Context-Sensitive Authorization in Interaction Patterns

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

Context-sensitive authorization in interaction patterns

Vincent Hourdin
MobileGov and I3S
930 Route des Colles - BP 145
06901 Sophia-Antipolis France
ferry@polytech.unice.fr

Jean-Yves Tigli
I3S (UNS - CNRS)
930 Route des Colles - BP 145
06903 Sophia-Antipolis France
tigli@polytech.unice.fr

Stéphane Lavirotte
I3S (UNS - CNRS)
930 Route des Colles - BP 145
06903 Sophia-Antipolis France
stephane.lavirotte@unice.fr

Gaëtan Rey
I3S (UNS - CNRS)
930 Route des Colles - BP 145
06903 Sophia-Antipolis France
rey@polytech.unice.fr

Michel Riveill
I3S (UNS - CNRS)
930 Route des Colles - BP 145
06903 Sophia-Antipolis France
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, and what are the implications when applied to interaction patterns. We experiment and validate our approach using context as an authorization factor for eventing in Web service for device (like UPnP or DPWS).

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Control,Information flow controls*; E.4 [**Coding and Information Theory**]: Formal Models of Communication

General Terms

Security, Design

Keywords

Context-awareness, access control, dynamic authorization, context-sensitive authorization

1. INTRODUCTION

Ubiquitous computing, under the leadership of Mark Weiser's vision [15], 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 [13] 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 shoud be used. If the person possesses the information related to each factor, it is assumed that this is the pretended person [11].

Authorization takes places both before system execution, to define policies of the security system, and 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 or controlled by applications, 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 authentified. 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 au-

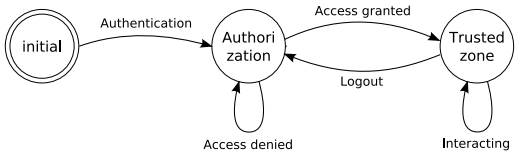

Figure 1: Static authorization

thentication 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 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.

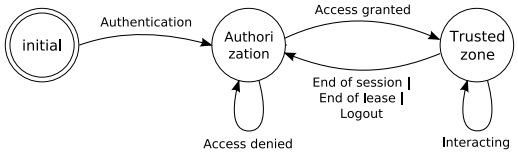

Figure 2: Quasi-static authorization

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 to be connected to a system forever. A password change, or the introduction of a new authentication factor in the access control system would eventually lead to user's credential reevaluation. As an example of such system in industry, we can cite Mobilegov Access Control [12] that uses infrastructure-based authentication in addition to password based authentication for different kind of systems.

2.3 Dynamic authorization

Static and *quasi-static* authorization are 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 [10]. 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. Yet, 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* authorization, *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 or when software infrastructure change.

While in *static* and *quasi-static* authorization subjects were trusted as long as they were logged or for a predefined time, in *dynamic* authorization, authorization 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 subjects 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.

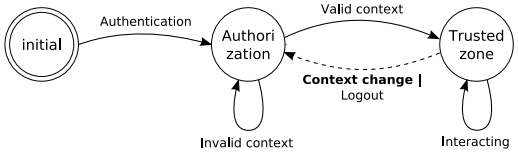

Figure 3: Event-driven dynamic authorization

With this second solution, trusted zone exit and re-entery 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 [11]: when using contextual information for access control, the authentication of the data itself must be done. Indeed, dynamic data are provided by sensors, and they can be simulated of falsified if protocols are not constrained as in [11]. In some cases with group behaviors, information can also be correlated with surrounding entities' to check forged information [7]. 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 [16] 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. Furthermore, one must consider that machine-to-machine communications play a more and more important role, and that privacy in those cases is not relevant.

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 environment of the application, thus it can authenticate dynamic information used for authorization.

2.4 Synthesis

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

Identity and infrastructure represent subjects information commonly used. Infrastructure and environment represent contextual information that can be used. User infrastructure is populated by all computing equipment 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, 11], 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.

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

Table 1: Classification of authentication factors dynamicity

To our knowledge, no project uses dynamic information for authorization when it is not accessed by the domain of the security system, like information from users devices, sensors available through the context of the user.

3. ACCESS CONTROL IN INTERACTION PATTERNS

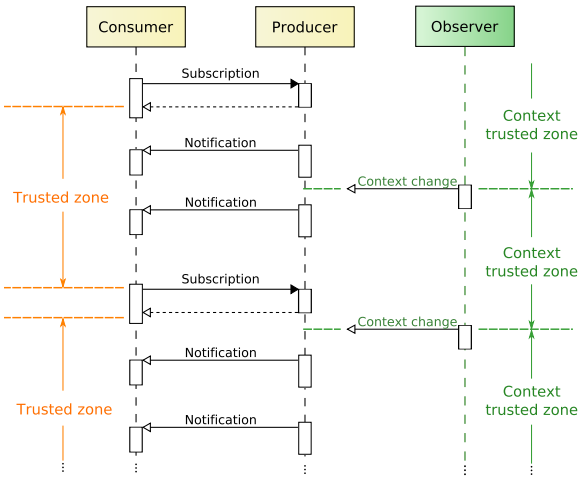

Figure 4: Sequence diagram of trusted zones in a publish/subscribe pattern

In this section, we focus on how access control is managed in interaction patterns. We consider two entities, A and B. A is the consumer. He receives information from the producer B. Thus, he has to be in a valid context, or in other words, A has to be authorized by B.

To emphasize where the problem is, we explain it for the well known publish/subscribe pattern [5] (Figure 4). Publish/subscribe systems are based on two kinds of interactions: the subscription and notifications. Notifications allow the event producer to send information to subscribed entities that he does not necessarily know.

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.

Notification is a purely asynchronous process, made of messages sent by the producer to the consumer. This process thus needs the consumer to be authorized to receive events. Since access control requires the consumer to send authentication and authorization information to the producer, it is practically done when the client subscribes.

However, since following interactions are only one way messages, authorization of the subscriber cannot be verified. For *static* authorization, as we have seen, this is not a problem because after subscription, it is not supposed to have changed 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 access, before the end of the lease, to avoid a service interruption. We call this lease of trust the trusted zone (Figure 4). This is also modeled in Figure 5: A subscribes and authenticates to B, which will allow A to receive notifications from B, until the subscription expires.

What can be done for dynamic *authorization of the recipient?* Figure 4 helps to understand where the problem exactly is. The context observer notifies when the context has changed into a non-authorized context. It is not connected to anything because the

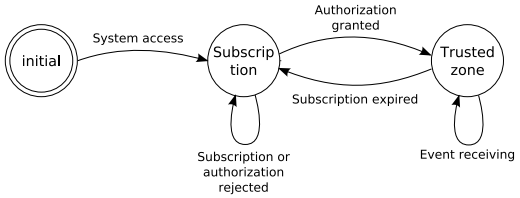

Figure 5: Access control in standard publish/subscribe systems

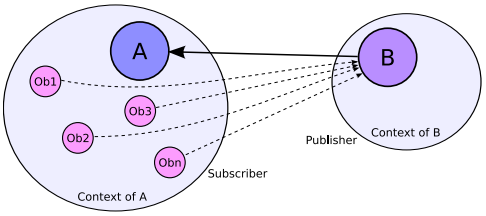

Figure 6: Asynchronous communication and contexts

producer uses the standard publish/subscribe leasing mechanism. Context changes happen while the consumer is in the trusted zone. With *dynamic* authorization, the producer would reenforce the authorization conditions as soon as an event from the context is received. **With *quasi-static* authorization, the consumer is still able to receive notifications, even while his context is not authorized**.

We define the *context trusted zone* as the period during which the producer can be certain that the consumer is authorized by its context, and obviously, still authenticated. Contrary to the trusted zone of usual interaction patterns in which information leak can occur, the *context trusted zone* ensures confidentiality of messages.

Bacon *et al.* [1] already explored access control based on contextual information in publish/subscribe systems; with more details, they focus on a Message Oriented Middleware (MOM) for large scale architectures with multiple administration domains. They use a dedicated security infrastructure for credential management (OASIS RBAC [2]). They apply access control only on event brokers since they are the link to inter-domain networks. Their solution is thus based on managing security through a layer below the application layer: 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 or message management 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 as a model (4.1) and we explain how it can apply to all kind of interaction patterns (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 entities, that notify the controlling entity 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 currently part of users' infrastructure. Figure 7 models the authorization process based on observer information. Once subject is authenticated, its authorization status is bound to the status of validity of observer information.

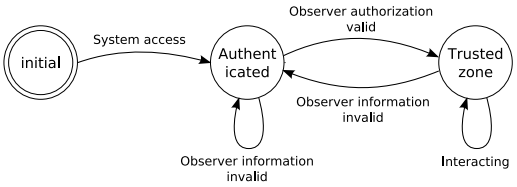

Figure 7: Authorization based on dynamic information with observers

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 allow 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 to all interaction patterns

We already took the example of publish/subscribe systems to describe how the problem could appear. However, other interaction patterns may suffer from the same information leak issue on context changes.

Synchronous interactions. The most representative synchronous interaction pattern is the request/response mode, used in

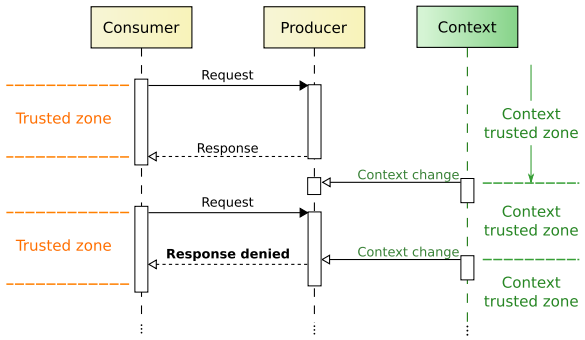

Figure 8: Sequence diagram with context trusted zones for request/response pattern

method invocation and Remote Method Invocation (RMI). In this pattern, two messages are used for each interaction. The first is sent by the consumer to request the execution of some procedure on the producer, possibly with parameters. The second message is sent by the producer to the consumer with the result of the processing.

We depicted in Figure 8 a *dynamic* authorization example for request/response patterns. As a synchronous pattern, it is usually supposed more secure than asynchronous patterns. But as we see in the figure, the same problem appears in this pattern too.

The first message is used by the consumer to send his contextual or authentication information in order to grant access to the method invocation. A context change can occur after this message has been sent, placing the consumer in a non-authorized context. Moreover, the execution time of the method may take several seconds, or even minutes. In mobile environments, the infrastructure changes often, and these circumstances can happen quite frequently.

With *dynamic* authorization, as soon as the context of the consumer gets unauthorized, the procedure processing can be stopped to spare resources, and the consumer is sent an access denied message. In contrast, with *quasi-static* authorization, the producer would not notice that the context has changed, and he would consider the consumer to be still in a trusted zone. The message potentially containing confidential information would be leaked.

Signaling and broadcasting interactions. The third main class of interaction pattern we could identify is the signaling or broadcasting. This pattern is probably the most complicated in which access control can be handled. In DPWS (*Device Profile for Web Services*) for example, WS-Discovery, which uses multicast messages for reactive discovery of Web services, is the only interaction scheme of DPWS that does not handle confidentiality [8]. The reason lies in the decoupling that it provides. Indeed, Eugster [5] has identified three types of coupling:

- time: the consumer and the producer have to be online at the same time. The message is not bufferized, except at operating system level if this is a distributed interaction.
- space: the consumer is known by the producer. In broadcasting and eventing patterns, producers and consumers are often called loosely coupled because they are not bound at design-time, nor designed specifically to execute one with each other. The space decoupling often leads to the fact that several consumers receive the producer's messages. Likewise, in complex publish/subscribe systems, there can be several producers sending messages in the same application.
- synchronization: the consumer is blocked until the producer sends the resulting message. This is typically how request/response is coupled. Asynchronous request/response actually decouples the synchronization of entities: the consumer can continue to execute and will be notified that the result requested earlier is ready.

Signaling and broadcasting are decoupled in space and synchronization. Most eventing systems also have at least these two decoupling. The problem actually appears on a lower level: the transport layer. Publish/subscribe systems are space decoupling from the producer's point of view, but not from the messaging system's point of view. Indeed, consumers have to subscribe, and consequently they are known from the subscription system. *Notifications are then sent using unicast messages to consumers*.

With broadcasting, consumers cannot be known. The pattern is purely one-way, like in TV broadcasting. They are considered in a trusted zone permanently. This is exactly the same problem that appears at the application layer of a publish/subscribe system. The producer may not be aware of subscriptions, and thus cannot deal with access control for each client. If we want to handle access control at the application layer, space decoupling has to use cryptography as a means of access control.

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

The *dynamic* authorization can be applied on interaction systems as long as there is at least one synchronous exchange for trust establishment. For signaling, a solution still exists when the consumer is able to reach the producer: the two-step signaling. A first message is broadcasted, containing no confidential information and only a basic description of how to reach the producer. The second step is initiated by consumers registering their interest for the information, like a subscription in publish/subscribe systems. Then, for notifications (broadcasts or signals), a group key encryption is used. Only consumers in authorized context will have access to the decryption key. As soon as the context of a consumer becomes unauthorized, the group key is changed and spread to other authorized consumers.

The dynamic authorization in eventing and in broadcasting patterns can be handled the same way because of the space decoupling they both offer. This decoupling allows us to consider these two patterns as a single problem for context-awareness and access control. The application of this contribution to a specific infrastructure will allow us to verify it.

5. APPLICATION FOR EVENTING IN WEB SERVICE FOR DEVICE

We chose to implement our context-sensitive authorization 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 two concepts: ubiquitous computing and space decoupling.

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 [14] 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 [1] 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 [2] and DPWS [8]. 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 several 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 for each eventing channel.

5.1 Service for device composition

To create applications from this infrastructure of services for devices, we use the Service Lightweight Component Architecture (SLCA) [6]. It allows to dynamically orchestrate and compose 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 thus has to be explicit: if a component needs to communicate remotely, it has to embed the code to do so. Obviously, we created some external tools that can generate predefined components. From Web services for devices description interfaces for example, we generate client components, that we call proxy components.

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 are generated, loaded and instantiated dynamically and automatically. Thus, we can follow the presence of a service in a container, by adding or removing proxy components when the service appears or disappears.

[1]The Service Location Protocol.

[2]Universal Plug and Play Forum: http://www.upnp.org/

Applications or new functionalities can be created from existing services on the infrastructure by managing an assembly of components inside a container. Proxy components are combined together or with purely functional components to transform information. SLCA components and services for devices communicate mostly using event-based communication patterns, which, more than decoupling entities and increasing dynamicity, will allow to react to context changes efficiently.

Finally, containers can export functionalities created by component assemblies as a new web service for device using *probe components*. Each container has a dynamic functional service interface. When a probe component is instantiated or destroyed, the interface is dynamically modified: a method or an event is added or removed. Consequently, interfaces of existing services can be cloned using adequate probe components. Such services can be secured by adding functional or proxy components to the assembly. Hierarchy in the model is possible but has to use the service layer, which, moreover, allows it to be distributed.

5.2 Composite service for device adaptation

Since compositions are based on lightweight components, service compositions are fully dynamic. A paradigm called Aspect of Assembly [14] 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, keeping black-box property of components. Modifications include adding components and bindings between them. Aspects of Assembly consist of two parts, like regular aspects found in Aspect-Oriented Programming (AOP) [9]: pointcut and advice. Pointcuts describe to which components the modifications described by advices have to be weaved (applied).

If some of the required components expressed in a pointcut are not available, the advice won't be weaved until they become all available. Since service discovery is a reactive process and that containers notifications are events too, aspects can be weaved in response to the appearance of a service (and thus a device) on the infrastructure.

Moreover, aspects of assembly provide associativity, commutativity and idempotence properties when several aspects are enabled to be weaved at the same time [14].

5.3 Implementation

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

We created a simple example of application, modelled in Figure 9. An event publisher service, which can be a sensor or any device, is secured by the composite service on the left. The client of this secured service is a composite service to simplify the figure. This can of course be applied to already existing service clients by only modifying the location (URL) of the service used with a security proxy composite service. Observers are managed in the context of the client by another composite service, to simplify communications.

An idea behind the use of lightweight components in composite services is to enable adapting non-functional concerns in the same

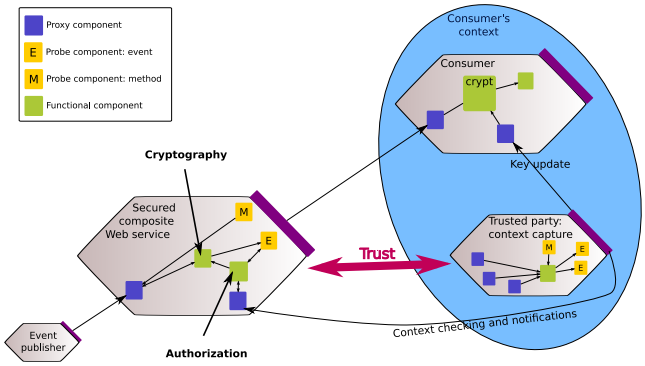

Figure 9: Implementation using SLCA

layer and the same way than the functional core of the application. We use aspects of assembly in the publisher's and subscriber's composite services to add the access control logic.

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 for all subscribers at the application layer, as we have studied in (4.2).

Events will be encrypted with a group key. When observers notify changes of contextual information, if an authorization rule becomes invalid, the security system in the composite service of the event producer will change the group key. 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, they can safely deliver the new key.

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.

5.4 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 maximum the value of the lease, since the authorization process is reprocessed 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.

6. 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. Finally, context can be actually used as 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 range of values for context information instead of equalities.

Acknowledgments

This work is supported by the French ANR research program VERSO in the project *ANR-08-VERS-005* called CONTINUUM.

7. 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 New York, 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 *In: Information Security for Global Information Infrastructures: IFIP TC 11 Sixteenth Annual Working Conference on Information Security*, 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. Tigli, S. Lavirotte, G. Rey, and M. Riveill. SLCA, composite services for ubiquitous computing. In *Proceedings of the International Conference on Mobile Technology, Applications, and Systems (Mobility)*. ACM Singapore, 2008.

[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 New York, 2005.

[8] F. Jammes, A. Mensch, and H. Smit. Service-oriented device communications using the Devices Profile for Web Services. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8. ACM New York, 2005.

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

[10] 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.

[11] 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.

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

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

[14] 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.

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

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