Artificial Intelligence and Multi-Agent Systems Laboratory

University "Politehnica" of Bucharest

Université Pierre et Marie Curie Paris

Towards a MAS-Based Model for Ambient Intelligence

1st PhD Research Report

Andrei Olaru

AI-MAS Laboratory, Computer Science Department, University "Politehnica" of Bucharest
in co-tutelle with
Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie

PhD Thesis Title: A Context-Aware Multi-Agent System for AmI Environments

Supervisors:
Prof. Adina Magda Florea
AI-MAS Laboratory, Computer Science Department, University "Politehnica" of Bucharest
Prof. Amal El Fallah Seghrouchni
Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie

September 2010

Contents

1 Introduction

1.1 What is Ambient Intelligence?

The term of Ambient Intelligence has been coined at the dawn of the 21st century, in 2001 [DBS+01], 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 ((from spoken, wise advice to traffic and weather control).

Ambient intelligence should represent the third wave in computing [RVDA05]. 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.

The origins of Ambient Intelligence lie in the domain of Pervasive Computing, or Ubiquitous Computing – or UbiComp. These terms were introduced by Weiser in 1991 [Wei95, Wei93]. The elements of UbiComp come from previously existing domains like Distributed Computing and Mobile Computing, adding the idea of smart spaces, invisibility, localization and dealing with uneven conditions [Sat01] (see Figure 1).

While Ubiquitous Computing means that people are surrounded with devices that offer different types of services, Ambient Intelligence emphasizes the idea of *intelligent* aggregation of services and action, the idea of a proactive and invisible system that helps the user, and the existence of intelligent interfaces with which the user can interact in an intuitive manner.

The realization of Ambient Intelligence as Mark Weiser [Wei95] or the ISTAG Group [DBS+01] have seen it is still a long way ahead, many features of Ambient Intelligence environments have been identified, as presented below.

By being a large system, an AmI environment must necessarily be organized on layers. The same happens with the Internet today: by being a distributed, decentralized, large, flexible and multi-purpose system, it is organized on layers. For AmI the following layers have been proposed [Seg08] (see Figure 2):

- 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.

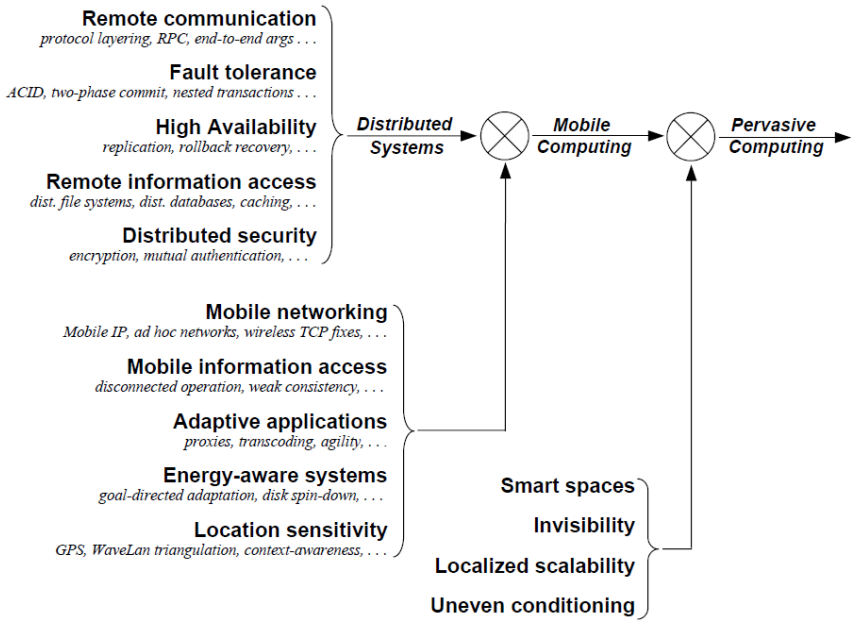


Figure 1: Elements in the Pervasive Computing approach (from [Sat01]).

- the **interoperability** layer is vital to AmI, in order for devices to be able to communicate freely, using uniform protocols above this level.
- the **intelligent services** layer is the layer that makes AmI truly "intelligent". 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.
- the last layer is the **interface**: gestures, speech, voice recognition, face recognition 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.

All of the layers presented above pose many specific challenges to researchers. But as a whole, AmI must have two important features: first, be **proactive** and **context-aware**, taking the right action at the right time; second, remain **non-intrusive**, by not disturbing the user and letting the user focus on the activity rather than on the interface with the computer [RVDA05]. Also, considering the amount of personal information that AmI environments will manage in a real case, it is essential that AmI systems are **secure** and **privacy-aware**. While all the layers of an AmI environment must contribute to obtaining these features, it is the layers of Intelligent Services that has the most of the contribution in taking pro-active, context-aware decisions that appropriately assist the user.

1.2 Scenarios

In the domain of Ambient Intelligence, research goals are many times driven by scenarios that help envisage a world enriched by ambient, pervasive, intelligent services [DBS+01, Wei95, Sat01, BB02, KBM+02, VRV05]. So far, scenarios have most times presented the

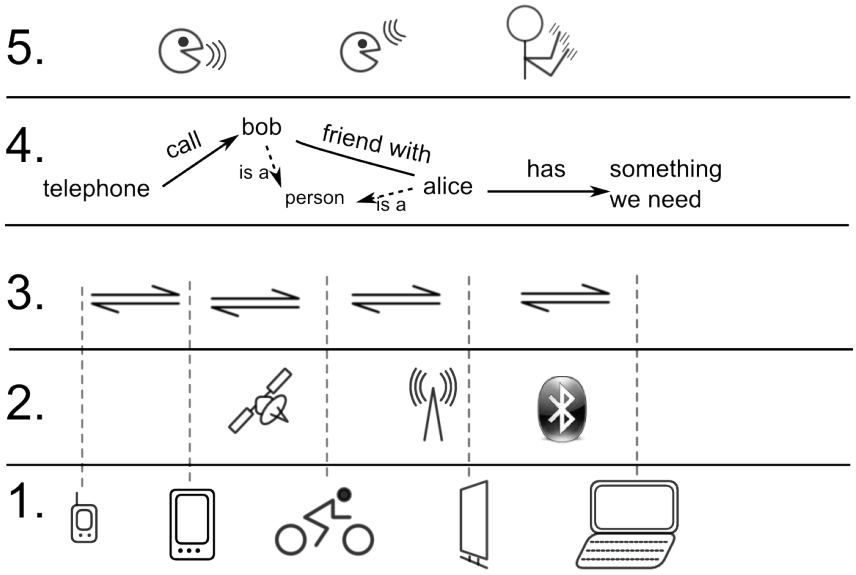


Figure 2: Layers of an AmI environment: hardware, interconnectivity, interoperability, intelligent services, advanced interfaces (based on [Seg08]).

perception that one person has upon the system, with few details on how the system should work in the background in order to deal with realistic requirements.

When thinking about Ambient Intelligence, it is important to remember the number of devices that will be composing AmI environments, as well as the number of people that will be using them. This is what, in the course of our research, led us to try to build scenarios that are more oriented towards distributed, scaling implementation [OSF10]. We present one of them below:

> On the largest stadium of an European capital, a concert is going to be held, by a popular rock group 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. All participants share the space, time and activity context. 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.
>
> 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 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 and the group can meet again.

1.3 AmI and MAS

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 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 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 [Fer99] can contributed to the realization of AmI. Agents offer features that are very much needed by AmI, like reactivity, autonomy, pro-activity, the possibility of reasoning and anticipation [RAS08]. 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 design from the local point of view and the global behaviour is emergent.

While many other researchers used software agents for the implementation of AmI environments, there are usually two approaches that each is different from ours. One of the approaches uses a small number of agents that retain user preferences and query context information, but this approach are 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.

1.4 Research Approach

The goal of my PhD thesis is to design and implement a context-aware multi-agent system, as an implementation for an AmI environment. Considering the realistic scale that is needed for such an environment, our research approach is based on the following elements: distribution of the system, use of software agents for all devices and services in the system, self-organization of the agents in order to obtain emergent behaviour, and the appropriate,

context-aware representation of knowledge in order to obtain the adaptive and anticipative behaviour of the system [Ola10].

The purpose of this report is to go into details with respect to several elements of this approach, namely the use of software agents and the use of mechanisms of self-organization in order to obtain emergent behaviour among the agents.

The following section presents a state of the art of the use of MAS for Ambient Intelligence, updated from the past report [Ola10]. Section 3 discusses the implementation of a MAS for information exchange and sharing, based on self-organization, and section 4 describes the Ao Dai prototype of a context-aware MAS, implemented in CLAIM. The last two sections are dedicated to the conclusions and pointers to future work.

2 Implementing AmI Environments Using Multi-Agent Systems

2.1 A State of The Art

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 Figure 2, most of these approaches use agents, indeed, at the level of the "intelligent" layer – the layer situate under the interface. Why is that?

This upper layer that resides below the interface is vital for the "intelligent" features of AmI. It is this layer that must work with knowledge and semantic information and that must assure that users get the information that they need, where and when they need it. It is here where AmI must use the achievements of Artificial Intelligence, trying to model, understand, anticipate and, finally, assist the people in the real world. For these tasks, AmI must feature sensing capabilities, autonomy, reasoning, proactivity, social abilities and learning [RAS08]. Considering these features, one especially appropriate paradigm for the implementation of this layer in an AmI system is the agent-oriented paradigm.

In the field of agent-based Ambient Intelligence platforms there are two main directions of development: one concerning agents oriented towards assisting the user, based on centralized repositories of knowledge (ontologies), and one concerning the coordination of agents associated to devices, and potentially their mobility, in order to resolve complex tasks that no agent can do by itself, also considering distributed control and fault tolerance.

The first approach is closer to Intelligent User Interfaces and local anticipation of user intentions, coming from the field of intelligent personal assistants. For instance, embedded agents form an AmI environment in the iDorm implementation [HCC+04]. 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 [CFJ+04] 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 [SGK05] 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 [SM06] 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 [CMTT08] 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.

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 [Sat04] 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 [CFLZ05] 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 behaviour 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 [LW05] 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.

Project Name	knowledge representation	ontologies	context-awareness	learning	security - privacy	mobile agents	scalability	flexibility	centralized	FIPA-compliant
iDorm [HCC+04]	-	-	-	Yes	-	-	?	-	Yes	-
Spatial Agents [Sat04]	-	-	-	Yes	Yes	Yes	?	-	Yes	-
EasyMeeting [CFJ+04]	Ont.	SOUPA	Yes	-	Yes	-	?	Yes	Yes	Yes
SodaPop [Hel05]	-	-	-	-	-	-	Yes	Yes	No	-
LAICA [CFLZ05]	-	-	-	-	-	-	Yes	Yes	partial	-
MyCampus [SGK05]	CBR	Yes	Yes	Yes	Yes	-	Yes	Yes	Yes	some
AmbieAgents [LW05]	CBR	Yes	Yes	-	Yes	-	Yes	-	partial	Yes
ASK-IT [SM06]	-	some	Yes	some	-	-	-	No	Yes	Yes
CAMPUS [SBS+08]	Ont.	Yes	Yes	some	-	-	Yes	Yes	No	Yes
Dalica [CMTT08]	tuples	Yes	-	-	-	-	-	-	partial	-

Table 1: Features of the systems described in Section 2.1: 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; compliancy with FIPA protocols.

The CAMPUS framework [SBS+08] considers issues like different types of contexts [CK00] 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 [HK04, Hel05] 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.

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 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, autonomy, 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 [SBS+08, SGK05]) is a better way to deal with the design of such a complex system as a flexible, generic Ambient Intelligence environment.

2.2 Context-Awareness

It is hard 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 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.

Ever since the first works on context-awareness for pervasive computing [DAS99], certain infrastructures for the processing of context information have been proposed [HL01, HHS+02, LW05, HI06, BDR07, FAJ04]. There are several layers that are usually proposed, going from sensors to the application: sensors capture information from the environment, there is a layer for the preprocessing of that information, the layer for its storage and management, and the layer of the application that uses the context information [BDR07]. This type of infrastructures is useful when the context information comes from the environment and refers to environmental conditions like location, temperature, light or weather. However, physical context is only one aspect of context [CK00]. Moreover, these infrastructures are usually centralized, using context servers that are queried to obtain relevant or useful context information [DAS99, LW05]. In our approach [OSF10], 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.

Modeling of context information uses representations that range from tuples to logical, case-based and ontological representations [PRL09, SLP04]. These are used to determine the situation that the user is in. Henricksen et al use several types of associations as well as rule-based reasoning to take context-aware decisions [HI06, BBH+10]. However, these approaches are not flexible throughout the evolution of the system – the ontologies and rules are hard to modify on the go and in a dynamical manner. While ontologies make an

excellent tool of representing concepts, context is many times just a set of associations that changes incessantly, so it is very hard to dynamically maintain an ontology that describes the user's context by means of a concept. In this paper we propose a more simple, but flexible and easy-to-adapt dynamical representation of context information, based on concept maps and conceptual graphs. While our representations lacks the expressive power of ontologies in terms of restrictions, a graph-based representations is very flexible and extensible, so support for restriction may be added as future work.

In this work we will present two systems that use different approaches regarding context-awareness: in one, context is modeled by some simple and generic measures of context, that are attached to the pieces of information – the measures describe the urgency of the information, its validity and its relatedness to certain domains of interest (see Section 3.4); in the other, context is modeled implicitly by using a hierarchical structure for agents, that is mapped against the different types of contexts that are considered by the system (see Section 4).

3 Self-Organizing Agents for Ambient Intelligence

3.1 The Problem

We see an Ambient Intelligence environment like 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. The problem that we asked is: given a certain piece of information, how to deliver that piece of information to the interested users – the users to which that information is relevant? This is a problem that is addressed to the intelligent services layer of an AmI environment and can be solved by a middleware, between the lower layers of the environment and the applications that need the information. We have named this middleware AmIciTy:Mi , as a part of the AmIciTy Ambient Intelligence environment that we are in the process of building.

The design of the MAS that is presented in this section 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 implementation is based on two ideas: the use of agents and the application of mechanism of self-organization. In this project we tried to define a structure for individual agents that will allow them, by means of large numbers and intense interaction, to fulfill the global desired goal – the context-aware sharing of information – and this by means only of limited knowledge and reasoning, and local behaviour and communication.

Let us explain the purpose of the system more clearly: a large number of agents is given, that have a location in space and cover a certain rectangular area. The agents can only communicate locally, with their neighbours. A user of the system can insert a piece of data into the system, that contains indications of the domains to which it is of interest, of its importance, and of its validity. The purpose of the system is to distribute the said piece of knowledge among the agents in the system, so that other users – in other locations – that are interested in the data will be able to know it.

The process should work for many pieces of data with different parameters, and this should all be done using reduced resources in terms of memory and processing.

3.2 Design Rationale

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 the quantity of information that will be exchanged. Agents are one of the paradigms that can be used for the implementation of distributed. Moreover, their qualities – such as autonomy and pro-activity – make them especially appropriate for the implementation of AmI [RAS08].

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 be necessarily 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 [CFLZ05]. Moreover, self-organization allows a system to be robust and fault-tolerant [Hey02], 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. This project also was an experiment to see what kind of emergent properties can be obtained in a system formed of cognitive, as opposed to reactive, agents [OF09].

Why local behaviour? First, it will be very difficult to mange 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 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 (also see section 3.4). We see context as vicinity in a domain that considers space, time, social relationships, computing resources and actions / intentions.

3.3 System Structure

The AmIciTy middleware is being developed keeping in mind the scale of a real Ambient Intelligence scenario. In such a situation, there is a huge number of users and (possibly "intelligent") devices: sensors, actuators and more advanced human-machine interfaces. These devices communicate permanently and exchange a huge 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 must be completely distributed. Each agent in the middleware is assigned to and is executed on a device. There might me 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 wireless.

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. 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

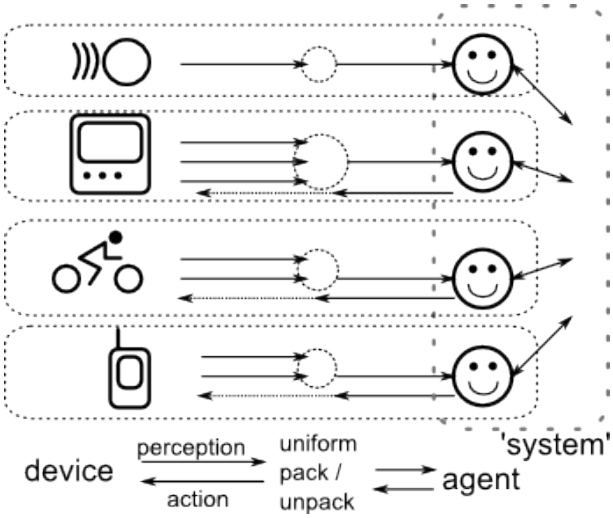


Figure 3: The structure of the middleware, as seen from the perspective of the devices. The "system", i.e. the middleware, is actually formed of the agents that compose it. There is a packing / unpacking step in the interface - agent communication so that the communication will be uniform over all the middleware. The devices and the agents are circled together to point out that the agents are executing on the devices, and are using the devices' communication hardware. Although not shown in the figure, there may also be more agents per device

an entity that is formed by the totality of the agents composing it.

In the current implementation of AmIciTy:Mi , we have focused on the realization of the agents and of their internal workings, as well as the knowledge representation, context measures and interaction protocol. As a result, the agents are running in a simulated environment where they are aware of their neighbours and they can receive pieces of data from the exterior, with the purpose of sharing them with the other agents.

A practical application to the studied system would be, for instance, a network of sensors in a building. The sensors have low computational capabilities so their behavior must be simple and use little memory. They sense events in the environment, like temperature or the presence of people. The system is completely decentralized, but information must be available for retrieval from any of the sensing devices. Therefore, sensors aggregate their sensor information into larger chunks of data and "insert" it into the system, which, by using only local information and communication between neighbour sensors, distribute the data and make it uniformly available throughout the building. This is only an example. The system is generic and was built to fit a greater number of possible applications.

3.4 Context-Awareness

By context we understand the conditions in which an event occurs and that are related to the event [CK00]. Context-awareness is the feature of a system (here, an AmI system) that makes the system behave differently in function of these conditions.

Being context-aware means providing to the user information which is *compatible* with

the context of the user. More precisely, the context of the information must be compatible with the context of the user in order for the information to be *relevant*. In our experiments, compatibility is actually between the information and the agent's context.

To provide context-awareness for information sharing, we propose four simple and generic aspects of context-awareness: first, space is implicitly considered, because of the structure of the system, that relies on local behaviour 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 (estimated by the source).

We see context compatibility as *proximity* between the two contexts – the context of the new information and the context of agent – in terms of the aspects enumerated above. 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 behaviour 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). In function of the other aspects of context-awareness, the information will only stay in the area or will spread further. Also, all other measures equal, agents will give less relevance to information related to a farther location.

Time persistence – shows for how long is the information 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 in function of 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 neighbours the agent will send the information to. This way, pressure controls how quickly the information spreads.

Context compatibility, or *relevance* of new information is calculated in function of the measures of context associated with the new information and in function of the context of the agent, comprised only 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 (see the next section), and is represented in the interval [0, 1], with 0 meaning it refers to *this* agent and asymptotically growing to one for longer distances; persistence is a value in the interval [0, 1], with 1 meaning the information is valid forever, and 0 meaning it has expired; specialty is a vector in which each component has a value in the interval [0, 1] showing the degree of relatedness with a certain domain of interest, and the whole vector has a maximum norm of 1; pressure is also a value in the interval [0, 1].

When calculating 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 calculated 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 calculated as a mean of the 4 numbers – all in the same interval – distance, persistence, pressure and similarity. 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).

3.5 Agent Structure

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 a simple processor assigned to a sensor and on a powerful computer. The agents are cognitive, but hold little knowledge and form very simple plans. 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 behaviour 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 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. At this point, the system is generic and does 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. The *recursive depth* of a fact gives its distance to the source of the information, which is used in calculating relevance.

There are also other pieces of knowledge that an agent has but are not represented in the knowledge base. This is the knowledge of the agent's neighbours. Direct neighbours are stored in the neighbour list. But the agent also knows about the existence of other, farther, agents, from the facts in its knowledge base. Also refer to Figure 4 (b).

This structure allows the agent to hold information about what it knows but also about

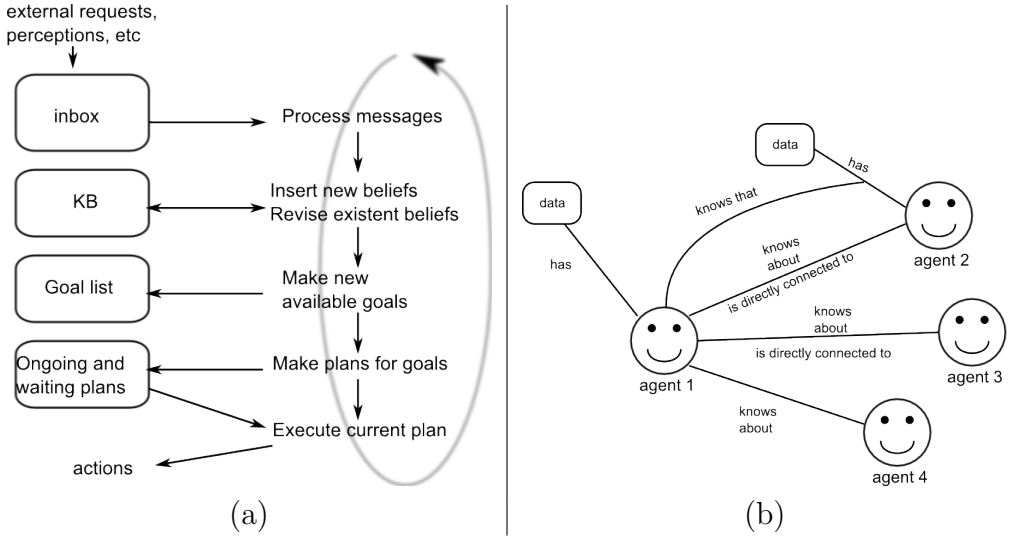


Figure 4: (a) The basic execution cycle of an agent. (b) Example of content of an agent's (the one in the center) knowledge base. Several relationships are displayed, like *knows about*, *is connected to*, *has*, *knows*

what other agents know. This is how an agent can calculate the Specialty of neighbour agents.

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

3.6 Agent Behaviour

At the beginning of each cycle (see Figure 4 (a)) 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 neighbours.

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 importance of *Inform* goals is taken from the relevance of the information to be sent. Importance for the *Free* goal is calculated in function of 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 neighbours to inform of the corresponding fact. The number of neighbours 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 neighbour has. At each cycle

the agent will execute one action in its current plan.

So, the stages that an agent goes through during a step of the system's evolution are:

- **Receive messages** from neighbour agents. There exist only *Inform* messages.
- **Revise beliefs**. This is done based on information received from the other agents. Facts that other agents inform about are integrated in the knowledge base. Duplicate or circular facts are found and removed. Context measures of facts are not modified.
- **Check ongoing or waiting plans** for completion (was the goal achieved?) and test whether they have become impossible. In the case of completion the plan is discarded and the pressure of the corresponding facts is canceled. The interest towards the facts, on the other hand, remains constant.
- **Make plans**. First, update the list of available goals: add or update the *Free* goal if necessary; update and add *Get* goals for the facts in the knowledge base. Then, take the goal with the highest current importance. If there is no plan for it, make a plan composed of the actions needed to be taken and put it in the list of ongoing plans.
- **Execute plans**. Take the plan associated with the most important goal and execute its next action (if the plan is not waiting). If there are no more actions to be performed, move the plan to the list of waiting plans. It will be checked for completion in the next cycle.
- **Fade memory** of all facts in the knowledge base. Pressure of all facts is faded. Facts in which the agent has no interest, whose persistence has reached zero, or that have low pressure may be discarded ("forgotten"). This step is necessary in order to avoid overwhelming the agent with useless facts and too old concerns.
- **Revise pressure and interest**. The pressure on the agent is recalculated according to the facts in its knowledge base, as a weighted mean of the pressures of the individual facts, giving more importance to high-pressure facts. The specialty is updated as well, according to the pieces of data that the agent holds and according to its recent activity (the resulting specialty is also calculated as a mean).

The behaviour of the agent changes in function of its context measures. On the one hand, 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"). Then, the agents update their own specialty according to the facts that they have.

Feedback. There is an important aspect of feedback that exists in the AmIciTy:Mi . First, the way in which agents update specialty: agent's specialty influences the facts that they consider relevant and send, sent facts influence other agent's specialty and knowledge, and in return they receive facts with a certain specialty. Second, pressure influences relevance directly, so feedback loops form, caused by information with high pressure. Feedback leads to an aspect of self-organization by the formation of emergent specialty groups.

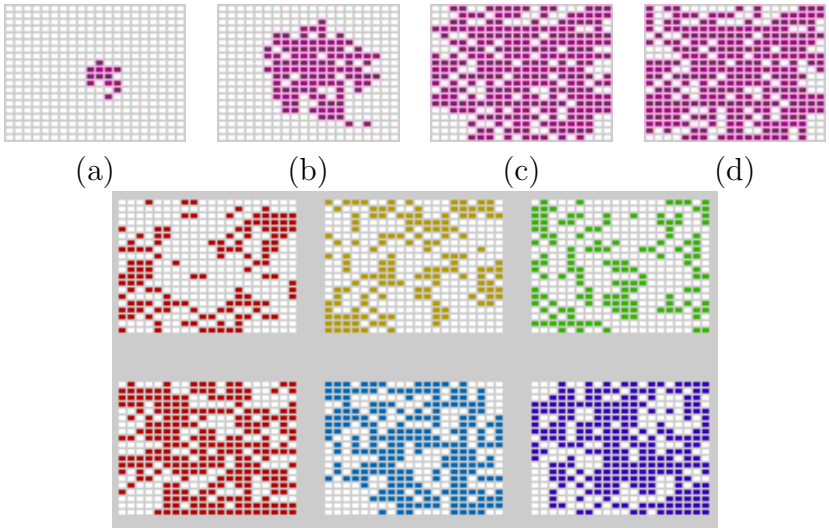


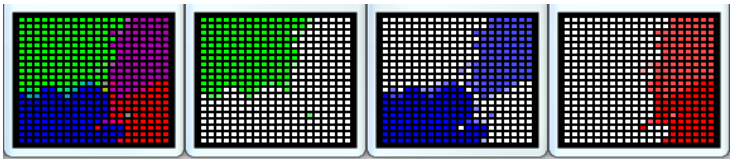
Figure 5: (a) - (d) The distribution of a certain piece of data over several steps of the system's evolution, in a system already holding 4 other pieces of data. (e) The simultaneous distributions of 6 pieces of data after 150 steps. (from [OGF10b])

3.7 Results

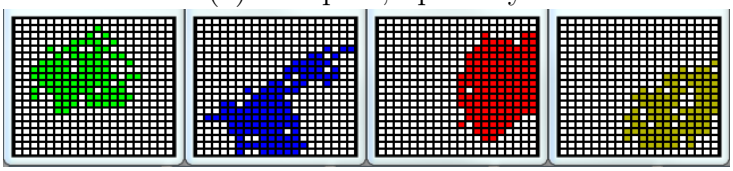
The project called AmIciTy:Mi started with a multi-agent system for the distribution of data [OGF09b, OGF10b]. The idea was to have many cognitive, but limited, agents that, as a system, would retain a number of pieces of data. Even if a certain agent did not have all the pieces of data at a certain time, nor did it have much information about the area around it, the system had the emergent property that the data was well distributed and did not get lost – although agents were able to remove data from their storage.

The multi-agent system was implemented as a Java application, running on a single machine. At each step in the simulation, each agent executes one cycle – however, each agent only receives the messages that were sent in the previous step. This implementation was chosen because the purpose was to study the behaviour of the system, in conditions that will allow maximum performance and that let us focus on the properties of the system rather than on implementation issues.

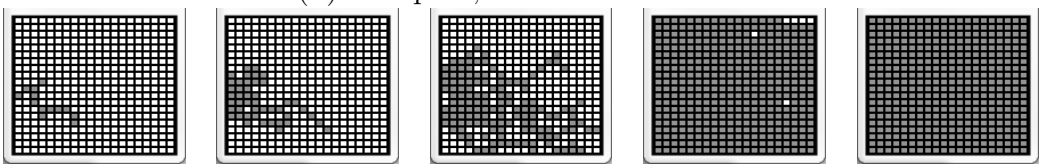
Figure 5 shows some results obtained in this first phase of the project: the evolution of the distribution of a certain piece of data, as well as the simultaneous distributions of 6 pieces of data. In the figure, each cell represents an agent, and the colour of the cell is related to the data to which the graph corresponds. In Figure 5, although there are "holes" in the distribution, i.e. there are agents that do not hold a certain piece of data, the system changes all the time, so the "holes" move incessantly in the distribution. In the last result, where there are 6 pieces of data in the system, agents have capacity of 4, and try to keep a quarter of their capacity free for potential new data, so it is normal that the pieces of data are divided approximately in half. The results were obtained on a system using 400 agents. The scenario was fairly simple: insert each piece of data, at a certain time, into one single agent in the system (which may have been in a corner of the grid or in tis center).



(a) – step 20, Specialty

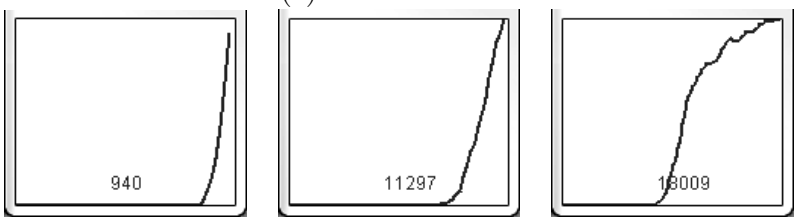


(b) – step 35, Facts for $D5 - D8$



step 42 step 45 step 48 step 60 step 64

(c) – facts for $D9$



steps 0-48 steps 0-60 steps 0-100

(d) – fact number for $D9$

Figure 6: (a) Agent interest for domains A, B and C. In the first image interest in all domains is represented. Next three images represent the interest for one domain of interest each, respectively. (b) Resulting distributions for facts of interest to domains A, B, C and $A + C$, respectively. (c) Evolution of the distribution of a fact with high pressure, starting from the left side of the grid. (d) The number of facts in the system referring the high-pressure information; the number represents the maximum value in the graph. (from [OGF10a])

A subsequent phase in the project's development moved towards the idea of context-awareness [OGF10a, OGF09a], as described in Section 3.4, but only considering two measures of context: *pressure* and *interest* (*interest* later became *Specialty*). Agents were able to form interest towards certain domains and then they would be more interested in information about those particular domains. However, agents were still placed in a rectangular gird, and their internal algorithm was fairly complicated. See some results in Figure 6 (a) and (b): first, some facts are inserted into the system (not shown), and the agents form certain interests. Then, when a new series of facts is inserted, it is easy to see that they follow the previously formed interests. Pressure of the facts acts not i terms of direction, but in terms of speed. Observe in Figure 6 (c) that the facts with high pressure spread in the whole system in just 20 steps. The number of agents having the facts risis exponentially (Figure 6 (d)).

These results were obtained using the following scenario: first, insert into the system 4 pieces of information, with different Specialties, in the corners of the grid, that will

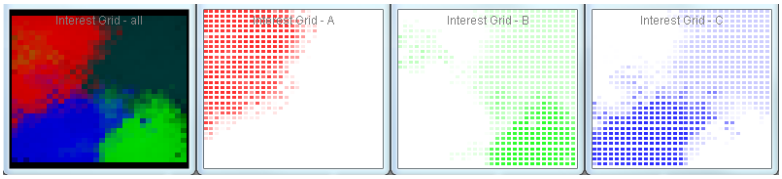
form agents' Specialties. Once agent having clear Specialties, insert another 4 pieces of information, this time in the center of the grid, and see if they follow the previously formed Specialties. For instance, in Figure 6, it is easy to observe that the facts regarding domain A (in green) spread towards the top-left corner of the grid; the facts regarding domain B (in blue) followed two directions: top-right and bottom-left; facts regarding domain C (in red) spread towards the right side of the grid; and facts regarding both domains A and C spread towards the bottom-right corner (where there was interest for domain C) – they don't reach the top-left part because agents in the center are already 'busy' with the facts regarding domain B. The last part of the scenario concerns inserting a fact with no particular Specialty (in gray), but with very high pressure – this spreads quickly, covering the grid in just 20 steps, and also does not influence the distributions of the other facts.

But it is the latest version of the project that yielded the most interesting results [OG10]. These results were obtained by the simplification of the structure and functioning of the agents, and also by the tweaking of the constants in their algorithm in order to obtain better results. We will present in the following paragraphs a few details on the implementation, and the results that we have obtained.

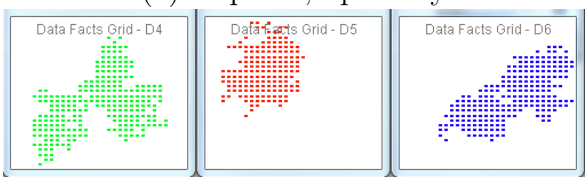
In this last version, we have tried to simplify as much as possible the cycle of the agent: this way, it will be easier to execute the agent on a simpler device. Therefore, an agent A does the following:

- agent A receives at most m messages (m is decided in function of the pressure upon the agent – if pressure is high the agent takes at least 2 messages; otherwise, it may take more, or even all messages). Messages only contain facts of which other agents want to inform A;
- for each fact F received (which is of the form $\langle Agent,\ knows,\ \langle Agent,\ knows,\ \ldots \langle Agent,\ knows,\ Fact \rangle\rangle\rangle$), agent A stores two facts: 1 is the $Fact$ – the actual information, that the agent now knows. The other is F, containing data about what the sending agent knows, and, potentially nested inside, about what other agents know;
- the agent checks its plans for completion;
- the agent fades pressure and persistence of the facts in its knowledge base; it calculates the pressure upon itself by the average of pressures of the facts that it knows; also its Specialty is updated, considering new facts that have been received;
- agent A makes a plan for its most important goal. Goals can be $Free$ goals, or $Inform$ goals (see Section 3.6);
- the agent takes one action from the plan for its most important (pressing) goal, and executes it.

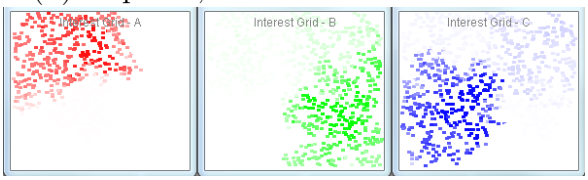
The results were of the same nature as the ones before (Figure 6), however many improvements were obtained. First, the performance of the system increased considerably: the system executed 200-250 steps in the multi-agent system's time (in one step, each agent carries out one cycle) in about 10 seconds on a machine with an Intel Core 2 Duo



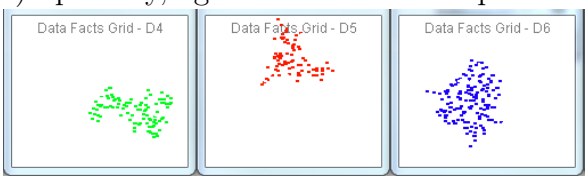
(a) step 120, Specialty



(b) step 230, Distribution of new facts



(c) Specialty, agents are in random positions



(d) Respective fact distribution

Figure 7: (a), (b) Specialty of agents for all domains and domains A, B and C, and the distribution of facts, 100 steps later, for facts with Specialties related to domains B, A and C. (c), (d) For random locations of agents, simultaneous snapshots of agent specialties and respective facts.

3.33GHz CPU, with 2GB of RAM memory, considering that there are 1000 agents (in fact 31^2) in the system.

This performance is based not only on the very short cycle of the agent, but also on a special data structure, implemented by the authors, called a QuickSet: a set of values that can be ordered, the set being kept sorted at all times; it allows updating the elements, but the elements don't have to be removed and re-added – modified elements move in the set at their new position, therefore performing only the operations that are strictly necessary.

Figure 7 presents the results obtained in this last version of the system. The test scenario that was used was slightly different: first, insert many facts, at random positions, with low pressure and persistence, and various specialties; then, insert 3 'test' facts, with medium pressure and high persistence, and with specialties regarding each domain; finally, insert one or two facts with high pressure.

As a further improvement, we have also implemented the possibility for the agents to have random positions. They communicate only with agents that are very close to them. The results were good, however, due to the fact that agents now usually have less then 8 neighbours (as it was the case with the grid layout), so spreading the information takes more time.

3.8 Future Work

Implementing and simulating the prototype of AmIciTy:Mi was an experience that led to many lessons learned. First, we have studied how a system of simple, but cognitive agents, can manifest emergent properties related to the context-aware distribution of information. Second, we have succeeded in implementing agents that are very simple, but successfully lead – by means of only local knowledge and actions – to a spreading of data that is coherent beyond their limited knowledge.

The future brings many challenges: further testing is necessary, also including agent mobility, and testing of fault tolerance (agents that can go out of the system, or disappear for certain amounts of time); better measures (other than visual results) must be developed in order to evaluate the system; distributing the system, so that it would execute on several machines, is also required to validate it; finally, execution of agents on various devices that move around will prove that the system is indeed worthy of being implemented at a larger scale.

Acknowledgment

I would like to thank my colleague Cristian Gratie that has an enormous contribution to the implementation of this project – especially in monitoring tools – and that has participated in the taking of many design decisions. Without him, the development of the project would have been much slower. I would also like to thank Sofia Neata, who has added some measures for the evaluation of the system.

4 A Context-Aware Multi-Agent System in CLAIM

This section presents a second approach to the implementation of Ambient Intelligence. If AmIciTy:Mi deals with the scalability of future AmI environments, the Ao Dai project – Agent-Oriented Design for Ambient Intelligence – studies in more detail the connection between agents and context-awareness, in which context is represented in a more advanced manner than in AmIciTy:Mi .

The Ao Dai project has been implemented in collaboration with Thi Thuy 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 machine, during the 5th NII-LIP6 Workshop, held in June 2010 in Paris, France.

4.1 CLAIM and Sympa

As an agent-oriented programming language, CLAIM [SEFS04] eases the task of implementing multi-agent systems. 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 following code shows a part of the definition of agent *PDA* in the Ao Dai project:

```
defineAgentClass PDA(?w,?h,?x_i,?y_i){
   authority = null;
   parent = null;
   knowledge = {location(?x_i,?y_i); type(1);}
   goals = null;
   messages = null;
   capabilities = {
      message = PDAatLoc (?name,?x_new,?y_new);
         condition = null;
         do{send(this,migrateTo(?name))}
         effects = null;
      }

      migrate{
         message = migrateTo(?name);
         condition = not(Java(PDA.isParent(this,?name)));
         do{send(this,removeOldNavi(?name))
.moveTo(this,?name).send(this,demandNavi(?name))}
         effects=null;
      }
      ...
   processes={send(this,starting())}
```

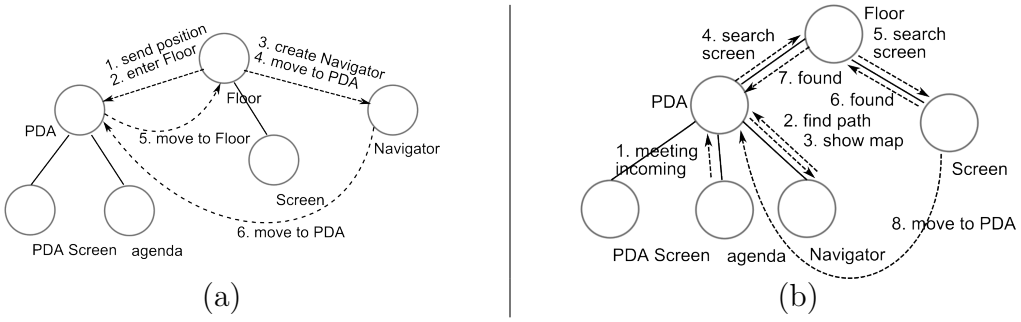


Figure 8: Sequences of messages exchanged between agents: (a) *Floor* announces *PDA* of its new position, and instructs it to move as its child, then creates a *Navigator* that will offer services to *PDA*; (b) *Agenda* announces a new meeting, *PDA* asks a path from *Navigator*, which in turn requires a larger screen – which is searched on the floor, and found, then *Screen* moves as a child of *PDA*.

```
    agents=null;
}
```

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 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, the language 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.

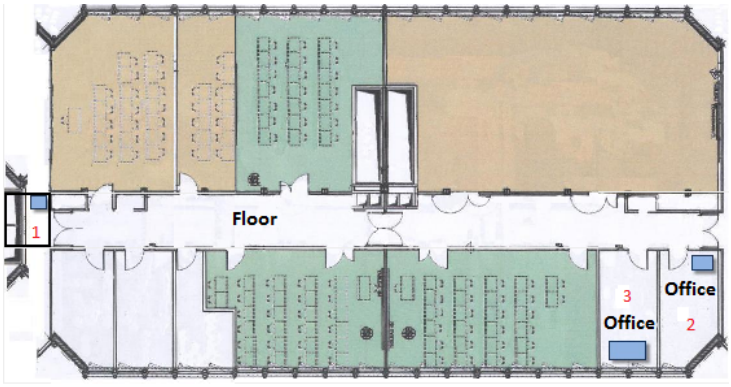


Figure 9: The map shown by different screens in Ao Dai. There are three *Site* agents: *Floor* and two *Office* agents. Each one has a child of type *Screen*, representing the screens in the different places. The user starts on the floor (1) then moves to one office (2) and then to the other (3).

4.2 Scenario

In this project, we have studied several scenarios including the following (also see Figure 8): 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 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,

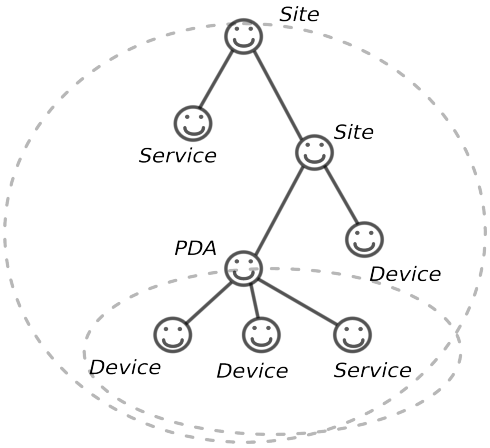


Figure 10: Example of an abstract logical hierarchy for the Ao Dai project: The root *Site* offers a *Service* and also contains another *Site* that contains a *Device*. The user is inside the second site, and its *PDA* can offer him the capabilities of two *Devices* and a *Service* (that may be a mobile agent executing on the PDA.)

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*.

4.3 System Architecture

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 [DA00], because it reflects an important set of physical contents. 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.

In the implementation of the scenario presented in Section 4.2, 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 10 and 8.

- 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 third type, *PDA* agent, as said before, 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, but is known that, in real applications, an additional layer is needed to capture the sensors information and translate then in useful data.

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 of Cardelli [CG00] 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 *PDA* agent is a child of the respective *Site* agent. The children of *PDA* – devices or services – are also in the same context. When the user moves to another room, the *PDA* agent changes parent and, along with it, its children move as 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 away from *PDA*.

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, a *Service* agent is created and becomes a child of *PDA*. It is easy for the service to interrogate its parent in order to find out more about its capabilities. Conversely, it is easy for *PDA* to check on its children – *Services* or, more important, *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 PDA (see Section 4.2). 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.

4.4 System Behaviour

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 withing 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 8). 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.

4.5 Future Work

The presentation and demonstration of the Ao Dai project at the NII-LIP6 Workshop have received very good reaction from both French and Japanese researchers. After subsequent discussions, it is very likely that the development of the Ao Dai project will be continued, jointly with a team from the NII Laboratory.

There are several future steps in this research. Among these, the integration of better mechanisms of anticipation, support for more types of contexts and improved context representation into the project will bring it closer to dealing with realistic requirements.

Acknowledgments

I would like to thank Thi Thuy Nga Nguyen and Diego Salomone, master students with the LIP6 laboratory, as well as prof. Amal El Fallah Seghrouchni, for implementing and, respectively, coordinating the development of this project. Excerpts from our paper accepted at the PRIMA 2010 conference [SONS10] have been included in this report.

5 Conclusion

The domain of Ambient Intelligence is a vast and still very new one. Problems raised by designing and implementing an AmI environment are many and complex. It is hard to even precisely specify the requirements and the necessary functionalities of such an environment.

During my PhD thesis, I study the possibilities and the potential offered by the use of software agents and multi-agent system in the domain of AmI. More precisely, considering a layered perspective on an AmI environment, the use of software agents in the composition of the intelligent services layer – that should deal with moving information between devices in such a way that the user will be assisted non-intrusively and in a manner that will appear intelligent, and that will be secure and privacy-aware. In all these issues, context-awareness is a fundamental keyword.

So far, this work has dealt with two issues. On the one hand, how to design an AmI environment that would be scalable and dependable. The answer may lie in the use of a multi-agent system based on principles of self-organization, in which agents – that can easily adapt to the computational power of the device it is running on – have little, and local knowledge, and execute only actions with a local effect, in order to obtain, at the level of the system, coherent distributions of information. The results we have obtained are encouraging, and the measures of context that were devised – although simple and generic – proved to be effective in controlling the spread of information among agents.

On the other hand, how to use a high-level, agent-oriented language for the design of a context-aware AmI environment. This time, the application featured fewer agents, but the implementation was more specific, and the context representation was more advanced: context-awareness was implicit, by mapping the contexts to logical structures of agents.

6 Future Work

As the central element of Ambient Intelligence seems to be context-awareness, it is clear that more work needs to be carried out in this direction. Future work will explore more advanced ways to represent context, while keeping the representation flexible and manageable by agents that execute on both powerful and tiny devices.

The main challenge of future research is to merge the features of the two applications that were developed. This way, it will be possible to build a system that has the advantages of both.

Another challenge is a better representation of context. We must try to find a representation that is generic – that is not confined to a specific application – but that is simple and flexible enough so that, depending on available resources, an agent can have variable levels in the detail and the accuracy of context representation.

Also as a part of future research there is the need to go from the level of simulation to the actual implementation of the presented concepts in agents that execute on real devices.

More concrete scenarios than the one presented in Section 1.2 will be developed, to act as validation scenarios for the AmIciTy project.

Acknowledgments

This work has been 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. Parts of this work have also been supported by Laboratoire d'Informatique de Paris 6 (LIP6), UPMC, Paris, France.

References

[BB02] G. Banavar and A. Bernstein. Software infrastructure and design challenges for ubiquitous computing applications. *Communications of the ACM*, 45(12):92–96, 2002.

[BBH+10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2010.

[BDR07] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[CFJ+04] H. Chen, T. Finin, A. Joshi, L. Kagal, F. Perich, and D. Chakraborty. Intelligent agents meet the semantic web in smart spaces. *Internet Computing, IEEE*, 8(6):69–79, 2004.

[CFLZ05] Giacomo Cabri, Luca Ferrari, Letizia Leonardi, and Franco Zambonelli. 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, 2005.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.

[CK00] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, November 2000.

[CMTT08] S. Costantini, L. Mostarda, A. Tocchio, and P. Tsintza. DALICA: Agent-based ambient intelligence for cultural-heritage scenarios. *IEEE Intelligent Systems*, 23(2):34–41, 2008.

[DA00] A.K. Dey and G.D. Abowd. 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, 2000.

[DAS99] A.K. Dey, G.D. Abowd, and D. Salber. A context-based infrastructure for smart environments. *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, pages 114–128, 1999.

[DBS+01] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.C. Burgelman. Scenarios for ambient intelligence in 2010. Technical report, Office for Official Publications of the European Communities, February 2001.

[FAJ04] Ling Feng, Peter M. G. Apers, and Willem Jonker. Towards context-aware data management for ambient intelligence. In Fernando Galindo, Makoto Takizawa, and Roland Traunmüller, 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, 2004.

[Fer99] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley, 1999.

[HCC+04] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman. Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems*, pages 12–20, 2004.

[Hel05] Michael Hellenschmidt. Distributed implementation of a self-organizing appliance middleware. In Nigel Davies, Thomas Kirste, and Heidrun Schumann, editors, *Mobile Computing and Ambient Intelligence*, volume 05181 of *Dagstuhl Seminar Proceedings*, pages 201–206. ACM, IBFI, Schloss Dagstuhl, Germany, 2005.

[Hey02] F. Heylighen. The science of self-organization and adaptivity. *The Encyclopedia of Life Support Systems*, pages 1–26, 2002.

[HHS+02] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. *Wireless Networks*, 8(2):187–197, 2002.

[HI06] K. Henricksen and J. Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37–64, 2006.

[HK04] M. Hellenschmidt and T. Kirste. A generic topology for ambient intelligence. *Lecture notes in computer science*, pages 112–123, 2004.

[HL01] J.I. Hong and J.A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2):287–303, 2001.

[KBM+02] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, and B. Serra. People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002.

[LW05] Till Christopher Lech and Leendert W. M. Wienhofen. 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, 2005.

[OF09] Andrei Olaru and Adina Magda Florea. 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, 2009. ISSN 2066-4451.

[OG10] Andrei Olaru and Cristian Gratie. Agent-based information sharing for ambient intelligence. In Mohammad Essaaidi, Michele Malgeri, and Costin Badica, 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, 2010.

[OGF09a] Andrei Olaru, Cristian Gratie, and Adina Magda Florea. 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, 2009. ISSN.

[OGF09b] Andrei Olaru, Cristian Gratie, and Adina Magda Florea. Emergent properties for data distribution in a cognitive MAS. In George Angelos Papadopoulos and Costin Badica, 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, 2009. ISBN 978-3-642-03213-4.

[OGF10a] Andrei Olaru, Cristian Gratie, and Adina Magda Florea. Context-aware emergent behaviour in a MAS for information exchange. *Scalable Computing: Practice and Experience - Scientific International Journal for Parallel and Distributed Computing*, 11(1):33–42, March 2010. ISSN 1895-1767.

[OGF10b] Andrei Olaru, Cristian Gratie, and Adina Magda Florea. Emergent properties for data distribution in a cognitive MAS. *Computer Science and Information Systems*, 7(3):643–660, June 2010. ISSN 1820-0214.

[Ola10] Andrei Olaru. A context-aware multi-agent system for AmI environments. Technical report, University Politehnica of Bucharest, University Pierre et Marie Curie Paris, 2010. February.

[OSF10] Andrei Olaru, Amal El Fallah Seghrouchni, and Adina Magda Florea. Ambient intelligence: From scenario analysis towards a bottom-up design. In Mohammad Essaaidi, Michele Malgeri, and Costin Badica, editors, *Proceedings of IDC'2010, the 4th International Symposium on Intelligent Distributed Computing*, volume 315 of *Studies in Computational Intelligence*, pages 165–170. Springer, 2010.

[PRL09] M. Perttunen, J. Riekki, and O. Lassila. Context representation and reasoning in pervasive computing: a review. *International Journal of Multimedia and Ubiquitous Engineering*, 4(4):1–28, October 2009.

[RAS08] C. Ramos, J.C. Augusto, and D. Shapiro. Ambient intelligence - the next step for artificial intelligence. *IEEE Intelligent Systems*, 23(2):15–18, 2008.

[RVDA05] G. Riva, F. Vatalaro, F. Davide, and M. Alcañiz, editors. *Ambient Intelligence*. IOS Press Amsterdam, 2005.

[Sat01] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.

[Sat04] I. Satoh. Mobile agents for ambient intelligence. *Proceedings of MMAS*, pages 187–201, 2004.

[SBS+08] A.E.F. Seghrouchni, K. Breitman, N. Sabouret, M. Endler, Y. Charif, and J.P. Briot. Ambient intelligence applications: Introducing the campus framework. *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'2008)*, pages 165–174, 2008.

[SEFS04] A. Suna and A. El Fallah Seghrouchni. Programming mobile intelligent agents: An operational semantics. *Web Intelligence and Agent Systems*, 5(1):47–67, 2004.

[Seg08] Amal El Fallah Seghrouchni. Intelligence ambiante, les defis scientifiques. presentation, Colloque Intelligence Ambiante, Forum Atena, december 2008.

[SGK05] N.M. Sadeh, F.L. Gandon, and O.B. Kwon. Ambient intelligence: The MyCampus experience. Technical Report CMU-ISRI-05-123, School of Computer Science, Carnagie Mellon University, July 2005.

[SLP04] T. Strang and C. Linnhoff-Popien. A context modeling survey. *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp*, pages 1–8, 2004.

[SM06] N.I. Spanoudakis and P. Moraitis. 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, 2006.

[SONS10] Amal El Fallah Seghrouchni, Andrei Olaru, Thi Thuy Nga Nguyen, and Diego Salomone. 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*, 2010. accepted for publication.

[VRV05] M. Vallée, F. Ramparany, and L. Vercouter. 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, 2005.

[Wei93] M. Weiser. Some computer science issues in ubiquitous computing. *Communications - ACM*, pages 74–87, 1993.

[Wei95] M. Weiser. The computer for the 21st century. *Scientific American*, 272(3):78–89, 1995.