*Journal of Universal Computer Science, vol. 12, no. 3 (2006), 284-296*
*submitted: 30/10/05, accepted: 15/1/06, appeared: 28/3/06* © *J.UCS*

# A Mechanism for Solving Conflicts in Ambient Intelligent Environments

**Pablo A. Haya**
(Dept. de Ingeniería Informática UAM - EPS, Spain
Pablo.Haya@uam.es)

**Germán Montoro**
(Dept. de Ingeniería Informática UAM - EPS, Spain
German.montoro@uam.es)

**Abraham Esquivel**
(Dept. de Ingeniería Informática UAM - EPS, Spain
Abraham.Esquivel@uam.es)

**Manuel García-Herranz**
(Dept. de Ingeniería Informática UAM - EPS, Spain
manuel.garciaherranz@uam.es)

**Xavier Alamán**
(Dept. de Ingeniería Informática UAM - EPS, Spain
Xavier.alaman@uam.es)

**Abstract:** *Ambient Intelligence* scenarios describe situations in which multitude of devices and agents live together. In this kind of scenarios is frequent to see the appearance of conflicts when modifying the state of a device as for example a lamp. Those problems are not as much of sharing of resources as of conflict of orders coming from different agents. This coexistence must deal also with the desire of privacy of the different users over their personal information such as where they are, what their preferences are or to whom this information should be available. When facing incompatible orders over the state of a device it turns necessary to make a decision. In this paper we propose a centralised mechanism based on prioritized FIFO queues to decide the order in which the control of a device is granted. The priority of the commands is calculated following a policy that considers issues such as the commander's role, command's type, context's state and commander-context and commander-resource relations. Finally we propose a set of particular policies for those resources that do not adjust to the general policy. In addition we present a model pretending to integrate privacy through limiting and protecting contextual information.

**Keywords:** Intelligent Environments, Ambience Intelligence, Context Awareness, Privacy
**Categories:** H.3.1, H.3.2, H.3.3, H.3.7, H.5.1

## 1 Introduction

The term Ambient Intelligence (AmI) appears in 1999 due to a report [1] of the Committee of Experts of the European Community's Information Society Technology

Programme (ISTAG). In 2001 ISTAG published a set of scenarios and recommendations defining the AmI vision.

AmI proposes a new Society of the Information focused in the user. For doing so it requires of efficient services that provide a more friendly interaction. In this new vision persons are surrounded by thousands of intelligent devices and interfaces merged in daily life objects. At the same time, devices and users coexist in an environment able to reason and react in a personalised fashion. The final goal is to achieve a non invasive help of the technologies of the information in the daily tasks of people.

AmI roots can be found in the combination of three different technologies: Ubiquitous Computing [2], Context Awareness Applications [3] and Intelligent Environments [4]. Those three areas have been object of study within the Department of Computer Engineer of the UAM. For that study it has been built an environment based on the living room of a digital home. Lately, the study has been extended to learning settings (classrooms, learning at home, etc.)

In an environment shared by numerous distributed components a variety of control problems arise when accessing the resources. Within the Intelligent Environments area different solutions based on security policies [5] and access control lists [6] have been proposed.

Once the security mechanism has been established we should face the problem arising when two or more components pretend to take the control over the same resource simultaneously (i.e. a device such as a lamp). The solution to this conflict pass through choosing which component has the privilege of accessing first the resource. A problem related with this one has been studied within the domain of distributed mutual exclusion.

The closest solutions to the Intelligent Environments can be found in Multiagent Systems. Open Agent Architecture [7] and Hive [8] are two platforms that have faced this problem. A possible approach is to use of a centralised implementation [9]. For doing so a coordinator component is required for receiving all requests and deciding which component will have permission to access the resource. Compared to a centralized solution, distributed algorithms provide more robustness due to the lack of a main central component. In the other hand they are less efficient due to the number of messages that they need to transmit between processes and they also suffer of a higher complexity in their implementation.

Our proposal for solving those conflicts within an Intelligent Environment pleads for simplicity and efficiency. In addition, we propose a global model of the world as the more effective way of achieving co-operation among heterogeneous agents. Thus we have chosen a centralised solution from the logical point of view, although its implementation may be centralised or distributed. On the other hand, the middle layer gluing the components of the Intelligent Environment (see section 2) lies in a common repository of information thus making easier the implementation of a centralised algorithm.

In the scenarios present in an Intelligent Environment priorisation between requests turns out to be essential. Generally it is common to find different roles between users of the environment (father/son, owner/guest, teacher/student, administrator/user…) and the desire to make a distinction when the action is executed by one or the other. In this manner the preference in the access of a resource is

established according to the role played by the user. However, in an Intelligent Environment there is another question to solve that has not been taken in consideration by the mechanisms explained before. It is not rare to find situations in which a user needs to take the control over a resource being used by other user. Considering the different roles and the situation of the environment the system may take the decision of withdrawing the control from the entity currently controlling the resource. These kinds of decisions are critical in scenarios such as a house, where the security modules (i.e. gas, flooding or thief detectors) may have preference to the rest of modules in case of emergency.

Our proposal consists on a centralised queuing mechanism in which the actions over the resources are prioritised. The requests are stored in the queues according to their priority, so the first element exiting the queue is the one with the higher priority. Finally we decided to use preemptive priority queues. These queues are characterized by the fact that if an incoming request has a priority higher than the one in course, then it takes the control of the resource. Oppositely, in non preemptive queues the incoming request has always to wait for the current request to finish. However, priority mechanisms force agents to consider that their requests may not be attended: a low priority request may stay indefinitely in the queue if requests with higher priorities keep continuously arriving.

Thus we should contemplate the case in which a high priority request takes the control over a resource indefinitely. A simple and effective method for solving this indefinite wait is to establish a time limit in reserving a resource. For example, the Jini middleware [10] provides a mechanism that allows a component to reserve the access to a resource for a limited fixed time. In order to keep the control of the resource, the component should update the concession before it expires.

In a similar way, to avoid the perpetual waiting of processes to get their requests satisfied, an expiring time is been added to the requests: when this time has elapsed the requests are not valid any more. This temporal constraint is established by the requesting process so it can control whether its request is outdated or not.

The next section will go over the architecture of the implemented middleware layer. In the following sections it will be explained the proposed conflict solving mechanism and how it is implemented and integrated within the middleware layer. Finally, privacy issues in the proposed architecture are discussed.

## 2 Context layer

A domotic environment has been created at the department of Computer Science of the UAM, simulating the living room of a conventional home [11]. This environment integrates different hardware and software components that allow controlling the various devices of the living room. Several applications sensible to the context of the user have been developed, as well as web-based [12] and oral [13] user interfaces.

A middleware layer has been developed to facilitate the integration of devices, applications and user interfaces. This middleware layer has been named “context layer” since the coordination mechanism is focused in the interchange of context information between the components of the environment.

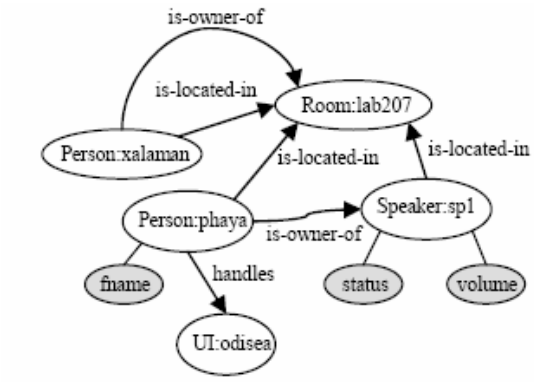


*Figure 1: The blackboard stores information of the environment lab207. It contains two persons (xalaman and phaya) and one speaker (sp1). Xalaman is the owner of the environment and phaya is the owner of the speaker and is using the odisea oral user interface*

The proposed context layer is based in a blackboard architecture with the following characteristics:

- *A common data model*: the information stored in the blackboard follows a common model. The context of the environment is represented through entities defined by a set of properties. It is possible to create relations between the entities, in the form of a graph, to describe the current situation of "the world".
- *A central repository*: the blackboard stores all the contextual information of the environment. This information could consist, for example, on the change on a property (i.e. an opened door) or on a new entity being added or deleted from the blackboard (i.e. somebody entering or living a room).
- *An asynchronous event mechanism:* in addition to accessing directly to the information, the data sources also publish the changes of the context. The subscribed consumers receive those changes from the blackboard, in a proactive way.
- Context information comes from sources of different natures. This information is stored in the blackboard as a graph of entities where each entity is represented according to a common schema. In this manner agents can handle a unified vision of the context, independently of the source and level of abstraction. In Figure 1 a simplified example of a snapshot of the blackboard is shown.

## 3 Conflict resolution mechanism

When an agent pretends to conduct an operation, it sends a command that is received by the context layer. Two operations -*Modify property* and *Add/Erase relation*- result in a change of the state of the entities and in the configuration of the environment. Conflicts take place when several agents use these operations affecting to the same

property or relation. This is particularly problematic for actions implying physical changes in the environment, for example to turn on or off devices, to change the channel of the television or to lower the volume of the loudspeaker.

In order to solve this kind of situations, a queue is provided for each property of an entity and for each relation between two entities. When one of the previously described commands is sent, a priority is assigned to it (see section 4), in addition to other parameters (see section 3.1). Then the command is stored in the corresponding queue.

In every moment, each queue is composed of a set of commands sorted according to its priority, among which the one with greater priority is considered as the "active command". When a new command arrives it is stored in the queue; if its priority is greater than the priority of the active command then it becomes the active one, otherwise it will be stored in the queue. Whenever a command is activated, it is executed once and pertinent changes are applied to the blackboard (and possibly to the physical environment).

In order to avoid the problem of a command with the maximum priority blocking indefinitely the queue, an expiring time is established for each command. This way, once the command has been executed it will remain as the active command until it expires or until another command with greater priority replaces it. For example, a direct order from a user to turn on the light of a room will have greater priority than commands sent by the module of energy saving, but only during some time interval. A command will be executed as many times as it activates, that is to say, as many times as it accesses the first position of the queue without having expired.

A restriction is imposed such that each queue can only contain one command per agent. If an agent sends a second command to a given queue, its previous command is eliminated. On the other hand, agents can annul their commands from a given queue, or from all queues.

Command queues are created and destroyed according to the necessities. When a command arrives, it is checked whether the corresponding queue already exists, otherwise it is created. Oppositely, when a queue contains no commands, it is eliminated. In Figure 2 it is shown the behaviour of a queue when arriving different commands with different priorities and expiring times.

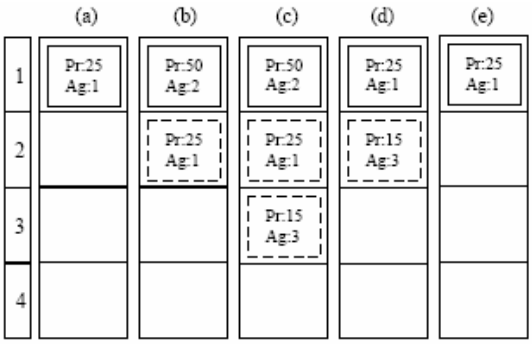


*Figure 2: evolution of a command queue when arriving orders from three different agents (Ag 1, Ag 2 and Ag 3) with different priorities (Pr)*

### 3.1 Parameters of a command

A set of parameters is established for each command. These parameters are:

- *Owner of the command*. It identifies which agent has sent the command.
- *Priority*. A positive integer.
- *Mode*. It determines how the expiring time is calculated (see Table 1).
- *Expiring time*. It indicates from what moment the command is not valid any more. This parameter is only applicable if the mode is "expiring time". Time is measured in milliseconds.
- *Name*. It associates a name to the command. This name allows the agent to refer the command for eliminating it from the queue. Several commands can be grouped under the same name, so they can be erased simultaneously.

The *owner of the command, mode, expiring time* and *name* are established by the agent who sends the command, whereas the *priority* is decided by the context layer based on the priority policies (see section 4.1). Table 1 describes the different modes from which the emitter can choose the expiring time mode of a command.

| MODE | DESCRIPTION |
|---|---|
| *Expiring time* | The expiring time of the order is the value indicated by the *expiring time* parameter. |
| *After executing* | The command disappears from the priority queue once it has been executed. |
| *Instantaneously* | If it has the highest priority, it is immediately executed. Otherwise, it is discarded. |

*Table 1: Different modes to establish the expiration of a command*

## 4 Policies for establishing the priority of the commands

The priority mechanism decides the order in which a set of commands are executed. This order is based on the priority of the commands and their expiration time; the key of the mechanism lies in the policies used to assign these priorities. A policy is a function that, given a command, decides which is the priority that should be assigned to it. There is a default policy, and the possibility of defining specific policies. The administrator of the environment has to decide, for each resource, which is the policy to be applied.

### 4.1 Policies for priority assigning

This section describes the default policy. The design of this policy is influenced by two considerations. On the one hand, the policy must be the sufficiently flexible and generic to include as many cases as possible. On the other hand, the implementation must be simple, so it does not suppose an excessive computational overload.

In order to decide the priority of a given command different variables are considered. These variables can be divided in three groups: (1) those relative to the

emitter of the order; (2) those relative to the command; and (3) those relative to the environment.

### 4.1.1 Variables relative to the emitter of the order

Two variables are included in this group: "who is the emitter of the order" and "which the role of the emitter is". The former can take two values: the command comes from a user (U) or the command comes from an application (A).

On the other hand, the *emitter of the command* can assume different roles depending on the relation with the resource to modify. In this sense the emitter can be owner (O) of the resource, or can be just a guest (G). Both roles are indicated according to the existence or not of a relation in the blackboard of the type *owner-of* between the entities representing the emitter and the resource. In addition, the relation *owner-of* can also be established between an emitter and a complete environment. In this case, the emitter is considered owner of all the resources within that environment.

### 4.1.2 Variables relative to the command

The second group is formed by just one variable indicating the *type of the order*. The range of this variable depends on who is emitting the command. If the emitter is a user, the type of order can be a *direct command (*C) or a *preference of the user (P).* The former represents an order that has been generated as the result of a direct action of the user, like for example, to press a button. The latter represents an order originated automatically after satisfying a condition previously imposed by the user. In case the emitter is an application, two types of orders are defined: *ordinary command(L)* or *security command(H).* Security commands are those coming from the applications in charge of the security of the environment, whereas the ordinary commands correspond to the rest of orders.

### 4.1.3 Variables relative to the environment

Finally, the last group includes two variables relative to the state of the environment. The first one defines the *alert level*, taking as values *normal* or *emergency*. This second value is reserved for serious emergency situations like, for example, a case of fire or flood. The second variable defines the *privacy level*. Three different levels of privacy are defined: *low, normal* and *high*. A room with low privacy level would correspond to a public location, shared by several users; a normal level could be established for private locations, pertaining to a user, where a high level is used for environments with important privacy constraints.

## 4.2 Calculating the priorities

When a command is received, the value of each of its policy variables is recovered. Each possible emitter has a unique identity code that is sent along with the order. This code allows finding out from the context layer blackboard who is the emitter of the order and what type of orders emits. The emitter's role is deduced from the relation between the emitter and the resource being modified. Finally, the security and alert status are obtained from the properties of the environment entity.

These values form a tupla where the first element defines the emitter, the second the role, and the third the type of order. For example, an owner user, having sent a direct order is defined by the tupla (UOC), whereas a command form a guest application would get the tupla (AGL).

The priority will be calculated according to the position that occupies the tupla in a list. The order of the list will be based on the alert status and the security level of the environment. As Table 2 shows, four lists have been defined where the tuplas are ordered from greater to smaller priority.

| | Order of preference | | | |
|---|---|---|---|---|
| | Emerg. | Privacy | | |
| | | Low | Normal | High |
| 1 | UOC | UOC | UOC | AOH |
| 2 | UGC | UGC | AOH | UOC |
| 3 | AOH | UOP | UGC | AOL |
| 4 | AGH | AOH | UOP | UOP |
| 5 | AOL | AOL | AOL | UGC |
| 6 | AGL | UGP | UGP | UGP |
| 7 | UOP | AGH | AGH | AGH |
| 8 | UGP | AGL | AGL | AGL |

*Table 2: Lists of priority-ordered tuplas ordered according to the alert and privacy status*

Once obtained the position occupied by the tupla in the corresponding list, the priority is calculated according to equation 1:

$$Prio = MINPRIO + pos * (MAXPRIO - MINPRIO)/8 \quad (1)$$

Where MAXPRIO and MINPRIO are two constants determining the maximum and minim priorities that can be assigned and pos is the position of the tupla in the list.

The emergency list is used when the environment is in emergency state, independently of the privacy level. In the case of a normal alert state, the list to be used is determined by the privacy level.

As it can be seen in the emergency list, direct orders from the users have maximum priorities, no matter whether they are owners of the resource or not. Next they will be considered the orders from owner applications, followed by those of guest applications. Finally, the orders with smaller priority will be the users' preferences.

For a normal situation in the environment and a low privacy level, the actions of the users, no matter if owners or guests, will have the maximum priority. Next, owners preferences, and commands sent by the applications pertaining to the environment are considered. Finally, guest preferences and commands sent by external applications will be applied.

In case of a normal privacy level, the list of priorities stays identical to the previous one except for the security modules that reach the second position in the list, surpassed solely by owners' direct actions.

Finally, the high privacy level list gives the highest priority to the commands sent by the security modules of the environment, followed by the commands emitted by owner users and rest of applications of the environment. User actions or applications with no relation with the environment get minor priorities.

This mechanism allows to easily change policies, and to quickly adapt the general policy of priorities to new necessities by simply changing the order of the lists.

## 5 Policies of particular priority assignment

It is possible that some of the properties of a resource do not adapt to the mechanism of prioritization previously described. For example if the priority should depend on the value of the property, which is not contemplated in the previous policy. This happens in the case of the volume of a loudspeaker, which must follow some social conventions that are not considered in the default policy. In this case, it makes more sense to use a specific policy to decide which of the possible values is to be set for the volume of the loudspeaker. If two or more agents want to modify the volume, the social norms, generally, determine to set the lower one; especially if we are speaking about user preferences that are triggered automatically. In order to be able to reflect this type of behaviour it is necessary to add a mechanism that allows associating ad-hoc policies to replace the default policy in certain properties.

```
<class name="speaker">
  <property name="Left_Volume">
    <paramSet name="policy">
      <param name="ad-hoc">
        polite_speaker_volume
      </param>
    </paramSet>
  </property>
  <property name="Right_Volume">
    <paramSet name="policy">
      <param name="ad-hoc">
        polite_speaker_volume
      </param>
    </paramSet>
  </property>
</class>
```

*Figure 3: XML definition of the priority policy for a loudspeaker. Two properties are defined: Left_Volume and Right_Volume. There is a policy defined for both, that is an "ad-hoc" policy named polite_speaker_volume. The coding of this policy enforces the volume to get the lowest value of the preferences of the people in the room*

In the example of Figure 3 a language of definition of the blackboard [14] based on XML has been used. This language allows associating to all entities of the same class a particular policy. For doing so a new parameter denominated *ad-hoc* is introduced in the definition of the class. The value of this parameter will be a string with the name of the special policy to be applied to the property.

Three specific policies are currently implemented, but this set can be easily extended as new policies are needed:

- *Loudspeaker volume*. The priority is inversely proportional to the volume requested for the loudspeaker.
- *Best to turn off/close*. If the order is to turn off (or to close) the resource then it gets the maximum priority. For example, it may apply to a TV set.
- *Best to turn on/open*. This would be opposite to the previous case and will benefit those commands turning on or opening the resource. For example, it would be the policy to apply for controlling a door.

## 6 Privacy issues

Command queues deal with conflicts arising when entities try to access a resource simultaneously; collision of contradictory orders is the focus. But there is another type of conflict that appears in Intelligent environments, where users supply personal information pretending to receive some services in exchange. Those are the privacy conflicts, that deal with the problem of how users can establish the compromise between the use of their private information and the services they obtain in return. According to Allan Westin [15] privacy is

*"The right of the individuals to determine by themselves when, how and what private information is disclosed"*

Accordingly with this view, we propose five key elements for our model of privacy:

- *Owner*. Who decides how the privacy of the information is administered. The owner can be a person or a group. In the latter case it is necessary to decide which is the criterion to be applied: unanimity, majority or the most restrictive option.
- *Receiver*. Who receives the right to consult the information. The receiver can be either a person or a software component. An important issue is to establish an authentication mechanism assuring that the correct identification of the receiver.
- *Context*. Another factor of great importance is the context, both of the owner and of the receiver. The location is a relevant contextual variable since the required privacy will vary depending on where the information is being disclosed (for example, in a loudspeaker in a public space). The required privacy will also vary depending on where is the owner. Thus, a person being recorded in video may show different reactions depending on if he is at

home or at a public event. Another important aspect to consider within the context is the people accompanying the receiver. For example, when a user receives a message, the system can decide to send it to her nearest screen, as long as there is nobody around; otherwise a privacy violation may happen.

- The *media* used to spread the information is relevant for the owner of the information, since she can establish different restrictions depending on using a device or another.
- *Use*. This element plays its role once the information has arrived to the receiver. Restrictions in storing, modifying, reproducing or sending it to other users may apply.

In figure 4 we show an example of how the privacy model is added to the entities of the blackboard.

Additionally, all the privacy mechanisms can be applied not only to properties but also to relations. In this way, a private relation will not be shown but to its owner while a public or protected relation will be revealed to the receiver. It is important to emphasize that, since the destination entity of the relation may not have the same privacy constraints as the source one it is possible for the receiver to have access to the source entity and to know the existence of the relation but not to access the information of the destination entity.

```
<entity name= "Roberto" type="person">
  <property name="location">
    <paramSet name="privacy">
      <param name="type">private</param>
    </paramSet>
  </property>
  <property name="mobile_number">
    <paramSet name="privacy">
      <param name="type">protected</param>
        <param name="receivers">Alicia+r</param>
      <param name="use">viewing</param>
    </paramSet>
  </property>
  <paramSet name="privacy">
    <param name="type">public</param>
  </paramSet>
</entity>
```

*Private property*

*Alicia has read privilege for this property, only when Roberto is at home, but she could neither store nor reproduce the mobile number or the conversations*

*The entity is public*

*Figure 4: An entity with privacy parameters*

## 7 Conclusions and future work

This paper proposes a centralised mechanism of priority assignation to solve the conflicts arising when two or more agents try to modify the information managed in the context layer blackboard. When an agent sends a command, it is stored in an

command queue. The proposed mechanism decides which of the commands is executed according to:

- A default policy of assignment of priorities that depends on whom is the emitter of the order, which is the type of the order, what relations exist between the emitter and the resource, what relations exist between the emitter and the environment and what is the current state of the environment.
- A particular policy is defined for each resource that does not adjust to the default policy.

Currently we are considering other aspects such as the time each command has remained in the queue and the number of times an agent has sent a command. With those alternatives, mutual exclusion would be guaranteed and monopolizing of a resource, by agents with an a-priori elevated priority sending a great number of consecutive requests, would be avoided.

On the other hand we have presented a model for integrating privacy management for the different entities that form our active environment. We propose a model based on who want to access the information, where the owner of the information is and which use is pretended for the information. It is shortly expected to count with a demonstrator to begin making tests and obtaining results to check the efficiency of our model.

Resolution of the information is not contemplated yet. Currently, once the access is granted it is so for all the information. In the future it is expected to add some granularity to certain information, bringing the possibility to access the same information with higher or lower resolution. We are also working to add some security policies, complementing privacy.

The proposed approach is currently implemented in a real Ambient Intelligence environment, which is described in [11], and will be validated experimentally in the near future.

**Acknowledgements**

This project is funded by the Spanish Ministry of Science, project number TIN2004-03140.

## References

[1] Weyrich, C. Orientations for Workprogramme 2000 and beyond. ISTAG Report, 1999

[2] Weiser, M. Some computer science issues in Ubiquitous Computing. Communications of ACM, 36(7) pp. 75-84, 1993.

[3] Schilit, B. et al. Context-Aware Computing Applications, IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[4] Cook, D. and Das, S. Smart Environments Technologies, Protocols and Applications, Wiley-Interscience, 2005.

[5] Kagal, L. et al. A Policy Language for a Pervasive Computing Environment. IEEE 4th Int. Workshop on Policies for Distributed Systems and Networks. 2004.

[6] Rattapoom T. Security and Privacy in the Intelligent Room, Master's Thesis for MIT, 2002.

[7] Martin, D. et al. A framework for building distributed software systems, 13(1&2), pp. 91-128, 1999.

[8] Minar, N. et al. Hive: Distributed agents for networking things. In Proceedings of ASA/MA'99, 1999.

[9] Gajos, K. Rascal - a Resource Manager For Multi Agent Systems In Smart Spaces, CEEMAS, 2001

[10] Arnold, K. et al. The Jini Specification, Addison-Wesley, 1999.

[11] Haya, P. et al. A prototype of a context-based architecture for intelligent home environments, CoopIS, 2004.

[12] Alamán, X. et al. Using context information to generate dynamic user interfaces, HCI International, 2003.

[13] Montoro, G. et al. Spoken interaction in intelligent environments: a working system, Advances in Pervasive Computing, Eds. Austrian Computer Society (OCG), 2004.

[14] Haya, P. et al. Extending an XML environment definition language for spoken dialogue and web-based interfaces, AVI, 2004.

[15] Westin, A. Privacy and Freedom. New York, NY, Atheneum. 1967.