



International Journal on Artificial Intelligence Tools  
© World Scientific Publishing Company

## Agent-Based, Context-Aware Information Sharing for Ambient Intelligence

ANDREI OLARU, CRISTIAN GRATIE

*Computer Science Department  
University Politehnica of Bucharest  
313 Splaiul Independentei  
Bucharest 060042, Romania  
cs@andreiolaru.ro, cristian.gratie@cs.pub.ro*

Received (Day Month Year)

Revised (Day Month Year)

Accepted (Day Month Year)

In a future vision of Ambient Intelligence – or AmI – our surrounding environment will integrate a pervasive, interconnected network of sensors, intelligent appliances and computer-like devices. This implies, on the one hand, hardware and interface related issues, and, on the other hand, a layer of context-aware services that manages the large quantities of information generated throughout a system formed mostly of devices with limited capabilities. This paper presents the first steps toward the realization of the AmIciTy framework: a multi-agent system that relies on local interaction and the self-organization of agents, having as purpose the context-aware sharing of pieces of information. The paper presents the structure of the system, the design of the agents, the manner of building scenarios, experiments and the evaluation of a prototype.

*Keywords:* Multi-Agent Systems, Context-Awareness, Ambient Intelligence.

### 1. Introduction

One of the priorities of current development in the ICT domain is Ambient Intelligence, or AmI.<sup>6</sup> Ambient Intelligence deals with assisting people in their day to day activities, by means of an integrated, ubiquitous electronic environment that is interconnected by a heavy-duty network infrastructure and provides intelligent user interfaces. AmI will use a very large number of electronic devices that have different capabilities, different sizes and different performance, all of them interconnected by wireless or wired networks, working together and cooperating toward the resolution of tasks. By means of these devices, AmI will be sensitive to the environment and to the presence and state of people, and will be able to react to their needs and actions.<sup>1,19</sup> There are great challenges in the development of AmI, like advanced human-machine interfaces, knowledge representation, context-awareness, device heterogeneity, and many hardware-related requirements.

Many of these challenges relate to the layer between the hardware / network and the intelligent user interface. That is, once we have the devices, the sensors, the

interconnecting network, a certain level of interoperability and the user interfaces, the system will generate a large amount of information, that needs to reach the users that are interested in it. Such a system must not only be context-aware, proactive and anticipative, but, as the concept of Ambient Intelligence requires, must also be reliable and fault-tolerant. These are the challenges that we try to address with the AmIciTy framework for Ambient Intelligence.

The first steps toward the realization of the framework rely on the design of AmIciTy:Mi middleware, focused on the context-aware transfer of information, in a reliable and decentralized manner, using a multi-agent system. Agents are used because they answer to the requirements of AmI: autonomy, proactivity, reasoning, reactivity.<sup>19</sup> The decentralized paradigm is chosen because AmI must be reliable and dependable,<sup>23</sup> offering as many as possible of its services to the user, no matter what happens in the rest of the system, and regardless of the user's position and activity.

This paper presents the design of a multi-agent system for context-aware information sharing, in which agents interact only locally and use only very little knowledge on the environment. Context-aware behavior is obtained by means of a few context measures that are simple and generic, yet effective for obtaining a controlled spread of information among the agents: *pressure* tells how important the information is; *specialty* tells what the information is related to (domains of interest); *persistence* tells for how much time the information is valid. Depending on these measures and on its own knowledge, each agent decides whether to share the piece of information with others and whom to share it with. This decision based on the agent's knowledge makes the behavior of an agent, as well as the emergent behavior of the whole system, appear intelligent.

The system has been implemented and an appropriate example application is a wireless sensor network for the tracking of events (like the passage of people or vehicles). By means of the designed context-aware behavior, events captured by the smart sensors get to be known by the other agents / sensors to which the events are relevant.

This work continues previous research regarding emergent properties in cognitive agent systems<sup>17</sup> and is only a component of our ongoing research towards creating a middleware for Ambient Intelligence. Two other components study the topology of the agent system<sup>7</sup> and a more advanced representation for context information.<sup>18</sup>

The next section presents some work in fields related to our research. Section 3 describes the design of the system and the measures of context-awareness. Section 4 is dedicated to details on the structure and behavior of individual agents. The experimental scenarios and results are presented in Section 5. The last section draws the conclusions.

## 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.<sup>4,5,9,21</sup> There is however research that concerns larger number of agents, distributed control, and fault tolerance:

Context handling is considered by the AmbieAgents infrastructure,<sup>12</sup> 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<sup>8</sup>, but context queries are handled in a centralized way, making it efficient but not very scalable.

The LAICA project<sup>2</sup> 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<sup>22</sup> 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 represen-

tation 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.<sup>16</sup> 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<sup>10</sup> 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.

During the design of the middleware, we have studied many references in the domain of self-organization. Emergent properties can be of great help when dealing with limited agents but with heavy interaction. There has been much development in multi-agent systems that have emergent properties, especially by taking inspiration from biological systems.<sup>13,14</sup>

Self-organization has already been extensively used for wireless networks<sup>15</sup>, taking inspiration from both biological and social (as well as other) models. Query and routing are particular features that are present in some form in our model, but in a different context and with a different final goal.

Finally, some mechanisms that we use are similar to the Directed Diffusion model<sup>11</sup> 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 more context measures for better control of the spread.

### 3. System Design

The AmIciTy middleware is developed keeping in mind the scale and requirements of a real Ambient Intelligence scenario. In such a situation, there is a large number of users and devices (especially very many sensors and actuators). The devices communicate permanently and exchange a large quantity of information, coming from all the sensor perceptions, the users themselves, and from information aggregation. Most of the devices that are used have limited storage and processing capacity.

This is why we propose a middleware that is completely distributed. Each software agent in the middleware is assigned to and is executed on a device. Agents can communicate only if they are in a certain vicinity of each other, which is suited to our example of wireless sensor network.

Distribution and local interaction are also justified by the fact that most information is many times only relevant in a certain location or in a certain area. Decentralization also brings more reliability. From the very beginning the system was built so that it would rely on self-organization mechanisms. By means of self-organization, although generally the agents' capacities are limited and behavior is only local, as a whole the system may perform more complicated functions and appear intelligent.<sup>2</sup>

### 3.1. Context-awareness

By context we understand the conditions in which an event occurs and that are related to the event<sup>3</sup>. Context-awareness is the feature of a system (here, an AmI system) that makes the system behave differently depending on these conditions. Context information is used to calculate the relevance of different pieces of information, so that the user of the system will only receive the information that is relevant to him / her, in the current situation

To provide context-awareness for information sharing, we propose four simple and generic aspects of context-awareness, one of them being handled implicitly and the other three being represented by means of simple numeric values. First, space is inherently considered, because of the structure of the system, that relies on local behavior and communication. Second, temporal context is implemented as a period of validity for each piece of information. Third, each piece of information is related to certain domains of interest. Last, each piece of information carries a direct indication of its relevance (estimated by the source).

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

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

**Time persistence** – shows for how long the information is relevant. When its validity expires, the agents start discarding the piece of information.

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

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

The last three measures above (the first is not explicit) are numerical values: persistence is a value in the interval  $[0, 1]$ , with 1 meaning the information is valid forever, and 0 meaning it has expired; specialty is a vector in which each component has a value in the interval  $[0, 1]$  showing the degree of relatedness with a certain domain of interest, and the whole vector has a maximum norm of 1; pressure is also

6 *Andrei Olaru, Cristian Gratie*

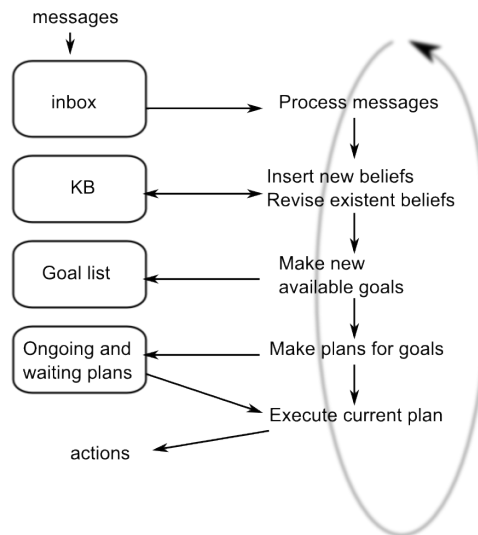


Fig. 1. The basic execution cycle of an agent.

a value in the interval  $[0, 1]$ .

Pressure, speciality and persistence are associated with the pieces of information by the source of the information – which is outside the system (in our example, they are decided upon by the smart sensor). The goal of this research was to see how the spread of information can be controlled by means of this measures.

These measures are associated with pieces of information present in the system, but agents themselves also feature indications of the context that they are in, namely their speciality and their pressure.

#### 4. Agent Design

Agents in AmIciTy:Mi have been designed so that they are simple, flexible, and so that an agent with the same structure can run both on the simple processor of a sensor and on a powerful computer. The agents are cognitive, and their model is inspired by the BDI model of agency<sup>20</sup>. In our experiments, particular attention has been given to agents that hold very small knowledge bases and that would be suited for very small devices like sensors.

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

#### 4.1. Agent Structure

The general structure and behavior of the agent is presented in Figure 1. Each agent has a message inbox, a knowledge base (KB), a list of available goals and a list of current plans.

The information in the agent's knowledge base is stored in *Facts*, where Facts are tuples of the form  $\langle Agent, knows, Fact \rangle$ . Note that the definition is recursive. At this point, the system is generic and does not study a real-life application. Therefore, facts that would normally represent useful information coming from the environment are replaced with Facts containing a *DataContent* placeholder, that has an identifier for tracing Facts relating to that information.

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

An agent also has measures of pressure and specialty of its own. These are global over its knowledge base. The agent's pressure is calculated as a weighted mean of the pressure of the facts that it knows, giving more weight to the facts with higher pressure. The specialty of an agent is initially a null vector, and it is updated with a certain factor depending on the specialties of facts that the agent knows. This way, the specialty of an agent "adapts" to the information that is currently being shared in the surroundings of the agent.

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

#### 4.2. Agent Behavior

The behavior of the agent has been designed so that it would be flexible and adaptive to context. It must be able to work on a very limited machine and also be able to process more information if working on a more powerful computer. The general behavior of the agents is quite common for cognitive agents: in its execution cycle, the agent processes messages that arrived in its inbox, integrates the new knowledge, then chooses a goal to make plans for, it makes a plan, and then executes one or more actions from the current ongoing plan. This cycle is presented in Figure 1. More details on the behavior are presented below.

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

In the next phase the agent forms a list of potential goals. There are two types of goals that an agent can have: *Inform* other agents of some information or *Free* some storage capacity. Each goal is assigned an importance, and the most important goal will be chosen as an intention. The reason for which we consider *Free* as a goal of the

agent is to have a uniform representation of all actions that an agent can perform, that are related to its knowledge or to its communication with other agents. The importance of *Inform* goals is computed according to the context measures of the corresponding fact:

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

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

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

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

Formally, we can consider that an agent  $A$  is defined as a tuple  $A = \langle KB, S_A, P_A, Goals, Plans \rangle$ , can receive and send messages  $m \in M$  and has the following functions:

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

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

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

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

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

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

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

The behavior of the agent was built and tweaked so that it will contain mechanisms enabling self-organization of the system. More precisely, there are feedback loops that are created: the agent state is influenced by the information it receives, and the decisions of what information it disseminates are influenced by its state; moreover, the agent will receive information that it itself has disseminated. This leads to the formation of "communities" inside the system, that have common specialty.

## 5. Implementation and Experiments

A proof-of-concept prototype was implemented in Java, with support for placing the agents in a grid structure, with direct communication only among adjacent agents, but also with support for a random placement of the agents and a maximal communication range. Experiments with 950 to 1000 agents were run, focused on observing characteristics of the spreading of data through the agent system, such as: the speed of the spreading, the coverage reached by each data piece and the particular areas preferred for spreading. The outputs of the experiments show how these characteristics are linked with the measures of context assigned to the information.

### 5.1. Scenarios and Visualization

The scenarios that were studied follow the following pattern:

- insert several pieces of information into the system, with different specialties and let them spread until their maximum coverage is reached and certain areas of interest are established in the system;
- insert "test" pieces of information, of different specialties, and observe how they spread according to previously established areas of interest, while the interest of each agent suffers only small changes; this stage also shows how old (and less relevant) information is forgotten by the agents as they receive the new data;
- insert one or two pieces of information of no particular specialty (equally related to all domains of interest) but with very high pressure, and observe if and how fast they reach all the agents in the system

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

color corresponding to one of the three domains. We will call the three domains  $A$ ,  $B$  and  $C$ .

As with any distributed system that is based on emergent behavior, a great number of experiments were needed in order to observe and tune the system. For this purpose, scenarios needed to be used, with different parameters and containing many events. Therefore, we have used XML files to characterize the scenarios in a simple and effective manner. The XML files use special tags designed to allow the specification of complex scenarios. For example, generating 15 instances of some data and placing them at random positions in the system is performed by the following XML snippet:

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

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

Several tools have been developed to help visualize the evolution of a system formed of a large number of agents. We have used two types of graphical outputs: distribution representations and graph representations. The distribution representation is a 2-dimensional representation of the agent system, showing one dot (or one cell) for each agent, placed at its respective location (see for instance Figures 2 to 6). Depending on the particular type of the distribution, there are fact distributions – the cell is visible if the respective agent has the data and the color of the cell depends on the specialty of the data (see Figure 2 (left), 4 and 5); per-domain interest distributions – the cell is visible if the agent is interested in that domain, the hue depends on the domain and the intensity of the color depends on the degree of interest (see Figure 3 (b), (c), (d)); and global interest distributions – the cell shows the specialty of the agent, the color of which is taken directly from the specialty vector (see Figure 2 (right) and Figure 3 (a)). Other types of visualization tools include the distribution of agent balance (Figure 7 (a) - (d)) and the graph for the average agent balance over time (Figure 7 (e)).

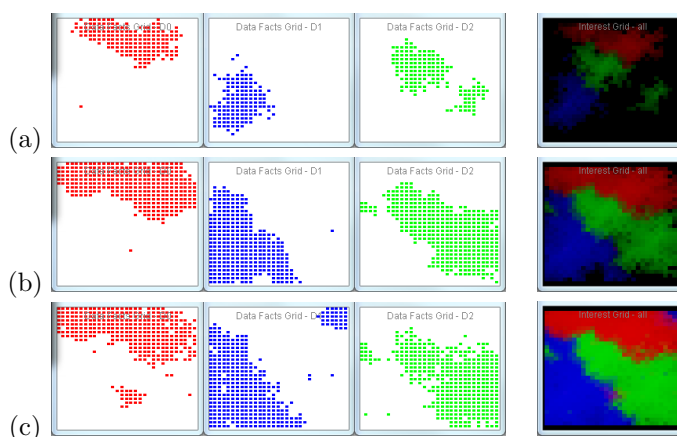


Fig. 2. The spread of three data pieces (left) and the evolution of agent interests (right) at simulation step: (a) 31; (b) 60; (c) 130. The corresponding data, from left to right, are  $A$ ,  $C$ ,  $B$ .

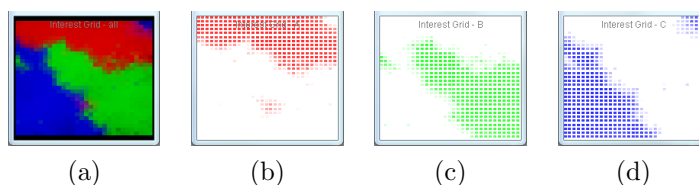


Fig. 3. The agent interests at step 130: (a) global, (b) for domain  $A$ , (c) for domain  $B$ , (d) for domain  $C$ .

### 5.2. Evaluation

Let us first see how the system evolves in the case of a scenario in the lines described in Section 5.1. What we are interested in is to see how the context measures that we have defined influence the spread of the “test” pieces of data.

Initially – at the start of the system’s evolution – none of the agents is interested in a particular domain. During the first phase of the experiment, three data pieces are used, with the following specialties regarding the given domains:  $(1.0, 0.0, 0.0)$ ,  $(0.0, 0.0, 1.0)$  and  $(0.0, 1.0, 0.0)$  – call them  $A$ ,  $C$ ,  $B$  according to the corresponding domain. 15 instances of each of the three data are injected in the system, into randomly chosen agents, at regular intervals until simulation step 150. Since the specialty of agents is influenced by the specialty of the data that is received by the agent, the agents’ interests will be grouped in contiguous regions, as can also be seen in Figure 2. Note that the regions only overlap at their borders. Figure 3 shows the agents’ interests as a whole and for each separate domain.

The next phase of the experiment consists in injecting, at simulation step 130 and around the center of the system, 3 new data pieces with the following specialties:  $(0.0, 1.0, 0.1)$  – call it  $Bc$ ,  $(1.0, 0.1, 0.0)$  – call it  $Ab$ , and  $(0.1, 0.0, 1.0)$  – call it  $Ca$ . Figure 4 shows what happens with these new data, as well as with the old data.

12 *Andrei Olaru, Cristian Gratie*

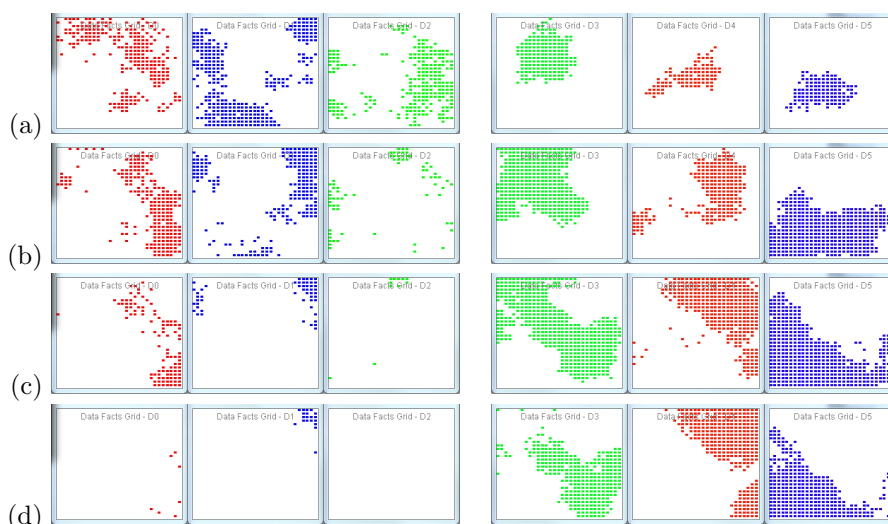


Fig. 4. Agents forget old data (left) as they receive new data (right) at simulation steps: (a) 152; (b) 170; (c) 210; (d) 271. The corresponding data, from left to right, are: *A*, *C*, *B*, *Bc*, *Ab*, *Ca*.

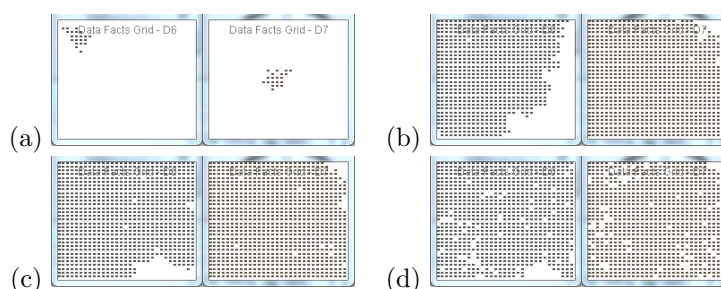


Fig. 5. Data with high pressure and relatedness to no particular domain spreading through the system at simulation step: (a) 211; (b) 300; (c) 350; (d) 447.

As shown in the figure, the old data is forgotten as new data arrives. Furthermore, the distribution of the new data is in accordance with the already established agent interests. Note that the less important specialty component (0.1 value) is also relevant to the spreading. Indeed, the *Ab* piece reaches agents interested in *A* but also the agents interested in *B*.

The last part of the scenario tests the behavior of the system for data that is equally related to all domains, but has high pressure. Two such pieces of data are inserted in the system, one at the upper left corner and the other at the center of the grid, at the simulation step 200. From the snapshots in Figure 5, one can see that these data manage to reach most agents. This last part of the scenario also shows that the existence of two data with high pressure in the system is not a problem (both of them spread to all agents).

The results are not limited to the grid structure (which is again a more con-

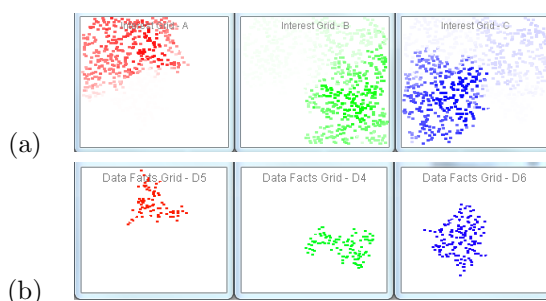


Fig. 6. (a) Areas of specialization according to the three domains of interest, with agents randomly placed. (b) Resulting distribution of data.

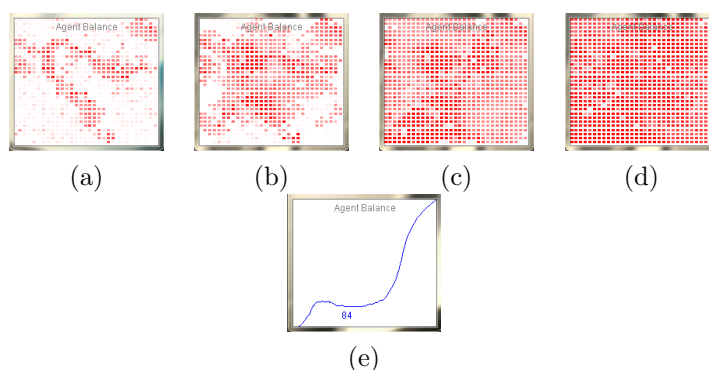


Fig. 7. (a)-(d) Evolution of the agents' balance at steps 130, 152, 170 and 210 – more intense color means better balance. (e) The graph of the average balance over all agents, between steps 0 and 210.

venient way for visualization purposes). Experiments have also been performed on agents placed randomly in the environment and communicating only with agents at a distance under a certain threshold. The obtained results were similar in nature, with the observation that the information took longer to spread, due to the many points where agents were too far to communicate. Results of the same type of experiment as above, but with randomly placed agents, are presented in Figure 6.

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

For further evaluation of our system, we have also developed a measure of satisfaction for agents. This measure deals with how relevant the facts that an agent has (that it has received from other agents) are with respect to its specialty. For every fact, an overall degree of usefulness is calculated, considering the history of

the agent's specialty (which is recorded throughout the agent's evolution) and the fact's specialty. While it is calculated using the agent's history, this measure is instantaneous. Based on it, a *balance* for the agent is calculated, that measures the balance between the useful facts (useful over a certain threshold) and the useless facts.

Let us observe how the agent balance changes throughout the system's evolution. We will use the same scenario that we used earlier, and we will observe the snapshots at the same steps. Figure 7 shows the evolution of the agent's balance. One can see that at start agents contain many facts that do not regard their specialty, therefore their balance is low. Later, as "test" data begins to spread, the specialty of agents is already formed, so the facts are deemed useful – the balance of agents is high, reaching 90% useful facts towards step 250. This means that the system functions so that the satisfaction of the agents reaches good levels after a certain time.

## 6. Conclusion

This paper presents a multi-agent system for the context-aware sharing of information. It uses limited agents that communicate locally, together with simple, generic measures of context, in order to produce relevant results. Experiments were carried out on a large number of agents in which the different measures of context showed relevant influence on how the information spreads, although agents have very limited knowledge about the system.

## Acknowledgments

This work was supported by CNCSIS - UEFISCSU, project number PNII - IDEI 1315/2008 and Grant POSDRU 5159. We would like to thank Sofia Neata for her contribution to this project during her undergrad internship.

## References

1. Juan Carlos Augusto and Paul J. McCullagh. Ambient intelligence: Concepts and applications. *Computer Science and Information Systems (ComSIS)*, 4(1):1–27, 2007.
2. Giacomo Cabri, Luca Ferrari, Letizia Leonardi, and Franco Zambonelli. The LAICA project: Supporting ambient intelligence via agents and ad-hoc middleware. *Proceedings of WETICE 2005, 14th IEEE International Workshops on Enabling Technologies, 13-15 June 2005, Linköping, Sweden*, pages 39–46, 2005.
3. Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, November 2000.
4. Harry Chen, Timothy W. Finin, Anupam Joshi, Lalana Kagal, Filip Perich, and Dipanjan Chakraborty. Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 8(6):69–79, 2004.
5. Stefania Costantini, Leonardo Mostarda, Arianna Tocchio, and Panagiota Tsintza. DALICA: Agent-based ambient intelligence for cultural-heritage scenarios. *IEEE Intelligent Systems*, 23(2):34–41, 2008.

6. K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.C. Burgelman. Scenarios for ambient intelligence in 2010. Technical report, Office for Official Publications of the European Communities, February 2001.
7. Amal El Fallah Seghrouchni, Andrei Olaru, Thi Thuy Nga Nguyen, and Diego Salomone. Ao dai: Agent oriented design for ambient intelligence. In *Proceedings of PRIMA 2010, the 13th International Conference on Principles and Practice of Multi-Agent Systems*, 2010.
8. Ling Feng, Peter M. G. Apers, and Willem Jonker. Towards context-aware data management for ambient intelligence. In Fernando Galindo, Makoto Takizawa, and Roland Traummüller, editors, *Proceedings of DEXA 2004, 15th International Conference on Database and Expert Systems Applications, Zaragoza, Spain, August 30 - September 3*, volume 3180 of *Lecture Notes in Computer Science*, pages 422–431. Springer, 2004.
9. Hani Hagrass, Victor Callaghan, Martin Colley, Graham Clarke, Anthony Pounds-Cornish, and Hakan Duman. Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems*, pages 12–20, 2004.
10. Michael Hellenschmidt. Distributed implementation of a self-organizing appliance middleware. In Nigel Davies, Thomas Kirste, and Heidrun Schumann, editors, *Mobile Computing and Ambient Intelligence*, volume 05181 of *Dagstuhl Seminar Proceedings*, pages 201–206. ACM, IBFI, Schloss Dagstuhl, Germany, 2005.
11. Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *Proceedings of MOBICOM 2000*, pages 56–67, 2000.
12. Till Christopher Lech and Leendert W. M. Wienhofen. AmbieAgents: a scalable infrastructure for mobile and context-aware information services. *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*, pages 625–631, 2005.
13. Marco Mamei, Matteo Vasirani, and Franco Zambonelli. Self-organizing spatial shapes in mobile particles: The TOTA approach. In Sven Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, editors, *Engineering Self-Organising Systems, Methodologies and Applications (after ESOA 2004 workshop)*, volume 3464 of *Lecture Notes in Computer Science*, pages 138–153. Springer, 2004.
14. Jean-Pierre Mano, Christine Bourjot, Gabriel Alejandro Lopardo, and Pierre Glize. Bio-inspired mechanisms for artificial self-organised systems. *Informatika (Slovenia): Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini*, 30(1):55–62, 2006.
15. Kevin L. Mills. A brief survey of self-organization in wireless sensor networks. *Wireless Communications and Mobile Computing*, 7(1):823–834, 2007.
16. Conor Muldoon, Gregory M. P. O’Hare, Rem W. Collier, and Michael J. O’Grady. Agent factory micro edition: A framework for ambient applications. In Vassil N. Alexandrov, G. Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *Proceedings of ICCS 2006, 6th International Conference on Computational Science, Reading, UK, May 28-31*, volume 3993 of *Lecture Notes in Computer Science*, pages 727–734. Springer, 2006.
17. Andrei Olaru, Cristian Gratie, and Adina Magda Florea. Emergent properties for data distribution in a cognitive MAS. *Computer Science and Information Systems*, 7(3):643–660, June 2010. ISSN 1820-0214.
18. Andrei Olaru, Amal El Fallah Seghrouchni, and Adina Magda Florea. Graphs and patterns for context-awareness. In Paulo Novais, Davy Preuveneres, and Juan M. Corchado, editors, *Proceedings of International Symposium on Ambient Intelligence, University of Salamanca (Spain), 6-8th April*, volume 92 of *Advances in Intelligent*

16 *Andrei Olaru, Cristian Gratie*

*and Soft Computing*, pages 165–172. Springer, 2011. ISBN 978-3-642-19936-3, ISSN 1867-5662.

19. Carlos Ramos, Juan Carlos Augusto, and Daniel Shapiro. Ambient intelligence - the next step for artificial intelligence. *IEEE Intelligent Systems*, 23(2):15–18, 2008.
20. Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In Victor R. Lesser and Les Gasser, editors, *Proceedings of the ICMAS-95, First International Conference on Multiagent Systems, June 12-14, San Francisco, California, USA*, pages 312–319. San Francisco, CA, The MIT Press, 1995.
21. Norman M. Sadeh, Fabien L. Gandon, and Oh Byung Kwon. Ambient intelligence: The MyCampus experience. Technical Report CMU-ISRI-05-123, School of Computer Science, Carnegie Mellon University, July 2005.
22. Ichiro 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*, volume 3446 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2004.
23. Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.