Context-sensitive authorization for asynchronous communications

Vincent Hourdin, Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, Michel Riveill

Context-sensitive Authorization for Asynchronous Communications

Vincent Hourdin[1], Jean-Yves Tigli[2], Stéphane Lavirotte[2], Gaëtan Rey[2], Michel Riveill[2]
Mobilegov - I3S[1], I3S (UNS – CNRS)[2], France
{*hourdin,tigli,lavirott,rey,riveill*}*@unice.fr*

Abstract

Main requirement of recent computing environments, like mobile and then ubiquitous computing, is to adapt applications to context. On the other hand, access control generally trust users once they have authenticated, despite the fact that they may reach unauthorized situations. We analyse how dynamic information can be used to improve security in the authorization process, especially in the case of asynchronous communications, like messaging or eventing. We experiment and validate our approach using context as an authorization factor for eventing in Web service for device (like UPnP or DPWS) [1].

1. Introduction

Ubiquitous computing, under the leadership of Mark Weiser's vision [14], has made computing evolve toward multi-device, multi-user, and highly dynamic environments. Miniaturization of hardware and new wireless communication networks have created new devices, worn by users or surrounding them. Due to mobility, devices appear and disappear frequently in such environments.

The major concern in ubiquitous or pervasive computing is adapting applications to users surroundings, and more generally, to their context. In this paper, we focus on limiting communications between entities that are in the same context, for security purposes. Indeed, information involved in ubiquitous computing communications is often privacy-sensitive, and we want to make sure it cannot be received or intercepted by non-authorized entities.

Access control [12] relies on and coexists with authentication, authorization and audit. Authentication can be made on information or persons: it establishes who issued a piece of information, or confirms the identity of a person. However, to ensure that the identity is correct, different authentication factors may be used. If the person possesses the information related to each factor, it is assumed that this is the pretended person.

Authorization takes places after the authentication phase, to grant a principal access to the controlled system. We will study in the following section that authorization is most often static and made one time, leading the users to be considered authorized for a long time. With context changes we cannot assume that a user is authorized throughout the duration of the use of an application, even if he is still identified. We will then explore works on dynamic authorization.

2. Authorization

To extend authorization in order to use dynamic information, we study how it has been handled in different systems. It appears that there are three types of authorization: *static*, *quasi-static*, and *dynamic*.

2.1. Static authorization

Historically, access control used static credentials to confirm user identity and was made only when entering the system. For example, the login phase of an operating system needs a login and a password to authenticate a user, and is made only when he logs in. It can also be an ID card, a fingerprint pattern, or an identification token. Infrastructure information is sometimes used to authenticate users. For example, the Network File System (NFS) access control uses, in its default configuration, the IP address of a client to grant him access, as long as he still uses the file system.

We model the access control process with state diagrams. In Figure 1, a user wants to use a system, and he has to authenticate himself in the first place. Since this is *static* authorization, if authentication is correct and matches an authorization rule, he stays authorized and considered trusted until he logs off.

2.2. Quasi-static authorization

Almost ten years ago, static information for authentication and authorization began to be seen as a limitation

[1]This work is part of the Continuum Project (French Research Agency) ANR-08-VERS-005

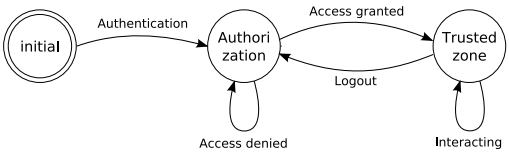

Figure 1. Static authorization

in several domains. In distributed computing for example, with Cholewka *et al.* [3], the task being done could affect access control on some objects. The task was extracted from the workflow of the application, and this dynamic information was considered to be the context of the application.

Later popularized by Web applications, session management has emphasized what we call *quasi-static* authorization. In these systems, credentials are rarely changed compared to the lifespan of an application. Authorization is made at first access of the system, and periodically renewed to keep users authorized in case of information change in authentication or authorization information. This mechanism is called leasing, and often used in publish/subscribe systems. We modelled it in Figure 2.

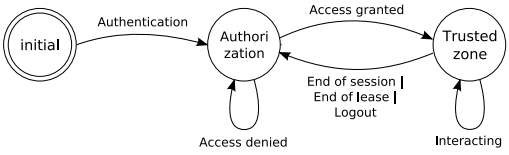

Figure 2. Quasi-static authorization

It is quite similar to the *static* authorization diagram, except that a loop appears between authorized and not-authorized states. Whenever the lease expires, the user has to be authorized again to return in trusted state.

Quasi-static authorization prevents users from being connected to a system forever. A password change, or the introduction of a new authentication factor in the access control system would finally lead to the reevaluation of the authorization when the application decides it. As an example of such system in industry, we can cite Mobilegov Access Control [11] that uses infrastructure-based authentication in addition to password based authentication for different kind of systems.

2.3. Dynamic authorization

Static authorization is also inadequate for ubiquitous computing in which user's context is an important concern, and is already a part of applications. Not using contextual information in security concerns could lead to granting a user access without considering his condition [9]. Contextual information is highly dynamic, because the user is likely to be moving, as much as other users in the same ambient space, with their attached devices. Sensors can also be fixed in the physical infrastructure, like temperature or light sensors. This dynamic information is used to invalidate user's authorization, even if he is still identified by standard authentication factors.

Thus, we introduce the *dynamic* authorization model for environments in which it is needed to frequently check if users are authorized due to changes in dynamic information used for authorization. This open gates to considering highly dynamic contextual information to be used in the access control process. As opposition to *static* and *quasi-static* authorizations, *dynamic* authorization requires to be rechecked according to changes in dynamic information. It is necessary to dynamically modify access permissions granted to users when context information changes. This is twofold: users access privileges must change, as well as resources must adjust their access permissions [9].

While in *static* and *quasi-static* authorization systems users were trusted as long as they were logged in the system, or after a timeout or lease time, with *dynamic* authorization, it must be checked at each operation in the system. This can be done in two ways:

- The first would be to reduce the lease time near zero, and thus needing principals to authenticate and subscribe all the time. Lease time has to be adapted to system's reactivity, which is around one second for ubiquitous computing applications for example. This is very inefficient and consequently a bad solution for embedded devices populating ubiquitous computing environments,

- The second, to be more efficient, would need the system to know user's context all along his use of the system. In that case, the system could react on user's context changes by enforcing authorization policies to determine if the user is still authorized and can be kept or not the trusted area. We modelled this system in Figure 3.

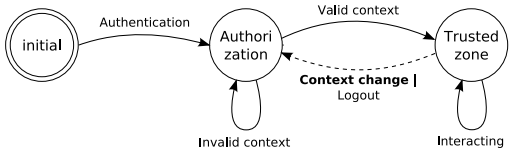

Figure 3. Event-driven dynamic authorization

With this second solution, trusted zone exit and re-entry are context-driven. Since the dynamics of the context and of the application are different, the access control process

is highly reactive. *Quasi-static* and *static* authorization process, in contrast, were driven by the application. However, new issues appear with *dynamic* authorization:

- *How can contextual information be collected by the security system?* As a context-aware system, regular contextual information collection can be done, using context *observers* [4].
- *How can it ensure that the information is authentic?* As stated Kindberg and Zhang, in their experience in the location-aware mobile computing CoolTown project [10]: when using contextual information for access control, the authentication of the data itself must be done. Indeed, dynamic data provided by sensors can be simulated of falsified. If sensors are not able to sign information, it has to be authenticated when users collect it. A trusted observer has to collect the same information than users in order to authenticate it, and verify that it is this information that is used by users to access the system. We will study more deeply this question in section 4.
- *What about privacy?* Of course, placing a trusted entity in users computing environment can be recusant. Westin [15] defined privacy as "*the ability to determine for ourselves when, how, and to what extent information about us is communicated to others*". If the trusted entity describes precisely how contextual information is used, it should be accepted by users.

A good example of such system are works of Bacon *et al.*, who introduce in [2] the OASIS (Open Architecture for Securely Interworking Services) Role-Based Access Control. It uses credentials that a user possesses, along with side conditions that depend on the state of the environment, to authorize him to activate a number of roles. In their model, they define that environmental predicates can be used for environmental constraints or context-sensitive information. Environmental constraints can be checked by any entity in the same environment than the application, thus dynamic information used for authorization can be authenticated.

2.4. Synthesis

The Table 1 summarizes the types of authorization and information used for authentication.

Identity and infrastructure represent principals information commonly used. Infrastructure and environment represent contextual information that can be used. User infrastructure is populated by all computing equipments that are in the context of the user, like local and remote devices. Environment and system infrastructure gather all information that can be get by anyone or do not depend on the infrastructure of the user which has to be authenticated. Date and time are obviously considered as a part of the environment. In some cases [7, 10], location and speed can be considered as a part of the system infrastructure because sensors are part of the security system's domain, and thus can be easily verified.

Table 1. Classification of authorization factors dynamicity

	Static or *quasi-static*	*Dynamic*
Identity	Operating Systems login	?
User infrastructure	Mobilegov AC®	?
System infrastructure and environment	NFS	OASIS[2], CSAC [7]

Our research has proved that there is no project that uses dynamic information for authorization which is not captured inside the domain of the security system, like information from users devices or sensors available inside the context of the user.

3. Access control in communications paradigms

In this section, we focus on how access control is applied to synchronous and asynchronous communications. We consider two entities, A and B, A being granted by B.

3.1. Synchronous communications

Synchronous communications represent a class of communication paradigms that need communicating parties to be present at the same time, and thus be aware of the presence of each other. The most representative example is request/response mode, which can be used locally in function call, or remotely with RPC (Remote Procedure Call).

When A invokes some method of B, we need to check if A is authorized to access this service and thus the information returned by B. A has to authenticate itself to B if it wants the request to be processed and response returned. If authentication fails, the request can be dropped or rejected. We model these different states in Figure 4.

Enabling *dynamic* authorization in synchronous communications can easily be done at request time, since A has in any case to send a message to B to receive the response message.

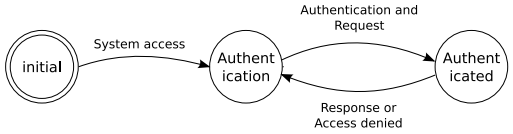

Figure 4. Synchronous communication access control

3.2. Asynchronous communications

Asynchronous communications can take many forms. They decouple in synchronization, and sometimes in space and time the communication between two entities [5]. They are fundamentally based on messages, like JMS (Java Messaging System), an example message-oriented middleware. Communication scheme that use such asynchronous communications are asynchronous procedure call and eventing, with for example publish/subscribe patterns.

Asynchronous method invocation from A to B delays response with a callback to A so that it does not have to wait for the response. We decompose this protocol as two message sending: the first from A to request method execution, and the second from B to give a result when the execution has completed. *Reactivity* is added to the system, since A is not actively waiting, but notified.

Asynchronous communications can also be pure eventing systems, that send messages only in one way, and generally to many recipients. Publish/subscribe systems [5] use event notifications to send information to subscribed entities. They are fully decoupled: data providers publish topics or content types, to which consumers subscribe. Providers produce events regardless if there are some subscriptions, they are generally not in charge of the delivering process.

The subscription is a synchronous process, like a request-response pattern. It is used by consumers to register their interest to a specific event channel and to give information about the connection that will be used to send events.

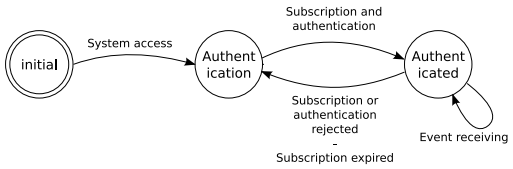

Figure 5. Asynchronous communication access control

However, since following interactions are only one way messages, like events notifications, authorization of the recipient cannot be verified. For *static* authorization, as we have seen, this is not a problem because after subscription, his credentials are not supposed to change or it is not important for system security. With *quasi-static* authorization, the subscription is accepted only for a defined validity time: the lease. Subscriber is trusted only for this time, and has to renew his subscription and authorization, before the end of the lease, to avoid a service interruption. This is modelled in Figure 5: A subscribes and authenticates to B, which will allow A to receive notifications from B.

What can be done for *dynamic* authorization of the recipient in asynchronous communications? Bacon *et al.* [1] already explored access control in publish/subscribe systems, for large scale architectures with multiple administration domains. They use a dedicated security infrastructure for credential management (OASIS RBAC [2]), and access control applies only to event brokers. Their solution is thus based on managing security through a layer lower than the application layer, which is the transport layer.

In the next section, we describe our contribution, how we handle *dynamic* access control for asynchronous communications recipients, in the application layer, and without needing a specific infrastructure for security purposes.

4. Context-based dynamic authorization

We have seen that in context-sensitive computing, *static* or *quasi-static* authorization cannot be used alone because some contexts are not compatible with the authorization granted in first place. We also have seen that an efficient solution would require a trusted entity from the security system to be placed in users' context to ensure the authentication of dynamic information used for access control. We present our solution, first as a model (4.1), then as an implementation (4.2).

4.1. Model

As depicted in Figure 6, the publisher B sends A messages. Rounds tagged with Ob_i represent context observers in A's context. To keep things as simple as possible, we consider that they both act as sensor information observer for A and B, and that they are trusted entities to B. The problem is described as follows: when B sends a one-way message to A, how can it ensure that A is in a context in agreement with B's policy for recipients?

Our contribution is to dynamically add trusted context observers in the context of the subscriber, that notify the publisher from changes in contextual information that are used for end-to-end access control.

Moreover, since most observers Ob_i provide contextual information related to a specific information on the near environment of A, they may vary along with user moves and changes in the infrastructure. Access control rules can thus be adapted to users' context, based on which observers are

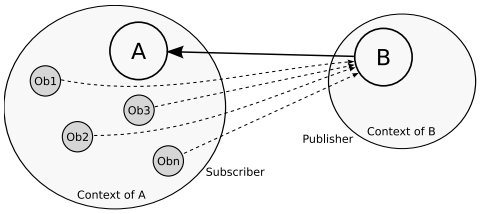

Figure 6. Asynchronous communication and contexts

currently part of users' infrastructure. Figure 7 models the authorization process based on observer information. Once principal is authenticated, its authorization status is bound to the status of validity of observer information.

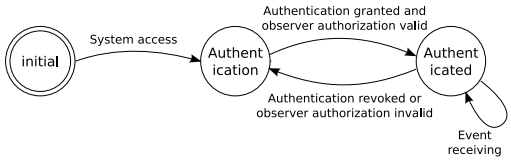

Figure 7. Authorization based on dynamic information with observers

When observers are present, authenticated, and that the value of the contextual information they provide corresponds to an authorized value, the access is granted. As example, the authorization computation is kept simple, based only on equalities between collected information of three observers and information known as valid by the access control system. We can express the authorization process with a logic rule: $grant \equiv Ob_1 \wedge Ob_2 \wedge Ob_3 \wedge valid(Ob_1) \wedge valid(Ob_2) \wedge valid(Ob_3)$. If all observers are present, and that the information they provide is valid, access is granted. As opposite, as soon as an observer information becomes unmet, a granted access is revoked: $denial \equiv \neg Ob_1 \vee \neg Ob_2 \vee \neg Ob_3 \vee \neg valid(Ob_1) \vee \neg valid(Ob_2) \vee \neg valid(Ob_3)$.

These rules are written as part of the authorization process to grant access to users. Several rules should exist for one user, each using different observers. This allows to grant users access based on contextual information while they evolve in not already known environments. Rules are evaluated depending on which observers are available.

4.2. Application for eventing in Service for Device

We chose to implement this solution with two specific architectures and paradigms: Web service for device for the software infrastructure, and publish/subscribe systems for asynchronous communication. Reasons of these choices revolve around one concept: ubiquitous computing.

For many years, service oriented architectures (SOA) have been used in home automation, mobile, pervasive and ubiquitous computing to represent as services the sets of functionalities offered by devices. They offer lots of features discussed in [13] such as encapsulation, dynamicity, discoverability and interoperability. They evolved from standard SOA to SOA for device (SOAD) by adding two main features: *decentralized reactive discovery* and *asynchronous communications*.

Decentralized reactive discovery has been popularized by projects such as SLP [2] or Jini. They suppress the need of a service registry tracking all services active in a network domain. They use multicasted or broadcasted messages to notify that services appear or disappear. Asynchronous communications used by SOAD like Jini are events in a publish/subscribe scheme.

These evolutions allow to create reactive dynamic distributed applications, suitable for ubiquitous computing environments. In addition, when Web technologies are used to implement SOAD, interoperability between all entities is enabled, whether they are heterogeneous devices or simple software services. Only two implementations of Web services for devices currently exist: UPnP [3] and DPWS [4]. UPnP has been created by the UPnP Forum, under the leadership of Microsoft in 1999. It has never be standardized, but is used in many objects of everyday life, like home gateways, or media centers. DPWS appeared in 2004, as a replacement for UPnP, and as a technology based on Web services standards, like WS-Discovery or WS-Eventing.

Publish/subscribe systems use $1 \rightarrow N$ communication scheme: a publisher is able to accept several subscriptions from different clients. Thus, all consumers are notified when issuing an event. This feature will require that observers are managed for each subscriber to the eventing channel, and not simply for each eventing channel.

Service for device composition. To create applications from this infrastructure of services for devices, we use the Service Lightweight Component Architecture (SLCA) [6]. It dynamically orchestrates and composes services for devices using lightweight components. Components are called lightweight because they execute in the same memory addressing space, the same process, and the same component container. The container provides the least possible technical services, also known as non-functional concerns helpers. Distribution has to be explicit.

Containers manage assemblies of components fully dynamically. Component types can be loaded and unloaded, component instances and bindings between them can be added or removed at run-time. Proxy components of Web

[2] The Service Location Protocol.

[3] Universal Plug and Play Forum: http://www.upnp.org/, June 2009

[4] Device Profile for Web Services. http://www.ws4d.org/, June 2009

services for devices are generated, loaded and instantiated dynamically. Thus, we can follow the presence of such service in a container, by adding or removing a proxy component when the service appears or disappears.

Finally, containers can export functionalities created by component assemblies as a new Web service for device using *probe components*. The hierarchy in the model thus uses the service layer, allowing it to be distributed.

Composite service for device adaptation. Since compositions are based on lightweight components, service compositions are fully dynamic. A paradigm called Aspect of Assembly [13] allows to adapt composite services according to specified rules. Aspects of assembly are pieces of information describing how an assembly of components will be structurally modified. They consist of two parts, like regular aspects found in Aspect-Oriented Programming (AOP) [8]: pointcut and advice. Pointcuts describe to which components the modification described by advices have to be weaved (applied).

Moreover, aspects of assembly provide associativity, commutativity and idempotence properties when several aspects are enabled to be weaved at the same time [13].

Implementation. The service for device infrastructure and SLCA are used for all parts of the application: publisher, subscriber and observers. Observers can be trusted entities from the publisher point of view thanks to dynamic insertion of authentication components with aspects of assemblies.

The idea behind the use of lightweight components in composite services is also to enable adapting non-functional concerns at the same layer than the functional core of the application. We use aspects of assembly in the publisher's and subscriber's composite service to add the access control logic (Figure 8). Since we manage all concerns of the application on the same layer, we cannot deal directly with subscriptions handled by the underlying service infrastructure. We have to manage authorizations of all subscribers at the application layer.

In Bacon works [1], group cryptography is used to ensure confidentiality of events between trusted brokers. Encryption keys are updated when principals are declared unauthorized instead of when they unsubscribe, which makes updates happen less frequently in this kind of environment. We use the same technique to ensure that non-authorized entities cannot receive messages.

Events are encrypted with a group key. When observers notify the security system in the event producer, the group key is changed. Modifications of the key are spread to the subscribers of the event channel using the observers. Indeed, since they are in subscribers' context and they are trusted parties, observers can safely deliver the new key.

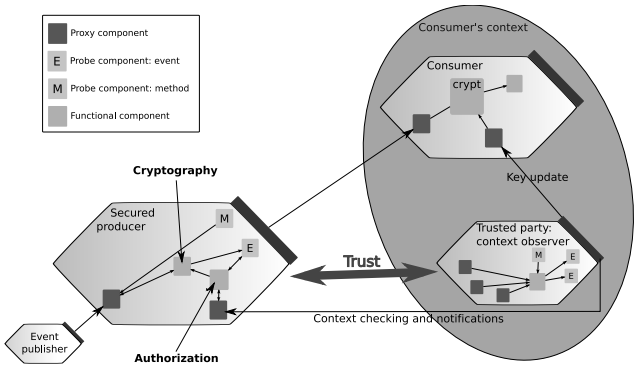

Figure 8. Implementation using SLCA

Aspects of assembly allow us to manage different authorization rules based on appearing and disappearing trusted observers in the environment. Thanks to properties of aspects of assembly, we can enable several rules to be used at the same time for *dynamic* authorization. Even if they are enabled, they won't apply until all observers needed by the rule, defined in pointcuts, are present. The reactive discovery process of Web service for device makes adaptation of authorization rules reactive. This is useful in cases of context overlappings and transitions, or simply to ensure that access won't be denied because of slight changes in the highly dynamic infrastructure of ubiquitous computing.

4.3. Validation

We validate our contribution by three means: we calculate the reactivity of the *dynamic* authorization process ; we compare the number of message exchanged for the access control process and the amount of unauthorized messages received with *quasi-static* authorization and with our *dynamic* authorization.

The process of taking into account changes in contextual information in the authorization involves several operations. Hence, the time elapsed between the variation of a contextual information and the modification of the authorization is the time needed for those operations: data processing by the observer (o), communication between the observer and the proxy component of the event provider (c), and reprocessing the authorization leading to a key change in the composite authorization service (p). $reaction\ time = o + c + p$. o and p are local data processing and take typically less than 1 ms to execute. c depends on how many hops there are between the subscriber and the event provider. In ubiquitous computing, wireless networks are often used, so c may suffer from an important variance. An average of 40 ms then constitutes the predominent value of the reaction time.

In *quasi-static* authorization, like lease-based systems, the value of the lease is several orders higher. The UPnP

specification for example recommends it to be at least half an hour. In security aware systems though, it shouldn't be less than one minute to be efficient enough. The reaction time would then be at most the value of the lease, since the authorization is reenforced at the same time.

The number of messages used for the authorization process in *quasi-static* authorization is periodically increased. Indeed, the leased subscription makes those messages to be send at every lease. Thus, this number follows a linear law, function of the time spent using the system. In *dynamic* authorization, messages are sent only when dynamic information is modified. It can be higher than the linear number of messages from *quasi-static* authorization if context changes more often than the lease time. Else, it can be lower in number of message sent, but still more reactive.

The number of received non-authorized messages in *dynamic* authorization is zero. In *quasi-static* authorization, depending on the rate of sent events and the length of the lease, it can be very important.

5. Conclusion and trends

We have described a solution that allows *dynamic* authorization policies based on dynamic information to be used to manage asynchronous communications access control. Reactive management of dynamic information changes makes the solution efficient. Context is effectively an improvement for access control systems in the authorization process.

Future works will study in what conditions the reactive discovery used by service for device can be secured with the implementation for publish/subscribe eventing we have described. We will also experience how easily we can modify the dynamic information validation to handle inequality operations, like ranges of values for context information instead of equalities.

6 References

[1] J. Bacon, D. Eyers, J. Singh, and P. Pietzuch. Access control in publish/subscribe systems. In *Proceedings of the second international conference on Distributed event-based systems*, pages 23–34. ACM, 2008.

[2] J. Bacon, K. Moody, and W. Yao. A model of OASIS role-based access control and its support for active security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, 2002.

[3] D. G. Cholewka, R. A. Botha, and J. H. P. Eloff. A context-sensitive access control model and prototype implementation. In *Proceedings of the IFIP TC11 Fifteenth Annual Working Conference on Information Security for Global Information Infrastructures*, pages 341–350. Kluwer Academic Publishers, 2000.

[4] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan. Context is key. *Commun. ACM*, 48(3):49–53, 2005.

[5] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM computing Surveys*, 35(2):114–131, 2003.

[6] V. Hourdin, J.-Y. Tigli, S. Lavirotte, G. Rey, and M. Riveill. SLCA, composite services for ubiquitous computing. In *Proceedings of the 5th International Mobility Conference*, Yilan, Taiwan, 2008. Singapore Chapter of ACM.

[7] R. Hulsebosch, A. Salden, M. Bargh, P. Ebben, and J. Reitsma. Context sensitive access control. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 111–119. ACM, 2005.

[8] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. marc Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP*. SpringerVerlag, 1997.

[9] Y. Kim, C. Mon, D. Jeong, J. Lee, C. Song, and D. Baik. Context-aware access control mechanism for ubiquitous applications. *Lecture Notes in Computer Science (LNCS)*, 3528:236–242, 2005.

[10] T. Kindberg, K. Zhang, and N. Shankar. Context authentication using constrained channels. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 14–21. IEEE Computer Society, 2002.

[11] Mobilegov. Mobilegov Access Control ®. See related information on http://www.mobilegov.com/, June 2009.

[12] R. Sandhu and P. Samarati. Access control: principle and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[13] J.-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, D. Cheung-Foo-Wo, E. Callegari, and M. Riveill. WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services. *Annals of Telecommunications (AoT)*, 64(3–4):197–214, Apr 2009.

[14] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, Sep 1991.

[15] A. Westin and O. Ruebhausen. *Privacy and freedom*. Atheneum New York, 1967.