An Aspect-oriented Middleware for Adaptation of Pervasive Systems

Abdelkrim Benamar [1], Noureddine Belkhatir [2] and Fethi Tarik Bendimerad [3]

**[1] Computer Science Department, University of Abou Bekr Belkaid,
Tlemcen, 13000, Algeria**

**[2] Computer Science Department, University of Pierre Mendes,
Grenoble, 38041, France**

**[3] Telecommunication Department, University of Abou Bekr Belkaid
Tlemcen, 13000, Algeria**

Abstract
The emerging pervasive platforms have to address several challenges such as mobility, multimodality, context-awareness and content adaptation. However, implementing adaptation using conventional development techniques is challenging as adaptation requirements tend to affect multiple elements of a pervasive environment. Moreover, a feature model for pervasive applications will help both to deploy various configurations of the middleware tailored to each device. But, several crosscutting variable features and dependencies between features are commonly found in pervasive computing. To address this problem we propose an aspect-oriented middleware platform, able to deal with the high dynamic issue of pervasive systems. In this paper, we present the key architectural concepts of our platform, and we provide implementation details of typical use case.

Keywords: *Pervasive computing, Adaptation, Middleware, Aspect-oriented applications.*

1. Introduction

As the advancement of embedded and communication technologies, growing quantities of computational units are attached to our surroundings or even to ourselves. Due to the increasing amount of computational resources and scarcity of human controls and supervisions, a new paradigm of computer utilization, known as pervasive or ubiquitous computing, was proposed by Weiser [22]. The pervasive computing has been considered as the third era of computing after main-frame and personal computing. In the scenario of pervasive computing, computational units and information processing activities are embedded in all surrounding and everyday devices, functioning invisibly. Human users are able to access information from more than one specific device and most likely from different physical locations, rather than engaged with one computer. According to this scenario, human users may not necessarily be aware of the existence of embedded devices and computations occurred behind the scene.

Therefore one of the challenges of pervasive computing is the definition of advance mechanisms that support the deployment and the tailored-configuration of pervasive applications through various kinds of devices with different capacities and characteristics. Another challenge is the provision of mechanisms that allow the runtime reconfiguration of pervasive applications for dealing with context changes. This means that a pervasive application have to deal with static and dynamic changes, so its architecture should be well modularized to facilitate its adaptation to the evolution of devices and environment.

A Software Product Line (SPL) approach would be very useful to express the different requirements of devices in terms of commonality and variability of a middleware platforms family. Normally, a middleware platform provides all the common services most used by distributed applications. But in pervasive applications resource constraint is an important limitation, so only a specific middleware platform configuration that fits the device characteristics must be installed. Feature modeling analyzes commonality and variability from a domain perspective [14]. Then, a feature model allows specifying where is the variability in an independent way to the core asset, and enables reasoning about all the different possible configurations.

A feature model for pervasive applications will help to specify different configurations of middleware according to existing devices profiles. This will allow the deployment of different versions of middleware tailored to the resource constraints of small devices and appliances. In addition, the explicit existence of a feature model specification will also help to support the dynamic reconfiguration of a middleware platform according to changing contexts, or to runtime variations of device resources. On the other hand, SPL promotes the notion of architecture centric software engineering [4]. Then, a feature model can be used as input for generating an architectural representation of a product line. Likewise most of current distributed applications, several crosscutting variable features and dependencies between features are commonly found in pervasive computing. Some of these crosscutting variables features are security, context-aware and fault-tolerance and so on. If we design these crosscutting features in a traditional way we will drastically reduce reusability, adaptability and the evolution of the product line. In order to solve this problem of crosscutting variables features and dependencies between features we will use an Aspect-Oriented (AO) approach. Aspect-Oriented Software Development (AOSD) promotes the separation of concerns at every stage of the software lifecycle, from requirements and architectural design (early aspects) to implementation. Then, using this approach we can define the middleware architecture in a more cohesive and decouple way, alleviating the reconfiguration and the evolution tasks. In principle, AO middleware is defined as a middleware platform that supports the execution of AO applications. On the other hand, the internal platform architecture of such middleware is not full AO. Some efforts have been done towards the definition of a truly AO middleware platform architecture, but only some specific middleware concerns are separated. In this paper we will focus on the feature model definition and we also present a use case middleware implementation.

The remainder of this paper is organized as follows. In Section 2 we present an overview of pervasive computing issues, SPL and feature models and concepts of AO middleware. Section 3 describes the design principles and outlines the feature model of our middleware. Section 4 provides the key architectural aspects and section 5 shows how they have proven to be viable in current implementations of actual use case. Section 6 sketches out main related work while section 7 concludes the paper and identifies directions for future work.

2. Backgrounds

2.1 Pervasive computing

To stress the challenge that comes with pervasive systems, we consider a typical ubiquitous scenario where one student plays also the role of administrator of an online student forum regarding main events in the campus. She is interested in news concerning her campus and accesses the web via a smartphone which is both General Packet Radio Service (GPRS) and IEEE 802.11/Wireless Fidelity (WiFi) enabled. While exploiting GPRS, she prefers having an imageless version of the news, but when she switches to the campus WiFi connection, which is fast and costless, she wants to get full web pages, while keeping on working in a seamless way. In both cases, content layout must be adapted to fit the smartphone display she uses and news must be formatted according to her forum style sheets. Besides, as she often drives to the campus, she wants to learn about news also via phone calls: when this happens, she authenticates by spelling a secret password and a synthesized voice reads to her the only news that matches her customized preferences. The above and similar ubiquity scenarios stress many currently debated research fields, such as mobility support and context awareness, multimodality and multi-channel access to content, content aggregation and service composition, variously interconnected.

Mobility needs are usually grouped into three categories: user, terminal and service mobility [2]. User mobility allows users to have a uniform and consistent view of their specific working environment (user preferences and/or service requirements) independent of their current location. Terminal mobility allows devices to move and (re)connect to different networks while remaining reachable and keeping communication sessions consistent. Resource mobility allows resources to move across different locations and still remain available independent of their physical location and the position of their clients.

Context-awareness refers to the capability of leveraging conditions of the user herself and of her surrounding physical and computational environment to provide more relevant or brand new services. To provide a brief example, a printing service could leverage user position and identity to choose the nearest printer to which she is authorized to send a document to. Though location is certainly a common and pre-eminent piece of context, there is much more to context than position and identity.

Services can also exploit time notion, knowledge of device capabilities and user preferences to process requests in the most suitable way, as well as discover and interact with other computing-capable objects or query the environment features for available pieces of information. For instance, web browser requests usually convey device software capabilities by declaring the client software version, the available support for graphical formats and so on. An exhaustive definition of context could be actually illusive, since it can in general refer to every known piece of information about user and environment. Thus, the most reasonable solution for obtaining ubiquitous context is probably to assemble context information from a combination of context related services [1], possibly deployed both at client side and at network infrastructure or server side.

Device heterogeneity calls for multimodal interfaces and content adaptation: users often need to access some unique content or application via different user interfaces and they also need to obtain information presented according to their preferences or device features. As an example, a user may request a service by using one of different input modes, such as keyboard, hand-writing or speech recognition, gestures and so on. In response, she could get different corresponding output formats, such as a text-only document, an image or a vocal reading. We refer to multi-channel access as the ability of providing a same service or information content through different media channels and platforms.

2.2 Adaptation of distributed software systems

Adaptation proved, along the years, to be a major issue towards the development of dependable distributed software systems. In principle, we may distinguish three basic types of adaptation situations based on the targeted needs [15]. First, we have corrective adaptation that aims at dealing with faults causing failures in the constituents of a system. Second, we have perfective adaptation that targets changes performed towards meeting the evolving functional and non-functional requirements of the system. Finally, we have adaptive reconfiguration aiming at the proper functioning of devices and their hosted applications that are dynamically integrated in a computing system without prior knowledge of the functional constraints (e.g., available functionalities and resources) imposed by this system. The first two types of adaptation were typically targeted by stationary distributed systems. On the other hand, the need for the last type of adaptation arose with the latest emergence of pervasive computing systems. An in between evolution with respect to these two system domains were nomadic computing systems, which added wide area mobility to stationary distributed systems and were a precursor to pervasive computing systems. There, mobility makes the computing environment less predictable than in stationary systems, thus as well implying the need for adaptive reconfiguration, to a lesser extent, however, than in pervasive systems.

Adaptation in stationary distributed systems – architecturally modeled in terms of components and connectors [9] – concerns adding, removing or substituting components or connectors. Changes should take place at runtime to avoid compromising the availability of the overall system.

Being one step further, pervasive computing systems aim at making computational power available everywhere. Mobile and stationary devices will dynamically connect and coordinate to seamlessly help people in accomplishing their tasks. For this vision to become reality, systems must adapt themselves with respect to the constantly changing conditions of the pervasive environment: (i) the highly dynamic character of the computing and networking environment due to the intense use of the wireless medium and the mobility of the users; (ii) the resource constraints of mobile devices, e.g., in terms of CPU, memory and battery power; and (iii) the high heterogeneity of integrated technologies in terms of networks, devices and software infrastructures.

2.3 Software Product Lines and Feature Models

Product Line Software Engineering (PLSE) has the ability to exploit commonality and manage variability among products from a domain perspective [14]. In the feature-oriented approach, commonalities and variabilities are analyzed in terms of features. A feature is any prominent and distinctive concept or characteristic that is visible to various stakeholders [14]. The features can be organized into a feature model that represents all possible products of a software product line. Feature modeling analyzes commonality and variability from a domain perspective. Commonalities are modeled as mandatory features and variabilities are modeled as variable features which are classified as alternative or optional features.

In features diagrams three kinds of relationships are found: compose of (when the feature is composed of several sub-features), generalization or specialization (when the feature is a generalization of the sub-features), and implemented by (when the sub-feature is needed to implement the feature). Furthermore, for each variable feature, feature dependency analysis can identify dependencies between features. Examples of such dependencies are the mutual dependency and mutual exclusion relationships. Finally the features can be classified in four layers: capability, operating environment, domain technology and implementation technique. The capability layer is composed by the user visible characteristics, such us services, operations, non-functional characteristics and so on. The operating environment layer contains the environment in which the application is used. The domain technology layer is the way to implements the capability layer elements. Finally, in the implementation techniques layer are the techniques used to implements the capability layer elements.

2.4 Middleware for aspect-oriented applications

Many proposals claim to provide an AO middleware level, are really not fully AO platforms for developing AO applications. Our proposal tries to bring the aspect-oriented benefits to the middleware itself, in order to deal with heterogeneity of devices and communication technologies and the high dynamism of pervasive environments. Considering this approach, the final application will be just another platform component. This means, that the specific functionality of final pervasive applications could be developed using a non AO language.

This will encourage developers of pervasive applications, usually non experts in aspect-orientation, and more accustomed to low level programming to use a middleware. In an AO approach the middleware can be considered as a collection of aspects with many of them being completely standard (formerly known as common services such as persistence or security). Usually in non-AO middleware the list of common services offered by a platform is fixed, and is not possible neither extend nor customize them by end users. The aspects provide us the facility of add, remove or change an architectural element without modifying their rest of the architectural. In AO middleware the end user may add new services in a non-invasive way, by means of adding a new aspect, for adapting the platform for example to a new application domain. So, in an AO middleware is permitted to define extensions of an AO middleware that can be highly proprietary, developed by end users. Let’s consider the definition of a custom authentication. Using an AO approach end users can decide whether to use or not a security aspect depending on the execution environment. This can be realized as simple as, for secure environments (e.g. inside an Intranet) the security aspect is not considered as part of the aspect composition rules, but for insecure environments yes. In the latter case, the first user access will be intercepted and an authentication aspect will be evaluated after message delivery. Even more, since the security aspects are modeled separately from application components, it is easier to change their implementation than to use a password authentication.

3. Design principles

Most current middleware solutions adopt a TINA-like [5] architectural model and focus on enriching middleware with dedicated features that services and users in turn exploit to face mobile, multimodal and/or ubiquitous challenges. But when the number of functionalities increases and they need to interact with each other, middleware complexity inevitably grows, making this approach inadequate for facing the wide domain of ubiquity support. In our opinion, the only viable approach for dealing with such increasing complexity is simplifying middleware design by leaving it only the core of management and coordination functions, and by moving ubiquity feature logic outside the middleware layer. As a result, the middleware still adopts a TINA model because of its clear distinction of users, services/contents accessed by users, and middleware, but we simplify middleware logic by applying a pattern of delegation: we introduce entities that are responsible for realizing mobility and multimodality logic, leaving the middleware only coordination responsibility.

3.1 Users

In current multimodal and multi-channel scenarios, users access services and contents by means of a plethora of different hardware devices and client software interfaces. Many factors drive the rapid emergence of these needs among users. Technological innovation provides users with new types of wireless connectivity and new portable devices that allow them to communicate and work in much more effective ways.

Social reasons are also playing an important role: accessibility themes as well as part of research in the multimodal field aim at crossing the digital gap that separates impaired users from traditional web contents and services. In our vision, a truly ubiquitous platform should encompass and serve every kind of device and/or user interface, from a mere, largely diffused web browser to, say, a specific audio interface for visually impaired users.

3.2 Services

In our model, any kind of content-related action constitutes a service: from generation to content adaptation, from enrichment to delivery. Most of these operations are no longer available as predefined in middleware functions, but are accessible as third-party services. Services are characterized not only by their type and description, but also by their semantic behavior and by their binding requirements, so as to emphasize composition with one another. Thus, new services can be requested and deployed at any time, preventing the need to know them all at middleware start-up.

3.3 Middleware

Middleware is the most common solution that is widely used to facilitate interoperability and coordination in the presence of dynamism and heterogeneity. Research and developmental work in the area of middleware for distributed systems has been in progress for several years, both in academia and industry [3], worldwide. Over the last few years, however, several research groups across the world have been engaged in building and experimenting middleware solutions to address the challenging scenario of pervasive computing environments. In same context, we propose to follow the guidelines presented in [14] to design a correct feature model of middleware platform. Globally, the middleware will follow a microkernel and services structure. The microkernel term describes a form of operating system design in which the amount of code that must be executed in privileged mode is maintained to an absolute minimum [10]. As a consequence, the rest of services are built as independent modules that are plugged and executed by the kernel. In this way, we obtain a more modular and reusable system. Furthermore, we distinguish between mandatory base services and the rest of optional extra services that will be added according to applications requirements.

Then, as is shown in Fig. 1, the middleware feature is composed by two mandatory features, the microkernel and the services that can be basic or extra services. The basic services (e.g., lookup, discovery, communication, device manager and fault tolerance) are also a mandatory feature but the extra services (e.g., security, persistence, error handling, login, context-aware, position location…) are an optional feature. All these features are in the capability layer. This layer contains all the user visible characteristics. In our case the user of the middleware will be the applications, and then in this layer we found all the microkernel details and all the services. The middleware can be used in many kinds of devices (e.g., PDA, smartphone) and allow to use several communication types (e.g., wifi, bluetooth).

Moreover, the operation environment layer that represents the application environment provides these two features (device type and communication type). The device type feature will have many alternatives sub-features depending on the kind of devices. The same thing happens with the communication type feature. The domain technology used to implement the middleware is an aspect-oriented microkernel structure. On the other hand, for each service we will use its own domain technology, but for sake of simplicity we only draw two generic features, one for the base services and other for the extra services. Finally, the middleware is implemented through Aspect-Oriented Programming (AOP) technique. Then, the implementation technique layer supports the AOP feature.

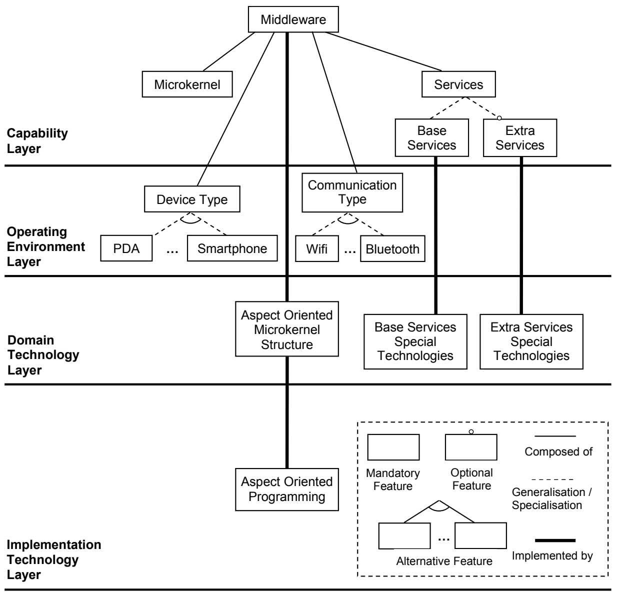


Fig. 1 Middleware Feature Model.

Fig. 2 shows the microkernel detailed feature models. The microkernel is composed by the container, factory, context manager and service manager sub-features. The container is the responsible to search the required services. The factory element creates and instantiates such services. The context manager has to be aware of all the context changes. We have found four entities that provoke context changes.

– Application restrictions: an application may vary and require different services that were required before of its variation (i.e., an application needs to be secure only in some moments, and then the optional security service will be active only in these moments)
– Environment properties: an environment can change and some services may not be executed in the same way before (i.e., if one device is missing)
– Device constrains: a pervasive system has to be aware of device constraints (i.e., if the battery is less than a low value maybe all the graphics in the application must be changed by text)
– User preferences: to process user requests in the most suitable way (e.g., let a user interested by accessing the web via a smartphone which is both GPRS and WiFi enabled. While exploiting GPRS, he prefers having an imageless version, but when he switches to the WiFi connection, he wants to get full web pages)

The last sub-feature of the microkernel is the service manager that manages the architecture of the middleware and runs the application. It has two sub-features, the architectural manager and the weaver. The last is an interpreter that composes all the elements of the middleware and application. Since the middleware uses

AOP approach we call this interpreter weaver. The architectural manager is composed by the architectural description and the selector. The architectural description is composed by two sub-features, the middleware PLA (Product Line Architecture) description and one instance of this product line. The selector is a tool that produces a particular product from a product line, taken account the context properties (device constraints, user preferences…etc).

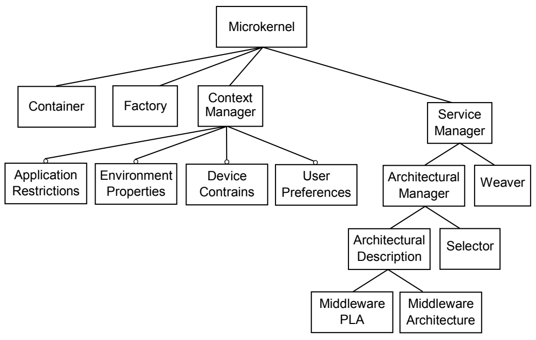


Fig. 2 Microkernel feature model.

4. Middleware architecture

As is shown in the previous section the middleware architecture will be composed by two components (the application itself and the microkernel) and several aspectual components (all the services). Likewise most of traditional middleware platforms architecture, we follow a layer approach (as shown in Fig. 3). The application level is in the top and the middleware itself is just after. The first sub-layer of the middleware is the application services like security, context awareness, error-handling, and so on. These services use the base services that are in the next layer, such as communication and lookup. Below these layers is the microkernel (as shown in the feature model description) which is composed by the container, the factory, the context manager and the service manager.
The product line architectural description of the middleware is place inside of the service manager. The microkernel, through the context manager, has to know the context properties. With this information and with the architectural description of the application, the service manager runs the automatic selector facility in order to instantiate a particular middleware architecture of the product line architecture. This is a static configuration of the middleware. Using the context properties, the factory wills instantiate the specific selected services. In order to run the application the service manager has the weaver that is the responsible of make the composition between aspectual components, corresponding to the services, and the components (microkernel and application). If a change is made in the context (user properties, device constraints ...) the microkernel will use the selector again and a new particular middleware of the product line will be instantiate. The responsible to perform this is the extra service context-aware that has access to the context manager. Then, this is the dynamic reconfiguration of the middleware.

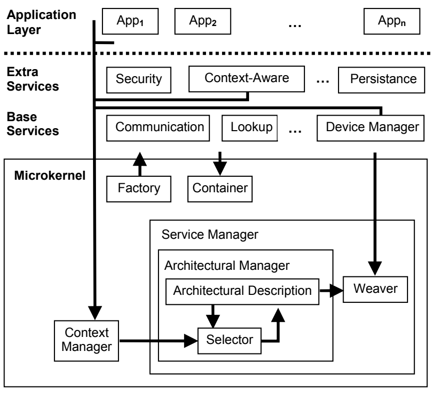


Fig. 3 Middleware architecture.

5. Middleware implementation

To illustrate the application of middleware solution, we present our experience with using AO middleware to modularize adaptation in a pervasive environment. Specifically, this application involves two pervasive devices (e.g., PDA, smartphone) to be deployed at strategic public locations across Tlemcen university campus. Furthermore, the prototype is aimed at supporting a range of applications including, but not limited to, displaying news, disseminating information on upcoming events and assisting visitors (and also staff and students) in navigating their way around campus. Visitors often need to find their way to various destinations around campus. The destination can be a physical location such as a building, department or auditorium or it can be an event such as a conference being hosted in a particular building. Furthermore, each new display added to the environment must adapt its specific properties to those of the environment.

5.1 Implementation details

When modularizing adaptation we need to address three specific facets of adaptation within our pervasive environment. The first two facets are application independent and relate to any application deployed in the environment while the third facet is specific to the navigation application:

- Display management: As the environment expands more displays will be incorporated into it. All new displays must have their specific properties adapted for use within the pervasive environment.

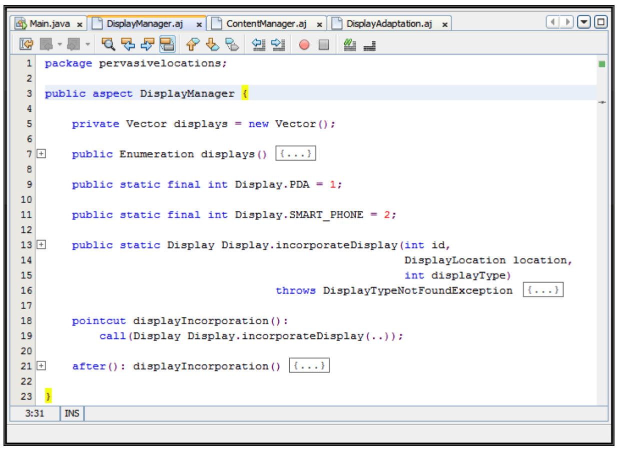
- Content management: The navigation content (an arrow in this case) is only one type of content to be displayed on the devices. There are other types of content that also need to be delivered to the devices. Furthermore, as new displays are added, the content already being displayed within the environment has to be made available on them as well.

- Display adaptation: As a new destination is added or an existing destination changed (e.g., change of venue for an event), the displays need to be adapted to guide the users to the correct destination. Furthermore, if a display is moved to a different location it should be adapted to display the content in a correct fashion based on its new location.

5.1.1 Display manager aspect

The `DisplayManager` aspect (as depicted in Fig. 4) encapsulates all functionality relating to incorporation of new displays or adaptation of their properties to the pervasive environment. The aspect maintains a collection of all displays incorporated into the environment and has a public method to traverse the collection (as shown in lines 5-7 of Fig. 4). This is useful for other elements of the system, especially the `ContentManager` aspect, which needs to access all the displays in the system from time to time as new content becomes available.

The `DisplayIncorporation` pointcut (as shown in lines 18-19 of Fig. 4) captures all calls to the static method introduced into the *Display* class. An *after* advice then adds the incorporated display to the display collections in the aspect as well as adapts the properties of the newly incorporated display to the pervasive environment.
Note that although the *DisplayManager* aspect affects only a single class, nevertheless it encapsulates a coherent concern. This use of an aspect is, therefore, very much in line with good separation of concerns practice.



```
1  package pervasivelocations;
2
3  public aspect DisplayManager {
4
5      private Vector displays = new Vector();
6
7      public Enumeration displays() {...}
8
9      public static final int Display.PDA = 1;
10
11     public static final int Display.SMART_PHONE = 2;
12
13     public static Display Display.incorporateDisplay(int id,
14                                                      DisplayLocation location,
15                                                      int displayType)
16                                   throws DisplayTypeNotFoundException {...}
17
18     pointcut displayIncorporation():
19         call(Display Display.incorporateDisplay(..));
20
21     after(): displayIncorporation() {...}
22
23 }
```

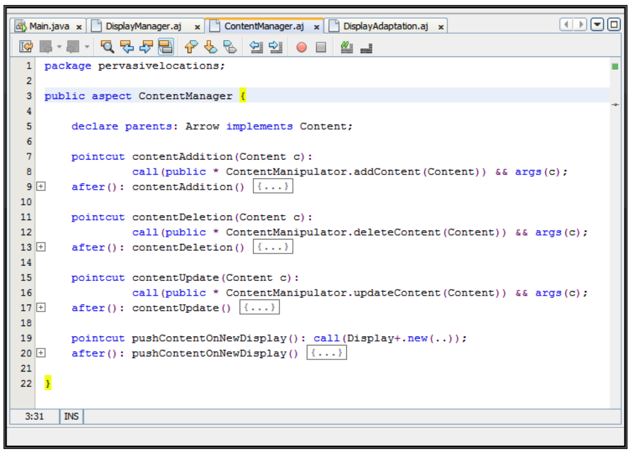
Fig. 4 Display manager aspect.

5.1.2 Content manager aspect

The `ContentManager` aspect is depicted in Fig. 5. It declares all types of content and must implement the *Content* interface (as shown in line 5 of Fig. 5). Note that in this case there is only one type of content, *Arrow*, shown but in practice the pervasive environment displays a variety of content. The *Content* interface provides an application independent point of reference for the pointcuts within the aspect, hence decoupling content management from the type of content being managed. Any classes that manipulate content in the pervasive applications deployed in the environment are required to implement the `ContentManipulator` interface, which specifies a number of methods for content addition, removal and update. Like the `Content` interface, the `ContentManipulator` interface also provides an application-independent point of reference to capture all content manipulation behavior within the applications in the environment, including the navigation application.

The `contentAddition` pointcut (as shown in lines 7-9 of Fig. 5) traps calls to `addContent` methods in all application classes manipulating content. An *after* advice for the pointcut then traverses all the displays registered with the `DisplayManager` and updates them with the new content. The `contentDeletion` and `contentUpdate` pointcuts (as shown in lines 11-17 of Fig. 5) and their associated advice perform similar functions upon content deletion and update.

The `pushContentOnNewDisplay` pointcut (as shown in lines 19-20 of Fig. 5) captures the instantiation of all sub-classes of the `Display` class. An *after* advice then pushes the available content onto the newly instantiated display.

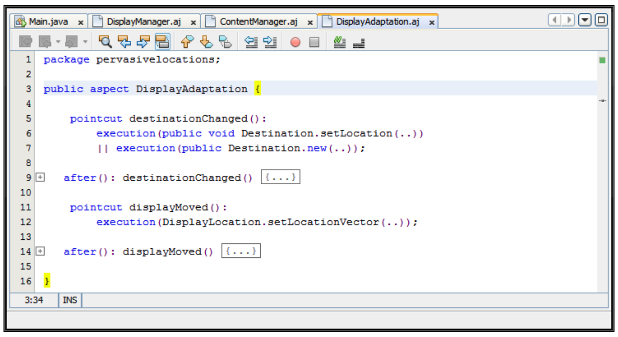


```
package pervasivelocations;

public aspect ContentManager {

    declare parents: Arrow implements Content;

    pointcut contentAddition(Content c):
            call(public * ContentManipulator.addContent(Content)) && args(c);
    after(): contentAddition() {...}

    pointcut contentDeletion(Content c):
            call(public * ContentManipulator.deleteContent(Content)) && args(c);
    after(): contentDeletion() {...}

    pointcut contentUpdate(Content c):
            call(public * ContentManipulator.updateContent(Content)) && args(c);
    after(): contentUpdate() {...}

    pointcut pushContentOnNewDisplay(): call(Display+.new(..));
    after(): pushContentOnNewDisplay() {...}

}
```

Fig. 5 Content manager aspect.

5.1.3 Display adaptation aspect

While the `DisplayManager` and `ContentManager` aspects are application independent and handle adaptation facets that span across applications in the pervasive environment, the `DisplayAdaptation` aspect, depicted in Fig. 6, is specific to the navigation application. The `destinationChanged` pointcut in this aspect (as shown in lines 5-9 of Fig. 6) captures the change in location of an existing destination or the creation of a new destination. An *after* advice for the pointcut invokes the adaptation rules for the displays to adapt the content accordingly. The `displayMoved` pointcut (as shown in lines 11-14 of Fig. 6) identifies that a display has been moved by capturing the change in its location vector. An associated *after* advice then proceeds to adapt the content of the moved display and any neighboring displays accordingly.

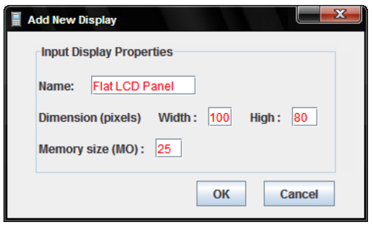


```
1  package pervasivelocations;
2
3  public aspect DisplayAdaptation {
4
5      pointcut destinationChanged():
6          execution(public void Destination.setLocation(..))
7          || execution(public Destination.new(..));
8
9     after(): destinationChanged() {...}
10
11     pointcut displayMoved():
12         execution(DisplayLocation.setLocationVector(..));
13
14    after(): displayMoved() {...}
15
16 }
```

Fig. 6 Display adaptation aspect.

5.2 Execution scenario

As mentioned earlier, our prototype involves only two types of displays (e.g., PDA, smartphone). Due to the high modularity of applications based AOP techniques each new display can be easily added to the environment (as shown in Fig. 7) and its specific properties are adapted to those of the environment.



Input Display Properties

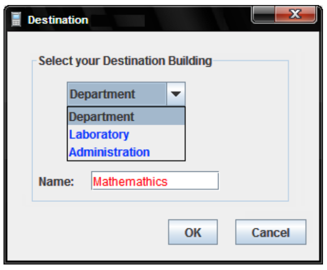
Name: Flat LCD Panel

Dimension (pixels) Width : 100 High : 80

Memory size (MO) : 25

Fig. 7 Adding a new display Flat LCD Panel.

Recall that our prototype is aimed at assisting users in navigating their way around campus. The destination can be either physical location such as department (as shown in Fig. 8.a) or event such as a conference (as shown in Fig. 8.b).

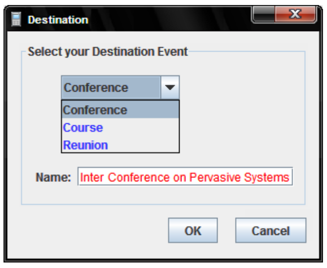


Select your Destination Building

Department

Name: Mathemathics

a. Physical location



Select your Destination Event

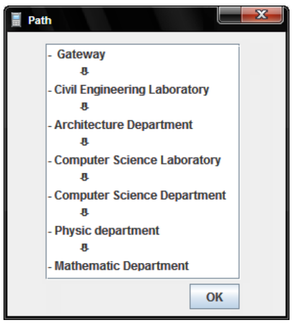
Conference

Name: Inter Conference on Pervasive Systems

b. Event

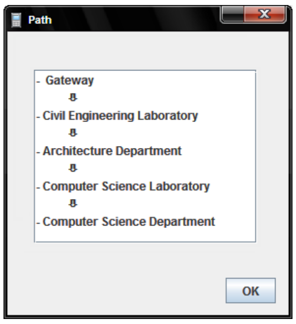
Fig. 8 Selection of destination.

Furthermore, the corresponding paths to the physical location or event (hosted in computer science department) are respectively shown in Fig. 9.a and Fig. 9.b.



- Gateway
- Civil Engineering Laboratory
- Architecture Department
- Computer Science Laboratory
- Computer Science Department
- Physic department
- Mathematic Department

a. Physical location



- Gateway
- Civil Engineering Laboratory
- Architecture Department
- Computer Science Laboratory
- Computer Science Department

b. Event

Fig. 9 Proposition of path.

5.3 Discussion

The three aspects used in this paper (e.g., `DisplayManager`, `ContentManager` and `DisplayAdaptation`) clearly demonstrate that AOP constructs provide an effective means to modularize both application independent and application specific facets of adaptation in a pervasive environment. The use of aspects makes it easier to not only adapt the environment to changes in content but also makes it possible to react to the reorganization of the displays in an effective fashion. Furthermore, any changes to the adaptation characteristics of the environment or the navigation application are localized within the aspects hence avoiding changes to multiple elements of the system that would have otherwise been required.

There are also interesting observations to be made about the design of the adaptation concern. Firstly, the use of `Content` and `ContentManipulator` as application independent points of reference makes it possible to decouple the `ContentManager` from application-specific content and content manipulation operations.

Moreover, this technique allows to decouple the persistence concern from application-specific data. On the other hand, we can observe that the notion of one large aspect (or one in any other AOP technique) modularizing a crosscutting concern does not make sense in the case of the adaptation aspect either. The three aspects and the `Content` and `ContentManipulator` interfaces together modularize adaptation (as shown in Fig. 10). While different classes and aspects modularize specific facets of the adaptation concern, it is the framework binding them together that, in fact, aspectises this particular crosscutting concern.

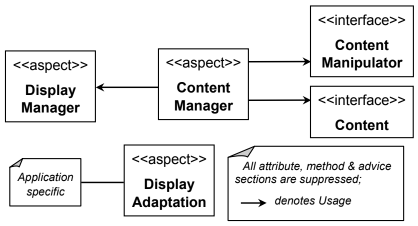

<<interface>> Content Manipulator
<<aspect>> Display Manager
<<aspect>> Content Manager
<<interface>> Content
Application specific
<<aspect>> Display Adaptation
All attribute, method & advice sections are suppressed;
denotes Usage

Fig. 10 Framework modularizing Adaptation.

6. Related works

Efforts in supporting ubiquity greatly concentrate now on developing services and middleware platform logic. Current ubiquitous and mobile environments obey some standards and specifications, presenting solutions for specific ubiquity aspects. Research follows some main directions, mostly concerned with design guidelines and feature standardization, middleware proposals and the idea of providing toolkits to create 'ubiquity-enabled' applications. However, these research directions tend to evolve separately from each other, focusing on particular problems or goals, and they altogether lack a unified approach to ubiquity support.

CoBrA (Context Broker Architecture) is a pervasive context-aware computing infrastructure that supports ubiquitous agents, services and devices, to behave intelligently, according to their situational contexts [11]. It is a broker-centric agent architecture, that provides knowledge sharing, context reasoning, and privacy protection support for ubiquitous context-aware systems, using a collection of ontologies, called COBRA-ONT, for modeling the context in an intelligent meeting room environment. These ontologies expressed in the Web Ontology Language (OWL), define typical concepts associated with places, agents, and events and are mapped to the emerging consensus ontologies that are relevant to the development of smart spaces.

GAIA is an infrastructure for smart spaces, which are ubiquitous computing environments that encompass physical spaces [18]. Ontologies are introduced in this system, as an efficient way to manage the diversity and complexity of describing resources, that is, devices and services. They work as a system specification for configuration management, by providing a standard taxonomy of the different kinds of entities, including applications, services, devices, users, and data sources. Therefore, these ontologies are beneficial for semantic discovery, matchmaking, interoperability between entities, and interaction between human users and computers. Additionally, the GAIA ontologies are used to make GAIA systems context-aware. They model contextual information, including physical, environmental, personal, social, application, and system contexts. They describe the relations between different entities and establish axioms and constraints on the properties of the entities that must be satisfied.

CoCA (Collaborative Context-Aware) is a collaborative, domain independent, context-aware middleware platform, which can be used for context-aware application development in ubiquitous computing [7]. It is an architecture for context-aware services, focused on context reasoning in ubiquitous computing environments, using semantic ontology and collaborative approaches. This model uses an ontology for modeling and management of context semantics and a relational database schema for modeling and management of context data. These two elements are linked through the semantic relations built in the ontology.

In [20] ontology-based approach is proposed for modeling ubiquitous computing applications employs ontology merging and alignment to capture and store in an ontology repository information needed by task-based activity spheres in order to represent their environment, goals, states, events and available resources. The dynamic behavior of the sphere is represented as a series of merged ontologies. Every ontology snapshot provides a static description of the sphere, each time it undergoes asynchronously a state transition. The state transitions can then be modeled based on Discrete Event Systems (DES) and the dynamic behavior can be controlled by using Supervisory Control Theory (SCT). Each activity sphere has a dual hypostasis: one concerning how this sphere conceives itself and the other one concerning how it is perceived by other activity spheres. Each activity sphere is described by its embedded ontological objects (i.e. ontologies and alignments). The ontological representation of activity spheres depends on their motivators (as a creator or observer) point of view. Ontologies are used for the semantic description of the components that are involved in the achievement of an activity, and alignments describe semantically their context. In this sense, ontologies and ontology alignments are the foundations of an activity sphere, which are used to deal with semantic interoperability, dynamic nature, context awareness and adaptive services.

PURPLE [13] is an adaptive, context-aware and component-based middleware for ubiquitous computing. The use of EFL (lightweight, efficient and reflective component platform running on Linux) offers the technique in multi-level reflection, which provides the advantages of flexibility in implementing and extension. Specifically, PURPLE authors discussed the context-awareness mechanism for system-level reconfiguration using strategy control and the working process in adaptation and system consistency control. Based on the testing result, they conclude that the overhead by introducing reflection is small enough to be negligible. Furthermore, the work does not address a number of key issues within ubiquitous computing domain, such as the support for mobility, agent-based programming paradigm.
CAST (Context-Awareness Simulation Toolkit), is a toolkit which allow to acquire, express and safely use the context information of ubiquitous computing environment [12]. CAST generates users and devices in a virtual home domain, designates their relation and acquires virtual context information. The created context information is reused by the request of application and put into use for context learning. Particularly, CAST gives a consideration to security in the process of context acquisition and its consumption. That is, with applying SSCM(SPKI/SDSI Certificate Manager) to test if the created context information was valid information and if the application that called for the context had legitimate authority to do so. CAST not only captures virtual context information, but it also guarantees the safe sharing of the context information requested by the application.

Scooby [17] provides an effective method for combining services to meet the needs of users in a variety of pervasive computing scenarios. Scooby uses a specifically designed composition language, which separates coding of services as building blocks from their composition into useful applications which respond to available resources. The Scooby language and middleware provide a rich system for matching advertised services with requirements, while using many familiar techniques from languages such as Java. Both the advertised and required characteristics of those services can be dynamic, so that the inevitable changes in capability and requirements that flow from pervasive computing scenarios can be accommodated. This approach lends itself to the development of end-user configuration tools, as explored in the NatHab project [16], where programming language APIs and exposure to source code present a significant obstacle to most users. The (re-)binding of services is an important part of the middleware, and the Scooby system provides for bindings which take account of both types and characteristics of the service in order to dynamically respond to the requirements of the application and the visible facilities. A content-based event communications model was employed, to support this approach.

In [21], an ambient intelligence (AmI) platform is proposed to facilitate fast integration of different control algorithms, device networks and user interfaces. This platform defines the overall hardware/software architecture and communication standards. It consists of four layers, namely the ubiquitous environment, middleware, multi-agent system and application layer. The multi-agent system is implemented using Java Agent DEvelopment (JADE) framework and allows users to incorporate multiple control algorithms as agents for managing different tasks. The Universal Plug and Play (UPnP) device discovery protocol is used as a middleware, which isolates the multi-agent system and physical ubiquitous environment while providing a standard communication channel between the two. An XML content language has been designed to provide standard communication between various user interfaces and the multi-agent system. A mobile ubiquitous setup box is designed to allow fast construction of ubiquitous environments in any physical space. The real time performance analysis shows the potential of the proposed AmI platform to be used in real-life AmI applications.

MyAds [6] is system capable of exploiting pervasive technologies to autonomously adapt the contents to the evolution of the interests among an audience, and validated the ideas behind it through experimental execution of a prototype platform on a test-bed of distributed machines. Results confirm the advantages of the use of audience-sensitive advertisement techniques, whereby the impact of the advertisement can be optimized towards a bigger part of the audience. Furthermore, they show that a proper exploitation of the auction-based allocation paradigm leads to an enhancement of the economic efficiency at both sides of the transaction, in particular at advertiser's side where the ratio between the cost of the investment and the final impact results is optimized in a way to increase the segment of the interested audience while reducing the sustained cost per person.

SARA [9] is a resource and location aware framework to support the large-scale deployment of heterogeneous applications in ubiquitous environments. The basic tenet of SARA revolves around the concepts of *virtual registries*, *most likely residence* and the *location vector*. Object registration and discovery are achieved by hashing the *object id* to obtain the physical co-ordinates of a *point (P)* within the network service area. The set of mobile nodes in the virtual registry containing *P* assume the responsibility of registering and maintaining information about the object. The basic design principle for SARA scheme is to use *geographical mapping* for the hashing as opposed to node mapping since nodes are mobile. SARA is scalable, since the hash function is uniform and this ensures that the information stored in the network is spread across the network. The simulation results showed that SARA was robust and scalable and performed particularly well as the density of nodes in the network increased. However, SARA fails to face some ubiquity challenges, as security needs to be incorporated into its service architecture at various levels. Also, node location information stored in the network must be protected to guarantee user privacy. The schemes developed to ensure user privacy must also take into account the resource constrained environment.
In [19] a resource optimized quality assured context mediation framework based on efficient context-aware data fusion and semantic-based context delivery. In this framework, contexts are first fused by an active fusion technique based on Dynamic Bayesian Networks and ontology, and further mediated using a composable ontological rule-based model with the involvement of users or application developers. The fused context data are then organized into an ontology-based semantic network together with the associated ontologies in order to facilitate efficient context delivery. Experimental results using SunSPOT, which is a small, wireless, battery powered experimental platform, demonstrate the promise of this approach.

7. Conclusion

In this paper we have shown the troubles of the pervasive systems and how a middleware approach can solve them. We have followed the layer structure for the middleware, with a microkernel and several services in order to get a more reusable and adaptable middleware. The adoption of a basic idea of decoupling middleware from services and users led to a simple and lightweight platform and granted our solution a twofold goal. On the one hand, middleware provides a unified approach to ubiquity support and addresses the issues of mobility, multimodality and content-adaptation in service provisioning; on the other hand, it can dynamically and automatically react to reconfigure services on a user context basis. Furthermore, we have proposed to use an AO middleware that allows dynamic reconfiguration, because the reconfiguration is an issue very important for context-aware pervasive system.

Mainly, in this paper we have focus in the design of the feature model of the middleware product family, that after it will be used to produce the middleware architectural product line. Then, also we have shown the architecture of the middleware following the layer approach. After, having the product line architecture we will start the implementation of the middleware, beginning with the microkernel and after with the services. During all this process new services of the middleware product line can appear as needed, then the product line feature model will have modified. We have completely implemented and deployed a use cases to verify the viability and effectiveness of our approach.

Encouraging results are calling for further research activities. We are currently working on dynamic composition of some services modeled as aspects and driven by an AO Domain Specific Language (DSL). This DSL will be an extension of an AO Architecture Description Language (ADL) with variability. Moreover, we plan to use AO-ADL to describe a real system, and we will discuss the main benefits of the symmetric decomposition model and the extension of AO-ADL connector with aspectual binding informatio(n.

References

[1] G. D. Abowd, and E. D. Mynatt, “Charting Past, Present, and Future Research in Ubiquitous Computing”, ACM Transactions on Computer Human Interaction, Vol. 7, No. 1, 2000.

[2] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli, “Dynamic Binding in Mobile Applications”, IEEE Internet Computing, Vol. 7, No. 2, 2003, pp. 34-42.

[3] A. Benamar, N. Belkhatir, and F. T. Bendimerad, “Proposition of Generic Deployment Platform for Component based Applications, Journal of Software Engineering, Vol. 2, No. 1, 2008, pp. 23-38.

[4] J. Bosh, Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, Addison Wesley 2000.

[5] M. Chapman, and S. Montesi, “Overall Concepts and Principles of TINA”, 1995 http://www.tinac.com/specificati-ons/documents/overall.pdf

[6] A. Di Ferdinando, A. Rosi, R. Lent, A. Manzalini, and F. Zambonelli, “MyAds: A System for Adaptive Pervasive Advertisements”, Pervasive and Mobile Computing. Vol. 5, 2009, pp. 385-401.

[7] D. Ejigu, M. Scuturici, and L. Brunie, “CoCA, A Collaborative Context-aware Service Platform for Pervasive Computing”, in 4th IEEE International Conference on Information Technology, pp. 297-302, Las Vegas, 2007.

[8] D. Garlan, and M. Shaw, “An Introduction to Software Architecture”, Technical Report, CMU-CS-94-166, Carnegie Mellon University, 1994.

[9] A. Gopalan, and T. Znati, “SARA: a Service Architecture for Resource aware Ubiquitous Environments”, Pervasive and Mobile Computing, Vol. 6, 2010, pp. 1-20.

[10] P. Greenwood, *et al.*, “AOSD reference architecture”, AOSD-Europe-ULANC-37, 2008.

[11] L. Kagal, V. Korolev, Chen, H. Joshi, A., and T. Finin, “Centaurus: A Framework for Intelligent Services in a Mobile Environment”, in 21st IEEE International Conference on Distributed Computing Systems, pp. 195-201, Washington 2001.

[12] I. S. Kim, Y.L. Lee, and H. H Lee, “CAST Middleware: Security Middleware of Context-awareness Simulation Toolkit for Ubiquitous Computing Research Environment”, in Huang, D.S. (eds.), ICIC 2006, LNCS, Vol. 344, pp. 506-513, Springer, Heidelberg.

[13] Z. Kuo, W. Yanni, Z. Zhenkun, W. Xiaoge, C. Yu, A Component-based Reflective Middleware Approach to Context-aware Adaptive Systems” in Lowe, D., Gaedke, M. (eds.), ICWE 2005, LNCS, Vol. 3579, pp. 429-434, Springer, Heidelberg.

[14] K. Lee, C. K. Kyo, L. Jaejcon, “Concepts and Guidelines of Feature Modeling for Product Line Software Engineering”, LNCS, Springer-Verlag, vol. 2319, 2002, pp. 62-77.

[15] P. Oreizy, N. Medvidovic, and R.N. Taylor, “Architecture-based Runtime Software Evolution”, in ACM International Conference on Software Engineering, Kyoto, 1998, pp. 177-186.

[16] T. Owen, I. Wakeman, B. Keller, J. Weeds, and D. Weir, “Managing the Policies of non-Technical Users in a Dynamic World”, in 6th IEEE International Workshop on Policies for Distributed Systems and Networks, Stockholm, 2005, pp. 251-254.

[17] J. Robinson, I. Wakeman, and D. Chalmers, “Composing Software Services in the Pervasive Computing Environment: Languages or APIs”, Pervasive and Mobile Computing. Vol. 4, 2008, pp. 481-505.

[18] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, “GAIA: A Middleware Infrastructure to Enable Active Spaces”. IEEE Pervasive Computing, Vol. 1, No 4, 2002, pp. 74-83.

[19] N. Roy, T. Gu, S.K. Das, “Supporting Pervasive Computing Applications with Active Context Fusion and Semantic Context Delivery”, Pervasive and Mobile Computing, Vol. 6, 2010, pp. 21-42.

[20] L. Seremeti, C. Goumopoulos, and A. Kameas, “Ontology-based Modeling of Dynamic Ubiquitous Computing Applications as Evolving Activity Spheres”, Pervasive and Mobile Computing, Vol. 5, 2009, pp. 574-591.

[21] K. I. K. Wang, W. H. Abdulla, and, Z. Salcic, “Ambient Intelligence Platform using Multi-agent System and Mobile Ubiquitous Hardware”, Pervasive and Mobile Computing, Vol. 5, 2009, pp. 558-573.

[22] M. Weiser, “Computer for the 21st Century”. Scientific American, Vol. 265, No 3, 1991, pp. 94-100.

Abdelkrim Benamar is currently a postdoctoral research fellow in Mobile and Pervasive Computing group in Computer Science Department at the University of Tlemcen, Algeria. He received his Ph.D. degrees in Computer Science from the University of Tlemcen in 2007. He received his MS and B.E. degree from Oran University, Algeria, in 1999 and 1992, respectively. His research interests include context-aware resource management in mobile, pervasive and grid computing environment.

Noureddine Belkhatir received his Ms degree in Computer Science from the University of Algiers, Algeria, in 1973. From the 1985 onwards he has been working in the Computer Science Department at the University of Grenoble, France. Now he is full professor in Computer Science and has been Chairman of the above Department from 2005 to 2010. He took part to several European and national projects on distributed computing and massively parallel architectures. He has published about 70 papers on national and international journals and on Conference Proceedings.

Fethi Tarik Bendimerad is a University Professor of Engineering and the Founding Director of the Telecommunication Laboratory at the University of Tlemcen, Algeria. His current research interests include wireless sensor networks, mobile and pervasive computing, design and modeling of smart environments, pervasive security, resource and mobility management in wireless networks, and mobile grid computing.