Ao Dai: Agent Oriented Design for Ambient Intelligence

Amal El Fallah-Seghrouchni, Andrei Olaru, Thi Thuy Nga Nguyen, Diego Salomone

Ao Dai: Agent Oriented Design for Ambient Intelligence*

Amal El Fallah Seghrouchni[1], Andrei Olaru[12] **,
Thi Thuy Nga Nguyen[13], and Diego Salomone[1]

[1] Laboratoire d'Informatique de Paris 6, University Pierre et Marie Curie,
4 Place Jussieu, 75005 Paris, France
[2] Computer Science Department, University Politehnica of Bucharest,
313 Splaiul Independentei, 060042 Bucharest, Romania
[3] Institute of French-Speaking Countries for Informatics,
42 Ta Quang Buu, Hanoi, Vietnam
amal.elfallah@lip6.fr,cs@andreiolaru.ro
ngaagn@gmail.com,diego.salomone@sma.lip6.fr

Abstract. In this paper we present mobile Multi-Agent Systems (MAS) as a specific paradigm to design intelligent and distributed applications in the context of Ambient Intelligence (AmI). We discuss how mobility, coupled with MAS, can be useful to meet the requirements of AmI. Indeed, the main features of mobile MAS, such as natural distribution of the system, inherent intelligence of the agents, and their mobility help to address a large scope of distributed applications in the domain of AmI. Other features of MAS, like multi-agent planning, context-awareness and self-adaptation are also very useful to bring an added value to AmI applications. They allow the implementation of both intelligent and collaborative agent behavior. This paper presents the Ao Dai project, that employs the mobile MAS paradigm, and serves as a prototype AmI environment. We also illustrate the functioning of the application through a scenario of user guidance in a smart environment.

Keywords: Ambient Intelligence, Mobile Multi-Agent Systems, Context-Awareness

1 Mobile Multi-Agent Systems

A Multi-Agent System (MAS) is an organization of a set of autonomous and potentially heterogeneous agents acting in a shared and dynamic environment. MAS represents (e.g. manages, models and / or simulates) physical systems (in robotics) or, more often, software systems. The MAS keystone is the double inference mechanism that is used by the agents. Agents, unlike other design

* Original publication at http://www.springerlink.com/content/4335601848jx0362

** This author is a PhD student in cotutelle between University Politehnica of Bucharest and University Pierre et Marie Curie.

paradigms such as objects or components, distinguish the level of task completion (or problem solving) from the level of solution control. Thus, they may act, observe their actions and change their own course of action. Agents have specific properties such as autonomy (an agent controls its condition and its actions regardless of any outside intervention); reactivity (an agent senses its environment and reacts to its changes); pro-activity (an agent tends to generate and achieve goals all by itself); and sociability (an agent interacts with other agents in the system). Within a MAS, agents interact to achieve cooperative (e.g. distributed problem solving) or competitive (e.g. coalition formation, auction) group behavior. Finally, a MAS is deployed in a environment that impacts its dynamic behavior.

The agent-based paradigm is particularly appropriate for the implementation of Ambient Intelligence [6, 16], because agents offer features that originate from the field of Artificial Intelligence and that are vital to the needs of Ambient Intelligence [11]. Autonomy is useful because individual devices in an Ambient Intelligence environment must be able to act on their own, without the need for user intervention or permanent control from centralized components. Learning can serve to adapt to the user's habits. And reasoning – as well as the capability to make plans – is what makes a system appear intelligent to the user.

The agent-oriented paradigm is also useful in modeling real-world and social systems, where optimal solutions are not needed and problems are solved by cooperation and communication, in a fully distributed fashion [11]. Currently, several agent-oriented programming languages exist [2], that allow the programmer to describe an application only by specifying the behaviour of individual agents.

Such an agent-oriented programming language is CLAIM, that also features a deployment platform for agents, called Sympa [14]. In CLAIM, each agent has a knowledge base, offers to the exterior a certain number of capabilities and is capable of both reactive (by means of rules) and proactive behaviours. More importantly, the multi-agent system has a structure that is inspired from ambient calculus [3]: agents are placed in a hierarchical structure and an agent can have another agent as parent, as well as several other agents as children. Agents in CLAIM are mobile – they are able to change the host on which they are executing, and they are also able to change their place in the hierarchical structure. Moreover, when an agent moves, its children move with it automatically.

Mobility means that agents can move (or migrate) within the organization of their associated MAS. In our framework, migration allows for dynamics that cover several aspects:

- the structure of the MAS (the organization of agents) may change over time due to openness (arrival and departure of agents) and to the evolution of functional requirements (creation / removal of agents).
- the dynamics of acquaintances between agents may appear (arrival or creation of agents), others may disappear (departure or removal of agents) and / or change (e.g. for mobile agents).

– the environment of the MAS may change which requires that agents perceive the changes and take them into account incrementally.

It is the hierarchical structure of CLAIM, as well as the strong mobility that it offers, that makes it especially appropriate for the implementation of an Ambient Intelligence system. That is because CLAIM makes it easier to implement context-awareness. An agent's ambient – formed by itself and all if its children – can represent a context. Agents can represent smart places, can manage smart devices, or can offer services.

The next section discusses several aspects in the implementation of Ambient Intelligence, like context awareness and representation. Section 3 describes the scenario and the implementation of the Ao Dai project – a proof-of-concept Ambient Intelligence systems implemented in CLAIM. The last section draws the conclusions.

2 Context-Awareness

One of the central features that makes distributed systems "intelligent" is *context awareness*. One of the definitions of context is *the set of environmental states and settings that either determines an application's behaviour or in which an application event occurs and is interesting to the user* [4]. One important point in the above definition is the relevance to the user. Either an event must be relevant to the user, or the application's behaviour must change so that it becomes relevant to the user. Context-awareness is the characteristic of an application that makes it change its behaviour depending on, and according to, context.

Research in the domain of context awareness has shown that there are many aspects of context. One classification of context [4] divides it into computational context – available computing and networking resources, including the cost for using them; user context – user's profile, location, people and objects nearby, social situation; physical context – light and noise levels, temperature, traffic conditions, etc; and time context – the current time coordinate of the user and related information (like the season, for instance). Context can be further classified [5] as primary – sensed directly by sensors and specialized devices – and secondary – which is inferred from the primary context.

If many authors consider context as merely a set of sensed values [1, 7], a particularly interesting approach to context-awareness is taken by Henricksen et al [8, 9], that model context as associations between entities or between entities and attributes, where an entity can be a person, a place, a communication device, etc. These associations can be of different types: static – associations that remain fixed for the lifetime of the entity; dynamic and sensed – obtained from sensors, usually transformed afterwards, changing frequently and subject to sensing errors; dynamic and derived – information that is inferred, usually from sensed or static associations; dynamic and profiled – introduced explicitly by the user, leading to greater reliability, but also subject to staleness.

In a context-aware system, there are several layers that deal with context information. One possible organization [15] uses three layers: data acquisition,

data interpretation and data utilization. However, considering that much context information is volatile (e.g. user's location and time), a context-aware system must also feature components for the degradation of context information.

Another important point in context-aware applications is the representation of context information. The choice of the representation technique is closely related to the system itself but some approaches are more appropriate to the field of AmI, like ontology-based models. This technique is the most promising for context modeling in ubiquitous environments [13]. It combines the assets of logic-based models and object-oriented technology [10], showing a higher level of robustness and expressiveness with the possibility of semantic representation.

In AmI systems, the heterogeneity of entities makes the global context representation more difficult due the differences between the context models of each agent. The ontology-based approach allows the different representations since it permits the agents to compare and share information. We need to process the information to compare the similarities between the possible representations to eventually arrive at a common understanding [12]. To avoid this problem, the most part of the implemented projects of Ubiquitous Computing usually work with a smaller part of a bigger scenario. For the sake of simplicity, they cover a closed environment with a global ontology as the base for context representation.

The main drawback of this approach is the definition of a centralized and universal ontology to be used by the system and all of its agents. In open AmI applications, the sensing capacity and incoming agents may change over time, affecting the system's needs. Thus, the MAS should be able to absorb, in some way, the new ontology information and, also, provide tools for the new agents' communication. This distributed ontology issue is an active research domain in part because of the Semantic Web [4] requirements.

3 Ao Dai project

3.1 Ao Dai Project scenario

In this project, we have studied several scenarios including the following (see also Figure 1): a user has a meeting in a building that he / she does not previously know. When arriving at the right floor, the user's PDA automatically connects to a local wireless access point. A CLAIM agent executes on the user's PDA – we will call this agent *PDA*. Another agent executes on a local machine and manages the context of the building's floor – call it *Floor*. *Floor* detects the presence of the user's PDA, and instructs the *PDA* agent to move in the agent structure and become a child of *Floor*. The movement is only logical: the agents keep executing on the same machines as before.

When *PDA* enters the floor, *Floor* also spawns a new agent – called *Navigator* – and instructs it to move as a child of *PDA*. This time, the movement is not only logical: *Navigator* is a mobile agent that actually arrives on the user's PDA and will execute there for all the time during which the user is on the floor.

[4] Semantic Web: http://www.w3.org/2001/sw/

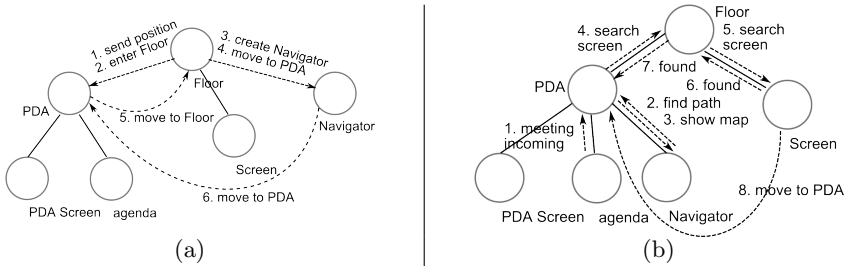

Fig. 1. Sequences of messages exchanged between agents: (a) *Floor* announces *PDA* of its new position, and instructs it to move as its child, then creates a *Navigator* that will offer services to *PDA*; (b) *Agenda* announces a new meeting, *PDA* asks a path from *Navigator*, which in turn requires a larger screen – which is searched on the floor, and found, then *Screen* moves as a child of *PDA*.

The *Navigator* can provide *PDA* (and, inherently, the user) with a map of the floor, can translate indications of the floor's sensors (sent to *Navigator* by *Floor*, and through *PDA*) into positions on the graphical map, and can calculate paths between the offices on the floor. *Navigator* is an agent that offers to the user services that are available and only makes sense in the context of the floor.

For displaying the map, *PDA* may detect that its screen is too small too appropriately display the map, so *PDA* will proactively initiate the search for a larger screen in the nearby area. The search can have several criteria: the space in which the search will take place (the current office, a nearby office, the whole floor), the range in which to search, and the minimal size of the searched screen. Devices are searched by the capabilities they offer – in this case the *display* capability is needed. *PDA* sends the query to its parent – *Floor* – which in turn locates among its children an agent *Screen*, that manages a physical screen that fits the requirements: it is located near the user and it is available. *Screen* answers the query and *PDA* asks it to move to become its child. Being a child of *PDA* also marks the fact that *Screen* is in use by the user, and *PDA* gains control over the displayed information. Agent *Screen* may either run on the actual intelligent screen, or may only manage the screen while being executed on a server. When the user moves farther from the screen, the PDA will detect that the context is no longer compatible and will free *Screen*, which will return to be a child of *Floor*.

3.2 Implementation

In the Ao Dai project, we have implemented a prototype of multi-agent system that handles several aspects of context-awareness, like user's location, available resources and user preferences. We have based ourselves in an extension of the scenario defined above. The project has been developed by Thi Thuy Nga Nguyen, Diego Salomone Bruno and Andrei Olaru, under the supervision of Prof. Amal El Fallah Seghrouchni.

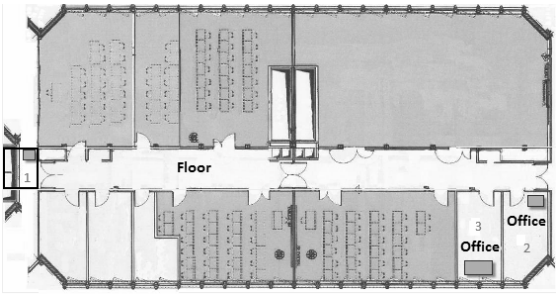

Fig. 2. The map shown by different screens in Ao Dai. There are three *Site* agents: *Floor* and two *Office* agents. Each one has a child of type *Screen*, representing the screens in the different places. The user starts on the floor (1) then moves to one office (2) and then to the other (3).

The prototype is implemented in CLAIM and executes on the Sympa platform. It features several types of agents: *Site*, which is used for "smart" places like *Floor* and *Office*; *PDA*, which directly assists the user from his personal device; *Navigator* and *Agenda*, which offer services to the user; and *Screen*, which represents a "smart" device with the capability of displaying information.

The prototype has been demonstrated during the 5th NII-LIP6 Workshop held on June 21-22 in Paris, France. The prototype was run on 2 machines. The *Floor* agent (of type *Site*) ran on one machine, and two *Office* agents (also of *Site* type) ran on the other machine. The floor and the two offices all featured screens of different sizes, managed by *Screen* agents (see Figure 2). During the demonstration, a *PDA* agent entered the floor, becoming a child of the *Floor* agent. A *Navigator* was created and sent to *PDA*. When the time of the meeting approached, *Agenda* announced *PDA*, which asked *Navigator* to find the path to the right office. *PDA* also searched for a larger screen, and found one near to the user, and automatically used it to display the map and the path. When the user – together with the PDA – moved to an office, the screen was freed and *PDA* with all children (*Agenda* and *Navigator*) moved to the other machine. There, the user explicitly requires a large screen, and *PDA* finds an appropriate one in the next room, and announces the user. The user then moves to the other office and *PDA* and all of its children move to become children of the agent managing that office. To simulate the interaction between the user and his personal agent *PDA*, an interface was created in Java (see Figure 2).

3.3 Programming in CLAIM

As an agent-oriented programming language, CLAIM [14] eases the task of implementing MAS. It works on top of Java, giving direct access to Java resources if needed. This language is based on explicit declaration of agent's characteristics. The following code shows a part of the definition of agent *PDA* in the Ao Dai project.

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

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

When the agent *PDA* (the PDA is initially characterized by its location and the size – w, h – of its screen) receives a message about its new location, it will execute the action "migrate". In this action, it checks if its actual location is already the location in the message (the variable ?*name*). If it is, the agent ignores the message. Otherwise, it moves to the new site by calling the function "moveTo()". If the new site is located in another computer in the network, the agent and its children will migrate to the new computer.

These characteristics are used to build the hierarchical relationship between agents in CLAIM. As a result, the MAS will be a set of hierarchies distributed over a network [14]. In the Ao Dai project, the agents of type Floor and Office ran on different machines to simulate the agents' migration.

The developer, in this case, need not to worry about the code migration and registration problems that may arise. The language takes care of it, concentrating the agents' information on the *Administration System* (see Figure 3). To address the security issues concerning mobile code, CLAIM offers some features like the agent's authority validation. The language also allows the developer to decide if an agent must have some special access or if an agent must have some resource denied. The sum of these features creates a powerful platform to the development of agent-oriented mobile applications.

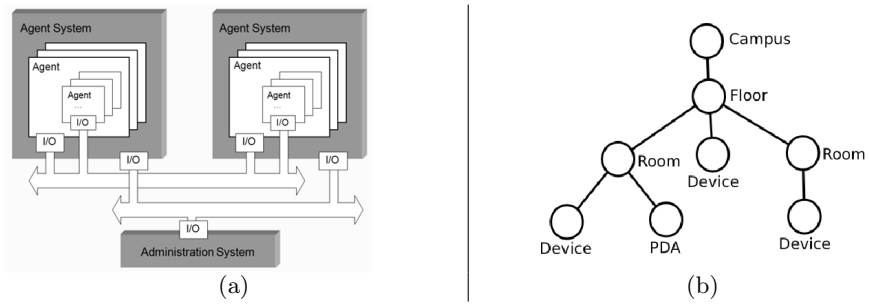

Fig. 3. System distribution in CLAIM: (a) Distribution over the network with each system deployed on a different machine; (b) An example hierarchy in Ao Dai.

3.4 Ao Dai agents

The given scenario has three major types of agents: Site agent (Floor, Office), Device / Service agent (Navigator, Agenda, Screen) and PDA agent. The latter with the specific role of representing the user during the simulation.

- The *Site* agent is used to determine the physical relationship between the agents. It means that an Office agent is a child of a Floor agent only if it is physically located on the given floor.
- The *Service* (or *Device*) agent has the capability to offer to the other agents some specific service. It may be in a direct or indirect way, like showing some information on the screen or advising other agents of the user meeting.
- The *PDA* agent works like a personal device that follows the user through his tasks. The most important features of this agent are that the PDA moves physically with user and has the CLAIM capability of managing requests for services or devices. It also stores the user's preferences.

3.5 Context Representation in Ao Dai

Location is, notably, the most used type of context in applications [5], because it reflects an important set of physical contents. In the Ao Dai project, besides location, we also consider, as part of the user's context, the available computing resources around him and his preferences.

In the first version of this project, the context is directly sensed (in a simulated manner) by the *PDA* and the *Site* Agents, but it is known that, in real applications, an additional layer is needed to capture the sensor information and translate it in useful data.

The context-awareness in Ao Dai is done by exploiting the particular hierarchical agent structure that is offered by the CLAIM language. In CLAIM it is very easy for the developer to instruct agents to move from one parent to another, and an agent moves automatically along with its entire sub-hierarchy of agents. This resembles the mobile ambients of Cardelli [3] and is an essential advantage when implementing context-awareness. That is because agents,

while representing devices or locations, can also represent contexts, allowing the developer to describe, in fact, a hierarchy of contexts.

For example, when the user is inside a room, its *PDA* agent is a child of the respective *Site* agent. The children of *PDA* – devices or services – are also in the same context. When the user moves to another room, the *PDA* agent changes parent and, along with it, its children move as well, therefore changing context. Some devices may not be able to move along with the user (e.g. fixed screens, etc.) so they will determine that the new context is incompatible with their properties, moving away from *PDA*.

But context is not only about location, and the hierarchical structure that is offered by CLAIM can be used for easy implementation of other types of context. One of them is computational context. When the user uses a service, a *Service* agent is created and becomes a child of *PDA*. It is easy for the service to interrogate its parent in order to find out more about its capabilities. Conversely, it is easy for *PDA* to check on its children – *Services* or *Devices* – in order to find the resources and capabilities that the user is able to use.

One last type of context that is handled in Ao Dai is user preferences. The user is able to input preferences on the capabilities of devices that it needs to use. These preferences are then integrated in the queries that are launched by the PDA (see Section 3.1). While the structure offered by CLAIM is not directly useful for this aspect, the preferences help find not only the closest device with the required capability, but also the closest device that fulfills certain user requirements. Preferences can also be used to limit the range of the search, which is meaningful from the context-aware point of view: a *Device* that is closer in the agent hierarchy also shares more context with the user.

3.6 Interaction Protocol

In a highly distributed AmI environment, a good representation of context and context-related relations between devices means that most of the communication will happen only at a local level, within the structure formed by these relations. In Ao Dai, the CLAIM agent hierarchy facilitates this: agents sharing a parent share a context.

To preserve the hierarchy, agents interact only with their parent and their children. Take for example the search for devices (see Figure 1). When agent *PDA* wants to search for a device with a certain capability and certain criteria, it must send a request to its parent, for example agent *Floor*. Once the request is received, agent *Floor* searches itself to see if it has the requested capability and satisfies the criteria. If it does, *Floor* answers immediately to agent *PDA*, in the other case, it searches in all of its children (if any) except the agent who invoked the search (agent *PDA*). After all of its children have answered, agent *Floor* checks if there are one or more children that have the capability requested and satisfy the criteria. If it has a confirmation answer, it sends the search result which contains the information about the found device(s) to agent *PDA* and the search is finished. If not, agent *Floor* has to search in its parent (if any). After the parent has answered, agent *Floor* sends the search result to agent *PDA* and

finishes the search. The process is executed recursively. User preferences can be used to limit the range of the search to closer contexts.

The advantage of using such a protocol in conjunction with mapping context over the agent hierarchy is that the search will usually end very quickly, assuming the user will most times ask for devices that are likely to exist in his context. The search is executed in the current context first, and then in the parent context and sibling contexts.

4 Conclusion

In this paper we have discussed the use of Mobile Multi-Agent Systems for Ambient Intelligence. Features like distribution, inherent intelligence of the agents, and mobility make MMAS a natural solution for the problems raised in the implementation of Ambient Intelligence environments. Other features of MAS, like multi-agent planning, collective learning and adaptation bring added value by allowing intelligent collaborative behaviour.

Additional challenges that MAS have to deal with in the context of Ambient Intelligence are issues like context-awareness, anticipation and user modeling. The paper discusses some of these issues and then presents the Ao Dai project, a prototype AmI environment, implemented as a multi-agents system, using the agent-oriented language CLAIM.

Ao Dai project is a preliminary work that will serve as a foundation of an international collaboration between four teams [5].

The prototype has been developed as a proof of concept and gave promising results. It shows that the hierarchy of the CLAIM language is very useful to capture different aspects of context-awareness. CLAIM also provides native primitives that allow agents to move – in a single step – between contexts, while their own context follows their movement.

As future steps in our research, integration of better mechanisms of anticipation, more types of contexts and improved context representation into the project will bring it closer to dealing with realistic requirements.

References

1. M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.
2. R. H. Bordini, L. Braubach, M. Dastani, A. E. Fallah-Seghrouchni, J. J. Gómez-Sanz, J. Leite, G. M. P. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, 30(1):33–44, 2006.
3. L. Cardelli and A. D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.

[5] MAS team from Paris 6, AIMAS from Politehnica of Bucharest, IFI form Hanoi and PUC-Rio from Brazil.

4. G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, November 2000.
5. A. Dey and G. Abowd. Towards a better understanding of context and context-awareness. *CHI 2000 workshop on the what, who, where, when, and how of context-awareness*, pages 304–307, 2000.
6. K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J. Burgelman. Scenarios for ambient intelligence in 2010. Technical report, Office for Official Publications of the European Communities, February 2001.
7. L. Feng, P. M. G. Apers, and W. Jonker. Towards context-aware data management for ambient intelligence. In F. Galindo, M. Takizawa, and R. Traunmüller, 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.
8. K. Henricksen and J. Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37–64, 2006.
9. K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. *Lecture notes in computer science*, pages 167–180, 2002.
10. R. Krummenacher, H. Lausen, T. Strang, and J. Kopeckỳ. Analyzing the modeling of context with ontologies. *International Workshop on Context-Awareness for Self-Managing Systems*, 2007.
11. C. Ramos, J. Augusto, and D. Shapiro. Ambient intelligence - the next step for artificial intelligence. *IEEE Intelligent Systems*, 23(2):15–18, 2008.
12. J.-P. Sansonnet and E. Valencia. Terminological heterogeneity between agents using a generalized simplicial representation. In M. P. Gleizes, G. A. Kaminka, A. Nowé, S. Ossowski, K. Tuyls, and K. Verbeeck, editors, *EUMAS*, pages 363–374. Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten, 2005.
13. T. Strang and C. Linnhoff-Popien. A context modeling survey. *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp*, pages 1–8, 2004.
14. A. Suna and A. El Fallah Seghrouchni. Programming mobile intelligent agents: An operational semantics. *Web Intelligence and Agent Systems*, 5(1):47–67, 2004.
15. J. Viterbo, L. Mazuel, Y. Charif, M. Endler, N. Sabouret, K. Breitman, A. El Fallah Seghrouchni, and J. Briot. Ambient intelligence: Management of distributed and heterogeneous context knowledge. *CRC Studies in Informatics Series. Chapman & Hall*, pages 1–44, 2008.
16. M. Weiser. The computer for the 21st century. *Scientific American*, 272(3):78–89, 1995.