Towards the development of Ambient Intelligence Environments using Aspect-Oriented techniques[*]

L. Fuentes, D. Jiménez, M. Pinto
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga
Boulevard Louis Pasteur, 35 29071 Málaga (SPAIN)
Email: {lff, priego, pinto}@lcc.uma.es

Abstract

Nowadays the interest in Ambient Intelligence Environments has grown considerably due to new challenges posed by the evolution of society requirements to more friendly environments. Ambient Intelligence technology is not fully developed and integrated in everyday life, but a lot of organisations are interested in it. On the other hand, Aspect Oriented Software Development is considered a growing technology that improves the modularity and adaptability of complex large-scale systems. Then, our goal is the development of an Ambient Intelligence Aspect-Oriented platform adapting DAOP, a component and aspect platform.

1. INTRODUCTION

Today computational devices are being used in hundreds of human activities, ranging from office work, industrial systems and domotic systems. Hardware and computational technologies have evolved to satisfy the services demanded from these activities, like phone communications or games. This evolution has been in parallel with the development of new kinds of devices. These new devices like the last generation of mobiles and PDAs, pose high CPU and storage capacity requirements demanded to perform the new services. Examining carefully the current state-of-art we can see that the future of all these new technologies is the convergence to the Ambient Intelligence concept or AmI.

The origins of the AmI are first found in 1991, when Mark Weiser wrote an article about Ubiquitous Computing [1]. For Weiser, the ubiquitous computing term is the opposed to virtual reality. Where virtual reality puts people inside a computer-generated world, the ubiquitous computing forces the computer to live in the real world. The Ubiquitous Computing is then referred as the capacity of integrating autonomous computational devices in the real world. The devices are able to extract data from the real world that surrounds them and perform some data processing in order to obtain results that can be used afterwards by others devices. The main characteristic of the ubiquitous computing is that minimises the computer device intrusion in real world. Ideally, the human beings in an AmI environment will not notice the devices.

As an example of this type of interactions we can consider the communication between a humidity sensor device and a garden irrigation device. The former device will inform the later one about humidity variations, and the irrigation device will take actions to preserve the environment humidity according to the user preferences. Another example that we will examine more carefully in this paper is the application of AmI technologies to a car equipped with several devices that communicate among themselves and provide some useful services like user identification, local traffic information services, road obstacle detection sensors or car diagnostic services.

Weiser shows the need of finding new ways to obtain a better integration of the information technology in everyday life activities. He postulates that this integration must include the people social behaviour and the technology accessibility as main concepts. The proposed concepts are very general, but the main idea from his work is that devices will need to be adapted to people and that this will only be possible by developing much more natural interfaces between human beings and machines (voice, hand gesture, etc.). But in spite of the efforts the project was not successful, mainly because in that date, the available technology does not meet the requirements necessary to support the proposed ideas.

In the last few years, Weiser's ideas were retaken when the new technologies like mobile and wireless networks started to evolve. In consequence, the AmI concept has been adopted after several meetings of the European ISTAG [2] (*Information Societies Technology Advisory Group*) and encompasses a broader vision of the ubiquitous computing idea proposed by Weiser. The meetings started in 2001 and the goals were to promote and extend the use of ubiquitous computing technologies in the 6th European Community Programme for Research and Technological Development.

The initial meeting ended with the creation of several documents for the IPTS (*Institute for Prospective Technology Studies*). The documents can be found at http://www.cordis.lu/ist/istag.htm. The ISTAG said that the examples proposed in those documents would be technologically viable for year 2010. In the documents, they have also pointing out the critical development areas of technology and the main fields of application (genomic, biotechnology, information society technology, nano-technology, nano-science, aeronautic and space, food production security, sustainable development and economic and politics sciences).

[*] This research was funded by the CICYT under grant TIC2002-04309-C02-02.

In addition to the advances in hardware and computational technologies AmI should take advantage of the new software development technologies that have emerged in the last years. The importance of applying advanced software technology have been made manifest after seeing the difficulties of developing concrete AmI oriented projects like Aura [12] or Oxygen [13]. In this sense, we think that AmI applications are good candidates to be modelled using Aspect-oriented Software Development techniques or AOSD [3]. Since AmI environments are dynamic and are characterized by the runtime changes of interactions among users and devices, presenting strong requirements of dynamic adaptability, they are good candidates to benefit from aspect separation techniques.

In AOSD, aspects are defined as properties of an application that cut across some or all the application objects or components [11]. The AOSD tries to identify, isolate and extract these properties from the application core functionality, modelling these properties as aspects. Aspects can evolve independently from component functionality, so applications become more modular and, in addition, we can reuse those aspects in others similar applications. Moreover, we can replace the number and type of aspects that are applied to an application without modifying the application code. We think that identifying and separating aspects we reduce the complexity of the evolution management of AmI applications both at design and at runtime. In this paper we will try to identify the most relevant aspects that are found when decomposing an AmI application in software components and how DAOP, a component and aspect platform can help us in the application development process.

After this introduction we will show in section 2 the most relevant aspects we found in AmI applications. In section 3, we will show our proposal towards the definition of a platform for AmI applications and finally, we will expose our conclusions and future work.

2. ASPECTS IN AMBIENT INTELLIGENCE ENVIRONMENTS

Before starting to explain which aspects we have identified in AmI environments, we will provide a brief summary of the characteristics that any AmI application must show and the problems that arise when we develop these applications. The three main characteristics are:

Ubiquitous Computing, that is the ability of providing computational capabilities to any device everywhere in a non-intrusive way. These devices range in size from a board to a simple tag. There are three problems that we face when developing Ubiquitous Computing applications. First, the limited amount of energy [7] that those devices have available to function. Second, the low computing power that those devices provide and finally, the limited device storage capability known as the “nomadic data” problem. See [5] and [6]. This last problem appears due to the limited storage capacity of devices that make impossible store and retrieve all the information generated by them from everywhere.

Ubiquitous Communication, that is the ability to communicate among them any kind of device. When we try to implement this characteristic, we face several different problems. The first one is the device and communication protocol heterogeneity. This heterogeneity prevents good communication interoperatibility between devices. The second problem is the dynamic nature of these environments. The applications are executed like being part of a large distributed application and the coordination between them is difficult. It requires a solid distributed network system, a homogeneous interchange information format, a communication coordination system, a dynamic aware location mechanism and a homogeneous way to achieve the heterogeneous devices interconnection. Another problem is the scalability in a distributed network system. When the system is crowded of devices, the communications channels become saturated and interferences and errors in communications start to rise. When this happens, the need of scalable adaptation strategies is a must. Finally, the last problem encountered is the communication security and privacy. The special nature of wireless communications makes them vulnerable to intrusions or data interception problems. We must assure the privacy of confidential and sensible data by encryption or others procedures.

Natural Interfaces. The main goal of Weiser was the integration of devices in a human world. We must develop new ways of human-device interaction. To achieve this, we must solve two problems. Non–intrusive hardware, the hardware devices must be easily integrated into everyday object and become “invisible” to people. So, people do not have to care about how to interact with it. Natural interfaces, the devices must provide alternative human interfaces like voice or hand gesture recognition. While non-natural ways of communication like keyboard or mouse interfaces must tend to disappear.

Ambient Intelligence relevant Aspects

After describing the development problems that AmI applications must face, our next goal is to identify the main properties common to most AmI applications. Usually, these properties are independent of the functionality of different devices and therefore, it is a good approach to model them as aspects. Now, we are going to describe the most important properties that we have found:

Access Control. Most part of AmI services define restrictions on which devices can access to them. For example, it will be a really bad idea to allow a child to open his parent door. The access control property applied to the AmI service will handle which actions can be done and which information will be available by defining a control access list for hardware or software components. This is a typical property that should be modelled as an aspect to allow the replacement of the access control mechanism without affecting the application code.

Authentication. All devices and users in the environment must have a unique identifier. The authentication service is provided by most part of non-trivial applications in AmI environments. The device or user authentication can be performed using a username and a password, a digital signature, a user voice recognition mechanism, a digital certificate or any other identification method. We can model this property as an aspect and select the adequate method to achieve the user or device authentication.

Awareness. The AmI environment is constantly changing, new services and applications appear and others disappear without warning. The awareness property will be responsible of notifying the changes in the state of devices. This property comes from the Collaborative Virtual Environments [4], but it is also applicable to AmI environments. We should select carefully the kind and amount of information transmitted because we must not flood the receiver with useless information. The awareness notifications are captured by the environment devices and the data retrieved keeps their environment perception updated. We think that this would be modelled as an aspect because detaching this code from device core functionality allows us to manage different levels of information that is sent or received and even modify this information before it reaches the target. This aspect is crucial in AmI environments due to their dynamic nature and the necessity of most devices in the environment to be aware of changes.

Coordination. Occasionally it is interesting to perform certain specials operations when several circumstances are met in the environment. For example, when an event is sent indicating that a user has opened an AmI car door, we probably need to coordinate several AmI car components to react to this event. A coordination aspect can be modelled to handle the event and send adequate messages to the involved components. It is interesting to model this as an aspect because in AmI environments this kind of interaction is very common.

Communication. AmI environments support heterogeneous devices with different communications protocols and data interchange structure. A communication aspect is useful to act as a bridge to interconnect different devices that normally cannot communicate. If we model this property as an aspect it would be possible to adapt the device communication protocol at runtime to accept communications from others devices.

Encryption. Security in AmI environments is a must, because all the communications are open and thus easy to intercept and alter. We require a sophisticated mechanism to adapt the communication trust necessities at any time. The encryption property decouples the encryption security system from devices. Modelling this property as an aspect permits the replacement of the encryption model without affecting the application. Another possible aspect use is when the device is not capable of providing this encryption mechanism due to processing or storage limitations and the aspect can redirect the task to other specialised devices in the environment.

Error handling and recovery. Applications in this moving and dynamic environment encompasses a high error rate, usually due to interferences, communication transmission errors, communication channel saturation or application unavailability. This aspect helps to achieve a better performance in this field providing specifics solutions considering the services and the devices available in the environment. For example, imagine that we are in a congress registration hall totally automated using AmI automatic registration devices. If one or several of this devices are out of service, an error handling aspect can select other working registration device from the environment and start the login process without reporting the error to the user.

Language Internationalisation. In an AmI environment it is possible that not all users understand the natural language of AmI interfaces or the voice messages of certain devices. For example, an English user probably does not understand a Japanese character display or message. The applications should be able to adapt automatically the interface language to the user preferences. This property can be easily modelled as an aspect since crosscuts several components, and it is used to adapt a component to a user profile.

Persistence and Nomadic Data or Pervasive Data [5]. Due to the fact that the information used by a device can be distributed in different locations, we must provide a mechanism to store it efficiently and the possibility of migrating this information to different devices while the user is moving. Thus, this property can be modelled as an aspect. Another use of this aspect is to remotely store information for devices that cannot locally store some information due to storage restrictions.

To finish this section, we are going to show an example of a simple AmI application modelled with aspects. Suppose that we have a car that is equipped with AmI technology devices. Each device executes one or more specialised programs. The car has a navigation computer, a device connected to a traffic station and some other utility devices like proximity sensor, speech detection devices, car engine diagnosis device, air conditioner, etc.

In the example, we are going to focus in devices that operate the car doors and windows. These devices are controlled by an AmI application modelled as a software component called "*Car Door Component*" as shown in Figure 1. In an AmI world there is no need of keys for cars. The user will be recognised by his voice when he approaches and orders the car to open the door. The authentication aspect will perform the user recognition task before executing the open door command. The authentication aspect is useful to avoid that unauthorised people open the door. If the user is correctly identified, the device will continue the order execution. After the authentication aspect has been applied, the car door device will open the door and finally the awareness aspect will be evaluated as shown in Figure 1. This aspect will deliver an event that contains information about who has open the door. This event is broadcasted to the environment so that all the components interested in it can catch and evaluate it. For example the navigation computer component after receiving the event executes a personalised greeting or adapt the seat to the user stored preferences, or notifying a traffic station about a new car in this street.

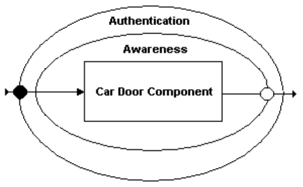

Figure 1 Aspects and Car Door Component.

We justify the use of aspects to be able to change the different authentication or awareness, methods without changing the application code. For instance, we can use digital signature or eye recognition instead of voice recognition for the authentication aspect. We can also (re) use the defined authentication and awareness aspects in others devices, like a car navigation device, a device that controls the driver seat preferences or an air conditioner device.

To finish this section we will highlight several questions. In our model, aspects can be executed in sequential or parallel mode. We can also define if an aspect will be applied before of after the component has processed a message. In the example we have executed the two aspects sequentially, authentication before the device command execution and awareness after completing the command execution. If any of the aspects fail during the evaluation, the execution process stops and an exception is raised notifying the problem. Finally the last issue is that most of the proposed aspects like authentication, access control awareness or encryption can be developed using any aspect platform or aspect oriented language (for example AspectJ [14]).

3. ASPECT-ORIENTED AMBIENT INTELLIGENCE PLATFORM

The CAM/DAOP platform has been designed by our group to support the development of component and aspect based distributed applications. We have successfully developed a Java/RMI implementation of the platform and several collaborative applications [4]. This platform defines components and aspects as the application building blocks and performs the weaving process at runtime [8]. You can find additional information about how DAOP performs the dynamic weaving between components and aspects in [8] and [9]. The language we use to specify a kind of component and aspect composition is performed using an Architecture Description Language called DAOP-ADL [10]. This language uses XML to explicitly describe the architecture of an application that can be modified by DAOP at runtime. This is a powerful feature to reconfigure systems such as AmI environments. Our actual efforts are oriented to adapt the DAOP platform to the AmI requirements.

The DAOPAmI platform

Following the Weiser's vision [1], in a typical AmI environment there are hundreds of devices. This leads us to a heterogeneous environment populated of devices with very different capabilities and requirements. The current DAOP platform kernel implementation exceeds the storage and processing capabilities of many of the AmI typical devices and the communication and security requirements of AmI applications. So, we present a new platform, named DAOPAmI that extends the DAOP platform capabilities and the DAOP-ADL language to support the specific needs of AmI applications. Now we are going to explain the modifications performed upon the DAOP platform to adapt it to the AmI applications requirements.

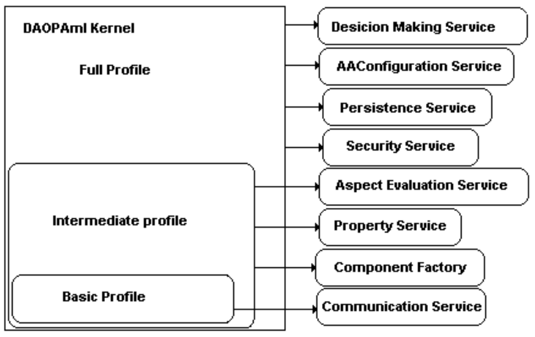

Figure 2 DAOPAmI kernel Architecture.

In [1] Weiser proposed three different devices categories according to its capabilities (taps, tabs and boards). We follow this standard division in our architecture classifying the devices in three profiles (which are similar to J2ME [15] profiles). Figure 2 shows the services that each profile supports. Now, we are going to characterize each one.

```
<component role="temperature">
....
  <delegatedComponent>
    <roleInstance>temperature1</roleInstance>
    <address >    150.214.108.46:1234</address>
  </delegatedComponent>
</component>
```

Figure 3 Component delegation definition.

Basic profile, this profile covers the simplest devices like sensors, identification cards, calendars or calculators, corresponding to Weiser's taps. As is shown in Figure 2, devices must implement at least the communication service that allows them to send asynchronous or synchronous messages to other devices. Currently, we have re-implemented this service using the J2ME communication API due to the low storage and processing capabilities of these devices. Additionally, they delegate the rest of platform functionality to other devices using a new service delegation mechanism provided by the DAOPAmI platform. Notice that devices not supporting the minimum DAOPAmI platform functionality are modelled as components inside other larger devices.

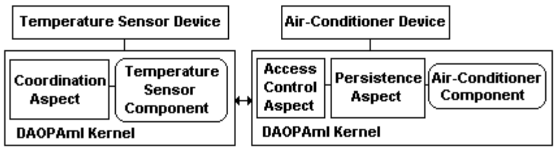

Figure 4 DAOPAmI Air-Conditioner device configuration.

DAOP uses role names to identify and address components and aspects. So we extend the component definition in DAOP-ADL with the physical address of the device modelled by this component. We also add a role instance name to identify individual components playing the same role (see Figure 3). Likewise, in the device side we specify the address of the software component that will receive the device output messages.

Intermediate profile, this profile comprises most of the typical AmI devices that have a medium computational and storage capacity like PDAs, mobile phones or Laptops corresponding to Weiser's tabs. They implement the basic profile functionality and additionally, as shown in Figure 2, the component factory service to manage software components. The property service that resolves some component and aspect data dependencies and finally the aspect evaluation service that manages the component-aspect weaving mechanism.

Full profile, this profile implements all the platform functionality including the intermediate profile and several additional services like the decision-making service, that performs automatic actions based on the environment and a set of logical rules. This service is very valuable to model flexible coordination aspects. The AAConfiguration service, that serves to configure applications and runtime. A persistence service to store temporal and persistent application data and finally a security service that guarantees the data and communications privacy. These devices have large amounts of storage and processing capabilities like advanced desktop computers and enterprise servers. This kind of devices are equivalent to Weiser`s boards.

In the current CAM/DAOP platform implementation, when an application starts, it retrieves the application architecture configuration from local storage or from an application repository. But, what happens if the device does not implement any storage facilities or cannot access directly to an application architecture repository. In this case the DAOPAmI kernel can be configured to retrieve this information from other kernel, using the communication service. After retrieving the application architecture data the application execution starts normally. But a second problem arises. What about if the application demands some services that the device cannot support due to resource or hardware restrictions? In this case, the kernel can be configured to communicate with other kernels delegating the service execution using the delegation mechanism.

Delegation mechanism example

Suppose that we have an air-conditioner device and a temperature sensor located inside a car equipped with AmI technology. The temperature sensor and the air-conditioner devices both support an intermediate kernel, as is shown in Figure 4. The first one is modelled as a temperature sensor component and a coordination aspect that is applied to the output messages of this component. The component takes periodic heat measures and emits an event containing this information. The coordination aspect evaluates this event and distribute it to others components like the air-conditioner. The air-conditioner device is modelled by a component that manages the physical device plus two aspects. The access control aspect ensures that the incoming messages come from valid device sources and the persistence aspect stores a list of received messages.

But, what happens if the sensor device cannot support this configuration due to kernel memory constrains? To solve this, the temperature sensor device delegates the services execution to other kernel. In this new scenario, the temperature sensor application architecture definition has been modified, as explained before, to delegate all kernel services, except communication service, to the kernel executed in the sensor array device shown in Figure 5. The temperature sensor components and aspect are executed within the sensor array kernel, and all events and messages sent to the temperature sensor component are redirected to the sensor array kernel to be evaluated.

An example: an AmI car

Following the previous car example, now we are going to show a more complete car door configuration using several of the previously identified aspects. As shows Figure 6, when a user approaches to the car and order "*open the door*", an authentication aspect will be executed in order to determine the user identity before the voice reaches the speech recognition component. After identifying the user voice as a valid one, the speech recognition component

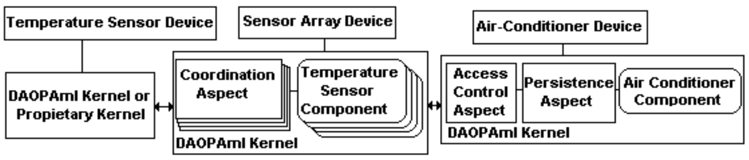

Figure 5 Temperature sensor service delegation.

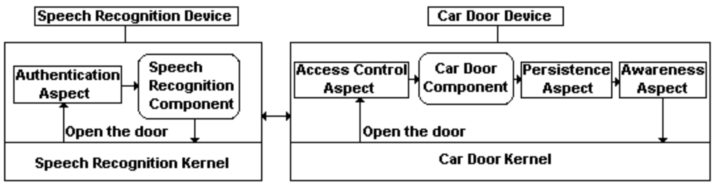

Figure 6 Extended Car Door device kernel configuration.

processes the voice signal and sends a message to the car door component notifying about the command. Afterwards, the car door kernel receives the message and the access control aspect is executed verifying that the user has the right to perform the action. After the verification, the component executes the requested command and then, the persistence aspect is applied to save an activity record. Finally the awareness aspect is executed, which broadcasts information about the performed action to the environment that can be used by other components.

This example shows how the DAOPAmI kernel provides support to AmI application development using an AOSD approach. The use of aspects let us handle the AmI applications dynamic behaviour in a natural way and easily adapt it to unexpected situations. For example we can adapt the previous example to prevent that children can open the car door from inside. To achieve this, we add a new aspect before the access control aspect that retrieves the user location and profile including for example the user age from the AmI environment. With this information the new aspect determines if the command execution can proceed or not. Thus, we have modified completely the application behaviour without changing the code of existing components; we have just added a new aspect to the application architecture definition.

4. CONCLUSIONS AND FUTURE WORK

All ideas proposed in this article are a first approach to determine the architecture requirements to develop AmI applications and define the basic services that are needed. The next step will be extending the CAM/DAOP platform to support the new kernel services and the DAOP-ADL language to express the new requirements. We think that delegation mechanism is an improvement that solves problems related to resource-constrained AmI applications in the DAOP platform and proves that our AOSD approach is feasible.

We think that AmI technology will be very important and will affect not only the way we see the software development but also the society and how people interact with computers and technology in general in the near future. We also think that the AOSD technology can help to produce more configurable and easy to manage AmI environments. AmI technology is still in its first stages and we need to do a lot of work before reaching the Weiser Ideas [1]. Our proposal tries to mix the best of both AmI and AOSD technologies by adapting an existing component-aspect dynamic platform. We think that this eases the AmI application development process and encourage the aspect reuse.

5. REFERENCES

[1] Weiser, M., “The computer for the Twenty-First Century”, Scientific American 165, 1991, p. 94-104.

[2] Information Societies Technology Advisory Group. http://www.cordis.lu/ist/istag-reports.htm

[3] Aspect-Oriented Software Development http://www.aosd.net

[4] Pinto, M., Amor, M., Fuentes, L., Troya, J.M., ”Collaborative Virtual Environment Development: An Aspect-Oriented Approach”, Proceedings of DDMA'01, 2001.

[5] J. Kubiatowicz et al, “OceanStore: An Architecture for Global-Scale Persistent Storage”. Proceedings of the ASPLOS November 2000.

[6] The Coda File System. http://www.coda.cs.cmu.edu/

[7] Goldsmith A.J, “Design Challenges For Energy-Constrained Ad Hoc Wireless Networks”, IEEE Wireless Communications, August 2002.

[8] Pinto M., Fuentes L., Fayad, M.E., Troya, J.M., “Separation of Coordination in a Dynamic Aspect-Oriented Framework”, Proceedings of AOSD’02, April, 2002.

[9] Pinto M., Fuentes L., Fayad, M.E., Troya, J.M., “Towards an aspect-oriented framework in the design of collaborative virtual environments”, Proceedings of FTDCS’01 workshop, November, 2001.

[10] Pinto, M., Fuentes, L., Troya, J.M., “DAOP-ADL: an architecture description language for dynamic component and aspect-based development”, Proceedings of GPCE 2003. pp 118-137, Erfut, Germany 2003.

[11] Kiczales, et Al., “Aspect Oriented Programming”. Proceedings of ECOOP’97.

[12] Toward Distraction-Free Pervasive Computing. Project Aura. IEEE Pervasive Computing 2002. http://www-2.cs.cmu.edu/~aura/

[13] MIT Oxigen Project http://oxygen.lcs.mit.edu/

[14] AspectJ http://eclipse.org/aspectj/

[15] Java 2 Platform Micro Edition. http://java.sun.com/j2me/