AMBIENCES: ON-THE-FLY USAGE OF AVAILABLE RESOURCES THROUGH PERSONAL DEVICES

Beatriz Otero and Marisa Gil

Computer Architecture Department, Universitat Politècnica de Cataluña
Barcelona-TECH

ABSTRACT

In smart spaces such as smart homes, computation is embedded everywhere: in toys, appliances, or the home's infrastructure. Most of these devices provide a pool of available resources which the user can take advantage, interacting and creating a friendly environment. The inherent composability of these systems and other unique characteristics such as low-cost energy, simplicity in module programming, and even their small size, make them a suitable candidate for dynamic and adaptive ambient systems. This research work focuses on what is defined as an "ambience", a space with a user-defined set of computational devices. A smart-home is modeled as a collection of ambiences, where every ambience is capable of providing a pool of available resources to the user. In turn, the user is supposed to carry one or several personal devices able to interact with the ambiences, taking advantage of his inherent mobility. In this way, the whole system can benefit from resources discovered in the spatial proximity. A software architecture is designed, which is based on the implementation of low-cost algorithms able to detect and update the system when changes in an ambience occur. Ambience middleware implementation works in a wide range of architectures and OSs, while showing a negligible overhead in the time to perform the basic output operations.

KEYWORDS

Pervasive system; smart space; resource mobility; middleware; runtime management

1. INTRODUCTION

The development of data-centric and user-centric applications is growing and becoming increasingly common, to the extent that we find computational devices everywhere [1]. Smart spaces such as smart cities [2][3] and domotics are well-known examples of pervasive systems, also known as ubiquitous systems. However, Weiser's vision of pervasive computing as "invisible" computing, where people would not be aware of tools, and would have easy technological access, has not been accomplished yet. Nowadays "Internet of things" work again reveals a complex and powerful development, but far removed from the simplicity that Weiser's predicted.

These pervasive platforms tend to be based on a set of distributed cooperating components that share resources, thereby eliminating their inherent resource constraints. From the point of view of distribution, several components connect through the network to make a complete system. The system manages the components (or nodes) as well as the resources provided by them, which are viewed as a global resource pool accessible to everyone.

DOI : 10.5121/ijasuc.2013.4501

However, some relevant characteristics exist that make pervasive systems more complex in their management, and distinct from all other types of distributed systems. Connection tends to be heterogeneous, so different components can connect through different protocols and bandwidths, generally wireless networks (Zigbee, Bluetooth, Wi-Fi, etc.), with interactions between user and resources that require certain proximity. The system is built dynamically: it is mobile and components appear and disappear at any time [4][5]. The user might require resources provided by the environment in order to complete a service.

From the resources point of view, the components are usually different one from the other, usually mobile and resource limited (i.e. smartphones, sensor devices, or netbooks) in some respect (memory, battery power). To complete a service, the user might require resources provided by the environment so, by sharing their resources, they can improve their capabilities as well as the global system.

In this context, runtime and middleware play a leading role in building optimal systems, both from the point of view of the developers (making programming easy) and the performance (managing heterogeneous and dynamic resources). The main support that middleware should provide is spontaneous interaction, context management and application adaptation. In addition, smart spaces should be able to acquire and apply knowledge about their environment, and to adapt to their inhabitants in order to improve their experience in these environments [6].

Resource scheduling in these systems is aimed at taking advantage of the system resources as a global pool of different quality and power capabilities. This presents a challenge to current research, and is highly dependent on the system platform and goals. Although general ubiquitous systems have proven to be complex, it is possible to establish simple strategies for particular contexts.

In this paper a virtual model to represent user smart spaces, called ambiences, is proposed. This approach is based on a personal device context, a very typical scenario today, where a person owns several devices and would be able to define the most suitable application output. Examples of such ambiences are the user home, the user vehicle or the user him/herself. For these personal contexts we define some data structures to make the proper management of resources distributed into the ambiences, and we present intelligent algorithms to detect the availability of different resources in the system quickly, in order to be able to select the most suitable for a particular application execution.

The rest of the paper is organized as follows: Section 2 reviews the related work. Section 3 characterize the context and types of resources we can find for a specific kind of environment and user profile style; in Section 4 we introduce the middleware, defining the mechanisms and strategies to manage the resource availability. Section 5 present experimental results to evaluate the performance using the proposed algorithms. Finally, Section 6 shows the conclusions and points out new directions for future work.

2. RELATED WORK

Resource management for pervasive computing and ambience intelligence has been addressed by different works since years ago, projects such as, Aire [7], Aura [8], and Gaia [9] are a few examples. These projects have considered the automatic control of the physical conditions in smart spaces through automated sensing of the users and their environment, where automatic control facilitates the activities that the user must regularly undertake. To achieve this, the aforementioned projects have defined a software infrastructure to support the execution of tasks, able to maximize the use of available resources, and minimize user distraction. However, the

main characteristic of these solutions is that, they are oriented to the development of applications with the ability of advantaging pervasive resources, and do not consider a user-centric approach where the user can select one output device, and this selection is based on its own ambiences characteristics, user's own devices, space and mobility.

Other similar projects like the Dynamic Composable Computing (DCC) [10] uses a "join-the-dots" metaphor to create logical computer systems from the best set of wireless component parts available nearby; and MobiGo [11], which consists of a middleware system that migrates service states to achieve seamless mobility saving and resuming in other environments with adaptation of available resources. The main difference with the proposed middleware is that DCC relies on specialized devices to allow composition, and MobiGo use external services to access its data.

In [12], the authors propose an infrastructure and a protocol to incorporate pervasive computing based on cloud services. A major disadvantage in using cloud architectures for pervasive environments is the need for security policies for personal computers and small networks. This security concern is difficult to achieve and maintain in these ubiquitous mobile environments that have dynamic management of resources, and thus often lack central control.

Finally, regarding communication protocols, there exist works [13][14] describing proprietary protocols developed by Apple Inc. and Samsung companies respectively, to share digital multimedia contents using devices from these commercial brands either wirelessly or with a cable. However, a main drawback from these implementations is the interoperability among platforms and devices, these protocols have been designed considering specific hardware features and closed APIs, and therefore with the sole use of such implementations it is not possible to seamlessly and transparently share peripherals among different vendor devices. Thus, considering the proprietary characteristics of these protocols, this work focuses on the design of a low overhead protocol that allows different devices to interoperate, disregarding specific vendor features.

3. USER CONTEXT CHARACTERIZATION

Smart spaces are pervasive systems that work in a broad range of environments, each of them with quite different characteristics, ranging from smart cities to intelligent buildings or autonomous personal devices. Depending on the necessities, the system requirements and runtime design can differ notoriously, adding more or less complexity.

Our vision is based on a user-centric approach; meaning that, all devices and gadgets belong to the same owner and, as the user is moving from one zone to another, is the user's main device the one that will appear or disappear in the ambiences defined in the system, as a result, available devices and their resources can appear or disappear seamlessly as shown in Figure 1.

Figure 1. User moving from zone B to zone C

Examples of these environments are: an apartment, an office, or a vehicle. All these places today hold many gadgets and mobile devices belonging to the same person, several of them having the same or even more powerful capabilities, and, with enhanced characteristics (e.g., higher quality larger screen), or with energy-saving possibilities (e.g., different communication protocols, or output formats). It is important to note that specific or particular security controls, or concerns about privacy, are not necessary, because the user is inside a secure personal space.

In general, we can differentiate autonomous devices, which run their own OS and manage their own resources, from devices that can be considered as merely input/output devices, which provide data. The proposed approach aims to provide a resource pool for all the system that can be accessed from a main autonomous device that acts as the manager of all the resources, either local or external to this device. This pool is spatially grouped in ambiences, and even if, in the system there are some other autonomous devices, only one is the manager, therefore, the rest will only act as input/output providers. In this context, clearly, smart-phones have the same powerful ability to communicate and run all kind of applications as well as laptops, tablets or other general-purpose computer systems. Nevertheless, at a given moment, the user might be willing to benefit from available resources, thus, from a tablet device it can be possible to generate video that can be watched on a TV screen or on a laptop screen, this is, to advantage a better video quality, or just to benefit from a larger display. Given this scenario, the system is defined as a mobile context-aware system, centered on user mobility and running user-centric applications. The main characteristics that define our specific pervasiveness are:

1. The user wears the main device.
2. Other peripherals/gadgets can supply more quality services.
3. The user can move from one zone to another, but he/she is not continuously in movement.

The clearest example is a person at home, using smartphone or laptop applications, and from time to time moving from one room to another. In each room, he/she can use different available resources such as screens (TV, PC) or speakers, depending on the desired quality, energy, etc.
As mentioned before, the proposed solution is based on what has been defined as an “ambience”: the minimum unit to apply resource management policies. The basic environment is a collection of ambiences (e.g. rooms of one’s own house), each one with a set of computational devices. In this way the system can benefit from resources discovered in the spatial proximity. Figure 2 (left) shows an apartment with some ordinary computational devices. We refer to it as a physical model. The Figure 2 (right) also illustrates the equivalent virtual model with each room representing an ambience. Because the visible resources are changing, algorithms to detect and update the system when devices appear/disappear are required, as well as to decide which resources can be used by an application, and moreover, to decide which is the most suitable resource for a specific execution. Therefore, the context-aware algorithms must:

- Detect availability/unavailability of resources in the system on-the-fly while the application is running.

- Be able to adapt to changes in the environment due to user mobility, or when devices are no longer available (flat battery, for example).

- Produce effects to changing conditions, without involving the user in maintenance tasks: the user provides a description of his/her needs.

These phases are known in the field as sensing, adaption and effecting functionality, respectively [15].

As long as we have only one main device in the system, we cannot talk about a centralized or decentralized solution. However, we use a self-scheduling policy, based on the current ambience, and this approach is equivalent to a decentralized view: we believe a decentralized solution is always preferable, based on the particular location of each device. The manager –autonomous-device looks for resources and updates its own information, acting as an individual component. Based on their presence into the ambience, we distinguish between:

- *Static resources*: Resources belonging to an ambience. When the user enters this ambience the resource appears and is always available (i.e. screens, speakers).

- *Dynamic resources*: Resources that appear and disappear independently of the ambience. An example might be sensors.

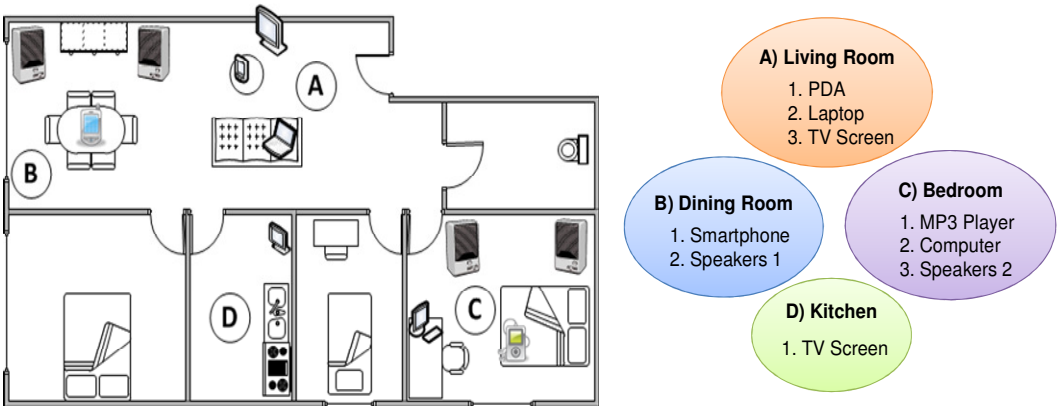

Figure 2. Correlation between the physical and virtual model: left) Physical model, home with furniture and devices in their corresponding room and place; right) Virtual model with an "ambience" representing each room and its associated devices

In the next section, we introduce the main component of our solution: the middleware in charge of managing communication between different devices and to represent and define ambience objects; and also how the system determines the resource it should choose to satisfy user requirements.

4. AMBIENCE RESOURCES MANAGEMENT

The foundation in the proposed system is the already existing ambience information. For each defined ambience, the system has their resources, their properties and the corresponding qualification, and can establish the use of the resource directly. This is applicable in particular for static resources, since at least the local resource (i.e. resource belonging to the main device) is always present. When moving to another ambience, again, the best resource selection is defined (based on quality context) and assigned immediately.

The key is to have the data structures needed to store and maintain information about the system state and resources, and then the mechanisms to detect dynamic changes as well as resource management policies to provide the best service to the user. A communication mechanism able to operate over a wide range of platforms is also necessary. All these elements are managed by a middleware; the first-class object situated in the device is acting as the system manager.

These data structures consist of: the *Ambience struct* representing an ambience and containing all its related information (e.g. position, list of associated devices, etc.); the set of all user defined

ambiences is then represented by the *Ambience list*. The *registered-device struct* that stores each of the devices information (e.g., type, status, locking, and qualification defined by the tuple [Video, Audio, Image]); thus the set of registered devices is then represented by an array of *devices descriptors*. Finally, the *available devices list* which comprises all available devices that are both, visible and connected and from where the output device is chosen. Then, the available devices list is a set of devices that have been previously registered by the user (i.e. a subset of the registered devices list) and which in turn have been associated to the current ambience. The *available devices list* is ordered according to the tuple [*priority, device index*], in the way that, the middleware will always select the first element. In cases where, there are not available devices, the middleware will then select from the local resources (*device index=0*).

To connect to the current output device, a client-server architecture is established, using the well-known socket communication paradigm as the standard interface and available for use in a wide number of platforms and Oss, from WSN to specific embedded systems to general-purpose laptops or computers. The communication link between devices, use the socket connection with the IP address obtained from the *device struct*. The client starts the communication request and once the server accepts the connection, starts to stream the data packets, thus the output device can play the content on the requested resource.

If the user moves to another ambience, different possibilities exist for determining the user mobility that causes an ambience change. To define the current ambience explicitly, we can have a position sensor (we are in the living room, or in the car, for example); or we can define a threshold of appearing/disappearing static resources to indicate that we have already entered a new ambience and what this ambience is. All this should be done in a simple way. To keep information and changes in the specified context, the following issues must be addressed:

- Establish the initial ambience and policy, and prioritize the best output device based on these;
- Detect when devices dynamically appear/disappear in order to update the system state information;
- Change the ambience and then move to a completely different resource table.

We consider that users are always in a particular ambience (the base ambience) based on the current user position. Although all -or almost all- devices can move, appearing and disappearing from a specific ambience, the local resources are always available in all ambiences. The user configures the system by introducing “ambiences” and “ambience devices” into the system. Devices are added in the registered devices list, and from now on, only these devices can be managed by the system.

A multithreaded scheme is used to manage this information and lists updating: controller, monitoring and workers. The monitoring thread checks if the registered devices are available in the current ambience and update the corresponding lists. The controller is the main thread in which many actions are taken as selection of resources based on policies, aggregation of new resources, and updating of statuses. The workers threads actually are in charge of streaming the data to the selected device.

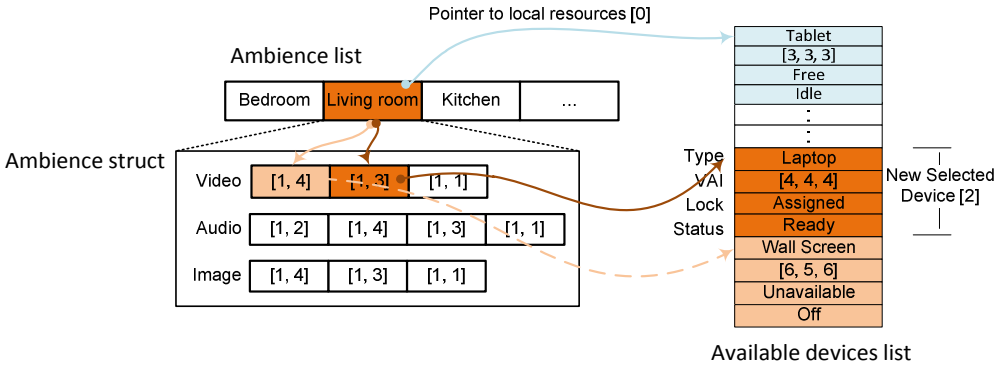

Figure 3. Best output device selection on a specific ambience and policy

4.1. Resource selection policies

Consider an scenario where the base ambience is the living room, and the available devices are a laptop and a wall screen, just as shown in Figure 3. In this scenario, it is assumed that the user is watching a multimedia file, and then the most suitable output device is the wall screen according to its capabilities values (first in the list). Now, consider that the wall screen is not available, the middleware is able to detect this failure, and will look for the next available device in the list, in this case, the laptop, which is available and able to show the multimedia file. The device is chosen from the available device list, which is generated based on a policy. In the initial approach, three different policies to choose a device in a base ambience have been implemented. Each policy sorts the device list from a particular ambience by assigning a priority value to the different devices. This value changes according to the criteria of the policy used. To explain each policy, four devices and three categories are defined, according to their file extension: video (e.g. *avi, mpg, mp4*), audio (e.g. *mp3, wav*) and image (e.g. *jpg, gif, png*). Table 1 shows the value that represents the capability to play the multimedia categories for each category and device considered.

Table 1. Device list qualifications according to its resources capabilities.

Device type	**Video**	**Audio**	**Image**
Computer	4	4	4
TV	6	5	6
Music player	0	6	0
Smartphone	2	2	2

The first policy for choosing a device is called the *Output-Quality policy*. This policy gives priority to the device which has the best audio-visual quality for playing a particular multimedia format. For example, if the user wants to play a video file, and considering the multimedia categories defined above, then the chosen device is the TV, because it has the best video value of the available devices.

The second policy is called the *Cost Estimation policy*. This policy chooses the device based on the combination of two quality criteria: the best quality for playing a particular multimedia output format (Q) and the signal-quality connectivity device (C). The objective is to calculate a priority value (P) from these criteria using the harmonic mean, as follows:

$$P = \frac{2}{\frac{1}{Q} + \frac{1}{C}}$$

Table 2 shows an example of this policy considering two devices in an ambience and the user wishing to play a video file. The select device is the TV, according to the priority value calculated.

Table 2. Cost estimation policy.

Device type	Connectivity device (C)	Multimedia output format (Q)	Priority value (P)
TV	3	6	4
Computer	3	3	3

Finally, the last policy is called the *Manual policy*. This policy is the natural way of selecting a device, enabling users to choose the output device directly through an application menu. Because devices are "visible" to the user, this policy does not satisfy the properties of a pervasive system, although we consider it interesting to offer the user this possibility.

4.2. Ambience adaptation

When the user moves, conditions and many resources may also change. In this situation we have two scenarios: devices that appear/disappear, and eventually an ambience change. The first situation to occur is when a device disappears because the user is moving and the device is no longer accessible: when the resource in use has already disappeared the application has to adapt to the new resource, which may not yet have appeared. Our first approach for this situation is to move to local.

For example, the initial situation is as follows: he/she is in the living room and is using the TV screen to watch a music video. At a given moment, the user exits the room and the connection with the TV is lost, so he/she continues the video visualization through the next on the list: the Smartphone screen.

If the disappearing resource is a dynamic, for example a temperature sensor, our first approach is simply to stop the application from running. In this case it does not matter whether it is the user or the sensor that disappears from the "Base ambience". The same situation may be caused by a device error, and the response will be the same.

When the number of resources is continuously disappearing, we may assume that the ambience is changing. In this case, it is necessary to detect the new ambience in order to react and manage the current possibilities. The question is how many resources are required to indicate that the user is moving from one ambience to another.

A position sensor can be used to simplify the identifications of a particular ambience. This sensor can also help to identify quickly if we are going to a different ambience. Depending on the platform sensors, different options exist to indicate that the user is leaving an ambience: for example, we can establish spatial ambience distributions if allowed, and then entering/exiting depending on the location movement. Another way to consider that the user are entering a new ambience therefore changing his current base ambience is, to discover the number of resources that disappears/appears which indicates that the user is leaving an ambience and hence entering in a new one.

We establish a MAX_DISAPPEAR threshold value: when the number of disappearing ambience resources reaches this value, the system considers the user has left the ambience. In addition, a MIN_APPEAR value determines that a sufficient number of resources are appearing to indicate

that we are in the resources ambience. As soon as the change in "Base ambience" is detected, default resources are updated and output/input redirected to the new one, should this be needed. This shift from one ambience to another may well give rise to a "limbo" situation, in which the resource in use has already disappeared but the application that must adapt to the new resource has yet to appear. Our first approach for this situation is again to move to local.

Finally, the possibility that the user-worn device has no local version of the peripheral is not addressed in this paper.

5. EXPERIMENTAL RESULTS

We evaluate the overhead introduced by the middleware to validate our framework, based on the code size and the time spent in the basic communication operations for sending data.

Given the aforementioned conditions, the initial experiment scenario considers two ambiences of $9m^2$, and with a maximum distance of 2m between two different devices, both with two different and available server devices, a Linux-based High-quality screen and an Apple MacBook. All devices in the ambiences are communicating through a 100Mb/s Wi-Fi connection.

The user wears a MeeGo OS tablet, and the programming language used for the middleware is the framework Qt/C++, which allows GUI applications software to be developed. In the case of the output devices, the server program is developed in Java. The size of the middleware is 368Kb and the server program is 8Kb. Both sizes are small enough to be contained in any device especially resource constrained devices. To test the proposed approach three multimedia files are considered: they consist of a video clip (mp4) of 5.1Mb, a song (*mp3*) of 2.4Mb and a picture (*jpg*) of 39Kb. The user interacts with the middleware through the user interface and selects the policy of public available shared resources. Then, the middleware selects the most suitable device based on the *Output-Quality policy*, as defined in Section 4.1. Once the server device has been selected, the local device streams the corresponding information; if server resource becomes unavailable (noisy environment, out of power, etc.), the main thread selects another device (if available) based on priorities calculated by the *Output-Quality policy*. Note that if no other external resource is available or if the user leaves the ambiences, the local device will continue playing the multimedia files. It is also worth mentioning that the process of detecting an ambience change is too expensive for any mobile device considering the current constraints in battery resources; however, we are exploring new mechanisms to afford this limitation.

As may be deduced from Table 3, the middleware overhead impact on the overall transmission rates is not significant. The time overhead introduced by the middleware is constant and around of 1.2s. The time overhead to send the output to an external device is 8ms, which in this specific kind of application is negligible. And finally, in case of connection failure due to the device disappearance, i.e. changing from ambience, the time taken was in the range of 1.2s and 13.4s.

Table 3. Time overhead results.

Middleware / Type of file transferred	**Video (secs)**	**Audio (secs)**	**Image (secs)**
Without middleware	70,064	160,062	0,063
With middleware in local device	71,326	161,317	1,319
With middleware in output device	71,335	161,325	1,326
With middleware and connection failure	84,755	174,69	Not apply

6. CONCLUSIONS AND FUTURE WORK

In this paper we present a mobile context-aware framework based on pervasive personal systems. The framework focuses on a user-centric environment: the user wears the main computing device (mobile device) and the available resources (mobile and fixed) are located among the different user-defined ambiences in which, given a user's location (current ambience), different output peripherals become available and could provide better quality.

The proposed framework dynamically handles ambience detection. If a user's location changes, currently used peripherals become unavailable and disappear., Thus, the system continues the execution by selecting the default resource (local), and if a threshold of a specific amount of disappeared devices is reached (user moving to a different ambience) the system detects the change of ambience and identifies it in order to publish the new ambience-assigned peripherals based on the defined policy.

The policy management and the proposed algorithms have been tested and the results obtained demonstrate that the overhead introduced by the middleware for streaming the information to the output device (external peripheral) is negligible and thus perfectly suitable for such environments. Moreover, both middleware and server applications are small enough to be contained in any device, including resource constrained devices.

The current implementation is limited by the validation of a threshold indicating that the number of disappeared devices in an ambience has been reached. As future work, we propose to improve dynamicity by predicting the next ambience (on the fly), as the user moves from one ambience to another, by dynamically validating his/her location changes, and therefore enhancing performance and quality of service. Furthermore, we have identified other needs such as the implementation of multi-tasking, which in turn will imply definition of more complex policies to establish priorities among applications and the possibility of preemptive/nonpreemptive policies to access devices once they are already in use.

Finally, we introduce a "contextualized" security component for places where information must be protected due to its confidentiality. Examples of these places are shopping malls or hospitals. In the latter case, doctors and other clinical professionals frequently work with files and images that must be protected, and should not be shared by everyone. Such images must also be deleted and made inaccessible for people who may subsequently use the same devices (for example, patients who may later use the same room).

ACKNOWLEDGEMENTS

The authors would like to thank Carlos Medina for his valuable support with the first prototype implementation. We are also grateful to Jetzabel Serna for her helpful comments and suggestions. This work has been supported by the European Commission through the HiPEAC-3 Network of Excellence (FP7/ICT-217068), the Spanish Ministry of Education (TIN2012-34557), and the Generalitat de Catalunya (2009-SGR-980).

REFERENCES

[1] Kostakos V., O'Neill E. and Penn A. (2006). Designing urban pervasive systems. *IEEE Computer*.

[2] Nippun A. A., Kiran G. and Sudarshan TSB (2010). "Intelligent lighting system using wireless sensor networks". *International Journal of Ad hoc, Sensor & Ubiquitous Computing (IJASUC)* 1(4), pp. 17-27.

[3] Waltersand J., Kanter T. and Rahmani R. (2013). "Establishing multi-criteria context relations supporting ubiquitous immersive participation". *International Journal of Ad hoc, Sensor & Ubiquitous Computing (IJASUC)* 4(2), pp. 59-77.
[4] Kalapriya, K. et al. (2004). "A framework for discovery in pervasive computing for mobile aware task execution". *Computer Frontiers (CF'04)*, Ischia, Italy.
[5] O'Keeffe D. (2010) "Distributed complex event detection for pervasive computing". *Technical Report UCAM-CL-TR-783*, Cambridge.
[6] Cook D. and Das S. (2004). Smart environments: Technology, protocols and applications. *John Wiley and Sons*, pages 424. ISBN: 0-471-54448.
[7] Hanssens N., Kulkarni A., Tuchinda R. and Horton T. (2002) "Building agent based intelligent workspaces". In *proceedings of the ABA Conference*.
[8] Garlan D., Siewiorek D. P., Smailagic A. and Steenkiste O. (2002) "Project Aura: Toward distraction-free ubiquitous computing". *IEEE Pervasive Computing 1(2)* pp. 22-31. Available from: http://www.cs.cmu.edu/ aura/docdir/aura-pervasive02.pdf.
[9] Roman M., Hess C., Cerqueira R., Ranganathan A., Campbell R. H. and Nahrstedt K. (2002). "A middleware infrastructure for active spaces". *IEEE Pervasive Computing 1 (4)*, pp. 74-83.
[10] Roy, W., P. Trevor, et al. (2008). "Dynamic composable computing". *In proceedings of the 9th workshop on Mobile computing systems and applications.* Napa Valley, California, ACM.
[11] Xiang, S. and Umakishore R. (2007) "MobiGo: A middleware for seamless mobility". In *proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE Computer Society.
[12] Youssef A. (2013). "Towards pervasive computing environments with cloud services". *International Journal of Ad hoc, Sensor & Ubiquitous Computing (IJASUC)* 4(3), pp. 1-9.
[13] Brownlee, J. (2010). "September iPod Event: In iOS 4.2, AirTunes Becomes AirPlay". *Cult of Mac. Cultomedia*. Retrieved April 11, 2011.
[14] AllShare. Available from: http://www.samsung.com/global/allshare/pcsw/
[15] Jurmu M. (2007). "Resource management in smart spaces using context-based leases". University of Oulu (Finland), Department of electrical and information engineering. *Master's thesis*, pages 77.

AUTHORS

Beatriz Otero received her Ph.D. degree in Computer Architecture from Universitat Politècnica de Catalunya (UPC) in 2007. She also obtained her Ph.D degree in Computer Science from Universidad Central de Venezuela (UCV) in 2006. She has been working with the UPC at the Department of Computer Architecture (DAC) since 2000 and currently, she is an Assistant Professor and Researcher in this Department. Her research interest includes the design and implementation of system software for distributed computing.

Marisa Gil, Associate Professor at the Universitat Politecnica de Catalunya (UPC) since 1988. She received her Ph.D. in Computer Science from the UPC in 1994. Prof. Gil is a member of HiPEAC, the European Network of Excellence. She is also a Senior Member of the ACM and a Senior Member of the IEEE (IEEE Education, IEEE Computer Society and IEEE WIE). She has served on the board of Spanish IEEE Education. She is also co-founder of MuIN (Network Spanish Women in Informatics). Marisa Gil's research is primarily concerned with the design and implementation of system software for parallel computing, to improve resource management. Her work focus mainly in the area of OS, middleware and runtime multicore architectures support. She is participating in the HiPEAC Network of Excellence and in the TERAFLUX European project.