Generic Role Assignment: A Uniform Middleware Abstraction for Configuration of Pervasive Systems

Muhammad Haroon, Marcus Handte, Pedro José Marrón
Sensor Networks and Pervasive Computing Group
Universität Bonn and Fraunhofer IAIS
Bonn, Germany
{haroon|handte|pjmarron}@cs.uni-bonn.de

***Abstract*—Pervasive computing envisions distraction-free support for user tasks by means of cooperating devices that are invisibly integrated into the environment. Due to device mobility and continuous changes in context, pervasive systems need to be adaptive to realize this vision. To simplify their development, pervasive computing middleware ease the resulting configuration tasks. In the past, middleware developers have used independent abstractions to support different task. In this paper, we argue that support for the configuration should be built on top of one generic abstraction. For this, we introduce the abstraction of a role and we outline how role assignment can be used to support configuration. Finally, we also show that our approach has additional benefits such as improved reuse of configuration logic and increased flexibility to support novel configuration tasks.**

Keywords: pervasive computing, configuration, role assignment

I. INTRODUCTION

Pervasive computing envisions distraction-free support for user tasks by means of cooperating devices that are invisibly integrated into everyday objects. To realize this vision, pervasive systems must optimally leverage the distinct capabilities of the devices that are present in the user's surroundings. Due to changing user context and fluctuations in device availability, pervasive systems can be very dynamic. Thus, in order to achieve the overarching vision, pervasive systems need to be able to manage changes. More specifically, they need to be able to deal with changes in a way that suits the users' expectations and needs.

In practice, dealing with changes can be either the responsibility of the applications or the users. If changes ought to be handled primarily by applications, application development becomes significantly more complicated. On the other hand, if changes are managed primarily by the users, the necessary manual interventions can easily cause an undesirable and even intolerable level of distraction from the actual user task. For this reason, pervasive computing middleware systems usually provide abstractions and mechanisms that simplify or automate the management of changes.

The primary management tasks encountered in pervasive systems are the dynamic formation of an execution environment, the configuration of a distributed application and the context-dependent assignment of access privileges. In the past, researchers have developed a number of abstractions that are tailored towards these tasks for a variety of different scenarios. However, since these tasks have different goals, the abstractions and mechanisms differ.

The same holds true, when comparing middleware systems that are targeted at different scenarios. There, the abstractions and mechanisms may differ even when solely looking at a single configuration task. The resulting heterogeneity has a number of drawbacks. They include the repetitive implementation of similar mechanisms and a reduced flexibility in combining them.

While it might not be desirable or even possible to provide comprehensive support for all tasks in all scenarios with a single abstraction and a single mechanism, a closer analysis of the tasks shows that they exhibit significant commonalities. Specifically, since they are all targeted at dealing with the dynamics of the environment, they all need to continuously identify sets of devices whose context matches certain constraints. Once identified, the sets are then used for the actual task, e.g. to define the execution environment, to distribute an application or to assign access rights.

In this paper, we argue that the continuous context-dependent identification of sets of devices is a key concern of pervasive computing middleware systems. Furthermore, we argue that this concern should be supported by a single uniform abstraction. As a suitable candidate for an abstraction, we propose the concept of a role and we discuss how role assignment can be used as the generic basis to support configuration of pervasive systems. The resulting benefits of this are improved reuse of configuration logic and increased flexibility to support novel configuration tasks.

The remainder of this paper is structured as follows. In the next section, we identify and describe the three fundamental configuration tasks encountered in pervasive systems, namely environment, application and access configuration. On the basis of this description, we outline their commonalities. In Section III, we introduce roles as a uniform abstraction to support the configuration tasks. In Section IV, we discuss the resulting benefits and we discuss how novel combinations of the tasks can be supported easily by the proposed abstraction. Finally, Section V concludes the paper with a short summary and an outlook on ongoing work.

This work is partially funded by the European Union through the PECES research project and the CONET network of excellence.

II. Pervasive Computing Configuration Tasks

The main three configuration tasks in pervasive systems are: the configuration of the execution environment, the configuration of distributed applications and the configuration of access to resources. These three tasks can be easily motivated by looking at the typical hardware and software stack of a pervasive system depicted in Figure 1. At the lowest level, the systems consist of hardware which may be equipped with traditional operating systems. On top of this, the pervasive computing middleware abstracts from the heterogeneity of the hardware and enables the cooperation of the devices. At the top of the stack, applications leverage the capabilities of the overall system. Crosscutting through all components, there are mechanisms that enforce and ensure security.

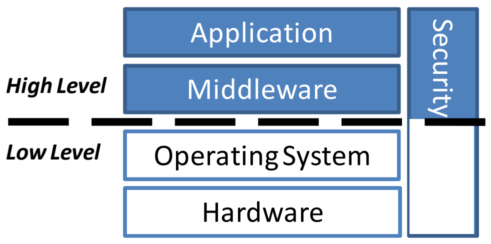

Figure 1: Typical Hardware and Software Stack.

It seems obvious that at some point, either at compilation or execution time, each layer needs to be configured. For instance, at the lower levels, networking must be setup to support the spontaneous interaction of devices. In this paper, however, we focus on configuration tasks that are specific to enable the cooperation of devices. By doing this, we can separate the stack into a low level that is common to all systems and a high level that deals with the specifics of pervasive systems. At the high level, the tasks are the configuration of environment in which the devices collaborate, the configuration of an application that leverages the devices and the configuration of access to resources in the environment. In the following, we describe these three configuration tasks in detail.

A. *Environment Configuration*

The most basic configuration task that needs to be handled by every pervasive computing middleware system is the definition of a scope for its mechanisms. In the following, we refer to this scope as execution environment and we refer to the task of defining and identifying it as environment configuration. The necessity to define a scope follows from the fact that many mechanisms, such as device and service discovery or context management, are hard to scale up to a global scale without relying on some powerful and potentially expensive computing infrastructure, whose use might have undesirable security and privacy implications.

As a consequence, most middleware systems limit the size of the execution environment to a smaller set of devices that can be managed without or with an inexpensive infrastructure. The common underlying assumption hereby is an inverse correlation between the relevance of devices and their physical distance from the users. On the basis of this assumption, execution environments are typically defined either location-centric or proximity-centric. For both approaches, the maximum scope is ultimately defined by network connectivity. Since the scope of a local area network may span multiple buildings and multi-hop routing can easily extend the range of wireless networks, network connectivity alone is not a suitable limiting factor.

Location-centric approaches tackle the problem by defining the environment on the basis of a spatially limited location. Classical middleware systems that support the idea of smart environments such as [7], [8] and [15] fall into this category. In these systems, the execution environment is often limited to certain buildings, e.g. a hospital or a factory, or to a particular room, e.g. a meeting or a class room. Due to the fact that the environments are fixed, the association of stationary devices with the environment is usually performed manually. Based on their current location, mobile devices that are carried by users are then dynamically integrated into their surrounding environment.

In contrast to location-centric approaches which are bound to a fixed location, proximity-centric approaches define the environment as the set of nearby devices. The center of the environment is usually a user which is represented by one or more mobile devices. Thus, the environment changes whenever the user moves. The notion of nearby depends on the middleware system and it may be defined on simple and static network-centric metrics such as hop-count as done in [1] and [2], for example. Another alternative is the use of more complex or even application-specific metrics as proposed in [9]. Of course, it is also possible to use spatial proximity directly in cases where a suitable location system is available.

B. *Application Configuration*

Beyond the configuration of the environment, the configuration of applications is another fundamental task of a pervasive computing middleware. The necessity of application configuration is easily motivated by revisiting the characteristics of pervasive systems and applications. Since devices will be integrated into everyday objects, they will become as specialized as the objects themselves. As a consequence, applications that strive for intuitive and distraction-free task support will need to use the distinct functionality of multiple devices. Combined with the dynamics of the available set of devices, this creates the need to configure and adapt the applications.

The prototypical approach to support application configuration with middleware is to introduce some abstract description of the functionality available in the environment and the functionality that is needed by the application. The middleware can then validate, compute, optimize, implement and adapt the configuration. However, the exact mechanisms for application configuration depend both on the application model, which may be service-based [1], [2], component-based [3], [15] or agent-based [5] and the dynamics of the execution environment.

If the environment is mostly static the configuration can be done manually and upfront. This approach has been utilized by early versions of Gaia [14], for example. There, the requirements and the composition of a component-based application are described by a developer in the form of an application generic description. In order to execute the application in a certain smart environment, the administrator provides a mapping from components to devices in the form of an application customized description. An alternative but also manual approach is taken by the Speakeasy system. In contrast to providing a description of the application, Speakeasy enables the user to combine components arbitrarily. This increases the flexibility but it may also lead to meaningless compositions.

If the execution environment exhibits a higher degree of dynamics, providing a specified mapping between the parts of the applications and the available devices is hardly viable. To mitigate this, more recent versions of Gaia can automate the mapping process by making use of semantic descriptions [13]. In addition, it might not be possible to specify the exact composition of the functionality of the application in advance, as the environments may change drastically. To cope with it, systems like [3] or [11] refrain from specifying a single configuration but use an indirect specification via contractual dependencies or goals.

C. Access Configuration

Last but not least, some pervasive computing middleware systems also support the dynamic configuration of access to resources and information in the environment. Although applicable in general, typical access control mechanisms based on the identity of users are not well-suited to support the dynamics of pervasive systems. Instead of basing access control on centrally managed certificates or shared secrets, it may be preferable to base access control on context information about the users and devices.

Since many physical environments are already equipped with physical access control mechanisms such as fences with gates and doors, the physical location of users and devices is generally considered to be an unobtrusive way of regulating access to computing resources and information in a pervasive setting. When relying on location information or context information in general, a key challenge is the authentication of the relevant parts of the context information. For location information in particular, there are multiple systems such as [4] or [18] that enable its authentication. For other types of context information, context fusion approaches like [12] may be adapted accordingly. On top of authenticated context information, it is possible to use existing access control models [10], [16] to define access rights. In addition, it is possible to extend them to specific domains such as collaboration [17].

D. Commonalities

Despite their different purposes, environment, application and access control configuration essentially share the same common basis since they effectively solve a configuration problem. This becomes apparent when differentiating between the result of the configuration task, the computations that need to be done to produce it and the actual use of the result.

In environment configuration, the result of the configuration task is a set of devices that forms the execution environment. To define this set, the middleware systems utilize different types of constraints. Location-centric approaches introduce constraints on the location whereas proximity-centric approaches introduce constraints that approximate spatial proximity. The resulting set of devices is then used to define the scope of the environment.

In application configuration, the result of the configuration task is a set of devices whose responsibility is to provide certain functionality for an application. To define the set, the middleware systems rely on a combination of constraints that are pre-determined by the corresponding application model and constraints that are defined by the application developer. The resulting set is then used to host the distributed application.

Finally, in access configuration, the result of the configuration task is a set of devices that exhibits certain predefined access rights. To define the set, the middleware systems can restrict the set on the basis of constraints such as the identity of their users or their context. The devices contained in the set are granted access to the resources whereas the devices that do not match the constraints are simply denied.

We can, therefore, conclude that at least from a theoretical point of view, the fundamental basis of these three configuration tasks is the identification of a set of devices on the basis of constraints on their context. Clearly, from a more concrete point of view, there are also subtle differences. For instance, when performing access configuration, the middleware system must ensure that the context information is authentic which is not necessary for other types of configuration. Yet, as we will discuss in the following sections, introducing a uniform abstraction for configuration has a number of interesting and relevant benefits.

III. Roles as a Uniform Configuration Abstraction

In order to create a uniform abstraction for configuration, we need to introduce a clear separation between the result of a configuration task, the computations that need to be done to produce it and the use of this result. In the following, we first provide a general overview how such a separation can be achieved by relying on roles. Thereafter, we outline a basic system architecture and we describe its individual building blocks in more detail.

A. Overview

A role is essentially a tag that can be assigned to one or more devices. As a result, we may reason about roles from the perspective of single device, i.e. whether the device has a certain role, or from the perspective of the overall system, i.e. which set of devices has a certain role. Thus, by using a particular assignment of roles to devices, we may describe the result of arbitrary configuration tasks.

By definition, a role can be assigned to any device as long as there are no further constraints that limit the assignment. To enable the automated computation of an assignment that reflects a particular goal of a configuration task, we introduce rules. Rules define contextual constraints on the assignment of roles to devices. The simplest form of contextual constraint that is generally useful for all configuration tasks is a simple filter.

An example for such a filter is to demand that all devices should be at a certain location. Depending on the configuration task, there are additional classes of rules that need to be supported. In the next subsection, we motivate and describe the different classes by revising the previously introduced tasks.

The set of rules together with their corresponding roles form a role specification. Given that the necessary contextual information can be captured by sensors or other types of information sources, we can use an algorithm to automatically assign roles to the devices whose context satisfies the constraints specified by their rules. Alternatively, we may also empower a user to manually assign roles in order to support cases where the necessary context information is not available or where an automatic assignment is not desirable.

Since roles are just a tag, they are independent from their usage. In order to make use of roles, we need to use them with other mechanisms. These mechanisms are specific to the configuration task and thus, there is little reason to integrate them. To give some examples, for access control, the roles may be directly used as part of a role-based access control model. Similarly, for application configuration, the roles may be used to deploy and wire a set of components. Naturally, this may require further specifications, e.g. to define access privileges or to define the mapping between roles and components as well as their wiring.

It is worth noting that a similar concept has also been proposed to configure wireless sensor networks [6]. However, the role specification and algorithms of that work are specific to monitoring tasks in wireless sensor networks. As a consequence, the overall architecture differs significantly. In the following, we discuss how the underlying principles of the concept can be put to use in a pervasive setting.

B. *System Architecture*

To clarify and to explain how the concept of generic role assignment can be used to support configuration tasks, we outline a basic system architecture. The individual layers and their high-level building blocks are depicted in Figure 2. Conceptually, the three main layers are context management, role assignment and services that use an assignment. Note that in order to keep the architecture simple, we refrain from discussing distribution issues. It should be clear that in a pervasive system, functionality such as context management, requires some form of distribution and that there are various middleware systems including [1] and [2] that can be used as basis for a distributed implementation.

To automate a configuration task, the system needs to be able to automatically capture context information from sensors or other information sources. The context management layer is responsible for abstracting from the details of gathering, fusing or accessing context information and it delivers a uniform provisioning interface. If the configuration task is not security critical, a simple interface for accessing context information is sufficient. In other cases, it is necessary to provide additional information such as the source and the timeliness of the data as well as the probability of its correctness.

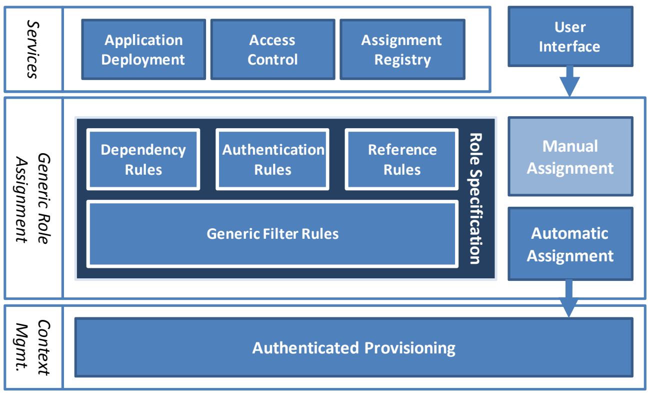

Figure 2: Generic Role Assignment System Architecture.

On top of the context management layer, the generic role assignment layer provides the functionality to define role specifications that consist of a set of roles with associated rules. Once a role specification is passed to the role assignment layer, it can automatically perform the assignment using context information. To do this, the assignment layer must provide an algorithm that computes an assignment. In addition, the layer may also allow a user to manually distribute roles in order to support configuration tasks that cannot or shall not be automated. Based on the configuration task described in Section II, the role specification should support four different classes of rules which we detail in the following:

- **Filter Rules**: The most basic way of restricting the assignment of roles to devices is to apply simple filters which ensure that all assigned devices share certain context. As an example, consider a location-centric environment configuration that restricts the resulting environment to the set of devices at a certain fixed location. There, one may define that the role "member" should only be assigned to the devices at "room 5".

- **Dependency Rules**: Filter rules are appropriate when the context is known in advance. For some configuration tasks, the context may not be fixed but it may depend on the context of other roles. For example, within an application configuration, one might define that two roles should be assigned to devices with the same screen size. Similarly, for proximity-centric environment configuration, one might define that the distance between the devices should not exceed a certain threshold.

- **Reference Rules**: To simplify the use of environment configurations, it is useful to support references from a role specification to an assignment. Thus, as opposed to re-specifying the environment when specifying application configurations, one may simply reference another assignment. Towards this end, reference rules enable a specification to export an identifier. Other specifications may then import this identifier to define a scope, for instance, by using dependency rules.

- **Authentication Rules**: To ensure that an assignment can be used for security critical applications, it is necessary to introduce an additional class of rules. Authentication rules extend other types of rules by specifying additional conditions on the properties of the source of the context information. As an example, an authentication rule might define that the context used by a filter rule should be provided by a particular sensor and the probability for its correctness should exceed a certain percentage.

In order to support the implementation of further services on top of the role assignment layer, the layer provides an interface to query the roles of a device and to query the devices that posses a certain role. Due to changes in context, the assignment may change at runtime. Thus, besides from query interfaces, the role assignment layer can also provide a notification interface that allows interested parties to receive events upon changes of the assignment of roles. Given such a fairly generic interface, one may use the role assignment layer directly within applications. Alternatively, it is possible to implement further middleware services on top of the role assignment layer.

When looking at the configuration tasks that have been introduced earlier, two obvious services are application deployment and access control. An application deployment service can be used to associate the components of a distributed application with a particular role specification. After the role specification is passed to the assignment layer, it computes a role assignment that can be used to distribute the components. Using the notification interface, the deployment service can react to changes, e.g. by adapting the deployment according to the changes. Similarly, an access control service may allow the specification of access rights. Upon access to a resource, the service can then check whether the accessing device exhibits a certain role. Clearly, if such a privilege validation is done in a distributed setting, this may require further security mechanisms to ensure that the role assignment is not forged.

Another interesting service is an assignment registry that can be accessed remotely. In order to enable a more flexible interaction between role specifications and role assignments that are not created by the same system, a remote assignment registry can export (parts of) an assignment. By using reference rules, a remotely accessible assignment registry enables one system to attach a role specification to running assignment in another system. As we will explain in the next section, this can be used to support interesting and novel scenarios.

IV. Benefits of a Role Abstraction

As outlined in the previous section, the proposed role abstraction can be used to support configuration tasks in a broader sense than they are usually found in the literature. Additionally, the clear separation of the specification and computation of an abstract configuration from the task of using the configuration within a mechanism has two main benefits: Improved reuse of configuration logic and increased flexibility for the specification of configurations.

Improved reuse is the immediate result of the uniform abstraction layer. Given that the targeted configuration tasks share a common basis, i.e. the context-dependent identification of sets of devices, the implementation is likely to share parts of the same logic. If roles are used as basis for supporting all tasks, this logic is implemented once in the form of a generic role assignment algorithm. Beyond the pure reuse of code, however, there are additional benefits for the user and application developer. They both benefit from the fact that there is only one abstraction and one specification language that need to be understood. Clearly, one may argue that this specification language could be more complicated since not all classes of rules are required for all configuration tasks. On the other hand, the specification of the classes of rules that are useful for several tasks is identical. In contrast to that, the isolated support of individual configuration tasks is likely to result in different specification constructs. The same holds from the point of view of the user, since manual configuration is now done using a uniform abstraction.

More interesting than improving the reuse of code is the benefit of increased flexibility for specifying configurations. Since the different classes of rules are integrated into a uniform role specification, the classes of rules that are primarily targeted at one particular configuration task can be immediately used for other tasks as well. As an example, one may use authentication rules, resulting from access configuration, to configure the execution environment. This ensures that the devices which are part of the environment actually exhibit the context in accordance with the specified authentication constraints. In addition, since roles are not tied to a particular mechanism, they can also be stacked on top of each other, which may be used to simplify more complex configuration tasks. To show that such configuration tasks are not just a theoretical construct, we present the following two use cases: configuration of a composite environment and configuration of a partially secured application.

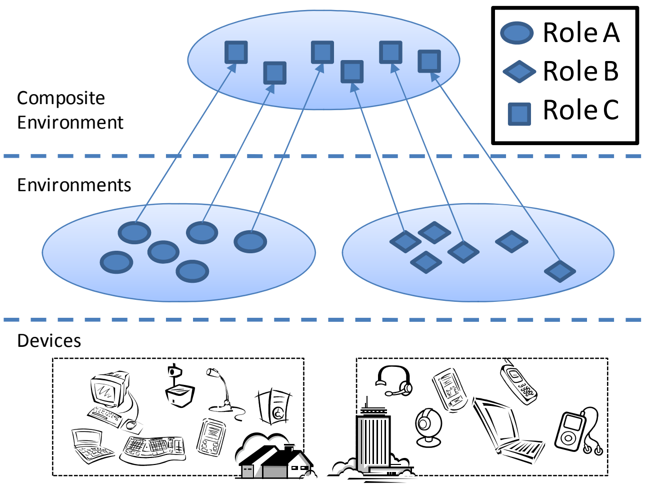

Figure 3: Composing Complex Environments.

A. Configuration of a Composite Environment

As discussed in Section IIA, the underlying assumption for environment configuration is a dependency of relevance on distance. While this dependency does exist in many scenarios, there are some applications that do not exhibit it. The remote control of home appliances and remote collaboration applications are two of the most prominent examples. Here, simply increasing the size of environment to include such cases is hardly an option since some middleware mechanisms such as device discovery are usually based on locality.

In order to support such cases, the reference rules in the role specification can be used to define composite environments. As depicted in Figure 3, the basis for a composite environment can be two environments that are managed by spatially disjoined systems (using roles A and B). We can then define the composite environment as (a subset of) the combination of the roles assigned in each environment, thus creating a new environment (using role C) whose members are locally managed by each system. On top of the composite environment, we may then use mechanisms that do not require locality or we can perform further configuration tasks, for instance, to configure a distributed application.

B. *Configuration of a Partially Secured Application*

Another example of a more complex task is application configuration that needs to make use of the authentication rules to enforce security constraints. As a motivation for it, consider a typical presentation scenario, e.g., at a workshop, which is supported by mobile devices. The speaker may use his mobile devices to switch between slides whereas the audience may use their devices to receive a copy of the slides.

When configuring the distributed application that controls and copies the slides of the presentation, the individual parts of the application exhibit vastly different security requirements. The speaker may want to ensure that only his mobile devices can control the slide show. However, limiting the environment to devices of the speaker is not an option since the audience must be included to receive copies of the slides. In order to support such a scenario, it is possible to define the environment and the parts of the application that receive a copy of the slides without using authentication rules. Yet, to ensure that the control of the slide show is only available to the speaker, the parts of the application that enables this, need to be configured with adequate authentication rules. The ability to arbitrarily mix different classes of rules in the proposed role-based system provides the flexibility needed to cover such scenarios.

V. Conclusion

In this paper, we have described the three primary configuration tasks that are typically found in pervasive systems and have identified their commonalities. We have then discussed how they can be supported uniformly by a generic role abstraction. We argue that the uniform abstraction provided by roles is superior to isolated abstractions that are geared towards a single task. The key reasons for this are the improved reuse and the increased flexibility that can be achieved by relying on a uniform abstraction. We are convinced that the increased flexibility can be used to support new configuration tasks that become relevant when pervasive computing technology is used to implement more complex scenarios.

We are currently in the process of implementing the proposed configuration abstraction as part of the PECES project. A primary goal of this project is the development of system software to enable the seamless integration of smart spaces independently of their physical location.

References

[1] E. Aitenbichler, J. Kangasharju, M. Mühlhäuser, "MundoCore: A Light-weight Infrastructure for Pervasive Computing", *Pervasive and Mobile Computing*, vol.3, n. 4, pp. 332-361, August 2007

[2] C. Becker, G. Schiele, H. Gubbels, K. Rothermel, "BASE - A Micro-broker-based Middleware For Pervasive Computing", *First IEEE International Conference on Pervasive Computing and Communications (PerCom 03)*, pp. 443-451, March 23-26, Fort Worth, USA, 2003

[3] C. Becker, M. Handte, G. Schiele, K. Rothermel, "PCOM - A Component System for Pervasive Computing", *Second IEEE International Conference on Pervasive Computing and Communications*, pp. 67-76, Orlando, USA, March 2004

[4] Y. Cho, L. Bao, M. T. Goodrich, "LAAC: A Location-Aware Access Control Protocol", *Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services*, pp.1-7, July 2006

[5] M. H. Coen, B. Phillips, N. Warshawsky, L. Weisman, S. Peters, P.Finin, "Meeting the computational needs of intelligent environments: The metaglue system", *1st International Workshop on Managing Interactions in Smart Environments*, pp.201-212, December 1999

[6] C. Frank, K. Römer, K., "Algorithms for generic role assignment in wireless sensor networks", *Third international Conference on Embedded Networked Sensor Systems*, pp.230-242, San Diego, California, USA, November 2005

[7] D. Garlan, D. P. Siewiorek, A. Smailagic, P. Steenkiste, "Project Aura: toward distraction-free pervasive computing", *IEEE Pervasive Computing*, vol.1, n. 2. pp.22-31, April-June 2002

[8] B. Johanson, A. Fox, T. Winograd, "The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms", *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 67-74, April-June 2002

[9] S. Kabadayi, C. Julien, "A Local Data Abstraction and Communication Paradigm for Pervasive Computing", *Fifth IEEE International Conference on Pervasive Computing and Communications,* pp.57-68, New York, USA, March 2007

[10] J. Park, R. Sandhu, "The UCON$_{ABC}$ usage control model", *ACM Transactions on information and System Security*, vol.7, pp.128-174, Feburary 2004

[11] J. M. Paluska, H. Pham, U. Saif, G. Chau, C. Terman, S. Ward, "Structured Decomposition of Adaptive Applications", *Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pp. 1-10, March 2008

[12] A. Ranganathan, J. Al-Muhtadi, R. H. Campbell, "Reasoning about Uncertain Contexts in Pervasive Computing Environments", *IEEE Pervasive Computing*, vol. 3, n. 2, pp. 62-70, April-June 2004

[13] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. H. Campbell, M. D. Mickunas, "Olympus: A High-Level Programming Model for Pervasive Computing Environments", *Third IEEE International Conference on Pervasive Computing and Communications*, pp.7-16, March 2005

[14] M. Román, R. H. Campbell, "A Middleware-Based Application Framework for Active Space Applications", ACM/IFIP/USENIX international middleware conference, pp.433-454, 2003

[15] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, K. Nahrstedt,"Gaia: A Middleware Infrastructure to Enable Active Spaces", *IEEE Pervasive Computing*, pp. 74-83, October-December 2002

[16] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role-Based Access Control Models", *IEEE Computer*, vol.29, n. 2, pp.38–47, August 1996

[17] G. Sampemane, P. Naldurg, R. H. Campbell, "Access control for Active Spaces", *Eighteenth Annual Computer Security Applications Conference*, pp.343-352, December 2002

[18] T. Kindberg, K. Zhang, N. Shankar, "Context Authentication Using Constrained Channels", *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pp.14-21, June 2002