Laboratoire d'Informatique
de Paris 6

University "Politehnica" of
Bucharest

Université Pierre et Marie
Curie, Paris

A Context-Aware Multi-Agent System for AmI Environments

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

Progress Report

Rapport d'avancement

Andrei Olaru

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

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

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

June 2011

Contents

1 Introduction

1.1 Thesis Context

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.

1.2 Problem Setting

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 [Riva et al., 2005]. 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.

As a complex system, an AmI environment is organized on several layers [El Fallah Seghrouchni, 2008] (see also Figure 1):

- 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, as well as a wide range of protocols: WiFi, Bluetooth, Infrared, GSM, etc.
- the **interoperability** layer is vital to AmI, in order for devices to be able to communicate freely, using uniform protocols above this level.
- the **application** 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 [Riva et al., 2005]. 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**. An important part in solving these issues is held by the application layer. One of the prominent solutions that have been proposed for the implementation of the functionalities offered by the application layer comes from Artificial Intelligence [Ramos et al., 2008], and is represented by Multi-Agent Systems [Ferber, 1999].

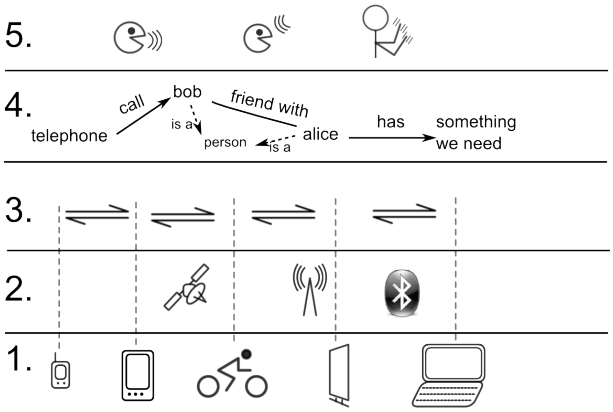

Figure 1: Layers of an AmI environment: Hardware (1.), Network / Connectivity (2.), Interoperability (3.), Intelligent services / applications (4.), Advanced interfaces (5.). (based on [El Fallah Seghrouchni, 2008])

The goal of this work is to answer the question "How to build a multi-agent system for the application layer of an Ambient Intelligence environment?" [Olaru, 2010]. Such a multi-agent system would have two roles: on the one hand, work with information at a semantic level, exchange and share information in order to deliver the potentially relevant information to the potentially interested agent / user; on the other hand, to offer to the user intelligent services resulting from information sharing. Its function remains general and independent of specific domains of applications. Above this layer there may be the interface, or there may be specialized applications (or specialized agents) that use the information for domain-specific tasks. Below this layer resides the non-intelligent, not semantic-oriented infrastructure for the transport of information and interoperability of protocols and formats.

Besides the use of agents, our approach relies on other principles: first, the use of *context-awareness* to guide the behavior and reasoning of agents; second, the *distribution of the system* and use of self-organization to achieve reliability and robustness.

In modeling the application layer of AmI by means of a multi-agent system, we focus on three aspects: **the topology of the system** – with whom (what other agents) do the agents interact and exchange information; **the behavior of agents** – how do agents exchange information; and the implementation of **context-aware reasoning** – how to solve problems and what information to exchange. Our approach is based on previous results and experiments regarding the three aspects enumerated above. We have previously developed a decentralized MAS for the context-aware exchange of information, a MAS where the topology of the system is related to agent context, and a representation of context information that allows the detection and resolution of problems.

1.3 Example Scenario

In order to illustrate the presented approach, let us use a part of a scenario that we have developed jointly with team Honiden-Lab from the NII Institute in Tokyo:

> Alice is one of the best students in the Computer Science course in her university. The university features a "smart", AmI-enabled campus, and is located in a city with several AmI-enabled systems, for instance the public transport system. Among many other functionalities that are possible thanks to Ambient Intelligence, it is possible for the Professor that holds the Computer Science course, when he arrives in the classroom, to know which students are already present and to see an estimation of the arrival time of the missing students. This way the Professor can decide when to start the lecture. Today, Alice is in a transit train and is supposed to arrive on time, but due to unexpected events, the train is going to be late. Automatically, the Professor will be notified that Alice will be at least fifteen minutes late, and therefore he will begin the course even if she hasn't arrived.

The scenario features important elements of our approach: the behavior of the system is context-aware, and the context is changing (dynamic context); the presented behavior 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. 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. See Section 4 (especially Figure 9) for more details on the different types of agents and relations.

The following section presents the state of the art in the implementations of AmI environments, as well as in the implementation of context-awareness. The proposed agent behavior, system topology and context representation are presented in Sections 3, 4 and 5, respectively. Each section is accompanied by its conclusions. The last section discusses perspectives and future work.

2 Agent-Based AmI Environments – State of the Art

2.1 AmI Implementations and Middleware

In the field of agent-based Ambient Intelligence platforms there are two main directions of development: one concerning agents oriented toward assisting the user, based on centralized repositories of knowledge (ontologies), and one concerning the coordination of agents associated with 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, by [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

EasyMeeting, by [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 with 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.

Other projects that use similar approaches are ASK-IT and DALICA, presented in the works of Spanoudakis [Spanoudakis and Moraitis, 2006] and Constantini [Costantini et al., 2008], respectively.

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]) employs mobile agents to offer functionality on the user's devices. Whenever a device (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. 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 capabilities to the user.

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. However, here the middleware itself is not agent-oriented and is not distributed.

The AmbieAgents infrastructure ([Lech and Wienhofen, 2005]) is proposed as a scalable solution for mobile, context-ware information services. Context Agents manage 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.

The CAMPUS framework ([El Fallah Seghrouchni et al., 2008]) considers issues like different types of contexts 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.

It easy to observe that different agent systems 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 systems 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 only on 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 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 and Context Representation

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. 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 [Dey et al., 1999], certain infrastructures for the processing of context information have been proposed [Hong and Landay, 2001, Harter et al., 2002, Lech and Wienhofen, 2005, Henricksen and Indulska, 2006]. There are several layers that are usually proposed [Baldauf et al., 2007, Feng et al., 2004], 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 [Baldauf et al., 2007]. 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 [Chen and Kotz, 2000]. Moreover, these infrastructures are usually centralized, using context servers that are queried to obtain relevant or useful context information [Dey et al., 1999, Lech and Wienhofen, 2005]. In our approach [Olaru et al., 2010a], 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 [Perttunen et al., 2009, Strang and Linnhoff-Popien, 2004]. 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 [Henricksen and Indulska, 2006, Bettini et al., 2010]. 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 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; second, a graph-based representation of context information that, by means of graph matching, allows identifying the relevant information and also the detection and solution of problems in the user's context.

3 Context-Aware Agent Behavior for AmI

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. The question that we are trying to answer 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 application layer of an AmI environment.

Answering this question considering that, at realistic scale, an AmI system will have to deal with a very large number of users and devices, poses several challenges: how to keep the system distributed and decentralized, while keeping agents small enough to fit on resource-constrained devices?

The implementation [Olaru et al., 2010c, Olaru et al., 2010b, Olaru and Gratie, 2010] is based on two principles: the use of light cognitive agents and the implementation of mechanisms of self-organization. That is, 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 behavior and communication.

In this first phase, the agent topology was simple: the agents were placed in a rectangular grid, and an agent could only communicate with its 8 neighbors. To provide context-awareness for information sharing, we proposed four simple and generic aspects of context-awareness:
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). Also, if all other measures are 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: the agent will treat the information with higher priority and the agent will send it to more neighbors. 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 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: $Dist \in [0, 1]$, 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

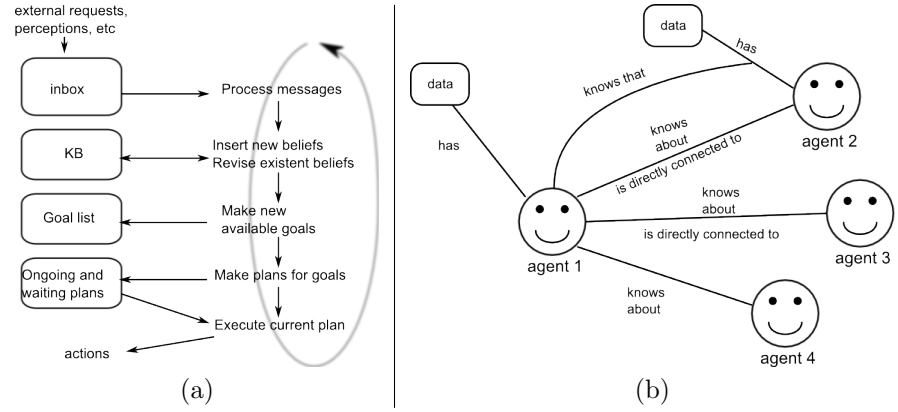

Figure 2: (a) The basic execution cycle of an agent. (b) Example of content of an agent's (the one in the center) knowledge base. Several relationships are displayed, like *knows about*, *is connected to*, *has*, *knows*

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

3.1 Agent Design

Agents 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 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 applications where different types of devices are involved, agents may have knowledge bases of different sizes.

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 the prototype, 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 explicitly 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. 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 neighbour agents (see also Figure 2 (b)).

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.

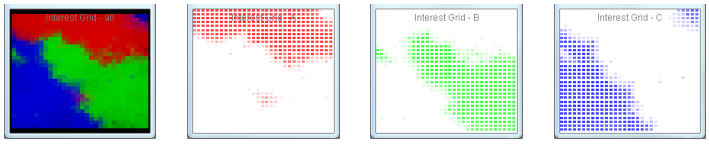

Figure 3: The agent interests at step 130: (a) global, (b) for domain A, (c) for domain B, (d) for domain C.

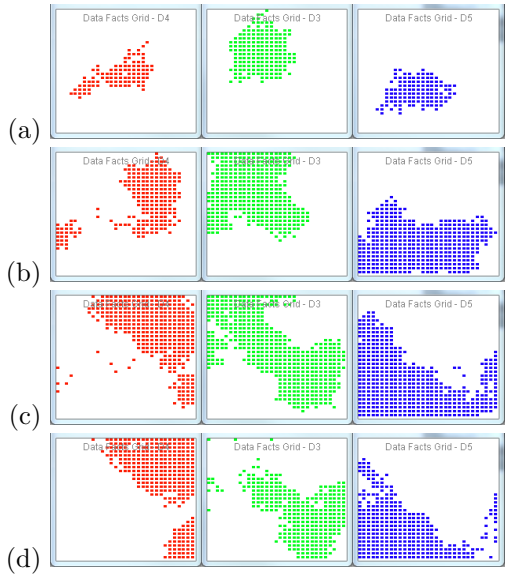

Figure 4: Agents receive new data at simulation step 152 (a) and data starts spreading. Distribution of facts at steps 170 (b); 210 (c); 271 (d). The data is relevant to domains A, B and C (from left to right).

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 (some free capacity must always be available in order to be able to receive information). Each goal is assigned an importance: for *Inform* goals it is the relevance of the information to be sent; for *Free* goals it is computed as a function of how full the agent's storage capacity is, reaching a maximum of 1 when the knowledge base consumes all available capacity. The agent makes a plan for the goal with the highest importance: for *Free* goals, the agent decides what fact(s) 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, computed 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.

3.2 Results

The goal of this application was to have many cognitive, but limited, agents that, as a system, would retain a number of pieces of data. Agents were able to form interest towards certain domains and then they would be more interested in information about those particular domains.

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. This implementation was chosen because the purpose was to study the behavior of the system, in conditions that will allow maximum performance and that let us focus on the properties of the system rather than on other implementation issues. All experiments involved 1000 agents.

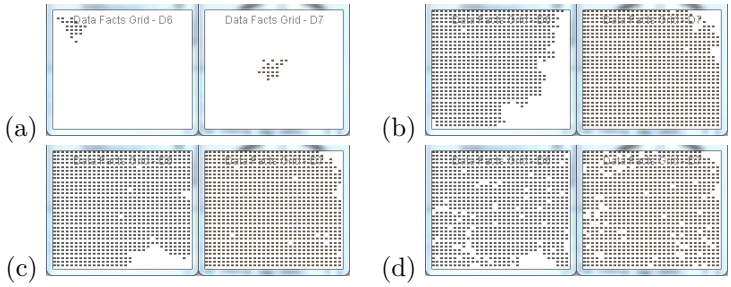

Figure 5: Data with high pressure spreading through the system at simulation step: (a) 211; (b) 300; (c) 350; (d) 447.

The test scenario was the following: first, insert into random positions in the system information of three different specialties, with low pressure and persistence. This would form the agents' specialties. Then, insert into the system three "test" pieces of information, with higher persistence, and see how they spread through the system. Last, insert two pieces of information with very high pressure, and with no particular specialty. The goal is to be able to control the spread of the information by means of the measures of context.

The results of the experiments show that indeed we can achieve control over the distribution of data: notice that the spreading in Figure 4 follows the agents specialties, visible in Figure 3. Data with high pressure spreads quickly and covers the whole system, as one can see in Figure 5. It is important to note that the existing data is not significantly influenced by the high pressure data – they can coexist in the system.

Similar results have also been obtained for agents placed in random positions, communicating with neighbors in a certain range, therefore these results re not bound to the rectangular grid topology.

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.

3.3 Lessons Learned

The experiments that are described in this section taught us how to build and tune the behavior of individual agents so that, even with little knowledge, a large number of agents would have a coherent, emergent, global behavior. The core of the agent's cycle is focused on information exchange, based on the relevance of the information, which is computed considering some simple and generic measures of context.

In the scenario presented in Section 1.3, this behavior would be 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. In order to obtain this, the agents need a better topology (see Section 4) and a semantic-aware manner of calculating the relevance of information (see Section 5), but the basic behavior of agents would remain the one presented in this section.

4 Mapping Context Structure and Agent Hierarchy

This section presents a second aspect in the implementation of Ambient Intelligence. If the previous project 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 agent hierarchy and context-awareness.

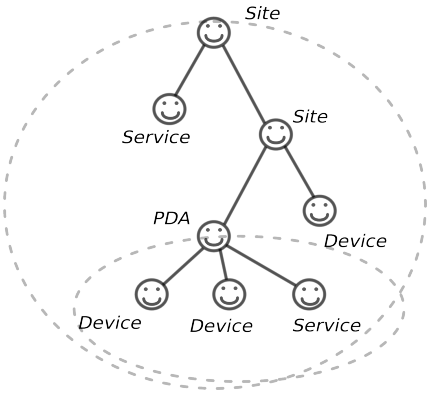

Figure 6: Example of an abstract logical hierarchy for the Ao Dai project: The root *Site* offers a *Service* and also contains another *Site* that contains a *Device*. The user is inside the second site, and its *PDA* can offer him the capabilities of two *Devices* and a *Service* (that may be a mobile agent executing on the PDA.)

The challenge in developing a MAS topology that remains decentralized and scalable is how to make the connections between agents related to their context, and how to keep them context-aware when the context changes.

The solution is based on two aspects: the use of the agent-oriented, ambient-calculus-inspired programming language CLAIM; and the definition of context-related relations and types of agents.

4.1 CLAIM and Sympa

As an agent-oriented programming language, CLAIM [Suna and El Fallah Seghrouchni, 2004] 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.

In CLAIM, 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).

Being inspired from ambient calculus [Cardelli and Gordon, 2000], an important part of CLAIM is that agents are mobile, they are part of an agent *hierarchy*, and when an agent moves, it sub-hierarchy moves with it. This idea is central to our approach. 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 span across multiple machines.

A very simple example of how this may be useful can be seen in the agent hierarchy in Figure 6: there may exist a structure of physical *Sites*, each of which may contain devices or services. The subtree of a *Site* represents entities in that physical context. However, the subtree of a *PDA* agent represents its computational context: the user is able to use, by means of the PDA, several services and devices. When the PDA moves (i.e. changes parent in the hierarchy, due to physical movement of its user), the sub-agents (the ones that are allowed to) will move with it, and so the agent hierarchy remains coherent with the structure of context.

4.2 Types of Agents and Relations

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.

We can extend the example in Section 4.1 by supporting more types of context [Chen and Kotz, 2000,

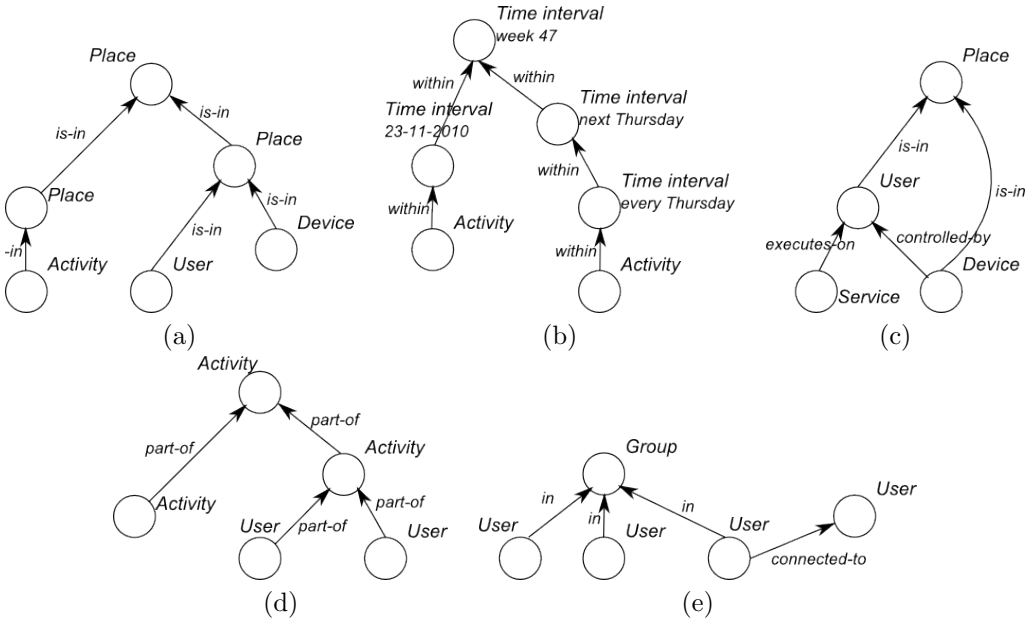

Figure 7: Possible relations between agents, relative to the different types of context: *is-in* links agents of different types to agents of type *Place*; *within* links agents to *Time intervals*; *executes-on* and *controlled-by* are specific to *Service* and *Device* agents; *part-of* links agents to *Activity* agents; and *in* and *connected-to* are relations between *User* agents and/or *Group* agents.

Henricksen and Indulska, 2006]: space, time, activity, social and computational. Each of these types of context is considerably hierarchical, but the hierarchies of different types of context may intersect. This is why we must adapt CLAIM to support more complex connections between agents. In order to perform the mapping of system structure to context, we introduce several types of agents and relations.

- for spatial context – places – the *Place* agent and the *is-in* relation. This relation can exist between subordinate *Place* agents, or between *Place* agents and other types of agents – like *User*s, *Device*s, *Service*s or *Activities*. For instance, when a *Service* agent is a child of a *Place* agent, it means that it is a context-aware service that is offered within that place (and is only relevant in that place). *Place* agents execute on machines that are connected to the network access points in those respective spaces.
- for computational context – devices and services – the *Device* and *Service* agents and several relations: *of* and *executes-on* for services, *controlled-by* and *is-in* for devices. When a device is not in use, it will only have one parent, a *Place* or another *Device*, and the relation will be *is-in*. However, when used, it will become a logical child of the *User* agent, by means of the relation *controlled-by*, while keeping the link with its location (in the case of devices that do not move – like presentation screens). For services, *of* defines the entity that the service belongs to, and *executes-on* links the service to the devices where it is executed. When the device it executes on moves out of the context the service belongs to, the service will return to its original parent.
- for temporal context – time and time intervals – the *Time interval* agent and the *within* relation. The *within* relation designates time intervals contained by larger intervals, or *Activities* that take place in a certain interval of time. A time interval (that is relevant in relation with some activity) may span across multiple higher-level intervals. For this type of complex relations, the time interval will have as a "main" parent the interval which contains the present moment.
- for activity context, the *Activity* agent and the *part-of* relation. The *Activity* agents execute on a machine that is related to the organization of the activity, or to the user that coordinates the activity – for activities where multiple users take part. For the user's personal activities, the *Activity* agent is a child of the *User agent*, linked by a *of* relation.
- finally, for social context, the *User* agent is used to represent users of the system. The *User* agent executes on the user's PDA, or on any device that the user is currently using. For representing relationships between users of the system, two relations are used: the *in* relation shows that the user is part of a larger group of users – managed by a *Group* agent. The *connected-to* relation shows that two users are connected, without them being part of a common group, taking part in a

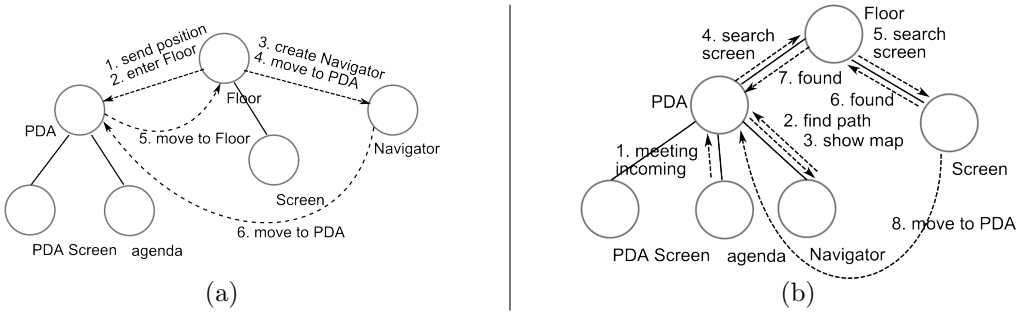

Figure 8: Sequences of messages exchanged between agents: (a) *Floor* announces *PDA* of its new position, and instructs it to move as its child, then creates a *Navigator* that will offer services to *PDA*; (b) *Agenda* announces a new meeting, *PDA* asks a path from *Navigator*, which in turn requires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

common activity or being in the same place. While the *in* relation is hierarchical, the *connected-to* relation is not.

The different types of context and associated agents and relations are presented in Figure 7.

4.3 The Ao Dai Project: Results

The Ao Dai project has demonstrated the validity of our approach to mapping context to agent structure. In the project, only a subset of the agent types was used, and the implementation has been done in classical CLAIM (no multiple parents) [El Fallah Seghrouchni et al., 2010].

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 successfully demonstrated in a semi-simulated environment, running on two machines, during the 5th NII-LIP6 Workshop, held in June 2010 in Paris, France.

In this project, we have studied several scenarios including the following: a user has a meeting in a building that he / she does not previously know. When arriving at the right floor of the building, the user starts being assisted navigating towards the right office, and the system also helps the user a room with a presentation screen of appropriate size. Besides location, we also consider as part of the user's context, the available computing resources around the user and the user's preferences.

In the implementation of the scenario, there are three major types of agents:

- 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 at that 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's meeting.
- The third type, *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 the user and has the CLAIM capability of managing requests for services or devices. It also stores the user's preferences.

When arriving at the floor, the user's PDA automatically connects to a local wireless access point. A CLAIM agent executes on the user's PDA. The *Floor* agent (of type *Site*) 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.

Next, *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

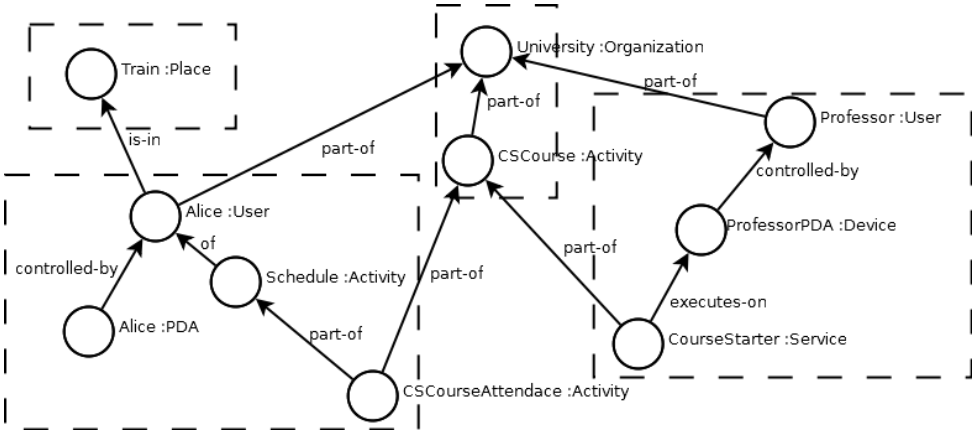

Figure 9: Agent topology for the scenario from Section 1.3. Dashed boxes group agents running on the same machine.

sensors (sent to *Navigator* by *Floor*, and through *PDA*) into positions on the graphical map, and can compute paths between the offices on the floor. *Navigator* is an agent that offers to the user services that are available and only make sense in the context of the floor (see also Figure 8 (a)).

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 range in which to search (the current office, a nearby office, the whole floor) and the minimal size of the searched screen.

The context-awareness in Ao Dai is done by exploiting the particular hierarchical agent structure that is offered by the CLAIM language. 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*.

Agents interact only with their parent and their children. Take for example the search for devices (see Figure 8 (b)). 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 is received, agent *Floor* searches itself to see if it has the requested capability and it satisfies the criteria, then (in case of failure) searches in all of its children (except the agent who invoked the search), then it searches its parent (if any). 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.

4.4 Lessons Learned

The Ao Dai project showed that hierarchies of mobile agents, implemented in CLAIM, can be used to map context structure, therefore obtaining inherent context-awareness, distribution, and communication only inside the current context.

Based on the success of Ao Dai, relations and types of agents have been developed for a more complete mapping of context. Returning to our example in Section 1.3, We can design the topology of the agents as in Figure 9. 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

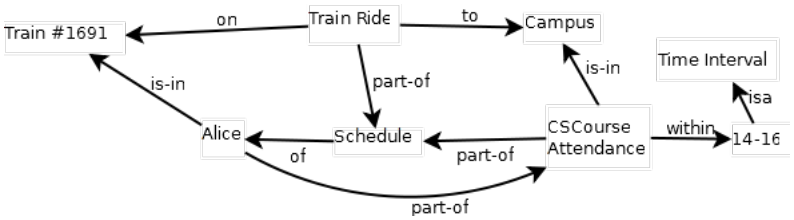

Figure 10: The context graph of the agent assisting Alice, showing information about Alice's activity.

Figure 11: Context pattern matching the time interval in which Alice will attend the CS course.

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 3, there may be other agents receiving this information as well, but they will discard it as the information is not relevant to them. In the next section we will see how relevance may be computed in a more advanced manner than in Section 3.

5 Graphs and Patterns for Context-Awareness

The last aspect of our approach is the representation of context inside the agents, and the way in which agents are able to compute the relevance of the information they receive.

The goal of the work presented in this section is to design a formalism that can be used by agents to represent information and the context of information, to compute the relevance of information, and to help the user in solving problems, in a generic manner.

The challenges of finding a good representation of context information is that we want it to be open and flexible – be able to include new values and structures without having to redefine ontologies, and also be able to aggregate easily new information with the existing knowledge of the agent.

The representation for context information that we have chosen is based on graphs. It is similar to RDF graphs and to concept maps, in that the edges of the graph are predicates. Each agent has a graph $G = (V, E)$ that contains the information that is relevant to its function. For instance, the agent that assists Alice from the scenario in Section 1.3 would have a graph that contains parts of Alice's schedule (for the sake of simplicity we will omit the *Scheduler* agent in this section), like in Figure 10. Note that although some edges and nodes in the graphs may have the same labels and semantics as elements from the agent hierarchy in Section 4, the two graphs are very different: these are data structures stored inside the agent, and those in Section 4 represent the topology of the agent system, are not represented explicitly, and are exterior to the agents.

Our contribution is represented, however, not by the representation itself, but by the introduction of context patterns [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. A pattern s is defined as:
$G_s^P = (V_s^P, E_s^P)$
$V_s^P = \{v_i\}, v_i = string \mid ? \mid URI, i = \overline{1,n}$

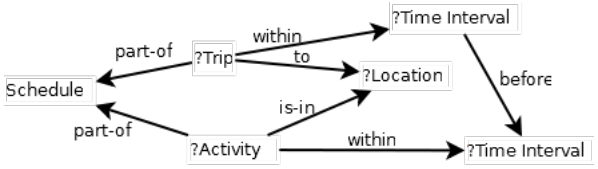

Figure 12: Context pattern expressing that before an activity at a location there must be a trip to that location. Note that subgraphs of the type ? $\xrightarrow{isa}$ *Concept* have been replaced, for simplicity, with nodes labeled ?*Concept* .

$E_s^P = \{e_k\}, e_k = (v_i, v_j, E\_RegExp), v_i, v_j \in V_s^P, k = \overline{1,m}$, where E_RegExp is a regular expression formed of strings or URIs.

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 11. 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.

5.1 Context Matching

An agent has a set of patterns that it matches against the current context (graph G), and against information that it receives from other agents. A pattern G_s^P (we will mark with " ^{P} " graphs and elements that contain special features like ? nodes) *matches* a subgraph G' of G, with $G' = (V', E')$ and $G_s^P = (V_s^P, E_s^P)$, iff there exists an injective function $f : V_s^P \to V'$, so that
(1) $\forall v_i^P \in V_s^P, v_i^P =?$ or $v_i^P = f(v_i^P)$ (*same value*)
and
(2) $\forall e^P \in E_s^P, e^P = (v_i^P, v_j^P, value)$ we have:
-if *value* is a string or an URI, then the edge $(f(v_i^P), f(v_j^P), value) \in E'$
-if *value* is a regular expression, then it matches the values $value_0$, $value_1$,... , $value_p$ of a series of edges $e_0, e_1, ..., e_p \in E'$, where $e_0 = (f(v_i^P), v_{a_0}, value_0)$, $e_k = (v_{a_{k-1}}, v_{a_k}, value_1) k = \overline{1,p-1}$, $e_p = (v_{a_{p-1}}, f(v_j^P), value_p)$, $v_{a_l} \in V'$.

That is, every non-? vertex from the pattern must match a different vertex from G'; every non-RegExp edge from the pattern must match an edge from G'; and every RegExp edge from the pattern must match a series of edges from G'. Subgraph G' should be minimal.

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.

Example. We can presume that agent *Alice* knows that agent *CSCourseAttendance* is interested in the pattern $CS\ Course\ Attendance(\xrightarrow{within}? \xrightarrow{isa} Time\ Interval) \xrightarrow{part-of} Schedule \xrightarrow{of} Alice$.[1] 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.

[1] From here on we will 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).

5.2 Problem Solving

A *Problem* is a graph G^P that is a partial instantiation of a pattern G_s^P, according to the current context. It is the union between (1) the subgraph G' of G that *k-matches* pattern G_s^P (with $k \geq 1$) and (2) the part of G_s^P that is not matched by G' (the *unsolved* part of the problem). The problem remains associated with the pattern: for the pattern $G_s^P = (V_s^P, E_s^P)$ and the k-matching subgraph $G' = (V', E')$, the problem p is a tuple (G_s^P, G_p^P), where G_p^P is the problem's graph:
$G_p^P = G' \cup G_x^P$
$G_x^P = (V_x^P, E_x^P)$ *(the unsolved part)*, where
$V_x^P = \{v \in V_s^P, v \notin dom(f)\}$
$E_x^P = \{e \in E_s^P for\ which\ condition\ (2)\ is\ not\ fulfilled\}$
G_x^P is a subgraph of G_s^P.

To continue with the same scenario as above, let us suppose that agent *Alice* has a pattern that describes the situation in which Alice must attend an activity which is at a certain location, and she must get to that location in some way – the pattern in Figure 12. This pattern *2-matches* Alice's context graph: the missing edges are the $\xrightarrow{within}$ edge for the trip and the $\xrightarrow{before}$ edge. Since the train is late, it sends to its passengers information on when the train will arrive in different stations. Considering there is a station called *Campus*, Alice will receive, among other information that is irrelevant, the graph $Trip(\xrightarrow{to} Campus)(\xrightarrow{on} Train\ \#1691) \xrightarrow{within}$ 15:30-16:15. This new information partially matches the pattern, but more importantly, it matches the unsolved part of Alice's problem that is based on this pattern, therefore it is integrated in Alice's context information. Other patterns may then be used to modify the interval for the course attendance, or a specialized application may do it, based on the $\xrightarrow{before}$ relation between the two intervals. The new interval for the course attendance is then sent to *CSCourseAttendance* (since it matches the pattern that it is interested in) and from there the information will reach *CSCourse* and *CSCourseStarter* (see Section 4.4).

5.3 Lessons Learned

We have already used the representation presented in this section for the modeling of several scenarios of Ambient Intelligence. The power of context patterns lies in the fact that, for smaller devices, agents may hold fewer and smaller patterns, and the matching of smaller patterns will also require less computational power. This makes the use of context patterns and context matching flexible.

Context patterns allow an agent to easily detect the pieces of information that are relevant to it, and also, by knowing about patterns of other agents, to know which pieces of information may be interesting for other agents. This way, by applying the agent behaviour described in Section 3 and using the context-based agent topology described in Section 4, a multi agent system may serve as an application layer for Ambient Intelligence.

6 Perspectives and Future Work – A New Platform

The work that has been done so far studied three aspects of using multi-agent systems for the implementation of Ambient intelligence, all related to context awareness: first, how to control the spreading of information in a decentralized agent system, using simple and generic context measures; then, how to map the structure of the agent system to the structure of context, so that agents will interact and move within their context, helping communication of relevant information and decentralization of the system; finally, how to use context patterns in order to detect relevant information.

The next step is to integrate the three aspects and the associated implementations into one single application that can serve as an application layer for AmI environments, with the following features:

- the topology of agents will be context-aware, using the agent types and relations presented in Section 4;
- the behavior of agents would be centered on information sharing and exchange, as in Section 3;

relevance would be computed not by using abstract domains of interest, but matching between the context of the information and the agent's context patterns;
- the agents would hold information represented as graphs, as well as sets of context patterns that are relevant to the agent's function (see Section 5), in their knowledge bases, and use context matching to detect relevant information.

We have already started implementing the platform, using the Jade platform as an underlying basis for agent management and mobility and using the CLAIM agent-oriented language for the management of agent behavior and mobility. The agent's knowledge will be represented by context graphs and patterns, and their behavior will be using a graph matching algorithm to match context.

References

[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.

[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.

[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.

[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.

[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.

[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.

[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., 2010] 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.*

[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.

[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.

[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.

[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.

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

[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.

[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.

[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.

[Olaru et al., 2011] 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.

[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.

[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 - Scientific International Journal for Parallel and Distributed Computing*, 11(1):33–42. ISSN 1895-1767.

[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.

[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.

[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.

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

[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.

[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.

[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.