A Context-Aware Multi-Agent System for AmI Environments

Andrei Olaru

Advisors:

Prof. Adina Magda Florea
University "Politehnica" of Bucharest

Prof. Amal El Fallah Seghrouchni
Université Pierre et Marie Curie (Paris 6)

2011

FONDUL SOCIAL EUROPEAN
Investeşte în oameni!
Programul Operaţional Sectorial pentru Dezvoltarea Resurselor Umane 2007 – 2013
Proiect POSDRU/6/1.5/S/16 – *Doctoranzi în sprijinul inovării şi competitivităţii*

UNIVERSITATEA **POLITEHNICA** DIN BUCUREŞTI
Facultatea de Automatică şi Calculatoare
Catedra de Calculatoare

UNIVERSITATEA PIERRE ET MARIE CURIE (PARIS VI)
Ecole Doctorale EDITE de Paris
Laboratoire d'Informatique de Paris 6

Nr. Decizie Senat 212 din 30.09.2011

TEZĂ DE DOCTORAT

Un sistem multi-agent dependent de context pentru medii de Inteligență Ambientală

A Context-Aware Multi-Agent System for AmI Environments

Autor: Ing. Andrei Olaru

COMISIA DE DOCTORAT

Preşedinte	Prof. dr. ing. Dumitru POPESCU	de la	Universitatea Politehnica Bucureşti
Conducător de doctorat-1	Prof. dr. ing. Adina Magda FLOREA	de la	Universitatea Politehnica Bucureşti
Conducător de doctorat-2	Prof. dr. ing. Amal EL FALLAH SEGHROUCHNI	de la	Universitatea Pierre et Marie Curie (Paris VI)
Referent	Prof. dr. ing. Costin BĂDICĂ	de la	Universitatea din Craiova
Referent	Prof. dr. ing. Salima HASSAS	de la	Universitatea Claude Bernard (Lyon 1)
Referent	Prof. dr. ing. Cristian GIUMALE	de la	Universitatea Politehnica Bucureşti
Referent	Prof. dr. ing. Nicolas MAUDET	de la	Universitatea Pierre et Marie Curie (Paris VI)

Bucureşti 15 decembrie 2011

UNIVERSITÉ PIERRE ET MARIE CURIE - PARIS 6

ÉCOLE DOCTORALE EDITE

en cotutelle avec

UNIVERSITÉ POLITEHNICA DE BUCAREST

T H È S E

pour obtenir le titre de

Docteur en Sciences

de l'Université Pierre et Marie Curie - Paris 6

Mention : Informatique

Présentée et soutenue par

Andrei Olaru

Un système multi-agents sensible au contexte pour les environments d'intelligence ambiante

A Context-Aware Multi-Agent System for AmI Environments

Thèse dirigée par

Amal El Fallah Seghrouchni *et*

Adina Magda Florea

préparée au Laboratoire d'Informatique de Paris 6 (UMPC)

et au Département d'Informatique (UPB)

soutenue le 15 décembre 2011

Jury :

Rapporteurs :	Salima Hassas	-	Université Claude Bernard – Lyon 1
	Costin Bădică	-	Université de Craiova
Directeurs :	Amal El Fallah Seghrouchni	-	Université Pierre et Marie Curie
	Adina Magda Florea	-	Université Politehnica de Bucarest
Examinateurs :	Nicolas Maudet	-	Université Pierre et Marie Curie
	Cristian Giumale	-	Université Politehnica de Bucarest

Abstract

Ambient Intelligence is the vision of a ubiquitous electronic environment that is non-intrusive, but also pro-active, and that helps people in a personalized and context-aware manner, in their day-to-day tasks. Most implementations of Ambient Intelligence systems realized over the last decade have focused on implementing complete systems that serve specific purposes. In our work we focus on that layer of an AmI system that handles the semantic-aware exchange of information in order to deliver the relevant information to the interested user.

This thesis presents the research towards the realization of a multi-agent system for Ambient Intelligence, that assures the context-aware flow of information, and in which context-awareness is integrated so that agents naturally manage and share context information, in a generic manner. Our approach to building a context-aware multi-agent system for Ambient Intelligence relies on three aspects: a graph-based representation of context information, that is coupled with the definition of context patterns for situation recognition; a context-related topology of the system, in which neighborhood relations reflect the existence of shared context between agents; and a local agent behavior that is based on mechanisms of self-organization in order to provide coherent global results.

The theoretical research that is presented in this work has been validated by means of three projects: AmIciTy:Mi deals with the spread of information in a system formed of a large number agents; the Ao Dai prototype demonstrates the mapping of context structure to agent hierarchies; and the Ao Dai platform, which is a multi-agent system for the implementation of AmI applications.

Contents

List of Figures

List of Tables

Acknowledgements

I would like, in this chapter, to thank the people without whom this research would not have been possible, and to the people who have inspired me and advised me before and throughout this time of my life.

I would like to thank to my two PhD advisors, Professor Adina Magda Florea and Professor Amal El Fallah Seghrouchni, for keeping me on the right track but also for giving me a degree of autonomy that has allowed me to make some of my own choices. Their help and council, in problems not only related to research, but also to administrative or lateral issues, have been priceless.

I would like to thank to the colleagues that have worked together with me for the implementation and, sometimes, for the design of the projects that are presented in this work. I would like to thank to Cristian Gratie, which was an invaluable partner during the AmIciTy:Mi project and a worthy coauthor of our papers together. I would like to thank to Diego Salomone Bruno for the participation in the implementation of the Ao Dai prototype, to Thi Thuy Nga Nguyen for our collaboration both in the Ao Dai prototype and, later, in the Ao Dai platform, and to Marius-Tudor Benea, for his will to continue working on interoperation with JADE-LEAP, in spite of the problems that arose. I would like to thank to Sofia Neaţă, who has accepted to work on a platform that she was not familiar with, for her short internship.

I would like to thank to all the other people that gave me advice during this work, and that I have not mentioned here. Very special thanks go to Claudia Marinica and Professor Fabrice Guillet for introducing me to scientific research during my Master internship in Nantes. Additional thanks go to my office colleagues during some of my time in Paris, Cédric Herpson and Sylvain Ductor.

Special thanks also go to the people that gave me logistical assistance through this time, as I was never able to be in two universities at once. Among them, Andreea Urzică, Mihaela Puică and Cédric Herpson.

I would like to thank to my parents, for their moral and financial support during this time, and again to my father for asking me many times what my thesis is about and discussing with me the concepts in my thesis.

Introduction

The term of Ambient Intelligence has been coined at the dawn of the 21st century, in 2001, with the report of the ISTAG group [Ducatel et al., 2001], when it became one of the priorities in the ICT domain in the European Union and worldwide. Ambient Intelligence – or AmI, for short – was envisaged as a ubiquitous, unitary, electronic environment that would assist people in many or all of their life's aspects and in a considerably varied number of manners.

Ambient intelligence should represent the third wave in computing. After the mainframe and the personal computer, in the age of Ambient Intelligence the devices become invisible, by being integrated in all objects and materials. This makes everything become "smart" and, by means of communication, everything around us will collaborate in order to offer more complex functions and more relevant results. AmI also represents an evolution of what is now the Internet: web-based, collaborative and social services that assist the user in daily activities.

Ambient Intelligence is a vast domain, which gave way to many directions research and development. We could consider an AmI system as containing several layers: the hardware layer, the network / interconnectivity layer, an interoperability layer, the application / smart services layer, and the intelligent user interfaces. This work focuses on building solutions for the application layer of Ambient Intelligence, more precisely on designing and building a multi-agent system for AmI that integrates context-awareness, so that agents naturally manage and share context information.

This Thesis takes place in the context of a long collaboration between Prof. Amal El Fallah Seghrouchni and Prof. Adina Magda Florea, that has already yielded the PhD Thesis of Alexandru Suna, several exchanges concerning Master students and the FP7 project ERRIC.

This Thesis is in cotutelle between University Pierre et Marie Curie and University Politehnica of Bucharest. It is being funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/6/1.5/S/16, by Laboratoire d'Informatique de Paris 6 (LIP6), and by Agence Universitaire de la Francophonie.

Motivation

Many implementations of Ambient Intelligence systems so far have tried to realize complete AmI systems, containing elements from all layers, especially hardware, application and interface. But trying to build the hardware, the middleware, the interfaces and the applications resulted in the implementation losing either generality, or representation power, or flexibility and scalability.

This is why we chose to focus on only one layer of an AmI system: the layer that works with information at a semantic level, and that is central to the context-awareness of the whole system.

As many other researchers, we have chosen the agent-oriented paradigm for the modeling and implementation, because agents offer features that are important to AmI, notably autonomy and pro-activity.

In order to make this multi-agent system act as a middleware for a wider range of future AmI systems, one priority of this work was to keep the system generic (as opposed to domain- or application-specific). Requirements of availability and dependability also demand for system distribution, and the possibility for agents to work alone or in organizations of various sizes. Moreover, deploying agents on devices with a wide range of capabilities – from small sensors to powerful workstations – meant that agents needed to use flexible and adaptable representations and algorithms.

Objectives

The research question that this work will attempt to answer is **"How to build a multi-agent system for the application layer of Ambient Intelligence?"**.

As a result of the requirements presented in the previous section, several **main goals** have been defined for this research:

- to develop a multi-agent system model for Ambient Intelligence that features self-organization, context-awareness and anticipation;
- to develop several scenarios that emphasize the requirements of real-scale Ambient Intelligence environments;
- to develop a simulation testbed that implements the elements of the said scenarios, to serve for experiments with AmI platforms;
- to implement and experiment with the developed model, using the simulation testbed, in order to prove the model's validity as a component of an Ambient Intelligence environment.

As research went on toward these goals, some **secondary goals** also emerged. Among others, we can cite:

- realization of a state of the art in scenarios for Ambient Intelligence, identification of the features presented and classification on layers;
- a state of the art in implementations for Ambient Intelligence, notably implementations using multi-agent systems;

- evaluation of approaches to context-awareness in Ambient Intelligence;
- realization of a survey of self-organizing multi-agent systems, particularly in the case of cognitive agents;
- identification of algorithms and approaches for graph matching, focused on labeled graphs;
- development of a platform for the simulation and testing of complex systems formed of a large number of agents;
- development of tools for the evaluation, visualization and logging of the behavior of multi-agent systems formed of a large number of agents;
- development of algorithms and paradigms for cognitive agents that use mechanisms of self-organization to form systems with emergent behavior;
- development of measures of context-awareness and of an improved representation for context information;
- development of context-aware features for the agent system relying on more aspects of context (specifically, relying not only on spatial context);
- development of agent and relation types for the description of a system topology that reflects the context of agents;
- improvement of the CLAIM agent-oriented programming language and adaptation of CLAIM agents to execute on the JADE agent development framework;
- development of a textual and a graphical representation for directed and attributed graphs, based on the extraction of their linear components.

The realization of these objectives is presented throughout this work. A detailed view on the different parts and chapters of this work is presented in the next section.

Structure of the Thesis

The purpose of the first chapter – **Chapter 1 "Defining the problem"** – is to state the research problem that we are trying to solve in the rest of this work. In the first section of the chapter we will examine in detail the concept of Ambient Intelligence, by looking at the most relevant scenarios in the literature, at the applications of the concept, and at the features that are expected to be provided by AmI. These features will be further classified and discussed individually in Section 1.2.

Based on the better understanding of the features of AmI, we propose some new scenarios, that are directed less on how AmI will be viewed by the user, and more on hinting how an AmI system should work on the inside and what requirements it should fulfill (Section 1.3).

After a brief look into some paradigms that are good candidates for the implementation of AmI, like, context-awareness, and the use of self-organization (Sections 1.4.1, 1.4.2 and 1.4.3), we will state the goals of this research in the last section (1.5), leaving to the other chapters of this work the detailing of how these goals have been fulfilled.

After the goals of this research are clear, the next chapter – **Chapter 2 "A**

State of the Art in Related Fields of Study" – examines work that is related to our research: the integration of software agents and multi agent systems in the implementation of Ambient Intelligence; context-awareness and the representation of context; and finally self-organization in cognitive agent systems.

The review of agent-based AmI environments (Section 2.1) looks into how agents can be used for AmI, what features they provide and what AmI layer(s) they compose, and whether they are used more like individual, autonomous, reasoning entities or more like connected parts of a distributed whole.

Since context-awareness is a defining feature of AmI, in Section 2.2 we review the current representations of context information, as well as how context information is retrieved and exchanged among agents. We emphasize the works on the representation of context as associations and on the use of ontologies.

The last section of the chapter (2.3) discusses self-organization and mechanisms that can be used to coordinate distributed entities without central control, especially in the context of using cognitive agents.

In building a Multi-Agent System for Ambient Intelligence, the first aspect that we have focused on was agent behavior – **Chapter 3 "Agent Behavior: Relying on Self-Organization"**. That is, how should agents exchange information so that interesting information reaches interested agents, without centralized control.

After discussing the motivation for the approach, we present in Section 3.2 the AmIciTy:Mi middleware for AmI applications, a proof-of-concept that demonstrates how agents can rely only on local knowledge and interaction for the sharing of information, but obtain a global, emergent result at the level of the system, where the spread of information can be easily controlled by means of some simple context measures. The platform has underwent considerable testing and yielded favorable results, clearly defining a valid agent behavior. The results of the evaluation, as well as additional tools for the definition of scenarios and visualization of the system, are presented in Section 3.3.

The focus moves away from agent behavior, and toward the topology of the system in **Chapter 4 "Structuring the Agent System"**, in which research deals with mapping the hierarchical structure of some context aspects to a logical hierarchy of agents. The Ao Dai project (see Section 4.2) is implemented in the CLAIM agent-oriented programming language and assigns an agent to each element of context. Dynamic context is supported by means of mobile agent hierarchies – an agent can move seamlessly together with its entire context.

Context-awareness is the focus of what is probably the most important chapter of this work – **Chapter 5 "Improving Context-Awareness"**. The chapter contains the formal model for a holistic approach to context-awareness. The elements for which AmIciTy:Mi used simplified approaches – namely system topology and context representation – are improved for a realistic and more powerful model.

Inside the agent – Section 5.2 – context information is represented by the graph of concepts which are relevant to the current state of the agent. The interests of agents are defined as a set of context patterns, which are graphs with generic elements that match a wider range of situations.

Outside the agent – Section 5.3 – the topology is reflecting the context of the agent, that is, the neighborhood relations with the other agents reflect the shared context between them. This approach is based on the Ao Dai project (see Chapter 4) but deals with a wider range of context aspects.

The connection of the two sides – the inside and the outside of the agent – is done through the behavior of agents, which is very similar to the behavior of AmIciTy agents, but this time uses context graphs and patterns and the new agent topology. This behavior, together with an extended example based on the reference scenario (see Section 1.3.4), is presented in Section 5.4.

The practical outcome of this research, and the implementation of the concepts presented in the previous chapters, have lead to the development of an original platform – **Chapter 6 "A New Platform for AmI Applications"**. The motivation for this platform is the need for an appropriate testing suite for the application layer of Ambient Intelligence. The platform is underpinned by the JADE agent developing framework, uses the S-CLAIM agent-oriented programming language – a simplified and improved version of CLAIM – and features tools for the centralized visualization and tracing of the agent system. The architecture of the platform and implementation details are presented in Section 6.2. Section 6.3 gives an insight on how the platform was tested.

The last Chapter of this thesis – **Chapter 7 "Conclusions"** – summarizes the achievements of this work. What has been accomplished is, more than anything, research on how to build a multi-agent system to serve as a middleware for Ambient Intelligence, at the application layer. Details on the individual contributions of this work are given in Section 7.1.2.

This work is only a phase, a beginning. Many paths for the development of the concepts that we have introduced remain open and only too little explored. The implementations that were realized deserve to be improved and extended, and many concepts need further testing in applications and scenarios always closer to real life. Some of these potential targets for the future are presented in the last Section of this work, Section 7.2.

Chapter 1

Defining the Problem

The purpose of this chapter is to state the research problem that we are trying to solve in the rest of this work. In the first section of this chapter we will examine in detail the concept of Ambient Intelligence, by looking at the most relevant scenarios in the literature, at the applications of the concept, and at the features that are expected to be provided by AmI. These features will be further classified and discussed individually in Section 1.2.

Based on the better understanding of the features of AmI, we propose some new scenarios, that are directed less on how AmI will be viewed by the user, and more on hinting how an AmI system should work on the inside and what requirements it should fulfill (Section 1.3).

After a brief look into some paradigms that are good candidates for the implementation of AmI, like Multi-Agent Systems, context-awareness, and the use of self-organization (Sections 1.4.1, 1.4.2 and 1.4.3), we will state the goals of this research in the last section (1.5), leaving to the other chapters of this work the detailing of how these goals have been fulfilled.

1.1 The Vision of Ambient Intelligence, or AmI[1]

Ever since the first computers came into existence, people hoped that one day the computers would get from being used by governmental or corporate actors to helping the common individual in his day-to-day tasks, in an intuitive way, and that computers would interact with humans naturally, just like other humans[2]. Several steps forward toward this goal have been taken: the personal computer in the late 1970s and the 1980s; the rise of the Internet in the 1990s; and the spread of handheld "smart" devices in the decade after the year 2000. However, even with the wide range of services and features that are now available to the user by means of Internet-connected devices, the system

[1] Some parts of this work have been published in our paper for IDC 2010 [**Olaru** et al., 2010a].

[2] This is shown to us in the science-fiction movies across the decades, from *The Forbidden Planet* and *2001: A Space Odyssey*, to *Minority Report*.

formed by these services fails on two important aspects: it does not appear as intelligent to the user (despite actually considerable advances in technology and performance) and it still requires a certain degree of tech savviness in order to be used, as the interaction between the system and the user does not occur in the same natural manner as between two humans.

Ambient Intelligence is a concept that was introduced in the late 1990s and that primarily addresses the two issues mentioned before. Ambient Intelligence is a ubiquitous electronic environment that is non-intrusive, but also pro-active, and that helps people, in a personalized and context-aware manner, in their day-to-day tasks. While the notion of Ambient Intelligence was introduced in 2001 by the ISTAG report [Ducatel et al., 2001], it is based on the older concept of Ubiquitous Computing introduced by Weiser a decade earlier [Weiser, 1993, Weiser, 1995].

Ambient Intelligence should represent the third wave in computing, after the mainframe and the personal computer, by integrating the computing power into the environment, and, instead of using dedicated interfaces to interact with the user, should use AmI-enabled everyday objects. This will make the computers "invisible" and, by using everyday objects as interface, more intuitive and natural to use.

But beyond these vague statements, when it comes to the implementation of the actual AmI environments, it is necessary to understand what should AmI actually mean and to clearly set the features that AmI should provide. In the past decade a considerable volume of research was carried out in the domain of Ambient Intelligence (see section 2.1 for details). However, it is hard to find the balance between the implementation of a complete system with a specific coverage (in terms of space or function) and the research of a specific layer of Ambient Intelligence with general applications. Moreover, there are many applications of AmI that are focused on interface or on interconnectivity issues (which is of course very useful for AmI as a domain) and less on the "intelligent" aspect that will give a clear advantage for the use of AmI [Ramos et al., 2008, Böhlen, 2009, Aarts and Grotenhuis, 2011].

In our work, we argue that an essential aspect in a true Ambient Intelligence system is information, and how information moves through the system, between devices, and between users – Weiser calls this "a world of information conveyers" [Weiser, 1993]. This work will propose mechanisms and representations that assure that information will be exchanged and delivered in a way that is relevant, context-aware, and intelligent.

Throughout the rest of this section, we will present some scenarios and applications from the AmI literature (Sections 1.1.1 - 1.1.3), the expected features of an AmI system (Section 1.1.2), as well as a classification of these features on five layers (Section 1.1.4).

1.1.1 Scenarios

First of all, Ambient Intelligence is a vision. A dream about the future. A future when people will be able to use the Internet and computing capabilities without being confined to a chair and a desk, or even to the small and cluttered screen of an attention-grabbing mobile device. When computing will be free and all around us. When the environment will understand what we want (and need) without us having to explicitly specify that in an interface with its own rules and language. When the artificial intelligence will become ambient and will interact with us much like a human would, in a natural way.

This vision has been described in the field of Ambient Intelligence (AmI) primarily by means of scenarios. These are stories in which the main character(s) use and interact with the Ambient Intelligence. From the scenarios we can extract the features of AmI and insights into how AmI should work, and how it should be organized.

In this section we will examine some relevant scenarios for Ambient Intelligence and Ubiquitous Computing, and then we will extract and classify the features that are described in these scenarios.

One of the first and, without doubt, most important scenarios for Ambient Intelligence is the scenario devised by Weiser in 1995. He called it Ubiquitous Computing, but it referred to a very similar concept. It was the ISTAG advisory group that was among the first to use the term of Ambient Intelligence and that devised 4 scenarios and analyzed them in detail.

Weiser's "Sal" Scenario

We will begin with Weiser's scenario featuring Sal [Weiser, 1995], one of the first scenarios for Ambient Intelligence. The first thing that one should notice in the scenario is that when the alarm clock – an intelligent appliance – asks "coffee" and Sal answers, the only responses that the clock can interpret are "yes" and "no". That is, the appliance can only understand events (in this case, the utterance of a positive or negative response) that are relevant to its function.

Intelligent windows that can display traces of neighbours that passed on the street (as well as public information like weather). Their function may be the output of a centralized service, but it may be completely local: when the neighbours pass, the window senses their presence.

While having breakfast, Sal marks some news (from a printed paper) with a smart pen, having the associated text sent to her office. The text comes from a service related to the paper (perhaps a website), that may be contacted by the pen itself or by an unspecified intermediary agent. If the paper sends the data directly to Sal's office, that may be a privacy concern (why should the paper know Sal's contact data?). A solution would be that the request is anonymous and the response comes back to the agent, which in turn sends it to the office.

Other services are mentioned, relating to the most common points in AmI

scenarios: localization and information on points of interest. Then, Weiser presents more advanced uses for tabs – small displays with two buttons and wireless connectivity: gestures, storage of small pieces of information, and information-related layout. What is relevant here is that each tab relates to one piece of information, and that individual tabs can be used to point out various events. In the end, a context-aware application: finding a person (Mary) that shared a certain context, in terms of space, time, and event, and, by this association, finding the person's contact details. Finding Mary may not be special if the meeting exists in an online calendar's events and invitations (like the Google Calendar today), but it may be particular if the search is done by associations between context data related to the time of the meeting, the place, the number of people, and also to the fact that Sal did not previously know Mary.

The ISTAG Scenarios for Ambient Intelligence in 2010

The scenarios created by the ISTAG group envision Ambient Intelligence in 2010 [Ducatel et al., 2001]. While now, in 2010, most interfaces presented have not yet been developed, it is surprising how many of the services exist in one way or another. In the scenarios and in the annexed analysis, the authors emphasize the hardware and human interface features. In the following paragraphs, we will also make observations on possibilities for the internal functionality of the system.

The "Maria" scenario emphasizes two directions: first the movement of the user's electronic identity (keys), preferences and data with her, seamlessly, as well as the capacity to easily use the local resources: vehicles, utilities, computing and communication capabilities; second, the easy interaction with payment services and with the trusted storage for her presentation. We will observe that, apart for some details, almost all technologies and many of the services already exist at the present time. What lacks is the facility in their use and, although the scenario does not necessarily imply that, their interoperability and unification under a common framework.

The "Dimitrios" scenario is based on two elements: first, the very advanced digital avatars (D-Me) that can speak multiple languages and, more importantly, that can take decisions and interact naturally with other persons; second, the ease with which information transits the system: the senior person's D-Me contacts Dimitrios because he has the same heart condition, and Dimitrios's D-Me finds a child of the same age and situation with his own, for socializing and educational purposes. It is relevant here the way in which information is available to users that are interested in it, or that may take action as a response. The information is made available based on common context: similar heart condition, similar age and educational/financial situations. It is not specified how these services work internally, but, like in a sort of social networking, a distributed model may be applied.

The "Carmen" scenario includes a range of services that already exist to a certain extent: car pooling, internet shopping, smart fridges, traffic information, vehicle-to-vehicle communication. Again, it is the element of uniformity and facility that lacks in the present. There is no unified system that does all that.

Moreover, all systems that she uses seem to be centralized. It is worth noting that, in order for all people to use such a great number of centralized services, a powerful infrastructure is needed; or, a smart way to bring decentralized services closer to the users, and more related to their context.

Finally, the "Anette and Solomon" scenario is placed again in a farther future, as AmI features natural communication and advanced semantic processing capabilities, being able to "converse", "suggest", and even help the users with decisions and with a part of their work. More interesting features for a present implementation of AmI are identity checking, scheduling, and selecting information that is appropriate for the current context to make public for other users.

What we find interesting in the ISTAG scenarios, apart from the advanced human-machine interfaces, is the capability of the system to provide information and services just in time; also, a range of services that already exist now on the Internet, but that should be easier to access, in a more uniform way. The design of such an AmI system is not detailed and leaves us with many questions: how will the system support offering just-in-time services, in a very continuous and frequent manner, to the great majority of people on the planet, as the quantity of data will greatly surpass the requirements of today's Internet?; the services presented do not seem to have much in common – aggregating information coming from different services may lead to more and better capabilities; dependability of the system is also important: how can one assure a dependable service, that it is unlikely to fail and that will be truly ubiquitous, using all possibilities to offer the services to the user.

Other Relevant Scenarios

Satyanarayanan mentions two short scenarios [Satyanarayanan, 2001] that are more locality-oriented: in the first one, the AmI system (named Aura) notifies the user, while surfing the web at the airport, that the quality of the wireless network is much better at another gate, as at her current location there are many users accessing the Internet. One could argue that, if AmI is available for everyone, all will receive such a notification, so the system should consider the event in which all users will start moving toward the other gate. In the second scenario, the files of a user move with him, automatically, from his computer to his PDA, and from the PDA to the projection computer. It also allows voice editing of the user's slide presentation. During the presentation, information related to the attendees' emotional states is sensed and the system suggests the presenter to not present a certain "sensitive" slide. In these scenarios, context is important: the proximity between the user and available resources, the user's current activity that involves certain files and certain devices, the relationship between the user, his / her activity and the states of nearby people.

Banavar and Bernstein present a scenario [Banavar and Bernstein, 2002] that is focused on the seamless transfer of network connection and use of local peripheral devices (keyboard, screen) in order to assure a continuous video conversation across several situations, as well as comfortably working with personal files, using an interface in the car, on a PDA, or in a plane. This is

made possible by intelligent usage of available computing and communication resources and by just-in-time decisions.

Kindberg et al [Kindberg et al., 2002] emphasize the necessity for web-present objects, places and devices and the need to establish relations between these, according to the current context, usually in function of the user's activity. When the user Veronica arrives in a new city, her PDA automatically proposes links to interesting places to see. When she desires to communicate with her friend Harry, depending on his availability, a telephone call or an e-mail message are proposed. In an office building, she can easily connect to an available printer by pointing her PDA at it, and her PDA can also retrieve information about objects nearby that are tagged and have web-presence (i.e. feature a page on the web).

Vallée et al also describe a scenario [Vallée et al., 2005] in which screens and support for video-calling are located automatically. Here, some more about the internal functioning of the system is described: intelligent agents sense the context and decide on how to announce the phone call when there are people around, what to tell to the caller, and how the video-call is redirected to the room in which the receiver of the call moves to.

A somewhat similar scenario – but directed toward conflicting context information – is described by Bikakis and Antoniou [Bikakis and Antoniou, 2010]: a teacher that has finished the class earlier than scheduled remains in the classroom to check his email; his phone must decide if to ring or not: although the schedule indicates that the class is ongoing, the camera in the room can confirm that there is but one person in the room, therefore the class has actually ended.

Tracking users and providing them with useful information is also discussed in the scenario of Viterbo et al [Viterbo et al., 2008], as well as setting preferences in function of the current activity. One interesting aspect that is rarely discussed by other authors is the need for devices to be able, in the context of heterogeneous software and a distributed system, to align their ontologies – the semantic meaning that they assign for various terms that are used in communication.

There is another class of scenarios, that we will not discuss in detail, that concerns isolated environments: the smart home and the smart conference room are the most common examples. In the smart home projects (see for instance [Augusto and McCullagh, 2007, Bauchet et al., 2009]), various appliances, sensors and devices must be able to track the user, to assure a certain degree of automation, and, especially in the case of elderly and/or disabled people, to detect health disorder or other situations in which assistance may be needed. In the smart conference room [Johanson et al., 2002], challenges are related to the facilitation of file, control and image transfer between personal and public devices and screens. The difference between this type of local scenarios and the scenarios discussed earlier is that in these cases we are talking about a trusted environment and in which distribution is not absolutely necessary. One challenge here is the necessity to make heterogeneous devices interoperable [Hellenschmidt, 2005, Johanson et al., 2002].

Finally, we will refer the work of Bohn et al [Bohn et al., 2005] on Ambient Intelligence, that offers a complete perspective on the implications and concerns related to AmI, from several points of view: economy, privacy, reliability, ethics, social compatibility and acceptance. The authors also describe several interesting (albeit questionably ethical) scenarios: real-time shopping, information collection and shopping done automatically by smart products (silent commerce), perfect price discrimination and more advanced personalized schemes of payment, cross-marketing products – all in function of the user's context.

1.1.2 Features

The scenarios for Ambient Intelligence are primarily a source for what the features of AmI should be. But in the scenarios we can only see facets of these features, i.e. we see them only through the perspective of the user in its one interaction that is presented in the story. Let us look at some features of AmI more closely.

As Weiser puts it, "there is more information available at our fingertips during a walk in the woods than in any computer system, yet people find a walk in the woods relaxing and computers frustrating". For the user, Ambient intelligence must be like a familiar corner in nature: pervasive, natural, predictable. But in order to assist people, it must also be pro-active and intelligent. The features of AmI are also its greatest challenges. The most important feature of AmI is that it should be uniform, integrated, intuitive and familiar.

In the scenarios we see a lot of features that are related to interface (because that is what the user sees and feels): Weiser's intelligent windows, the intelligent refrigerator, speech recognition, even weather control. But while interfaces are very important, one must not forget that the information that they display and / or acquire are managed by a system that, even if is not (and should not be) visible, is very complex. We will focus in this section on looking into the features that are more interesting from this point of view.

AmI must be **ubiquitous**, **pervasive**. Its architecture must support a large number of mobile devices that incessantly share large quantities of information, without the user's knowledge (but not against the user's preferences). More than that, its model must be reliable: people will get used to it and will not be able to live normally without it; in order to be invisible, people must not notice it is there, but also they must not notice when it is not. These requirements call for a distributed, redundant system, like the Internet is today. Moreover, decentralization and locality are required by the fact that most of the generated information is not needed and does not make sense outside a certain domain of space, time and social relations (acquaintances). So AmI should be *distributed* and work at a *local* level.

AmI must be **natural**, by using advanced multi-modal, intuitive interfaces. That requires AmI to be *adaptive* and flexible: as any advanced technology, some users will choose to not use it at all, some will use it only for specific tasks, and for some it will mean an essential component of their lives. AmI

must adapt to all and only require the attention that the user is willing to invest. AmI must be predictable and transparent, being able to make the user understand why that information and services are there and how the system works in principle. This means that the basic principles that make AmI work should be *simple* and easy to understand by anyone and also make AmI *generic* and *adaptive*.

But AmI must also be **pro-active** and smart. It must take adequate action, without intruding. The action must be taken only if the user would understand the causality, and only if the user would approve the action. This is where *context-awareness* has a leading role. If the user has communicated many times with a friend, and both will be attending the same event, it is normal to automatically provide the friend with information that the user has on the event, because there is enough common context. But there is no sense in sending sensible user information to strangers on the street, except when this may be necessary in case of an emergency, when immediate action is imperative and privacy is not prevalent.

Related to the last to paragraphs is also the **traceability** of AmI reasoning. Traceability significantly improves user acceptance, as users are more tolerant of incorrect actions taken by context-aware applications if they are able to understand that they have a rational basis [Paymans et al., 2004, Henricksen and Indulska, 2006]. Of course that few users will actually be able to understand the complexity of a true AmI system, but the user should be offered with incrementally complex (and incrementally complete) explanations for the actions pro-actively taken by the system.

AmI must provide a **predictable**, natural flow of information. Like in social networks and shopping sites, one can assume that the user will be interested in things that are related to what he already knows, to what he does, and to the people that he is acquainted with. It is unlikely that someone is normally interested in something that bears no relation whatsoever with any part of his or her life. Useful information is information that is related to the context that the user is in. Relevance of information may be defined as *proximity* or *compatibility* between the context of the user and the context of the information (or the information itself).

As context is based on this sort of **locality**, this also solves the problem of information overload. The user can only do one thing at a time, be in only one place, only a number of past actions are still relevant and only a limited number of actions can be planned. So the context space of the user is limited and will only be related with a limited amount of information, that itself can be sorted according to its degree of relevance toward the current context.

There is one more very important issue related to context-awareness and AmI: the same AmI system will have to support **more than one user** at a time (as we usually see in the scenarios). In public spaces there are a great number of users that AmI has to be able to notify and to assist without them losing privacy (or important information) to other users. We will discuss these features in Section 1.3.3.

1.1.3 Applications

After presenting the scenarios and features, we should also take a quick look at the applications that have so far been developed for Ambient Intelligence.

Research in this domain is vast and varied. And that is normal, considering the complexity of what Ambient Intelligence should be. AmI literature and surveys [Cook et al., 2009, Augusto et al., 2010] show that there are several directions of development that are related to AmI: sensing, reasoning, acting, human-computer interaction, and privacy and security.

However, the same surveys show us that there is something missing. Authors notice that very little has been done in the field of reasoning [Cook et al., 2009, Augusto and McCullagh, 2007]. Although "intelligence" is a word that is part of the name of the domain, the smartness of the applications is very specific in most of the implementations, and cannot be used in a generic way in all AmI scenarios. Much more time goes into the development of attractive features and interfaces than in making those features be used in an intelligent manner.

Most leading applications try to cover all the essential aspects of AmI: sensing, recognition, human-computer interaction, in projects that address certain aspects of everyday life.

HP Cooltown is a framework that allows the association of web pages to physical objects, smart spaces, and people, thus making them *web-present* [Kindberg et al., 2002]. Whenever the user is in a certain place, a web page with information about that place will be available; whenever the user is near a web-present object, a simple scan with the mobile phone will bring the user to the object's URL.

Many AmI applications relate to Smart Homes. Building a Smart Home / Smart Apartment is relatively easier, as it has a low number of individuals that occupy it, which are known and whose habits are easier to learn; it occupies a clearly defined space. Features usually relate to the intelligent (i.e. balancing lower energy consumption with comfort) control of lights, heating, etc [Augusto and McCullagh, 2007]. Here we can refer projects like iDorm [Hagras et al., 2004], the SpacialAgent architecture [Satoh, 2004], the SodaPop model for home appliances [Hellenschmidt, 2005], the ACHE Adaptive Home Architecture [Mozer, 2005], and the projects CASAS and MUSE ([Crandall and Cook, 2009] and [Lyons et al., 2010], respectively).

Some applications – also relating to Smart Homes – are directed towards people with psychical disabilities (like dementia) and are suppose to assist them in leading a normal life and carry out their usual activities without the non-stop presence of a nurse [Bauchet et al., 2009, Perakis et al., 2009, Soler et al., 2010].

Other applications that use clearly-defined spaces are Smart Rooms, in which the features that prevail are related to the management of devices, display inputs, lights, and room configurations. Among these are the iRoom project [Johanson et al., 2002] and the SodaPop model featuring self-organizing de-

vices [Hellenschmidt and Kirste, 2004].

Finally, especially in the field of applications that use more generic representations for context information, there are some projects deployed in larger physical spaces: the Gaia project [Ranganathan and Campbell, 2003], AmbieAgents [Lech and Wienhofen, 2005] – deployed at the Oslo airport, and the MyCampus project, deployed on an actual campus [Sadeh et al., 2005].

In the projects that we have cited we can see that artificial intelligence is used (when it is used at all) for the recognition of images and of activity – therefore for the extraction of information. However, this information is not used in an intelligent manner afterwords, but it is fed to reactive systems that associate an action to an input.

It is this particular aspect that we are dealing with in this work. In order to discuss about a better application of AI in AmI, we have to create scenarios that rely more on AI (Section 1.3), and finally use more heavyweight AI components to enable true Ambient Intelligence (Sections 1.4.1 to 1.4.3).

1.1.4 Layers

Ambient Intelligence should be the third wave in computing. It can also be viewed as an evolution of what the Internet is today. Therefore it is expectable that it will be a very complex system, with components spanning many areas in software and hardware development. As the Internet is today, it is easy to organize the components of AmI into several layers (based on a presentation by El Fallah Seghrouchni [El Fallah Seghrouchni, 2008]).

- the **hardware** layer is composed of all the devices that are part of the AmI environment: sensors, actuators, controls (e.g. light switches), mobile phones, displays, laptops, computers, etc. The devices in the hardware layer are extremely heterogeneous from the point of view of computational and storage capacities, but all of them must feature some sort of connectivity.
- the **interconnectivity** layer allows connections between the devices in the hardware layer. It may use wired or wireless networks, and it may use a wide range of protocols: WiFi, Bluetooth, Infrared, GSM, etc. While a device may feature various types of network interfaces, connections may be unstable, so this layer should support seamless switching between connections (see the scenario by Banavar and Bernstein [Banavar and Bernstein, 2002]). This corresponds to the lower layers of the Internet: physical to transport.
- the **interoperability** layer is vital to AmI, in order for devices to be able to communicate freely, using uniform protocols above this level. There is little research in this area, as most projects use specific protocols and representations, however there are many issues that relate to interoperability, from compatibility of protocols to uniform representations for context [Perttunen et al., 2009] and ontology alignment [Viterbo et al., 2008].
- the **application** layer is the layer that most relates to AmI's "intel-

ligence". It offers services that have semantic awareness and that are adapted to the user's context. This is where different devices and applications collaborate in order to solve problems more efficiently. This layer itself may be divided into two sublayers: a lower one that is related to the **generic exchange of information** between entities, and one that performs **application-specific processing** to the information, feeding it either to the interface layer (and the user) or back to the sublayer below, so that it can be exchanged and delivered to other interested entities. It is this layer that our work focuses on.

- the **interface** is what the users "see" (i.e. perceive), and is an important part of all AmI scenarios. Gestures, speech, voice recognition, facial recognition, smart materials, glass interfaces and many other means of human-machine communication make AmI compatible with people that are not previously trained to use a computer and also make AmI environments more comfortable and intuitive to use. While there are important technological challenges in the hardware and HCI mechanisms related to the intuitiveness and the uniformity of interfaces, these interfaces can only appear as part of something intelligent if they offer relevant information, that is offered by the layer below.

We can group the features of Ambient Intelligence systems depending on their respective layers. This has been done in Table 1.1.

1.2 What Does Ambient Intelligence Really Mean?

Many scenarios for Ambient Intelligence are a mix of elements from several layers of Ambient Intelligence. Most times, they are focused on two of the layers: intelligent interfaces and context-aware services. Sometimes, issues of interoperability and connectivity are also mentioned, but much more rarely (though the challenges of these layers are not yet solved).

Mixing these elements may be confusing to the designer of an AmI system that needs to be implemented by a research team: as Ambient Intelligence is a vast domain, it is hard for the same team to develop connectivity features, context-awareness, the interoperability layer and new interfaces at the same time. That is why most implementations, while trying to tackle all these problems, result in a system that is not flexible or scalable, or that is specific to a certain application domain, and is by far not as impressive as the initial scenarios described.

The specific features that the scenarios introduce, as said in Section 1.1.2, mostly relate to two topics: intelligent interfaces and context-aware services. The interfaces are suppose to be more natural, and more intuitive for people to use. However, these interfaces are not fully specified, and it is hard to realize exactly how they would work: for instance, in Weiser's scenario, Sal uses a smart pen to circle a quote from the newspaper and the paragraph is sent to her computer at work – there is no mention how does the system know where to send the text: is there a particular pen for each destination? Does the pen have buttons for favorite destinations? How easy would it be for the user to

Layer	Features
hardware	• integration of sensors and actuators in everyday objects • a large number of devices of various shapes and sizes that allow easy touch-, voice- or gesture-based input, as well as large displays (see today's smartphones and tablets as Weiser's tabs and pads) • smart energy management and wireless battery charging
network	• pervasive wireless networks • pervasive (both indoor and outdoor) positioning systems (using satellite- or ground- based reference) • easy transition of connections from one access point to another • transparent transition between different protocols (GSM / WiFi / Bluetooth) and network infrastructures (centralized vs peer-to-peer)
interoperability	• uniform services working with interoperable representations • ontology alignment
application & intelligent services	• semantic- and context-aware exchange of information • detection of relevant information • recognition of current and anticipation of future situation and context
interface	• flexible interfaces usable with tiny, medium or large screens • visually rich and intuitive representations • support for multiple input methods for the same operations: gestures, voice, touch, etc. • support for hearing / visually impaired or otherwise disabled people

Table 1.1: Features extracted from the scenarios in Section 1.1.1 and the applications presented in Section 1.1.3. See also Section 1.1.2.

send the quote to a new destination? The same questions linked to the system specification can also be asked for some other scenarios as well. These are not necessarily questions related to the interface, but also to the semantic level of AmI.

Other questions that may arise are related to the flexibility and the scalability of the scenarios: in a scenario we see how the central character is using one specific feature in a way that seems intuitive enough. But how will the user be able to use a large number of services – as computers and the Internet allow him today – that may not be able to offer non-overlapping interfaces by means of the same, well known, devices. Also, it is very important to think about how will the services be offered to a large number of users, in a dependable and reliable way. These are questions that we discuss in the current section.

1.2.1 Scale

In an ideal future deployment, Ambient Intelligence would be a unified system that interconnects devices that are present in every object, in every material, in the whole world. It should capture information from at least tens of sensors for every cubic meter. The system should assist several billions of users continuously, in any situation, provide them with the appropriate information and actions with no delay, and even with a certain degree of anticipation. Its actions would be the result of aggregating and reasoning upon large quantities of information. The issue of scale is prevalent.

What AmI can take inspiration from is today's Internet, that interconnects billions of computers in a network that although unified, is neither centralized nor governed by a unique entity. The Internet uses *simple*, *layered* protocols that work over a distributed environment, that has only *local* central servers.

1.2.2 Preferences

As opposed to most mass-produced goods, IT has accustomed people to the possibility of customization of the interface and behavior of software, Internet pages, and their computer-related environment in general.

In an AmI environment, the computer is embedded in the environment, and therefore the user is able to customize the whole of his environment (among others, by means of actuators which are embedded in everything) – see the "Maria" scenario [Ducatel et al., 2001].

Respecting the preferences of a user is simple (in the limits of technical capabilities). However, the challenge with regard to respecting preferences arises when multiple users use the same environment: what if two users share the same room and one of them likes the air to be warm and the other likes it chilly? What if they like the room lit in different shades of color? What if two users that control a large display simultaneously (for instance because they are collaborating on a project) have different and conflicting sets of preferred representations?

1.2.3 Flow and the Disappearing Technology

Weiser [Weiser, 1995] gives several examples of "disappearing technologies", but probably the most compelling is writing: in present times, knowing how to read and write (still) is for most people a natural thing. AmI should make computer-enabled things natural presences in our environment.

There is also the question of "flow" (see Chapter 5 in [Riva et al., 2005]) and focus of attention. Handwriting does not require (for most people) an effort to do the actual writing – our focus is on the meaning of what we write. Computers make people focus their attention on the computer and the software, rather than on the task. By making the computer more natural and by em-

bedding it into the environment, people will be able to focus on their tasks again.

But making a technology "disappear" is difficult. Because it must never "appear" again. When we write, the handwriting never issues errors, and as long as one has the pen and paper one can do it. Computers show errors. Mobile phones run out of battery. Pro-active software can be annoying. These sort of events break the flow and change the focus of attention. For it to remain "invisible" the computer must never err, it must always have a backup solution, it must be redundant. The user must not notice the technology is there (or at least not think about it), but must not notice when it is not, either.

What AmI must also assure is minimal distraction. With so much information available, it is very easy to get distracted by potentially relevant news and notifications. A great challenge for AmI is that it must correctly manage the information provided to the user so that the user will always be able to remain focused on the ongoing task.

1.2.4 Anticipation

Being intelligent means being proactive. It means being able to recognize the situation and act on it before the user points out that action is needed. The opposite of proactive behavior is reactive behavior, which is characteristic of the current "stupid" technology: the software executes actions at the initiative of the user – like when the user issues a command, or pushes a button.

But proactive behavior needs anticipation (since the system must act *before* the user asks for it) of the user's needs. Coupled with *flow*, in order not to disrupt the user's attention the anticipated behavior must also be adequate, and, in a way, predictable.

Let us have an example: the user works at a desk, reading some documents; at some point the user notices something and needs to write something; the user reaches for the pen which is beyond the user's visual focus, but the user *knows* it is there; now, if anybody would have taken the pen out of a drawer and put the pen there, even if it would be more comfortable for the user, it would be unexpected and awkward, breaking the user's focus. The environment has to work in predictable, natural ways. And so must the Ambient Intelligence.

Predictability does not come immediately. Being natural is a property that is build in time and through tradition. For instance, that makes that computers will be much more natural for our children than for ourselves, not necessarily because the interface has changed (although it will), but because they were born in a world where computers were already everywhere.

1.2.5 A Word About Security, Privacy, and Interest

In a world where there are sensors in everything, that gather large quantities of information from the environment, that recognize our activity and trace our

history, it is easy to become worried with information being put to malevolent uses and in the hands of the wrong people.

The challenges related to the security and privacy of AmI have been raised by several authors [Bohn et al., 2005, Wright et al., 2008]. To our work, this is relevant from the point of view of questions related to interest and relevance: while a piece of information may be interesting for a certain party, is it that the said party should have access to that information?; also, if an entity (like a commercial store for instance) deems an announcement (an advertisement) as interesting for a certain individual (according to the user's profile), is it that the individual will indeed find that receiving that information is useful and not annoying?

Other issues relate to the social changes that will be brought by Ambient Intelligence. We can see today that without access to the Internet people's common-sense knowledge greatly decreases compared to what it was 50 years ago, and that people are so used to GPS navigation systems that most would certainly be lost without one. Considering the amount of useful information that can be brough by Ambient Intelligence to everyday life, how will this change our life and our behavior?

While these questions are not directly related to this work, it is important to keep them in mind when building any infrastructure dedicated to Ambient Intelligence.

1.3 Some New Scenarios

This section will present several new scenarios, that give an insight on how an Ambient Intelligence system may work internally. Details are focused on the application layer of the system, on how information is managed and how decisions are taken depending on context. Having this focus, we will consider that users are using today's hardware and connectivity, but devices will be enriched with AmI agents, that will form the application layer of the system.

1.3.1 Adaptability and Scalability

As we have seen in Section 1.1.2, AmI must be adaptive and scalable. In order to remain "invisible", it must be able to assist the user no matter what the conditions; also, it must be able to assist simultaneously a very large number of users, exchanging large quantity of information. These three scenarios have been presented in our IDC 2010 article [**Olaru** et al., 2010a].

> **Scenario 1.** A senior person walks on the street towards her house. In the pocket she has a mobile phone with an AmI software agent installed, featuring Bluetooth and GSM connectivity and in communication with a multipurpose sensor that monitors vital signs. As she does not like technology very much, the AmI

agent has been configured to communicate the least possible, so it normally does not connect with any other agents in the surroundings. The person lives in a small basement apartment. She climbs down the stairs, she misses one of the last steps and falls. She loses consciousness for a few moments.

In this short time, by means of the vital signs sensor, the AmI agent detects that the situation is not life threatening and no major injury occurred. However, care may be needed. There is no need for an ambulance, but there is a personal medical assistant that cares for this particular person and that should be called right away. This reasoning is done by a dedicated module of the AmI agent, that is especially designed for senior or disabled people. The medical nurse is reachable by phone, but there is no GSM signal at this location. The AmI agent searches for another resource that can offer communication services. It activates Bluetooth and finds a device that also runs an AmI agent. It provides to the other agent some information about the context: there is someone that needs urgent communication by phone. No personal details are provided. The other agent detects that this context fits its activity history: it has helped with this kind of actions before and it is even configured to do that without confirmation from the owner. It accepts the task. The agent of the senior person then gives to the other agent a number and a message to be sent.

While the senior becomes conscious again, the AmI agent receives, by means of another Bluetooth phone in the area, the confirmation from the nurse. She will arrive in just a few minutes.

Scenario 2. On the largest stadium of an European capital, a concert is going to be held, by one of the most popular rock groups of the time. Hundreds of thousands of people are participating. Most of them have mobile phones or smartphones which run AmI agents. Young people are more permeable to new technologies, and the agents are configured to communicate with other agents that share the same context, while keeping personal data private. At the concert, all participants share space-time coordinates, as well as the event that they are participating in. AmI agents form a temporary, anonymous social network, communicating not by means of the Internet or by GSM, but by local connectivity like Bluetooth or WiFi ad-hoc networking. They exchange, anonymously, interesting news or links that are related to the event and to the band. The users made that information public and are not necessarily aware of these exchanges, and will view the new data after the concert. Sometimes they exchange data intentionally, sending each other interesting links.

As the concerting band will be an hour late, the organizers send this information to the agents that manage the WiFi access points in the area. In turn, these agents disseminate the information to the devices connected to WiFi. The information is of great relevance to

the participants, so it spreads fast among the devices of the people on the stadium. In case other users that are not participating to the event received the information, their AmI agents will discard it because their users are not participating in the event, so the information is not relevant.

Finally, the concert begins. Towards the end, a pyrotechnic event causes a fire on the stage. For security reasons, the public must be evacuated. Panic breaks out. The GSM network soon becomes unavailable and the WiFi hotspots are overloaded. Special emergency devices connect to Bluetooth phones that are located near the exists and send them directions towards the exit. From device to device, the urgent information quickly reaches all participants. AmI agents are capable of calculating the relevance of received information according to the number of links it went through, and choose which exit would be closer.

A few days after the concert, a group of participants that shared, intentionally, a lot of images and links, but not any personal details or contact information, want to find each other again. By using the concert site and the fact that they shared so much, their AmI agents are capable of communicating again and the group can meet again.

Scenario 3. Marc is a researcher from France is on the trip towards the venue of a conference he is attending, currently during an intermediate 3-hour stop on the Athens airport. At the gate, he opens his laptop, on which several AmI agents are running. He marked on his schedule several activities connected to the conference and to this trip. One AmI agent, searching possible interesting data on the Internet, finds the following associations: from the attendants at the conference, one is from the same country – France. He has a public calendar on his website (and also a photo), that specifies a flight to the conference venue, in exactly the same time interval. There are no details on how to contact an AmI agent in relation with the other participant so no further details can be retrieved. Still the considerable amount of context awareness makes the AmI agent inform Marc of the findings including the name and the photo. Marc takes a look around and, indeed, spots the other researcher nearby. He goes to him and, politely, makes contact. The two researchers can now talk to pass the time to the flight and will be able to share a taxi ride from the destination airport to the conference venue.

1.3.2 Problem Solving

In many scenarios for AmI, the features that are presented are mostly about offering people the same kind of services that they are offered today, but in a manner that makes them easier to access. There are however some scenarios

that suggest that AmI could be more than that: that Artificial Intelligence may be integrated in AmI in such a way that assistance will go beyond accessibility, and will include solving of problems [Ducatel et al., 2001, Ramos et al., 2008]. Take the following scenario [**Olaru** et al., 2011]:

> Albert is a researcher in the field of Artificial Intelligence. His professional agenda contains, among other activities, attending the AI Conference, which is held in Paris, at the CNAM. Albert has already booked a flight from Heathrow airport to Charles de Gaulle airport, but has not yet thought about how to get from the airport to CNAM. Celia, another researcher in the same field, will also attend the conference. Besides booking a flight, she has also noted in her agenda that she will be taking a taxi from the airport to the venue of the conference (this reminder will help her later be prepared with money to pay for the taxi). Albert and Celia know each other, and the communication between their respective agents will help solve Albert's problem.
>
> Albert and Celia are both users of the AmIciTy Ambient Intelligence system. First, the system will detect the need for a means of transportation for Albert. Then, based on communication with Celia's agent and exchange of information with Celia's agenda, Albert's agent can suggest that a taxi may be an appropriate solution for Albert as well.

In this scenario we hardly make any reference to the hardware that is used, or to the interface. What is important is that the users are offered an intelligent service, that solves a problem for them. And this is indeed the purpose of AmI: to assist users in they daily lives, in a pro-active – but non-intrusive – manner.

A problem of the same type (not necessarily a scenario) is the following:

> A researcher is on the last day before the deadline for an important conference. As usual, writing the article has been left for the last moment, so it is critical for the researcher to finish the article before midnight. A colleague sends him a message regarding an interesting link that he found that relates to their research field. However, the AmI system decides that, *although the link is relevant to the field in general*, but not to the article, it will show a notification for the message only on the following day, knowing that it is *very common* that any disturbance on the last day before the deadline is badly received.

The scenario above is, again, not related in any way to hardware or interfaces. It is about the decisions that the system has to make, based on context and on previous experience.

1.3.3 Multiple Users and Collaboration

The great majority of scenarios for AmI have a central character that we follow through a series of instances in which he or she uses the features of the AmI system. But the users are not alone. In public spaces, all or most of the people around will also be users of the AmI system, and, for instance, a vocal advertisement that says the name of the user passing by (like in the movie Minority Report) may be heard by other users as well, producing a potential lose of privacy. However, users that know each other may be happy to be notified if they happen to be in the same shopping mall, at the same time. Context is everything. In our work (including working together with a team from the NII institute), we have developed some potentially interesting scenarios that deal with the existence of multiple users. The scenarios also include some more classic AmI features, that use existing devices from the environment, controlled intelligently, in order to assist the users.

> **Scenario 1.** On the floor of the laboratory two researchers Alice and Bob arrive almost simultaneously, with two different elevators situated at some walking distance one from the other. Alice and Bob are going to attend a research meeting for the Panda project, in room 42, which is somewhere between the two elevators. Both researchers have the meeting in their PDAs' agendas. They are at the laboratory for the first time and feel lost in the maze of corridors.
>
> When Alice goes out of the elevator she waits a little time and the lights near her dim, except from one light which is further down the hall, which burns more intensely it means that is the right direction. While approaching the intense light, the light dims and a light which is further indicates now the right direction. Alice then meets a group of students that pass on the hall in the other direction and all the lights return to normal. However, it happens that to her right, on the wall, there is a small display – she hears a short sound that grabs her attention and she reads on the screen "Turn right", together with an image of what she sees around her and an arrow indicating the way.
>
> When getting close to room 42, she sees Bob, who she doesn't previously know. But near them a screen on the wall lights up and displays "Panda meeting in room 42", and the light next to the door to room 42 blinks discreetly. At the same time, Trudy reaches the same area. She is a professor that works in a different department and has just opened a document on her PDA and tries to read the small writing – promptly she hears a sound that is specific to her notifications and sees a screen, different from the one Alice and Bob are using, displaying her document at an acceptable size.
>
> In the mean time, Alice and Bob enter the room, where, even if there was no one inside, the lights are already on and the room's projector displays the welcome message.

Scenario 2. Al, a member of the Panda team, working on the Panda 3 project[3], enters office EF301 that has been assigned to the team for the time this project is taking place. The EF301 office is one of the intelligent rooms of the university. The AmI system detects the movement and, based on previous observations, determines that there is on more person in the room. As it is customary, the lights grow a little brighter (the team usually works in semi-obscurity – it is their preference, so they can focus better on their work) so that other members of the team can see who has joined them and they are able to greet each other. Al gets seated at his computer and places his phone on the desk. By identifying the phone, and also the face of the user (the computer features a webcam) the system identifies Al and shows him his desktop (synchronized from the cloud), but only some windows are initially selected for display – those related to his work with the Panda team, and those he usually needs when he is in this office.

As the team works intensely, some feel warmer (although the temperature in the room remains constant) and they notify that to the system using gadgets on their screens. The system aggregates their preference and also considers the hierarchy in the team in order to calculate the new appropriate room temperature.

At a certain point, a notification tells them it is time to have a meeting. They all gather around the interactive table display, the lights go brighter, and audio recording of the discussion begins. Relevant information for the discussion are taken from the members' files and loaded into the interface of the table display. After the meeting, the members go home and all notes and modifications that were made during the meeting will be available in their personal workspaces when they will take on work again.

In the scenarios above we see two things: first, the use of devices that exist in the present day, used (many times working together) as intelligent, intuitive interfaces; second, the scenarios consider the existence of multiple users, and the changes that the AmI system needs to perform in order to adapt to that. *Preference aggregation* is one of the central features to this approach. Also the use of multiple means of notification, that are changed when the presence of other users requires it.

One issue that has been little discussed in the scenarios of this section is anticipation. Let us give an example of simple anticipative behavior that can make a huge difference in the life of users:

It is dark outside, and Celia is coming through the hall toward her office. She enters the office and the lights are already on. She knows that the lights are not on when she is not there: it is the AmI system that turns them on, anticipatively, before she enters

[3]"Panda" and "Panda 3" are fictitious names.

the office. When she will leave, the lights will go out right after she closes the door, but nobody will see that. People are already used to finding the lights on in all rooms they enter.

1.3.4 Reference Scenario

Finally, let us present the scenario that we will use for the examples and case studies in this work:

Scenario. Alice is a student in Computer Science. In the afternoon, she has a lecture on project management. Usually it is held in a classical amphitheater, but today she receives a message on her PDA that it will be held in a new laboratory in her university, called the SmartRoom. The message also contains indications on how to get there.

Alice is arriving to the campus by train. Due to traffic problems, the train will be 15 minutes late. Since the train is AmI-enabled, a notification pops up on Alice's PDA to tell her the train will be late.

In the mean time, the Professor that teaches the Computer Science course arrives in the SmartRoom. An application on his PDA allows him to see how many students are currently missing, and also what is their estimated time of arrival. Today it is only Alice that is missing, but the Professor is informed by the application that she is not expected to be more than 15 minutes late. Since she is one of the best students in the course, he decides to wait.

Since she is late, she is the last one of the 15 students to arrive. When she gets sited, all lights go down automatically, the presentation screen turns on showing the first slide of today's lecture, and the Professor starts the presentation. There are microphones for each student in the room, as well as for the teacher, but only the teacher's microphone is on for the time of the presentation. Later, during the time for questions, the students' microphone get activated as well.

The second part of the lecture is dedicated to some hands-on activity. The students are invited to choose among three activities in which to participate. Using their PDAs or their laptops, they can see their friends' preference for the activities, as well as choose the one they prefer. In the end, some groups are formed.

One activity is to read some descriptions of projects and try to present their strong and weak points. This activity only requires one or two large screens, so the SmartRoom allocates the screens for this activity, and the students move closer to them.

Another activity implies only a discussion between students, so they can sit at a table. The light above the table stays fully lit,

as the lights that are closer to the screens get dimmed so that the screens are more visible.

Alice participates in the third activity, which is to design a management strategy for a project. This activity is allocated to the two large touchscreens in the room. They get activated as the students assigned to the activity come close to the screens. As they work together, adding ideas to the content on the screens, Alice starts referring concepts from some previous work of hers, for another course. The SmartRoom, in collaboration with her personal agent, detects the compatibility and asks Alice if the whole file should be made visible for the other students. Alice agrees and the file is displayed on a third screen. A notification sound is played near the screen to draw the attention of the students.

At the end of the activities, the teacher passes by each group of students and evaluates their work, using an application on his PDA. Because the teacher has poor vision, the content on the screen is automatically magnified by the application when the teacher is viewing it. After evaluation and comments, the students' work is saved, and as they go back to their seats the screens turn off automatically. Before leaving, a message on the students' PDAs asks them to give anonymous feedback on the lecture. After everybody leaves the room, the lights go out all by themselves.

This scenario has been developed together with colleagues from Honiden-Lab in the NII Institute in Tokyo in order to be implemented in the Smart Room that has been built at the Honiden Lab.

The scenario is a balance between purely information-driven features and AmI features perceived as more common: we have detection and recognition of people, as well as large touch-based displays; we also have context recognition, collaborative work and usage of patterns.

1.4 Elements of the Research Approach

1.4.1 Multi-Agent Systems and the Agent-Oriented Paradigm

When dealing with AmI at its real future scale, it is clear that it must be distributed. The Internet that we know today is fully distributed in terms of domain-name services and routing, and for years Grid services and cloud computing are on the rise. All heavy-load web pages and services (search engines and the likes) are also distributed, sometimes using data centers around the world. As the future of what is now the Internet, Ambient Intelligence will reach a much higher number of devices and users. It is clear that for scalability (see Section 1.2.1) a more distributed architecture is necessary, and one that will also inherently support anticipation and context-awareness as a building-block function.

This is where the agent-oriented development paradigm comes in, and where the research in Multi-Agent Systems [Ferber, 1999] can contribute to the realization of AmI, because agents are the natural solution to the requirements of AmI.

Agents offer features that are very much needed by AmI, like reactivity, autonomy, pro-activity, reasoning or anticipation [Ramos et al., 2008]. In fact, agents offer the possibility of moving from the distributed computing paradigm – where the designer specifies the protocol and processing as seen from the global level – to a paradigm based on local reasoning and interaction, where agents are designed from the local point of view and the global behavior is emergent.

By being **autonomous**, the agents may function even if other components of the system fail or are unreachable, helping make the system invisible, and not disrupting the user's activity.

By being **anticipative** and **pro-active**, agents are able to recognize the conditions for certain situations, and are able to take action by their own decision, before the user requires it.

By forming an **open system**, new agents can join the system or be created and present agents can leave or be absorbed by other agents without radically affecting the system.

By using **local behavior**, agents can focus on the current context of the user and also depend less on the existence of centralized control.

The idea of using agents for the implementation of Ambient Intelligence is not new. However, there have been many approaches to doing that. Considering the layered perspective presented in Section 1.1.4, most of these approaches use agents, indeed, at the level of the "intelligent" layer. One of the approaches uses a small number of agents that retain user preferences and query context information, but this approach is not very generic or scalable. The other approach is focused on the coordination and self-organization of agents, but the individual agents do not have a flexible knowledge and context representation.

In this work, we are trying to develop a system that will both scale but also be able to work with more advanced (but nevertheless flexible) knowledge representation. However, we will not try to develop components for all of AmI's layers. Instead, we will focus on the application layer, and on the exchange of information between agents.

1.4.2 Context-Awareness

It is difficult to talk about Ambient Intelligence without mentioning context-awareness. Many systems with applications in Ambient Intelligence implement context-awareness as one of their core features (see Table 2.1). In previous work in the field of context-awareness there are usually two points of focus: one is the architecture for capturing context information; the other is the

modeling of context information and how to reason about it.

Context is any information about entities that are relevant to the interaction between the user and the system [Dey, 2001]. Context-awareness is the ability of a system to autonomously adapt to the current context, in order to provide a better response and experience for the user [Viterbo et al., 2008].

Context information may be categorized in several groups. For instance, depending on the nature of the context, there are four types of context information [Chen and Kotz, 2000]: computational context – available resources, network quality and related information; user context – profile of the user, people nearby, social situation; physical context – lighting, temperature, traffic conditions; time context – time of day, date of the year. Modeling of context information uses representations that range from tuples to logical, case-based and ontological representations (see the surveys of [Perttunen et al., 2009, Strang and Linnhoff-Popien, 2004]).

But beside this type of information that is "exterior" to the system, context may also be characterized by the user's activity [Henricksen and Indulska, 2006], and information about current focus [Brézillon and Brézillon, 2007].

In our work we have used an approach to context-awareness that involves two aspects: first, the implicit modeling of context by using a hierarchical structure for agents, that is mapped against the different types of contexts that are considered by the system (see Chapter 4 and Section 5.3); second, a graph-based representation of context information that, by means of graph matching, allows identifying the relevant information, detecting the situation and solving problems related to missing information (see Section 5.2).

1.4.3 Adding an Aspect of Self-Organization

Considering the amount of communicating devices in a real-scale AmI scenario, one may be tempted to look in the domain of self-organization for solutions on how such a system may achieve a certain goal. Emergence and self-organization [Heylighen, 1989, Heylighen, 2002] provide the means to obtain complex properties out of a large number of interacting simple individuals. The possibility to obtain novel, non-additive effects of causal interactions made flourish research in both natural and artificial systems with emergent behavior. In order to model and design artificial systems, inspiration was taken either from self-organizing inanimate complex systems, or from complex systems of simple living beings (like ants, wasps, spiders, etc). This is because these types of individuals are easier to understand and model.

A self-organizing system is a system that, as a form of adaptation to external conditions, achieves a state of dynamical equilibrium that is characterized by a certain level of organization; organization is achieved without any external or centralized control. The advantages offered by self-organizing systems are many, relating to adaptability and flexibility on the one hand, and to robustness, fault tolerance and self-healing, on the other. These properties result from the fact that organization is not imposed to the individuals from above,

but it is the behavior of the individuals that contains a natural tendency towards the organized state – the organization *emerges* from the level below [Shalizi, 2001].

In our work we have used self-organization mechanisms to provide the system with emergent behavior. This way, although agents have a local behavior in which they only communicate with their neighbors and only have local knowledge, the information that they exchange can reach agents that are much farther, but can still be controlled in a context-aware manner (see Chapter 3 and Section 5.4).

1.5 Setting the Goals for This Research[4]

In the framework of this layered perspective, as the state of the art presented before suggests, many challenges in AmI lie at the level of the fourth layer – ambient applications and services. These challenges relate to the fact that, in a real-scale AmI environment, huge quantities of information will be generated throughout the system. Part of this information is private or meant only for certain users; part of it is relevant and / or interesting only in certain context, or to certain people. Moreover, AmI is mainly formed of devices that are not reliable from the point of view of the system: they may go in and out of the network coverage, they may shutdown or wake up without notice, and they may use heterogeneous hardware and different (in type, quality and cost) ways to connect to the network. Considering these conditions, the way that information should move through the system and how it should used and modified by the individual devices is particularly challenging, and it is a central concern of any AmI environment that aspires to be reliable and trustworthy.

This research tries to answer some of these challenges, by **developing a multi-agent system for an AmI system's application layer**. The proposed solution relies on several key features: **system distribution** – a reliable AmI environment must feature distributed control, so that its functioning will not be vitally affected by the disappearance or temporary unavailability of any device in the environment; the use of **cognitive software agents** that have a flexible behavior depending on the capabilities of the device they are executing on; the use of **self-organization mechanisms** to offer the means to coordinate a large number of agents and obtain, by means of local interaction and without the need for centralized control, global properties at the level of the system; **context-awareness** as core feature of the multi-agent system, affecting both the structure of the agent system and the behavior of agents in order to offer an optimal experience.

The research goals fulfilled by this research are the following:

- to develop a multi-agent system model for Ambient Intelligence that features self-organization, context-awareness and anticipation;
- to develop several scenarios that emphasize the requirements of real-scale

[4] As specified by the PhD Thesis proposal [Olaru, 2010].

Ambient Intelligence environments;
- to develop a simulation testbed that implements the elements of the said scenarios, to serve for experiments with AmI platforms;
- to implement and experiment with the developed model, using the simulation testbed, in order to prove the model's validity as a component of an Ambient Intelligence environment.

The result of the second goal is presented in Section 1.3. We will see in Chapters 3-6 how the other goals have been fulfilled.

Chapter 2

A State of the Art in Related Fields of Study

Now that the goals of this research are clear, this chapter examines work that is related to our research: the application of software agents and multi agent systems in the implementation of Ambient Intelligence; context-awareness and the representation of context; and finally self-organization in cognitive agent systems.

The review of agent-based AmI environments (Section 2.1) looks into how agents can be used for AmI, what features they provide and what AmI layer(s) they compose, and whether they are used more like individual, autonomous, reasoning entities or more like connected parts of a distributed whole.

Since context-awareness is a defining feature of AmI, in Section 2.2 we review the current representations of context information, as well as how context information is retrieved and exchanged among agents. We emphasize the works on the representation of context as associations and on the use of ontologies.

The last section of this chapter (2.3) discusses self-organization and mechanisms that can be used to coordinate distributed entities without centralized control, especially in the context of using cognitive agents.

2.1 Agent-Based AmI Environments

In the field of agent-based Ambient Intelligence platforms there are two main directions of development: one concerning agents oriented towards assisting the user, featuring learning and complex reasoning, and using centralized components (knowledge repositories, ontologies, etc); and one concerning the coordination of agents associated to devices, and potentially their mobility, in a context of more simple functionality, also considering distributed control and fault tolerance.

2.1.1 Using Few, Complex Agents

The first approach of using MAS for AmI is closer to Intelligent User Interfaces and local anticipation of user intentions, coming from the field of intelligent personal assistants. In this approach, agents are complex, use learning algorithms, and use centralized components to retrieve external data. Inter-agent communication is scarce, except for when the central components are designated as one or more agents.

For instance, embedded agents form an AmI environment in the iDorm implementation [Hagras et al., 2004]. Agents are used here to manage the diverse equipment in a dormitory, resulting in the control of light, temperature, etc. They learn the habits of the user and rules by which to manage those parameters. The system does not require the attention of the user, except for those moments where the user is unhappy with the system's decision and overrides the controls. This way the system learns and, in time, becomes invisible to the user. The organization of the system is fairly simple, and its main component is a central agent associated with the building.

EasyMeeting [Chen et al., 2004] is an agent-based system for the management of a "smart" meeting room. It is centralized, and it manages all devices in the room by means of reasoning on appropriate action. It is based on the Context Broker Architecture (CoBrA) and uses the SOUPA ontology.

MyCampus [Sadeh et al., 2005] is a much more complex system, in which agents retain bases of various knowledge about their users, in what the authors call an e-Wallet. There are also agents associated to public or semi-public services (e.g. printers). The e-Wallet manages issues related to security and privacy. It represents knowledge using OWL and accesses resources as Web Services. The e-Wallet provides context-aware services to the user and learns the user's preferences. Other components of the system are the Platform Manager and the User Interaction Manager, that offer directory and authentication services in a semi-centralized way.

The ASK-IT project [Spanoudakis and Moraitis, 2006] uses agents for the assistance of elderly and impaired persons. It uses the FIPA PTA (Personal Travel Assistance) architecture. There are several types of agents that have different specialization: information retrieval, environment configuration, user monitoring, service provision, etc. The structure and functions are however quite rigid, and there is little adaptation or flexibility of the system's features.

DALICA [Costantini et al., 2008] is a multi-agent system that uses location data for the dissemination of information about cultural assets. It can monitor visitors and also monitor the transportation of said assets. It features an interesting architecture that combines continuous Galileo positioning with the use of ontologies and user profiles.

Tapia et al present [Tapia et al., 2010] a few projects related to healthcare and support for elderly / mentally disabled people. All projects use multi-agent systems: ALZ-MAS [Corchado et al., 2008] is a centralized system that follows the person throughout daily activities using RFID tags and the ZigBee

protocol; Telemonitoring homecare [Alonso et al., 2009] is an experimental architecture that interconnects heterogeneous wireless sensor networks and sends monitoring information to a central monitoring central through an RSS feed; finally, Fusion@ [Tapia et al., 2009, Tapia et al., 2010] relies on a decentralized architecture in which agents search for services to provide to the users (it is SOA-based), and a special kind of low footprint agents – *Interface agents* – reside on the users' mobile devices.

2.1.2 Using Many, Simple Agents

The second approach to agent-based AmI platforms concerns solving different issues like user mobility, distributed control, self-organization and fault tolerance, having a more global perspective on how an AmI platform should function.

The SpacialAgents platform [Satoh, 2004] is a very interesting architecture that employs mobile agents to offer functionality on the user's devices. Basically, whenever a device (supposedly held and used by a user), which is also an agent host, enters a place that offers certain capabilities, a Location Information Server (LIS) sends a mobile agent to execute on the device and offer the respective services. When the agent host moves away, the agent returns to the server. Sensing the movement of agent hosts in relation with LISs is done by use of RFID tags. The architecture is scalable, but there is no orientation towards more advanced knowledge representation or context-awareness, however it remains very interesting from the point of view of mobile agents that offer new capabilities.

The LAICA project [Cabri et al., 2005] brings good arguments for relying on agents in the implementation of AmI. It considers various types of agents, some that may be very simple, but still act in an agent-like fashion. The authors, also having experience in the field of self-organization, state a very important idea: there is no need for the individual components to be "intelligent", but it is the whole environment that, by means of coordination, collaboration and organization, must be perceived by the user as intelligent. The work is very interesting as it brings into discussion important issues like scalability, throughput, delegation of tasks and a middleware that only facilitates interaction, in order to enable subsequent peer-to-peer contact. The application is directed towards generic processing of data, which is done many times in a fairly centralized manner. The structure and behavior of agents is not well explained, as their role in the system is quite reduced – the middleware itself is not an agent. However, the architecture of the system remains very interesting.

The AmbieAgents infrastructure [Lech and Wienhofen, 2005] is proposed as a scalable solution for mobile, context-ware information services. There are three types of agents: Context Agents manages context information, considering privacy issues; Content Agents receive anonymized context information and execute queries in order to receive information that is relevant in the given context; Recommender Agents use more advanced reasoning and ontologies in

order to perform more specific queries. The structure of the agents is fixed and their roles are set. Although it may prove effective in pre-programmed scenarios, the system is not very flexible.

The CAMPUS framework [El Fallah Seghrouchni et al., 2008] considers issues like different types of contexts [Chen and Kotz, 2000] and decentralized control. It uses separate layers for different parts of an AmI system: *context provisioning* is close to the hardware, providing information on device resources and location, as well as handling service discovery for services available at the current location; *communication and coordination* manages loading and unloading agents, directory services, ACL messaging and semantic mediation, by using the Campus ontology; *ambient services* form the upper layer, that agents can use in order to offer other services in turn. The architecture is distributed, having only few centralized components, like the directory service and the ontology.

There are other resembling proposals for AmI middleware that do not explicitly employ agents. Hellenschmidt et al [Hellenschmidt and Kirste, 2004, Hellenschmidt, 2005] propose a generic topology and a self-organising middleware for ambient intelligence (called SodaPop), aimed at coordinating appliances. The devices are not controlled by agents, but by *SodaPop Daemons* that share many features with agents, like reactivity, negotiation capabilities, and a certain degree of autonomy. Each appliance is modelled as having a user interface, an interpreter, a control application and several actuators. Between these units there are three channels, respectively: the events channel, the goals channel and the action channel. The middleware puts these channels in common and introduces negotiation and conflict resolution, so that, for instance, as a result of user input on a device, the controller on another device can action the first device together with a third device. The architecture is very interesting, however scalability is not brought into discussion.

A framework that deals with low-footprint agents for Ambient Intelligence is Agent Factory Micro Edition (AFME) [Muldoon et al., 2006]. It uses Java for mobile devices and LEAP and is FIPA-compliant. The work is focused on an architecture for agents, which are cognitive and have a modular structure that allows interchangeability of perception, affect and reasoning components.

2.1.3 A Survey of Existing Applications and Projects

We have summarized some features that are relevant to our work, as they are manifested by the systems that we have reviewed above, in Table 2.1. It easy to observe that different agent system consider different aspects of Ambient Intelligence and adopt different approaches to their implementation – for instance regarding centralization of the system. It is also worth noting that few of the system address only the problem of the middleware, and many of them are trying to propose a complete architecture, from the sensing level to the user interface.

In our work, we are trying to focus on only one layer of an Ambient Intelligence environment, and use agents only for what they are good at: reasoning, au-

Project Name	knowledge representation	ontologies	context-awareness	learning	security - privacy	mobile agents	scalability	flexibility	centralized	FIPA-compliance
iDorm	-	-	-	Yes	-	-	?	-	Yes	-
Spatial Agents	-	-	-	Yes	Yes	Yes	?	-	Yes	-
EasyMeeting	Ont.	SOUPA	Yes	-	Yes	-	?	Yes	Yes	Yes
SodaPop	-	-	-	-	-	-	Yes	Yes	No	-
LAICA	-	-	-	-	-	-	Yes	Yes	partial	-
MyCampus	CBR	Yes	Yes	Yes	Yes	-	Yes	Yes	Yes	some
AmbieAgents	CBR	Yes	Yes	-	Yes	-	Yes	-	partial	Yes
ASK-IT	-	some	Yes	some	-	-	-	No	Yes	Yes
CAMPUS	Ont.	Yes	Yes	some	-	-	Yes	Yes	No	Yes
Dalica	tuples	Yes	-	-	-	-	-	-	partial	-
ALZ-MAS	-	-	some	-	Yes	-	?	some	Yes	-
Fusion@	-	-	some	-	-	-	Yes	Yes	No	-
AFME	-	-	-	-	Yes	?	Yes	Yes	-	Yes

Table 2.1: Features of the agent-based systems described in Sections 2.1.1 to 2.1.3: manner of knowledge representation; use of ontologies; implementation of context-awareness; learning capabilities; consideration of security and privacy-awareness; use of mobile agents; support for scalability; flexibility of the architecture; centralized vs decentralized system; compliance with FIPA protocols.

tonomy, proactivity. We assume that the information can be provided by the layers below, and that interfacing with the user can be done in the layer above – we believe that applying a layered structure (which is done by some of the mentioned architectures [El Fallah Seghrouchni et al., 2008, Sadeh et al., 2005]) is

a better way to deal with the design of such a complex system as a flexible, generic Ambient Intelligence environment.

There are many other projects of Ambient Intelligence, that do not use agent-oriented implementations. These are usually more directed towards specific applications, and the communication between entities, as well as the elements of reasoning, follow fixed, specific protocols. Among projects like these we can primarily cite projects related to the Smart Home – the ACHE Adaptive Home Architecture [Mozer, 2005], and the projects CASAS [Crandall and Cook, 2009] and MUSE [Lyons et al., 2010] – or to the care of elderly / mentally disabled people – for instance the Archipel, ALADDIN and PERSONA projects [Bauchet et al., 2009, Perakis et al., 2009, Soler et al., 2010].

2.2 Context-Awareness

Context-awareness is a central issue in the field of Ambient Intelligence. Proactive, but non-intrusive behavior would not be possible without a proper understanding from the side of the AmI system of the user's context. Actions of the system must appear to be natural and well integrated in the current situation.

Context has been defined as "any information that can be used to characterize the situation of entities (i.e. a person, a place or an object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves"[1] [Dey, 2001]. Therefore, context-awareness is not only the ability to adapt the system's reaction to the current situation, but also to decide the action to be taken by looking at the user's context.

Many authors consider context as relating almost exclusively to location, time, and other instantaneous properties of the physical environment. But there is more to context than that. First, there are more types of context – e.g. computational, temporal, user-related [Chen and Kotz, 2000]. Second, context is not only formed by the properties of said context types – like where the user is located, what time it is, what is the temperature outside and what capabilities the current network connection has – context is also defined by associations between various facts that relate to the user, facts which are not necessarily contextual information of the said types. For instance, it would probably be unwise to disturb a researcher with unimportant messages on the last day before a conference's deadline. While this decision is context-aware, it is something that does not relate almost at all to any properties of the physical, computational or social environment of the user, nor to its profile or personalization options. In our work, it is this type of context-awareness that we are trying to implement, that is based on the detection of associations and similarity between various pieces of knowledge.

In the following section we will look at how context-awareness is implemented

[1] This is just one of the existing definitions for context, but it is most likely the most popular.

in various AmI-related projects. Next, we will give details about how context is represented in the literature, with a focus on AmI applications (Section 2.2.2). As for the context-aware behavior of agents we use the matching of context graphs and patterns, in the final section we will briefly introduce some algorithms for graph matching, with a focus on the matching of labeled graphs.

2.2.1 Context-Awareness in AmI

Research discussing context-awareness for Ambient Intelligence applications generally revolves around two issues: on the one hand, the infrastructure for capturing and processing context information (where context information is of physical nature); on the other hand, the modeling of and the reasoning on context information.

The proposed infrastructures for processing context information usually contain several layers [Baldauf et al., 2007]: sensors capture information from the environment; there is a layer for the preprocessing of that information; a layer for its storage and management; and finally the layer of the application that uses the context information.

One of the early infrastructures of this type is the Aware Home project [Dey et al., 1999]. It uses *Context Widgets* as an abstraction for the sensors that capture context information of various types. There are cases when data needs to be reinterpreted, and that is done by *Context Interpreters*. Data from multiple widgets can be aggregated by *Context Servers*. An application can get its context information from Widgets or from Servers, and it can also use Interpreters.

Harter et al [Harter et al., 2002] present a very interesting context-awareness infrastructure that uses *Bat units* to detect the position and orientation of people and objects, used for *Follow-me* applications. However, it only deals with spatial elements.

Other architectures for the provisioning of context information use middleware for the processing of the data (e.g. the LAICA or MUSIC projects [Cabri et al., 2005, Kirsch-Pinheiro et al., 2008]) or context servers, that store the context information and make it available by means of queries (for instance projects like MyCampus [Sadeh et al., 2005] or AmbieAgents [Lech and Wienhofen, 2005]).

This type of infrastructures is useful when the context information comes from the environment and refers to environmental conditions like location and weather, or to health-care related sensed information. Indeed, several surveys of context-aware applications [Chen and Kotz, 2000, Baldauf et al., 2007, Perttunen et al., 2009] show that location is the most used type of context.

However, physical context is only one aspect of context [Chen and Kotz, 2000]. There are projects that consider other types of context – e.g. the MIMOSA middleware [Malandrino et al., 2010] or the CML model for context information [Henricksen and Indulska, 2006] – but they are relatively few and quite

specific.

Moreover, the infrastructures are usually centralized, using context servers that are queried to obtain relevant or useful context information. In our approach we attempt to build an agent-based infrastructure that is decentralized, in which each agent has knowledge about the context of its user, and the main aspect of context-awareness is based on associations between different pieces of context information. All agents should have a more or less equal role in the transfer of context information.

There is also a strong directionality that characterizes these infrastructures: context information flows from the sensors toward the application level, with various intermediary components that allow for additional processing. But more often than not, the applications themselves are unable to create context information themselves, and insert it into the system. In an AmI environment where devices interact with the user, they use context information, but also obtain information that can be used in turn by other users.

2.2.2 Context Representation

Context can be modeled in various manners, leading to a great range of representations for context information, that have been surveyed repeatedly [Bolchini et al., 2007, Perttunen et al., 2009].

Most simple representations use key-value pairs or tuples to retain the measures for various aspects of context, like, for instance, the position of the user. For example Feng et al [Feng et al., 2004] propose that context should be represented as an n-dimensional vector of values from different domains (scalars, strings, sets).

Ontology-based representations

Logic-based formalisms use mechanisms that come from Artificial Intelligence / Knowledge Representation. Prominently used among these are description logics and ontologies [Bettini et al., 2010]. Although ontologies allow for the representations of complex concepts and constraints, they are not adequate for the representation of dynamic context.

Moreover, ontological representations are not appropriate for resource constrained devices. One solution is to keep ontologies in centralized servers and query the servers to obtain the information. Another solution that has been studied [Preuveneers and Berbers, 2008] is to efficiently encode ontological information for less powerful systems.

On a related note, there has been a significant body of work in the domain of ontology alignment, which is vital for a viable implementation of Ambient Intelligence systems [Viterbo et al., 2008, Laera et al., 2007]. However, this is not the subject of this work. We assume that all agents in the system work with terms from the same ontology (where it is the case), or that ontologies have already been aligned.

Associations

The *situation* is the physical, social and cultural space (context) in which the activity is carried out (see Chapter 2 in [Riva et al., 2005]). In case of pseudo-simultaneous activities, the situation is characterized also by the other activities that are being carried out at the same time. We can consider the situation as being defined by what concepts are related to the user at the present time, and how they are related.

Henricksen et al use several types of associations as well as rule-based reasoning to take appropriate decisions depending on [Henricksen et al., 2002, Henricksen and Indulska, 2006, Bettini et al., 2010]. A graphical graph-like representation is presented, that can also be serialized into XML-based files [Robinson et al., 2007]. The model – called CML, or Context Modeling Language – is based on the Object-Role Model. The associations are of types like *engaged in* – between a person and an activity; *located at* – between a person or a device and a Location; *permitted to use* – between a person and a device; etc. What is important is that relations are also annotated with their origin (e.g. sensed, profiled, derived, etc) and an indication of quality.

In associative context representations we can also include the Awareness Marks model [Hervás et al., 2011], in which the traces of interaction between the user and the application remain in the system, in order to provide the user with improved, adapted services in the future. The *awareness mark* created in the interaction between a user and an application or a device can be used in the future to improve the interaction between the same user and application, or between another user and the same application (*cross-user interaction*), the same user and another application (*cross-object interaction*), or even a different user and application (*cross-element interaction*).

These two models are very close to our own work, because they use associations and even graph-like representations. However, the two models are not very flexible, or not very powerful, respectively: in CML, the types of relations / associations are predefined, and reasoning is based on rules; in Awareness Marks, it is not mentioned how uniform the *mark* representation is. In our work we define a more flexible (albeit more loose) model, that uses the same representation across the system and that is more adapted to the dynamic aspect of context.

Uncertainty

Several surveys focus on dealing with the uncertainty and inaccuracy of sensed or otherwise obtained context information [Strang and Linnhoff-Popien, 2004, Schmidt, 2006, Bettini et al., 2010]. These conditions occur because the information may have become obsolete (or stale) – for fast-changing context, because the sensor is not accurate – like with GPS positioning data, because the methods to indirectly obtain context information may not be accurate, or because different methods are used to obtain aggregated / processed information

– making the results inconsistent.

Conflicting context information – i.e. different components of the system reporting conditions that may lead to opposed actions – is discussed by Bikakis and Antoniou [Bikakis and Antoniou, 2010], that propose the use of defeasible logic [Nute, 2001], that allows conflicting rules and exceptions.

2.2.3 Context and Situation Recognition

In AmI, the vital features of pro-activity and anticipation are based on the ability of the system to recognize the user's context – or situation – and to act accordingly. There are two aspects of recognition: on the one hand, recognition of patterns and action, without a formal model for context, done directly by the sensing entities; on the other hand, recognition of situation based on a representation of context information, where producing the context information and recognizing the situation can be done by separate entities.

In the first case, we are talking about some clear kinds of situation recognition, involving patterns in the user's perceivable actions: driving behavior, movement patterns, medication patterns, patterns in the use of resources, body stance, facial recognition. These patterns are recognized by entities that are close to the perceiving end of the system, and the information about the recognized events is then passed forward, or acted upon directly (also see Section 4 in the survey by Cook et al [Cook et al., 2009]). These types of recognition refer to fairly basic notions (albeit not easy to recognize), from which primary context information is obtained.

In the second case, the recognition is done at a higher, more abstract level, and is based on recognizing patterns in the context information already extracted by other elements of the system. In the great majority of cases, the link between situation and action is made by means of rules. Condition of the rules specify certain value for tuple components; or a logical relation between propositions defining the context; or the existence of certain predefined relations, with high certainty, between elements of the context; or a certain result of reasoning upon the context information, for ontology-based representations [Bettini et al., 2010]. What is to be noted as prevalent in these approaches to recognition is that most times the result of the recognition is not fed back into the system – it therefore cannot be used directly by other entities in the system.

In our approach, an important part of the recognition happens midway between sensing and application-specific behavior, the representation for context is uniform, and the results of the recognition are disseminated and can be used by other entities in the system in the same way (and represented in the same way) as primary context information.

2.2.4 Graph Matching for Context-Awareness

The representation for context information that we propose in this work (see Chapter 5) is based on graphs and on the matching of graphs representing the user's context with graphs representing possible situations, a match meaning the user is in that situation.

Our approach to context representation is rooted in existing knowledge representation methods like semantic networks, concept maps and conceptual graphs [Novak and Cañas, 2006, Sowa, 2000]. These structures can be used to describe situations (and context) in a more flexible manner and using less memory than ontological representations.

This approach is related to the CML model in the sense that it too uses a graph-like structure [Henricksen and Indulska, 2006], but in our approach the relations and concepts to not have relevance by themselves, but because they match a preexisting pattern.

As new information – represented by a graph – is assembled with existing information, our approach also relates to the K-MORPH framework, that works with aggregating different pieces of information – called *knowledge artifacts* – in order to solve problems, discarding artifacts that are not useful [Hussain and Abidi, 2009]. This approach is particularly interesting with respect to our work, although the authors do not use it in relation to Ambient Intelligence or to any distributed environment.

Graph matching is also used by the MUSIC project to compute context similarity [Kirsch-Pinheiro et al., 2008]. However, the MUSIC project computes similarity based on the properties of concepts that compose the graph; our approach is based on the structural similarity of the graph, not on the properties of concepts.

Algorithms for Graph Matching

Since we use context similarity, i.e. graph matching, for the recognition of situation, we will also briefly mention some works in the field of graph matching.

There are two main fields of research in which graph matching and similarity are used [Conte et al., 2004]. One is the detection of the elements in an image, or the similarity of images. The other is ontology (or schema) alignment.

In image recognition, graphs are used to represent the vicinity between various elements obtained as a result of image segmentation. The purpose is to detect features in an image, like for instance human faces [Leung et al., 1995]. Edges and nodes are not attributed / labeled. Nodes represent the elements of the image, and an edge between two nodes means that, in the image, the two elements are neighbors. Matching can be exact or inexact. Among the most important algorithms for matching of unlabeled graphs are tree-search algorithms [Ullmann, 1976, Cordella et al., 1998, Cordella et al., 2004] and algorithms for the matching of a graph against a library of graphs [Messmer and Bunke, 1999, Messmer and Bunke, 2000]. Some algorithms, especially those for inexact matching [Bengoetxea et al., 2002], are based on powerful mathematical in-

struments – like expectation maximization [Luo and Hancock, 2001], graduated assignment [Gold and Rangarajan, 1996], and learning of assignment coefficients [Caetano et al., 2009].

Ontology / schema alignment or mapping, on the other hand, use labeled graphs in which nodes represent the elements of the schema and edges represent the relations. Most times it is nodes – i.e. concepts – that must be matched, in order to find the equivalence between concepts from different vocabularies or ontologies [Gašević and Hatala, 2005, Hu et al., 2005]. While the fact that nodes and edges are attributed brings this type of matching closer to our work, the purpose of these algorithms is to match different vocabularies; our purpose is to match graph patterns against graphs using the same vocabulary, which in a way is closer to the purpose of image recognition.

2.3 Emergence and Self-Organizing Agents

Whenever a large number of entities interact intensively and form feedback loops, they form a complex system. Certainly, an ideal deployment of Ambient Intelligence could be considered as one such system, containing a large number of devices continuously exchanging information and adapting to the environment.

Complex systems are difficult to control, because it is hard to predict the outcomes of actions, as they are shaped by the intricate feedback loops in the system. Complex systems are also characterized by emergent properties and phenomena, which are unexpected with respect to the design of the individual elements.

However, appropriate design of the entities and tweaking of their properties may actually lead to intended results at the level of the whole system, and may be used to control the system not by means of centralized or hierarchical components, but by means of the very feedback loops that produce emergence. The resulting complex systems also feature fault-tolerance and robustness in the face of system-level damage, as well as adaptation features.

The notion of distributed control is what makes emergence and self-organizing systems interesting with respect to our work, as using self-organization mechanisms may help in obtaining a distributed and robust deployment of AmI.

2.3.1 Definitions

What we know is that emergence appears in the context of complex systems – composed of a large number of interacting entities [Amaral and Ottino, 2004]. Emergence needs two levels of perspective: the inferior, or micro level of the individual entities and the superior, or macro level of the whole system. A simple definition is that "emergence is the concept of some new phenomenon arising in a system that wasn't in the system's specification to start with" [Standish, 2001]. A more elaborated definition is that "a system exhibits

emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel with respect to the individual parts of the system" [De Wolf and Holvoet, 2005]. An "emergent" is a notion that can represent a property, a structure or a behavior that results from emergence.

The essence of emergence is not actually the novelty or the unexpectedness of the emergent – as these will fade at later experiments although the emergents will stay the same – but the difference between the description of the individual and the description of the emergent, from the point of view of an observer [Standish, 2001, Randles et al., 2006]. If the minimal description of the individual is taken, it cannot be used to describe the emergents resulting from the system, therefore the emergent is considered as novel and, potentially, unexpected.

Some features of emergence can be particularly emphasized [Goldstein, 1999, Heylighen, 2002, De Wolf and Holvoet, 2005]:

- emergence occurs out of the interactions between the parts of a complex system;
- emergence is defined in relation with two levels – it is manifested at the higher level, arising from interactions at the lower level;
- the representation of the emergents cannot be reduced to the specification of the individuals;
- emergents only arise after a certain time in which the system has evolved;
- once they occurred, emergents will maintain identity over time;
- emergents arise without any centralized or exterior control;
- the emergent phenomenon is robust and flexible, i.e. it is not influenced by damage on the system (even if the emergent is temporarily not observable, it will arise again as the system evolves).

2.3.2 Using Reactive Agents

In the field of multi-agents systems, most examples that demonstrate emergent properties use reactive agents [Serugendo et al., 2006]. This is because they are easier to implement and control, and they are suitable for small devices with low computational power. But, more than anything, systems of reactive agents are easier to predict and to design so that they will manifest self-organization or other emergent properties. Notable examples of emergents in reactive agent systems relate to the formation of a certain geometrical or geometry-related structure or behavior. Self-organization is reached by mechanisms that are usually inspired by nature and the behavior of animal societies (generally insects) [Mano et al., 2006].

For example, "smart dust" uses forces of attraction and repulsion and in some cases simple leader election algorithms to dispose individuals in a multi-level circular layout [Beurier et al., 2002], or in a ring or lobed shape, that is resilient to structural damage [Mamei et al., 2004, Zambonelli et al., 2004]; "spider" agents roam through the pixels of an image and use stigmergy to mark the different areas of the image [Bourjot et al., 2003]; local behavior can

be used for the gathering of resources in a single area [Randles et al., 2007], for the foraging of food or the transportation of loads [Unsal and Bay, 1994]; in a very interesting example of dynamic self-organized behavior, agents establish, by means of emergent coordination, traffic directions through narrow corridors for the transportation of resources between two areas [Picard, 2005].

2.3.3 The Advantages of Cognitive Features

Although reactive agent systems may be very useful, there are many advantages that a cognitive agent has over a reactive agent. First, it is proactive. Even if there are no signals, perceptions or stimuli from the environment, a cognitive agent may act by itself, taking action according to its objectives. Second, a cognitive agent is aware of its situation and may reason about it. It is aware of what it is supposed to fulfill as final goal and is capable of making plans and taking action towards the realization of its goal. The cognitive agent can use its experience and information about the environment and the consequences of past actions to develop a better plan every time a similar situation occurs.

Ricci et al. support the concept of *cognitive stigmergy* [Ricci et al., 2007], that is inspired from natural, reactive, self-organization (stigmergy, introduced by Grassé [Grasse, 1959]) but is using cognitive agents and a more complex environment. The environment is composed of *artifacts* that agents are using and in relation to which they make annotations. The usage of shared resources in the environment by one agent driven by local goals influences the other agents that have access to the resource, leading to feedback and eventually to organization. The research shows encouraging results in applying the concepts from self-organizing reactive architectures to cognitive agent systems.

Other studies of emergent behavior in cognitive agents exist. Emergence of norms in a social game is possible through *social learning* (learning of society-imposed norms) [Hales and Edmonds, 2003, Sen and Airiau, 2007]. Gleizes et al. show that local cooperative behavior of cognitive agents can lead, by means of emergence, to better system-wide results [Gleizes et al., 1999]. The study is continued, using the AMAS (adaptive multi-agent system) technology as a framework that facilitates emergent functions based on cooperation [Capera et al., 2003].

However, the use of cognitive agents in self-organizing systems is still very reduced. While, in the present, numerous real-life applications of self-organization exist (see the book of Prokopenko [Prokopenko, 2008]), they are based on reactive behavior of the individual entities. It is true that these systems are easier to verify and control, and it is also true that even in reactive agent systems it is hard to find a good balance between explicitly designed behavior and implicit, emergent behavior [Prokopenko, 2007], but the features of reactive systems are limited by the absence of reasoning in agents.

2.3.4 Design Guidelines and Methodologies

Besides taking inspiration from biological systems, there is also another approach to the design of self-organization, that is used mostly in wireless sensor networks. WSNs use simple individual units and, more importantly, need to consume little power. This approach is also many times validated formally, not only by simulation. It consists of different algorithms used for routing, resource allocation, structure formation, synchronization, power conservation and resilience [Dolev and Tzachar, 2009, Mills, 2007]. Bernadas et al. demonstrate communication services based on this type of self-organizing algorithms, with individual agents having a more complex internal behavior and manifesting features like pro-activity and a certain amount of reasoning [Bernadas et al., 2008]. A model for a self-organizing system for communication systems is proposed by Marinescu et al., using cognitive agents that have local goals and specific available actions. The model is verified formally and relevant results are shown [Marinescu et al., 2008]. However, these models address particular concerns in the functionality of a system and are not very flexible. It is a considerable problem to build a model that will address more than one concern at a time, and in a generic manner [Marinescu et al., 2008, Mills, 2007].

In this work, it is not our intent to obtain a self-organizing system *per se*. Instead, we want to use mechanisms that are used in self-organizing systems to obtain distributed control and fault tolerance. These mechanisms are based on feedback loops, and on behavior of the individual entities that generates a positive reaction for actions that are likely to help the system move toward a desired state. They are also based on intense communication, and on an element of randomness to assure variation and to get the system out of a local optimum state [**Olaru** and Florea, 2009, **Olaru** et al., 2009b].

Chapter 3

Agent Behavior: Relying on Self-Organization[1]

In building a Multi-Agent System for Ambient Intelligence, the first aspect that we have focused on was agent behavior. That is, how should agents exchange information so that interesting information reaches interested agents, without centralized control?

We see an Ambient Intelligence environment as a large number of devices that serve the needs of their respective users. The devices are mostly going to deal with information: delivering relevant information to interested users, aggregating, filtering and reasoning about information – being "information conveyors" [Weiser, 1993]. The problem that we asked is: given a certain piece of information, how to deliver that piece of information to the interested agents – the agents that can use that information to help the users? This is a problem that is addressed to the intelligent services layer of an AmI environment.

3.1 Approach

The design of the MAS that is presented in this chapter started from the following idea: at realistic scale, an AmI system will have to deal with a very large number of users and an even greater number of devices that communicate between each other. The system was conceived bearing in mind the goals presented in Chapter 1 (especially Section 1.5): use of cognitive software agents and context-awareness, decentralization, local behavior, possibility of deployment on devices with different capabilities. Obtaining a coherent global result (at the level of the whole agent system) is done by using mechanisms of self-organization.

Why agents? Ambient Intelligence environments need to be distributed as much as possible in order to be able to deal with the loads imposed by

[1] The results of the work presented in this chapter have been published in several papers between 2009 and 2011 [**Olaru** and Florea, 2009, **Olaru** et al., 2010c, **Olaru** et al., 2010b, **Olaru** and Gratie, 2010].

the quantity of information that will be exchanged. Agents are one of the paradigms that can be used for the implementation of distributed computing. Moreover, their qualities – such as autonomy and pro-activity – make them especially appropriate for the implementation of AmI [Ramos et al., 2008].

Why self-organization? A large portion of the devices in AmI environments will be smart sensors and actuators, which will have a minimal processing power and storage capacity. But that doesn't mean that they must necessarily be controlled by a centralized entity. In order for AmI to be "intelligent", there is no need for the individual entities that it consists of to be intelligent [Cabri et al., 2005]. Moreover, self-organization allows a system to be robust and fault-tolerant [Heylighen, 2002], which are features very much needed by AmI, which will have to be dependable and adaptive to changing conditions. By means of techniques that are used in self-organizing systems, we have obtained properties at the global level of the system by using local interaction and knowledge.

Why cognitive agents? Because cognitive does not necessarily mean very complex. Agents working with small knowledge bases can be much more useful than purely reactive agents [Marinescu et al., 2008]. This project was also an experiment to see what kind of emergent properties can be obtained in a system formed of cognitive agents (as opposed to the more popular reactive agents) [**Olaru** and Florea, 2009, **Olaru** et al., 2010c].

Why local behavior? First, it will be very difficult to manage all information in an AmI environment in a centralized way, or even in a hierarchical structure. Second, there will be no need too. It is very likely that users will only be interested in information that is related to something close to them – close in location, time, activity or social relations. We are not supporting locality in terms of only location, but also of time, acquaintances and computing resources.

Context-awareness? Context-awareness is an essential component of any AmI environment. We see context as vicinity in a domain that considers space, time, social relationships, computing resources and actions / intentions.

Following the results of may experiments and the study of previous work in the field of self-organization, we have devised the following principle for obtaining coherent information sharing at the level of the system, by using only local agent interaction and knowledge: **An agent should send the pieces of information it is interested in to neighbors that the agent believes may be interested in those pieces of information.**

Determining if a piece of information is interesting (or relevant) to an agent should be done based on context information. Neighborhood relations between agents – i.e. the topology of the system – should also be defined according to context. We will discuss these issues in Chapters 4 and 5.

In this chapter we will discuss details on the implementation of the agent behavior, according to the principle above, in the experimental platform called AmIciTy:Mi , in which we have focused on tweaking agent behavior, and in

which the topology of the system and the context representation have been simplified: the topology of the system is based only on spatial relations (proximity between the positions of agents); context is represented by four simple numerical measures that are meant to control the flow of information through the system.

3.2 AmIciTy:Mi

The application layer of AmI can also be seen as a middleware: assuring the context-aware exchange of information, using the hardware and network to transmit the information, and delivering the relevant information to the application-specific components and the to the interface. We have named this middleware AmIciTy:Mi , as a part of the AmIciTy Ambient Intelligence environment that we are in the process of building. AmIciTy:Mi was implemented together Cristian Gratie[2].

The AmIciTy middleware is being developed keeping in mind the scale of a real Ambient Intelligence scenario. In such a situation, there is a very large number of users and (possibly "intelligent") devices: sensors, actuators and more advanced human-machine interfaces. These devices communicate permanently and exchange a very large quantity of information, coming from all the sensor perceptions, the users themselves, and from information aggregation. To complicate the problem even more, most of the devices that are used have limited storage and processing capacity.

This is why the middleware is completely distributed. Each agent in the middleware is assigned to and is executed on a device. There might be more agents that are connected to the same device, especially if they handle different functions. Agents will communicate directly only with the agents assigned to devices in a certain vicinity. Communication may be done by means of a wired network but most communication between personal devices will be done wirelessly.

From the perspective of the devices, the middleware is only accessible by means of the agents that run on the device. The general view of the system from this perspective is shown in Figure 3.1. The device's interface communicates with the software agent(s) on the device, by means of a uniform data structure, packing the sent data into such a structure, and unpacking received data from the structure. When the agent receives new information from the exterior or from the device, it reasons about this information and, if it considers that adequate or necessary, it sends the information to other agents in its vicinity.

It is important to point out that AmIciTy:Mi does not exist as a separate entity, but is an entity that is formed by the totality of the agents composing it.

[2]Cristian Gratie is a colleague and fellow PhD student at University Politehnica of Bucharest. His work was especially directed towards (but not limited to) the development of the format and processing of scenario files, and towards the implementation of some of the visualization and logging tools.

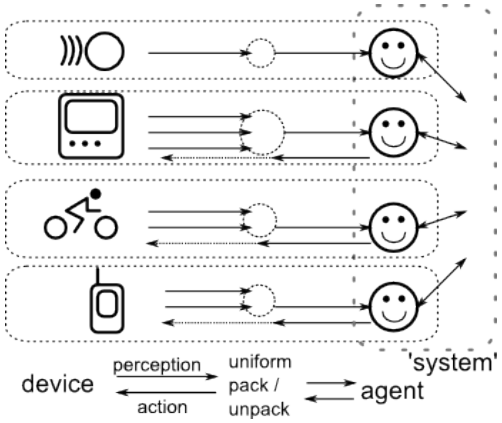


Figure 3.1: The structure of the middleware, as seen from the perspective of the devices. The "system", i.e. the middleware, is actually formed of the agents that compose it. There is a packing / unpacking step in the interface - agent communication so that the communication will be uniform over all the middleware.

3.2.1 Context Measures

By context we understand the conditions in which an event occurs and that are related to the event [Dey and Abowd, 2000, Chen and Kotz, 2000]. Context-awareness is the feature of a system (here, an AmI system) that makes the system behave differently depending on these conditions.

Since AmIciTy:Mi was focused on agent behavior and controlling information sharing across the system, context representation was kept simple. The objective was to be able to control different aspects of the flow of information through the system by means of some simple and generic numerical context measures.

Four context measures were used, one of them being handled implicitly and the other three being represented by means of simple numeric values. First, space is inherently considered, because of the structure of the system, that relies on local behavior and communication. Second, temporal context is implemented as a period of validity for each piece of information. Third, each piece of information is related to certain domains of interest. Last, each piece of information carries a direct indication of its relevance.

A more detailed description of these aspects of context-awareness, together with their influence on how information is shared and spread through the system is presented below:

Local behavior and interaction – leads to inherent location awareness. New information will first reach the agents in the area where the information was created (e.g. where the event took place). Depending on the other aspects of context-awareness, the information will only stay in the area or will spread further. Also, when all other measures are equal, agents will give less relevance to information related to a farther location.

Time persistence – shows for how long the information is relevant. When

its validity expires, the agents start discarding the piece of information.

Specialty – shows how the information relates to some *domains of interest*. In time, agents form their own notion of specialty depending on the information that they have. New information is considered more relevant if it is more similar to the agent's specialty, and agents share relevant information first, and they share it with agents that are more likely to consider it relevant. This influences the direction in which information is spread.

Pressure – shows how important it is for the information to spread quickly. Pressure translates into higher relevance and the agent will treat the information with higher priority. Also, the higher the pressure, the more neighbors the agent will send the information to. This way, pressure controls how quickly the information spreads.

Context compatibility, or *relevance* of new information is computed as a function of the measures of context associated with the new information and of the context of the agent, comprised of an indication of specialty. In order to be able to aggregate and compare the different measures, all are quantified and bounded, and their ranges are all scaled to the interval [0, 1]:

- locality has an explicit quantification as the distance to the source of the event: $Dist \in [0, 1]$, with 0 meaning it refers to *this* agent and asymptotically growing to 1 for longer distances;
- persistence: $Pers \in [0, 1]$, with 1 meaning the information is valid forever, and 0 meaning it has expired;
- specialty is a vector $Spec \in [0, 1] \times \cdots \times [0, 1]$ in which each component shows the degree of relatedness to a certain domain of interest, and $||Spec|| \leq 1$;
- pressure is also a value $Pres \in [0, 1]$.

When computing the relevance of new information, distance, persistence and pressure are introduced directly in the computation of relevance. Specialty is compared against the specialty of the agent. Similarity between the two is computed as follows:

$$Similarity = 1 - \sqrt{\frac{\sum(S1_i - S2_i)^2}{n\ domains\ of\ interest}} sin\alpha, \quad \alpha = arccos(\frac{S1 \cdot S2}{|S1||S2|})$$

where $S1$ and $S2$ are the two specialty vectors, and the sum is for all domains of interest. The formula has been chosen in order to give lower similarity to vectors that are at greater angle (different specialties) but also to give higher similarity when one vector is less specialized than the other.

Relevance is computed as

$$Rel = \frac{Dist + Pers + Pres + Similarity}{4}, Rel \in [0, 1]$$

This allows for different types of important facts – a fact can be equally important if it has high pressure, or if it is of great interest to the agent (similar to its specialty).

Pressure, specialty and persistence are associated with the pieces of information by the source of the information – which is outside the system. The goal

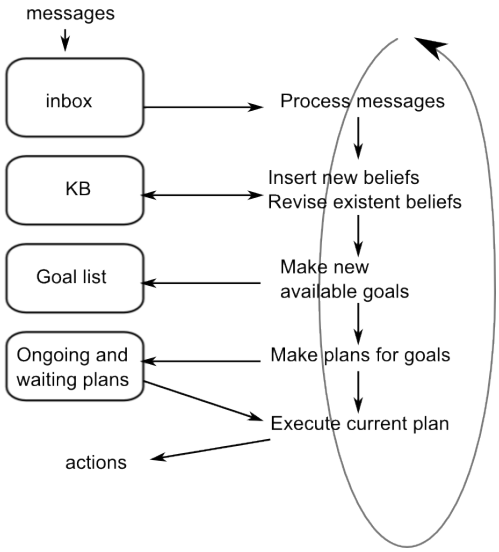


Figure 3.2: The basic execution cycle of an agent.

of this research was to see how the spread of information can be controlled by means of this measures.

These measures are associated with pieces of information present in the system, but agents themselves also feature indications of the context that they are in, namely their specialty and their pressure. The agent's pressure is computed as a weighted mean of the pressure of the facts that it knows, giving more weight to the facts with higher pressure. The specialty of an agent is initially a null vector, and it is updated with a certain factor depending on the specialties of facts that the agent knows. This way, the specialty of an agent "adapts" to the information that is currently being shared in the surroundings of the agent.

3.2.2 Agent Design

Agents in AmIciTy:Mi have been designed so that they are simple, flexible, and so that an agent with the same structure can run both on the simple processor of a sensor and on a powerful computer. The agents are cognitive, and their model is inspired by the BDI model of agency [Rao and Georgeff, 1995]. In our experiments, particular attention has been given to agents that hold very small knowledge bases and that would be suited for very small devices like sensors.

In the design of the agents, inspiration was also taken from the human behavior and thinking. As the quantity of information that will pass through an agent's knowledge base over time is quite large and the agent will be unable to (and it would probably be useless to) store it all, the agent must be able to sort its knowledge according to its relevance, and it must be able to "forget" information that is of no more use or of insufficient relevance.

The general structure and behavior of the agent is presented in Figure 3.2.

Each agent has a message inbox, a knowledge base (KB), a list of available goals and a list of current plans.

The information in the agent's knowledge base is stored in *Facts*, where Facts are tuples of the form

$$\langle Agent,\ knows,\ Fact \rangle$$

Note that the definition is recursive. In prototype phase, the system did not study a real-life application. Therefore, facts that would normally represent useful information coming from the environment are replaced with Facts containing a *DataContent* placeholder, that has an identifier for tracing Facts relating to that information.

This structure allows the agent to hold information about what it knows but also about what other agents know. This is how an agent can compute the specialty of neighbor agents.

In the presented experiments we have used very limited maximum sizes for the knowledge bases of agents, to show that the agents need very little storage capacity in order to manifest context-aware behavior. In applications where different types of devices are involved, agents may have knowledge bases of different sizes.

3.2.3 Agent Behavior

The behavior of the agent has been designed so that it would be flexible and adaptive to context. It must be able to work on a very limited machine and also be able to process more information if working on a more powerful computer. The general behavior of the agents is quite common for cognitive agents: in its execution cycle, the agent processes messages that arrived in its inbox, integrates the new knowledge, then chooses a goal to make plans for, it makes a plan, and then executes one or more actions from the current ongoing plan. This cycle is presented in Figure 3.2. More details on the behavior are presented below.

At the beginning of each cycle the agent checks the messages in the inbox, by integrating facts in the knowledge base, if they are new. The agent also infers that the sender knows the fact, which contributes to the agent's knowledge about its neighbors.

In the next phase the agent forms a list of potential goals. There are two types of goals that an agent can have: *Inform* other agents of some information or *Free* some storage capacity. Each goal is assigned an importance, and the most important goal will be chosen as an intention. The reason for which we consider *Free* as a goal of the agent is to have a uniform representation of all actions that an agent can perform, that are related to its knowledge or to its communication with other agents. The importance of *Inform* goals is computed according to the context measures of the corresponding fact:

- pressure is in the interval [0, 1] and is used directly;

- similarity between the fact's specialty and the agent's specialty is computed based on the distance between the two specialty vectors and on the angle between them – the calculus has two effects: a larger angle means less similarity; however, if one of the vectors is significantly smaller in module (much less specialized), similarity will be higher. The result is a number in the interval [0, 1];
- recursive depth of the fact (facts that refer to farther agents are less important), normed in the interval [0, 1].

Importance is computed as the mean value of the three components, allowing for different types of important facts – a fact can be equally important if it has high pressure, or if it is of great interest to the agent (similar to its specialty).

Importance for the *Free* goal is calculated depending on how full the agent's storage capacity is, reaching a value of 1 (highest importance) when the knowledge base consumes all available capacity. The agent must always have some capacity free for new facts that come from other agents.

After choosing a goal, the agent makes a plan for it. For *Free* goals, the agent decides what facts to discard. For *Inform* goals, the agent decides what neighbors to inform of the corresponding fact. The number of neighbors to inform is directly related to the pressure of the fact. Agents are chosen according to their estimated specialty, calculated as a mean specialty of the facts that the agent knows the neighbor has. After creating the plan, the agent places it in a queue of ongoing plans. At each cycle the agent will execute one action in its current plan. Once a plan is completed the agent moves to execute the next plan. It is possible however to promote plans corresponding to more important goals to the top of the queue, so that urgent actions will be performed first.

Formally, we can consider that an agent A is defined as a tuple

$$A = \langle KB, S_A, P_A, Goals, Plans \rangle$$

An agent can receive and send messages $m \in M$ and has the following functions:

$learn : M \times S_A \times P_A \times KB \to KB$ – the agent integrates the information from a message in the knowledge base;

$update\_specialty : KB \times S_A \to S_A$

$update\_pressure : KB \times P_A \to P_A$

$deliberate : KB \times Goals \to Goals$ – update goals;

$plan : Goals \times Plans \to Plans$ – creating a new plan does not modify the other plans, but may change the order of the queue of plans;

$act : Plans \times KB \to M \times KB$ – the result of a plan may be an update to the KB or a message that is sent.

The behavior of the agent changes depending on its context measures. Specialty directly affects the relevance that is associated with various facts. Higher relevance associated to facts makes them better candidates for *inform* messages sent to other agents, and lower relevance makes facts better candidates for removal (or "forgetting").

3.3 Evaluation

The AmIciTy:Mi proof-of-concept implementation focused on obtaining an emergent, coherent, distribution of information across the agent system, while keeping a very low quantity of knowledge in individual agents.

3.3.1 Platform Details

The proof-of-concept prototype was implemented in Java, with support for placing the agents in a grid structure or at random positions, with direct communication only among adjacent agents – in the grid – or between agents within a certain range – for agents at random positions.

Experiments with 950 to 1000 agents were run, focused on observing characteristics of the spreading of data through the agent system, such as: the speed of the spreading, the coverage reached by each data piece and the particular areas preferred for spreading. The outputs of the experiments show how these characteristics are linked with the measures of context assigned to the information.

The implementation was focused on speed – so that more experiments could be carried out. The platform did not use multiple execution threads, but rather the agents were executed sequentially: at each "step" of global time, the agents' algorithm was executed for one step, one agent after the other. Messages sent in one step of the global time were only made available to agents in the next step.

The simple agent behavior and this method of execution meant that between 3 and 5 steps of global time could be executed in one second, on a Intel Core 2 Duo 2.5GHz processor.

3.3.2 Desired Outcome

The desired outcome of experiments was to be able to control the distribution of information in the agent system by means of the context measures.

The most important context measure is Specialty. The intention is to have the information characterized by a certain specialty reach the agents in the system that have similar specialty.

But temporal context is also important: some information expires more quickly than other, so our intention is to be able to control how long the pieces of information remain in the system, by means of persistence.

Also related to context is the fact that some information should reach agents with more urgency. This is what pressure is for, and the intention is to be able to control the speed with which the information spreads by means of pressure.

3.3.3 Experimental Scenarios

As with any distributed system that is based on emergent behavior, a great number of experiments were needed in order to observe and tune the system. For this purpose, scenarios needed to be used, with different parameters and containing many events.

In all experiments we have used specialties that relate to three domains of interest. This is not a limitation of the model or of the implementation, it is just meant to allow for a better visualization of the results. Thus, we consider the specialty vector describing the data or the interest of the agent as a color, with each basic color corresponding to one of the three domains. We called the three domains A, B and C.

In order to allow easier specification of scenarios, we have used XML files to characterize the scenarios in a simple and effective manner. The XML files use special tags designed to allow the specification of complex scenarios. For example, generating 15 instances of some data and placing them at random positions in the system is performed by the following XML snippet:

```
<event type = "inject" pressure="0.1" persistence=".05">
     <event.time min="0" max="150" count="15" dev="2" />
     <event.domain a="1" b="0" c="0" />
     <event.location.x min="0" max="30" dev="2" for-each="time"/>
     <event.location.y min="0" max="30" select="1"
                       for-each="location.x,time" />
</event>
```

That is, an "inject" event is characterized by its time – which in the example is between steps 0 and 150, by the domain of the inserted data, and by the location – x and y, spanning all the grid. There are 15 moments in time at which events will happen, for each moment there is an x location, for each time and x location there is a y location. The features of the system (number and position of agents) are also given in the scenario XML.

Random values are specified by the interval for the value, and the deviation (the `dev` attribute in the XML file). All random events are generated using a seed, also given in the scenario XML. When running the simulation, a new seed can be generated, or the previous seed can be used, leading to the exact same scenario (with the same random values) as before, therefore allowing reproduction.

The scenarios that were studied follow the following pattern:

- insert several pieces of information into the system, with different specialties and let them spread until their maximum coverage is reached and certain areas of interest are established in the system;
- insert "test" pieces of information, of different specialties, and observe how they spread according to previously established areas of interest, while the interest of each agent suffers only small changes; this stage also shows how old (and less relevant) information is forgotten by the

agents as they receive the new data;
- insert one or two pieces of information of no particular specialty (equally related to all domains of interest) but with very high pressure, and observe if and how fast they reach all the agents in the system

Using the type of scenarios specified above, we can verify the objectives of our system: that the direction of the information spread is controlled by specialty; that higher pressure means information spreads faster; that persistence controls how long does information remain in the system.

3.3.4 System Monitoring and Visualization

The behavior of complex systems is extremely difficult to observe in simple graphical representations and the study of the system's behavior means, most times, the minute observation of the activity log of each agent. This is the reason why some visualization tools have been developed.

There are two types of graphical outputs: distribution representations and graph representations. The distribution representation is a 2-dimensional representation of the agent system, showing one dot (or one cell) for each agent, placed at its respective location and having a color that indicates different things depending on the type of the representation (see for instance Figures 3.3, 3.4 or 3.7). Depending on the particular type of the distribution, there are:

- fact distributions – the cell is visible if the respective agent has that data and the color of the cell depends on the specialty of the data (see Figure 3.3 (left), 3.5 and 3.6);
- per-domain interest distributions – the cell is visible if the agent is interested in that domain, the hue depends on the domain and the intensity of the color depends on the degree of interest (see Figure 3.4 (b), (c), (d));
- global interest distributions – the cell shows the specialty of the agent, the color of which is taken directly from the specialty vector (see Figure 3.3 (right) and Figure 3.4 (a));
- agent balance distribution – the color intensity in the cell shows the balance of the agent (see Figure 3.8 (a) - (d)).

Graphs representations are value-time graphs that show the evolution of a particular value over time. The value is the result of the aggregation of a certain characteristic over the whole agent system, taking either the maximum value or the mean value across the system. The numeric representation of the instantaneous value is also displayed under the graph. An example of a graph representation can be found in Figure 3.8 (e).

When running the platform, an automatic window layout feature places the relevant distribution and graph representation across the screen, so that the system can be easily followed through its evolution.

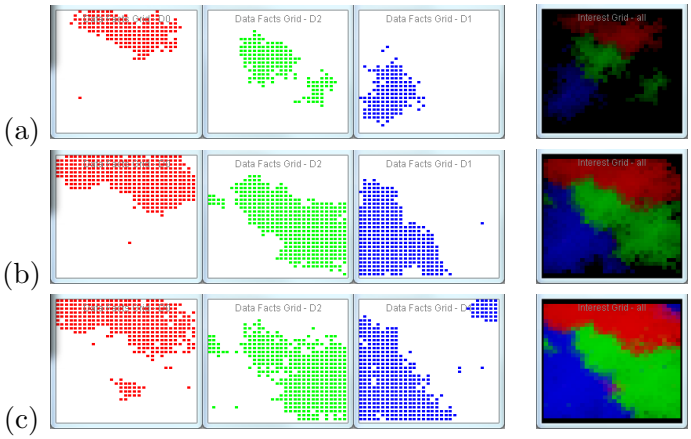


Figure 3.3: The spread of three data pieces (left) and the evolution of agent interests (right) at simulation step: (a) 31; (b) 60; (c) 130. The corresponding data, from left to right, are related to domains A, B, C.

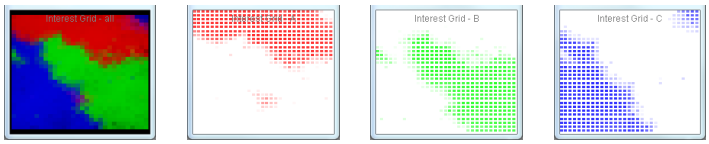


Figure 3.4: The agent interests at step 130: (a) global, (b) for domain A, (c) for domain B, (d) for domain C.

3.3.5 Tweaking the Parameters

The behavior of the agents was built and tweaked so that it will contain mechanisms enabling self-organization of the system. More precisely, there are feedback loops that are created: the agent state is influenced by the information it receives, and the decisions of what information it disseminates are influenced by its state; moreover, the agent will receive information that it itself has disseminated in the past.

One of the problems that a badly tweaked agent system suffers from is message or goal overload. If the agents send too many messages, and there is a goal generated for each received messages, they will quickly overload. Tweaks relate to how many messages should the agent process in one execution step; or to how quickly an agent should forget old / unimportant goals.

Knowledge-base related tweaks relate to how should the importance of the *Free* goal depend on the occupied capacity of the KB; to how quickly should fact persistence fade for facts relating to the agent itself and to facts that relate to other agents; or to how should agent Specialty update when receiving new pieces of information.

3.3.6 Results of Experiments

Let us first see how the system evolves in the case of a scenario in the lines described in Section 3.3.3. What we are interested in is to see how the context measures that we have defined influence the spread of the "test" pieces of data.

Initially – at the start of the system's evolution – none of the agents is interested in a particular domain. During the first phase of the experiment, three data pieces are used, with the following specialties regarding the given domains: $(1.0, 0.0, 0.0)$, $(0.0, 0.0, 1.0)$ and $(0.0, 1.0, 0.0)$ – call them A, C, B according to the corresponding domain. 15 instances of each of the three data are injected in the system, into randomly chosen agents, at regular intervals until simulation step 150. Since the specialty of agents is influenced by the specialty of the data that is received by the agent, the agents' interests will be grouped in contiguous regions, as can also be seen in Figure 3.3. Note that the regions only overlap at their borders. Figure 3.4 shows the agents' interests as a whole and for each separate domain.

The next phase of the experiment consists in injecting, at simulation step 130 and around the center of the system, 3 new data pieces with the following specialties: $(0.0, 1.0, 0.1)$ – call it Bc, $(1.0, 0.1, 0.0)$ – call it Ab, and $(0.1, 0.0, 1.0)$ – call it Ca. Figure 3.5 shows what happens with these new data, as well as with the old data. As shown in the figure, the old data is forgotten as new data arrives. Furthermore, the distribution of the new data is in accordance with the already established agent interests. Note that the less important specialty component (0.1 value) is also relevant to the spreading. Indeed, the Ab piece reaches agents interested in A but also the agents interested in B.

The last part of the scenario tests the behavior of the system for data that is equally related to all domains, but has high pressure. Two such pieces of data are inserted in the system, one at the upper left corner and the other at the center of the grid, at simulation step 200. From the snapshots in Figure 3.6, one can see that these data manage to reach most agents. This last part of the scenario also shows that the existence of two data with high pressure in the system is not a problem (both of them spread to all agents).

The results are not limited to the grid structure (which is again a more convenient way for visualization purposes). Experiments have also been performed on agents placed randomly in the environment and communicating only with agents at a distance under a certain threshold. The obtained results were similar in nature, with the observation that the information took longer to spread, due to the many points where agents were too far to communicate. Results of the same type of experiment as above, but with randomly placed agents, are presented in Figure 3.7.

The results show how generic measures of context can be used, together with a simple (and fast) agent behavior, in order to obtain context-aware behavior. Local knowledge and simple context measures meant that knowledge bases of agents did not need to hold more than 12 root facts, among which the mean recursive depth was 2, meaning that very little memory was used.

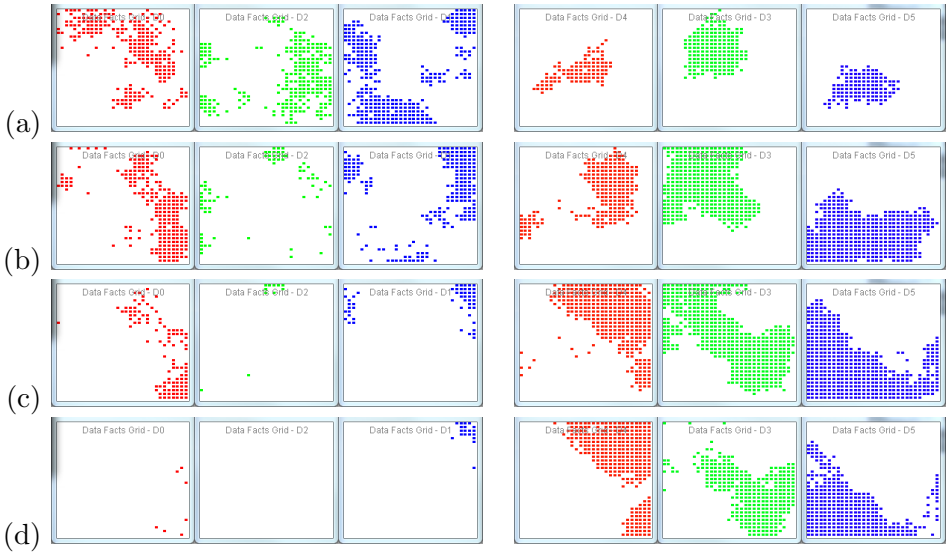


Figure 3.5: Agents forget old data (left) as they receive new data (right) at simulation steps: (a) 152; (b) 170; (c) 210; (d) 271. The corresponding data, from left to right, are related to domains: A, B, C, Ab, Bc, Ca.

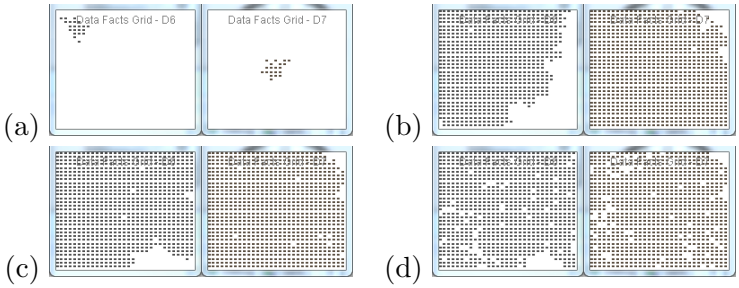
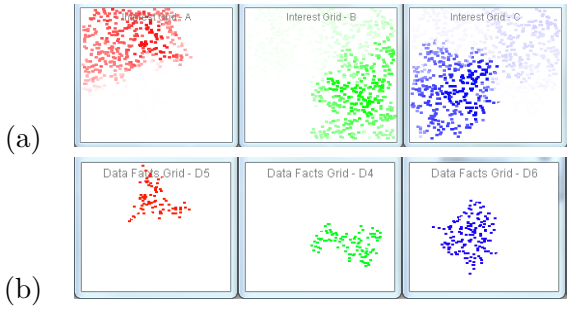


Figure 3.6: Data with high pressure and relatedness to no particular domain spreading through the system at simulation step: (a) 211; (b) 300; (c) 350; (d) 447.

For further evaluation of our system, we have also developed a measure of satisfaction for agents. This measure deals with how relevant the facts that an agent has (that it has received from other agents) are with respect to its specialty. For every fact, an overall degree of usefulness is calculated, considering the history of the agent's specialty (which is recorded throughout the agent's evolution) and the fact's specialty. While it is calculated using the agent's history, this measure is instantaneous. Based on it, a *balance* for the agent is calculated, that measures the balance between the useful facts (useful over a certain threshold) and the useless facts.

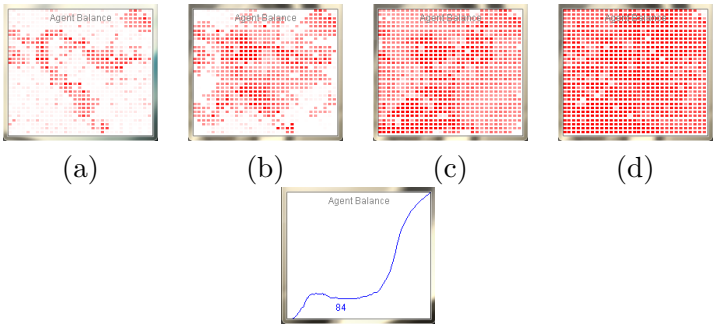
Let us observe how the agent balance changes throughout the system's evolution. We will use the same scenario that we used earlier, and we will observe the snapshots at the same steps. Figure 3.8 shows the evolution of the agent's balance. One can see that at start agents contain many facts that do not regard their specialty, therefore their balance is low. Later, as "test" data begins to spread, the specialty of agents is already formed, so the facts are deemed useful – the balance of agents is high, reaching 90% useful facts toward



(a)

(b)

Figure 3.7: (a) Areas of specialization according to the three domains of interest, with agents randomly placed. (b) Resulting distribution of data.



(a) (b) (c) (d)

(e)

Figure 3.8: (a)-(d) Evolution of the agents' balance at steps 130, 152, 170 and 210 – more intense color means better balance. (e) The graph of the average balance over all agents, between steps 0 and 210.

step 250. This means that the system functions so that the satisfaction of the agents reaches good levels after a certain time[3].

3.4 Lessons Learned

The most important thing that we have learned through the AmIciTy:Mi experiments was that the principle stated in the preamble of Section 3.1 works, and that the distribution of information in the agent system can be controlled by means of the measures of context presented in Section 3.2.1: pressure for speed, persistence for time span, specialty for direction and covered area.

The project has helped design an agent behavior that is based on local interaction and self-organization mechanisms, and that can use knowledge bases of variable size, including very small ones.

Let us look at how such an agent behavior would work for the reference scenario from Section 1.3.4, more precisely for the first part of the scenario, in which Alice's train is delayed and the Professor is informed that Alice will be slightly

[3]The contributions related to agent balance have been implemented by Sofia Neaţă, a Bachelor student at University Politehnica of Bucharest, during he internship at the AI-MAS Laboratory.

late for the course.

The scenario features important elements of our approach: the behavior of the system is context-aware, and the context is changing (dynamic context); the behavior observed in the scenario is based on the exchange of information between several agents (Alice's agent, the Professor's agent, the agent managing the Computer Science Course, the agent managing the train, etc.), connected by relations based on the context of the entities (see also Chapter 4). Examples of such relations are: Alice and the Professor are both part of the same activity – the Computer Science Course, which is part of the University. The activity of Alice may be managed by a specialized Schedule agent. The service that gives information about the present and missing students may be itself an agent – the CourseStarter.

The agent behavior presented in this chapter is necessary so that the information about Alice being late is disseminated, starting from Alice's agent, passing through Alice's Scheduler, through the agent managing the Computer Science Course and reaching the CourseStarter service which will display it to the Professor. There may be other agents with which Alice's agent communicates, but they will not be considered as potentially interested in the information that Alice is late for the CS Course. Even if this information is sent to the University agent (as Alice is a student at the University) the University agent will most likely discard it as it is not interested in the particulars of Alice's attendance to courses.

Of course, in order to obtain the desired outcome, the agents need a better topology and a semantic-aware manner of computing the relevance of information, but the basic behavior of agents would remain the one presented in this section.

3.5 Perspectives

The step that naturally followed the AmIciTy:Mi was the development of the aspects that were kept simple in the platform: agent topology and representation of context. Agents should be neighbors not only if they have nearby positions, but also if they share other types of context. This has been studied in the Ao Dai project, presented in Chapter 4. Moreover, context should be represented in a more advanced manner than just by a Specialty vector and measures of persistence and pressure. This is why a graph-based representation of context, together with context patterns, have been developed, as presented in Chapter 5.

But there are many perspectives for the AmIciTy:Mi platform itself. It makes a great platform to study complex systems formed of large number of agents with similar behavior. It could be extended to support agents of different sizes and agents that move through space. It could also be used to develop new measures for context, as well as new measures for evaluating the evolution of the system.

Chapter 4

Structuring the Agent System[1]

Following the implementation of and the experiments with the AmIciTy:Mi project – presented in the previous chapter, it was clear that more structure was needed for the system, beyond the simple space-related topology of the AmIciTy agents.

An essential factor in the effort to find a better system topology was the work with the CLAIM agent-oriented programming language (Computational Language for Autonomous, Intelligent and Mobile agents) and the Sympa platform (Système Multi-Plateformes d'Agents) [Suna and El Fallah Seghrouchni, 2004]. Beside the advantage of being agent-oriented, CLAIM is inspired by process calculus, and more precisely by the ambient calculus of Luca Cardelli [Cardelli and Gordon, 2000]. By having mobile ambients as foundations of the language, CLAIM agents are characterized by mobility, and specifically hierarchical mobility. That is, agents are placed in a hierarchy and when one agent moves to another machine, its entire sub-tree of agents moves with it, automatically.

Our approach to creating a context-aware agent topology – presented in Section 4.1 – is based on the idea of mapping the hierarchical aspects of context to a hierarchy of agents. This approach has been validated for a first time with the Ao Dai project.

4.1 Approach

From the point of view of the application layer, it is context-awareness that is the solution for a predictable, natural flow of information. Like in social networks and shopping sites, one can assume that the user will be interested in things that are related to what he or she already knows, to what he or she does, and to the people that he or she is acquainted with. It is unlikely that someone

[1] The results of the work presented in this chapter have been published in our paper for PRIMA 2010 [El Fallah Seghrouchni et al., 2010b, El Fallah Seghrouchni et al., 2011].

is normally interested in something that bears no relation whatsoever with any part of his or her life. Relevant information is information that is related to the context that the user is in. Context has several aspects: physical space, time, activity (current, past or planned), social relations and (computational) resources. Considering a space in which the dimensions relate to these five aspects, relevance of information may be defined as *proximity* in this space.

As context is based on this sort of locality, this also solves the problem of information overload. The user can only do one thing at a time, be in only one place, only a number of past actions are still relevant and only a limited number of actions can be planned. So the context space of the user is limited and will only be related to a limited amount of information, that itself can be sorted according to its degree of relevance toward the current context.

The need for more structure for the system comes from two directions. First, in the context of the AmIciTy framework, the agent behavior is to exchange interesting information with the potentially interested neighbors in the topology of the system. But agents that are potentially interested in the information should share some context with the agent that is sending the information. That is why the structure of the system should be based on shared context.

Moreover, our commitment to a distributed system favors a topology that is decentralized, but a policy is needed to know which agent should communicate with which. Spatial proximity is one option, that is adequate for wireless sensor networks, for instance. But we chose to use other aspects of context as well. One of them – that has been integrated in Ao Dai – is computational context, i.e. the relations between devices, services, and spaces.

4.1.1 CLAIM and Sympa

As an agent-oriented programming language, CLAIM eases the task of implementing multi-agent systems [Suna and El Fallah Seghrouchni, 2004]. It works on top of Java, giving direct access to Java resources if needed. Agents implemented in CLAIM are executed using the Sympa platform, that manages the agents' life cycle and also their mobility.

The CLAIM language is based on explicit declaration of agent's characteristics. For example, the code in Figure 4.1 shows a part of the definition of agent *PDA* in the Ao Dai project.

Agents are characterized by their parent in the agent hierarchy, their knowledge – represented as first order predicates, their goals, messages that they can receive, capabilities, processes that they execute, and agents that are their children. Capabilities are activated by certain messages that are received, or certain conditions that can occur (that are verified continuously).

For instance, in the example above, when the agent *PDA* receives a message about its new location, it will execute the action "migrate". In this action, it checks if its actual location is already the location brought by the message (which is represented by the variable ?*name*). If it does, the agent ignores

PDA.adf agent definition file

```
1.  defineAgentClass PDA(?w,?h,?x_i,?y_i){
2.    authority = null;
3.    parent = null;
4.    knowledge = {location(?x_i,?y_i); type(1);}
5.    goals = null;
6.    messages = null;
7.    capabilities = {
8.      message = PDAatLoc (?name,?x_new,?y_new);
9.        condition = null;
10.       do{send(this,migrateTo(?name))}
11.       effects = null;
12.     }

13.     migrate{
14.       message = migrateTo(?name);
15.       condition = not(Java(PDA.isParent(this,?name)));
16.       do{send(this,removeOldNavi(?name))
            .moveTo(this,?name).send(this,demandNavi(?name))}
17.       effects=null;
18.     }
19.     ...
20.   processes={send(this,starting())}
21.   agents=null;
22. }
```

Figure 4.1: Sample of CLAIM code, used in the Ao Dai project for the PDA agent.

the message, otherwise, it will move to the new site by calling the function "moveTo()". If the new site is located in another computer in the network, the agent *PDA* and all its children will migrate to this new computer.

It is important to observe that agents are part of an agent hierarchy. There is one or more logical trees of agents, each agent being able to have a parent and a certain number of children. This idea of having an agent hierarchy is central to our approach.

It is also important that CLAIM agents are mobile, featuring strong mobility: when they move to another machine, their execution continues without losing knowledge, messages or capabilities. The developer, in this case, need not to worry about the code migration and registration problems that may arise. The language takes care of it, concentrating the agents' information in the *Administration System*. To address the security issues concerning mobile code, CLAIM offers some features like the agent's authority validation, and also allows the developer to decide if an agent must have some special access or if an agent must have some resource denied. The sum of these features creates a powerful platform to the development of agent-oriented mobile applications.

From the point of view of this work, there are two structures that agents are part of: the physical structure, where each agent executes on a certain machine; and the logical structure, where agents are part of logical hierarchies, that may well exist across multiple machines.

4.1.2 Structure of Context vs. Hierarchy of Agents

Although many AmI applications presented in the literature consider context-awareness as being related only to location and smart spaces, there much more to context than that (see also Section 2.2). Research in context-awareness shows that there are many aspects of context. One classification of context [Chen and Kotz, 2000] divides context into computational context – available computing and networking resources, including the cost for using them; user context – user's profile, location, people and objects nearby, social situation; physical context – light and noise levels, temperature, traffic conditions, etc; and time context – the current time coordinate of the user and related information (like the season, for instance). Context can be further classified [Dey and Abowd, 2000] as primary – sensed directly by sensors and specialized devices – and secondary – which is inferred from the primary context. Another type of context is activity – information about the current activity of the user, and related devices and people [Henricksen and Indulska, 2006].

In our work, we deal with 5 types of context: spatial context, computational context, temporal context, activity context, and social context. One can observe that the structure of the 5 types of context is, to a significant extent, hierarchical: places are part of larger places; computational resources cover certain spaces, are located in certain places, and services run on certain devices; time can be iteratively divided in intervals of time; activities are formed of sub-activities; and social groups may also have hierarchies. Moreover, activities take place in a certain interval of time, and concern certain users, which, by participating in the same activity, form a group. As these types of context are hierarchical, it means that the current context of a user is in relation with several of these hierarchies: the user is in a certain place, at a certain time, participating in a certain activity, using certain devices and services.

Since both context and the CLAIM agent programming language are related to hierarchical structures, we can attempt two connect the two, by mapping the structure of the multi-agent system (implemented in CLAIM) to the structure of the current context. This offers not only a representation of context by means of the structure of the multi-agent system itself, but it also offers a means for agents to easily communicate only within their context. This is the central idea of the Ao Dai project.

4.2 The Ao Dai Project

The purpose of the Ao Dai project is to implemented a simple AmI-oriented multi-agent system scenario, in which agents are assigned to different elements of context – places, devices, services, users – and hierarchical relations between the agents reflect the hierarchical structure of context. In the scenario, a user is pro-actively assisted in navigating through the floor of a building and in locating computational resources needed for a presentation.

The Ao Dai project has been implemented in collaboration with Thi Thuy

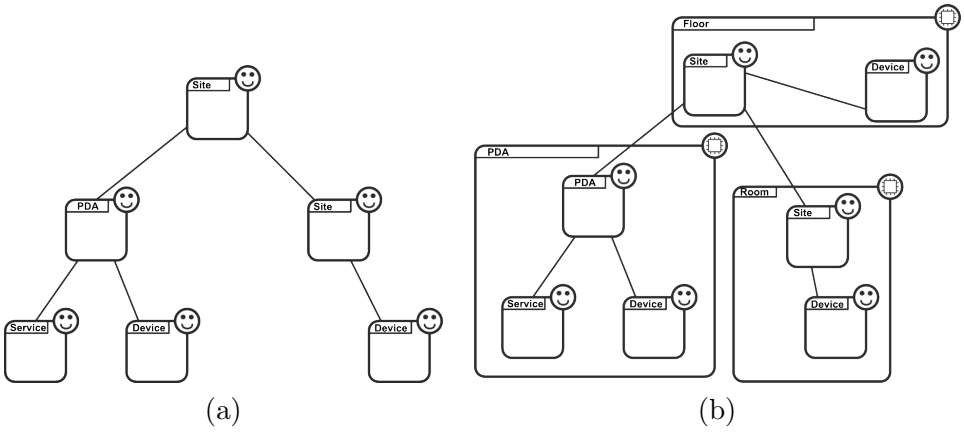


Figure 4.2: Visual representations of sample agent hierarchies: logical (a), and both physical and logical (b).

Nga Nguyen and Diego Salomone-Bruno, under the supervision of prof. Amal El Fallah Seghrouchni. The scenario presented below has been demonstrated in a simulated environment, running on two different machines, during the 5th NII-LIP6 Workshop, held in June 2010 in Paris, France[2].

4.2.1 Design Idea

The architecture of the Ao Dai system revolves around one critical idea: mapping different contexts to different parts of the logical hierarchy of agents formed by the parent / children relationships in CLAIM agents.

Location is, notably, the most used context in applications, because it reflects an important set of physical contents [Dey and Abowd, 2000]. In the Ao Dai project, besides location, we also consider as part of the user's context the available computing resources around him and his preferences.

The context-awareness in Ao Dai is done by exploiting the particular hierarchical agent structure that is offered by the CLAIM language. In CLAIM it is very easy for the developer to instruct agents to move from one parent to the other, and an agent moves, automatically, along with its entire sub-hierarchy of agents. This resembles the mobile ambients described by Cardelli [Cardelli and Gordon, 2000] and is an essential advantage when implementing context-awareness. That is because agents, while representing devices or locations, can also represent contexts, allowing the developer to describe, in fact, a hierarchy of contexts.

For example, when the user is inside a room, its agent is, in the hierarchy, a child of the agent managing that room. The children of user's agent – devices or services – are also in the same context. When the user moves to another room, the user's agent changes parent and, along with it, its children move as

[2]Workshop held in collaboration by the National Institute of Informatics in Tokyo and the Laboratory of Computer Science of University Paris 6. Details at `http://www-desir.lip6.fr/~herpsonc/5workshopNii/program.htm`

well, therefore changing context. Some devices may not be able to move along with the user (e.g. fixed screens, etc.) so they will determine that the new context is incompatible with their properties, moving back to be children of the agent managing the room.

But context is not only about location, and the hierarchical structure that is offered by CLAIM can be used for easy implementation of other types of context. One of them is computational context. When the user uses a service, an agent is created that offers that service and that becomes a child of the user's agent. It is easy for the service to interrogate its parent in order to find out more about its capabilities. Conversely, it is easy for the user's agent to check on its children – services or devices – in order to find the resources and capabilities that the user is able to use.

One last type of context that is handled in Ao Dai is user preferences. The user is able to input preferences on the capabilities of devices that it needs to use. These preferences are then integrated in the queries that are launched by the agents. While the structure offered by CLAIM is not directly useful for this aspect, the preferences help find not only the closest device with the required capability, but also the closest device that fulfills certain user requirements. Preferences can also be used to limit the range of the search, which is meaningful from the context-aware point of view: a device that is closer in the agent hierarchy also shares more context with the user.

4.2.2 Scenario

In the project, we have studied several scenarios including the following (see also Figure 4.3): a user has a meeting in a building that he / she does not previously know. When arriving at the right floor, the user's PDA automatically connects to a local wireless access point. A CLAIM agent executes on the user's PDA – we will call this agent *PDA*. Another agent executes on a local machine and manages the context of the building's floor – call it *Floor*. *Floor* detects the presence of the user's PDA, and instructs the *PDA* agent to move in the agent structure and become a child of *Floor*. The movement is only logical: the agents keep executing on the same machines as before.

When *PDA* enters the floor, *Floor* also spawns a new agent – called *Navigator* – and instructs it to move as a child of *PDA*. This time, the movement is not only logical: *Navigator* is a mobile agent that actually arrives on the user's PDA and will execute there for all the time during which the user is on the floor. The *Navigator* can provide *PDA* (and inherently the user) with a map of the floor, can translate indications of the floor's sensors (sent to *Navigator* by *Floor*, and through *PDA*) into positions on the graphical map, and can calculate paths between the offices on the floor. *Navigator* is an agent that offers to the user services that are available and only makes sense in the context of the floor.

For displaying the map, *PDA* may detect that its screen is too small too appropriately display the map, so *PDA* will proactively initiate the search for a larger screen in the nearby area. The search can have several criteria: the

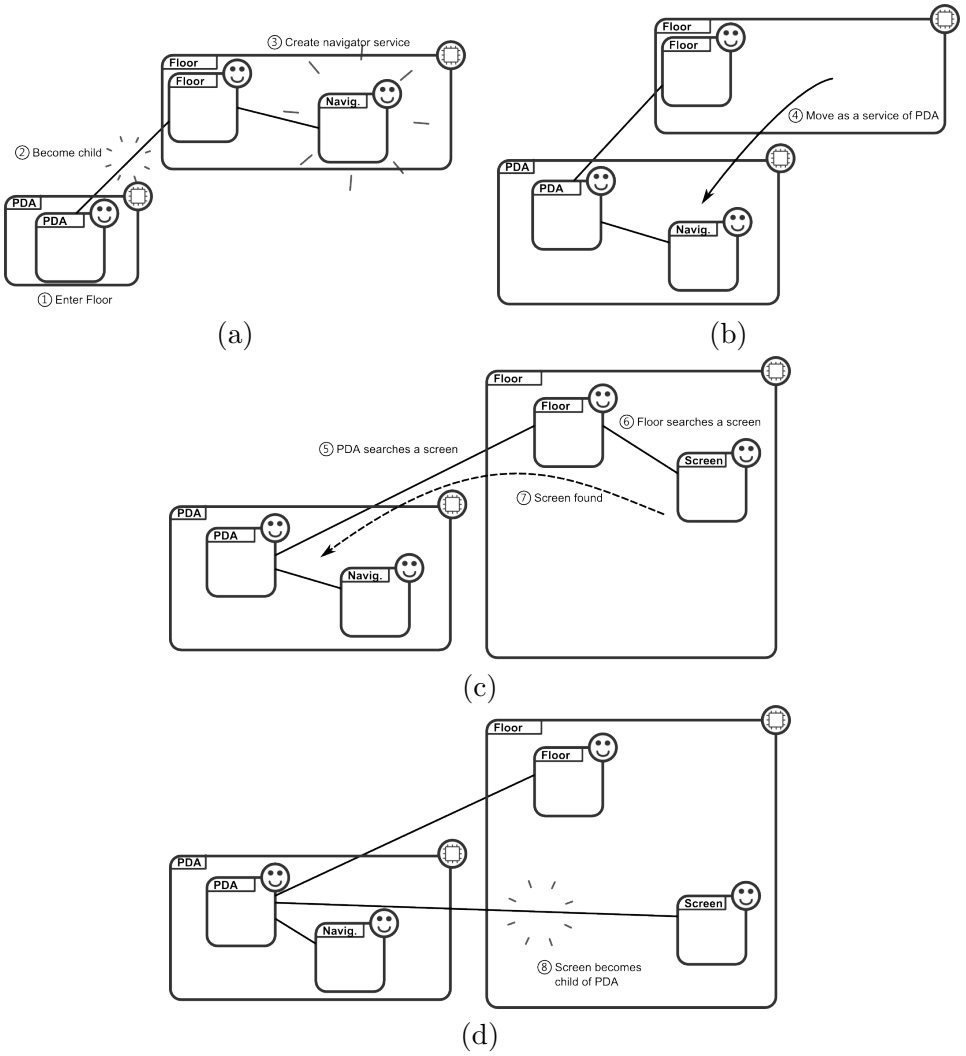


Figure 4.3: Steps of the Ao Dai scenario: a user with a PDA enters the floor, *Floor* takes *PDA* as a child and creates a *Navigator* service (a); *Navigator* is sent as child of *PDA* (b); *PDA* requires a screen, which is found as child of *Floor* (c); *PDA* gains control of *Screen*.

space in which the search will take place (the current office, a nearby office, the whole floor), the range in which to search, and the minimal size of the searched screen. Devices are searched by the capabilities they offer – in this case the *display* capability is needed. *PDA* sends the query to its parent – *Floor* – which in turn locates among its children an agent *Screen*, that manages a physical screen that fits the requirements, is located near to the user and is available. *Screen* answers the query and *PDA* asks it to move to become its child. Being a child of *PDA* also marks the fact that *Screen* is in use by the user, and *PDA* gains control over the displayed information. Agent *Screen* may either run on the actual intelligent screen, or may only manage the screen while being executed on a server. When the user moves farther from the screen, the PDA will detect that the context is no longer compatible and will free *Screen*, which will return to be a child of *Floor*.

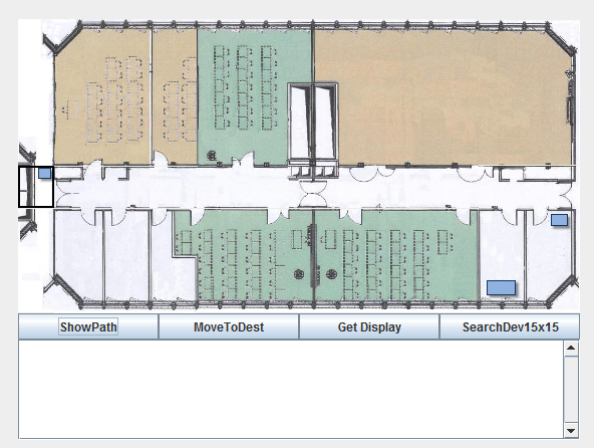


Figure 4.4: The interface for the simulation, displaying the floor plan of a corridor in the LIP6 laboratory.

4.2.3 Implementation

In the implementation of the scenario presented in Section 4.2.2, there are three major types of agents: the *Site* agent (of subtypes *Floor* and *Office*), the *Device/Service* agent (e.g. *Navigator* agent, *Agenda*, *Screen*) and the *PDA* agent, the latter with the specific role of representing the user during the simulation. Also see Figures 4.2 and 4.3.

- The *Site* agent is used to determine the physical relationship between the agents. It means that an Office agent is a child of a Floor agent only if it is physically located on the given floor.
- The *Service* (or *Device*) agent has the capability to offer to the other agents some specific service. It may be in a direct or indirect way, like showing some information on the screen or advising other agents of the user meeting.
- The *PDA* agent works like a personal device that follows the user through his tasks. The most important features of this agent are the fact that the PDA moves physically with user and has the CLAIM capability of managing requests for services or devices. It also stores user's preferences. It is important to note that the PDA actions will depend mostly of the user's current context.

In the first version of this project, the context is directly sensed (in a simulated manner) by the *PDA* and the *Site* Agents. In real applications, an additional layer would be needed to capture the sensor information and translate it into useful data.

In a highly distributed AmI environment, a good representation of context and context-related relations between devices means that most of the communication will happen only at a local level within the structure formed by these relations. In Ao Dai, the CLAIM agent hierarchy facilitates this: agents sharing a parent share a context.

To preserve the hierarchy of agents, agents are allowed to interact only with their parent and their children. Take for example the search for devices (see Figure 4.3). When agent *PDA* wants to search for a device with a certain capability and certain criteria, it must send a request to its parent, for example agent *Floor*. Once the request received, agent *Floor* searches itself to see if it has the requested capability and it satisfies the criteria. If it does, *Floor* answers immediately to agent *PDA*, in the other case, it searches in all of its children (if any) except the agent who invoked the search (agent *PDA*). After all of its children have answered, agent *Floor* checks if there are one or more children that have the capability requested and satisfy the criteria. If it has a confirmation answer, it sends the search result which contains the information about the found device(s) to agent *PDA* and the search is finished. If not, agent *Floor* has to search in its parent (if any). After the parent has answered, the agent floor sends the search result to agent *PDA* and finishes the search. The search process is executed recursively. User preferences can be used to limit the range of the search to closer contexts.

The advantage of using such a protocol in conjunction with mapping context over the agent hierarchy is that the search will usually end very quickly, assuming the user will most times ask for devices that are likely to exist in his context. The search is executed in the current context first, and then in the parent context and sibling contexts.

4.3 Lessons Learned

The experience of the Ao Dai project has taught us valuable lessons about the implementation of Ambient Intelligence applications. On the one hand, it was about the requirements of creating a scenario closer to real-life, and about the difficulties of simulating such a scenario. Real-life scenarios come with the need for credibility, and it is more difficult to restrict the world of the application to just a few elements. Also, the simulation of scenarios requires the tools to input events to the system, and the tools to visualize the system. Although these were fairly primitive in the first implementation of Ao Dai, we have learned what requirements exist for future implementations.

On the other hand, we have learned about how context can be linked to the topology of the agent system, and about the advantages of such an architecture. First, the possibility of local interaction, and local searches, in a distributed environment. Although the Ao Dai project only used few agents, there was no central agent. Second, the hierarchical structure that is related to context makes it much easier to understand how the system works, and the communication between agents becomes much more intuitive.

Another important thing that was learned was that CLAIM is a language that is particularly appropriate for the implementation of this type of application, and its hierarchical structure for the agent system is of great help for the representation of context that we have introduced.

4.4 Perspectives

The Ao Dai project was not an isolated project. It has lead to several followups, both in theoretical development as well as in the development of the platform.

As a theoretical proof-of-concept, the research that followed extended the idea of mapping context to structure of the agent system. More types of context were considered, leading to several types of agents and specific hierarchical relations between agents, depending on the type of shared context (see Section 5.3).

As a platform for AmI applications, the Ao Dai project lead to the creation of a new agent platform, underpinned by the JADE Agent Development Framework [Bellifemine et al., 2001], and using a cleaner and simpler version of CLAIM, that was named S-CLAIM, and that is more focused on agent-characteristic features, rather than being a full-featured programming language (see Chapter 6).

Last but not least, the Ao Dai project, initially a collaboration between students from several universities in Europe, Brazil and Vietnam[3], has become the base for a new project lead by teams from the LIP6 laboratory in Paris, the AI-MAS laboratory in Bucharest, and Honiden-Lab at the NII institute in Tokio.

[3]MAS team from University Pierre et Marie Curie (Paris 6), AI-MAS from University "Politehnica" of Bucharest, IFI institute form Hanoi and PUC-Rio University from Brazil.

Chapter 5

Improving Context-Awareness

There are many definitions for what context is. One of the dictionary definitions refers to context as *"the interrelated conditions in which something exists or occurs"*[1]. Another definition states that context is *"that which constrains something without intervening in it explicitly"* [Brezillon and Pomerol, 1999]. These vague definitions point to the fact that the elements that compose the context have an influence on what is happening, but they are somewhat external to it.

In the field of Ambient Intelligence and context-aware applications, there is one definition that is more cited than others, given by Dey in 2001: *"Context is 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].

As for *context-awareness*, being context-aware is that property of an application that *it automatically adapts to discovered context, by changing the application's behavior accordingly* [Chen and Kotz, 2000]. For AmI applications and infrastructures, context is fundamental, as the capabilities of being proactive and non-intrusive, and natural and intuitive, require the application to understand the context and the expectations of the user – the same action for the same causes may be regarded as appropriate in some contexts and as unappropriate in others.

To exemplify, let us reiterate our short example in Section 1.3.2: a researcher on the last day before the deadline for an important conference, writing the article (that has been left for the last moment) may be annoyed to be notified of a message from a colleague regarding a interesting link that relates to their research field. Normally that wouldn't happen, but in *this particular context*, it is best to show the notification for the message only on the following day.

[1] The second meaning of the word "context" (the first refers to texts) from `http://www.merriam-webster.com/dictionary/context`.

Level	Context-aware elements	Result
System (outside the agent)	Context-aware relations between agents	Context-aware topology
Agent (inside the agent)	Context representation and context patterns	Context-aware behavior

Table 5.1: Elements of our approach: the level, the implementation, and the result.

A context-aware application (and an AmI application in particular) must be able to **access context information** (e.g. know what is the activity of the researcher, know when is the deadline for the article), must **understand the context** (e.g. understand the relations between the different facts, be able to evaluate the importance of the message that is received) and must be able to **decide upon the correct context-aware action** (e.g. know about previous user experience and expectations, and also actually be in charge of managing notifications).

Our goal is to **integrate context-awareness in a multi-agent system for Ambient Intelligence, so that agents naturally access, process and share context information**.

This chapter presents our approach to context representation and context-aware behavior that is placed in the context of the approach to an AmI middleware presented in Chapter 3, but much more improved. Section 5.1 gives a holistic perspective on our approach to context-awareness and describes the formal model for the multi-agent system. Following are details on the two aspects of context-awareness: first, a better representation for the context of information and for the context of the agent, as well as means to recognize relevant information in the given context, using *context graphs*, *context patterns* and *context matching* – Section 5.2; second, a context-oriented topology for the agent system that is built upon the concepts developed in Chapter 4, but using a more general mapping between context and agent system topology, which is also integrated with the context graphs inside the agent – Section 5.3. Last, an improved agent behavior is described, based on the elements of context-awareness in the previous sections – Section 5.4.

5.1 A Holistic Approach to Context-Awareness

Our approach to building a multi-agent system for the application layer of Ambient Intelligence features two aspects: the context-aware topology of the agent system and the use of context patterns to recognize relevant information. Beside these two elements, the behavior of the agents relies on the exchange of information with neighbor agents in order to spread interesting information (with respect to the current context).

Relying on local information exchange has proved effective in previous studies (see Chapter 3), in which pieces of information successfully spread in a multi-agent system where the agents had a preference only for certain types of

information. The information did spread quickly and correctly, even though each agent had only a very small knowledge base that referred only to its own information and to some of the neighbors' information. However, AmIciTy:Mi used a space-based topology – two agents were neighbors (i.e. could communicate directly) only if they were within a certain distance of each other – and a simple indication of context – information was characterized by some domains of interest, its persistence and a measure of importance.

In this chapter we extend our approach with an improved topology – that considers shared context of more types – and improved computing of relevance – the relevance of the information is computed using a graph-based representation for information and also graph patterns that represent the interest of the agent. We present the elements of our approach in Table 5.1.

5.1.1 A Formal Model for the Multi-Agent System

The multi-agent system is organized on three levels: containers, agents, and knowledge / context information. The current state of the system represents the current state of the world, according to the perspective of the system as a whole. Likewise, the context graphs held by each agent represent the current state of the world from the perspective of the agent.

Containers are assigned to physical machines, and at a moment of time each agent executes on one container. Each agent has knowledge, represented by its context graph, and by its context patterns. Each of the three levels forms a graph, that is a subgraph of the whole three-level graph, that we will call the **Tri-Graph**.

The **containers** form a complete subgraph $ContainerGraph$:
$ContainerGraph = (Containers, Connections)$
$Containers = \{Container \mid Container$ is a container that allows the execution agents $\}$
$Connections = \{(C_i, C_j) \mid C_i, C_j \in Containers\}$

$Connections$ shows from what container to what other container can the agents move. In this work it will be considered as containing a pair for each two containers.

The **agents** form the subgraph $AgentGraph$ in which edges are labeled with types of relations related to shared context (see Section 5.3.2): *is-in*, *part-of*, etc.:
$AgentGraph = (Agents, AgentRelations)$;
$Agents \subset AgentNames$;
$AgentRelations = \{(A_i, A_j, Relation)\}$ where $A_i, A_j \in Agents$ and $Relation \in AgentRelationTypes$;
$AgentRelationTypes = \{is\text{-}in, part\text{-}of, of, in, controlled\text{-}by, executes\text{-}on\}$.

The assignment between **agents and containers** is done by a supplementary component of the Tri-Graph – $AgentLocations$, that links the container level with the agent level:

$AgentLocations \subset Agents \times Containers \times \{resides\text{-}on\}$, where
$\forall A \in Agents, \exists\, (A, C_i, resides\text{-}on) \in AgentLocations$, and
$\nexists j \neq i, (A, C_j, resides\text{-}on) \in AgentLocations$.

An **agent** A is a tuple $A(Name, CG_A, Patterns, \mathcal{R}, I, Goallist)$, containing the agent's name, context graph, set of patterns, relations with other agents, interest information, and goal list. These components will be discussed in detail in Sections 5.2.2 and 5.4.

The relations component of the agent describes all the relations of the agent with the other agents:
$\mathcal{R} \subset (AgentNames - \{Name\}) \times AgentRelationTypes \times \{in, out\}$.

The **Tri-Graph** is formed by the reunion of the graphs for containers, agents, and agent knowledge:
$Tri\text{-}Graph = (Nodes, Edges)$, where
$Nodes = Containers \cup Agents \cup \bigcup_{A\ agent} CG_A.Concepts$
$Edges = Connections \cup AgentRelations \cup AgentLocations$
$\cup \bigcup_{A\ agent} CG_A.Relations$

Note that, as we have $Containers \subset Concepts$, $Agents \subset Concepts$, and also $AgentRelationTypes \subset Relations$, it is possible that the reunions of nodes and edges from the agents' context graphs already contain the other components of the equations above, but that is not strictly necessary.

Also note that an agent may "know" any part of the Tri-Graph (at any level), therefore $\forall A\ agent, CG_A \subset Tri\text{-}Graph$.

In the sections that follow, we will discuss the details of context-awareness inside the agent – context graphs, patterns, and matching – and outside the agent – the context-aware topology of the agent system.

5.2 Context-Awareness Inside the Agent[2]

As we have seen in Section 2.2, context-awareness is assured by obtaining context information – which is usually related to physical factors, most notably location – and deciding upon appropriate action by means of rules.

There are several issues regarding context-awareness as presented by many AmI applications in the field [Perttunen et al., 2009, Bettini et al., 2010]: either the representation of context is very simple (tuples or key-value pairs), being easy to process but very limited in possibilities; or the representation is very complex, ontologies are used, and the processing is most times externalized to "context servers". Either way, there is little exploitation of the actual relations between concepts.

In our approach towards an implementation of Ambient Intelligence, we are

[2]The results of the work presented in this section have been published in our paper for IsAmI 2011 [**Olaru** et al., 2011]

trying to build mechanisms and representations that facilitate a more flexible approach to AmI and context-awareness, while in the same time are easy to implement and can work on resource-constrained devices.

After discussing the elements that lead to our approach in Sections 5.2.1, we present in Section 5.2.2 a simple formalism that allows agents in a multi-agent system, that have only local knowledge, to share and process context-related information and to solve problems by using context matching (Section 5.2.3).

5.2.1 "Context as a Dressing of a Focus"[3]

When the agents feature a more advanced representation of their knowledge, one may ask what is the difference between the knowledge of the agent and the information about context. Context information is a part of agent's knowledge (it is something that the agent knows) but what knowledge is not context information?

As pointed out by the work of Brezillon [Brézillon and Brézillon, 2007], **the discriminating factor between what they call external knowledge and contextual knowledge is the focus**. In case of user-application interaction, the focus of the user's attention. Contextual information is that information which is related to the current focus, and everything else is external knowledge. The focus changes either as a cause of external events, or as a cause of internal decisions.

But how do we know what is the focus? And how do we know what is relevant with respect to it? In the experiments presented in Chapter 3, the agent was interested in information that was compatible with its specialty. As new relevant information was received, with slightly different specialties, the specialty of the agent changed. In those experiments, we could consider that the focus was current specialty – this influenced what information was considered relevant, and also changed with the newly received information. So we can consider that as the relevance of information is depending on context, conversely what characterizes the focus is what information is considered as relevant.

Based on the work of Henricksen and her colleagues [Henricksen et al., 2002, Henricksen and Indulska, 2006], that represent context information as a set of associations, we choose to represent context information as a graph – the Context Graph of the agent. Information in this graph will be all the information that the agent has and that is currently relevant to the agent's activity. The general focus of the agent's activity will be defined by the situations that it is able to handle, which will be represented by means of context patterns, that are graphs with generic labels, and which fit a certain range of "concrete" graphs. An agent's patterns also show what kind of information is relevant to the agent.

While representing knowledge as graphs is nowhere near new, this work brings

[3]The quotation reads the title of the paper by Brezillon and Brezillon [Brézillon and Brézillon, 2007].

two important novelties: first, the representation of context information as graphs; second, using matching of context patterns against the context graph to detect context compatibility.

5.2.2 Defining Context Graphs and Patterns

Each agent A has a *Context Graph* $CG_A = (V, E)$ that contains the information that is relevant to its function.

$CG_A = (V, E)$, where $V \subset Concepts$ and $E = \{edge(from, to, value, persistence) \mid , from, to \in Concepts, value \in Relations, persistence \in (0, 1]\}$

The elements of *Concepts* and *Relations* are strings or URIs; *Relations* also contains the empty string, for unnamed relations.

For instance, the agent that assists Alice (from our reference scenario in Section 1.3.4) would have a graph that contains parts of Alice's schedule, like in Figure 5.1.

In order for agents to be able to recognize situation, we introduce context patterns [**Olaru** and Florea, 2010a, **Olaru** et al., 2011]. A pattern represents a set of associations that has been observed to occur many times and that is likely to occur again. Patterns may come from past perceptions of the agent on the user's context or be extracted by means of data mining techniques from the user's history of contexts. Commonsense patterns may come from public databases or be exchanged between agents. A pattern is also a graph, but there are several additional features that makes it match a wider range of situations. For instance, some nodes may be labeled with "?"; also, edges may contain regular expressions.

An agent has a set of context patterns *Patterns*:

$Patterns = \{(G_s^P, relevance, persistence) \mid s \in PatternNames, G_s^P \text{ a graph pattern}, relevance, persistence \in (0, 1]\}$.

A **context pattern** s is defined as a graph G_s^P that has some special properties, as follows[4]:

$G_s^P = (V_s^P, E_s^P)$, the graph is the pair of vertices plus edges.

$V_s^P = \{v_i^P(label) \mid label \in Concepts \cup \{?\}\}$, with the condition that

$\exists\, i \neq j, v_i^P.label = v_j^P.label \Rightarrow v_i^P.label = v_j^P.label =?$, that is there cannot be multiple nodes with the same label, if the label is not a question mark.

$E_s^P = \{(from, to, label, characteristic, actionable)\}$,

with $from, to \in V_s^P$, $label \in \mathcal{R}egexps(Relations)$,

$characteristic$, $actionable \in (0, 1]$,

where we noted $\mathcal{R}egexps(Alphabet)$ the set of all regular expressions that can be build over *Alphabet*.

Besides the vertices and the label of an edge, there are to other features that define it: *characteristic* defines how characteristic the edge is for the pattern,

[4] We will mark with " ^{P} " graphs and elements that contain ? nodes, regular expressions, and other generic features.

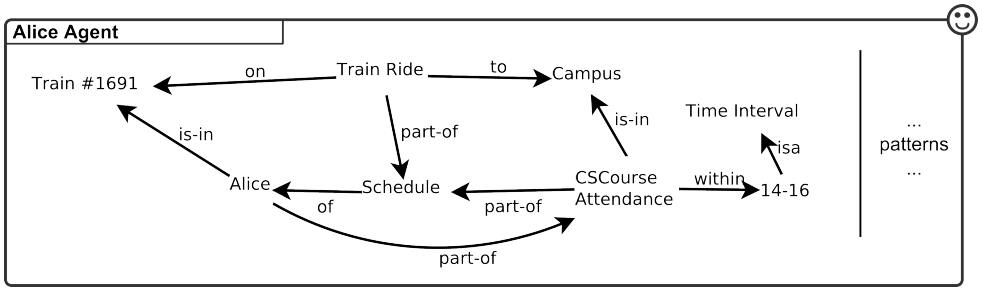


Figure 5.1: The context graph of the agent assisting Alice, showing information about Alice's activity.

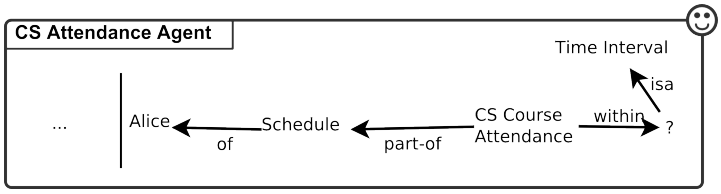


Figure 5.2: Context pattern matching the time interval in which Alice will attend the CS course.

and influences the measurement of how well a pattern matches a subgraph; *actionability* measures how correct it would be for the agent to infer the existence of this edge in the context graph if the pattern matches a subgraph in CG, but this edge is missing.

Beside the graph itself, the pattern is also characterized by two other properties that are stored in the *Patterns* component of the agent description: *relevance* shows how important an information from the context graph is, if it is matched by the pattern; *persistence* shows for how long a new information will persist after being matched by a pattern. See section 5.4 for details.

Both *relevance* and *persistence*, as well as *characteristic* and *actionable* of each edge are set in the process of pattern creation or mining, that is not covered in this work.

Patterns represent situations that are relevant to the function of the agent. Therefore, the agent will be interested in information that matches these patterns. Take for example agent *CSCourseAttendance*, that manages Alice's attendance to the course: among others, it will be interested in information about when will Alice actually attend the course. This can be expressed by the pattern in Figure 5.2. That is, the pattern matches graphs that contain the time interval in which Alice will be attending the course. Since Alice will be late, we would want the *CSCourseAttendance* agent to receive information about the new interval in which Alice will be attending the course, with an updated start time.

5.2.3 Recognizing Context: Context Matching

An agent has a set of patterns that it matches against the current context, and against information that it receives from other agents.

A *match i* between a context pattern G_s^P and an agent A's context graph CG_A is defined[5] as $M_{A\text{-}si}(G'_A, G_m^P, G_x^P, f, k_f)$.

G'_A, G_m^P, G_x^P are graphs, with $G'_A \subset CG_A$, $G'_A = (V', E')$, $G_m^P = (V_m^P, E_m^P)$, $G_x^P = (V_x^P, E_x^P)$, where

$V_m^P \cap V_x^P = \emptyset$, $V_m^P \cup V_x^P = V_s^P$

$E_m^P \cap E_x^P = \emptyset$, $E_m^P \cup E_x^P = E_s^P$

That is, $G'_A \subset CG_A$ is a full match for the *solved part* G_m^P of pattern G_s^P. What is left of the pattern is the *unsolved part* G_x^P (also called *the problem*). There is no intersection between the solved and unsolved parts of the patterns.

The function $f : V_m^P \to V'$ is injective, and:

(1) $\forall v_i^P \in V_m^P, v_i^P.label =?$ or $v_i^P.label = f(v_i^P).label$

and

(2) $\forall e_i^P \in E_m^P$, with $e_i^P.from, e_i^P.to \in V_m^P$, $e_i^P.value$ matches (as a regular expression) the sequence $value_0 \cdot value_1 \cdot \dots \cdot value_p$ of values from edges $e_0 = (f(e_i^P.from), v_{a_0}, value_0) \dots e_k = (v_{a_{k-1}}, v_{a_k}, value_{k+1}) \dots e_p = (v_{a_{p-1}}, f(e_i^P.to), value_p)$, with $k = \overline{1, p-1}$, $a_k \in \{0, \dots, |V'| - 1\}$ and $v_{a_k} \notin f(V_m^P)$.

That is, every non-? vertex from the solved part must match a different vertex from G'_A; every non-RegExp edge from the solved part must match an edge from G'_A; and every RegExp edge from the solved part must match a series of edges from G'_A. A supplementary condition is that G'_A does not contain other nodes or edges than the ones that are matched by the pattern (G'_A is minimal).

The number $k_f \in (0, 1]$ indicates how well the pattern G_s^P matches G'_A in match $M_{A\text{-}si}$, and is given by the normalized sum of the *characteristic* factors of matched edges, i.e.

$$k_f = \frac{\sum\limits_{e_i^P \in E_m^P} e_i^P.characteristic}{\sum\limits_{e_j^P \in E_s^P} e_j^P.characteristic}$$

We will also use the term that a pattern G_s^P *k-matches* (matches except for k edges) a subgraph G' of G, if condition (2) above is fulfilled for $m - k$ edges in E_s^P, $k \in [1, m-1]$, $m = ||E_s^P||$ and G' remains connected and minimal. A k-matching pattern with k above a certain threshold may indicate a problem in the situation of the user: the pattern matches, therefore the user is in the specified situation, but some elements are missing, therefore it may mean that the agent should try and retrieve those elements.

Equivalently, we can define the match of any two non-generic graphs G_X and G_Y – where G_Y has the place of the "pattern" – as $M_{G_X\text{-}G_Y i}(G'_X, G_m^P, G_x^P, f, k)$,

[5] There may be multiple matches between the same pattern and the same graph.

since a graph is a particular case of graph pattern, with no "?" nodes and with no edges using $\mathcal{R}egexp$ operators.

Example. We can presume that agent *Alice* knows that agent *CS Course Attendance* is interested in the pattern in Figure 5.2[6] . This pattern fully matches the context graph held by agent *Alice*, with the solution $CS\ Course\ Attendance\ (\xrightarrow{within}$ 14:00-16:00 $\xrightarrow{isa} Time\ Interval) \xrightarrow{part-of} Schedule \xrightarrow{of} Alice$. Agent *Alice* will send to *CSCourseAttendance* this solution, thus informing it of Alice's participation to the course.

5.2.4 An Algorithm for Context Matching

In order to perform the operation of context matching, we have devised a matching algorithm. The algorithm returns, for a graph G and a pattern G_s^P, the subgraph(s) G' (G'_i) of G that fully match(es) the pattern G_s^P or, should no such subgraph exist, the subgraph(s) G' (G'_i) of G that k-match(es) G_s^P, for the minimal existing k (the full match has $k = 0$).

The matching algorithm is described in the following paragraphs (see also Figure 5.3). We consider that there is known a node $v_M^P \in V_s^P$ which is one of the nodes with the maximum difference between its out-degree and its in-degree.

First, create a queue $MatchQueue$. $MatchQueue$ will contain parts of the pattern that partially match the G graph, as *matches* – tuples (G'_i, G_{xi}^P) formed of the k-matching subgraph of G and the not matching part of the pattern: $G'_i \subseteq G$ and $G_{xi}^P = (V_{xi}^P, E_{xi}^P)$:
$V_{xi}^P = \{v \in V_s^P, v \notin dom(f)\}$ for f the matching function;
$E_{xi}^P = \{e \in E_s^P$ *for which condition (2) in the matching of pattern* G_i^P *is not fulfilled*$\}$.

Next, for each edge e_k^P in E_s^P that is a non-$RegExp$ edge and that has one or more matches in graph G, add one match (or more matches) $m = (e_r, G_s^P \setminus \{e_k^P\})$ in $MatchQueue$, where e_r is the edge (or one of the edges) in G that match e_k^P, i.e. $value(e_k^P) = value(e_r)$. Edges in $MatchQueue$ are primarily sorted according to distance from v_M^P (ascending) and also according to distance to the closest leaf (also ascending).

The next step in matching is to try to grow the existing single-element matches to cover most of graph G: for each match $m = (G'_i, G_{xi}^P)$ in $MatchQueue$ all edges that are in G_{xi}^P and that are outgoing from $G_s^P \setminus G_{xi}^P$ (i.e. outgoing from the part that already matches the graph) are explored, and matches are attempted with the outgoing edges of G'_i. Matching edges and nodes are

[6] We use the following notation for graphs written in text, using labeled (if the edge is labeled) right arrows, parentheses and stars ("$\star$"): a graph with three nodes A, B and C and two edges, from A to B, and from B to C, is written as $A \to B \to C$; a tree with the root A having two children B and C is written as $A(\to B) \to C$; a graph with three nodes forming a loop is written as $A \to B \to C \to \star A$ (the star is are used because the node A has been previously referred before (and its definition is elsewhere); finally, a graph with two loops ABCA and ABDA is written as $B(\to C \to A \to \star B) \to D \to \star A$ (every edge appears only once).

```
1.  Match(G, G^P) (G = (V, E), G^P = (V^P, E^P)):        // matching a graph to a pattern
2.    MatchQueue ← ∅
3.    v_M^P ← v_i^P
        with max(outdegree(v_i^P) − indegree(v_i^P))
4.    for each e_k^P = (v_i^P, v_j^P, val) ∈ E^P
5.      if val = string or val = URI                    // the value is fixed
6.        for each e_r = (v_s, v_t, val') ∈ E
7.          if (val = val' or val' = ∅)
              and (v_i^P and v_j^P match v_s and v_t)
8.            MatchQueue ← MatchQueue∪
                ({e_r}, {e_k^P, v_i^P, v_j^P}, G^P \ {e_k^P, v_i^P, v_j^P})   // enqueue single-edge match
9.  sort MatchQueue by distance from v_M^P              // best is closest to v_M^P
      and by distance to leafs                          and closest to a leaf

10. while MatchQueue ≠ ∅
11.   m = (G', G_m^P, G_x^P) ← extract(MatchQueue)     // the matching subgraph, the
                                                        matching part of pattern, the un-
                                                        matched part of pattern
12.   for each e_k^P = (v_i^P, v_j^P, val) ∈ G_x^P
13.     if v_i^P ∈ G_m^P                                // edge is outgoing from match
14.       if e_k^P ∈ m^1,                               // the edge is matched
            with m^1 ∈ MatchQueue ∪ Matches
15.           if G_m^P ∪ e_k^P matches G'',             // full match required
                with G' ⊆ G'' ⊆ G
                and m can merge with m^1                // matches are compatible
16.             m' ← merge(m, m^1)                      // merge the two matches
17.             MatchQueue ← MatchQueue ∪ m'            // (keep MatchQueue sorted)
18.       else
19.         find a match m^1 for e_k^P, starting from v_i^P   // v_i^P is already matched in G
20.         if m can merge with m^1                     // matches are compatible
21.           m' ← merge(m, m^1)                        // merge the two matches
22.           MatchQueue ← MatchQueue ∪ m'
23.   for each e_k^P = (v_i^P, v_j^P, val) ∈ G_x^P
24.     if v_j^P ∈ G_m^P                                // edge is incoming to match
25.       [same as for outgoing edges]
26.   remove m from MatchQueue
27.   Matches ← Matches ∪ m                             // keep sorted by k
28. return Matches
```

Figure 5.3: The matching algorithm for a graph and a pattern.

added to m. Edges already in $MatchQueue$ are tried first and, if matching, their match is merged with m. The same is done for incoming edges. If no more edges or nodes can be added to the match, the match is removed from $MatchQueue$ and added to the $Matches$ list.

The algorithm ends when there are no more elements in $MatchQueue$. It returns the list $Matches$, sorted ascending according to the k of the matches.

5.3 Context-Awareness Outside of the Agent

The need for structuring the agent system in a context-aware manner comes from the fact that the agent behavior – developed in AmIciTy:Mi (see Chapter 3) – is based on local interaction and local knowledge. But "local" must not

necessarily be local in space. Other aspects of context may be used. This was first explored in the Ao Dai project – see Chapter 4 – where the agent system was hierarchized in order to reflect the hierarchical nature of two types of context – space and computational elements.

This section will build upon the ideas in the Ao Dai project, but extending the hierarchic structure to more types of context, in order to obtain an context-aware topology for the agent system.

In the context of a decentralized and scalable agent system, one challenge is with whom should agents communicate. That is, in the topology of the agent system, who should be the neighbors of an agent?

The agent system relies on the context-aware exchange of information between an agent and its neighbors. But an agent should only exchange information with another agent that may consider that information as relevant. And that can happen only if the agents share some type of context. For instance, if the agents are part of the same activity, or if their users are located in the same space. Two agents that share no context would not have any information that is relevant to both of them.

Therefore, **the topology of the system should be induced by context**: if two agents share context, they should be neighbors. This creates a topology that is an overlay over the actual topology of network that interconnects the agents. We can consider that the underlaying network allows direct connections between any two agents (i.e. the connection graph is complete).

5.3.1 Hierarchizing the Different Aspects of Context

Chen and Kotz have identified four types of context [Chen and Kotz, 2000]: computational context – available computing and networking resources, including the cost for using them; user context – user's profile, location, people and objects nearby, social situation; physical context – light and noise levels, temperature, traffic conditions, etc; and time context – the current time coordinate of the user and related information (like the season, for instance). Further, in the work of Henricksen and her colleagues [Henricksen et al., 2002, Henricksen and Indulska, 2006], activity context bears a major role. Consequently, **we deal with 5 types of context: spatial context, computational context, temporal context, activity context, and social context**.

Working with specific aspects of context has two important consequences. On the one hand, we can create specific types of agents and of inter-agent relations, allowing for specific behaviors depending on the common shared context. On the other hand, all of these aspects of context have, to some extent, a hierarchical structure (see Section 4.1.2), and therefore we are able to use the mechanisms that are offered by CLAIM for hierarchical agent mobility. However, as – differently from the Ao Dai project – now we have multiple hierarchies (one for each aspect of context) that are intertwined, we need to extend CLAIM (see Chapter 6) in order to support "secondary parents".

Agent type	Possible incoming relations (and their sources)	Possible outgoing relations (and their destinations)
Place	*is-in* (from *Activity*, *User*, *Device*, *Service*, *Place*	*is-in* (toward *Place*)
Activity	*part-of* (from *User*, *Group*, *Activity*, *Service*)	*of* (toward *User*), *part-of* (toward *Activity*)
Device	*executes-on* (from *Service*)	*is-in* (toward *Place*), *controlled-by* (toward *User*)
Service	-	*executes-on* (toward *Device*), *is-in* (toward *Place*), *part-of* (toward *Activity*)
User	*controlled-by* (from *Device*), *of* (from *Activity*), *connected-to* (from *User*)	*part-of* (toward *Activity*), *in* (toward *Group*), *connected-to* (toward *User*)
Group	*in* (from *User*)	*part-of* (toward *Activity*)

Table 5.2: The possible relations between different agent types, resulting from mapping of context to system topology.

5.3.2 Agent Types and Relations

As in the Ao Dai project, we keep different agent types for each aspect of context. That is, an agent will be primarily interested in information relating to one aspect of context, for instance related to a place, to a user, etc.

We have defined the following types of agents and of neighborhood relationships, to match the different aspects of context (a more organized perspective on the relations between agents is presented in Table 5.2; a graphical representation of example agent topologies using these relations is presented in Figure 5.4):

- for spatial context – the *Place* agent and the *is-in* relation;
- for computational context – the *Device* and *Service* agents and the *of* (linking to the agent to which the offering of the service is related, like a place) and *executes-on* relations for services and *controlled-by* (linking the device to the user that controls it) for devices;
- for activity context – the *Activity* agent and the *part-of* relation;
- for social context – the *User* and *Group* agents and the *in* (linking a user to a group) and *connected-to* (linking two users) relations.

While temporal context is an aspect of context that we consider, we do not have a specialized agent for time intervals (which would be the hierarchical element of temporal context), as an agent that manages a time interval does not make much sense: since the internal context representation, as well as the relations between agents, happen in the present – therefore shared temporal context is already achieved. However, should specific indications of time intervals be needed – for instance in the case of activities, availability of devices or services, and temporary user groups – this information may be attached to the agent's knowledge.

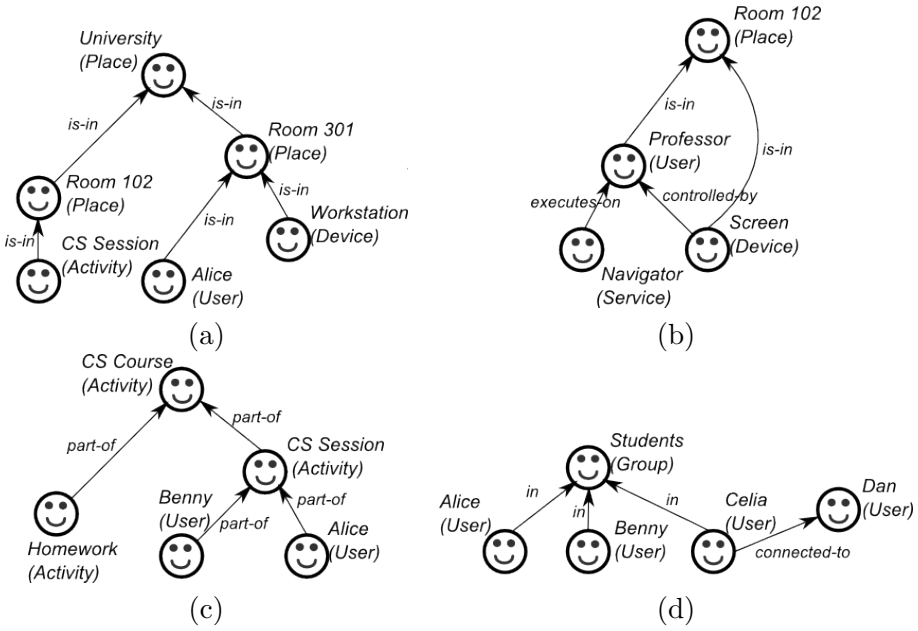


Figure 5.4: Examples of possible agent topologies. The examples are centered around different types of context: spatial, computational, activity, and user / social.

5.4 An Improved Agent Behavior

Although the AmIciTy agent behavior presented in Section 3.2.3 assures the coherent spreading of information in the agent system, controlled by context measures, both the agent topology and the context measures that were used were fairly simple. In the preceding sections of this chapter, we have formalized an improved manner of handling context information, as well as an improved agent topology.

5.4.1 Principles

The principles that the improved agent behavior is based on are the ones that have been presented throughout this work:

- an agent should send pieces of information it is interested in to neighbors that the agent believes may be interested in those pieces of information (Section 3.1);
- the discriminating factor between external knowledge and contextual knowledge is the focus (Section 5.2.1);
- the topology of the system is induced by context: if two agents share context, they should be neighbors (Section 5.3).

5.4.2 Description

As stated in Section 5.1, an agent A is modeled as a tuple
$A(Name, CG_A, Patterns, \mathcal{R}, I, Goallist)$ with the following components:

$Name \in AgentNames$, the name of the agent;
$CG_A = (V, E)$, where $V \subset Concepts$
and $E = \{edge(from, to, value, persistence) \mid from, to \in Concepts, value \in Relations, persistence \in (0, 1]\}$, the *Context Graph* of the agent where elements of concepts and relations are strings or URIs; *Relations* also contains them empty string, for unnamed relations; note that $AgentNames \subset Concepts$;
$Patterns = \{(G_s^P, relevance, persistence) \mid s \in PatternNames, G_s^P$ a graph pattern, $relevance, persistence \in (0, 1]\}$, the patterns of the agent (see Section 5.2.2 for details);
$\mathcal{R} \subset (AgentNames - \{Name\}) \times AgentRelationTypes \times \{in, out\}$, all relations with other agents;
$I = \{(Agent, s, factor) \mid Agent \in AgentNames, s \in patternNames, factor \in (0, 1]\}$, the observed interests of other agents for different patterns;
$Goallist = \{Goal(G', G_s^P, importance) \mid G' \subset CG_A, importance \in (0, 1], G_s^P \in Patterns\}$, the sharing goals of the agent, which contain a certain piece of information (subgraph of CG_A) that matches a pattern.

All agents are defined by three essential behaviors: **pattern matching** (situation recognition and pro-activity related tot the user), **information sharing** (pro-activity related to other agents), and **information integration** (reactivity). These behaviors are described in pseudo-code in Figure 5.5. The agent behavior is identical to the behavior presented in Section 3.2.3, just that instead of working with specialties and pressures, now the agent works with context patterns and context matching.

An important process that is not illustrated in the algorithms is the removal of parts for the context graph. When new edges are added, their *persistence* is set according to the indications of the pattern that contains the edge. With time, persistence of the edges in the context graph of the agent (CG_A) fades, and as it reaches zero, the edge is removed (along with any resulting isolated nodes).

All actions that an agent can take are related to pattern matches and added edges (creation of relations between concepts). The actual actions that are connected to the addition of edges may also have associated with them some special procedures, allowing the agent to actually change its environment accordingly. This is why the only edges that are "actionable" should be the ones that correspond to effects that the agent can actually create.

From the perspective of the multi-agent system, a special case of action is the inference and creation of relations between agents. According to its patterns, an agent can infer that in some case a relation should exist with another agent (inter-agent relations are one of *is-in, part-of, of, in, controlled-by, executes-on, within*), by effectively adding an edge in CG_A between the concepts representing itself and the other agent. Should that be the case, the agent will also inform the management of the multi-agent system (as well as the other agent) of this new relation. The same happens when the edge disappears.

```
· Pattern Matching for agent A
1. foreach (G_s^P, relevance, persistence) ∈ A.Patterns
2.   match G_s^P against CG_A
3.   obtain M_A-si(G'_A, G_m^P, G_x^P, f, k)
4.   if k > matching_threshold                          // match is considered valid
5.     if k < 1                                         // incomplete match
6.       while ∃e^P ∈ E_x^P with
           e^P.value has no Rexgexp operators           // e^P.value ∈ Relations
           and e^P.actionable > actionability_threshold
           and (e^P.from ∈ V_m^P or e^P.to ∈ V_m^P)     // at least on matched end
           and (e^P.from ∈ V_m^P or e^P.from ≠?)        // the unmatched end ...
           and (e^P.to ∈ V_m^P or e^P.to ≠?)            // ... is not undetermined
7.         if e^P.from ∉ V_m^P                          // insert corresponding nodes
8.           insert e^P.from ∈ CG_A and adjust M_A-si
9.         if e^P.to ∉ V_m^P
10.          insert e^P.to ∈ CG_A and adjust M_A-si
11.        insert in CG_A new edge(e^P.from, e^P.to,    // insert actioned edge
             e^P.value, persistence)
12.        adjust M_A-si
13.    (re)insert (G'_A, G_s^P, k · relevance) in Goallist   // sorted by importance

· Sharing for agent A
1. while Goallist.size > 0
2.   extract first (G'_A, G_s^P, importance) from Goallist
3.     potentials =                                     // potentially interested agents
         {(Ag, s', factor) ∈ I_A | s' = s, factor > 0}
4.     sort potentials by factor
5.     fraction = importance · potentials.size          // more important information...
6.     send G'_A to first fraction agents in potentials // ... gets sent to more agents

Receiving for agent A
1. receive G'_Ag from agent Ag
2. foreach (G_s^P, relevance, persistence) ∈ A.Patterns
3.   match G'_Ag against G_s^P                          // is the important to agent A?
4. obtain match M_Ag'-si(G'_Ag, G_m^P, G_x^P, f, k) with best k // best k is greatest k
5. if k > matching_threshold
6.   add / adjust (Ag, s, k) in A.I                     // check agent Ag's interests
7.   match G'_Ag against CG_A
8.   obtain match M_A-Ag i(G'_A, G_Ag-m, G_Ag-x, f, k)
       with best k
9.   if k > matching_threshold
10.    foreach edge ∈ G'_Ag-x.E'
11.      integrate edge in CG_A                         // add the new information
12.      edge.persistence = persistence                 // get indication from pattern
13.      foreach edge ∈ G'_A.E'
14.        edge.persistence =                           // refresh persistence indication
             max(edge.persistence, persistence)
```

Figure 5.5: Pseudo-code for the behavior of context-aware agents.

5.4.3 An Extended Example

In order to give a better feel about how the context-aware behavior for agents works, let us return to our scenario, presented in Section 1.3.4.

The scenario describes the following situation: Alice, a user of AmI services, is on a train, on the way to attending a session of the Computer Science course

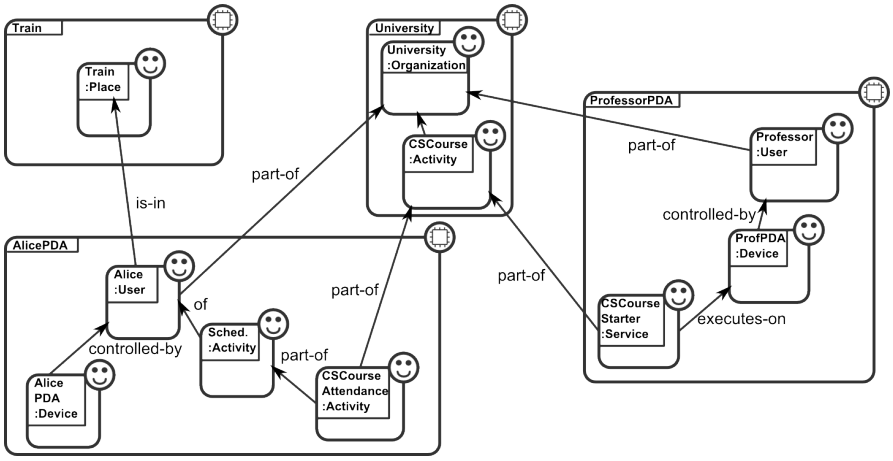


Figure 5.6: Agent topology for the scenario from Section 1.3.4. Agents running on the same machine are circled accordingly.

at the University at which she is a student. Alice's train will be a few minutes late. Concurrently, the Professor that is going to teach the course is waiting for the course to begin, using the CourseStarter service to be notified which students will be late, and for how long.

Considering an agent assigned to each element of context – devices, services, users, places, activities – the topology of the system is presented in Figure 5.6. Having this topology, when the information that Alice will be late is generated by her *Scheduler* (having used information sent to *Alice* by *Train*), it will inform the agent managing her CS Course attendance, which will disseminate this information to *CSCourse*, which will inform the *CourseStarter*. This way, the information spreads among agents having a common context, here the common context being mainly related to activity. Considering the agent behavior described in Section 5.4, there may be other agents receiving this information as well, but they will discard it as the information is not relevant to them.

But how will the agents know what information to send? This is where context patterns come in. We can safely presume that the *Train* agent knows that, among others, the user Alice is on the train, therefore having $Alice \xrightarrow{is\text{-}in} Train$ #1691 as a subgraph of CG_{Train}. It is also reasonable to believe that the *Train* agent has a pattern $TrainRide(\xrightarrow{on} Train$ #1691$)$ $(\xrightarrow{to} ?Place)$ $(\xrightarrow{within} ?Time\ Interval)$ $\xrightarrow{(part\text{-}of\ |\ of)^*} ?User \xrightarrow{is\text{-}in} Train$ #1691). This pattern matches the information about the time interval of a train ride that is part of the activities of a specific user (and has a specific destination). Domain-specific information and algorithms will allow the train to estimate the time intervals for the destinations of all the users. Then, this information will be sent to the users that are potentially interested – that is, each match will be sent to the appropriate user.

Having this new information, Alice's agent will update its own context graph, with the result displayed some pages ago, in Figure 5.1. Now take the pattern in Figure 5.2, which is one in the set of patterns of agent *CS Course*

Attendance. CG_{Alice} matches this pattern. Since Alice's agent knows that *CS Course Attendance* is interested in this pattern, it will send to it the information. Similarly, the information will be disseminated until it reaches agent *CourseStarter*.

5.5 Lessons Learned

This chapter presents a new, holistic approach to context-awareness in a decentralized multi-agent system for Ambient Intelligence. This approach has two main aspects: first, situation recognition and problem solving by means of graph patterns that match the context graph – a graph which represents the context information for the current situation of the agent.

There are several advantages to this proposal: using graphs, both for the agent topology and for the representation of context information and context patterns, visualization of this information is easy to understand, and can be easily followed when context changes (unlike propositional logic for instance). It is easy to notice how patterns match context graphs, for instance.

Another advantage is that using graphs means the possibility of using all already existing (and well researched) graph algorithms, especially for the graph matching problem.

Although graphs are fairly simple structures, using them to represent knowledge can be effective and powerful, especially as, although a certain range of predefined relations are used, new relations can be added to context graphs and to context patterns.

5.6 Perspectives

We believe that our approach has a lot of potential, that has not yet been entirely explored. Both context-awareness outside and inside the agent are new contributions, therefore it is only intensive testing that will fully validate their adequacy to various applications.

Temporality has only been little explored. User's (or agent's) history, as well as planned activities, could be formalized as special elements and treated as such.

A path to explore would be the creation of relations between peers, or sibling agents. This would help the system's decentralization, and would lower the load on a parent with many children.

Another aspect that needs a testing in practice and non-simulated applications is the use of the *characteristic* and *actionable* features of patterns, which have the power to greatly increase the possibilities of applying patterns.

Chapter 6

A New Platform for AmI Applications

In order to be able to test the agent-based model that we have developed, an implementation is necessary. Although some parts of this work were tested using simple or partial implementations of the system, the final result would need, on the one hand, to integrate all the theoretical concepts that we have presented and, on the other hand, to feature functionalities like distributed communication, mobility and centralized visualization of the agents.

This chapter gives details on the implementation of the Ao Dai platform (which is different from the Ao Dai prototype). The realization of the platform has been a collaborative effort of Andrei Olaru, Thi Thuy Nga Nguyen and Marius-Tudor Benea, under the supervision of prof. Amal El Fallah Seghrouchni and with the assistance of Cédric Herpson[1].

6.1 Why Build a New Platform?

There were several reasons for which the Ao Dai platform has been created. First, both the teams in Bucharest and Paris were in need of a platform for the deployment and testing of AmI applications. Second, the Ao Dai prototype showed that the CLAIM language and hierarchical mobility have great potential for use in context-aware applications. However, the CLAIM language was overly complicated, it was somewhat difficult to read, the Sympa platform used mobility and communication primitives that were hard to interoperate with other platforms, and the implementation was generally not modular and difficult to extend. Extension was needed as interoperation was required with web services, an improved knowledge representation was going to be used, and there was a need to be able to implement new types of agents on top and besides CLAIM agents.

[1] Nga is currently a PhD student in cotutelle between University Pierre et Marie Curie (Paris 6) in France, and IFI Hanoi, in Vietnam. Tudor was a master student at University Politehnica of Bucharest, on internship at Paris 6. Cédric is a PhD student at Paris 6.

The features that were required from the new platform were the following:

- the use of CLAIM or a similar language for the high-level implementation of agents – this meant either working with the existing code for parsing and interpreting CLAIM definitions, or writing a new one, which meant that we could also bring modifications to the specifications of the language;
- a modular and extendable structure that allows for easy future development of new modules or improvement of existing modules – that is, a clear organization of the code in a hierarchy packages, and loose coupling of components;
- an improved representation of knowledge (with respect to CLAIM) that is easily translatable from and to XML and other serializable formats – because of the need to store knowledge or to load knowledge from disk, and also to display knowledge in a user-readable form; also support for interchangeable knowledge formats was necessary, so that for instance some agents can use propositional logic, others can use knowledge maps, etc;
- potentially deployable on mobile devices – as the platform would be directed toward Ambient Intelligence applications, smartphones and other mobile devices are key components of any AmI environment; this also meant, among others, that the platform would have to implement (or use a framework that implements) standard communication primitives;
- good possibilities of traceability and visualization – as we have learned from the AmIciTy:Mi project (where we had such tools) and from the Ao Dai prototype (implemented in CLAIM, which did not offer such tools) it is very important and useful to be able to visualize and follow the evolution of the system as a whole and of individual agents, at varying levels of detail;
- the use of scenario-based simulation – that is, the existence of a machine-readable format for scenarios and of components that use scenarios written in this format to produce simulations that are repeatable and do not need input from the user (all input is generated based on the scenario), as in the AmIciTy:Mi project;
- the possibility of integration with other platforms and protocols – that is, the use of standard formats and protocols (e.g. FIPA-compliant), and the possibility of communicating with the agents from the exterior of the platform, through standard protocols (e.g. web services).

The new platform was designed and built considering these requirements. However, the platform has not been implemented from the ground up. Existing components that fitted the requirements were used, with the purpose of increasing modularity and also for easier understanding of the platform by others.

For instance, the new platform is underpinned by JADE – the Java Agent Development Framework – which is a popular and easy-to-use environment for the deployment of multi-agent systems. Moreover, the CLAIM language was not totally discarded. We have worked with a simplified and improved semantics, while keeping its theoretical fundaments intact.

6.2 Architecture and Components

In order to make the architecture of the platform easy to extend and improve, we have adopted a modular approach, based on several components and layers, as follows:

- the *Core* component, containing the classes for agents, organized on several layers:
  - agent communication, mobility, and management – JADE agents are used;
  - agent logging and visualization – agents must send data about their activity, as well as their neighbors, to a centralized entity. Each agent also displays a window on the screen of its execution host;
  - hierarchical mobility for agents – protocols and behaviors that allow agents to automatically move together with their parents;
  - web service access – functionality to expose agents as web services and allow agents to access web services;
  - S-CLAIM interpretation and execution – a parser for S-CLAIM agent description files, and the components that transform the definition into actual behavior;
  - Knowledge Base – an interchangeable component that allows access to knowledge through a standard set of functions;
  - context-awareness – use of context matching for problem solving and exchange of relevant context information.
- the *Simulation* component, serving for the repeatable execution of scenarios:
  - uses as input XML files that completely define the execution scenario;
  - deploys the agents according to the scenarios, on the specified containers and machines;
  - sends "external" event messages, equivalent to perceptions of agents in a real environment;
- the *Visualization* component, that assures the centralized visualization of agent activity:
  - receives log reports and mobility events from agents;
  - displays all agent logs in a centralized, chronological manner;
  - displays the system structure (topology), showing relations between agents;
  - provides components for the automatic layout of agent windows on the screen of the machine they execute on.

An informal view of these components is presented in Figure 6.1. In the following sections we will give details on the building blocks for the platform and on the S-CLAIM agent language.

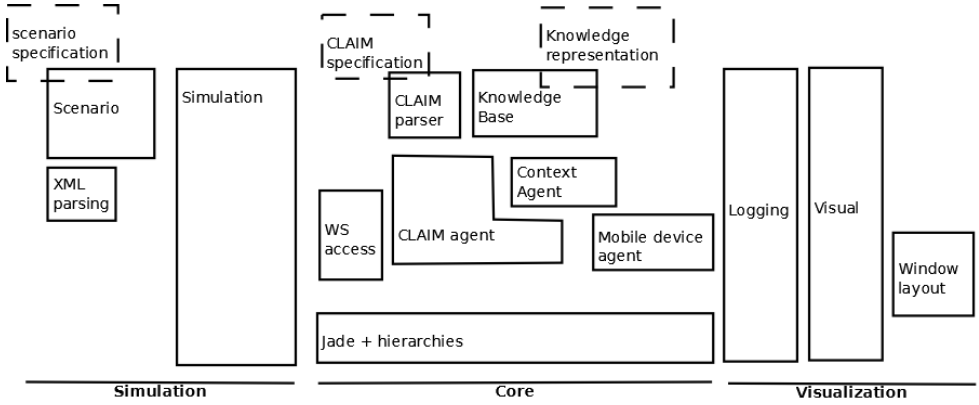


Figure 6.1: An informal visual representation of the components of the Ao Dai platform. With solid border, actual components of the implementation. In dotted line, specifications and formats that characterize the inputs of the components.

6.2.1 Building Blocks

A first building block of the Ao Dai platform was the JADE Agent Development Framework[2] [Bellifemine et al., 1999, Bellifemine et al., 2001]. It is implemented in Java. It can be deployed on multiple machines, agents can join or leave at any time (open system), and it handles agent communication, mobility and management. Being built upon Java, JADE ca be easily deployed on different platforms. Third-party modules allow the deployment of agents on mobile devices (for instance Android smartphones or tablets) and interoperability with web services.

For logging the agent's activity, we have developed a wrapper for Apache's log4j[3], that allows for quick configuration of a log that shows the output to the console, displays it in a window, and sends it (as a JADE message) to the visualization agent simultaneously.

A component has also been developed for the deployment of S-CLAIM agents on mobile Android devices[4], using the JADE-LEAP add-on[5]. This add-on allows the execution of a JADE container on a devices running the Android operating system, enabling the execution of JADE agents on the device, and the movement of agents to and from the device, practically seeing the Android device in the same way as a workstation. Changes were, however, necessary, to the visualization component on the Android devices, due to reasons of platform compatibility, screen size, etc.

For the interoperability of Ao Dai agents with other components and infrastructures, the agents integrate primitives for both the access of SOAP and RESTful web services, and for the exposure of agent capabilities as web services. This has been done with the help of the WSIG and WSDC JADE

[2] http://jade.tilab.com/
[3] http://logging.apache.org/log4j/
[4] http://www.android.com/
[5] http://jade.tilab.com/community-addons.php

CourseAgent.adf2 agent definition file

```
1. (agent Course ?courseName ?parent
2.   (behavior
3.     (initial register
4.       (send ?parent (struct message managesCourse this ?courseName))
5.     )
6.
7.     (reactive registerUser
8.       (receive assistsUser ?agentName ?userName)
9.       (addK (struct knowledge userAgent ?userName ?agentName))
10.    )
11.    ...
12.    (reactive changeRoom
13.      (receive managesRoom ?roomAgentName ?roomName)
14.      (condition (readK (struct knowledge scheduling ?courseName ?roomName)))
15.      (addK (struct knowledge roomAgent ?roomName ?roomAgentName))
16.      (forAllK (struct knowledge userAgent ?userName ?userAgentName)
17.        (send ?userAgentName
             (struct message scheduling ?courseName ?roomAgentName))
18.      )
19.      (in ?roomAgentName)
20.    )
21.  )
22.)
```

Figure 6.2: Sample of S-CLAIM code, used in the Ao Dai platform scenario for the *CS Course* agent.

add-ons.

The window layout component – for the layout of various windows (for instance each agent has its own window) across the screen – has been ported from the AmIciTy:Mi project. The advantage of using an automatic layout tool for windows is that, in the case of many windows, it is not the user who needs to manually move each window to the desired position, at each run of the simulation.

6.2.2 S-CLAIM

Our main goal while creating S-CLAIM ("Smart CLAIM") was to make it as easy to use as possible, so that it could be a great tool even for those which were not very familiar with other programming languages. This was in fact also the goal of CLAIM (Computational Language for Autonomous, Intelligent and Mobile agents) – to be an agent-oriented programming language that was simple to use by agent designers.

However, the semantics of CLAIM are trying to cover a large range of functionality, but are still not powerful enough to allow for the complete description of the agent's algorithms or visual components. Therefore, for anything more than the simplest examples, the developer must implement some functionality in Java anyway. Moreover, the syntax of CLAIM is not very easy to read. For these reasons, and also because the Sympa execution platform for CLAIM agents was using non-standard means of communication and agent mobility

and was hard to expand or modify, we have chosen to redesign CLAIM and to reimplement the agent execution platform (this time using JADE).

Semantics

The semantics of S-CLAIM are closely based on the CoCoMo semantics developed by Abdelkader Behdenna, during his Master internship at LIP6 in 2009. They are a simplification of the semantics of the CLAIM language [Suna and El Fallah Seghrouchni, 2004], which reduces the list of primitives to only the ones which are characteristic to agent management and interaction. S-CLAIM specifies the following primitives (classified by their destination) – more on the syntax of these primitives unde the "Syntax" section:

- **communication:**
  - *send* – send a message to another agent, reply to a message or access a web service;
  - *receive* – receive a message from another agent, or receive an invocation as a web service;
- **mobility**:
  - *in* – become a child of another agent; if the agent is not bound to its container, the agent will also move, together with its hierarchy;
  - *out* – exit the context of the parent agent;
- **agent management**:
  - *open* – order the dissolution of a child agent, recovering all the capabilities and knowledge of the child agent;
  - *acid* – dissolve itself into the parent agent, the parent recovering all the capabilities and knowledge of the agent;
  - *new* – create a new agent, that becomes a child of the creating agent;
- **knowledge management**:
  - *addK* – add a piece of knowledge to the knowledge base;
  - *removeK* – remove the piece of knowledge matching a pattern;
  - *readK* – read the first piece of knowledge matching a pattern, filling the unspecified parts of the pattern;
  - *forAllK* – iterate over all the knowledge that matches a certain pattern, at each iteration executing a series of statements, using the selected piece of knowledge;
- **control primitives**:
  - *condition* – condition the activation of a behavior on a logical operation or the result of a boolean function;
  - *if* – condition the execution of a block of statements;
  - *wait* – interrupt the agent behavior for the specified amount of time;

The semantics of S-CLAIM includes only the primitives that involve agent management, knowledge management or agent communication, plus a strict selection of control primitives. This leaves out all algorithmic parts of the agent's description, processing functions, arithmetic or logic operation, etc. These can be implemented in other, classic, programming languages (only Java is supported for the moment), and can be invoked in the same manner as S-CLAIM-specific primitives.

Syntax

The syntax of S-CLAIM uses parentheses, like LISP, in order to make the code cleaner and simpler. Variables in S-CLAIM are preceded, as in CLAIM, by a question mark. An example of S-CLAIM code is presented in Figure 6.2.

A usual agent definition contains the name of the defined type of agent, the parameters for the agent, and its list of behaviors. For the moment three types of behavior are specified:

- *initial* – executed at the creation of the agent;
- *reactive* – executed as consequence of receiving a message and, optionally, fulfilling some conditions;
- *proactive* – goal-oriented behavior that executes without the need external events.

For instance, an agent that only has one initial behavior would be defined as

```
(agent SimpleAgent ?destination
  (behavior
    (initial sender
      (send ?destination (struct message hello))
    )))
```

S-CLAIM uses structures as sole form of data structuring. A structure is build by using parentheses, the keyword *struct* and the list of members of the structure. Usually the first member of the structure represents the type of the structure. Communication and knowledge management primitives, for instance, assume that the structures that are received or passed as parameters have the *message*, and *knowledge* type, respectively.

The syntax of primitive invocation is LISP-like as well. It is formed by the enclosing parentheses, the name of the primitive or function, and the parameters. While the language is not typed, each S-CLAIM primitive makes assumptions about the type of its parameters. For instance, the *send* primitive assumes that its second argument is a structure of type message.

The S-CLAIM language relies in a great measure on the use of patterns. For instance, the *receive* primitive takes a set of parameters that are expected in the received message, of which some may be bound to values and some may not. While the bound parameters impose a certain form to the message, the unbound parameters will be bound to the values in the message present at those positions. In the same manner, the *readK* and *forAllK* primitives work with patterns, on the one hand selecting only some of the agent's knowledge – the pieces that match the pattern – and on the other hand binding the unbound variables to the specific values that correspond to the selected piece of knowledge.

As an example of S-CLAIM code, take the sample in Figure 6.2. It is easy to understand: initially, the agent registers by sending to its parent information about what course is managed by the agent (the name of the course, as well as the parent agent, are given as parameters). A reactive behavior for user

registration is activated when a message with this purpose is received, and in this case the agent stores the information in its knowledge base. Finally, another reactive behavior is activating in case of a change of rooms, i.e. a message about the room for the course is received, and the agent had already been scheduled to a different room; consequently, the information is updated, all the users that will participate in this course are notified, and the agent becomes a child of the agent managing the room.

Java functions

While S-CLAIM is a programming language, it does not try to reinvent what existing programming languages already do very well – it only proposes primitives that are directly related to the agent, it's components, and the other agents in the system. These primitives can be used inside agent descriptions that make it easy for the agent designer to focus on the agent-related features.

There are processes that cannot be easily performed with the default primitives. For instance, algorithmic processing (e.g. list processing) or arithmetic functions. This is why the developer can attach one or more Java class files that contain the functionality, and call the functions in exactly the same way as S-CLAIM primitives. For example, if arithmetic comparison is needed, a very simple Java function will be attached, called *gt*. It will be invoked in S-CLAIM as:

```
(if (gt ?number 5)
...
)
```

All S-CLAIM-attached Java functions have the same prototype: they take a *Vector* of values, some of which may be unbound. When the function ends, some of the unbound values may be bound, and in this way the function is able to fill missing pieces in patterns. The function also returns a boolean value, that can be used in the S-CLAIM code in *if* or *condition* statements. This behavior is similar to predicates in Prolog.

6.2.3 Scenarios

One of the features that is central to being able to correctly test a platform like Ao Dai is the possibility to execute scenarios easily and repeatedly. This means, on the one hand, that once the test scenario is configured, running the scenario can be done at the touch of a button. On the other hand, it means that it is possible to run the exactly same scenario over and over again, with the same results.

These requirements have been developed when building AmIciTy:Mi , and the solution for Ao Dai is similar. The scenario is specified by means of an XML file that contains information on the containers and agents to be created, on the initial knowledge of agents, and on the events to generate. Part of the specification of the scenario is presented in Figure 6.3. At this point, each machine that should be part of the experiment should run its own scenario

scenario.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <scen:scenario ... >
3.    <scen:jadeConfig ... />
4.    <scen:adfPath>scenario/phase2</scen:adfPath>
5.    <scen:initial>
6.      <scen:container name="Administration">
7.        ...
8.        <scen:CLAIMAgent name="CourseCSAgent" type="CourseAgent">
9.          <scen:parameter>
10.           <pr:param name="parent" value="UniversityUPMCAgent" />
11.           <pr:param name="courseName" value="CSCourse" />
12.         </scen:parameter>
13.         <scen:knowledge>
14.           <kb:relation relationType="hasUser">
15.             <kb:node>CSCourse</kb:node>
16.             <kb:node>Alice</kb:node>
17.           </kb:relation> </scen:knowledge>
18.       </scen:CLAIMAgent>
19.     </scen:container>
20.
21.     <scen:container name="RoomContainer">
22.       <scen:CLAIMAgent name="Room04Agent" type="RoomAgent" />
23.     </scen:container>
24.     ...
25.   </scen:initial>
26.   <scen:timeline>
27.     <scen:event time="2000" >
28.       <scen:CLAIMMessage>
29.         <scen:to>SchedulerUPMCAgent</scen:to>
30.         <scen:protocol>newSchedule</scen:protocol>
31.         <scen:content>( struct message newSchedule
              ( struct knowledge scheduledTo CSCourse Room04 ) )</scen:content>
32.       </scen:CLAIMMessage> </scen:event>
33.     </scen:timeline>
34.   </scen:scenario>
```

Figure 6.3: Sample of scenario specification, from the Ao Dai scenario.

file. On remote machines however, the only information in the scenario files is the name of containers to be created on those machines. The features that are specified in the scenario file are the following:

- JADE configuration – sets details like the name of the main JADE container, the IP address and port of the platform and of the local machine, and the id of the platform;
- path for the agent description files and packages with Java code attachments;
- containers – names of the containers to be created on the local machine, and, in the case of the machine with the main container, the names of remote containers where agents will be created;
- agents – inside the description of each container, the agent creation information is specified: agent name, agent type (name of the description file), initial knowledge of the agent, code attachment, agent parameters, gui information;
- event information – timeline specifying what events to trigger at what

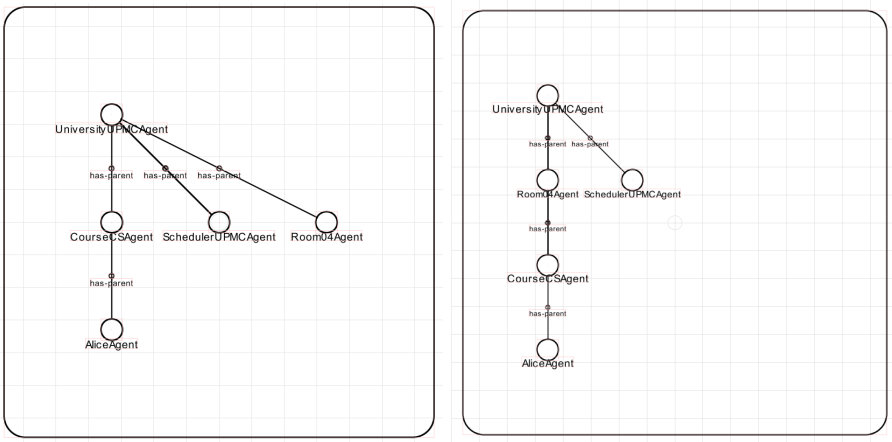


Figure 6.4: Sample visualization of agents in the Ao Dai scenario, before and after the hierarchical movement of the *CourseCS* agent as a child of *RoomAgent*.

moments of time (in simulation time). Events are messages that will be sent to the specified agents.

At the beginning of the simulation, the platform is booted on each machine, and empty containers are created on the appropriate machines (according to local scenario files). Then, a special agent called the *Simulation Agent*, running on the main container, creates all agents that are specified in the scenario, and instructs them to move to their assigned containers. Then, the simulation timer is started, and events are generated by the simulation agent, by sending messages to the specified agents, according to the scenario file.

6.2.4 Visualization

The visualization component of the platform offers three features: centralized logging, centralized visualization of the agent system, and an automated window layout tool for the layout of windows across the screen. The window layout feature is ported from the implementation of AmIciTy:Mi . The automated layout allows for the agents' windows to be placed on the screen in non-overlapping areas, saving the tester the time of moving the windows by hand. An example of layout, shot during the running of the test scenario, is presented in Figure 6.5.

Centralized logging is obtained by sending, from the visualization layer of each agent, and through the wrapper for log4j, the logging messages of the agent, to a specialized *Visualization Agent* that sorts the log messages by their timestamp and displays them in order. The log messages are sent by chunks at a fixed time interval, in order not to overload the network.

Centralized visualization of the system's topology is obtained by sending from each agent to the Visualization Agent messages that specify the location and parent of the agent. This data is aggregated into a graph, that is then displayed using the same algorithm that is used for the linear textual display of graphs

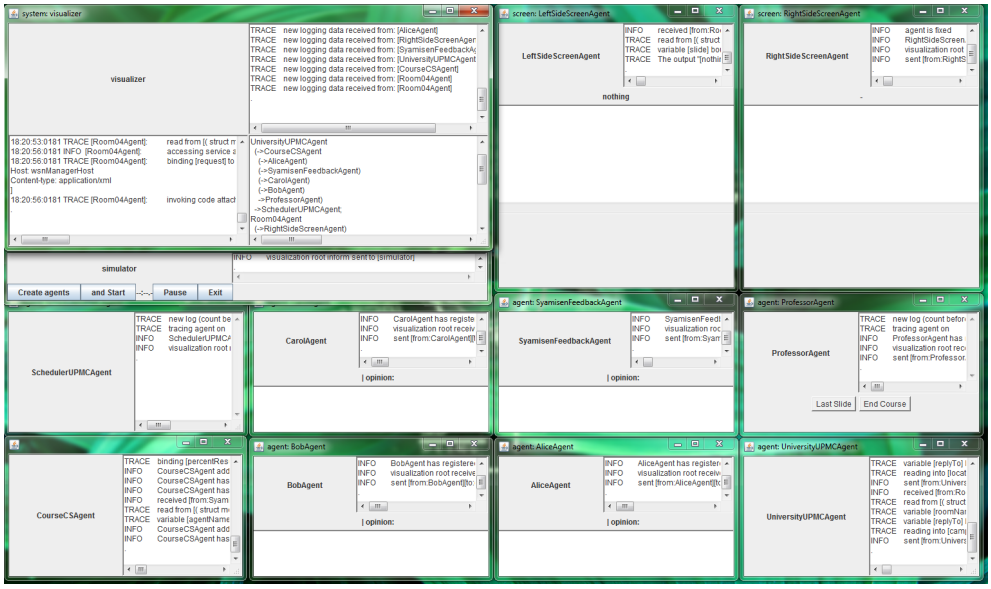


Figure 6.5: Visualization of agent windows, layed out across the screen of the local machine. The *Visualization* and *Simulation* agent use specific windows. Some agents feature input or output areas, beside their agent log.

(as in the example in Section 5.2.3). For instance, the visualization of the initial and final states of the system (considering the scenario from the previous section) is presented in Figure 6.4.

6.2.5 Web Service Integration

A multi-agent system for Ambient Intelligence is only a component in a complete AmI environment. It is necessary for such a system to interoperate with other components, most importantly with perceiving and acting components, and with user interfaces. While there are many means of communication between software components, one of the most used is web services. It allows very loose coupling between the two parties, and is relatively easy to implement.

This is why we have chosen to integrate web services into the Ao Dai platform, in two aspects. First, to make agents and their capabilities available to other components, as web services. Second, to allow agents to contact other parties by means of web services.

In order to keep the simple syntax and semantics that we have devised for S-CLAIM, web service access was integrated using the same constructs, with very slight modifications.

All S-CLAIM agents expose their reactive behaviors. This is done with the help of the WSIG add-on for JADE. In the future, the agent will be able to select which behaviors should be available as web services. The agent exposes itself as one service, and each reactive behavior is available as an operation associated with the service. If the behavior has been invoked as a web service, the agent replies to the web service invocation with a message, just the same

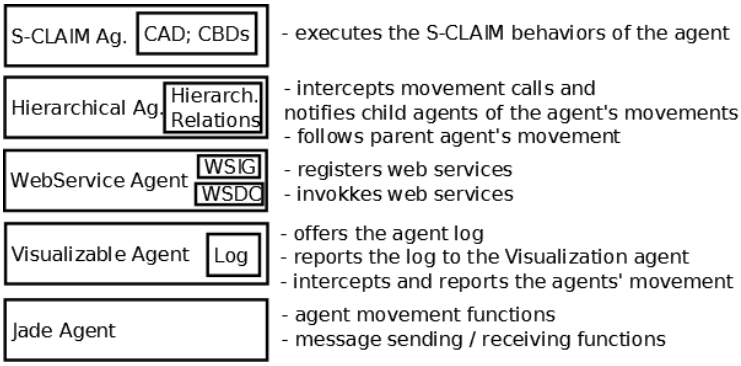


Figure 6.6: Internal layered structure of an Ao Dai agent.

as it would reply to a normal S-CLAIM message. In fact, there is no difference in the S-CLAIM code between an activation of the behavior by means of a message or an invocation by means of web services. For instance, the following code will echo the parameter of the request:

```
(reactive echoService
  (receive echoService ?echo)
  (send (struct message echoing ?echo))
)
```

All S-CLAIM agents are also able to invoke web services, almost in the same way as they would send and receive messages, with some differences, as web services are synchronous. Web service invocation is done by means of the WSDC add-on for JADE. For instance, the code snippet below invokes the behavior defined above, from another agent, by means of web services instead of S-CLAIM/JADE messages:

```
(send ?testAgent
  (struct message echoService helloWorld)
  http://localhost:8080/wsig/ws/
  (struct message echoing ?echo)
)
```

The difference from a S-CLAIM message is that the URL of the service must be specified, and a structure for receiving the immediate reply is included. The last argument is not needed if the reply is not considered of use.

6.2.6 Agent Structure

In the Ao Dai platform, agents are internally organized on several layers. This is useful for the modularity of the system, for the separation of concerns and for the ease of debugging. Moreover, new layers of the agent can be added without much effort.

At this point in the development of the platform, the agent consists of five layered components, that are named after their corresponding Java classes in the implementation (see also Figure 6.6):

- JADE GuiAgent – the basic JADE agents that can have a Gui. This layer manages agent movement and communication at the level of the JADE platform, and is the base of the entire structure of Ao Dai agents.
- VisualizableAgent – it handles agent visualization from two points of view: on the one hand, it offers to the upper layers the *log* objects that gathers the agent's logging data, and it reports the logging data and movement / parent changes of the agent to the Visualization agent; on the other hand, it manages the window of the agent, and integrates it in the layout on the screen;
- WSAgent (or web service agent) – offers functionality that is used by the S-CLAIM agent to expose its capabilities as web services, through the WSIG JADE add-on, and also to access SOAP or RESTful web services, through the WSDC JADE add-on;
- HierarchicalAgent – manages hierarchical mobility of agents, exposing objects for hierarchical relations between agents, and implementing behaviors for both instructing an agent's children to follow the agent, and for an agent to receive movement orders from its parent; agent settings allow agents to be fixed to their containers, this meaning they will not follow their parent in the movement;
- S-CLAIM agent – the agent that executes using the provided S-CLAIM description; contains the symbol tables for the agent, and also the set of behavior descriptions that describe the agent's capabilities; this layer accesses the Knowledge Base component for adding, removing, or interrogating knowledge.

6.2.7 AmI Agent Modeling in the Ao Dai Platform

The Ao Dai platform has been primarily designed for the testing and simulation of multi-agent systems for Ambient Intelligence. More precisely, for the implementation of the agent topology and agent behavior presented in the other chapters of this thesis. Because of this, there are many elements of context-awareness that are already integrated in the foundation of the platform.

The most important of these is the hierarchical movement of agents. By means of the *HierarchicalAgent* layer of the Ao Dai agents, once a parent of the agent is defined, the agent will move together with its parent. In some cases, this may not be intended as the agent should be bound to a particular container, and this is the reason for the *fixed* parameter of the agent. Hierarchical movement means that an agent keeps (some of) its context when it moves, and is kept in the parent's context when the parent moves.

The way the S-CLAIM language is built leads to a tendency for agents to communicate predominantly only with their parent and children. This is in concordance with the principles set in Chapter 4, as this way information is exchanged between agents that share some aspects of context.

Not least, S-CLAIM programming relies in a great measure on patterns. While in the current implementation they are not graph patterns, but linear pat-

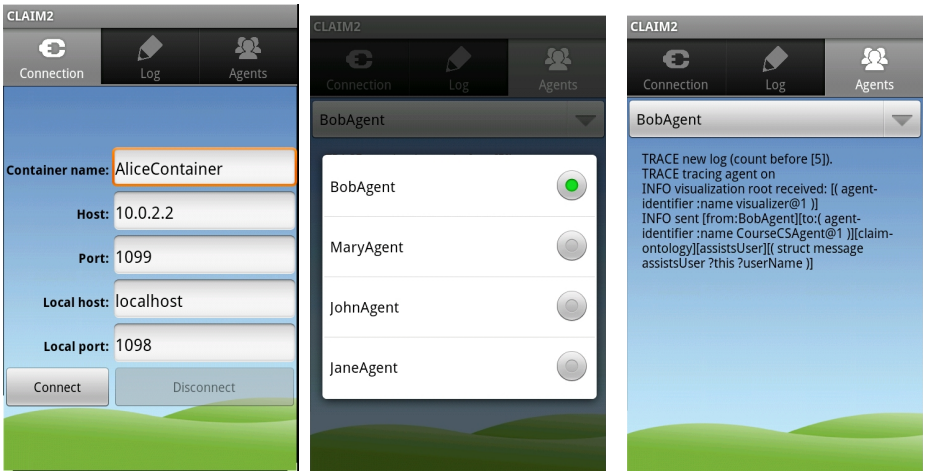


Figure 6.7: Interface to Ao Dai agents, on Android OS. Screenshots by Marius-Tudor Benea.

terns, the language constructs are oriented towards the use of structures with incompletely bound values. For instance, a behavior specifies the pattern for the message that activates it: if the pattern is completely specified, this only restricts the form of the message; but if the pattern is incomplete, it both restricts the message and it allows completing the pattern with new information, from the message. Pattern orientation is also visible in the primitives for knowledge management, especially in the case of the *readK* and *forAllK* constructs.

6.2.8 Deployment on Mobile Devices

Ambient Intelligence is meant to interface with the users of the system mainly by means of small devices with low capabilities. Such devices are, among others, the present day smartphones. These devices have too low capabilities to run fully fledged operating systems (and there is no need for that either). This is why we have also looked into how to deploy Ao Dai agents on mobile devices, more precisely on the Android operating system[6], which is continuously growing in market share. This has been done[7] with the help of the JADE-LEAP add-on. The fact that JADE uses Java also meant that most components of the Ao Dai platform work on Android out-of-the-box.

Modifications that needed to be done mainly related to agent loading and JADE-related operations, and to the interface. Given the reduced screen real-estate, the agents were viewed using an interface for the selection of one agent at a time. The friendly interface (visible in Figure 6.7) contains three tabs, in which the user can connect to the platform, can select agents and view their interface (for instance, their log).

[6] http://www.android.com/

[7] This work has been by Marius-Tudor Benea, a student of University Politehnica of Bucharest, on his Master internship at LIP6.

6.3 Experiments

As with any implementation, the Ao Dai platform needed to be experimented with in order to test it and to validate the concepts that underpinned it.

The process of development and testing of the platform went through the following phases:

- implementation and testing the original scenario of the Ao Dai prototype using normal JADE agents;
- experiments with configuring the execution through XML-based scenario files, including correct XML parsing and validation;
- usage of automatically generated events, using the scenario files;
- testing of the logging infrastructure and of primary visualization tools;
- implementation and testing of the representation of agent system structure in text, through linearized graphs;
- testing of the parsing of the first .adf2 files and transforming them to correct agent description structures;
- testing of the execution of S-CLAIM constructs (control structures, message sending and receiving, mobility);
- testing the correct creation and deployment of agents, based on the scenario file; this means creating the agent in the correct container and correct initialization of the agent with agent description, parameters and initial knowledge;
- testing of correct management of S-CLAIM agents' symbol table – variable management, assignment and scope.
- testing the distributed deployment of the platform, using two machines, and including creation of agents on a remote machine, movement of agents during the scenario, etc; testing was done over a fairly restricted network featuring firewalls and strict IP rules;
- testing the graphical representation of the topology of the multi-agent system;
- testing the deployment of CLAIM agents on a mobile device; mobile agents were deployed both on an Android emulator, as well as on an actual Android device, using the network of the laboratory to communicate;
- testing and improvement of the logging and visualization infrastructure in the context of using mobile devices, which are not compatible with all the Java components in a PC environment;
- testing the platform with new scenarios, by new developers (internship students); this validated not only the non-dependance of the platform on a specific scenario, but also the relative facility of working with the platform by developers that were not previously familiar with it;
- testing of the interoperation of the platform with both SOAP and RESTful web services;
- deployment and execution of the platform in a novel, distributed environment, and interoperation with the components of the SmartRoom (see Section 6.3.3).

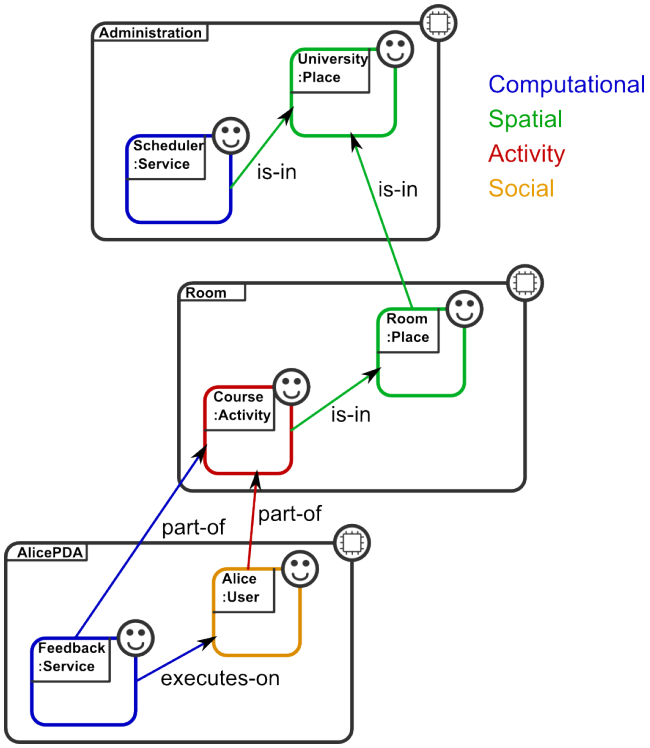


Figure 6.8: The agents in the scenario for the Ao Dai platform.

6.3.1 Scenario

It is in fact for the purpose of testing this platform that the reference scenario presented in Section 1.3.4 was created, in collaboration with the team from Honiden-Lab at the NII Institute in Tokyo. For the first tests of the Ao Dai platform, we have used just some segments of this scenario. The tested scenario is the following:

Scenario. Alice, Bob, and Carol are students in the Computer Science course. Today, the course is not held in the usual room, but in the SmartRoom. Once the scheduling system performs this change, the students are automatically notified. When the first person enters the room, the lights are turned on, and the main screen displays a welcome message. When all the students are in the room, the course is ready to start and the first slide of the Professor's presentation is displayed on the screen. When the Professor signals the end of the presentation, the room is reconfigured for the activities phase of the course. The students divide into two groups, each going in front of one of the large screens. The students write opinions regarding a subject in the interface provided by their agents, and the opinions are shown on the screen next to them. When the groups change and students move from one screen to the other, the opinions they wrote move with them and are displayed on the appropriate screen.

6.3.2 Modeling the Scenario

We have agentified the presented scenario using several agents: *Alice*, *Bob* and *Carol* are the agents that manage the activity of their respective users; *Course* is the agent that manages the Computer Science Course; *University* and *Room* are agents managing the smart places designated by their names; *Scheduler* and *Feedback* are services that deal with the scheduling of courses, and feedback for a specific course, respectively. The agentification of the scenario is presented in Figure 6.8. The figure presents the final state of the system. Initially, *Course* is not a child of *Room*, as the scheduling has not yet been announced.

Several contexts were represented in the model of the scenario. Agents communicated and moved inside these contexts, in a hierarchical manner. The existing contexts were:

- the spatial and organizational context of the University, in which the communication between the *Course* and *Room* Agents took place;
- the spatial and computational context of the Room, in which all agents entered, and in which agents could use the two available screens and could configure the room's lighting;
- the activity context of the CS Course, due to which the students were notified of the change in the location of the course;
- the spatial context of the two screens, which displayed the opinions of users situated in their vicinity.

6.3.3 Testing and Demonstration

A large number of experiments were performed with the Ao Dai platform, and many more will follow. The platform has been tested on one machine, and then in a distributed environment, and, thanks to JADE, agent mobility works fine in spite of restrictions, for instance like firewalls. Centralized visualization of the agents is extremely helpful in a distributed environment, and the architecture allows the log of crashed agents to be visible even if the agents have crashed on a remote machine.

The platform has also been tested using agents that moved to and from an Android device, which means that the platform is now even more appropriate for the development of AmI applications, as smartphones are fundamental in the architecture and scenarios of AmI.

But most importantly, the platform has been tested in the SmartRoom of the Honiden-Lab. The scenario described in Section 6.3.1 was used. The platform was deployed on multiple machines, and the agents communicated with the SmartRoom components by means of web services.

The SmartRoom offers two separate components: on the one hand, control of the lights, microphones, and different screens in the room; on the other hand, detection of the users in the room and of their locations, by means of RFID technologies.

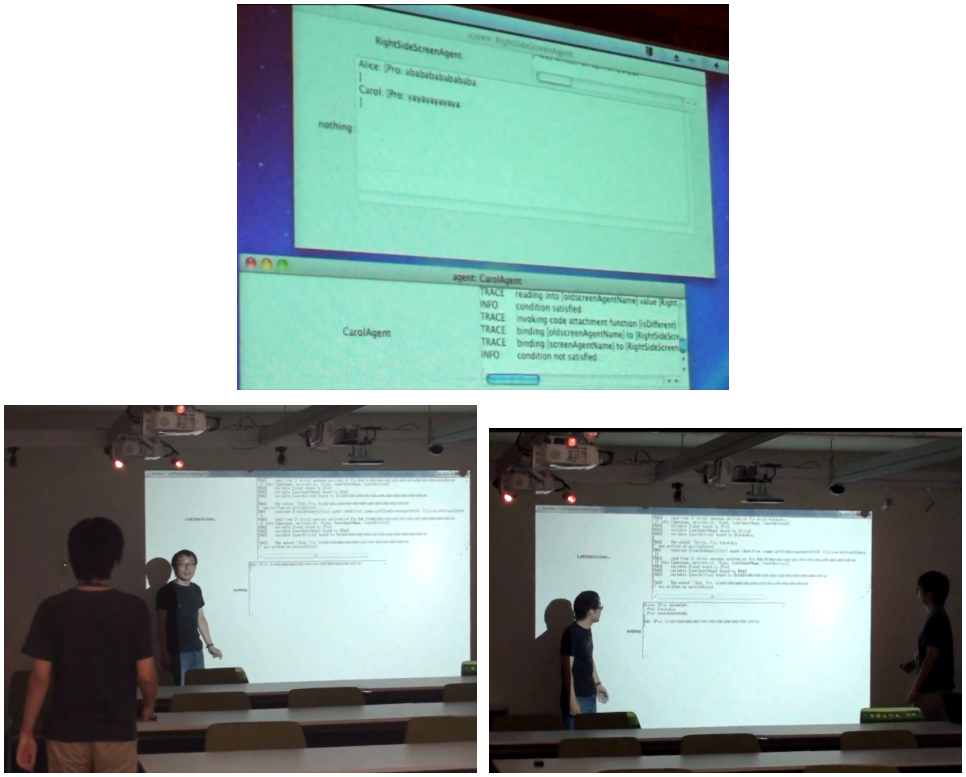


Figure 6.9: Images from the testing of the Ao Dai platform in the SmartRoom. The two last images show how opinions are moved to the back screen after the student changes location. As further proof of the platform's flexibility, one of the machines (above) runs Apple OS X and the other (below) runs Microsoft Windows 7.

While none of the developers of the Ao Dai platform was present at Honiden-Lab before, or during the simulation, the team at Honiden-Lab successfully deployed and experimented with the platform, at their first try, without any significant incidents. Moreover, members of the team successfully made slight modifications in the S-CLAIM code, without having any previous experience or contact with the language. The experiments were recorded on video, and some screenshots are presented in Figure 6.9. The video was subsequently demonstrated at the 6th NII-LIP6 Workshop held in Paris, in October 2011[8].

The success of the experiments and of the demonstration show that the platform is easy to use and to deploy. JADE offers interoperability and many useful components and add-ons. S-CLAIM offers a simple, easy to understand language for agent programming, that requires no prior experience. Web service integration showed that the platform can interoperate with other components and infrastructure, in a reliable manner, with almost no time spent on integration.

[8] Workshop held in collaboration by the National Institute of Informatics in Tokyo and the Laboratory of Computer Science of University Paris 6. Details at http://www-desir.lip6.fr/~herpsonc/6workshopNii/index.htm

6.4 Lessons Learned

Developing the Ao Dai platform has been an essential step in the development of a multi-agent system for Ambient Intelligence. The existence of this platform means that now Ambient Intelligence applications will be much easier to implement. This is because the integration of context-awareness, but also thanks to the fact that the S-CLAIM language is easy to use and understand.

Moreover, the *Scenario* and *Visualization* components make it easy to deploy agents in a distributed setting, simulate events, and follow the evolution of the system, all in a centralized manner (although decentralization is a priority in our research, centralized control and visualization of agent simulations is obviously an advantage).

Last but not least, like the Ao Dai prototype (see Chapter 4), the Ao Dai platform has been an occasion for the reinforcement of the collaboration between the universities to which the group that implemented it belong[9], and a new step in the collaboration with the Honiden Laboratory in Tokyo.

6.5 Perspectives

The Ao Dai platform is still in an early stage of development. Its modular and general structure allows the easy addition of new components.

Among these, more support for the pro-active, goal-oriented behavior of agents will be added. Other knowledge representation components may also be implemented. New language primitives for S-CLAIM may be introduced. Also, in order to become a more powerful language, S-CLAIM needs a library of already implemented Java functions, that will facilitate the implementation of the algorithmic aspects of S-CLAIM agents.

More experimentation with the platform is needed. Further extensions of the tested scenario are envisaged, as well as the inclusion of mobile devices (for which the platform is ready, from the implementation point of view). Further collaboration with the team from the Honiden Laboratory will allow more tests with the platform in the SmartRoom and potentially extensions beyond the space of only one room.

[9] University Pierre et Marie Curie in Paris, University Politehnica of Bucharest, and the IFI institute in Hanoi.

Chapter 7

Conclusions

This chapter concludes the PhD thesis. In closing, the work that was presented in the chapters will be summarized, in order to obtain a holistic view of this research (Section 7.1). The contributions of this work will be presented as a list, in Section 7.1.2.

This work yielded many results, many of which have good potential and deserve further improvement. The perspectives of this work as a whole, as well as of the individual concepts, developments and implementations, is presented in Section 7.2.

7.1 What Has Been Accomplished

The question that this research has answered is "How to build a multi-agent system for Ambient Intelligence?". The requirements for this research were, among others, system distribution, the use of cognitive agents, the use of mechanisms of self-organization, and last but definitely not least, the integration of context-awareness.

The central idea that characterizes the result of this work is **the integration of context-awareness in a MAS for AmI, in a way that allows agents to naturally manage and share context information**, while being able to perform, at the same time, application- and domain-specific tasks, and also respecting the requirements that were set.

7.1.1 Building Multi-Agent Systems for Ambient Intelligence

Building a context-aware multi-agent system started with a close inspection of Ambient Intelligence as a vision of the future of computing, in Chapter 1. Existing scenarios and applications have been presented, and the specific AmI features have been extracted and classified (Section 1.1). We have discussed the challenges of AmI, especially regarding scale, flow, anticipation, and security (Section 1.2).

One important section of the chapter is Section 1.3. This is a part of our contribution, and it is here that we present several **novel scenarios for AmI**, which are directed less towards the hardware used and the interfaces displayed, and more towards the functioning of the system as a whole, especially in collaborative or public environments.

The fundamental elements that we have chosen for our work were Multi-Agent Systems, Context-Awareness and the aspect of Self-Organization (Section 1.4). The last section of the chapter sets the goals for the research. The rest of the chapters show how these goals have been fulfilled.

In Chapter 2 of this work we have reviewed work in the related fields of study: agent-based AmI environments, context-awareness and self-organizing agent systems. The main conclusions that we have drawn were that agent systems in AmI need more flexibility but at the same time more powerful representations for context. Context-awareness is many times not a "first-class citizen" in the applications, and only some aspects of context are considered – notably location.

The model for a context-aware MAS for Ambient Intelligence that we propose, and that represents the main contribution of this work, is based on three aspects, presented in Chapters 3, 4 and 5, namely **agent behavior, system topology and context representation**. A very important thing to remember is that these three aspects are integrated with one another, and form a coherent, unitary model.

Agent behavior is based on local interaction and local knowledge. "Local" not only in terms of space, but of context – this can mean local in terms of social relations, or within the same activity, etc. The central idea for agent behavior is **"share interesting information with potentially interested neighbors"**. This behavior has been developed and validated in a first instance through the AmIciTy:Mi prototype, using between 900 and 1000 agents in a simulated environment that share information characterized by some **simple measures of context: pressure, specialty and persistence**. For the prototype we have also developed a simulation testbed directed towards execution speed, a specification for the generation of randomized scenarios, as well as visualization tools that allow the developer to follow the evolution of the system as a whole. The agent behavior and the AmIciTy:Mi prototype are presented in Chapter 3.

System topology is based on agent context. That is, neighborhood relations form hierarchies that are mapped to the hierarchical structure of different aspects of context. For instance, the agent managing an activity that is taking place in a certain room will have as "parent" the agent that manages the room, which itself is a "child" of the agent managing the building. But the agent managing the activity may also have as "parent" the agent managing the larger course that the activity is part of. The basic principle is that **two agents are neighbors if they share context**.

An initial prototype that validates this approach to system topology was Ao Dai, that considered spatial and computational context, as well as user pref-

erences. The project implemented a simple AmI-directed multi-agent-system scenario in which the agents represented elements of context – places, devices, services, and a user – which interacted in order to pro-actively help the user navigate and find computational resources in a building where the user comes for the first time. This project is presented in Chapter 4. The lessons learned in this project lead to more aspects of context (spatial, temporal, activity, computational and social) being considered, and the result was the more complete model presented in Section 5.3.

Context representation internal to the agents is based on two factors: the context graph – which stores the information that is currently relevant to the activity of the agent, as a graph of concepts and relations between concepts – and the context patterns – a set of graphs with generic elements that represent situations that the agent is able to recognize. The context patterns can be matched against the context graph in order to recognize the situation of the agent. Partial matches may lead to problem solving by the addition of new edges in the graph. Subgraphs that match context patterns are considered as interesting, and are shared with other agents. Upon receiving information, the interest of neighbor agents (in the context-based topology) can be computed by matching the patterns against the newly received information. The basic principle is that **information is interesting (or relevant) if the graph that represents it matches one of the patterns of the agent**. The formal model for a multi-agent system integrating context-awareness is presented in Chapter 5.

The concepts, agents and context models that were developed in Chapters 3, 4 and 5 needed to be tested by means of a prototype unifying all of them. This prototype is presented in Chapter 6. It has several layers: underpinnings for agent management, communication and mobility are assured by the JADE Java Agent Development Framework; we have developed a visualization layer that assures the centralization of agent logs, as well as the centralized graphical visualization of the system topology (Section 6.2.4); we have adapted and simplified the CLAIM agent-oriented programming language, having as result the S-CLAIM language and the implementation of ClaimAgents, that use the S-CLAIM specification to execute on top of JADE (Section 6.2.2); finally, we have implemented Context Agents, that use the CLAIM language coupled with a graph-based knowledge base and specific functionality in order to share context information and act in a context-aware fashion. Moreover, we have developed an XML-based format and associated agents for the automated simulation of the system, using scenarios, assuring easy and repeatable experiments (Section 6.2.3).

The results of the work presented in this thesis have been published in 20 research papers, of which 3 papers in ISI indexed journals (two of which awaiting publication), 2 papers in B+ journals, 10 in ISI indexed conference proceedings, 4 in the proceedings of international, peer-reviewed conferences, and 1 in the proceedings of an international summer school student session. Of these papers, 15 were published as first author. Two more articles (first author) are awaiting acceptance.

7.1.2 Contributions

The main original contributions of this work are the following:

- we have conceived several novel scenarios for Ambient Intelligence; the focus of these scenarios was towards the functioning of the system as a whole, toward adaptability and scalability of the system, towards problem solving, and towards collaborative and public environments, or settings involving multiple users (Section 1.3);
- we have realized a state of the art of multi-agent systems for developing Ambient Intelligence applications, observing the features that are offered and the tendency to use either few and complex agents or many simple agents, trading off powerful representations for flexibility (Section 2.1); we have also investigated the integration of context-awareness in Ambient Intelligence applications, noticing the balance between complexity and power of the context information representation, on the one hand, and the decentralization of the system, on the other hand (Section 2.2); a special mention is dedicated to algorithms for the matching of graphs, with a focus on the matching of labeled graphs (Section 2.2.4);
- we have realized a state of the art of self-organizing multi-agent system, noticing the relative lack of self-organizing systems formed of cognitive agents; we have also had a focus on the mechanisms through which the intended emergents are obtained (Section 2.3);
- we have proposed a model of distributed information spreading based on a set of context measures that allow the developer to control the spreading of information in a multi-agent system formed of a large number of agents. These context measures are pressure – controlling the speed of the spreading; specialty – controlling the direction and areas where the information spreads; and persistence – controlling the validity of the information; locality is also considered as an implicit measure (Section 3.2.1);
- we have developed an agent behavior that uses mechanisms of self-organization (e.g. positive and negative feedback loops, a certain level of randomness, etc), that results in a controllable spread of information at the global level (the level of the system), although individual agents only have local knowledge and communicate only with their immediate neighbors (Section 3.2.3);
- we have designed and implemented a simulation testbed for multi-agent systems formed of a large number of agents, directed toward quick, repeatable simulations running on a single machine, and validated it through several experiments in different settings (Section 3.3);
- we have developed an XML-based format for scenario files that allows the repeatable testing of large multi-agent systems, and that specifies the layout of agents and the time and nature of events; all parameters can be random with a defined interval and deviation (Section 3.3.3);
- we have designed and implemented visualization tools for large multi-agent systems, that allow the visualization of the evolution of the system as a whole, as well as of the evolution of individual agents; the tools include the visualization of instant value of one parameter for each agent of the system, the visualization of the time evolution of an aggregated

parameter over the whole system, selection and inspection of individual agents, as well as centralized logging (Section 3.3.4);

- we have devised a model for context-aware multi-agent systems in which the topology of the system (neighborhood relations) relates to the context of the agents and to the structure of the context (Sections 4.1 and 5.3);
- we have validated the context-based system topology through the design and development of the Ao Dai project, in which agents are assigned to context elements (like places, services and devices) and hierarchical relations between agents are mapped to the structure of the context; the project was implemented in CLAIM and demonstrated at the 5th NII-LIP6 workshop, held in June 2010 in Paris (Section 4.2);
- we have proposed a unified formalism for the representation of context information, both inside and outside the agent; the formalism is original and is based on a graph, in which different subgraphs represent the knowledge of agents, the relations between agents, and the assignment of agents to containers (or devices) (Section 5.1.1);
- we have conceived a formalism for graph patterns, that allows a pattern to match a wider range of individual graphs, using nodes with unspecified labels, as well as edges labeled with regular expressions (Section 5.2.2);
- we have developed an algorithm for the matching of a context pattern against a context graph, that allows partial matches and outputs the subgraph matching the pattern, the matched part of the pattern, and a numeric measurement for the fraction of the pattern that has been matched (Section 5.2.3);
- we have improved the agent behavior for the spreading of information using local interaction (as presented in Section 3.2.3) to integrate the context-aware system topology and the use of context patterns for the recognition of relevant situations (Section 5.4);
- we have simplified the semantics of the CLAIM agent-oriented programming language from the point of view of the number and categories of primitives, and we have simplified its syntax to use fewer constructs and special characters, having as result the S-CLAIM language which is simpler, cleaner, easier to read and easier to interpret (Section 6.2.2);
- we have implemented functionality for hierarchical mobility of JADE agents, in a similar fashion to the mobility of the original CLAIM agents (Section 6.2);
- we have implemented classes and functionality that allows a JADE agent to execute the agent description extracted from the S-CLAIM code, also using the functionality for hierarchical mobility (Section 6.2.2);
- we have proposed a formalism for the linear representation of graphs, and two algorithms for the linearization of a graph (for representation in text) and for the extraction of linear components of a graph (for graphical representation) (see the example in Section 5.2.2);
- we have implemented a visualization infrastructure that allows the centralization of agent logs, as well as the centralized visualization of the dynamic topology of the system; the visualization uses the algorithm for the linearization of graphs (Section 6.2.4);
- we have developed an XML-based format for the specification of sce-

narios for distributed multi-agent systems, assuring easy and repeatable simulation of JADE-based (and not only) multi-agent systems (Section 6.2.3);
- we have developed a MAS-based platform for AmI applications, focused on the application layer of Ambient Intelligence, using a graph based representation for context information, and underpinned by the S-CLAIM agent-oriented programming language (Section 6.3).

7.2 Future Work

This work is only a phase, a beginning. Many paths for the development of the concepts that we have introduced remain open and only too little explored. The implementations that were realized deserve to be improved and extended, and many concepts need further testing in applications and scenarios always closer to real life. Some of these potential targets for the future are presented in this last Section of the thesis.

There are many perspectives for the AmIciTy:Mi platform. It makes a great platform to study complex systems formed of large number of agents with similar behavior. Its advantages are speed – allowing for the realization of many simulations – and the existing visualization tools.

AmIciTy:Mi can be extended to support agents of different sizes (being able to store more or less information). Also, agents that move through space would make a great addition. The scenario files would be improved to be able to specify agent sizes and paths. The study of moving agents would allow more insights in how information spreads, in a more realistic system,.

It could also be used to develop new measures for context, that would be as simple, but also as effective as the ones already developed. Conversely, the development of new measures for evaluating the evolution of the system, and new tools for the visualization of the system.

The Ao Dai project has already lead to several followups, both in theoretical development as well as in the development of the platform, namely the extension of the considered aspects of context, on the one hand, and the development of the S-CLAIM language and the new MAS platform, on the other hand.

But maybe more importantly, the Ao Dai project founded a collaboration between students from several universities in Europe, Brazil and Asia – MAS team from University Pierre et Marie Curie (Paris 6), AI-MAS from University "Politehnica" of Bucharest, IFI institute from Hanoi and PUC-Rio University from Brazil. This collaboration is likely to continue.

As for context-awareness (both inside and outside the agent), we believe that our approach has a lot of potential, that has not yet been entirely explored. Both context-awareness outside and inside the agent are new contributions, therefore it is only intensive testing that will fully validate their adequacy to

various applications.

Temporality has only been little explored. User's (or agent's) history, as well as planned activities, could be formalized as special elements and treated as such.

Another aspect that needs testing in practice and non-simulated applications is the use of the *characteristic* and *actionable* features of patterns, which have the power to greatly increase the possibilities of applying patterns.

Very much needed future work involves the implementation of more, and more varied scenarios, using the context-aware system topology and context patterns. It is only by experiment that the real power of these representations will be understood.

The MAS platform for AmI is to be further developed and extended; its different enhancements will be easy as it relies on well-implemented components that have a high degree of generality. It can prove to be not only an AmI experiment, but also a very useful and simple to use platform for the deployment and testing of multi-agent systems. The S-CLAIM agent-oriented language is easier to use than CLAIM, and has a good chance of becoming a more widely used agent language.

List of publications

Florea, A. M., Kalisz, E., and Olaru, A. (2009). Levels of emergent behaviour in agent societies. In Proceedings of CASYS'09, the 9th International Conference on Computing Anticipatory Systems, August 3-8, Liege, Belgium, pages 81-88. American Institute of Physics (AIP) Conference Proceedings. ISSN 0094-243X, ISBN 978-0-7354-0579-0 (ISI Proceedings).

Olaru, A., Marinica, C., and Guillet, F. (2009). Local mining of association rules with rule schemas. In Proceedings of CIDM 2009, the IEEE Symposium on Computational Intelligence and Data Mining, March 30 - April 2, Nashville, TN, USA, IEEE Symposium Series on Computational Intelligence, pages 118-124. (ISI Proceedings).

Marinica, C., Olaru, A., and Guillet, F. (2009). User-driven association rule mining using a local algorithm. In Cordeiro, J. and Filipe, J., editors, Proceedings of ICEIS 2009, the 11th International Conference on Enterprise Information Systems, May 6-10, Milan, Italy, pages 200-205. ISBN 978-989-8111-85-2 (ISI Proceedings).

Olaru, A. and Florea, A. M. (2009). Emergence in cognitive multi-agent systems. Proceedings of CSCS17, the 17th International Conference on Control Systems and Computer Science, MASTS Workshop, May 26-29, Bucuresti, Romania, 2:515-522. ISSN 2066-4451.

Olaru, A., Gratie, C., and Florea, A. M. (2009). Context-aware emergent behaviour in a MAS for information exchange. In Proceedings of ACSys09, 6th Workshop on Agents for Complex Systems, in conjunction with SYNASC 2009, September 26-29, Timisoara, Romania, pages 17-22.

Olaru, A., Gratie, C., and Florea, A. M. (2009). Emergent properties for data distribution in a cognitive MAS. In Papadopoulos, G. A. and Badica, C., editors, Proceedings of IDC 2009, 3rd International Symposium on Intelligent Distributed Computing, October 13-14, Ayia Napa, Cyprus, volume 237 of Studies in Computational Intelligence, pages 151-159. Springer. ISBN 978-3-642-03213-4 (ISI Proceedings).

Olaru, A., Gratie, C., and Florea, A. M. (2009). Measures of context-awareness for self-organizing systems. In Proceedings of EUMAS 2009, 7th European Workshop on Multi-Agent Systems, Dec 17-18, Ayia Napa, Cyprus.

El Fallah Seghrouchni, A., Florea, A. M., and Olaru, A. (2010). Multi-agent

systems: a paradigm to design ambient intelligent applications. In Essaaidi, M., Malgeri, M., and Badica, C., editors, Proceedings of IDC'2010, the 4th International Symposium on Intelligent Distributed Computing, volume 315 of Studies in Computational Intelligence, pages 3-9. Springer. (ISI Proceedings).

El Fallah Seghrouchni, A., Olaru, A., Nguyen, T. T. N., and Salomone, D. (2010). Ao Dai: Agent oriented design for ambient intelligence. In Proceedings of PRIMA 2010, the 13th International Conference on Principles and Practice of Multi-Agent Systems.

Olaru, A., El Fallah Seghrouchni, A., and Florea, A. M. (2010). Ambient intelligence: From scenario analysis towards a bottom-up design. In Essaaidi, M., Malgeri, M., and Badica, C., editors, Proceedings of IDC'2010, the 4th International Symposium on Intelligent Distributed Computing, volume 315 of Studies in Computational Intelligence, pages 165-170. Springer. (ISI Proceedings).

Olaru, A. and Florea, A. M. (2010). A graph based approach to context matching. Scalable Computing: Practice and Experience, 11(4):393-399. ISSN 1895-1767 (B+ Journal).

Olaru, A. and Florea, A. M. (2010). A graph based approach to context matching. In Proceedings of SYNASC 2010, 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania.

Olaru, A. and Gratie, C. (2010). Agent-based information sharing for ambient intelligence. In Essaaidi, M., Malgeri, M., and Badica, C., editors, Proceedings of IDC'2010, the 4th International Symposium on Intelligent Distributed Computing, MASTS 2010, the The 2nd International Workshop on Multi-Agent Systems Technology and Semantics, volume 315 of Studies in Computational Intelligence, pages 285-294. Springer. (ISI Proceedings).

Olaru, A., Gratie, C., and Florea, A. M. (2010). Context-aware emergent behaviour in a MAS for information exchange. Scalable Computing: Practice and Experience, 11(1):33-42. ISSN 1895-1767(B+ Journal).

Olaru, A., Gratie, C., and Florea, A. M. (2010). Designing Self-Organizing Cognitive Multi-Agent Systems. In Filip, F. G. and Enachescu, C., editors, Advanced Computational Technologies, Romanian Academy Press (in print)

Olaru, A., Gratie, C., and Florea, A. M. (2010). Emergent properties for data distribution in a cognitive MAS. Computer Science and Information Systems, 7(3):643-660. ISSN 1820-0214 (ISI Indexed Journal).

Olaru, A., El Fallah Seghrouchni, A., and Florea, A. M. (2011). Graphs and patterns for context-awareness. In Novais, P., Preuveneers, D., and Corchado, J. M., editors, Proceedings of International Symposium on Ambient Intelligence, University of Salamanca (Spain), 6-8th April, volume 92 of Advances in Intelligent and Soft Computing, pages 165-172. Springer. ISBN 978-3-642-19936-3, ISSN 1867-5662 (ISI Proceedings).

Olaru, A. and Florea, A. M. (2011). Context-aware agents for developing AmI applications. Journal of Control Engineering and Applied Informatics, 13(4). (in print) (ISI Indexed Journal).

Olaru, A. and Gratie, C. (2011). Agent-based, context-aware information sharing for ambient intelligence. International Journal on Artificial Intelligence Tools, 20(6):985-1000. (in print) (ISI Indexed Journal).

El Fallah Seghrouchni, A., Olaru, A., Nguyen, T. T. N., and Salomone, D. (2011). Ao Dai: Agent oriented design for ambient intelligence. In Proceedings of PRIMA 2010, the 13th International Conference on Principles and Practice of Multi-Agent Systems, number 7057 in Lecture Notes in Artificial Intelligence, pages 259-265. Springer (in print) (ISI Proceedings).

Olaru, A, Florea, A. M., and El Fallah Seghrouchni, A. (2011). An agent-oriented approach for ambient intelligence. UPB Scientific Bulletin, Series C Electrical Engineering and Computer Science. (awaiting review)

Bibliography

[Aarts and Grotenhuis, 2011] Aarts, E. and Grotenhuis, F. (2011). Ambient intelligence 2.0: Towards synergetic prosperity. *Journal of Ambient Intelligence and Smart Environments*, 3(1):3–11.

[Alonso et al., 2009] Alonso, R., García, Ó., Saavedra, A., Tapia, D., de Paz, J., and Corchado, J. (2009). Heterogeneous wireless sensor networks in a tele-monitoring system for homecare. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pages 663–670.

[Amaral and Ottino, 2004] Amaral, L. and Ottino, J. (2004). Complex networks: Augmenting the framework for the study of complex systems. *The European Physical Journal B-Condensed Matter*, 38(2):147–162.

[Augusto et al., 2010] Augusto, J., Nakashima, H., and Aghajan, H. (2010). Ambient intelligence and smart environments: A state of the art. *Handbook of Ambient Intelligence and Smart Environments*, pages 3–31.

[Augusto and McCullagh, 2007] Augusto, J. C. and McCullagh, P. J. (2007). Ambient intelligence: Concepts and applications. *Computer Science and Information Systems (ComSIS)*, 4(1):1–27.

[Baldauf et al., 2007] Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

[Banavar and Bernstein, 2002] Banavar, G. and Bernstein, A. (2002). Software infrastructure and design challenges for ubiquitous computing applications. *Communications of the ACM*, 45(12):92–96.

[Bauchet et al., 2009] Bauchet, J., Pigot, H., Giroux, S., Lussier-Desrochers, D., Lachapelle, Y., and Mokhtari, M. (2009). Designing judicious interactions for cognitive assistance: the acts of assistance approach. *Proceeding of the eleventh international ACM SIGACCESS conference on Computers and accessibility*, pages 11–18.

[Bellifemine et al., 1999] Bellifemine, F., Poggi, A., and Rimassa, G. (1999). JADE - a FIPA-compliant agent framework. In *Proceedings of PAAM*, volume 99, pages 97–108. Citeseer.

[Bellifemine et al., 2001] Bellifemine, F., Poggi, A., and Rimassa, G. (2001). Developing multi-agent systems with JADE. *Intelligent Agents VII Agent Theories Architectures and Languages*, pages 42–47.

[Bengoetxea et al., 2002] Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2002). Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition*, 35(12):2867–2880.

[Bernadas et al., 2008] Bernadas, A., Alfano, R., Manzalini, A., Solé-Pareta, J., and Spadaro, S. (2008). Demonstrating communication services based on autonomic self-organization. *Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems-Volume 00*, pages 101–107.

[Bettini et al., 2010] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180.

[Beurier et al., 2002] Beurier, G., Simonin, O., and Ferber, J. (2002). Model and simulation of multi-level emergence. *Proceedings of IEEE ISSPIT*, pages 231–236.

[Bikakis and Antoniou, 2010] Bikakis, A. and Antoniou, G. (2010). Defeasible contextual reasoning with arguments in ambient intelligence. *IEEE Transactions on Knowledge and Data Engineering*, pages 1492–1506.

[Böhlen, 2009] Böhlen, M. (2009). Second order ambient intelligence. *Journal of Ambient Intelligence and Smart Environments*, 1(1):63–67.

[Bohn et al., 2005] Bohn, J., Coroama, V., Langheinrich, M., Mattern, F., and Rohs, M. (2005). Social, economic, and ethical implications of ambient intelligence and ubiquitous computing. *Ambient intelligence*, pages 5–29.

[Bolchini et al., 2007] Bolchini, C., Curino, C., Quintarelli, E., Schreiber, F., and Tanca, L. (2007). A data-oriented survey of context models. *ACM SIGMOD Record*, 36(4):19–26.

[Bourjot et al., 2003] Bourjot, C., Chevrier, V., and Thomas, V. (2003). A new swarm mechanism based on social spiders colonies: From web weaving to region detection. *Web Intelligence and Agent Systems*, 1(1):47–64.

[Brézillon and Brézillon, 2007] Brézillon, J. and Brézillon, P. (2007). Context modeling: Context as a dressing of a focus. *Modeling and Using Context*, pages 136–149.

[Brezillon and Pomerol, 1999] Brezillon, R. and Pomerol, J. (1999). Contextual knowledge sharing and cooperation in intelligent assistant systems. *Le Travail Humain*, 62(3):223–246.

[Cabri et al., 2005] Cabri, G., Ferrari, L., Leonardi, L., and Zambonelli, F. (2005). The LAICA project: Supporting ambient intelligence via agents and ad-hoc middleware. *Proceedings of WETICE 2005, 14th IEEE International Workshops on Enabling Technologies, 13-15 June 2005, Linköping, Sweden*, pages 39–46.

[Caetano et al., 2009] Caetano, T., McAuley, J., Cheng, L., Le, Q., and Smola, A. (2009). Learning graph matching. *IEEE transactions on pattern analysis and machine intelligence*, pages 1048–1058.

[Capera et al., 2003] Capera, D., Georgé, J., Gleizes, M., and Glize, P. (2003). Emergence of organisations, emergence of functions. In *AISB03 symposium on Adaptive Agents and Multi-Agent Systems*.

[Cardelli and Gordon, 2000] Cardelli, L. and Gordon, A. D. (2000). Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213.

[Chen and Kotz, 2000] Chen, G. and Kotz, D. (2000). A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College.

[Chen et al., 2004] Chen, H., Finin, T. W., Joshi, A., Kagal, L., Perich, F., and Chakraborty, D. (2004). Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 8(6):69–79.

[Conte et al., 2004] Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(3):265–298.

[Cook et al., 2009] Cook, D., Augusto, J., and Jakkula, V. (2009). Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277–298.

[Corchado et al., 2008] Corchado, J., Bajo, J., de Paz, Y., and Tapia, D. (2008). Intelligent environment for monitoring alzheimer patients, agent technology for health care. *Decision Support Systems*, 44(2):382–396.

[Cordella et al., 1998] Cordella, L., Foggia, P., Sansone, C., Tortorella, F., and Vento, M. (1998). Graph matching: a fast algorithm and its evaluation. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 2, pages 1582–1584. IEEE.

[Cordella et al., 2004] Cordella, L., Foggia, P., Sansone, C., and Vento, M. (2004). A (sub) graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10):1367–1372.

[Costantini et al., 2008] Costantini, S., Mostarda, L., Tocchio, A., and Tsintza, P. (2008). DALICA: Agent-based ambient intelligence for cultural-heritage scenarios. *IEEE Intelligent Systems*, 23(2):34–41.

[Crandall and Cook, 2009] Crandall, A. and Cook, D. (2009). Coping with multiple residents in a smart environment. *Journal of Ambient Intelligence and Smart Environments*, 1(4):323–334.

[De Wolf and Holvoet, 2005] De Wolf, T. and Holvoet, T. (2005). Emergence versus self-organisation: Different concepts but promising when combined. *Engineering Self Organising Systems: Methodologies and Applications*, 3464:1–15.

[Dey, 2001] Dey, A. (2001). Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7.

[Dey and Abowd, 2000] Dey, A. and Abowd, G. (2000). Towards a better understanding of context and context-awareness. *CHI 2000 workshop on the what, who, where, when, and how of context-awareness*, pages 304–307.

[Dey et al., 1999] Dey, A., Abowd, G., and Salber, D. (1999). A context-based infrastructure for smart environments. *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, pages 114–128.

[Dolev and Tzachar, 2009] Dolev, S. and Tzachar, N. (2009). Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. *Theoretical Computer Science*, 410(6-7):514–532.

[Ducatel et al., 2001] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., and Burgelman, J. (2001). Scenarios for ambient intelligence in 2010. Technical report, Office for Official Publications of the European Communities.

[El Fallah Seghrouchni, 2008] El Fallah Seghrouchni, A. (2008). Intelligence ambiante, les defis scientifiques. presentation, Colloque Intelligence Ambiante, Forum Atena.

[El Fallah Seghrouchni et al., 2008] El Fallah Seghrouchni, A., Breitman, K., Sabouret, N., Endler, M., Charif, Y., and Briot, J. (2008). Ambient intelligence applications: Introducing the campus framework. *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'2008)*, pages 165–174.

[El Fallah Seghrouchni et al., 2010a] El Fallah Seghrouchni, A., Florea, A. M., and **Olaru**, A. (2010a). Multi-agent systems: a paradigm to design ambient intelligent applications. In Essaaidi, M., Malgeri, M., and Badica, C., editors, *Proceedings of IDC'2010, the 4th International Symposium on Intelligent Distributed Computing*, volume 315 of *Studies in Computational Intelligence*, pages 3–9. Springer. (ISI Proceedings).

[El Fallah Seghrouchni et al., 2010b] El Fallah Seghrouchni, A., **Olaru**, A., Nguyen, T. T. N., and Salomone, D. (2010b). Ao Dai: Agent oriented design for ambient intelligence. In *Proceedings of PRIMA 2010, the 13th International Conference on Principles and Practice of Multi-Agent Systems*.

[El Fallah Seghrouchni et al., 2011] El Fallah Seghrouchni, A., **Olaru**, A., Nguyen, T. T. N., and Salomone, D. (2011). Ao Dai: Agent oriented design for ambient intelligence. In *Proceedings of PRIMA 2010, the 13th International Conference on Principles and Practice of Multi-Agent Systems*, number 7057 in Lecture Notes in Artificial Intelligence, pages 259–265. Springer.

[Feng et al., 2004] Feng, L., Apers, P. M. G., and Jonker, W. (2004). Towards context-aware data management for ambient intelligence. In Galindo, F., Takizawa, M., and Traunmüller, R., editors, *Proceedings of DEXA 2004, 15th International Conference on Database and Expert Systems Applications, Zaragoza, Spain, August 30 - September 3*, volume 3180 of *Lecture Notes in Computer Science*, pages 422–431. Springer.

[Ferber, 1999] Ferber, J. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley.

[Florea et al., 2009] Florea, A. M., Kalisz, E., and **Olaru**, A. (2009). Levels of emergent behaviour in agent societies. In *Proceedings of CASYS'09, the 9th International Conference on Computing Anticipatory Systems, August 3-8, Liège, Belgium*, pages 81–88. American Institute of Physics (AIP) Conference Proceedings. ISSN 0094-243X, ISBN 978-0-7354-0579-0 (ISI Proceedings).

[Gašević and Hatala, 2005] Gašević, D. and Hatala, M. (2005). Searching web resources using ontology mappings. In *Integrating Ontologies Workshop Proceedings*, page 33. Citeseer.

[Gleizes et al., 1999] Gleizes, M., Camps, V., and Glize, P. (1999). A theory of emergent computation based on cooperative self-organization for adaptive artificial systems. In *Fourth European Congress of Systems Science*.

[Gold and Rangarajan, 1996] Gold, S. and Rangarajan, A. (1996). A graduated assignment algorithm for graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(4):377–388.

[Goldstein, 1999] Goldstein, J. (1999). Emergence as a construct: History and issues. *Emergence*, 1(1):49–72.

[Grasse, 1959] Grasse, P. (1959). La reconstruction du nid et les interactions inter-individuelles chez les bellicositermes natalenis et cubitermes sp. la thorie de la stigmergie: essai d'interprtation des termites constructeurs. *Insectes Sociaux*, 6:41–83.

[Hagras et al., 2004] Hagras, H., Callaghan, V., Colley, M., Clarke, G., Pounds-Cornish, A., and Duman, H. (2004). Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems*, pages 12–20.

[Hales and Edmonds, 2003] Hales, D. and Edmonds, B. (2003). Evolving social rationality for MAS using "tags". *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 497–503.

[Harter et al., 2002] Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P. (2002). The anatomy of a context-aware application. *Wireless Networks*, 8(2):187–197.

[Hellenschmidt, 2005] Hellenschmidt, M. (2005). Distributed implementation of a self-organizing appliance middleware. In Davies, N., Kirste, T., and Schumann, H., editors, *Mobile Computing and Ambient Intelligence*, volume 05181 of *Dagstuhl Seminar Proceedings*, pages 201–206. ACM, IBFI, Schloss Dagstuhl, Germany.

[Hellenschmidt and Kirste, 2004] Hellenschmidt, M. and Kirste, T. (2004). A generic topology for ambient intelligence. *Lecture notes in computer science*, pages 112–123.

[Henricksen and Indulska, 2006] Henricksen, K. and Indulska, J. (2006). Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37–64.

[Henricksen et al., 2002] Henricksen, K., Indulska, J., and Rakotonirainy, A. (2002). Modeling context information in pervasive computing systems. *Lecture notes in computer science*, pages 167–180.

[Hervás et al., 2011] Hervás, R., Bravo, J., and Fontecha, J. (2011). Awareness marks: adaptive services through user interactions with augmented objects. *Personal and Ubiquitous Computing*, 15(4):409–418.

[Heylighen, 1989] Heylighen, F. (1989). Self-organization, emergence and the architecture of complexity. *Proceedings of the 1st European Conference on System Science*, pages 23–32.

[Heylighen, 2002] Heylighen, F. (2002). The science of self-organization and adaptivity. *The Encyclopedia of Life Support Systems*, pages 1–26.

[Hu et al., 2005] Hu, W., Jian, N., Qu, Y., and Wang, Y. (2005). GMO: A graph matching for ontologies. In *Integrating Ontologies Workshop Proceedings*, page 41. Citeseer.

[Hussain and Abidi, 2009] Hussain, S. and Abidi, S. (2009). K-MORPH: A semantic web based knowledge representation and context-driven morphing framework. In *Advances in Artificial Intelligence: 22nd Canadian Conference on Artificial Intelligence, Canadian AI 2009 Kelowna, Canada, May 25-27, 2009 Proceedings*, page 279. Springer-Verlag New York Inc.

[Johanson et al., 2002] Johanson, B., Fox, A., and Winograd, T. (2002). The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE pervasive computing*, pages 67–74.

[Kindberg et al., 2002] Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., and Serra, B. (2002). People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376.

[Kirsch-Pinheiro et al., 2008] Kirsch-Pinheiro, M., Vanrompay, Y., and Berbers, Y. (2008). Context-aware service selection using graph matching. In *2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC08), at ECOWS*. Citeseer.

[Laera et al., 2007] Laera, L., Blacoe, I., Tamma, V., Payne, T., Euzenat, J., and Bench-Capon, T. (2007). Argumentation over ontology correspondences in MAS. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8. ACM.

[Lech and Wienhofen, 2005] Lech, T. C. and Wienhofen, L. W. M. (2005). AmbieAgents: a scalable infrastructure for mobile and context-aware information services. *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*, pages 625–631.

[Leung et al., 1995] Leung, T., Burl, M., and Perona, P. (1995). Finding faces in cluttered scenes using random labeled graph matching. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 637–644. IEEE.

[Luo and Hancock, 2001] Luo, B. and Hancock, E. (2001). Structural graph matching using the EM algorithm and singular value decomposition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pages 1120–1136.

[Lyons et al., 2010] Lyons, P., Cong, A., Steinhauer, H., Marsland, S., Dietrich, J., and Guesgen, H. (2010). Exploring the responsibilities of single-inhabitant smart homes with use cases. *Journal of Ambient Intelligence and Smart Environments*, 2(3):211–232.

[Malandrino et al., 2010] Malandrino, D., Mazzoni, F., Riboni, D., Bettini, C., Colajanni, M., and Scarano, V. (2010). MIMOSA: context-aware adaptation for ubiquitous web access. *Personal and Ubiquitous Computing*, 14(4):301–320.

[Mamei et al., 2004] Mamei, M., Vasirani, M., and Zambonelli, F. (2004). Self-organizing spatial shapes in mobile particles: The TOTA approach. In Brueckner, S., Serugendo, G. D. M., Karageorgos, A., and Nagpal, R., editors, *Engineering Self-Organising Systems, Methodologies and Applications (after ESOA 2004 workshop)*, volume 3464 of *Lecture Notes in Computer Science*, pages 138–153. Springer.

[Mano et al., 2006] Mano, J.-P., Bourjot, C., Lopardo, G. A., and Glize, P. (2006). Bio-inspired mechanisms for artificial self-organised systems. *Informatica (Slovenia): Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini*, 30(1):55–62.

[Marinescu et al., 2008] Marinescu, D., Morrison, J., Yu, C., Norvik, C., and Siegel, H. (2008). A self-organization model for complex computing and communication systems. *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 149–158.

[Marinica et al., 2009] Marinica, C., **Olaru**, A., and Guillet, F. (2009). User-driven association rule mining using a local algorithm. In Cordeiro, J. and Filipe, J., editors, *Proceedings of ICEIS 2009, the 11th International Conference on Enterprise Information Systems, May 6-10, Milan, Italy*, pages 200–205. ISBN 978-989-8111-85-2 (ISI Proceedings).

[Messmer and Bunke, 1999] Messmer, B. and Bunke, H. (1999). A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32(12):1979–1998.

[Messmer and Bunke, 2000] Messmer, B. and Bunke, H. (2000). Efficient subgraph isomorphism detection: A decomposition approach. *Knowledge and Data Engineering, IEEE Transactions on*, 12(2):307–323.

[Mills, 2007] Mills, K. L. (2007). A brief survey of self-organization in wireless sensor networks. *Wireless Communications and Mobile Computing*, 7(1):823–834.

[Mozer, 2005] Mozer, M. (2005). Lessons from an adaptive home. *Smart Environments*, pages 271–294.

[Muldoon et al., 2006] Muldoon, C., O'Hare, G. M. P., Collier, R. W., and O'Grady, M. J. (2006). Agent factory micro edition: A framework for ambient applications. In Alexandrov, V. N., van Albada, G. D., Sloot, P. M. A., and Dongarra, J., editors, *Proceedings of ICCS 2006, 6th International Conference on Computational Science, Reading, UK, May 28-31*, volume 3993 of *Lecture Notes in Computer Science*, pages 727–734. Springer.

[Novak and Cañas, 2006] Novak, J. D. and Cañas, A. J. (2006). The origins of the concept mapping tool and the continuing evolution of the tool. *Information Visualization*, 5(3):175–184.

[Nute, 2001] Nute, D. (2001). Defeasible logic. In *Proceedings of INAP 2001, 14th International Conference on Applications of Prolog*, pages 87–114.

[Olaru, 2010] Olaru, A. (2010). A context-aware multi-agent system for AmI environments. Technical report, University Politehnica of Bucharest, University Pierre et Marie Curie Paris. February 2010.

[**Olaru** et al., 2010a] **Olaru**, A., El Fallah Seghrouchni, A., and Florea, A. M. (2010a). Ambient intelligence: From scenario analysis towards a bottom-up design. In Essaaidi, M., Malgeri, M., and Badica, C., editors, *Proceedings of IDC'2010, the 4th International Symposium on Intelligent Distributed Computing*, volume 315 of *Studies in Computational Intelligence*, pages 165–170. Springer. (ISI Proceedings).

[**Olaru** and Florea, 2009] **Olaru**, A. and Florea, A. M. (2009). Emergence in cognitive multi-agent systems. In *Proceedings of CSCS17, the 17th International Conference on Control Systems and Computer Science, MASTS Workshop, May 26-29, Bucuresti, Romania*, volume 2, pages 515–522. ISSN 2066-4451.

[**Olaru** and Florea, 2010a] **Olaru**, A. and Florea, A. M. (2010a). A graph-based approach to context matching. *Scalable Computing: Practice and Experience*, 11(4):393–399. ISSN 1895-1767 (B+ Journal).

[**Olaru** and Florea, 2010b] **Olaru**, A. and Florea, A. M. (2010b). A graph-based approach to context matching. In *Proceedings of SYNASC 2010, 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania.*

[**Olaru** and Florea, 2011] **Olaru**, A. and Florea, A. M. (2011). Context-aware agents for developing AmI applications. *Journal of Control Engineering and Applied Informatics*, 13(4). (in print) (ISI Indexed Journal).

[**Olaru** et al., 2011] **Olaru**, A., Florea, A. M., and El Fallah Seghrouchni, A. (2011). Graphs and patterns for context-awareness. In Novais, P., Preuveneers, D., and Corchado, J. M., editors, *Proceedings of International Symposium on Ambient Intelligence, University of Salamanca (Spain), 6-8th April*, volume 92 of *Advances in Intelligent and Soft Computing*, pages 165–172. Springer. ISBN 978-3-642-19936-3, ISSN 1867-5662 (ISI Proceedings).

[**Olaru** and Gratie, 2010] **Olaru**, A. and Gratie, C. (2010). Agent-based information sharing for ambient intelligence. In Essaaidi, M., Malgeri, M., and Badica, C., editors, *Proceedings of IDC'2010, the 4th International Symposium on Intelligent Distributed Computing, MASTS 2010, the The 2nd International Workshop on Multi-Agent Systems Technology and Semantics*, volume 315 of *Studies in Computational Intelligence*, pages 285–294. Springer. (ISI Proceedings).

[**Olaru** and Gratie, 2011] **Olaru**, A. and Gratie, C. (2011). Agent-based, context-aware information sharing for ambient intelligence. *International Journal on Artificial Intelligence Tools*, 20(6):985–1000. (ISI Indexed Journal).

[**Olaru** et al., 2009a] **Olaru**, A., Gratie, C., and Florea, A. M. (2009a). Context-aware emergent behaviour in a MAS for information exchange. In *Proceedings of ACSys09, 6th Workshop on Agents for Complex Systems, in conjunction with SYNASC 2009, September 26-29, Timisoara, Romania*, pages 17–22.

[**Olaru** et al., 2009b] **Olaru**, A., Gratie, C., and Florea, A. M. (2009b). Emergent properties for data distribution in a cognitive MAS. In Papadopoulos, G. A. and Badica, C., editors, *Proceedings of IDC 2009, 3rd International Symposium on Intelligent Distributed Computing, October 13-14, Ayia Napa, Cyprus*, volume 237 of *Studies in Computational Intelligence*, pages 151–159. Springer. ISBN 978-3-642-03213-4 (ISI Proceedings).

[**Olaru** et al., 2009c] **Olaru**, A., Gratie, C., and Florea, A. M. (2009c). Measures of context-awareness for self-organizing systems. In *Proceedings of EUMAS 2009, 7th European Workshop on Multi-Agent Systems, Dec 17-18, Ayia Napa, Cyprus.*

[**Olaru** et al., 2010b] **Olaru**, A., Gratie, C., and Florea, A. M. (2010b). Context-aware emergent behaviour in a MAS for information exchange. *Scalable Computing: Practice and Experience*, 11(1):33–42. ISSN 1895-1767 (B+ Journal).

[**Olaru** et al., 2010c] **Olaru**, A., Gratie, C., and Florea, A. M. (2010c). Emergent properties for data distribution in a cognitive MAS. *Computer Science and Information Systems*, 7(3):643–660. ISSN 1820-0214 (ISI Indexed Journal).

[**Olaru** et al., 2009d] **Olaru**, A., Marinica, C., and Guillet, F. (2009d). Local mining of association rules with rule schemas. In *Proceedings of CIDM 2009, the IEEE Symposium on Computational Intelligence and Data Mining, March 30 - April 2, Nashville, TN, USA*, IEEE Symposium Series on Computational Intelligence, pages 118–124. (ISI Proceedings).

[Paymans et al., 2004] Paymans, T., Lindenberg, J., and Neerincx, M. (2004). Usability trade-offs for adaptive user interfaces: ease of use and learnability. In *Proceedings of the 9th international conference on Intelligent user interfaces*, pages 301–303. ACM.

[Perakis et al., 2009] Perakis, K., Haritou, M., and Koutsouris, D. (2009). ALADDIN, a technology pLatform for the Assisted living of Dementia elDerly INdividuals and their carers. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pages 878–881.

[Perttunen et al., 2009] Perttunen, M., Riekki, J., and Lassila, O. (2009). Context representation and reasoning in pervasive computing: a review. *International Journal of Multimedia and Ubiquitous Engineering*, 4(4):1–28.

[Picard, 2005] Picard, G. (2005). Cooperative agent model instantiation to collective robotics. In *Engineering Societies in the Agents World V: 5th International Workshop, ESAW 2004, Toulouse, France, October 20-22, 2004: Revised Selected and Invited Papers*. Springer.

[Preuveneers and Berbers, 2008] Preuveneers, D. and Berbers, Y. (2008). Encoding semantic awareness in resource-constrained devices. *IEEE Intelligent Systems*, 23(2):26–33.

[Prokopenko, 2007] Prokopenko, M. (2007). Design vs self-organization. *Advances in applied self-organizing systems*, pages 3–17.

[Prokopenko, 2008] Prokopenko, M., editor (2008). *Advances in Applied Self-Organizing Systems*. Springer-Verlag, London, UK.

[Ramos et al., 2008] Ramos, C., Augusto, J. C., and Shapiro, D. (2008). Ambient intelligence - the next step for artificial intelligence. *IEEE Intelligent Systems*, 23(2):15–18.

[Randles et al., 2006] Randles, M., Taleb-Bendiab, A., and Miseldine, P. (2006). Harnessing complexity: A logical approach to engineering and controlling self-organizing systems. *International Transactions on Systems Science and Applications*, 2(1):11–20.

[Randles et al., 2007] Randles, M., Zhu, H., and Taleb-Bendiab, A. (2007). A formal approach to the engineering of emergence and its recurrence. *Proc. of EEDAS-ICAC*, pages 1–10.

[Ranganathan and Campbell, 2003] Ranganathan, A. and Campbell, R. (2003). A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 143–161. Springer-Verlag New York, Inc.

[Rao and Georgeff, 1995] Rao, A. S. and Georgeff, M. P. (1995). BDI agents: From theory to practice. In Lesser, V. R. and Gasser, L., editors, *Proceedings of the ICMAS-95, First International Conference on Multiagent Systems, June 12-14, San Francisco, California, USA*, pages 312–319. San Francisco, CA, The MIT Press.

[Ricci et al., 2007] Ricci, A., Omicini, A., Viroli, M., Gardelli, L., and Oliva, E. (2007). Cognitive stigmergy: Towards a framework based on agents and artifacts. *Lecture Notes in Computer Science*, 4389:124–135.

[Riva et al., 2005] Riva, G., Vatalaro, F., Davide, F., and Alcañiz, M., editors (2005). *Ambient Intelligence*. IOS Press Amsterdam.

[Robinson et al., 2007] Robinson, R., Henricksen, K., and Indulska, J. (2007). XCML: A runtime representation for the context modelling language. *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on*, pages 20–26.

[Sadeh et al., 2005] Sadeh, N. M., Gandon, F. L., and Kwon, O. B. (2005). Ambient intelligence: The MyCampus experience. Technical Report CMU-ISRI-05-123, School of Computer Science, Carnagie Mellon University.

[Satoh, 2004] Satoh, I. (2004). Mobile agents for ambient intelligence. In *Proceedings of Massively Multi-Agent Systems I, First International Workshop, MMAS 2004, Kyoto, Japan, December 10-11, 2004, Revised Selected and Invited Papers*, volume 3446 of *Lecture Notes in Computer Science*, pages 187–201. Springer.

[Satyanarayanan, 2001] Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17.

[Schmidt, 2006] Schmidt, A. (2006). Ontology-based user context management: The challenges of imperfection and time-dependence. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pages 995–1011.

[Sen and Airiau, 2007] Sen, S. and Airiau, S. (2007). Emergence of norms through social learning. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 1507–1512.

[Serugendo et al., 2006] Serugendo, G. D. M., Gleizes, M.-P., and Karageorgos., A. (2006). Self-organization and emergence in MAS: An overview. *Informatica*, 30(1):45–54.

[Shalizi, 2001] Shalizi, C. (2001). *Causal architecture, complexity and self-organization in the time series and cellular automata*. University of Wisconsin–Madison.

[Soler et al., 2010] Soler, V., Peñalver, A., Zuffanelli, S., Roig, J., and Aguiló, J. (2010). Domotic hardware infrastructure in PERSONA project. In *Ambient Intelligence and Future Trends-International Symposium on Ambient Intelligence (ISAmI 2010)*, pages 149–155. Springer.

[Sowa, 2000] Sowa, J. (2000). *Knowledge representation: logical, philosophical, and computational foundations*. MIT Press.

[Spanoudakis and Moraitis, 2006] Spanoudakis, N. and Moraitis, P. (2006). Agent based architecture in an ambient intelligence context. *Proceedings of the 4th European Workshop on Multi-Agent Systems (EUMAS'06), Lisbon, Portugal*, pages 1–12.

[Standish, 2001] Standish, R. (2001). On complexity and emergence. *Arxiv preprint nlin.AO/0101006*, pages 1–6.

[Strang and Linnhoff-Popien, 2004] Strang, T. and Linnhoff-Popien, C. (2004). A context modeling survey. *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp*, pages 1–8.

[Suna and El Fallah Seghrouchni, 2004] Suna, A. and El Fallah Seghrouchni, A. (2004). Programming mobile intelligent agents: An operational semantics. *Web Intelligence and Agent Systems*, 5(1):47–67.

[Tapia et al., 2010] Tapia, D., Abraham, A., Corchado, J., and Alonso, R. (2010). Agents and ambient intelligence: case studies. *Journal of Ambient Intelligence and Humanized Computing*, 1(2):85–93.

[Tapia et al., 2009] Tapia, D., Rodríguez, S., Bajo, J., and Corchado, J. (2009). FUSION@, a SOA-based multi-agent architecture. In *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, pages 99–107. Springer.

[Ullmann, 1976] Ullmann, J. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42.

[Unsal and Bay, 1994] Unsal, C. and Bay, J. (1994). Spatial self-organization in large populations of mobile robots. *Proceedings of the 1994 IEEE International Symposium on Intelligent Control*, pages 249–254.

[Vallée et al., 2005] Vallée, M., Ramparany, F., and Vercouter, L. (2005). A multi-agent system for dynamic service composition in ambient intelligence environments. *Advances in Pervasive Computing, Adjunct Proceedings of the Third International Conference on Pervasive Computing (Pervasive 2005)*, pages 1–8.

[Viterbo et al., 2008] Viterbo, J., Mazuel, L., Charif, Y., Endler, M., Sabouret, N., Breitman, K., El Fallah Seghrouchni, A., and Briot, J. (2008). Ambient intelligence: Management of distributed and heterogeneous context knowledge. *CRC Studies in Informatics Series. Chapman & Hall*, pages 1–44.

[Weiser, 1993] Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications - ACM*, pages 74–87.

[Weiser, 1995] Weiser, M. (1995). The computer for the 21st century. *Scientific American*, 272(3):78–89.

[Wright et al., 2008] Wright, D., Gutwirth, S., Friedewald, M., Vildjiounaite, E., and Punie, Y. (2008). *Safeguards in a world of ambient intelligence*, volume 1. Springer-Verlag New York Inc.

[Zambonelli et al., 2004] Zambonelli, F., Gleizes, M., Mamei, M., and Tolksdorf, R. (2004). Spray computers: Frontiers of self-organization for pervasive computing. *Proceedings of the 13th IEEE Int'l Workshops on Enabling Technologies, WETICE*, pages 403–408.