Mobile Networks and Applications Journal manuscript No.
(will be inserted by the editor)

A context-aware multi-agent system as a middleware for Ambient Intelligence*

Andrei Olaru · Adina Magda Florea* ·
Amal El Fallah Seghrouchni

Received: date / Accepted: date

Abstract There is currently a lot of work in Ambient Intelligence – or AmI – reporting on specific scenarios, or on implementations of particular cases. In the same time, there is a common agreement of the fact that AmI applications should be pervasive, covering a large number of devices, assisting a large number of people, and serving a large number of purposes. In an attempt to achieve scalable scenarios and implementations, we have focused our research on the development of a generic middleware layer for the context-aware transfer and exchange of information between devices. This paper presents a model for a such middleware, based on software agents, in which context-awareness is implemented both in the agent's representation of context information, and in the logical topology of the agent system. The model is oriented towards decentralization of the system and relies mostly on local behavior. The paper also reports on several proof-of-concept applications that have been developed and tested using the proposed model, proving thus the validity of the approach.

Keywords Ambient intelligence · Context-awareness · Multi-agent systems

* The final publication is available at `www.springerlink.com`

* Both Andrei Olaru and Adina Magda Florea are main authors of this paper.

This work has been supported by CNCSIS–UEFISCSU, project number PNII–IDEI 1315/2008.

Andrei Olaru · Adina Magda Florea
Department of Computer Science, University Politehnica of Bucharest
313 Splaiul Independentei, 060042 Bucharest, Romania
Tel.: +40-21-4029358
E-mail: cs@andreiolaru.ro, adina@cs.pub.ro

Amal El Fallah Seghrouchni
Laboratoire d'Informatique de Paris 6, University Pierre et Marie Curie
4 Place Jussieu, 75005 Paris, France
Tel.: +33-1-44278753
E-mail: amal.elfallah@lip6.fr

1 Introduction

Ambient Intelligence – or AmI, for short – is a vision of a future where people will be constantly surrounded by a very large number of devices – mostly sensors and actuators, but also smart appliances and devices – which, by means of communication and collective reasoning, will help people and assist them in their day-to-day tasks [6].

There are two aspects in the real-scale deployment of an Ambient Intelligence system that are central to our work. One of them is scale, and how will the system remain available and useful throughout high loads and/or faults. The other is information transfer. Mark Weiser, considered the father of Ubiquitous Computing, sees such an environment as a world of information conveyers [35].

In an ideal future deployment, Ambient Intelligence will be a unified system that interconnects devices that are present in every object, in every material, in the whole world. The system will assist everyone continuously, in any situation, provide them with the appropriate information and actions with no delay, and even with a certain degree of anticipation. This behavior would be the result of processing large quantities of information. These requirements call for features like robustness, resilience and availability.

The information in an AmI system comes from various parts of the system, reaching those users and devices that are in need of it. Thus, the exchange and transfer of relevant information is the backbone of any AmI system. This means that dedicated components for this purpose should exist in an AmI system. As AmI is deployed on devices with highly heterogeneous capabilities, these components should also be adaptive in order to appropriately use the device's capabilities for the storage and processing of information.

Moreover, the information that is processed by the system would be relevant with respect to a wide range of applications and purposes, and serve the needs of a great variety of users. Ideally all the necessary information should be transferred through a unitary system, using the same algorithms to compute relevance regardless of the user, application, and domain.

In this context, the question that we want to answer is: in a future real-scale AmI system, how should the system handle the generic part of the management and exchange of information? In this paper we look into how to build a middleware for Ambient Intelligence which deals with the generic transfer of relevant information from the entities (devices or users) that provide it, to the entities that need it. The middleware would transfer the information via the ubiquitous network in the AmI system, and it would offer the relevant information directly to the user via the interface, or to an intermediary layer of application-specific processing. In this way, many features that are common to many AmI applications and scenarios would be isolated in the middleware: availability, robustness, context information retrieval and generic context-aware actions (based on situation detection).

Our approach is based on two key elements. The first one is software agents as the building blocks of the middleware. Agents offer features that answer to key needs in Ambient Intelligence [9, 28]. Autonomy enables agents to act and take decisions with or without information from other agents, and to easily adapt to changing contexts. Reasoning and proactivity allow agents to act before the user requires it explicitly, making the AmI system be more useful and appear more intelligent. Especially coupled with mechanisms of self-organization, using a multi-agent system means more effective and reliable delivery of the relevant information, and a high level of decentralization.

Context-awareness is central to Ambient Intelligence, and it is the second fundamental element of our approach. Context is defined as the interrelated conditions in which something exists or occurs. In the case of Ambient Intelligence, it denotes those elements that affect the interaction between the user and the system. A context-aware application adapts its behavior to context. In our perspective, the relevance of information with respect to a user is directly linked to the context of that information and to the context of the user. This is why a middleware for information transfer should integrate context-awareness at a fundamental level.

The integration of context-awareness is done by the use of context patterns [24], which represent situations that are relevant to the agent, and may indicate the appropriate action that the agent should take. Context information and context patterns are represented by means of graphs, leading to more flexibility, and allowing the use of existing graph matching algorithms for situation detection. Context-awareness and improved performance is also obtained by the integration of context-related agent topologies, which employ mobile agents, and which are particularly appropriate in the condition of mobile devices and dynamic context [9].

This paper presents the model for a middleware that gathers the features presented above and shows how this model may be used in different scenarios and applications. The main features of the model, which are validated by implemented applications, are the context-aware topology of the agent system, the integration of context patterns and context matching to detect situation, and an agent behavior that relies on exchanging potentially interesting information between agents that share context.

After a look into the related work in the field in the next section, our general approach to the problem is presented in Section 3. The architecture of the system, the implementation of context-awareness, and the structure and behavior of agents are detailed in Section 4. Section 5 enumerates several proof-of-concept applications and experiments that constitute parts of the effort to build the modeled middleware. The last section draws the conclusions and gives some insight on future work.

2 Related Work

The idea of creating a middleware for Ambient Intelligence is not new. In fact, most systems for ambient intelligence feature some sort of middleware, as a layer between the human-machine interface and the hardware.

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 [4,5,13,31,33]. There is however research that concerns larger number of agents, distributed control, and fault tolerance, as we will see in the following.

The SpatialAgents platform [32] 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 [2] 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 [22] 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 [8] 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.

Agents with reduced memory and performance footprint for AmI have been developed in the Agent Factory Micro Edition project [23]. 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.

In our work, we are trying to focus only on one layer of an Ambient Intelligence environment, and use agents 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.

There are many implementations of middleware for Ambient Intelligence that do not rely on software agents. As in our model agents are a core component, it is difficult to compare it with non-agent-based approaches. The main advantage of agent-based systems is that the agent paradigm and agent-oriented development tools offer features (like autonomy and proactivity) that are integrated in the paradigm itself. Another important advantage is the intelligence of the system, which is at the core of the agent-oriented paradigm, which means that the system and the entities composing it can learn and reason by themselves, without the need to program it for every specific task [20, 30].

Some mechanisms that we use are similar to the Directed Diffusion model [19] used for wireless sensor networks, and we are using similar techniques to spread information through the system based on local agent interaction. However, we use a more complex context representation for a much more refined spread.

As for context-awareness, certain infrastructures for the processing of context information have been proposed [1,11,14,16,18,22], containing several layers: sensors, processing, storage and management, and application. This type of infrastructures are 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. Moreover, these infrastructures are usually centralized, using context servers that are queried to obtain relevant or useful context information.

Modeling of context information uses representations that range from tuples to logical, case-based and ontological representations [27]. Henricksen et al use several types of associations as well as rule-based reasoning to take context-aware decisions [16]. While ontologies make an excellent tool of representing known 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 concepts. In previous work we have proposed a more simple, but flexible and easy-to-adapt dynamical representation of context information, based on the notions of concept map and conceptual graph [24].

Our model differs from infrastructures in which context information is produced and consumed by entities situated below and above the middleware [2, 29], in that the agents are both able to extract information from what they receive, and to create new information and disseminate it into the system. Our

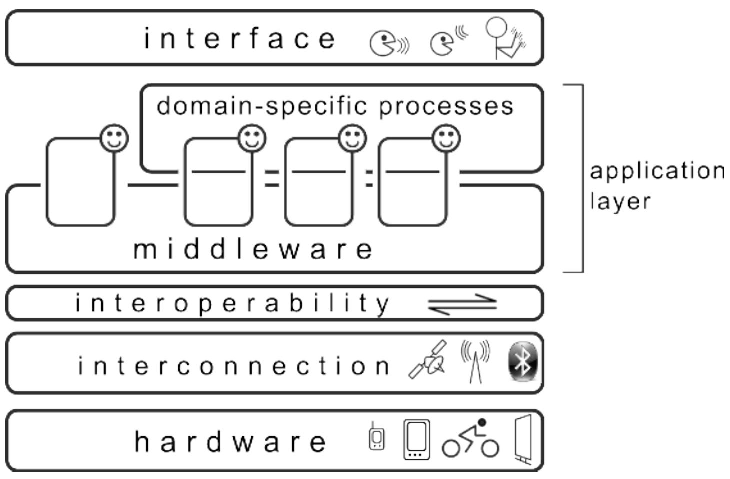


Fig. 1 A visual representation of the layers of an Ambient Intelligence system, as envisioned by Seghrouchni [7]. The application layer has been split between the middleware and domain-specific processes.

model for managing context is somewhat related to publish / subscribe architectures [21], but here the agents are both publishers and subscribers, and the information is not sent to / retrieved from centralized repositories, but sent directly to agents that may consider it interesting.

3 General Approach

We can view an Ambient Intelligence system as having several layers [7]. A visual representation of these layers is presented in Figure 1. The system is based on the hardware layer, composed of all the devices that are part of the AmI environment: sensors, actuators, intelligent appliances, smartphones and tablets, workstations and servers. All devices are interconnected by means of a ubiquitous network that uses various protocols, and mostly wireless transmission. The data that is transferred is offered to the layers above in a uniform format, and by means of standard protocols, assured by the interoperability layer. The context-aware, intelligent transfer of information, as well as specific context-aware services are offered by the application layer. The application layer interacts with the user by means of the intelligent interfaces, that support multi-modal, natural ways of communication with the human users.

Our aim is to build a middleware that will serve as a sublayer of the application layer, dealing with the context-aware, but generic (as in application-independent) transfer of information between the entities in the system. The application-specific processes will form a sublayer above the middleware. The middleware would be able to communicate with the user directly, through the interface, or would provide information to the application-specific processes.

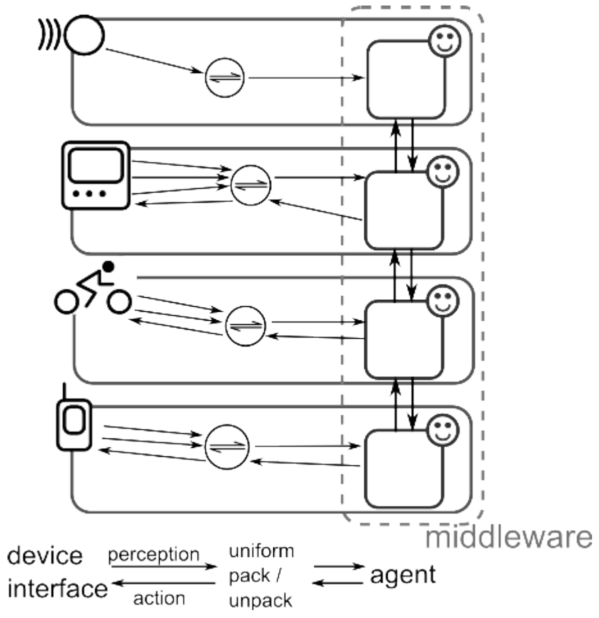


Fig. 2 A visual representation of the middleware. Software agents that compose the middleware execute on their respective devices. The agents use for input and output the sensors and interfaces provided by the devices, passing through an interoperability layer. The information exchanged between the devices is controlled by the agents.

The middleware would isolate several processes and features that are common to many, if not all, Ambient Intelligence applications and scenarios: the delivery of context information, captured from the environment, to the interested users or applications; the transfer of new information about users and their activity, to the concerned (and authorized) parties; the detection of the user's situation and context; in some cases, the decision on appropriate action that should be taken in the given situation. By isolating these processes in an AmI system, all there is left to do for users and applications is to "insert" information into the system, and this information will reach the other appropriate users and devices. The only requirement is that information inserted into the system respects the representation that is proposed. Obviously, it is still the application that performs domain-specific processing.

One important observation is that, as opposed to context-processing architectures that assure the flow of information from the perceiving entities (e.g. sensors) to the consuming entities (e.g. applications), the middleware that we propose is bidirectional – applications and users can also insert new or aggregated information into the system, and this will reach other applications and users.

In order to implement the middleware that we have described, we have chosen software agents as a building block. Agents – which are cognitive – can act by themselves, or collaborate as part of a multi-agent system [12].

Agents offer features that are very useful to Ambient Intelligence [28]: they are autonomous and adaptable, resulting in higher robustness for the system and in better service for the users; and they feature reasoning and are proactive, making the system appear as more intelligent and allowing anticipation. If used as an information transfer middleware for AmI, the flexibility and resilience of a multi-agent system, especially one featuring mechanisms of self-organization, means more effective and reliable delivery of the relevant information. A high level of decentralization would also lead to robustness and increased performance. In the middleware, it will be agents that will handle the flow of information between devices, and that will detect the context (using the information that they have) and choose appropriate action.

We also use mobile agents as entities that are able to offer services and domain-specific processing locally, helping both performance and a better and more privacy aware organization of information [9].

4 System Architecture

In the design of the system's architecture, there were two main priorities: first, keep a high degree of distribution and decentralization – this works in favor of the system's availability, robustness, and performance, especially coupled with the self-organization mechanisms integrated in the agents; second, isolate inside the middleware all processes related to the transfer of information between devices – this makes the system more modular and easy to be used together with different components.

In the resulting architecture, each agent in the middleware runs on a device that is part of the Ambient Intelligence system. There may be more agents on a device, that serve different purposes. The agent uses the interface of the device to communicate with the user, as well as the sensors and actuators that may exist on the device, to interact with the user's environment. To exchange information with agents situated on other devices, an agent uses the interoperability layer, as well as the networking capabilities of the device. This structure for the system is represented in Figure 2.

The positioning of agents with respect to the layers of the AmI system is presented in Figure 1: agents form the information exchange middleware, which interacts directly with the interface, or with the domain-specific processes. Agents use components from the interoperability, network and hardware layers to communicate by means of a uniform protocol, and also to retrieve sensor data and send actuator data in a uniform manner.

Our goal is to integrate context-awareness in the middleware for information exchange, so that agents in the multi-agent system naturally access, process and share context information.

Agents use context information to detect situations that have been preconfigured or that have been detected before, thus taking appropriate action. Pieces of context information are received from other agents and assembled into the context graph of the agent. Similarly, the agent sends pieces of context information to other agents that may be interested in them.

The integration of context awareness in the agent system has two aspects. On the one hand, the representation of context information inside the agents, coupled with the mechanisms for situation recognition. On the other hand, the context-aware topology of the agent system causes agents to exchange information only with other agents that share context with them, and also produces an implicit representation of context in the topology itself. These two aspects will be described together with the formal model for the middleware, in the sections below.

4.1 Formal Model

There are two important elements that are relevant to the functioning of the middleware:

- **containers** are entities that are bound to devices and where agents are able to move to, be create in, and execute on;
- **agents** are processes that execute inside containers, and that are able to move between containers, keeping their execution state intact; they are also able to communicate with other agents or components of the system.

For a good isolation of the information transfer processes, all communication between devices passes through the agents on those devices. This separates the concerns of domain-specific processing and information transfer.

The system is defined by two topologies, that are overlaid one on top of the other. These topologies are visible in the representation in Figure 3. While the topology of the network, which sets the relations between containers, is decided by the physical layout of the connections, the topology of the multi-agent system is based on logical hierarchies of agents, and changes depending on context, as we will see in Section 4.2. The multi-agent system is organized on three levels: containers, agents, and knowledge / context information. The state of the system at a given time represents the current state of the world, according to the perspective of the system as a whole. Likewise, the context graphs held by each agent represent the current state of the world from the perspective of the agent.

Containers are assigned to physical machines, and at any given time each agent executes on one container. The knowledge of each agent is represented by its context graph, and by its context patterns. Each of the three levels forms a graph, that is a subgraph of the whole three-level graph, that we will call the **Tri-Graph**. We will describe the components of the *Tri-Graph* in what follows.

The **containers** form a subgraph which we consider to be a complete (all containers can communicate with each other) – the *ContainerGraph*:

$ContainerGraph = (Containers, Connections)$

$Containers = \{Container \mid Container$ is a container that allows executing agents $\}$

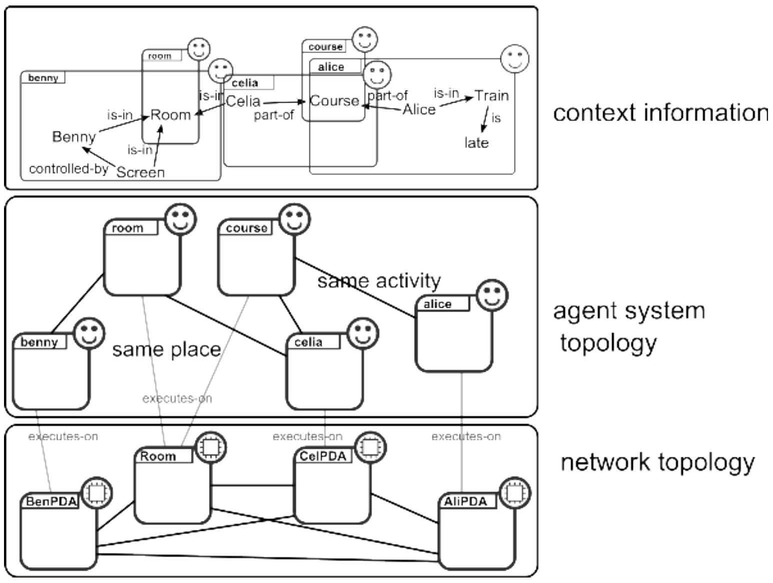


Fig. 3 A visual representation of the different graphs in the modeling of an example scenario involving 5 agents and 4 machines.

$Connections = \{\forall(C_i, C_j) \mid C_i, C_j \in Containers\}$

Connections shows from what container to what other container can the agents move. In this work it will be considered as containing a pair for each two containers. This works if the communication is done over the Internet through TCP/IP. This would change in a Wireless Sensor Network, for instance, where the connections between containers would depend on the physical layout of the sensor system.

The assignment between **agents and containers** is done by a supplementary component of the Tri-Graph: $AgentLocations \subset Agents \times Containers \times \{resides\text{-}on\}$.

4.2 Context-Awareness Outside the Agent

The **agents** form the subgraph $AgentGraph = (Agents, AgentRelations)$ in which edges are labeled with types of relations representing shared context.

$Agents \subset AgentNames$;

$AgentRelations = \{(A_i, A_j, Relation)\}$ where $A_i, A_j \in Agents$ and $Relation \in AgentRelationTypes$;

$AgentRelationTypes = \{is\text{-}in, part\text{-}of, of, in, controlled\text{-}by, executes\text{-}on\}$.

We propose the use of a **topology induced by context: if two agents share context, they should be neighbors**. Shared context can be a common activity, a common place, etc. The neighborhood relations in the topology represent context in a partial manner, by being related to the different types of context that are shared by the agents. Namely, two agents are neighbors in the topology of the system if and only if they share context. There are 5 aspects of context that are considered – the four aspects identified by Chen and Kots [3] and a fifth aspect identified by Henricksen et al. [16,17] – spatial context, temporal context, computational context, social context, and activity context.

For each type of context, we introduce types of agents and types of directed relations between agents, as follows:

- 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 useful only there, as it is specifically configured for 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: *executes-on* for services and *controlled-by* for devices, together with the *is-in* relation. In the case of devices, multiple parents for *Device* agents exist: 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 the device does not move – like a presentation screen).
- 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 the user's personal activities, the *Activity* agent is a child of the *User agent*, linked by a *part-of* relation.
- 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 other devices 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 common activity or being in the same place.

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

The role of the context-aware topology is, on the one hand, to organize the agent system depending on context, creating an implicit representation for context, and, on the other hand, to direct and restrict the communication between agents to particular contexts. There are two reason for this: first, a piece of information shared by an agent is not interesting for another agent that has no context in common with the sharing agent – information will remain in the context where it is relevant; second, when searching for devices, services, or other agents, the search will first take place in the current context, then in a larger context, and so on – it is more likely that the relevant results exist in a "closer" context. Restricting the communication only to agents that share context, by means of the context-aware topology, also leads to better performance of the system by restricting the number of agents that an agent is allowed communicate with.

4.3 Agent Structure and Context-Awareness Inside of the Agent

An agent A is modeled as a tuple $A(Name, CG_A, Patterns, \mathcal{R}, I, GoalList)$, with the following components:

$Name \in AgentNames$, the name of the agent;

$CG_A = (V, E)$, as defined below;

$Patterns = \{(G_s^P, relevance, persistence)\}$, defined below;

$\mathcal{R} \subset (AgentNames - Name) \times AgentRelationTypes \times \{in, out\}$, all relations with other agents (see Section 4.2);

$I = \{(Agent, s, factor) \mid Agent \in AgentNames, s \in patternNames, factor \in (0, 1]\}$, the observed interests of other agents for different patterns;

$GoalList = \{Goal(G', G_s^P, importance) \mid G' \subset CG_A, importance \in (0, 1], G_s^P \in Patterns\}$, the sharing goals of the agent, which contain a certain piece of information (subgraph of CG_A) that matches a pattern (see below).

The agent's Context Graph (CG_A) is in fact the agent's knowledge base, and represents all the information that the agent currently has (i.e. valid and relevant with respect to the current context), together with indications of persistence of the pieces of information (i.e. associations between concepts, represented as edges in the graph).

The set of *Patterns* represent the situations that the agent is able to recognize, and that it is interested in (in certain amounts, according to the *relevance* indication of the pattern). As presented in Section 4.5, the agent's patterns are at the base of the agent's behavior, and influence the information that it remembers, the information that it shares, and the actions that it takes.

The agent also knows the agents that are its neighbors in the topology, and the types of relations with them. These relations are dynamic, and they change when the context of the agent or of the neighbor changes. Information about these relations is also present in the Context Graph.

The agent is able to observe the interests of other agents, and this information is stored in the I component of the agent. Adding this component was one of the results of the studies and experiments performed with the AmIciTy framework, presented in Section 5.1. Knowing the interests of other agents means that the agent knows what pieces of information to send and whom to send them to. The interests of other agents are computed from the information that is received from them.

Finally, the *GoalList* contains pointers to the pieces of information to send to other agents, ordered depending on the importance of the goal. The goals of the agent are created when patterns match the context graph. Goals with lower priority may be removed before fulfilling, depending on the total length of the goal list.

Inside the agent, **context information** should be represented in a powerful, yet flexible manner, so that the same representation can be used on both capable and less capable devices. To support decentralization, agents should not rely strongly on centralized components, and must be able to use context information even in the lack of contact with the centralized components. An additional requirement is that agents should be able to easily aggregate information they receive, and that is interesting to them, with information already in their knowledge bases.

These are the reasons why we chose a graph-based, RDF-like representation for context information. Moreover, we introduced the notion of context patterns to define the interests of an agent and to help the agent detect the information that is relevant to its activity.

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

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

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

An agent also has a set of **context patterns** *Patterns*: $Patterns = \{(G^P_s,\ relevance, persistence) \mid\ s \in PatternNames,\ G^P_s$ a graph pattern, $relevance,\ persistence \in (0,1]\}$. The property $relevance$ shows how important an information from the context graph is, if it is matched by the pattern; $persistence$ shows for how long a new information will persist after being matched by a pattern.

A context pattern s contains a graph $G^P_s = (V^P_s, E^P_s)$ that has some special properties[1], i.e. can have question marks instead of vertex labels, and can have regular expressions as labels for edges. Edges are also characterized by two features: *characteristic* defines how characteristic the edge is for the pattern, and influences the measurement of how well a pattern matches a subgraph; *actionability* measures how correct it would be for the agent to infer the

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

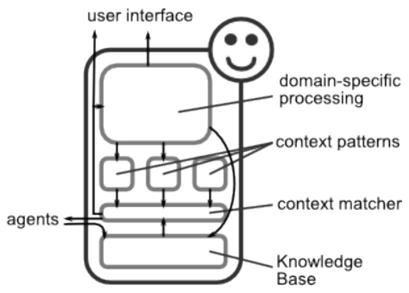


Fig. 4 An agent relies on its knowledge base to store external knowledge and context information. The agent's pattern matcher continuously matches patterns against the knowledge base, and activates domain-specific behavior or user notifications.

existence of this edge in the context graph if the pattern matches a subgraph in CG, but this edge is missing.

A **match** i between a context pattern G_s^P and an agent A's context graph CG_A is defined[2] [24] as $M_{A\text{-}si}(G'_A, G_m^P, G_x^P, f, k_f)$. G'_A, G_m^P, G_x^P are graphs: $G'_A \subset CG_A$ is the subgraph matched by the pattern, $G_m^P = (V_m^P, E_m^P)$ is the part of the pattern that matches G'_A (or solved part), and $G_x^P = (V_x^P, E_x^P)$ is the rest of the pattern, which is unmatched. There is no intersection (common nodes or edges) between G_m^P and G_x^P. When a pattern matches a graph, it means that every non-? vertex from the solved part matches (same label) a different vertex from G'_A, every non-*RegExp* edge from the solved part matches (same label for the edge and vertices) an edge from G'_A, and every *RegExp* edge from the solved part matches a series of edges from G'_A.

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

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

4.4 Unification of the Components

The **Tri-Graph** is formed by the union of the graphs for containers, agents, and agent knowledge: $Tri\text{-}Graph = (Nodes, Edges)$, where

$$Nodes = Containers \cup Agents \cup \bigcup_{A\ agent} CG_A.Concepts$$

$$Edges = Connections \cup AgentRelations$$
$$\cup\ AgentLocations \cup \bigcup_{A\ agent} CG_A.Relations$$

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

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

4.5 Agent Behavior

Agents in the middleware are supposed to handle all the information exchange between the different entities in the AmI system. Moreover, they should share and exchange information in a context-aware manner, so that the information reaches other agents that are interested in it, and that are suppose to obtain it in the current context.

The main components of the agent, as presented in Figure 4, are the Knowledge Base, the set of Patterns, and the pattern matcher. Other components are related to the communication between agents and the communication between the agent and the interface. An agent may also feature domain- or application-specific components. The agent communicates by means of information represented as graphs, mainly to share information with other agents. Both the interface layer and the domain-specific processes, as well as the other agents, are supposed to be able to work with this representation.

Agents in the middleware are defined by three behaviors in their work with pieces of information. First, the agent is able to **integrate** pieces of information that it receives from other agents, or from the layers above; second, the agent is able to **recognize context**, based on the context information that it has, detecting the situation in which the user is, and can either **take action** that is appropriate in the context, or **notify** the user (by means of the interface) or adequate processes in the agent of the context that was recognized; third, the agent **shares** context information that it considers relevant, with the agents to which it considers that information is relevant. Let us detail these behaviors bellow:

- An agent in the middleware receives various pieces of information from its environment: either from other agents in the middleware, or from the upper layers of the AmI system. When an agent receives a new piece of information from agent Ag, defined by a graph G'_{Ag}:
  - if the graph matches no context pattern with a sufficient k, the information is considered as irrelevant and discarded;
  - otherwise, the interest of agent Ag for the pattern(s) matched by the information is recorded in I;
  - match G'_{Ag} against CG_A (as graphs are particular cases of patterns, it is possible to match two graphs against each other); if any match has a sufficient k, assemble the new information in CG_A in the matched location;
  - otherwise, add G'_{Ag} to CG_A, without connection to the rest of the graph.
- Based on a continuous matching of its patterns against its context graph (CG_A), the agent is able to detect situations:
  - if a partial match is detected (with sufficient k), and there are actionable edges (above a certain threshold) that are connected to the matching part, the actionable edges are added to the graph; appropriate action, associated with the edges, is taken (examples include adding neighbors or moving to another machine);

– Sharing information is a fundamental process in our approach: the agent should share information that it finds relevant with agents that may be interested in that information:
  – if a partial or full match is found between a pattern and CG_A, the match is added to *GoalList* to be shared; importance of the goal is computed depending on the relevance of the pattern;
  – the agent considers the goals in the order of their importance;
  – when the agent makes a plan for a goal in *GoalList*, it chooses to share the information with the fraction of neighbors that are potentially interested in the matching pattern; the fraction is computed using information on the other agents' interests in I and the importance of the goal.

An important process that is not presented in the enumeration above is the removal of parts for the context graph. When new edges are added to the CG_A, it is because a pattern $s = (G_s^P, \mathit{relevance}, \mathit{persistence})$ k-matches a subgraph $G' \subset CG_A$ and a number of k edges are added to CG_A, with persistence according to the *persistence* parameter of the pattern. With time, persistence of the edges in the context graph fades, and as it reaches zero, the edge is removed (along with any resulting isolated nodes).

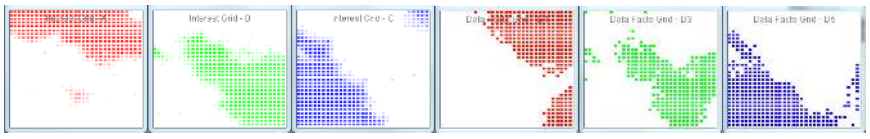
5 Model Validation

The different aspects of the model presented in the previous sections have been validated by means of several proof-of-concept applications. Each of these implementations validated one or more aspects of the presented model. Each aspect of the model was the answer to a specific question, and the applications needed to prove that the implementation correctly answered that question.

5.1 Large-Scale Emergent Information Dissemination

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 problem that is asked is: given a certain piece of information, how to deliver that piece of information to the interested users – the users to which that information is relevant? Development of the AmIciTy middleware was part of the answer to this question [25,26]. The implementation is based on two elements: the use of agents and the application of mechanisms of self-organization. In this application, agents used a behavior similar to the one presented in Section 4.5 in order to obtain context-aware sharing of information, by using local behavior and communication and limited knowledge and reasoning.

The experiments used a large number of agents that have a location in space, covering a rectangular area. The agents can only communicate with



(a) (b) (c) (d) (e) (f)

Fig. 5 In each panel, a view on the multi-agent system is presented: in panels (a)-(c), each cell corresponds to an agent and is colored according to the interest of the agent for a domain of interest (A, B or C, in this order), at simulation step 130; in panels (d)-(f), each cell in the grid corresponds to an agent and shows if it has a certain piece of information, at simulation step 271 – there are three pieces of information, each related to one of the three domains.

their spatial neighbors. The agents exchange pieces of information, characterized by several context-aware measures: first, **space** is implicitly considered, because of the structure of the system, that relies on local behavior and communication; second, temporal context is implemented as a period of validity for each piece of information, called **persistence**; third, each piece of information is related to certain domains of interest – it has a **specialty**; last, each piece of information carries a direct indication of its relevance (estimated by the source) – this is stored in the indiction of **pressure**.

The goal of the experiments was to observe how a piece of information that is inserted in the system reaches all agents to which it is relevant. Figure 5 shows some of the relevant results that have been obtained [25]: it is easy to see the direction of the spreading is strongly influenced by the preexisting specialty of agents. The other context measures also influence the spread: higher pressure makes information spread considerably faster; after the persistence of information expires, it quickly disappears from the system; and, of course, the first agents that get to know a piece of information are the ones closer to its source.

A secondary goal of the experiments was to show that the implemented agent behavior (using mechanisms of self-organization) leads to a scalable and robust system. Indeed, experiments show that information reaches a large number of agents (even agents not interested in it, that will quickly discard it, but not before disseminating it further). A piece of information may reach an interested agent by multiple, redundant paths. The performance of the system is also very good, and local behavior means the system is scalable.

This implementation showed that relying on a local behavior can lead to good global results. While the context measures were simple, they can be replaced with relevance computed using context patterns; and while the topology was simple, it can be replaced with a context-aware topology.

5.2 Context-Aware System Topology

The Ao Dai (Agent-Oriented Design for Ambient Intelligence) prototype [10] is an implementation of a multi-agent system using a context-aware topology. It validates the integration of context-awareness outside the agents.

The Ao Dai project has been implemented in CLAIM[3], an agent-oriented programming language [34] that is based on explicit declaration of agent's characteristics: knowledge, goals, messages, capabilities. It uses first-order predicate logic. CLAIM offers strong mobility of agents and allows agents to be placed in a logical hierarchy that can span across different machines.

The goal of the Ao Dai prototype is to demonstrate that using a hierarchical topology for the agent system, that maps to a partial, hierarchical representation of context, is a valid solution for building a decentralized system in which communication only happens between agents sharing context.

In Ao Dai, several context elements are represented by agents: places, computational devices, and users. Computational devices, as well as users, exist in certain places; computational devices can also be used by users. These relations can change over time, but the dynamic topology improves performance and scalability. For instance, searching for devices / services with certain capabilities is done first in the agent's subtree of agents, then the agents queries its parent, which in turn searches in a larger context. This procedure has two advantages: first, the communication is decentralized, and the system is able to scale; second, the first results will be found in a context that is closer to the user.

The Ao Dai prototype has been demonstrated in a simulated environment, running on two different machines, during the 5th NII-LIP6 Workshop, held in June 2010 in Paris, France[4].

5.3 A Platform for Testing AmI Applications

In order to further test the model that we have developed for a middleware for AmI, we have built a platform for the testing of agent-based AmI applications – the Ao Dai / tATAmI platform (towards Agent Technologies for Ambient Intelligence)[5]. It was implemented using a modular structure, and featuring tools for the visualization and tracking of agents, as well as for the realization of repeatable experiments, based on scenario files. The platform is underpinned by JADE[6] for the management and mobility of agents.

[3] 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.

[4] 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`

[5] The realization of the platform has been a collaborative effort of Andrei Olaru, Thi Thuy Nga Nguyen and Marius-Tudor Benea, under the supervision of prof. Amal El Fallah Seghrouchni and with the assistance of Cédric Herpson.

[6] Java Agent Development Framework `http://jade.tilab.com/`

The platform allows the implementation of various AmI applications and is meant to validate the model presented in Section 4.1 through the integration of all of its components.

The platform uses an evolved version of the CLAIM language, called S-CLAIM, which is simpler and easier to use. Figure 6 presents a snippet of S-CLAIM code, partially defining an agent. The definition is based on behaviors, which can be reactive or proactive. The agents in the Ao Dai platform use Knowledge Bases that are accessed by means of a small number of functions that use patterns to locate information in the Knowledge Base. The S-CLAIM code is designed to be simple enough so that it can be used by non-programmers – it relies on a small number of primitives and uses simple data structures.

There are two important features of S-CLAIM agents and of the Ao Dai platform, that come from the model that we propose for the middleware. First, as in CLAIM, agents have a logical hierarchy, that is kept organized as to map the structure of context; agents are mobile, so that it is easy for them to move, together with their children, in order to reflect changes in the context. Second, the S-CLAIM language is oriented toward working with patterns: information in the knowledge base is retrieved by means of matching it against a pattern, and also messages are matched against patterns before activating a behavior.

Because interoperability is important if we want to interface the middleware with other components of an AmI system, Ao Dai agents can use not only the FIPA protocol to communicate with other entities, but also Web Services. They can be both accessed as web services, and can access web services themselves. Web service integration is done using the same S-CLAIM primitives as for normal inter-agent communication.

Also important for an AmI-related component like the middleware that we propose is the possibility of deployment on various platform, smart devices being among the most relevant. This is why Ao Dai agents integrate the possibility to be deployed on Android[7] devices. This requires some changes in the interface of agents, but otherwise much of the code works out-of-the-box. Figure 7 presents the three tabs in the mobile interface: the connection with the platform, choosing agents, and visualization of the agent logs[8].

The Ao Dai platform has been tested as middleware for AmI applications using the SmartRoom at the NII Institute in Tokio – a room enriched with application-controlled devices and a wireless sensor network for the detection of people. The Ao Dai platform has been deployed on multiple machines and has been integrated with the different components of the room with virtually no effort.

The first experiments used the following **scenario**: the Computer Science Course is taking place in a new building of the University, that features sensors, people detection, devices controllable by means of web services, many

[7] `http://www.android.com/`

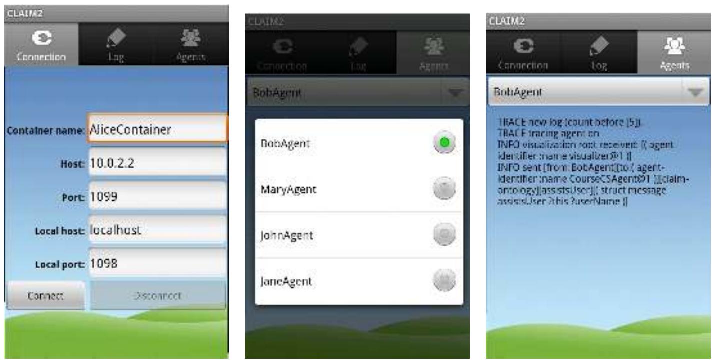
[8] These features have been developed by Marius Tudor Benea during his internship at LIP6.

```
CourseAgent.adf2 agent definition file
(agent Course ?courseName ?parent
  (behavior
    (initial register
      (send ?parent (struct message managesCourse this ?courseName))
    )

    (reactive registerUser
      (receive assistsUser ?agentName ?userName)
      (addK (struct knowledge userAgent ?userName ?agentName))
    )

    (reactive changeRoom
      (receive managesRoom ?roomAgentName ?roomName)
      (condition (readK (struct knowledge scheduling ?courseName ?roomName)))
      (addK (struct knowledge roomAgent ?roomName ?roomAgentName))
      (forAllK (struct knowledge userAgent ?userName ?userAgentName)
        (send ?userAgentName
         (struct message scheduling ?courseName ?roomAgentName))
      )
      (in ?roomAgentName)
    )
    ...
  )
)
```

Fig. 6 Sample of S-CLAIM code, used in the Ao Dai platform scenario for the *CS Course* agent.

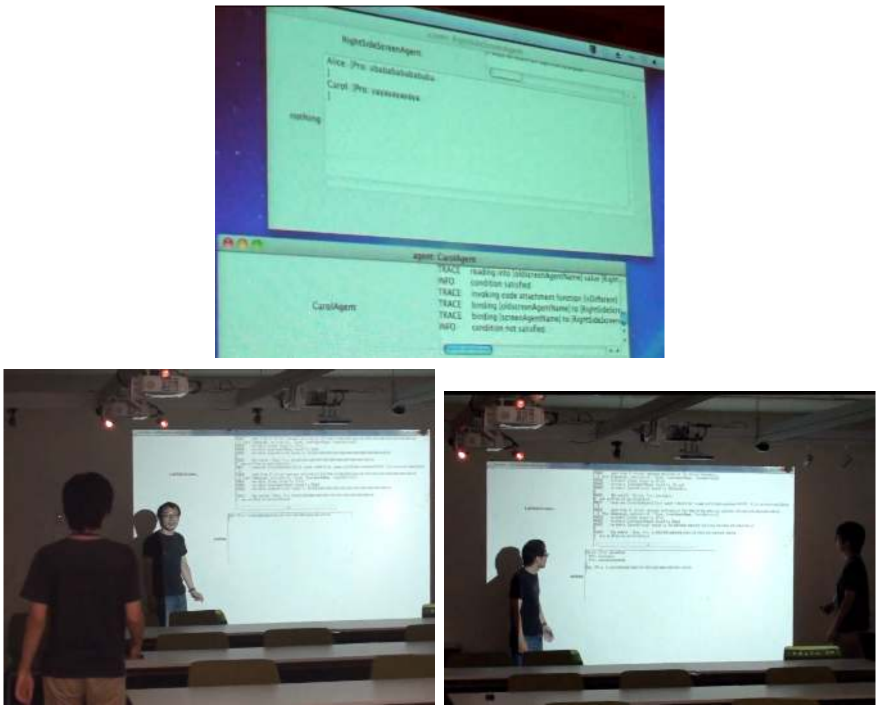


Container name: AliceContainer
Host: 10.0.2.2
Port: 1099
Local host: localhost
Local port: 1098

```
TRACE new log [count before [5]]
TRACE tracing agent on
INFO visualization root received: [( agent-identifier :name visualizer@1 )]
INFO sent [from: BobAgent][to:( agent-identifier :name CourseCSAgent@1 )][claim-ontology][assistsUser]]( struct message assistsUser ?this ?userName )]
```

Fig. 7 Interface of Ao Dai agents, on Android OS.

multi-purpose screens, and other AmI-specific equipment. As students that are new to the building approach the room where the course will be held, they are guided towards the room by messages on their PDAs. When the first person enters the room, it automatically configures itself (lights, screens, microphones) for the course, and it will also reconfigure in every of the different phases of the course. Among these, the activity phase consists of the students being divided into groups and discuss certain approaches to a problem, using their PDAs to enter opinions for or against the approaches. As they move in



```
TRACE   reading into [oldScreenAgentName] value [Right
INFO    condition satisfied
TRACE   invoking code attachment function [isDifferent]
TRACE   binding [oldScreenAgentName] to [RightSideScr
TRACE   binding [screenAgentName] to [RightSideScreen
INFO    condition not satisfied
```

Fig. 8 Images from the testing of the Ao Dai platform in the SmartRoom. The two last images show how opinions are moved to the back screen after the student changes location. As further proof of the platform's flexibility, one of the machines (above) runs Apple OS X and the other (below) runs Microsoft Windows 7.

the room and between the groups, the content they entered follows them on the large wall screens.

As the experiments were recorded on video, some images from the experiments are presented in Figure 8. In the figure, one can see the simple agent interface, showing the agent's activity log, and some input and output fields, and, in the second and third images, we can see how, as a student moves between the two large screens, his opinions move automatically on the second screen.

The easy implementation and correct execution of the scenario proved that the platform is appropriate to implement AmI applications, and that the presented model is valid with respect to managing context-related information.

While the scenario that was used to test the Ao Dai platform together with the SmartRoom was very simple, what was important was that the platform was very easy to deploy and that the agents were written in a language that is easy to learn.

6 Conclusions

While many approaches to the development of Ambient Intelligence environments attempt to deal with all layers of an AmI system, and result in application-specific implementations, the work presented in this paper focuses o a single layer of AmI applications, which is responsible with the generic, but context-aware exchange of information between devices. We have presented an agent-based middleware model and we have integrated context-awareness at two levels: inside the agent, the agent's behavior using context patterns to recognize situation and to decide upon appropriate action; outside the agent, the context-related topology of the system making communication among neighbors ”local” in terms of context, helping decentralization and performance.

We have also presented several proof-of-concept applications of our proposed model: the AmIciTy project for the study of the context-aware dissemination of information in a system formed of a large number of agents; the Ao Dai prototype demonstrating the use of mobile agents for the implementation of an AmI scenario; and the Ao Dai platform which offers tools for the simulation and testing of AmI applications, using context-aware agents and a dedicated agent-oriented programming language.

Future work deals with several aspects of our research. The agent-based model introduced in this paper must be further tested, using different scenarios and other environments. Temporal context, uncertainty, and privacy of the information must also be considered in our approach. While the development of the middleware has been oriented towards high performance and reliability, these features must be tested thoroughly using formally defined scenarios.

Acknowledgements The authors would like to thank Cristian Gratie for the participation in the development of the AmIciTy platform; Thi Thuy Nga Nguyen and Diego Salomone Bruno for their participation in the development of the Ao Dai prototype; and Marius Tudor Benea and again Thi Thuy Nga Nguyen for their participation in the development of the Ao Dai platform, as well as to Cédric Herpson for his assistance and advise.

References

1. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing **2**(4), 263–277 (2007)
2. Cabri, G., Ferrari, L., Leonardi, L., Zambonelli, F.: 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)
3. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College (2000)
4. Chen, H., Finin, T.W., Joshi, A., Kagal, L., Perich, F., Chakraborty, D.: Intelligent agents meet the semantic web in smart spaces. IEEE Internet Computing **8**(6), 69–79 (2004)
5. Costantini, S., Mostarda, L., Tocchio, A., Tsintza, P.: DALICA: Agent-based ambient intelligence for cultural-heritage scenarios. IEEE Intelligent Systems **23**(2), 34–41 (2008)
6. Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.: Scenarios for ambient intelligence in 2010. Tech. rep., Office for Official Publications of the European Communities (2001)

7. El Fallah Seghrouchni, A.: Intelligence ambiante, les defis scientifiques. presentation, Colloque Intelligence Ambiante, Forum Atena (2008)
8. El Fallah Seghrouchni, A., Breitman, K., Sabouret, N., Endler, M., Charif, Y., Briot, J.: Ambient intelligence applications: Introducing the Campus framework. 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'2008) pp. 165–174 (2008)
9. El Fallah Seghrouchni, A., Florea, A.M., Olaru, A.: Multi-agent systems: A paradigm to design ambient intelligent applications. In: M. Essaaidi, M. Malgeri, 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, *Studies in Computational Intelligence*, vol. 315, pp. 3–9. Springer Berlin / Heidelberg (2010). DOI 10.1007/978-3-642-15211-5_1. URL `http://dx.doi.org/10.1007/978-3-642-15211-5\_1`. ISBN 978-3-642-15210-8 (ISI Proceedings)
10. El Fallah Seghrouchni, A., Olaru, A., Nguyen, T.T.N., Salomone, D.: Ao Dai: Agent oriented design for ambient intelligence. In: Proceedings of PRIMA 2010, the 13th International Conference on Principles and Practice of Multi-Agent Systems (2010)
11. Feng, L., Apers, P.M.G., Jonker, W.: Towards context-aware data management for ambient intelligence. In: F. Galindo, M. Takizawa, R. Traunmüller (eds.) Proceedings of DEXA 2004, 15th International Conference on Database and Expert Systems Applications, Zaragoza, Spain, August 30 - September 3, *Lecture Notes in Computer Science*, vol. 3180, pp. 422–431. Springer (2004)
12. Ferber, J.: Multi-agent systems: an introduction to distributed artificial intelligence. Addison-Wesley (1999)
13. Hagras, H., Callaghan, V., Colley, M., Clarke, G., Pounds-Cornish, A., Duman, H.: Creating an ambient-intelligence environment using embedded agents. IEEE Intelligent Systems pp. 12–20 (2004)
14. Harter, A., Hopper, A., Steggles, P., Ward, A., Webster, P.: The anatomy of a context-aware application. Wireless Networks **8**(2), 187–197 (2002)
15. Hellenschmidt, M.: Distributed implementation of a self-organizing appliance middleware. In: N. Davies, T. Kirste, H. Schumann (eds.) Mobile Computing and Ambient Intelligence, *Dagstuhl Seminar Proceedings*, vol. 05181, pp. 201–206. ACM, IBFI, Schloss Dagstuhl, Germany (2005)
16. Henricksen, K., Indulska, J.: Developing context-aware pervasive computing applications: Models and approach. Pervasive and Mobile Computing **2**(1), 37–64 (2006)
17. Henricksen, K., Indulska, J., Rakotonirainy, A.: Modeling context information in pervasive computing systems. Lecture notes in computer science pp. 167–180 (2002). URL `http://www.springerlink.com/content/jbxd2fd5ga045p8w/`
18. Hong, J., Landay, J.: An infrastructure approach to context-aware computing. Human-Computer Interaction **16**(2), 287–303 (2001)
19. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: A scalable and robust communication paradigm for sensor networks. Proceedings of MOBICOM 2000 pp. 56–67 (2000)
20. Johanson, B., Fox, A., Winograd, T.: The interactive workspaces project: Experiences with ubiquitous computing rooms. IEEE pervasive computing pp. 67–74 (2002). URL `http://www.computer.org/portal/web/csdl/doi/10.1109/MPRV.2002.1012339`
21. Kindberg, T., Fox, A.: System software for ubiquitous computing. Pervasive computing pp. 70–81 (2002)
22. Lech, T.C., Wienhofen, L.W.M.: 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)
23. Muldoon, C., O'Hare, G.M.P., Collier, R.W., O'Grady, M.J.: Agent factory micro edition: A framework for ambient applications. In: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (eds.) Proceedings of ICCS 2006, 6th International Conference on Computational Science, Reading, UK, May 28-31, *Lecture Notes in Computer Science*, vol. 3993, pp. 727–734. Springer (2006)
24. Olaru, A., Florea, A.M., El Fallah Seghrouchni, A.: Graphs and patterns for context-awareness. In: P. Novais, D. Preuveneers, J. Corchado (eds.) Ambient Intelligence - Software and Applications, 2nd International Symposium on Ambient Intelligence (ISAmI

2011), University of Salamanca (Spain) 6-8th April, 2011, *Advances in Intelligent and Soft Computing*, vol. 92, pp. 165–172. Springer Berlin / Heidelberg (2011). DOI 10.1007/978-3-642-19937-0_21. URL `http://dx.doi.org/10.1007/978-3-642-19937-0\_21`

25. Olaru, A., Gratie, C.: Agent-based, context-aware information sharing for ambient intelligence. International Journal on Artificial Intelligence Tools **20**(6), 985–1000 (2011). DOI 10.1142/S0218213011000498. URL `http://www.worldscinet.com/ijait/20/2006/S0218213011000498.html`
26. Olaru, A., Gratie, C., Florea, A.M.: Context-aware emergent behaviour in a MAS for information exchange. Scalable Computing: Practice and Experience **11**(1), 33–42 (2010). URL `http://www.scpe.org/index.php/scpe/article/view/637`
27. Perttunen, M., Riekki, J., Lassila, O.: Context representation and reasoning in pervasive computing: a review. International Journal of Multimedia and Ubiquitous Engineering **4**(4), 1–28 (2009)
28. Ramos, C., Augusto, J.C., Shapiro, D.: Ambient intelligence - the next step for artificial intelligence. IEEE Intelligent Systems **23**(2), 15–18 (2008)
29. Ranganathan, A., Campbell, R.: A middleware for context-aware agents in ubiquitous computing environments. In: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, pp. 143–161. Springer-Verlag New York, Inc. (2003)
30. Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., Nahrstedt, K.: A middleware infrastructure for active spaces. Pervasive Computing, IEEE **1**(4), 74–83 (2002)
31. Sadeh, N.M., Gandon, F.L., Kwon, O.B.: Ambient intelligence: The MyCampus experience. Tech. Rep. CMU-ISRI-05-123, School of Computer Science, Carnagie Mellon University (2005)
32. Satoh, I.: 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, *Lecture Notes in Computer Science*, vol. 3446, pp. 187–201. Springer (2004)
33. Spanoudakis, N., Moraitis, P.: Agent based architecture in an ambient intelligence context. Proceedings of the 4th European Workshop on Multi-Agent Systems (EUMAS'06), Lisbon, Portugal pp. 1–12 (2006)
34. Suna, A., El Fallah Seghrouchni, A.: Programming mobile intelligent agents: An operational semantics. Web Intelligence and Agent Systems **5**(1), 47–67 (2004)
35. Weiser, M.: Some computer science issues in ubiquitous computing. Communications - ACM pp. 74–87 (1993)