AN AGENT-ORIENTED APPROACH FOR AMBIENT INTELLIGENCE

Andrei Olaru[1], Adina Magda Florea[2], Amal El Fallah Seghrouchni[3]

Majoritatea implementărilor de sisteme pentru Inteligenţa Ambientală realizate în cursul ultimului deceniu s-au concentrat asupra implementării de sisteme complete pentru scopuri specifice. În activitatea noastra ne concentrăm pe acel nivel dintr-un sistem de Inteligenţă Ambientală care se ocupă cu transferul de informaţie ţinând cont de semantică, în aşa fel încât informaţia relevantă să fie livrată utilizatorului interesat. Abordarea noastră foloseşte un sistem multi-agent caracterizat de comportament si cunoştinţe locale pentru agenţi, o topologie a sistemului în relaţie cu contextul, şi şabloane de context pentru detectarea situaţiei utilizatorului.

Most implementations of Ambient Intelligence – or AmI – systems realized over the last decade have focused on implementing complete systems that serve specific purposes. In our work we focus on that layer of an AmI system that handles the semantic-aware exchange of information in order to deliver the relevant information to the interested user. Our approach uses a multi-agent system that features local behavior and knowledge for agents, a context-related system topology, and context patterns for the detection of the situation of the user.

Keywords: Ambient Intelligence, Multi-Agent Systems, Context-awareness

1. Introduction

The term of Ambient Intelligence has been coined at the dawn of the 21st century, in 2001, with the report of the ISTAG group [1], 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.

[1]Assistant Professor, PhD, Computer Science and Engineering Department, University Politehnica of Bucharest, Romania, e-mail: cs@andreiolaru.ro

[2]Professor, Computer Science and Engineering Department, University Politehnica of Bucharest, Romania, e-mail: adina@cs.pub.ro

[3]Professor, Laboratory of Informatics of Paris 6, University Pierre et Marie Curie, France, e-mail: amal.elfallah@lip6.fr

Ambient intelligence should represent the third wave in computing [2]. After the mainframe and the personal computer, in the age of Ambient Intelligence the computing 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 [3]: the **hardware** layer is composed of all the devices that are part of the AmI environment – sensors, actuators, smartphones, displays, laptops, etc, – and is very heterogeneous from the point of view of computational and storage capacity; the **interconnectivity** layer allows connections between the devices in the hardware layer, and may use wired or wireless networks, 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; the last layer is the **interface** – gestures, speech, voice recognition and 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 [2]. 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 [4], and is represented by Multi-Agent Systems [5].

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?". 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. The information provided by this layer may be provided directly to the interface, or to specialized applications (or specialized agent components) that use the information for domain-specific tasks.

The focus of our work is on three aspects of the modeling and implementation of a such agent system: the behavior of the agent ("how does the agent exchange the information?"), the topology of the system ("with whom does the agent exchange the information?") and the representation of context ("how does the agent determine if the information is relevant?"). This paper will focus more on the last two aspects: topology (context-awareness outside the agent) and knowledge representation (context-awareness inside the agent).

In order to illustrate the presented approach, let us use a part of a scenario that we have developed jointly with a team from NII (Tokio):

Scenario: 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.

The following section presents the state of the art in the implementations of AmI environments, as well as in the implementation of context-awareness. Section 3 briefly presents the three aspects of our approach to building a multi-agent system for Ambient Intelligence. These aspects are then discussed in detail in Sections 4 (the topology of the system), 5 (context patterns) and 6 (agent behavior). The last section draws the conclusions.

2. **Related Work**

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.

There are agent-based systems for Ambient Intelligence that do not explicitly use context-awareness, and also some that do not use agents as a

distributed computing paradigm [6, 7, 8, 9]. There is however research that concerns larger numbers of agents, distributed control, and fault tolerance:

Context handling is considered by the AmbieAgents infrastructure [10], which is proposed as a scalable solution for mobile, context-aware information services. There are three types of agents: 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. Although it may prove effective in pre-programmed scenarios, the system is not very flexible. In this paper we are trying to provide a simpler agent structure, in a system that is based more on self-organization and less on controlled interaction between agents. Context-aware data management is also discussed by Feng et al [11], but context queries are handled in a centralized way, making it efficient but not very scalable.

The LAICA project [12] brings good arguments for relying on agents in the implementation of AmI. It considers various types of agents, some that may be very simple, but still act in an agent-like fashion. The authors, also having experience in the field of self-organization, state a very important idea: there is no need for the individual components to be "intelligent", but it is the whole environment that, by means of coordination, collaboration and organization, must be perceived by the user as intelligent. The work is very interesting as it brings into discussion important issues like scalability, throughput, delegation of tasks and a middleware that only facilitates interaction, in order to enable subsequent peer-to-peer contact. The application is directed towards generic processing of data, which is done many times in a fairly centralized manner. The structure and behavior of agents is not well explained, as their role in the system is quite reduced – the middleware itself is not an agent. However, the architecture of the system remains very interesting.

The SpacialAgents platform [13] 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 the 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.

Agents with reduced memory and performance footprint for AmI have been developed in the Agent Factory Micro Edition project [14]. The authors

succeed in implementing a reliable communication infrastructure by using reasonably simple agents, however there is no higher level view that includes more complex global behavior and there is no context-awareness.

The implementation of the SodaPop model [15] is another application sharing common features with our own, especially the use of self-organization for an AmI system, but it does not use the agent paradigm and it handles a quite specific case.

Ever since the first works on context-awareness for pervasive computing [16], certain infrastructures for the processing of context information have been proposed [17, 18, 10, 19]. There are several layers that are usually proposed [20, 11], 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 [20]. 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 [21]. Moreover, these infrastructures are usually centralized, using context servers that are queried to obtain relevant or useful context information [16, 10]. In our approach [22], 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 [23, 24]. 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 [19, 25]. 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 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. Elements of the Approach

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

Relying on local information exchange has proved effective in previous studies by the authors [26], in which pieces of information successfully spread in a multi-agent system where the agents had a preference only for certain types of information. The information did spread quickly and correctly, even though each agent had only a very small knowledge base that referred only to its own information and to some of the neighbors' information.

However, our first studies used a space-based topology – two agents were neighbors (i.e. could communicate directly) only if they were within a certain distance of each other – and a simple indication of context – information was characterized by some domains of interest, its persistence and a measure of importance.

Table 1

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

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

In this paper we extend our approach with an improved topology – that considers shared context of more types – and improved computing of relevance – the relevance of the information is computed using a graph-based representation for information and also graph patterns that represent the interest of the agent. We present the elements of our approach in Table 1.

4. Agent System Topology

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

The system that we are designing relies on the context-aware exchange of information between an agent and its neighbors. But an agent should only exchange information with another agent that may consider that information as relevant. And that can happen only if the agents share some type of context. For instance, if the agents are part of the same activity, or if their users are

located in the same space. Two agents that share no context would not have any information that is relevant to both of them.

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

In order to better map context to system topology, we have defined the following types of neighborhood relationships, to match the different types of context – spatial, computational, temporal, activity and social [21]: for spatial context – the *Place* agent and the *is-in* relation; for computational context – the *Device* and *Service* agents and the *of* (linking to the agent to which the offering of the service is related, like a place) and *executes-on* relations for services and *controlled-by* (linking the device to the user that controls it) for devices; for temporal context – the *Time interval* agent and the *within* relation; for activity context – the *Activity* agent and the *part-of* relation; for social context – the *User* and *Group* agents and the *in* (linking a user to a group) and *connected-to* (linking two users) relations.

A more organized perspective on the relations between agents is presented in Table 2.

Table 2

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

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

There are several advantages to using a context-based topology: on the one hand, information only spreads among agents that share common context, i.e. a piece of information is sent to an agent to which it is more likely to be relevant; on the other hand, when an agent searches for a piece of information,

the search will be confined to the context of the agent, i.e. where it is most likely to yield relevant results. Having the types of agents and relations predefined (but nevertheless still having a high degree of generality) allows for the implementation of some specific behavior depending on the type of the agent.

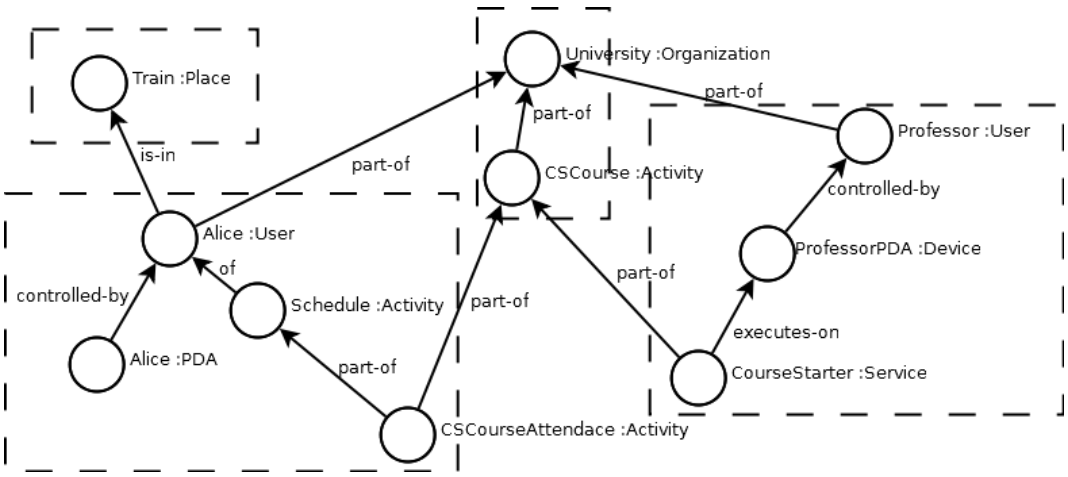

Fig. 1. Agent topology for the scenario from Section 1. Dashed boxes group agents running on the same machine.

Example. Returning to our example in Section 1, we can design the topology of the agent system as in Figure 1. Having this topology, when the information that Alice will be late is generated by her *Scheduler* (having used information sent to *Alice* by *Train*), it will inform the agent managing her CS Course attendance, which will disseminate this information to *CSCourse*, which will inform the *CourseStarter*. This way, the information spreads among agents having a common context, here the common context being mainly related to activity.

The use of a context-based topology has been validated through a proof of concept application using a subset of the presented relations and agent types [27], that has been demonstrated at the Fifth Joint NII-LIP6 Workshop on Multi-Agent and Distributed Systems in 2010[1].

5. Context Patterns

In the context of an Ambient Intelligence system that should be completely distributed and formed of devices with heterogeneous computing capabilities, one challenge is how should agents evaluate what is the information relevant to them and, potentially, to their neighbors? That is, how to know if the context of the information is compatible with the context of the agent?

The challenge 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

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

structures without having to redefine ontologies, and also be able to aggregate easily new information with the existing knowledge of the agent, all this without having centralized components for context processing.

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 our example scenario) 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 2.

Note that although some edges and nodes in the graphs may have the same labels and semantics as elements from the agent topology 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.

Also, as opposed to the relations in the agent system topology (see Section 4), the relations between concepts in the agent's context are not predefined, and can have any value – including values that coincide with the relations between agents in the system's topology.

Our contribution is represented, however, not by the representation itself, but by the introduction of context patterns [28]. 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:

$G_s^P = (V_s^P, E_s^P)$

$V_s^P = \{v_i\}, v_i = string \mid ? \mid URI, i = \overline{1,n}$

$E_s^P = \{e_k\}, e_k = (v_i, v_j, E\_RegExp), v_i, v_j \in V_s^P, k = \overline{1,m}$, 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 3. 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.

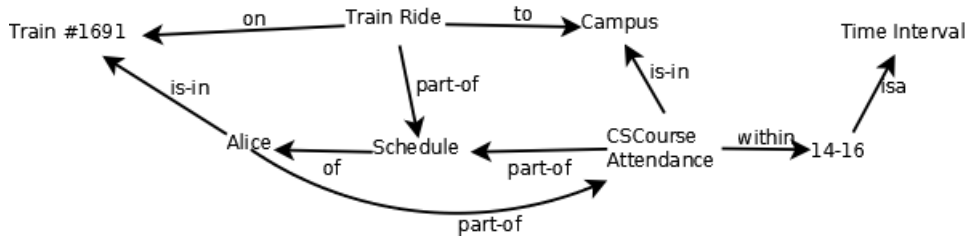

Fig. 2. The context graph of the agent assisting Alice, showing information about Alice's activity.

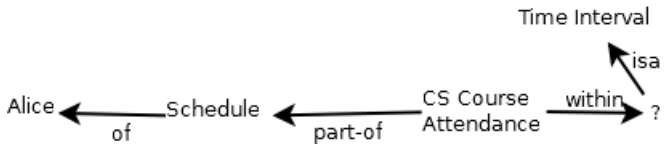

Fig. 3. Context pattern matching the time interval in which Alice will attend the CS course.

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 $value_0, \ldots$, $value_p$ of a series of edges $e_0, e_1, \ldots, e_p \in E'$, 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.

Example. We can presume that agent *Alice* knows that agent *CS Course Attendance* is interested in the pattern[2] . 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

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

Alice will send to *CSCourseAttendance* this solution, thus informing it of Alice's participation to the course.

Table 3

Pseudo-code for the behavior of an agent.

Algorithm for agent A	
$\cdot\ receivePatternFromNeighbor(G^P_{sB}, B)$	
$addToNeighborPatterns(B, G^P_{sB})$	// store the association
$\cdot\ receiveInformationFromNeighbor(G_{iB}, B)$	
$(G', k) = match(G_{iB}, G_A)$	// G_A: agent's context graph
$if(k > 0\ and\ k < integration\_threshold)$	
$merge(G_{iB}, G_A)$	
$else$	
discard	// either already known ($k = 0$) or not relevant enough
$\cdot\ doPatternMatching$	
$foreach(\ (Ag, G^P_s)$	// pair agent-pattern
$\in AgentPatterns \cup NeighborPatterns\ )$	// own or neighbors'
$(G', k) = match(G^P_s, G_A)$	// matching subgraph, k
$if(k = 0\ and\ Ag \neq A)$	// full match of neighbor pattern
$send(Ag, G')$	// send the information to the agent
$if(k = 0\ and\ Ag = A\ and\ isProblem(G^P_s))$	// an existing problem was solved
$removeProblem(G^P_s)$	
$informUserProblemSolved()$	
$if(k > 0\ and\ Ag = A$	
$and\ k < problem\_threshold)$	// almost matching
$addProblem(G^P_s)$	
$disseminatePattern(G^P_s)$	// wait for neighbors to provide solution

A pattern G^P_s *k-matches* (matches except for k edges) a subgraph G' of G, if condition (2) above is fulfilled for $m - k$ edges in E^P_s, $k \in [1, m-1]$, $m = ||E^P_s||$ and G' remains connected and minimal.

A k-matching pattern with k above a certain threshold may indicate a problem in the situation of the user: the pattern matches, therefore the user is in the specified situation, but some elements are missing, therefore it may mean that the agent should try and retrieve those elements.

6. Agent Behavior

The purpose of agents is to obtain information that is relevant to them – and therefore to the users. An agent A obtains information by *expressing interest* in certain types of information, and then waiting for information matching that interest to be shared with it. Expressing interest is done by sending context patterns to the neighbor agents. Then, neighbors will send information that matches the patterns to agent A.

We can identify certain components of the behavior of an agent: receiving a pattern from another agent, expressing interest; receiving an information from another agent, as a consequence of expressing interest; there is a pattern-match between a context pattern and the agent's context graph. These three components are presented in the algorithm in Table 3.

Example. Let us go back to our scenario: we know that the *CourseStarter* service (which is an agent) collects information on the details of the students' attendance to the course. This interest will be represented by a pattern of the form[3] $?Activity(\xrightarrow{within}?Time\ Interval) \xrightarrow{part\text{-}of^*\ of}?User \xrightarrow{part\text{-}of} CSCourse$. According to the agent behavior, it will send this pattern to the neighbors that are potentially interested by it – in this case, *CSCourse*. CSCourse knows the users that participate in the course, so it will be able to give multiple solutions to the sub-pattern $?User \xrightarrow{part\text{-}of} CSCourse$. Having instantiated the solutions, it will send one of the partially-solved, i.e. $?Activity(\xrightarrow{within}?Time\ Interval) \xrightarrow{part\text{-}of^*\ of} Alice$ to *CSCourseAttendance*, which has previously given information related to user Alice. According to the previous example (in Section 5), *CSCourseAttendance* already holds the appropriate information and returns the solution to *CSCourse*, which in turn will send it to *CourseStarter* (knowing the correspondence between the agents and the patterns they are interested in).

7. Conclusions and Future Work

Using agent for the implementation of a generic application layer for AmI, based on information exchange, has very good potential. In this context, we have developed an approach to building a MAS for AmI characterized by a context-aware topology, graph-based context representation and context patterns, as well as a behavior that assures that an agent is obtaining relevant information.

We have not discussed in this paper about the dynamic nature of context and how the agent topology should change in order to adapt to new contexts. Another aspect of future work is the integration of uncertainty in the representation of information.

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 and by Agence Universitaire de la Francophonie.

[3] We will shorten the writing of subgraphs of the from $? \xrightarrow{isa} Type$ to just $?Type$.

REFERENCES

[1] *K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten and J. Burgelman.* Scenarios for ambient intelligence in 2010. Tech. rep., Office for Official Publications of the European Communities, 2001.

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

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

[4] *C. Ramos, J. C. Augusto and D. Shapiro.* Ambient intelligence - the next step for artificial intelligence. IEEE Intelligent Systems, **vol. 23**, no. 2, pp. 15–18, 2008.

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

[6] *H. Chen, T. W. Finin, A. Joshi, L. Kagal, F. Perich and D. Chakraborty.* Intelligent agents meet the semantic web in smart spaces. IEEE Internet Computing, **vol. 8**, no. 6, pp. 69–79, 2004.

[7] *S. Costantini, L. Mostarda, A. Tocchio and P. Tsintza.* DALICA: Agent-based ambient intelligence for cultural-heritage scenarios. IEEE Intelligent Systems, **vol. 23**, no. 2, pp. 34–41, 2008.

[8] *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, pp. 12–20, 2004.

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

[10] *T. C. Lech and L. 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, pp. 625–631, 2005.

[11] *L. Feng, P. M. G. Apers and W. Jonker.* Towards context-aware data management for ambient intelligence. In *F. Galindo, M. Takizawa and R. Traunmüller*, eds., Proceedings of DEXA 2004, 15th International Conference on Database and Expert Systems Applications, Zaragoza, Spain, August 30 - September 3, vol. 3180 of *Lecture Notes in Computer Science*, pp. 422–431. Springer, 2004.

[12] *G. Cabri, L. Ferrari, L. Leonardi and F. 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, pp. 39–46, 2005.

[13] *I. Satoh.* 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, vol. 3446 of *Lecture Notes in Computer Science*, pp. 187–201. Springer, 2004.

[14] *C. Muldoon, G. M. P. O'Hare, R. W. Collier and M. J. O'Grady.* Agent factory micro edition: A framework for ambient applications. In *V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot and J. Dongarra*, eds., Proceedings of ICCS 2006, 6th International Conference on Computational Science, Reading, UK, May 28-31, vol. 3993 of *Lecture Notes in Computer Science*, pp. 727–734. Springer, 2006.

[15] *M. Hellenschmidt.* Distributed implementation of a self-organizing appliance middleware. In *N. Davies, T. Kirste and H. Schumann*, eds., Mobile Computing and Ambient Intelligence, vol. 05181 of *Dagstuhl Seminar Proceedings*, pp. 201–206. ACM, IBFI, Schloss Dagstuhl, Germany, 2005.

[16] *A. Dey, G. 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), pp. 114–128, 1999.

[17] *J. Hong and J. Landay.* An infrastructure approach to context-aware computing. Human-Computer Interaction, **vol. 16**, no. 2, pp. 287–303, 2001.

[18] *A. Harter, A. Hopper, P. Steggles, A. Ward and P. Webster.* The anatomy of a context-aware application. Wireless Networks, **vol. 8**, no. 2, pp. 187–197, 2002.

[19] *K. Henricksen and J. Indulska.* Developing context-aware pervasive computing applications: Models and approach. Pervasive and Mobile Computing, **vol. 2**, no. 1, pp. 37–64, 2006.

[20] *M. Baldauf, S. Dustdar and F. Rosenberg.* A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing, **vol. 2**, no. 4, pp. 263–277, 2007.

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

[22] *A. Olaru, A. El Fallah Seghrouchni and A. M. Florea.* Ambient intelligence: From scenario analysis towards a bottom-up design. In *M. Essaaidi, M. Malgeri and C. Badica*, eds., Intelligent Distributed Computing IV, Proceedings of the 4th International Symposium on Intelligent Distributed Computing - IDC 2010, Tangier, Morocco, September 16-18 2010, vol. 315 of *Studies in Computational Intelligence*, pp. 165–170. Springer Berlin / Heidelberg, 2010.

[23] *M. Perttunen, J. Riekki and O. Lassila.* Context representation and reasoning in pervasive computing: a review. International Journal of Multimedia and Ubiquitous Engineering, **vol. 4**, no. 4, pp. 1–28, 2009.

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

[25] *C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan and D. Riboni.* A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing, **vol. 6**, no. 2, pp. 161–180, 2010.

[26] *A. Olaru, C. Gratie and A. M. Florea.* Context-aware emergent behaviour in a MAS for information exchange. Scalable Computing: Practice and Experience, **vol. 11**, no. 1, pp. 33–42, 2010.

[27] *A. El Fallah Seghrouchni, A. Olaru, T. T. N. Nguyen and D. Salomone.* Ao Dai: Agent oriented design for ambient intelligence. In *N. Desai, A. Liu and M. Winikoff*, eds., Principles and Practice of Multi-Agent Systems, 13th International Conference, PRIMA 2010, Kolkata, India, November 12-15, 2010, Revised Selected Papers, vol. 7057 of *Lecture Notes in Computer Science*, pp. 259–269. Springer Berlin / Heidelberg, 2011.

[28] *A. Olaru, A. M. Florea and A. El Fallah Seghrouchni.* Graphs and patterns for context-awareness. In *P. Novais, D. Preuveneers and J. Corchado*, eds., Ambient Intelligence - Software and Applications, 2nd International Symposium on Ambient Intelligence (ISAmI 2011), University of Salamanca (Spain) 6-8th April, 2011, vol. 92 of *Advances in Intelligent and Soft Computing*, pp. 165–172. Springer Berlin / Heidelberg, 2011.