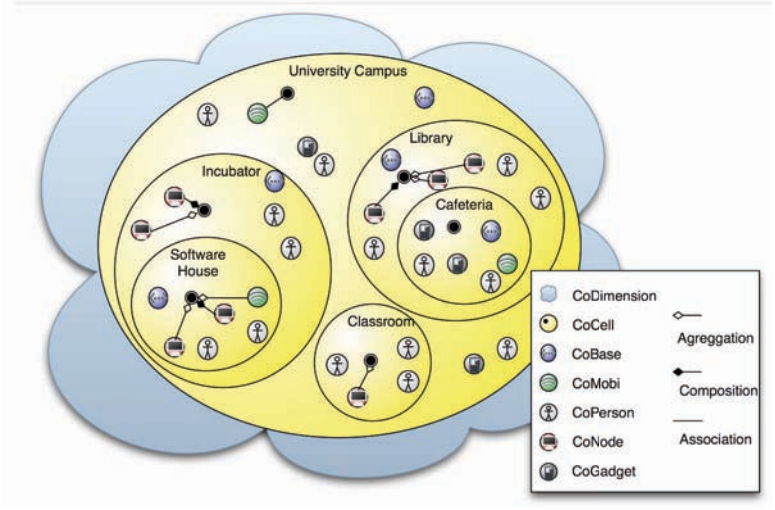

Figure 7.9: Sample scenario for the second experiment

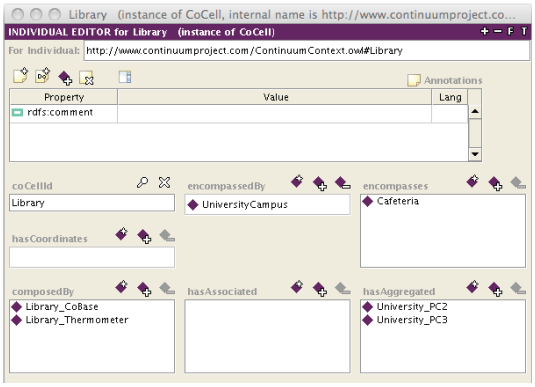

Figure 7.10: Instance modeling in Protégé

After all classes and instances had been represented, we used Pellet to perform two actions: check the consistency of the ontology and classify the taxonomy. The first one consists in ensuring that the ontology does not contain contradictory facts, while the second computes the relation between classes, generating a complete class hierarchy. As a consequence of this latter action, superclasses and equivalent classes of every modeled class were inferred.

Next, we performed some inferences in the KB. We created SWRL rules and, using Jess, we executed them. In Figure 7.11 we show a sample rule used in the experiments. In this rule, we defined that a CoCell offers the Services its CoBase provides.

```
CoCell(?cell) ∧ CoBase(?base) ∧ Service(?service) ∧
composes(?base, ?cell) ∧ provides(?base, ?service) →
offersService(?cell, ?service)
```

Figure 7.11: An SWRL rule to provide services to nodes

Depending on the rule, some inferred knowledge was produced, such as asserted individuals or asserted axioms. The new inferred knowledge was then added to the KB. Finally, using SPARQL, we performed some queries over the available data. We present one of the queries developed to illustrate this process (Figure 7.12). In this example, we try to find a CoCell that offerd an instance of a Service named *BusinessChat*.

```
PREFIX cco: <http://www.continuumproject.com/ContinuumContext.owl#>
SELECT ?cell
WHERE {
  ?cell cco:offersService cco:BusinessChat.
}
```

Figure 7.12: A SPARQL query to find available services

During the experiment, many use cases were developed in order to answer the research questions. Bellow, we summarize the main use cases (for a complete and detailed description refer to PESSUTTO, 2008a and PESSUTTO, 2008b):

1) Guessing the location of people based on the pinpoint of their mobile devices: we considered in this experiment devices that are normally carried by people, such as watches or cell phones; if we could infer the location of such devices, we could also guess the current position of their owners. We started by defining rules that associate devices to CoCells and that define the assumption that if some personal devices are at a specific place, their owners should be located in the same CoCell. After that, we create some queries to locate people based on their personal devices. The output is the name of the CoCell;

2) Finding the nearest CoBase from a GPS coordinate (Latitude, Longitude): each entity in the ontology is related to a set of coordinates (through the *hasCoordinates* property). We employed a simple algorithm to calculate the shortest distance between two points because, for each place, person, or thing, the *hasCoordinates* property is inherited. Unfortunately, it was not possible to carry on this solution using only rules, since SWRL does not support the necessary sine and cosine functions. The workaround was to use a programming language subroutine to calculate these;

3) Finding the available services and their location: the idea of this use case is to obtain a list of places, along with additional information, where a given

service is available. By combining this use case with the former, we can also infer the nearest service, in case of multiple availability;

4) Discovering how to get to a place: in this experiment, we want to obtain the path from one CoCell to another, in terms of the CoCells that one has to go through on the way to the final destination. In consequence, the return of the query should consist of the hierarchy of CoCells, i.e. the branches of the tree that should be crossed. Currently in SPARQL, there is no built-in support for querying hierarchical structures of an unknown depth. The only possible workaround is to repeat queries or to employ a fixed depth[37]. Such limitation can be surpassed by the use of a programming language and a well-known algorithm for the traversing of tree data structures;

5) Finding out how to reach some service: this use case is similar to the former one, the only difference being the distinctive (service instead of place) search. The solution is basically the same;

6) Obtaining information on where and when some event occurs or has taken place: in this experiment, we evaluate the use of historical data. The idea is a query that can answer the exact CoCell where a particular event occurred, or will occur, at a specific time. Although this use case is not particularly tied to the sample scenarios chosen, it is useful for the development of the former two experiments, which involve location. The result of this query is a specific CoCell and a timestamp. The latter piece of information is obtained in the form *Date*, *startTime*, and *endTime*. We can also obtain multiple results with the developed query;

7) Locating the past event that occurred at some place: this use case is similar to the previous presented experiment; the only difference is that the input is the CoCell and the output is the timestamp and the event data;

8) Finding where a person was before or during some event: in this experiment, we developed a query and some rules that help to pinpoint people before or during a specific event. For instance, in the sample scenario developed, it could help to find where the students are during a specific class, provided they are physically present at the University Campus. To conduct this experiment, a *CoPersonId*, the specific Event we are interested in, and the *startTime* are provided as input. The query returns all CoCells where a person had been located from *startTime* (passed as input) until the end of the event.

There are lessons to be learned from the results of the experiments. First, as expected, there is a significant increase in the expressiveness of the context information stored. Compared to our previous work, which used an ad hoc representation of context, based on key-value models, there is a huge difference. We can now express context information, which was not possible before, and in a simpler way. For instance, composition between places and past events had not been available in ISAM.

---

[37] As suggested in <http://thefigtrees.net/lee/sw/sparql-faq>.

In EXEHDA, context elements store the description of how context could be produced for that particular component and all possible context states (YAMIN, 2004). It was difficult to find a relation between different context elements in that middleware, since each element had a separate definition and their data could only be associated with the component itself. In our work, context can be associated with each and every class of the ontology. On top of that, the properties of classes can be employed to establish conditions not directly stored in the context database. For example, in one of the use cases developed (#1), we could guess a person's location by pinpointing some personal device.

Secondly, regarding the inference capabilities, we can deduce conditions and situations that are not explicitly represented in the stored information. Furthermore, very sophisticated queries can be posed, which increases the applicability of the information. To illustrate this conclusion, consider the last use case proposed (#8), in which we could discover where a person was during a particular event. The information was obtained merely by combining simpler context data, i.e. the pinpoint of people (or their personal devices), the location of a particular event, and timestamp data (time and date).

Thirdly, there is also the possibility of extending both the ontology to particular domains and the knowledge base. Considering the same scenario applied in the use cases, we could, for example, model classes specifically suited to the situation, such as course, class, professor, student, and so on. In fact, this is one of the chief characteristics of ontologies, i.e. proposing vocabulary for specific domains.

In addition, with the use of an ontology, it is easier to store historical context information. With the definition of a *Timestamp* class, data such as events and locations are stored with a corresponding *Date* and *Time*. Thus, we can infer historical-related information, such as past situations, as we demonstrate in the use cases 6, 7, and 8.

There are still some drawbacks in the use of this solution. SPARQL, the query language employed, is a recent proposition. It lacks optimization and widespread use. Besides, in one of the use cases proposed (#4), the lack of support for hierarchical queries prevented us from obtaining the expected result. Furthermore, SWRL, the language used for defining the rules, is still a proposition, and not a W3C recommendation. There is the omission of some typical rule language constructions, in favor of decidability. Additionally, we were unable to develop certain operations we needed in some use cases. For example, sine and cosine functions were necessary for distance determination (in use case #2).

Besides the weakness related to the immaturity of the technology, we could also foresee some other potential problems in our proposition. The quantity of stored data tends to increase, and large storage systems could be needed. Furthermore, with the increasing of the database size, there may also be an overhead at the time needed to find and relate context information. Regarding the inference capabilities, the more data we have stored, the wider the possibility of implicit context detection. However, the time needed to obtain this context also increases. We did not make a suitable analysis of the overhead of the time the reasoner takes to execute in our solution. But informally, we can affirm that for a relative small database, the time to execute an inference takes about a couple of hundred milliseconds, using current off-the-shelf and commercially available computers. Surely, this time could increase with larger databases and more

complex inferences. Nonetheless, we believe that with the constant improvement in the speed of processors, and in storage/memory capacities, this overhead can be lessened.

7.4 Case Study 3: Context Awareness Subsystem

In this last case study we are particularly interested in analyzing the possibilities of implementing some of the main features of the context awareness subsystem by employing current available tools and standards. We concentrated, in this particular case study, on the study of the technologies that could be used to implement the subsystem.

7.4.1 Objective

Our objective in this case study is to establish how to provide the main features of the context awareness subsystem, namely:

- An inquiry into how we can deal with the main features of the Aggregator service;
- The analysis of the possibility of storing context information in a database;
- An appraisal of the capability of the subsystem to find context data in different locations.

To attain these objectives, we have started by formulating our research questions in the next subsection.

7.4.2 Research Questions

Based on the objectives of this case study the following questions were formulated:

1) How can we combine context information from various sources and what means can we apply to take into account the users' preferences?
2) How can we store context information in a database and what are the main advantages and limitations involved?
3) What elements of the solution could provide distributed access and find the location of context data?

With these questions in mind, we were able to define the experimental environment.

7.4.3 Experimental Environment

We started this experiment with the same environment employed in the previous case study. However, we have considered other additional tools and protocols. The main tool discussed is Jena[38]. It consists of an open framework, created by Hewlett-Packard (HP), to the development of web semantics applications in Java (CARROLL et

[38] Jena can be obtained at <http://jena.sourceforge.net>.

al., 2004). Jena provides an API to handle data represented in RDF and OWL. The project includes a rule-based inference machine and an interface to other reasoners, such as Pellet. Another possibility offered by Jena is database integration, to provide persistence, or the use of in-memory storage. To query this data, we can utilize SPARQL queries. To do so, we have to use a Jena module named ARQ[39]. This module supports multiple query languages and engines.

Besides Jena, we have looked into the use of Java, since it is the only choice when using this framework. With Java, there is the possibility of employing all capabilities on hand, in terms of existing packages and libraries. Via JDBC (Java Database Connectivity), we can integrate our Jena developed programs with various database management systems (DBMS). Finally, we have studied the integration with an open source database, more specifically MySQL[40].

7.4.4 Experiments and Analysis of Results

We started this experiment by studying the technologies that could be employed to answer the posed questions. Due to time restrictions, we did not develop a prototype nor did we carry out a detailed implementation of this case study. Instead, we have concentrated on showing how we can achieve the desired features by applying the currently available standards and tools. A complete description of these experiments can be found in PESSUTO (2008b).

To answer the first question, we began by analyzing how we can combine context information from various sources. What we want to obtain is a way of aggregating the context associated with a particular entity, merging information from different sources. This aggregation can be obtained by using SWRL, with a rule to fulfill this need.

For instance, consider a specific situation in which we want to discover what events are associated with a place. To answer this question, we have to gather all the events of this specific place, and also of the composed ones. In order to do that, we need a rule stating, “if an event occurs in X, and Y encompasses X, then the event occurs in Y”. Figure 7.13 shows an SWRL rule that accomplishes this aggregation.

```
Place(?place1) ∧ Place(?place2) ∧ Event(?event) ∧
occursIn(?event, ?place1) ∧ encompasses(?place2,
?place1) → occursIn(?event, ?place2)
```

Figure 7.13: An SWRL rule to aggregate context information

We believe that in developing SWRL rules we can create conditions that allow complex aggregations of context, which is difficult to be obtained in other context models. Compared to the key-based model used in our previous project (ISAM), and also in projects such as Context Toolkit and Solar, the possibilities in terms of

[39] ARQ can be downloaded at <http://jena.sourceforge.net/ARQ/>.

[40] More about MySQL can be found at <http://www.mysql.com>.

aggregation of context here are wider and less restricted. The difficulty lies in developing the SWRL rules according to the aggregation needed. To lessen the programmer's effort, the infrastructure may provide a set of predefined rules and also a user interface that facilitates the aggregation of context data, automatically generating the rules.

Regarding the consideration of user preference, they can be employed during the aggregation process (as suggested in section 5.3.4), using the parameters specified there. We have to provide a way of storing these preferences associated with each Person in the ontology. This could also help answer one of the competency questions, "*Who is John?*", which was suggested after the definition of the Continuum ontology[41]. Although user preferences do not correspond to an exact description of who a person is, it can give us an approximate idea. Moreover, the ontology can be complemented with additional descriptions, such as physical characteristics, career, friends, and personal data (name, address, etc.).

As a solution for storing user preferences and other personal descriptions, we can employ the developed ontologies for social networks, such as the Friend-Of-A-Friend (FOAF) ontology (MIKA, 2004). FOAF proposes a formal representation for user profiles and friendship networks. In its vocabulary[42], we can describe people with their personal information, online accounts, projects, groups, documents, and images. The ontology was designed with further extension in mind, and additional preferences can be easily added to include the specific needs of the aggregator subsystem.

Considering the second research question, context information can be stored in a relational database by using Jena. Currently, it is possible to integrate this framework with MySQL, among other databases[43]. We can search the database using SPARQL queries, which are converted into standard SQL queries for the specific database engine. Jena's Fastpath Query Processing[44] performs this conversion.

Since the Continuum ontology was designed in Protégé, it is already defined as an RDF model stored in memory. We can then store this in a database, just by converting it through Jena's RDB ModelMakers. Once this database model is produced, we can connect by using a JDBC URL (Uniform Resource Locator), user, password, and database type. Figure 7.14 illustrates the main steps coded in Java, using Jena, to establish the connection with a MySQL database. Once connected, we can open an existing model with the last method shown (*openModel*).

The next step consists in accessing the database using SPARQL queries. For this, we have employed Jena's ARQ module. To illustrate this process, consider a sample question to obtain all entities, along with its respective classes, which are located in the CoCell that has the id "Library" (using the same sample scenario of the previous case study). The corresponding query using Jena and Java, is shown in Figure 7.15.

---

[41] The competency questions can be found in Appendix C.

[42] FOAF Vocabulary is described at <http://xmlns.com/foaf/spec/>.

[43] A complete list of compatible databases can be found at <http://jena.sourceforge.net/DB/index.html>.

[44] This feature is described at <http://jena.sourceforge.net/DB/fastpath.html>.

```
String M_DB_URL         = "jdbc:mysql://www.continuumproject.com/ContinuumContext";
String M_DB_USER        = "cac";
String M_DB_PASSWD      = "admin";
String M_DB             = "MySQL";  //database type
String M_DBDRIVER_CLASS = "com.mysql.jdbc.Driver";

Class.forName(M_DBDRIVER_CLASS);
IDBConnection conn = new DBConnection(M_DB_URL, M_DB_USER, M_DB_PASSWD,
                                      M_DB);        // create a database connection
Model prvModel = maker.openModel("ContinuumModel"); // open an existent model
```

Figure 7.14: Database connection using Jena

```
String queryString =
"PREFIX cco: <http://www.continuumproject.com/ContinuumContext.owl> " +
"SELECT ?entity ?entityClass " +
"WHERE {" +
"   ?entity cco:isAt ?coCell;" +
"           rdf:type ?entityClass." +
"   ?coCell cco:coCellId \"Library\"." +
"      }";

Query query = QueryFactory.create(queryString);
QueryExecution qe = QueryExecutionFactory.create(query, model);
ResultSet results = qe.execSelect();
```

Figure 7.15: Database query using ARQ

Ontologies are conceptually different from databases. While the latter is a data repository with queries that generally return the same data previously stored, the former consists of a data model, in which queries involve inferences, usually returning new knowledge, and considering rules, affecting the final result (IBM, 2008). Despite these differences, it is possible, as shown, to store an ontology in a database. To achieve this goal, we have used a Java interface (Jena) that addresses these issues.

Storing knowledge bases using DBMS is an approach currently employed by many proposals. This solution has the advantage of inheriting the vast experience of research and database use, including improvements in robustness, concurrency control, scalability, and recovery. However, because of the complexity of typical ontology queries, DBMS cannot perform optimally for this kind of application (LEE and GOODWIN, 2005). There are a number of improvements that should be investigated for the improvement of ontology storage in databases. Lee and Goodwin (2005) point out various directions for further work that should contribute to this advance, towards what has been named an Ontology Data Management System (ODMS).

Regarding the use of SPARQL, one limitation of this recommendation is that the focus is on querying. Therefore, it provides only four query forms (PRUD'HOMMEAUX and SEABORNE, 2008): *select*, to return data that matches a specific pattern; *construct*, to obtain the result of a query as an RDF graph, combining various queries solutions, based on a graph template; *ask*, to find whether a query pattern matches or not a specific condition (*Boolean* response); and *describe*, to restore an RDF graph that describes the resources found.

However, SPARQL does not specify an update language for RDF graphs (one that supports *insert*, *update*, and *delete*). Fortunately, researchers from HP have proposed a SPARQL-based language for updating RDF graphs[45]. The support of this language is

[45] This specification is available at <http://jena.hpl.hp.com/~afs/SPARQL-Update.html>.

available through ARQ. Besides, ARQ supports other extensions, which are still not provided by SPARQL, such as the clause *group by* and the use of expressions in a *select* clause.

The distributed access and location of context data is provided by one of the features on hand in the standard SPARQL specification, named *prefix*. This feature allows the definition of prefixes associated with Uniform Resource Identifiers (URIs). Consequently, we can use several distributed data sources in a simple way: we can just define each data source as a different prefix in the query (Figure 7.16 shows an example of trying to discover where and when an event, *WednesdayHappyHour*, takes place). The use of URIs as a universal identifier for the reference of entities and relationships is one of the benefits of the RDF data model. SPARQL inherits this feature, since it uses this model.

```
PREFIX cco: <http://www.continuumproject.com/ContinuumContext.owl#>
PREFIX owlt: <http://www.isi.edu/~pan/damltime/time-entry.owl#>
SELECT ?event ?place ?startTime ?endTime
WHERE {
   ?event rdf:type cco:Event;
     cco:EventId "WednesdayHappyHour";
     cco:takesPlaceAt ?place;
     owlt:begins ?startTimeD;
     owlt:ends ?endTimeD.
  ?startTimeD  owlt:inCalendarClockDataType ?startTime.
  ?endTimeD   owlt:inCalendarClockDataType ?endTime
}
```

Figure 7.16: SPARQL query with mutiple data sources

Currently, a prototype of the context awareness subsystem is under development employing all the solutions studied in this experiment.

8 CONCLUSION AND FUTURE WORK

It is still difficult to find software infrastructure covering all the necessary characteristics for ubiquitous computing. In the past, projects such as Aura, Gaia, One.World, and ISAM tried to accomplish many aspects of ubicomp. However, it is hard to concentrate on many different open research topics in one project. Many projects nowadays provide solutions for specific issues. In spite of this tendency, we think that a general solution to the field can help the development of pervasive software.

In this work, we have presented Continuum, a software infrastructure employing middleware and framework in ubicomp. To achieve this goal, we started by surveying the field and identifying the main challenges of the area. Subsequently, we proposed a comprehensive architecture to cover all the characteristics that should be addressed to accomplish these issues.

Continuum was proposed based on the requirements established in this comprehensive model. Figure 8.1 characterizes each service proposed for Continuum middleware and its correspondent position in the general architecture proposed. As observed, all characteristics planned for load time and runtime were covered in our proposition. Furthermore, related to design time, general considerations for the development of a Continuum framework were provided.

In the detailing of Continuum, we described an architecture for service support based on web services and SOA. According to Baldauf et al. (2007), the use of web services is an appropriate solution for context-aware systems, since it improves the standardization of formats and protocols for describing services and their communication mechanisms. Using this technology, we detailed various services organized into five subsystems. The proposition of Continuum infrastructure, considered as a horizontal thread in our thesis, has formed the necessary basis to propose the context-aware support for the system.

Context awareness constitutes the specific focus of our work, referred to as a vertical thread. We defined a context model, specifying an ontology to characterize context information, based on the model proposed for representation of the real world in Continuum. A methodology for storing context information, along with the means to distribute and place the information, have also been pointed out. Some services were then described to make use of this context.

Next, the state of the art in context-aware systems was presented, and a multi-tiered model was proposed. This model encompasses all the services that should be available in a context-aware system. We also evaluated some systems and compared them with Continuum. As an additional conclusion, we believe that our proposal is the most

general and the most complete, dealing with all the layers presented in the multi-tiered model, as seen in Figure 8.2.

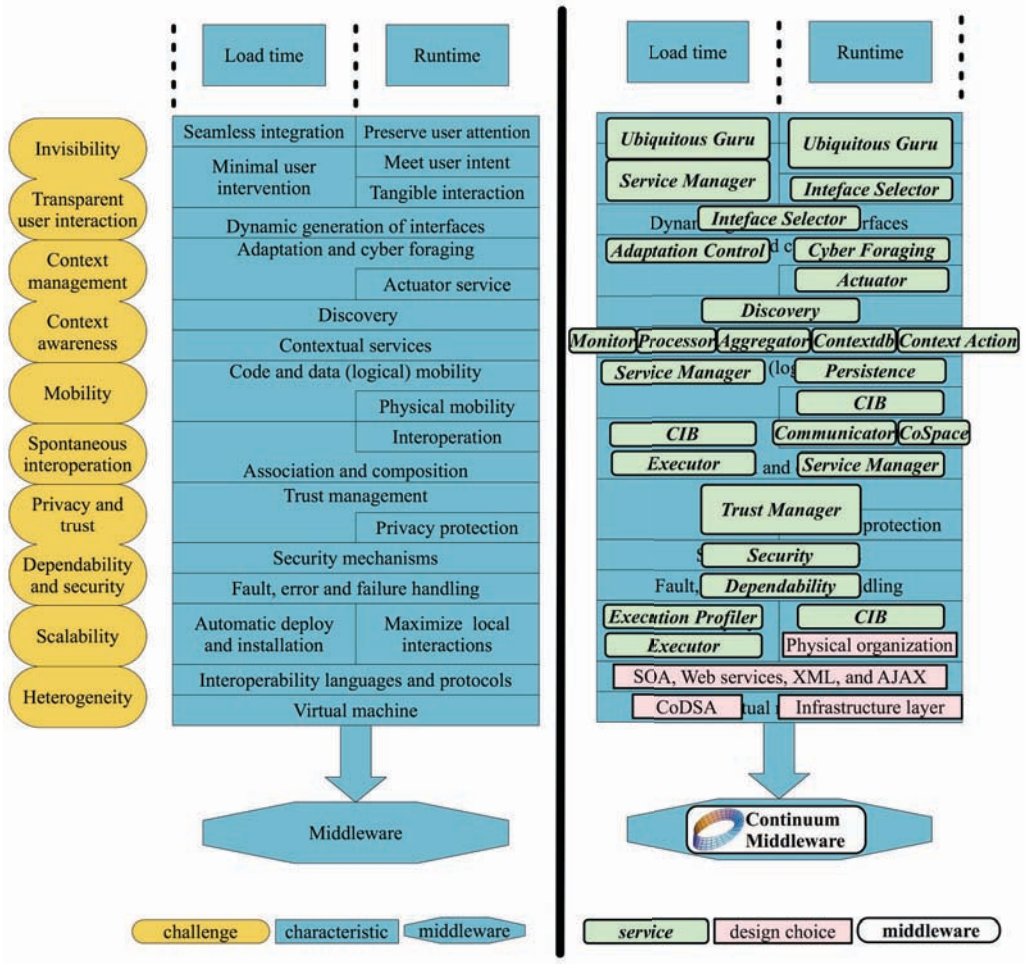

Figure 8.1: Relationships between comprehensive architecture and Continuum

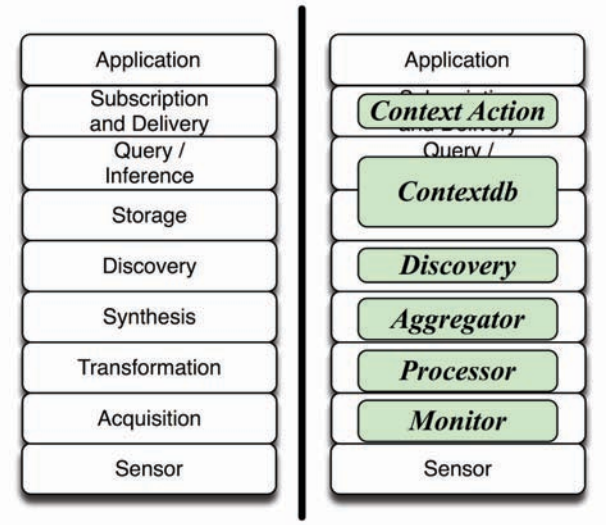

Figure 8.2: Relationships between multi-tiered context-aware model and Continuum

Finally, some experimental evaluations were conducted to analyze and assess Continuum. We have proposed three general experiments, based on case study methodology. In the first experiment, we discussed the proposition of the distributed service architecture. We demonstrate that its implementation is possible and present the main strengths and limitations that it could have.

The second case study regards the representation of context as an ontology and inference capability. As a result, we have listed some lessons learned. We conclude that, although the technology is still quite recent and immature, its use is promising and reinforces the utilization of web semantics for context representation.

The last experiment has shown how we can implement the context awareness subsystem employing current available tools and standards. Our conclusion is that it is possible to develop our context awareness subsystem employing only currently existing technology, although some improvements are desired to ensure better performance and a wider ranging use.

Briefly, the main contributions of the thesis are:

- A revision of the field of ubiquitous computing, comprehending its origins, evolution, and main challenges;
- A proposal of a general architecture model for ubiquitous computing, helping the community to develop and assess middleware and frameworks for this area. Another aim of this architecture is to highlight the requirements of a software infrastructure in the field;
- The proposition of the software infrastructure Continuum, as an evolution of the ISAM project;
- A model, along with a notation, for representing the real world in Continuum;
- The proposal of an architecture for service support, employing the use of SOA and web services;
- The suggestion of an infrastructure to improve the support of mobile devices in Continuum, using Web 2.0 concepts;
- The modeling of twenty services for a software infrastructure in ubiquitous computing, divided into four subsystems. Thirteen of those services are totally new, while seven are based on previous works, such as EXEHDA, ISAMadapt, ACTUS, and other graduate research projects;
- The suggestion of some general consideration for development of the Continuum framework;
- The design of a Context Awareness subsystem, encompassing the formalization of context representation; the use of resource discovery; the proposal of a database for historical context; the distribution and placement of context information;
- A proposition of a multi-tiered model for context-aware systems, emphasizing the main services;

- The assessment of the context awareness solutions proposed and a comparison between them and the state of the art in the area;
- The analysis and evaluation of the Continuum's main contributions. The method employed for validation was based on case studies, with three series of experiments;
- The development of a case study to discuss the proposition of the distributed service architecture. In this experiment, two services were modeled and implemented along with the conduction of some tests;
- The description of a case study to analyze the formal representation of context, along with its storage and inference capacity. This experiment was conducted by employing a set of tools and by defining a sample scenario;
- The last case study conducted, had the objective of examining what tools and standards could be employed in the context awareness subsystem.

As an additional contribution, we had some articles published in journals and proceedings during the development of this proposal:

- **IEEE PERVASIVE COMPUTING**. COSTA, C.; YAMIN, A.; GEYER, C. Towards a General Software Infrastructure for Ubiquitous Computing. **IEEE Pervasive Computing**, Los Alamitos, v.7, n.1, p. 64-73, Jan. 2008.
- **ACM SAC / SIGAPP 2008**. SILVA, L.; COSTA, C.; GEYER, C.; AUGUSTIN, I.; YAMIN, A. On the control of Adaptation in Ubiquitous Computing. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, SIGAPP, 23., Mar. 2008, Fortaleza. **Proceedings…** New York: ACM, 2008. p. 2228-2229.
- **WPUC 2007**. COSTA, C.; SILVA, L.; YAMIN, A.; GEYER, C. Uma Proposta de Infra-estrutura de Software de Segunda Geração. In: I Workshop on Pervasive and Ubiquitous Computing, WPUC, 2007, Gramado. Workshop on Pervasive and Ubiquitous Computing. **Anais…** 2007.
- **ERAD 2007**. COSTA, C.; GEYER, C. Uma Proposta de Arquitetura de Software para a Computacao Ubíqua. In: Escola Regional de Alto Desempenho, ERAD 2007, 2007, Porto Alegre. **Anais…** 2007. p. 85-86.
- **WPPD 2007**. COSTA, C.; SILVA, L.; YAMIN, A.; GEYER, C. . A Preliminary Outline for a Ubiquitous Computing Software Infrastructure. In: V Workshop de Processamento Paralelo e Distribuído, WPPD'2007, 2007, Porto Alegre. **Anais…** 2007.
- **WPPD 2007**. SILVA, L.; COSTA, C.; GEYER, C. ACTUS: a Framework for Adaptation Control in Ubiquitous Computing. In: V Workshop de Processamento Paralelo e Distribuído, WPPD'2007, 2007, Porto Alegre. **Anais…** 2007.
- **WPPD 2006**. COSTA, C.; GEYER, C. Um Modelo Genérico de Infra-estrutura de Software para a Computação Ubíqua. In: IV Workshop de Processamento Paralelo e Distribuído da UFRGS, WPPD'2006, 2006, Porto Alegre. **Anais…** 2006.

Continuum presents several opportunities for extensibility and future work. In the proposition presented for modeling the real world in the infrastructure, we can address the possibility of CoBase replication and specialization. In addition, various middleware services can be implemented, and some should be further described, such as Trust Manager and the aspect of tangible interaction in the Interface Selector service. The task-aware feature of Ubiquitous Guru must also be more deeply investigated. Moreover, the framework can be detailed and validated.

Related to the context-aware architecture there are also many aspects that can be improved. We can deal with networks of sensors, as suggested by the final context-aware system assessment. Additionally, the improvement of privacy and trust in the context-awareness architecture is also a possibility. Finally, there could be an in-depth investigation on specific kinds of context, such as emotional states.

We trust that the current work has helped bridge the distance between Weiser's vision of ubiquitous computing and the current distributed system scenario. This goal was accomplished by the proposition of Continuum context-aware service-based software infrastructure. We believe that the results presented in this thesis reinforce this objective.

Ubiquitous computing is considered today a *hot topic* in computer science. There are many articles, events, publications, funding, and on-going works related to the field. Although this seems to be an opportunity for research and development, it also establishes many constrains, and the need for wide-range investigations, levering the amount of work undertaken during the process of developing this thesis.

Still, the proposal of Continuum has been gratifying work, and we hope to give a long-lasting contribution to the area. We trust that this proposal could also be useful to the advance of ubiquitous software development. What is more, we consider that to fulfill Weiser's vision, future ubiquitous infrastructure should, as Continuum proposes, seamlessly integrate many different challenges.

REFERENCES

ABOWD, G.; MYNATT, E.; RODDEN, T. The human experience. **IEEE Pervasive Computing**, Los Alamitos, v.1, n.1, p. 48-57, Jan. 2002.

ADELSTEIN, F. et al. **Fundamentals of Mobile and Pervasive Computing**. New York: McGraw-Hill, 2005.

AGOSTINI, A.; BETTINI, C.; RIBONI, D. Experience Report: Ontological Reasoning for Context-aware Internet Services. In: INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS WORKSHOPS, PERCOMM, 4., 2006, Pisa. **Proceedings…** New York: IEEE, 2006.

AILISTO, H. et al. Structuring Context Aware Applications: five-layer model and example case. In: WORKSHOP ON CONCEPTS AND MODELS FOR UBIQUITOUS COMPUTING, 2002, Götenborg. **Proceedings…** Available at: < http://www.comp.lancs.ac.uk/~dixa/conf/ubicomp2002-models/papers-list.html>. Visited on: Apr. 2008.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A Survey of Peer-to-Peer Content Distribution Technologies. **ACM Computing Surveys**, New York, v.36, n.4, p. 335-371, Dec. 2004.

ANEROUSIS, N.; MOHINDRA, A. The Software-as-a-Service Model for Mobile and Ubiquitous Computing Environments. In: INTERNATIONAL CONFERENCE ON MOBILE AND UBIQUITOUS SYSTEMS: NETWORKING & SERVICES, MOBIQUITOUS, 3., 2006, San Jose. **Proceedings…** New York: IEEE, 2006. p. 1-6.

ARNSTEIN, L. et al. Labscape: a smart environment for the cell biology laboratory. **IEEE Pervasive Computing**, Los Alamitos, v.1, n.3, p. 13-21, July 2002.

AUGUSTIN, I. **Abstrações para uma Linguagem de Programação visando Aplicações Móveis Conscientes do Contexto em um Ambiente de Pervasive Computing**. 2004. 193f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

AUGUSTIN, I. et al. ISAM, Joining Context-Awareness and Mobility to Building Pervasive Applications. In: ILYAS, M.; MAHGOUB, I. (Ed.) **Mobile Computing Handbook**. Boca Raton: CRC, 2004. p. 73-94.

AUGUSTIN, I. et al. Towards Taxonomy for Mobile Applications with Adaptive Behavior. In: INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED COMPUTING AND NETWORKING, PDCN, 20., 2002, Innsbruck. **Proceedings…** Innsbruck: IASTED, 2002.

AVIŽIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, Los Alamitos, v.1, n.1, p. 11-33, Jan. 2004.

AVIŽIENIS, A. Infrastructure-Based Design of Fault-Tolerant Systems: how to get high-confidence computing for all. In: IFIP INTERNATIONAL WORKSHOP ON DEPENDABLE COMPUTING AND ITS APPLICATIONS, DCIA, 1998, Johannesburg. **Proceedings…** [S.l.:s.n.], 1998.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A survey on context-aware systems. **International Journal of Ad Hoc and Ubiquitous Computing**, Geneve, v.2, n.4, p. 263-277, Oct. 2007.

BANAVAR, G. et al. Challenges: an application model for pervasive computing. In: INTERNATIONAL CONFERENCE ON MOBILE COMPUTING AND NETWORKING, MOBICOM, 6., 2000, Boston. **Proceedings…** New York: ACM, 2000. p. 266-274.

BANAVAR, G.; BERNSTEIN, A. Software infrastructure and design challenges for ubiquitous computing applications. **IEEE Pervasive Computing**, New York, v.1, n.1, p. 92-96, Jan. 2002.

BARBOSA, J. L. V. et al. GHolo: A Multiparadigm Model Oriented to Development of Grid Systems. **Future Generation Computer Systems**, Amsterdam, v.21, n.1, p. 227-237, Jan. 2005.

BARBOSA, J. L. V. et al. Multiparadigm Model Oriented to Development of Grid Systems. In: INTERNATIONAL WORKSHOP ON PROGRAMMING PARADIGMS FOR GRIDS AND METACOMPUTING SYSTEMS, PPGaMS, 1., 2004, Krákow. **Proceedings…** New York: Springer-Verlag, 2004. p. 2-9.

BARBOSA, J. L. V. **GRANLOG**: Um Modelo Para Análise Automática de Granulosidade na Programação em Lógica. 1996. 167f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

BARBOSA, J. L. V. **Holoparadigma**: Um Modelo Multiparadigma Orientado ao Desenvolvimento de Software Distribuído. 2002. 213f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

BARCELLOS, A. M. P.; BELMONTE, V.; GEYER, C. The HetNOS Network Operating Systems: a tool for writing distributed applications. **ACM Operating Systems Review**, New York, v. 28, n. 4, p. 34-47, Oct. 1994.

BARDRAM, J. The Trouble with Login: on usability and computer society security in ubiquitous computing. **Personal and Ubiquitous Computing**, London, v.9, n.6, p. 357-367, Nov. 2005.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. **Scientific American**, New York, v. 284, n. 5, p. 34-43, May 2001.

BERNSTEIN P. Middleware: a model for distributed system services. **Communications of the ACM**, New York, v.39, n.2, p. 86-98, Feb. 1996.

BISCHOFS, L. et al. A Hierarchical Super Peer Network for Distributed Artifacts. In: DIGITAL LIBRARY ARCHITECTURES WORKSHOP, DELOS, 2004, Cagliari. **Proceedings…** Padova: Edizioni Libreria Progetto, 2004. p. 105-114.

CAHILL, V. et al. Using Trust for Secure Collaboration in Uncertain Environments. **IEEE Pervasive Computing**, Los Alamitos, v.2, n.3, p. 52-61, July 2003.

CANNY, J. The Future of Human-Computer Interaction. **ACM Queue**, New York, v.4, n.6, p. 24-32, July 2006.

CARROLL, J. et al. Jena: implementing the semantic web recommendations. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE, WWW, 13., 2004. **Proceedings…** New York: ACM, 2004. p. 74-83.

CAS, J. Privacy in pervasive computing environments – a contradiction in terms? **IEEE Technology and Society Magazine**, Princeton, v.24, n.1, p. 24-33, Mar. 2005.

CHEN, G. **Solar:** Building a Context Fusion Network for Pervasive Computing. 2004. 169f. Thesis (Doctor of Philosophy in Computer Science) – Department of Computer Science, Darthmouth College, Hanover.

CHEN, G.; KOTZ, D. **A Survey of Context-aware Mobile Computing Research**. 2000. 16 f. Technical Report – Department of Computer Science, Darthmouth College, Hanover.

CHEN, G.; KOTZ, D. Context Aggregation and Dissemination in Ubiquitous Computing Systems. In: IEEE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, WMCSA, 4., 2002, Callicoon. **Proceedings…** New York:IEEE, 2002a. p. 105-114.

CHEN, G.; KOTZ, D. Solar: an open platform for context-aware mobile applications. In: INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING, Pervasive, 1., 2002, Zurich. **Proceedings…** Heidelberg: Springer-Verlag, 2002b. p. 41-47.

CHEN, H. et al. SOUPA: standard ontology for ubiquitous and pervasive applications. In: ANNUAL INTERNATIONAL CONFERENCE ON MOBILE AND UBIQUITOUS SYSTEMS: NETWORKING AND SERVICES, MOBIQUITOUS, 1., 2004, Boston. **Proceedings…** [S.l.:s.n.]. p. 258-267.

CHETAN, S.; RANGANATHAN, A.; CAMPBELL, R. Towards Fault tolerant Pervasive Computing. **IEEE Technology and Society Magazine**, Princeton, v.24, n.1, p.38-44, 2005.

CHINNICI, R. et al. (Ed). **Web Services Description Language (WSDL) Version 2.0** Part 1: core language. W3C Recommendation. [S.l.]: W3C, 2007. Available at: <http://www.w3.org/TR/wsdl20/>. Visited on: Dec. 2007.

CHONG, C.; KUMAR, S. Sensor Networks: evolution, opportunities, and challenges. **Proceedings of the IEEE**, Piscataway, v. 91, n. 8, p.1247-1256, Aug. 2003.

COSTA, A. C. R.; DIMURO, G. P. Interactive Computation: Stepping Stone in the Pathway From Classical to Developmental Computation. In: WORKSHOP ON THE FOUNDATIONS OF INTERACTIVE COMPUTATION, FInCo, 2005, Edinburgh. **Proceedings…** Edinburgh: LFCS/University of Edinburgh, 2005. v. 1, p. 1-12.

COSTA, C. A. da. GEYER, C. F. R. Uma proposta de escalonamento distribuído para exploração do paralelismo na programação em lógica. In: SIMPÓSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES E PROCESSAMENTO DE ALTO DESEMPENHO, SBAC-PAD, 10., 1998, Búzios. **Anais...** Rio de Janeiro: UFRJ: SBC, 1998. p. 61-64.

COSTA, C. A. da. **Uma Proposta de Escalonamento Distribuído para a Exploração do Paralelismo na Programação em Lógica**. 1998. 104f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

COSTA, C. A. da; YAMIN, A. C.; GEYER, C. F. R. Towards a General Software Infrastructure for Ubiquitous Computing. **IEEE Pervasive Computing**, Los Alamitos, v.7, n.1, p. 64-73, Jan. 2008.

COSTA, C. A. da; YAMIN, A. C.; GEYER, C. F. R. Uma Proposta de Infra-estrutura de Software de Segunda Geração. In: WORKSHOP ON PERVASIVE AND UBIQUITOUS COMPUTING, WPUC, 1., 2007, Gramado. **Anais...** Porto Alegre: UFRGS: SBC, 2007.

COSTA, V. dos S. et al. **CloPN - Sistemas Escaláveis de Alto Desempenho para Programação em Lógica com Restrições**. 1999. Proposta de Colaboração CNPq-NSF Protem-CC-CNPq. Disponível em: <http://www.cos.ufrj.br/~vitor/clopn/>. Acesso em: set. de 2007.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed Systems**: concepts and design. Harlow: Addison Wesley, 2005. 927p.

DELIAS, N. et al. A Performance Comparison of Ontology Reasoning and Rule Engines. In: MOBILE AND WIRELESS COMMUNICATIONS SUMMIT, IST, 16., 2007, Budapest. **Proceedings…** [S.l.:s.n.], 2007. p. 1-5.

DEY, A. **Providing Architectural Support for Building Context-Aware Applications**. 2000. 170f. Thesis (Doctor of Philosophy in Computer Science) – College of Computing, Georgia Institute of Technology, Atlanta.

DEY, A. Understanding and Using Context. **Personal and Ubiquitous Computing**, London, v.5, n.1, p. 4-7, Feb. 2001.

DEY, A.; ADOWD, G.; SALBER, D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Application. **Human-Computer Interactions Journal**, [S.l.], v. 16, n. 2-4, p. 97-166, 2001.

DHESIASEELAN, A. **What's New in WSDL 2.0**. [S.l.]: O'Reilly Media, 2004. Available at: <http://webservices.xml.com/lpt/a/ws/2004/05/19/wsdl2.html>. Visited on: Dec. 2007.

DOU, D.; LEPENDU, P. Ontology-based Integration for Relational Databases. In: ACM SYMPOSIUM ON APPLIED COMPUTING, SAC, 21., 2006, Dijon. **Proceedings…** New York: ACM, 2006. p. 461-466.

DOURISH, P. et al. Security in the Wild: user strategies for managing security as an everyday, practical problem. **Personal and Ubiquitous Computing**, London, v.8, n.6, p. 391-401, Nov. 2004.

EDWARDS, W. Discovery Systems in Ubiquitous Computing. **IEEE Pervasive Computing**, Los Alamitos, v.5, n.2, p. 70-77, Apr. 2006.

ELRAD, T.; FILMAN, R; BADER, A. Aspect-Oriented Programming. **Communications of the ACM**, New York,v.44, n.10, p. 28-32, Oct. 2001.

FEHLBERG, F. **MultiS:** um servidor de contexto voltado à computação pervasiva. 2007. 91f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

FETZER, C.; HÖGSTEDT, K. Challenges in Making Pervasive Systems Dependable. In: SCHIPER, A. et al. (Ed.) **Future Directions in Distributed Computing**. Berlin: Springer-Verlag, 2002. p.186-190.

FLYVBJERG, B. Five Misunderstandings About Case-Study. **Qualitative Inquiry**, Thousand Oaks , v.12, n.2, p. 219-245, Apr. 2006.

FORGY, C. Rete: a fast algorithm for the many pattern/many object pattern match problem. **Artificial Intelligence**, Amsterdam, v.19, n. 1, p. 17-37, Sept. 1982.

FOWLER, M. **UML Essencial**: um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2005. 160p.

FRAINER, G. **Espaço Pervasivo de Arquivos**: habilitando acesso adaptativo e consciente da aplicação a arquivos em um ambiente pervasivo. 2008. 86 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

FRAINER, G. et al. Utilizando adaptação consciente de aplicação no acesso a arquivos em um ambiente pervasivo. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS

DE ALTO DESEMPENHO, 8., 2007, Gramado. **Proceedings...** Porto Alegre: Sociedade Brasileira de Computação, 2007. p. 103-110.

FRIDAY, A. Supporting services discovery, querying and interaction in ubiquitous computing environments. **Wireless Networks**, Hingham, v.10, n.6, p. 631-641, Nov. 2004.

FUGGETA, A.; PICCO, G. P.; VIGNA, G. Understanding code mobility. **IEEE Transactions on Software Engineering**, Los Alamitos, v.24, n.5, p. 342-361, May 1998.

GARCÉS-ERICE, L. et al. Hierarchical Peer-to-peer Systems. In: ACM/IFIP INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING, Euro-Par, 2003, Klagenfurt. **Proceedings…** Netherlands: Springer, 2003. p. 1230-1239.

GARLAN, D. et al. Project Aura: Toward Distraction-free Pervasive Computing. **IEEE Pervasive Computing**, Los Alamitos, v.1, n.3, p. 22-31,Sept. 2002.

GÄRTNER, F. C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. **ACM Computing Surveys**, New York, v.31, n.1, p. 1-26, Mar. 1999.

GEYER, C. F. R. et al. APPELO – Project Parallel Environment for Logic Programming. In: PROJECTS EVALUATION WORKSHOP, 1999, Rio de Janeiro. **Proceedings…** Rio de Janeiro: ProTem – CC/CNpq, 1999. p. 421-464.

GRIMM, R. et al. System support for pervasive applications. **ACM Transactions on Computer Systems**, New York, v.22, n.4, p. 421-486, Nov. 2004.

GRIMM, R. One.world: experiences with a pervasive computing architecture. **IEEE Pervasive Computing**, Los Alamitos, v.3, n.3, p. 22-30, July 2004.

GRIMM, R. et al. Systems Directions for Pervasive Computing. In: WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS, HotOS, 8., 2001, Elmau. **Proceedings…** [S.l.:s.n.], 2001. p. 147-151.

GRUBER, T. **Ontology**. Disponível em: <http://tomgruber.org/writing/ontology-definition-2007.htm >. Acesso em: nov. 2007.

GU, T.; PUNG, H.; ZHANG, D. Toward an OSGi-Based Infrastructure for Context-Aware Applications. **IEEE Pervasive Computing**, Los Alamitos, v.3, n.4, p. 66-74, Oct. 2004.

GU, T.; PUNG, H.; ZHANG, Q. A service-oriented middleware for building context-aware services. **Journal of Network and Computer Applications**, Amsterdam, v. 28, n. 1, p. 1-18. Jan. 2005.

HAO, W. et al. An Infrastructure for Web Services Migration for Real-Time Applications. In: INTERNATIONAL WORKSHOP ON SERVICE-ORIENTED

SYSTEM ENGINEERING, SOSE, 2., 2006, Shanghai. **Proceedings…** New York: IEEE. 2006. p.41-48.

HASELOFF, S. **Context Awareness in Information Logistics**. 2005. 247f. Thesis (Doktorin der Ingenieurwissenschaften) – Elektrotechnik und Informatik, Technischen Universität Berlin, Berlin.

HEIDEGGER, M. **Being and time**: a translation of Sein and Zeit. New York: State University of New York, 1996.

HENRICKSEN, K. **A Framework for Context-aware Pervasive Applications**. 2003. 201f. Thesis (Doctor of Philosophy in Computer Science) – School of Information Technology and Electrical Engineering, University of Queensland, Brisbane.

HENRICKSEN, K.; INDUSLKA, J. A Software Engineering Framework for Context-aware Pervasive Computing. In: IEEE ANNUAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS, PERCOM, 2., 2004, Orlando. **Proceedings…** Los Alamitos: IEEE Computer Society. 2004. p. 77-86.

HENRICKSEN, K.; INDUSLKA, J. Developing Context-aware Pervasive Computing Applications: models and approach. **Pervasive and Mobile Computing**, Amsterdam, v.2, n.2, p.37-64, Feb. 2006.

HOBBS, J.; PUSTEJOVSKY, J. Annotating and Reasoning about Time and Events. In: AAAI SPRING SYMPOSIUM ON LOGICAL FORMALIZATIONS OF COMMONSENSE REASONING, 2003, Stanford. **Proceedings…** Menlo Park: AAAI, 2003.

HOFFNAGLE, G. F. Preface. **IBM System Journal**, Danvers, v.38, n.4 , p. 502, 1999.

HOLMQUIST, L.; SCHMIDT, A.; ULLMER, B. Tangible Interfaces in perspective. **Personal and Ubiquitous Computing**, New York, v.8, n.5, p. 291-293, May 2004.

HORNECKER, E. A design theme for tangible interaction: embodied facilitation. In: EUROPEAN CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, ECSCW, 9., 2005, Paris. **Proceedings…** Netherlands: Springer, 2005. p. 23-43.

HORROCKS, I. et al. **SWRL**: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission. [S.l.]: W3C, 2004. Available at: <http://www.w3.org/ Submission/SWRL/>. Visited on: July 2008.

HOWERTON, J. Service-Oriented Architecture and Web 2.0. **IT Professional**, Los Alamitos, v.9, n.3, p. 62-64, May 2007.

INDULSKA, J.; SUTTON, P. Location Management in Pervasive Systems. In: WORKSHOP OF WERABLE, INVISIBLE, CONTEXT-AWARE, AMBIENT, PERVASIVE AND UBIQUITOUS COMPUTING, WICAPUC, 1., 2003, Adelaide. **Proceedings…** Sydney: Australian Computer Society, 2003. p. 143-152.

JING, J.; HELAL, A.; ELMAGARMID, A. Client-server Computing in Mobile Environments. **ACM Computing Surveys**, New York, v.31, n.2, p. 117-157, June 1999.

JUSZCZYK, L.; LAZOWSKI, J.; DUSTDA, S. Web Service Discovery, Replication, and Synchronization in Ad-Hoc Networks. In: INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY, ARES, 1., 2006. **Proceedings…** New York: IEEE, 2006. p. 847-854.

KAGAL, L.; FININ, T.; JOSHI, A. A Policy Language for A Pervasive Computing Environment. In: INTERNATIONAL WORKSHOP ON POLICIES FOR DISTRIBUTED SYSTEMS AND NETWORKS, 4., 2003. **Proceedings…** New York: IEEE. 2003.

KAPOR, M. Recollections on Lotus 1-2-3: benchmark for spreadsheet software. **IEEE Annals of the History of Computing**, Los Alamitos, v.29, n.3, p. 32 -40, July 2007.

KATO, H.; TAN, K. Pervasive 2D Barcodes for Camera Phone Applications. **IEEE Pervasive Computing**, Los Alamitos, v.29, n.4, p. 76-85, Oct. 2007.

KEEN, E. **A primer in phenomenological psychology**. New York: Holt, Rinehart and Winston, 1975.

KELLERMANN, F. **Uma Proposta de Arquitetura de Serviços Distribuídos utilizando Web Services na Infra-Estrutura de Software Continuum**. 2008. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação), UNISINOS, São Leopoldo.

KINDBERG, T.; FOX, A. A system software for ubiquitous computing. **IEEE Pervasive Computing**, Los Alamitos, v.1, n.1, p. 70-81, Jan. 2002.

LEE, Y.; LEUNG, K.; SATYNARAYANAN, M. Operation-based update propagation in a mobile file system. In: USENIX ANNUAL TECHNICAL CONFERENCE, USENIX, 1999, Monterey. **Proceedings…** Berkeley: USENIX Association, 1999. p. 43-56.

LEVY, E.; SILBERSCHATZ, A. Distributed File Systems: concepts and examples. **ACM Computing Surveys**, New York, v.22, n.4, p. 321-374, Dec. 1990.

LIN, K. Building Web 2.0. **Computer**, Los Alamitos, v.40, n.5, p. 101-102, May 2007.

LOKE, S. Context-aware Artifacts: two development approaches. **IEEE Pervasive Computing**, Los Alamitos, v.5, n.2, p. 48-53, Apr. 2005.

LOPES, J. **EXEHDA-ON**: uma abordagem baseada em ontologias para sensibilidade ao contexto na computação pervasive. 2008. 128 f. Dissertação (Mestrado em Ciência da Computação) – Escola de Informática, UCPEL, Pelotas.

LUBYTE, L. Reusing Relational Sources for Semantic Information Access. In: CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, PIKM, 2007, Lisboa, **Proceedings…** New York: ACM, 2007. p. 9-16.

LYYTINEN K.; YOO, Y. Issues and Challenges in Ubiquitous Computing. **IEEE Pervasive Computing**, Los Alamitos, v.1, n.1, p. 61-65, Jan. 2002.

MCGUINNESS, D.; HARMELEN, F. (Ed). **OWL Web Ontology Language Overview**. W3C Recommendation. W3C, 2004. Available at: <http://www.w3.org/TR/owl-features/>. Visited on: Jan. 2008.

MIKA, P. Social Networks and the Semantic Web. In: INTERNATIONAL CONFERENCE ON WEB INTELLIGENCE, 2004. **Proceedings…** Washington: IEEE, 2004. p. 285-291.

MODAHL, M. UbiqStack: a taxonomy for a ubiquitous computing software stack. **Personal and Ubiquitous Computing**, London, v.10, n.1, p. 21-27, Jan. 2006.

MORAES, M. C. **DIMI:** um Disseminador Multicast de Informações para a Arquitetura ISAM. 2005. 91f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MOREL, E. et al. Side-effects in PloSys or-parallel Prolog on distributed memory machines. In: COMPULOG NET MEETING ON PARALLELISM AND IMPLEMENTATION TECHNOLOGY, 1996. **Proceedings...** [S.l.:s.n.], 1996.

MOSER, L.; MELLIAR-SMITH, P.; ZHAO, W. Making Web Services Dependable. In: INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY, ARES, 1., 2006. **Proceedings…** New York: IEEE. 2006. p. 440-448.

MURPHY, A.; PICCO, G.; ROMAN, G. LIME: a coordination model and middleware supporting mobility of hosts and agents. **ACM Transactions on Software Engineering and Methodology**, New York, v.15, n.3, p. 279-328, July 2006.

NIEMELÄ, E.; LATVAKOSKI, J. Survey of Requirements and Solutions for Ubiquitous Software. In: MOBILE UBIQUITOUS COMPUTING CONFERENCE, 2004, Washington. **Proceedings…** New York: ACM, 2004. p. 71-78.

NOBLE, B. System support for mobile, adaptive applications. **IEEE Personal Communications**, Los Alamitos, v.7, n.1, p. 44-4, Feb. 2000.

NOY, N.; MCGUINNESS, D. **Ontology Development 101**: a guide to creating your first ontology. 2001. 25 f. Technical Report – Knowledge Systems Laboratory, Stanford University, Stanford.

NYLANDER, S.; BYLUND, M.;WAERN, A. Ubiquitous Service access through adapted user interfaces on multiple devices. **Personal and Ubiquitous Computing**, London, v.9, n.3, p. 123-133, May 2005.

O'REILLY, T. **What is Web 2.0**, 2005. Available at: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>. Visited on: Aug. 2007.

PACE, J. A. D.; CAMPO, M. R. Analyzing the role of aspects in software design. **Communications of the ACM**, New York, v.44, n.10, p. 66-73, Oct. 2001.

PAPAZOGLOU M. P.; GEORGAKOPOULOS, D. Introduction: Service-oriented computing. **Communications of the ACM**, New York, v.46, n.10, p. 24-28, Oct. 2003.

PARK, H.; LEE, J. A framework of context-awareness for ubiquitous computing middlewares. In: ANNUAL ACIS INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION SCIENCE, ICIS, 4., 2005, Jeju Island. **Proceedings…** Washington: IEEE Computer Society, 2005. p. 369-374.

PESSUTTO, O. D. **Semantic Web in the Continuum's Context-Aware Architecture**. 2008b. 92f. Memoir de Projet de Recherche (Master Informatique Spécialité Web Intelligence) – École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble (ENSIMAG), Institut Polytechnique de Grenoble (INPG), Grenoble.

PESSUTTO, O. D. **Web Semântica na Arquitetura de Consciência de Contexto do Continuum**. 2008a. 89f. Trabalho de Graduação (Graduação em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

PIAGET, J. **Biology and Knowledge**: an essay on the relations between organic regulations and cognitive processes. Chicago: The University of Chicago, 1971.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. W3C Recommendation. [S.l.]: W3C, 2008. Available at: <http://www.w3.org/TR/rdf-sparql-query/>. Visited on: July 2008.

RANGANATHAN, A.; CAMPBELL, R. Reasoning about Uncertain Contexts in Pervasive Computing Environments. **IEEE Pervasive Computing**, Los Alamitos, v.3, n.2, p. 62-70, Apr. 2004.

ROBINSON, P.; VOGT, H.; WAGEALLA, W. Some Research Challenges in Pervasive Computing. In: ROBINSON, P.; VOGT, H.; WAGEALLA, W. (Ed.) **Privacy, Security and Trust within the Context of Pervasive Computing**. Boston: Springer Science + Business Media, 2005. p. 1-16.

ROGERS, Y. Moving on from Weiser's Vision of Calm Computing: engaging ubicomp experiences. In: INTERNATIONAL CONFERENCE ON UBIQUITOUS COMPUTING, UbiComp, 8., 2006, Orange County. **Proceedings…** Heidelberg: Springer-Verlag, 2006. p. 404-421.

ROMÁN, M. et al. A Middleware Infrastructure to Enable Active Spaces. **IEEE Pervasive Computing**, Los Alamitos, v.1, n.4, p. 74-83, Dec. 2002.

RÓMAN, M.; CAMPBELL, R. H. Gaia: enabling active spaces. In: SIGOPS EUROPEAN WORKSHOP, 9., 2000, Kolding. **Proceedings…** Kolding: ACM, 2000. p. 229-234.

ROSS, P.; KEYSON, D. The case of sculpting atmospheres: towards design principles for expressive tangible interaction in control of ambient systems. **Personal and Ubiquitous Computing**, London, v.11, n.2, p. 69-79, Feb. 2007.

SAHA, D.; MUKHERJEE, A. Pervasive Computing: a paradigm for the 21st century. **Computer**, Los Alamitos, v.36, n.3, p. 25-31, Mar. 2003.

SAITO, Y.; SHAPIRO, M. Optimistic replication. **ACM Computing Surveys**, New York, v.37, n.1, p. 42-81, Mar. 2005.

SALBER, D.; DEY, A.; ADOWD, G. The Context Toolkit: aiding the development of context-enabled applications. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, SIGCHI, 17., 1999, Pittsburgh. **Proceedings…** New York: ACM, 1999. p. 4340441.

SATYANARAYANAN, M. Fundamental Challenges in Mobile Computing. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, PODC, 15., 1996, Philadelphia. **Proceedings…** New York: ACM, 1996. p. 1-7.

SATYANARAYANAN, M. Pervasive computing: vision and challenges. **IEEE Personal Communications**, Los Alamitos, v.8, n.4, p. 10-17, Aug. 2001.

SCHAEFFER FILHO, A. G. **PerDis:** um serviço para descoberta de recursos no ISAM Pervasive Environment. 2005. 103f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SCHAFFER FILHO, A. G. et al. Applying the ISAM Architecture for Genetic Alignment in a Grid Environment. In: WORKSHOP DE GRADE COMPUTACIONAL E APLICAÇÕES, WGCA, Rio de Janeiro. **Proceedings…** [S.l.:s.n.], 2005.

SCHROTH, C. Web 2.0 versus SOA: converging concepts enabling seamless cross-organizational collaboration. In: INTERNATIONAL CONFERENCE ON E-COMMERCE TECHNOLOGY, 9., 2007. **Proceedings…** New York: IEEE, 2007.

SCHROTH, C.; JANNER, T. Web 2.0 and SOA: converging concepts enabling the Internet of Services. **IT Pro**, Los Alamitos, v.9, n.3, p. 36-41, May 2007.

SEIGNEUR, J. Fostering sustainability via trust engines. **IEEE Technology and Society Magazine**, Princeton, v.24, n.1, p. 34-37, Mar. 2005.

SIEWIOREK, D. New frontiers of application design. **IEEE Pervasive Computing**, Los Alamitos, v.1, n.1, p. 79-82, Jan. 2002.

SILVA, L. C. **ACTUS**: a Framework for Adaptation Control in Ubiquitous Computing. 2008. Proposta de Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SILVA, L. C. **Primitivas para Suporte à Distribuição de Objetos Direcionadas à Pervasive Computing**. 2003. 123f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SILVA, L. C.; COSTA, C. A.; GEYER, C.; AUGUSTIN, I.; YAMIN, A. On the control of Adaptation in Ubiquitous Computing. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, SIGAPP, 23., 2008, Fortaleza. **Proceedings…** New York: ACM, 2008. p. 2228-2229.

SIRIN, E. et al. Pellet: A Practical OWL-DL Reasoner. **Journal of Web Semantics**, Amsterdam, v.5, n.2, p. 51-53, June 2007.

SMITH, M; WELTY, C; MCGUINNESS, D. (Ed). **OWL Web Ontology Language Guide**. W3C Recommendation. [S.l.]: W3C, 2004. Available at: < http://www.w3.org/TR/ 2004/REC-owl-guide-20040210>. Visited on: July 2008.

SOUSA, J. P. et al. Task-based adaptation for ubiquitous computing. **IEEE Transactions on Systems, Man, and Cybernetics, Part C**, New York, v.36, n.3, p. 328-340, May 2006.

STRANG, T; LINNHOFF-POPIEN, C. A Context Modeling Survey. In: INTERNATIONAL WORKSHOP ON ADVANCED CONTEXT MODELLING, REASONING AND MANAGEMENT, UbiComp, 1., 2004, Tokyo. **Proceedings…** Netherlands: Springer. 2004. p. 34-41.

STRINGER, M. et al. The Webkit Tangible User Interface: a case study of iterative prototype. **IEEE Pervasive Computing**, Los Alamitos, v.4, n.4, p. 35-41, Oct. 2007.

SUGUMARAN, V.; STOREY, V. The role of domain ontologies in database design: an ontology management and conceptual modeling environment. **ACM Transactions on Database Systems**, New York, v. 31, n. 3, p. 1064-1094, Sept. 2006.

SURIE, A. et al. Rapid trust establishment for pervasive personal computing. **IEEE Pervasive Computing**, Los Alamitos, v.6, n.4, p. 24-30, Oct. 2007.

TELLIS, W. Introduction to Case Study. **The Qualitative Report**, Fort Lauderdale, v.3, n. 2, p. 3-15, July 1997.

VANTHOURNOUT, K.; DECONINCK, G.; BELMANS, R. A taxonomy for resource discovery. **Personal and Ubiquitous Computing**, London, v.9, n.2, p. 81-89, Mar. 2005.

WANT, R. et al. Disappearing hardware. **IEEE Pervasive Computing**, Los Alamitos, v.1, n.1, p. 36-47, Jan. 2002.

WANT, R.; PERING, T. System Challenges for Ubiquitous & Pervasive Computing. In: INTERNATIONAL CONFERENCE OF SOTWARE ENGINEERING, ICSE, 27., 2005, St. Louis. **Proceedings…** New York: ACM. 2005. p. 9-14.

WEISER, M. Some computer science issues in ubiquitous computing. **Communications of the ACM**, New York, v.36, n.7, p. 75-84, July 1993.

WEISER, M. The Computer for the 21st Century. **Scientific American**, New York, v.265, n.3, p. 94-104, Mar. 1991.

WEISER, M. The world is not a desktop. **ACM Interactions**, New York v.1, n.1, p. 7-8, Jan. 1994.

WERNER, O. **Uma máquina abstrata estendida para o paralelismo E na Programação em Lógica**. 1994. 145f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

XU, Z.; ZHANG, S.; DONG, Y. Mapping between Relational Database Schema and OWL Ontology for Deep Annotation. In: IEEE/WIC/ACM INTERNATIONAL CONFERENCE ON WEB INTELLIGENCE, WI, 2006, Hong Kong. **Proceedings…** Washington: IEE Computer Society, 2006. p. 548-552.

YAMIN, A. C. **Arquitetura para um Ambiente de Grade Computacional Direcionada às Aplicações Distribuídas, Móveis e Conscientes de Contexto da Computação Pervasiva**. 2004. 204f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

YAMIN, A. C. et al. Towards merging context-aware, mobile and grid computing. **International Journal of High Performance Computing Applications**, Thousand Oaks, v.17, n.2, p. 191-203, May 2003.

YAMIN, A. C. **Um ambiente para a exploração de paralelismo na Programação em Lógica**. 1994. 104f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

YANG, B.; GARCIA-MOLINA, H. Designing a super-peer network. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, 19., 2003, Bangalore. **Proceedings…** Los Alamitos: IEEE Computer Society, 2003. p. 49-60.

ZHU, F.; MUTKA, M.; NI, L. Service Discovery in Pervasive Computing Environments. **IEEE Pervasive Computing**, Los Alamitos, v.4, n.4, p. 81-90, Oct. 2005.

GLOSSARY

Actuator – software-controlled devices that affect the real world.

Adaptation – the action of reacting to changes and creating a dynamic balance between available resources and applications needs. The process of adjusting aspects of applications to changes in operating environments.

Association – "the logical relationship formed when at least one of a given pair of components communicates with the other over some well-defined period of time" (COULOURIS et al., 2005).

Boundary Principle – the borderline of the system corresponds to those of the real world, as they are normally defined territorially and administratively (KINDBERG and FOX, 2001).

Composition – the action of nesting components within one another, making it easy to extend and compose applications.

Context – "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" (DEY, 2001).

Context Awareness – a system that is "cognizant of its user's state and surroundings (…)" (SATYANARAYANAN, 2001). The perception of the context. A user's context can have attributes such as physical location, emotional state, personal history, etc. Also known as perception.

Context Management – a system that modifies its behavior based on the perceived context information. The action of adjusting the system in response to sensed information. Basically consists in adapting the system. Also known as smartness. Using perception effectively (SAHA and MUKHERJEE, 2003).

Cyber Foraging – to "dynamically augment the computing resources of a wireless mobile computer by exploiting wired hardware infrastructure. (…) 'waste' computing resources to improve user experience" (SATYANARAYANAN, 2001). Cyber foraging uses near infrastructure for compute servers or data staging servers.

Dependability – the "ability to avoid service failures that are more frequent and more severe than acceptable" (AVIŽIENIS et al., 2004). Dependability encompasses availability (readiness for correct service), reliability (continuity of correct service), safety (absence of catastrophic consequences on the users and the environment),

integrity (absence of improper system alterations), and maintainability (ability to undergo modifications and repairs).

Deploy – to move code to the location of its execution and bring it into effective action.

Design time – the moment in time when the application is conceived, extended or maintained.

Discovery Services – the action of registering services and looking them up by their attributes. "Discovery lets services and devices spontaneously become aware of the availability and capability of pears on the network without explicit administration" (EDWARDS, 2006).

Error – "the part of the total state of the system that may lead to its subsequent service failure" (AVIŽIENIS, et al. 2004). Error propagation can lead to a failure.

Evaluation – the qualitative and quantitative (including testing) analysis of a system.

Failure – "an event that occurs when the delivered service deviates from correct service" (AVIŽIENIS et al., 2004). This deviation is caused by the propagation of error to the service interface. "The failure of a component causes a permanent or transient fault in the system" (AVIŽIENIS et al., 2004).

Fault – a defect at the lowest level that may cause an error. When a fault is active, it produces error; otherwise, it is dormant (AVIŽIENIS et al., 2004).

Follow-me semantics –the idea that users can go anywhere carrying the data and application they want, which they can use in a seamlessly integrated fashion with the real world.

Framework – an environment, composed of APIs (Application Program Interfaces), user interfaces, and tools, that simplifies software development and management in a specific domain (BERNSTEIN, 1996). Used to build software that runs on a middleware, which can be developed using existing frameworks.

Heterogeneity – "variety and difference" (COULOURIS et al., 2005). Different types of devices, networks, systems, and environments.

Interoperation – the interactions between components during association (COULOURIS et al., 2005). See also spontaneous interoperation.

Invisibility – the action of making computers disappear in the background. Acting unobtrusively, meeting user's expectations.

Load time – the moment in time in which applications are loaded to specific devices.

Logical Mobility – the mobility of components (applications, data and services). A special case is code mobility, a method that makes it possible to create a dynamic change of location in which objects execute.

Logical Sensor – a sensor that infers information from physical or virtual (software) sensors (INDULSKA and SUTTON, 2003).

Meet User Intent – to satisfy user expectation. To achieve this, systems need to capture user intention, acting at an almost subconscious level, and tune themselves without distracting users (SAHA and MUKHERJEE, 2003).

Middleware – "software that provides mediation between other software components, fostering interoperability between those components across heterogeneous platforms and varying resource level" (ADELSTEIN et al., 2005). "Software layer that provides a programming abstraction as well as masking heterogeneity" (COULOURIS et al., 2005).

Mobile Computing – computing services that go with people and become more present, providing expanded capabilities. This arises from advances in two areas: wireless networking and portable devices. With these devices the user can access information anywhere, regardless of their physical location or mobility (JING et al., 1999). Combined with network access, those services transform computing "into an activity that can be carried" (LYYTINE and YOO 2002).

Mobility – the ability to move applications and data freely and easily. Allows access anywhere and any time, regardless of location or displacement.

Ontology – "defines a set of representational primitives with which to model a domain of knowledge or discourse" (GRUBER, 2007).

Open Standards – common guidelines that are published and can be extended in various ways.

Pervasive Computing – a change in the view of computers and their use by humans. Computers are everywhere and are used not as distinct machines but rather as "sophisticated, computerized, networked machines" (HOFFNAGLE, 1999). Computers are parts of larger devices. Pervasive computing and ubiquitous computing are basically different terms used to describe the same concept. The main difference between both concepts is that pervasive computing is a bottom-up vision that emerged from the widespread exploitation of computing services, while ubiquitous computing is a top-down approach were these services are used in a transparent manner and integrated with the environment (ROBINSON et al., 2005).

Pervasive Dependability – dependability in the scope of pervasive or ubiquitous computing. See also dependability.

Physical Mobility – the mobility of users and devices.

Physical Sensor – a hardware device that acts as a sensor.

Privacy – the protection against access to personal data. The protection from being observed or disturbed by other users. The ability to control the accessibility of information about the user (COULOURIS et al., 2005).

Runtime – the moment in time when applications are already loaded and ready to execute or in execution.

Scalability – the property that a system has, so that it "will remain effective when there is a significant increase in the number of resources and the number of users" (COULOURIS et al., 2005).

Seamless integration – association and cooperation among various components in a transparent manner.

Security – guaranteed confidentiality (protection against disclosure to unauthorized individuals), integrity (protection against alteration and corruption), and availability (protection against interference with the means to access the resource). Some

authors also consider the guarantee of authenticity (guaranteeing the identity of components), authority (granting protection rights to users), and non-repudiation (protection against falsely denying sending the data).

Security Mechanisms – technology used in the system to enforce security policies. Protection mechanisms used to ensure security.

Security Policies – defining in a clearly and non-ambiguous form items to be protected. It does not specify how to obtain protection.

Semantic Web – an extension of the human-targeted web that brings meaning to contents in a software-understandable way.

Sensor – "a device that detects or measures a physical property and records, indicates, or otherwise responds to it" (NEW OXFORD AMERICAN DICTIONARY, 2007).

Smart space – "any physical space with embedded services" (COULOURIS et al., 2005).

Software sensor – a software component that acts as a sensor. Also named virtual sensor.

Spontaneous components – components designed to arrive and leave routinely (KINDBERG and FOX, 2002). Components must conform to a common interoperation model to interact.

Spontaneous interoperation – an "interaction with a set of communication components that can change both identity and functionality over time as its circumstances change. A spontaneously interacting component changes partners during its normal operation, as it moves or as other components enter its environment (…)." (KINDBERG & FOX, 2002). Changing the set of components that a certain component communicates with. Components needed to be associated before interoperating. See also association and interoperation.

Tangible Interaction – to create a richer interaction experience, by coupling digital information with physical artifacts, using the human body as an interface and combining real objects and devices with computers in interactive spaces (HORNECKER 2005).

Transparent User Interaction – to preserve human attention during human-computer interaction (HCI).

Trust – Truly believe in an entity. "the establishment of trust enables systems to exchange information even without the intervention of administrators to authorize these interactions" (ROBINSON et al., 2005).

Tuple – a sequence of ordered typed values.

Tuple Space – a global and persistent repository of tuples.

Ubiquitous Computing – the idea of integrating computers seamlessly, invisibly enhancing the real world. A "new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the background" (WEISER, 1991). Ubiquitous computing and pervasive computing are basically different terms used to describe the same concept. The main difference between both concepts is that ubiquitous computing is

a top-down approach were these services are used in a transparent manner and integrated with the environment, while pervasive computing is a bottom-up vision that emerged from the widespread exploitation of computing services (ROBINSON et al., 2005).

Verification – the process of checking where the system adheres to certain properties, called verification conditions.

Virtual sensor – the same as a software sensor. The source of sensor information is obtained by software.

Volatility – "assume that certain changes are common rather than exceptional. The set of users, hardware and software in mobile and ubiquitous systems is highly dynamic and changes unpredictably" (COULOURIS et al., 2005).

APPENDIX A CONTINUUM: UMA INFRA-ESTRUTURA DE SOFTWARE SENSÍVEL AO CONTEXTO E BASEADA EM SERVIÇOS PARA A COMPUTAÇÃO UBÍQUA

No clássico e visionário artigo sobre computação para o século 21, Mark Weiser (WEISER, 2001) resume o que é esperado da computação ubíqua (também chamada de ubicomp): acesso do usuário ao ambiente computacional, de todo lugar e a todo momento, por meio de qualquer dispositivo. A dificuldade reside em como desenvolver aplicativos que irão continuamente se adaptar ao ambiente e continuar funcionando, a medida que as pessoas se movem ou trocam de dispositivos (GRIMM et al., 2001). O desenvolvimento dessa área, entretanto, ainda é limitado pelo número exíguo de linguagens e ferramentas disponíveis (ROMÁN et al., 2002). Além disso, aplicações conscientes de contexto ainda estão sendo executadas em laboratórios ao invés de estarem presentes em ambientes reais do dia-a-dia (HENRICKSEN and INDULSKA, 2006).

Aplicações ubíquas precisam de um *middleware* para interoperar entre muitos dispositivos diferentes e as demandas do usuário final (SAHA and MUKHERJEE, 2003). O objetivo é esconder a complexidade do ambiente, isolando aplicações do gerenciamento explícito de protocolos, acesso distribuído à memória, replicação de dados, falhas de comunicação, etc. Um middleware também pode resolver problemas de heterogeneidade relacionados às arquiteturas, sistemas operacionais, tecnologias de redes e até mesmo de linguagens de programação, promovendo a interoperação entre esses componentes. Por outro lado, um *framework* é um ambiente, composto de APIs (Interfaces de Programação com os Aplicativos), interfaces com o usuário e ferramentas, simplificando o desenvolvimento de software e o gerenciamento em um domínio específico (BERNSTEIN, 1996). É possível utilizar framework para construir software que executa em um middleware, o qual pode ser desenvolvido utilizando frameworks existentes.

Um middleware deve permitir que o usuário acesse o ambiente computacional dele (dados e aplicativos) de qualquer lugar e a qualquer momento. Uma solução possível é aplicar a semântica siga-me (AUGUSTIN et al., 2004; YAMIN et al., 2003). A idéia desse conceito é que aplicativos e dados vão juntos com os usuários, fornecendo um ambiente virtual e adaptando ao contexto corrente. Essa adaptação é fundamental para a visão de computação ubíqua, e envolve a percepção do contexto (*context awareness* – sensibilidade ao contexto) e o próprio ajuste do sistema baseado na informação percebida (gerência do contexto).

A idéia defendida nessa tese é de que o uso de uma infra-estrutura de software especificamente orientada a ubicomp pode reduzir a distância entre a visão de Weiser e o cenário atual da computação distribuída. Para atingir esse objetivo, esse trabalho foca no desenvolvimento de uma infra-estrutura de software baseada em serviços para a computação ubíqua, empregando framework e middleware, denominada Continuum. Esta proposta difere de outros trabalhos, tais como Aura (GARLAN et al., 2002), Gaia (RÓMAN et al., 2002) e One.World (GRIMM et al., 2004), porque o foco principal é mais geral: Continuum não é especificamente destinado ao usuário ou um ambiente específico, mas sim para um visão global, não limitada por um escopo local ou pessoal.

O foco principal do trabalho é em sensibilidade ao contexto, de forma que o ambiente incorpore as características necessárias pelos usuários, melhorando à experiência no mundo real. Para atingir esse objetivo, foi proposta a redefinição da semântica siga-me: usuários podem ir para qualquer lugar carregando os dados e os aplicativos que desejam, os quais podem ser usados de forma imperceptível e integrada com o mundo real (*seamless integration*). Para atingir esse objetivo, o trabalho apresenta o detalhamento do Continuum como um sistema sensível ao contexto, abrangendo aspectos como: percepção das características do usuário e do entorno, histórico de dados de contexto e representação desses dados através de uso de ontologias, promovendo raciocínio e compartilhamento de conhecimento.

Resumidamente, as principais contribuições da presente tese são:

- Uma revisão da área de computação ubíqua, compreendendo origem, evolução e principais desafios (apresentada no capítulo 2);
- Uma proposta de um modelo de arquitetura abrangente para a computação ubíqua, auxiliando a comunidade a desenvolver e comparar middleware e frameworks nessa área. Outro objetivo dessa arquitetura é destacar os requisitos de uma infra-estrutura de software para a ubicomp (esse modelo é descrito no capítulo 3);
- A proposição da infra-estrutura de software Continuum, como uma evolução do projeto ISAM. O projeto ISAM vem sendo desenvolvido na UFRGS há alguns anos e envolve algumas teses de doutorado e diversas dissertações de mestrado. A descrição do Continuum como infra-estrutura de software é apresentada na parte I do texto da tese e detalhada no capítulo 4;
- Um modelo, bem como uma notação, para representar o mundo real no Continuum. Tal modelo considera as três entidades que podem ser distinguidas quando se trabalha com contexto (de acordo com DEY et al., 2001): *lugares*, *pessoas*, e *coisas*. O modelo define oito abstrações básicas para serem representadas e três tipos de relação (o detalhamento desse modelo é apresentado na seção 4.3);
- A sugestão de uma infra-estrutura para melhorar o suporte aos dispositivos móveis no Continuum, empregando conceitos de Web 2.0 (SCHROTH, 2007). Dessa forma, dispositivos móveis de propósitos especiais (*gadgets*) acessam a infra-estrutura de forma *ad hoc*, utilizando um navegador web e AJAX (detalhes dessa infra-estrutura são apresentados na seção 4.4);
- A proposta de uma arquitetura para suporte a serviços, empregando o uso da Arquitetura Orientada a Serviços (SOA) e *webservices* (PAPAZOGLOU and

GEORGAKOPOULOS, 2003). A idéia de obter funcionalidade como serviços disponibilizados via rede, corresponde a um modelo denominado Software como Serviço (SaaS). A modelagem dessa arquitetura está descrita na seção 4.5;

- A modelagem de vinte serviços para uma infra-estrutura de software na computação ubíqua, divididos em quatro subsistema. Treze desses serviços são totalmente novos, enquanto sete são baseados em trabalhos anteriores do grupo, tais como EXEHDA (YAMIN, 2004), ISAMAdapt (AUGUSTIN, 2004) e ACTUS (SILVA, 2008). A organização em subsistema é conceitual: não um elemento nela mesma; mas ao invés um grupo de serviços relacionados. Os subsistemas propostos são: Execução Distribuída (descrito na seção 4.6.1), Gerenciamento de Adaptação (detalhado na seção 4.6.2), Sensibilidade ao Contexto (especificado na seção 5.3) e Interação com o Usuário (apresentado na seção 4.6.3);
- A sugestão de algumas considerações gerais para o desenvolvimento do framework do Continuum. A proposta sugere que essa camada lide com características essenciais relacionadas na arquitetura abrangente (descrita na seção 4.7);
- O projeto da arquitetura de contexto do Continuum, incorporando a representação formal de contexto (através de uma ontologia), o uso de descoberta de recursos, a proposta do armazenamento de contexto histórico em um banco de dados e a distribuição / localização de informações de contexto. A apresentação do Continuum como um sistema sensível ao contexto está na parte II do trabalho e seu detalhamento é realizado no capítulo 5;
- A proposição de um modelo multicamadas para sistemas sensíveis ao contexto, enfatizando os principais serviços necessários. Analisando cada camada do modelo é discutido os principais conceitos de projeto relacionados a sensibilidade de contexto. O modelo é detalhado no capítulo 6;
- A avaliação da solução proposta para sensibilidade ao contexto e a comparação com o estado da arte na área (descritas na seção 6.3);
- A análise e avaliação das principais contribuições do Continuum. O método utilizado para validação foi baseado em avaliação experimental, ou seja, foram propostos três casos de estudo para apreciar a arquitetura de serviços distribuídos, a representação da ontologia com o respectivo armazenamento e o subsistema de sensibilidade ao contexto. O capítulo 7 apresenta os experimentos realizados e a análise dos principais resultados obtidos;
- O desenvolvimento de um estudo de caso para discutir a proposição da arquitetura de serviços distribuídos proposta no Continuum. Para atingir esse objetivo, foram modelados e implementados os serviços *Executor* e *Service Manager*. Um protótipo foi gerado e alguns testes realizados (o estudo de caso 1 é descrito na seção 7.2);
- A descrição de um estudo de caso relacionado com o Continuum como um sistema Sensível ao Contexto. Nesse estudo foi analisado a representação

formal do contexto e a capacidade de armazenar e inferir conhecimento. Utilizando um conjunto de ferramentas, tais como Protégé[46], Jess[47] e Pellet[48], a ontologia foi modelada. A seguir, foi definido um cenário exemplo para que uma base de conhecimento pudesse ser modelada. Utilizando esses dados, um conjunto de inferências foi conduzido (o estudo de caso 2 é detalhado na seção 7.3);

- O último estudo de caso realizado teve por objetivo analisar as possibilidades na implementação do subsistema de sensibilidade ao contexto, empregando as ferramentas e padrões abertos disponíveis atualmente. Nesse estudo, não foi criado nenhum protótipo. O trabalho consistiu em analisar tecnologias que poderiam ser empregadas no subsistema (o estudo de caso 3 é apresentado na seção 7.4).

Acreditamos que o trabalho desenvolvido ajuda a diminuir a distância entre a visão de computação ubíqua do Weiser e o cenário atual de sistemas distribuídos. Esse objetivo foi alcançado pela proposição da infra-estrutura de software Continuum, como pode ser observado pelos resultados apresentados. Por fim, consideramos que futuras infra-estruturas para ubicomp devem, assim como o Continuum propõe, integrar de forma imperceptível vários desafios diferentes.

[46] A ferramenta pode ser baixada em < http://protege.stanford.edu/>.

[47] Jess pode ser encontrado em <http://www.jessrules.com/>.

[48] Informações sobre o Pellet podem ser obtidas em <http://pellet.owldl.com/>.

APPENDIX B RELATED WORK ON CHALLENGES OF UBIQUITOUS COMPUTING

Debashis Saha and Amitava Mukherjee (2003) highlight scalability, heterogeneity, integration, invisibility, context awareness, and context management as challenges to be addressed. All these aspects, with the exception of integration, which is not directly debated, are also presented here; instead, we include discussion on integration in invisibility (focus on the seamless aspect) and in spontaneous interoperation (focus on association and communication).

Tim Kindberg and Armando Fox (2002) based their proposal on two fundamental characteristics: physical integration and spontaneous interoperation. They also emphasize some common areas in ubicomp scenarios, all directly or indirectly discussed in our proposal, namely: discovery, adaptation, integration, programming framework, robustness, and security.

The article written by Guruduth Banavar and his colleagues at IBM (2000) underlines a device-independent application development process, with a highly dynamic load time system that embraces discovery, negotiation, and dynamic selection of presentation. At execution, they propose a dynamic sharing of resources, the migration of applications, and failure detection and recovery. Of these, data sharing is the only feature that we do not list; instead, we consider it to be a part of (logical) mobility.

Eila Niemelä and Juhani Latvakoski (2004) propose the following requirements: interoperability, heterogeneity, mobility, survivability and security, adaptability, self-organization, and augmented reality with scalable content. Perhaps the main difference lies in the concept of self-organization, which amplifies the idea of adaptation, and of augmenting reality, by adding a virtual context to the one sensed by users.

Robert Grimm and his contemporaries at the University of Washington (2001) suggest three “fault lines” for ubicomp, caused by transparency, heterogeneity, and the use of a single abstraction for data and code. To address this last issue, they recommend maintaining data and functionality separated. In our article, this is not tackled, but it is possible to satisfy this condition by using a different representation for data, such as tuples.

An article by Intel researchers Roy Want and Trevor Pering (2005) proposes the following challenges: power management, discovery, user interface adaptation, and location-aware computing. We do not directly consider power management in this work; as an alternative, we present a more general approach: context management. The same applies to location awareness, which is also covered by context awareness.

Modahl et al. proposes a taxonomy for the building blocks of a software infrastructure, UbiqStack. It has five subsystems: registration and discovery; service and subscription; computation sharing; context management; and data storage and streaming (Modahl et al., 2006). The first four categories correspond roughly to our more generic discovery, interoperation, cyber foraging, and adaptation. We do not directly address the fifth one, but we believe that our proposal is more comprehensive, since we make allowance for several other categories. This is the only article from the list that proposes a software architecture for ubicomp, although Banavar and colleagues (2000) offer a new application model considering its life cycle.

APPENDIX C COMPETENCY QUESTIONS FOR CONTINUUM ONTOLOGY

- Where is John located?
- Where was John yesterday at 17:00?
- What services are available in this cell?
- What is the history of this place?
- Who is present in this cell now?
- What are the devices in this cell?
- What are the sensors available in this cell?
- Which node is the base of the cell?
- Which are the devices in the cell?
- What is my current location?
- What applications do I have?
- What are the actuators available in this cell?
- What information is attached to this cell?
- What are the resources of this cell?
- Which cell has a specific service?
- Is this device mobile?
- Is this device a gadget?
- Which is the outer cell of this cell?
- What devices are aggregated to the cell?
- What devices are composed inside this cell?
- What devices are associated to a cell?
- Who is John?

APPENDIX D LIST OF IMPORTANT TERMS IN CONTINUUM ONTOLOGY AND THEIR MEANING

- Action: an event performed by a being;
- Actuator: a particular type of CoNode, which is a software-controlled device that affects the real world;
- Application: a software component designed to fulfill a particular purpose;
- Being: a living organism, such as animals and human beings;
- *CoBase*: a device in charge of managing a CoCell and responsible for interaction with other cells;
- *CoCell*: an element that represents a place and comprises a CoBase and other nodes in that physical location. CoCells are nested forming a hierarchy;
- *CoDimension*: a place that physically abstracts the real world embracing all modeled entities;
- *CoGadget*: a device that accesses users' applications in a more ad hoc manner. In general, a special purpose device, such as a smart phone or a PDA;
- *CoMobi*: a device with mobile capacity, executing users' applications and making use of Continuum services. Usually, wirelessly connected and with a more restricted capacity, generally in terms of network latency, processing speed, available memory, and power supply;
- Component: each piece of software registered in Continuum, normally an object or set of objects;
- *CoNode*: a device executing users' applications and making use of Continuum services. Typically a stationary device;
- *CoPerson*: a person registered in the infrastructure that is physically present in one CoCell;
- Device: a special purpose thing. In the Continuum infrastructure it is a CoNode, CoBase, CoMobi, or CoGadget;
- Entity: a place, a thing, or a being with a distinct and independent existence;
- Event: a particular occasion that involves entities and occurs in a specific timestamp;

- Person: a human being;
- Place: a particular area already identified by the human notion;
- Resource: an attribute or a capability offered by a device. The device's assets;
- Sensor: a particular type of CoNode, which detects or measures a physical property;
- Service: a component of Continuum that supplies some functionality to the user. It follows service-oriented architecture (SOA) and is implemented as a web service;
- Space: information that denotes the location of a place, such as latitude and longitude;
- Thing: a material object that can be seen and touched;
- Timestamp: information that denotes date and time, in which a certain event occurs.