Context-Sensitive Security Framework for Pervasive Environments

Charles-Eric Pigeot, Yann Gripay, Marian Scuturici
LIRIS, INSA de Lyon*
Villeurbanne, France
Charles-Eric.Pigeot,Yann.Gripay,Marian.Scuturici@insa-lyon.fr

Jean-Marc Pierson
IRIT, Université Paul Sabatier
Toulouse, France
pierson@irit.fr

Abstract

Pervasive systems enable us to have an overview of what digital environments will look like in the future. Opportunities given by pervasive systems, both in terms of applications and services to the user are manifold and very promising. From the user point of view, privacy and security of her personal data is a real issue, which must be addressed to make pervasive systems accepted. A wide adoption of pervasive systems can not be possible without an integrated approach to security. We propose a model of security and privacy for pervasive environments, integrated with an architecture, namely PerSE, in which privacy is a main concern and is at the core of the conception.

1. Introduction

Security and privacy in pervasive environments are two key factors to make the technologies of these environments accepted by most of the users. The omnipresence of devices surrounding a user should bring her useful services, depending on her needs, in a reactive way (after the user has expressed her needs), or in a proactive way (anticipating her needs). We believe that those two characteristics are essential to pervasive environments, as well as invisibility (a user should not be aware of interactions between devices) and non-intrusiveness (in her personal life).

However, each user might want to control precisely how she interacts with her environment, *i.e.* which services or data she wants to share and in which context. To this end, she should have the possibility to define different context-aware access authorizations on her data: for example, a user may want to give access to her data only if she remains in a specific room of a building, and only to users who are located in her visually accessible neighbourhood. This example is simple and may be extended to much more complex conditions, but this kind of scenario is very likely to occur with the development of pervasive systems. A security and privacy system for pervasive environments should then enable a user to answer these three questions: Which resources (data, services) I want to share? With who I want to share these resources? And in which context (we'll see later how we define the context) I want to share these resources?

Therefore, security should be integrated between the different devices of a pervasive environment, but also into the devices themselves to control the access to data and services locally hosted. Moreover, the integration of security should not affect performances, especially on mobile devices where resources are limited.

As we'll see in the next section, some works propose access control for pervasive environments, but most of them do not take the user context into account in a satisfying way, though it should be central in pervasive environments, and do not address some pervasive-related issues. Moreover, some studies have shown that the perception of privacy in pervasive environments varies greatly upon users and that one of their main concern is the context in which they remain. Thus, there is a real need for proposing a user-centric privacy solution for pervasive environments.

We propose a security model and infrastructure for pervasive environments, based on two levels of security, using context-aware policies. Our solution may be seen as a step toward the non-intrusiveness of the environment in the personal life. This article is organized as follows. Section 2 discusses related work. In Section 3, we present our theoretical framework on which we have designed our solution. Section 4 presents our infrastructure, the rule-based, context-aware policies, and summarizes the process. In Section 5, we study a use case and its resolution with our solution. Section 6 presents implementation and evaluation aspects of our work, and Section 7 discusses the contributions, and future work and improvements.

2. Related Work

Discretionary Access Control (DAC) and Mandatory Access Control (MAC) were amongst the first access control

*This research is supported by the Scientific Council of INSA Lyon

solutions. However the most used security model is now Role Base Access Control (RBAC) [7], [18]. In RBAC, roles are assigned to users, and the roles have permissions on objects. RBAC is particularly well adapted to organizations like hospitals, enterprises, *etc.* with a very precise and predefined structure because it enables administrators to define and specify security policies that maps exactly the structure of the organization. Moreover, the concept of associating permissions to roles instead of permissions to users resulted in reducing administration costs. The authors of RBAC defined 4 models [7], [18]: $RBAC_0$, the basic model with users, roles and permissions, $RBAC_1$, which defines role hierarchies, $RBAC_2$, which defines constraints on roles, users, permissions, and $RBAC_3$, which gathers $RBAC_1$ and $RBAC_2$.

Although these four models have proven their efficiency and simplified greatly the security management for structured organizations, RBAC do not address issues related to pervasive environments: dynamicity, lack of structure, distribution, and one of the most important, context-awareness. For these reasons, other models have been proposed: Bertino *et al.* present Temporal-RBAC [2] which addresses temporal needs on RBAC with the introduction of time concepts and the support of periodic role enabling/disabling. This model was generalized by Joshi *et al.* with GTRBAC [14] (Generalized Temporal RBAC) that includes a set of language constructs for the specification of various temporal constraints on roles (role activation, role assignment, role permission, *etc.*).

Time management and dynamicity is an important feature for pervasive computing, but we believe that one of the strongest requirement of a security and access control system for pervasive architectures is the context-awareness. It was introduced in access control by Covington *et al.* ([5], [6]) with environment roles in GRBAC (Generalized RBAC), that allows context constraints on roles, *i.e.* context-aware policies definition. However the use of the context data is limited, and the formalization and definition of the context are not satisfying. Zhang and Parashar present DRBAC [21], for Dynamic RBAC, which tackles the dynamic access control needs for pervasive applications. Context data are collected by a "Context Agent" for each user, which triggers role transitions as context changes. But this model does not address important issues about dynamic and distributed access control. Hu and Weaver [10] present a similar model to DRBAC for distributed Healthcare applications, and provide useful definitions and formalization of the context. Our approach is quite similar, but we go further in the definitions, formalization, and usability of the context, which is the core of our approach.

Kumar *et al.* formally propose CS-RBAC [12], for Context Sensitive RBAC, which enables RBAC to enforce security policies dependent on the context of the attempted operations, the user and the object. However, the authors do not provide any satisfying context model, and context is used only as simple constraints. We believe that context can be used much more efficiently to produce real context-aware policies.

All these approaches tend to address problems related to the needs for access control in pervasive environments, but most of them don't solve all problems. Other works propose an integrated and secured architecture for pervasive environments in which privacy is the main requirement.

Langheinrich [13] describes a secured pervasive architecture named pawS, in which privacy assistant carried by users on their mobile devices negotiate the condition of data use before calling services, but the description of user privacy policy becomes complex if the number of entities increases. Hong and Landay propose Confab [9], an infrastructure for privacy-sensitive pervasive applications, that provides several customizable privacy mechanisms to define metadata on data to protect: number of use, time to live, *etc.* This approach does not deal with access control and context awareness. The Daidalos [4] approach is based on virtual identities of users (containing a subset of user data), that are changed and generated depending on the context: when two entities (a user and a service for instance) want to cooperate, they first need to agree on the data to exchange, as in pawS [13]. However, privacy preferences are defined in a static way.

Security policies should be described in a standard language, easily understandable and executable. Therefore, we chose XML to represent and implement our security policies. Numerous works have been realized in this domain, and XML has become a standard in this field. There are some XML-based policy language, such as XACML [20], WS-Policy [19], and SAML [17]. SAML defines an XML framework for exchanging authentication and authorization information to secure Web services, and relies on third-party authorities for provision of "assertions" containing such information, but is not designed to provide support to specify authorization policies. XACML is an XML framework for specifying context-aware access control policies for Web-based resources. WS-Policy is used to describe security policies in terms of their characteristics and supported features (such as required "security tokens", encryption algorithms, privacy rules, *etc.*). In fact, WS-Policy is a meta-language which can be used to create various policy languages for different purposes, and can be used to define an access control policy. Moreover, XACML and WS-Policy tend to become standards in security policy definition.

We needed a XML-based language to implement our policies, but we chose to define a new, simple XML syntax instead of using an existing language, mainly for simplicity and lack of time reasons. WS-Policy was too com-

plex and too expressive for our needs, and XACML, which is very close to our XML syntax and semantic, generates much heavier XML files due to its expressiveness. Anyway, translation of our rules to XACML rules is simple and may be done in a future work for more interoperability.

3. Theoretical Framework and Definitions

In this Section, we present the definitions and the framework for our model.

Pervasive Environment: A set of devices acting together in order to satisfy a user with minimal intrusion.

Base: A meta-service running on a pervasive device, enabling it to share its local resources. Each base is in charge of communications with other bases, in order to run in a smart and optimized way distributed services.

Resource: We define a resource as a data or a service. A data can be a value, a file, *etc.* A service is hosted on a base. A resource can belong to a user or to a base.

Entity: Generic term for either a base, a resource, or a user, *i.e.* the three elements that can interact with each other in our vision of a pervasive environment.

Group of entities: We define a group of entities as a gathering of entities (bases, users, resources).

Context Domain: The set of all possible context states of the pervasive environment.

Context SubDomain: A set of states included in the context domain ($ContextSubDomain \subseteq ContextDomain$).

Rule support: The context subdomain in which the rule is applicable.

Communication Rule: A communication rule is defined as an action to realize on messages coming from a sending entity or a group of entity, and sent to a recipient or a group of recipient, in a certain context. A communication rule can be represented by the tuple:

$$R_C = (\alpha, \sigma, \delta, \gamma) \quad \begin{cases} \alpha : \text{action to do on the message} \\ \sigma : \text{sender of the message} \\ \delta : \text{recipient of the message} \\ \gamma : \text{support of the rule} \end{cases}$$

Resource Access Rule: A resource access rule is defined as a permission or a group of permissions given to an entity or a group of entities on a resource or a group of resources in a certain context. A resource access rule can be represented by the tuple:

$$R_A = (\epsilon, \pi, \rho, \gamma) \quad \begin{cases} \epsilon : \text{entity to which the permission is granted} \\ \pi : \text{permission granted} \\ \rho : \text{resource on which the permission is granted} \\ \gamma : \text{support of the rule} \end{cases}$$

Communication Profile: We define a communication profile as the set of communication rules with the same support, *i.e.* that are valid in the same context. If p is a communication profile, r_i a communication rule and S_{r_i} the rule support of r_i, then:

$$p = \{r_1, r_2, \ldots, r_n\} \Leftrightarrow S_{r_1} = S_{r_2} = \ldots = S_{r_n}$$

Resource Access Profile: We define a resource access profile as the set of resource access rules with the same support, *i.e.* that are valid in the same context. If p is a resource access profile, r_i a resource access rule and S_{r_i} the rule support of r_i, then:

$$p = \{r_1, r_2, \ldots, r_n\} \Leftrightarrow S_{r_1} = S_{r_2} = \ldots = S_{r_n}$$

P_C is the set of all the communication profiles and P_A is the set of all the resource access profiles.

Security Policy: We define a security policy as the set of every communication profile and resource access profile defined by a user to protect her base. The way we define the policies can introduce conflicts between the rules. In order to have a non-contradictory policy, we define relations of priority.

Rule Priority: Let A be the set of all the possible actions on a message. We define on this set a relation of priority, noted $>$. Intuitively, if α_1 has a higher priority than α_2 and if the two actions are applicable in the mean time, then the action α_1 will be applied.

Profile Priority: We define the function Priority Φ:

$$\Phi : \begin{array}{ccc} P_C & \longrightarrow & [0,1] \\ p_c & \mapsto & k \end{array}$$

The value of this function for a profile represents the priority of this profile, defined by the user. If $\Phi(p_1) > \Phi(p_2)$, then the rules in p_1 will be chosen in priority compared to the rules of p_2 if a conflict occurs.

4. Privacy Architecture and Processes

4.1. Requirements for a secured pervasive architecture

Based upon the model we described in the previous section, we designed a security and privacy infrastructure for pervasive environments. This architecture, composed of three modules, is integrated in an existing pervasive environment named PerSE [8] (for **Per**vasive **S**ervice **E**nvironment) detailed in the next section.

The security infrastructure we propose is based on a two-level filtering system: communication filtering and resource access control. These two security levels use security policies, composed of security rules defined by the user, to derive the access decision. In our infrastructure, two modules are dedicated to this filtering, and the third module decides which policy to use depending on the context. Indeed, we believe that a strong requirement for a secured pervasive environment is the context-awareness: a user might want to change her privacy policy for different contexts, so in order to limit interaction with users, the system should proactively

enforce the user-defined policy corresponding to the current context situation.

Some works have tried to study the user perception toward security and privacy threats in pervasive environments. Beckwith [1] concludes from his studies that users have a very limited perception of potential threats and risks of these technologies. For example, electronic badges are not seen as a potential means to follow every movement of a user, but only a means to get to certain places and to open doors. Beckwith makes another important conclusion: the definition of security and privacy differs greatly from one user to another, and every user assesses privacy depending on different parameters and criterions. Moreover, the studies of Dey *et al.* [15] (and some others studies [11], [13]) clearly reveal that the quantity of personal information given in response to a request depends both on the identity of the request emitter and on user current context.

For these reasons, we decided to give to the user the opportunity to decide how the context is used in the security policies, and in which context a security policy is valid, that is to say which parameters she will use and which constraints she will put on these parameters. We will see in the next sections how we defined a language to help the user to describe a context.

4.2. PerSE

PerSE [8] represents our vision of a user-oriented pervasive environment, in which the user can access to resources (services, data) hosted on various surrounding devices by simply expressing an intention. Moreover, this platform is proactive and non-intrusive, two main characteristics of pervasive environments.

As a part of the PerSE environment, each device has to run a meta-service, the Base, enabling it to share its local resources. The PerSE Base is in charge of communications with other Bases, in order to run distributed services in a smart and optimized way.

A PerSE environment consists of many independent Bases, able to discover each other, and to send and receive messages through different communication channels (LAN, Wifi, Bluetooth) available on the devices.

In order to respond to user needs, a modeling of her intention is necessary. The PsaQL language [3] enables the user (or an application) to express her intention (called a *partial action*) describing the services the user wants to use and their possible location. The PerSE Base has then to interpret this intention into a connected graph of services meant to be executed (called a *complete action*).

The PerSE architecture (Figure 1) is composed of three layers, corresponding to the three main functionalities of the Base: Communication, Environment and Action. Between and within these layers, we integrated our security infrastructure, composed of three modules.

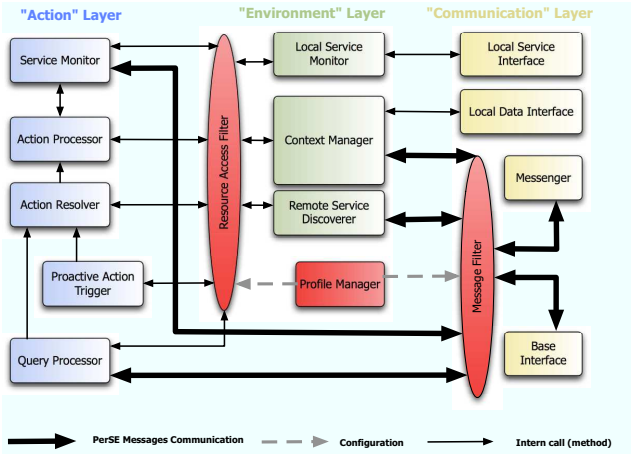

Figure 1. The PerSE Architecture

4.2.1 Communication Layer

The Communication layer is the lowest level layer and is in charge of the communications of the PerSE Base with its environment, that is to say other PerSE Bases.

The Local Data Interface and Local Service Interface modules handle the physical access to the local data and services of the Base. The Base Interface module is the local access point for a user who wants to interact with the Base, especially to start *partial actions*. The Messenger module is in charge of the communications between the Bases, by exchanging specific messages.

In this layer, the first security module, the Message Filter, acts as a filter on incoming and outgoing messages. In the PerSE environment, the communications between the different Bases rely on messages built with a specific structure (Figure 2). It is composed of two main parts: the header and the data. The header is divided in four layers, each of them containing information on the sender and receiver entity: base, service of the base, user, *etc.*

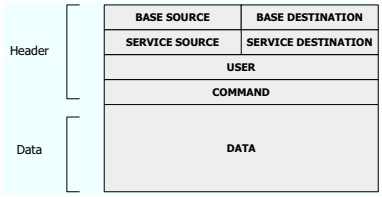

Figure 2. PerSE Message Structure

By using this information stored in the message structure, the Message Filter can decide to stop the message or to let it pass through. The decision is made by a communication policy enforcement, and this communication policy,

composed of communication rules, is defined by the user. The Message Filter is both a Policy Decision Point (PDP) and Policy Enforcement Point (PEP), since it renders authorization decision and performs access control. We'll see in the next sections how the user can define such policies.

4.2.2 Environment Layer

The Environment Layer manages the local knowledge of the Base on its environment. The Local Service Monitor manages all the available local services. The Context Manager handles both the access to the local context and the distant access to other Bases context through context request. The Remote Service Discoverer regularly sends requests to other PerSE Bases to maintain a local repository of the available services hosted on distant Bases.

This layer manages and has access to local data and services, so we decided to protect this access by a second filter, the Resource Access Filter situated between the Environment Layer and the Action Layer. The Resource Access Filter is an access controller: when a request from the Action Layer occurs, the Resource Access Filter, by enforcing a resource access policy defined by the user, decides whether the request is legal or not. The Resource Access Filter is a second PDP and PEP, since it makes decisions on access control and enforce these decisions.

The last security component of the architecture is the Profile Manager. Located between the two filters, it decides which policy to use at the time of the request, depending on the user context, and transmits this policy to the filters. The context in which a policy is applicable is specified by the user. The Profile Manager is the main module of the security architecture we designed, as it controls the two other components. The two filters represent two different levels of security, but it is possible to use only one of them. For example, if the administrator decides that he doesn't need communication filtering, the Message Filter can be deactivated. However, the Profile Manager as a Policy Administration Point (PAP) and Policy Decision Point (it decides which policy to use) must be present and active in the architecture and thus can not be optional.

4.2.3 Action Layer

The Action Layer is intended to gather the request from users or applications, and to execute actions to answer to these requests. The Query Processor receives PerSE messages, containing requests, and answers to these requests by obtaining the asked data in the Environment Layer. He also receives *partial actions*, triggering a new action. The Proactive Action Trigger watches over the context and maintains a history of executed actions. It can also produce *partial actions* proactively. The *partial actions* are transmitted to the Action Resolver to be transformed, depending on the context and the available services, in *complete actions*, executable by the Action Processor. The Service Monitor is used to monitor the execution of local and distant services.

4.3. Rule-based Communication and Access Control

User is the key actor of our security system. She can define security policies that will be applied on her Base. As we saw in the previous section, two types of policies, enforced at different levels in the architecture, can be defined, based on two types of rules: Communication policies, composed of communication rules, and resource access policies, composed of resource access rules. To this end, we defined two declarative languages to make easier the specification of the rules for the user.

```
<communication_rule> ::= DO <action><communication_part>
<action> ::= allow | deny | drop
<communication_part> ::= ON <communication>
    [USING <protocol>] <sender_part>
<communication> ::= incoming perse_messages | outgoing
    perse_messages
<protocol> ::= ip | tcp | udp | icmp
<sender_part> ::= FROM <sender> [<destination_part>]
    [<context_part>]
<sender> ::= all | <entity> | <group_of_entity>
<destination_part> ::= TO <destination>
<destination> ::= all | <entity> | <group_of_entity>
<group_of_entity> ::= 'Group ' <'a'-'z','A'-'Z','0'-'9'>
<entity> ::= <base> | <service> | <user>
<base > ::= 'Ba-' {<'a'-'z', 'A'-'Z', '0'-'9'>}
<service> ::= 'Se-' {<'a'-'z', 'A'-'Z', '0'-'9'>}
<user> ::= 'Us-' {<'a'-'z', 'A'-'Z', '0'-'9'>}
<context_part> ::= <inclusion> CONTEXTS <contexts>
<inclusion> ::= IN | NOT IN
<contexts> ::= <context_name> [',' <contexts>]
<context_name> ::= {<'a'-'z', 'A'-'Z', '0'-'9'>}
```

Figure 3. Communication Rule grammar

The Communication Rules syntax is described by a BNF grammar in Figure 3. Here are a few examples of Communication rules based on this grammar:

```
DO deny ON incoming perse_messages USING tcp FROM all
    IN CONTEXT neighbourhood
DO allow ON incoming perse_messages FROM Us-12
    IN CONTEXT temp_high
DO drop ON outgoing perse_messages FROM Us-12 TO Se-13
```

The first rule specifies that every incoming message using the protocol TCP should be blocked if the context defined as "neighbourhood" is valid (see section 4.4 for context definitions), whoever the user might be. The second rule allows every incoming message from the user identified as "Us-12" if the context defined as "temp_high" is valid.

The action part of the rule describes which action to execute on the incoming or outgoing message: allow (let pass through), deny (stop the message and notify the sender of this failure), and drop (delete the message without notification). The communication part tells on what type of communication the rule is valid: incoming message, outgoing

message, or others. Our architecture uses PerSE messages, as described in the examples, but we can assume that evolutions of the PerSE environment could introduce new types of communications. An optional part of the rule is the protocol part specifying a certain type of network protocol.

After the protocol part are the two sender and receiver parts of the rule. The receiver and sender can be either a user, a service or a base, or a group of each of these entities (groups are defined by the user as a set of entities). Each entity has a unique identifier used to fill in the message header fields (see Figure 2). For the moment, we trust the incoming message, *i.e.* we consider that the data stored in the fields are right and have not been modified during the communication. We discuss at the end of the article the issues related to this hypothesis. If the sender or receiver part specify a group, then the system has to determine which group the entity of the message belongs to before making its decision.

The final part of the rule is the context part, which describes in which context(s) the rule is applicable or not applicable. We'll see later in details how those contexts are defined. If this part is absent, the rule is applicable whatever the context may be, and is then similar to a basic firewall rule. If the context part is defined, the Message Filter behaves like a context-aware firewall.

These rules are quite expressive, and they enable the administrator to define a precise context-aware filtering policy for the PerSE Base. Moreover, the language used to describe these rules is similar to a natural language, which simplifies the expression of user preferences.

In the same way, we also defined a grammar to describe Resource Access Rules (Figure 4), which are higher level rules used to control access to local resources (data and services). Resource Access Rules define permissions for entities or group of entities on a resource or a group of resources in some contexts. When a request from a user comes at the base, and after the Message Filter has decided whether the message is allowed to go further, the Resource Access Filter, depending on the resource access rules, decides to provide the answer to the request or not.

```
resource_access_rule> ::= <subject> <permission_part>
<subject> ::= all | <entity> | <group_of_entity>
<permission_part> ::= <permission> DO <action_part>
<permission> ::= CAN
<action_part> ::= <group_of_action> ON <resource_part>
<group_of_action> ::= <action>  AND <group_of_action>
<action> ::= everything | nothing | read | modify |
    delete | execute | monitor | ...
<resource_part> ::= <resource> <context_part>
<resource> ::= all | {<'a'-'z', 'A'-'Z', '0'-'9'>}
<context_part> ::= <inclusion> CONTEXTS <contexts>
<inclusion> ::= IN | NOT IN
<contexts> ::= <context_name> [',' <contexts>]
<context_name> ::= {<'a'-'z', 'A'-'Z', '0'-'9'>}
```

Figure 4. Resource Access Rule grammar

In the following examples , the first rule specifies that the service identified as "Se-098" has the permission to modify and delete the resource "img18.jpg" if the context defined as "neighbourhood" is valid. This high level language is easy to understand, even for a basic user.

```
Se-098 CAN DO write AND delete ON img18.jpg
    IN CONTEXT neighbourhood
Ba-367 CAN DO execute AND monitor ON Se-13
    IN CONTEXT low_battery
Se-665 CAN DO everything ON Se-13
    IN CONTEXT people_in_room
```

4.4. Context and Profiles

A strong requirement that we have clearly identified before the conception of our model was the context awareness. In a pervasive environment, a security policy defined by a user should depend on the user's context, and on context information that seems important to her among the huge quantity of contextual data that a pervasive environment can gather.

Since all users don't use context in the same way, we choose to let the user herself define the context in which a rule is applicable, so that among all defined rules, only a subset will be applicable at the time of a request. We saw that the last parameter of rule definitions is used to specify the context of rules.

To define a context, we created a simple declarative language (Figure 5), similar to the ones we defined for the rules, but more powerful and expressive. This expressiveness enables the user to define precisely the context, using every contextual parameter available she might want to use. The need for such a language came when we had to chose an interpretable language to express the context functions. In Section 6, we explain why we chose *perl* to describe a context function executable by the system to determine automatically the context. However, if *perl* is a powerful language, and quite simple for advanced users, it can become very hard for a basic user who wants to describe a context with her own words but is not used to languages such as *perl*. We decided to define a simple language, similar to a natural language, to fill the gap between "low-level" languages and contextual parameters, and "high-level" languages and parameters (temperature, location, *etc.* are high-level expressions of contextual parameters), more understandable for a basic user.

A context has a name and a priority. The priority is a number situated in the interval [0,1] and we'll explain later the role of this variable. A context definition uses many parameters, called contextual parameters, which correspond to the type of information on the context. These parameters are taken from the context of the caller base and the local base. For instance, the contextual parameters used to define a context may be the temperature, the lighting, the

```
<context>::= CONTEXT <name> WITH PRIORITY <priority>
    USING <parameters > IS DEFINED BY <definition_part>
<name>::= {<'a'-'z', 'A'-'Z', '0'-'9'>}
<priority>::= 0.{<'0'-'9'>}
<parameters> ::= local_base | caller_base |
    local_and_caller_base
<definition_part> ::= <context_condition>
    [ AND <context_condition> ]
<context_condition> ::= <contextual_parameter> OF <base>
    IS <relation>
    <contextual_value>|<group_of_contextual_value>|
    <contextual_parameter> OF <base>|<perl_expression>
<contextual_parameter> ::= temperature | lightning |
     location | ... | trust
<base> ::= localbase | callerbase
<relation> ::=  equal to | superior to | inferior to |
    superior or equal to | inferior or equal to |
    included in | ... | not in
<group_of_contextual_value> ::= contextual_value
    [',' <group_of_contextual_value>]
<contextual_value> ::= {<'a'-'z', 'A'-'Z', '0'-'9'>}
```

Figure 5. Context Definition grammar

location,... The user can specify a relation that links the parameter to the value: equal to, superior to, *etc.*

```
CONTEXT trusty
  WITH PRIORITY 0.5
  USING caller_and_local_base
  IS DEFINED BY
    trust OF callerbase IS superior or equal to 0.7 AND
    location OF localbase IS equal to 'room 203'
```

Figure 6. Example of Context Definition

In the example (Figure 6), a user defined a context named "trusty", and the context will be "trusty" when the location of the local base is the room named "room 203", and when the trust mark (which we consider as a context parameter we can calculate locally depending on the history of the interactions with an entity) of the caller base is superior or equal to 0.7. This is only an example among the numerous contexts a user is able to define with this syntax.

As we saw, a rule is valid in a context, as specified with the last parameter of rules which is the name of a user-defined context. All rules applicable in the same context are gathered in profiles related to this precise context: a profile is a contextual security policy. A potential problem that may occur is that at the time of a request, more than one context are valid. At this moment, more than one security policies are applicable for the request, and some conflicts may occur between the rules of the policies. For this reason, we have introduced the context priority: if two or more contexts are applicable at the same time, the system will choose the context, then the security policy, with the highest priority. The priority guarantees that only one policy is enforced at a time. If two contexts have the same priority, then the system will choose the one defined first.

If a conflict occurs between the rules in a security policy, the conflict is resolved by the priority of action the user defines. Indeed, some actions are more important than others, and the user herself defines the importance of the actions: for example, in a very secured environment, the user will decide that the "deny" action on messages is more important than the "allow" action. We forbid the definition of two action with the same priority, to make easier and more meaningful the decision made. Indeed, two actions with the same priority would not be very coherent.

4.5. Summary

To make the understanding of our solution easier, we propose an outline (Figure 7) that summarizes the process of a request treatment by the security infrastructure.

When the request, encapsulated in a message, arrives on a device (1) (a PerSE Base for example), the first filter, the Message Filter, analyzes the request and asks to the Profile Manager the profiles (the security policies) to enforce (2). The Profile Manager asks its internal modules to gather information about the current context, and determines in which predefined context the bases are, by executing the corresponding *perl* functions (3). It then gives the profiles corresponding to the valid contexts to the Message Filter (4). The Message Filter decides what to do on the incoming message, by enforcing the given policies and applying the conflict resolution (5) by comparing priorities of profiles and rules.

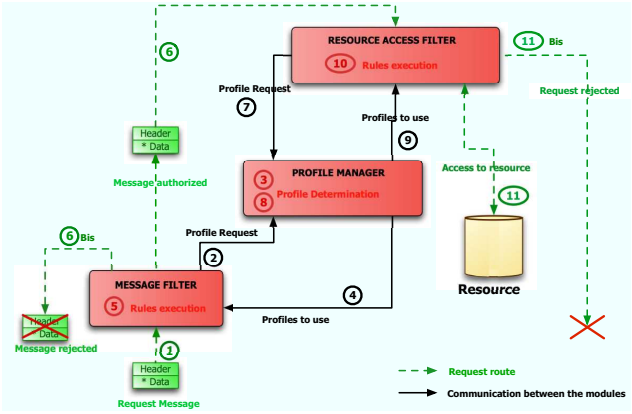

Figure 7. Request treatment process

If the message is not blocked (6 bis), then the message is authorized to enter the base (6), and the Resource Access Filter acts in the same way as the first filter: it asks the Profile Manager to determine the security policies to use (7), and the Profile Manager, with the contextual data it is able to gather, executes the *perl* functions (8) and notifies to the filter the profiles of rules to use (9). The Resource Access Filter enforces the policies (10) and gives access to the resource (if it is a resource request, or gives the services list if it is a service listing request, or executes the service if

it is a service execution request) to the entity which sent the request (11) or rejects the request (11 bis).

5. Use Case Study

In this section we study a use case very likely to occur in pervasive environments: a user would like to protect her resources and give access to some resources only if specific conditions on the context are fulfilled.

The scenario is simple (Figure 8): a laptop E_0, on which a PerSE Base is installed, shares a video sequence with its service ShareVideo. The base E_0 is situated in the room 502 in a building. Other users, each with a PerSE Base named E_1,...,E_7, would like to watch the video on their device. But the administrator of E_0 wants to share his video only with users equipped with a PDA and situated in the same room, as he doesn't trust the other rooms of the building. To this end, he has established a restriction on the use of the service: only users with a PDA and situated in the room 502 are authorized to execute it.

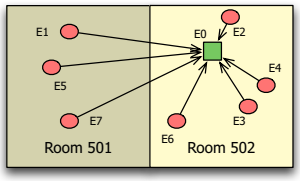

Figure 8. Use Case Scenario

The administrator has defined two types of policies on his base: communication policies, and resource access policies. Amongst all the communication rules defined, some do concern users E_1,...,E_7:

```
DO allow ON incoming perse_messages FROM Group1
    TO ShareVideo
DO drop ON incoming perse_messages FROM Us-E6
    TO ShareVideo
DO deny ON incoming perse_messages FROM Us-E7
    TO ShareVideo
```

The group named "Group1" consists of E_1, E_2, E_3, E_4, E_5. The user has chosen not to take into account any contextual information in these rules.

However, in the resource access policies, some rules are defined to give authorizations on ShareVideo to entities in a precise context:

```
Group1 CAN execute AND monitor ON ShareVideo
    IN CONTEXT neighbourhood_PDA
```

With the definition of these rules, the user has defined the context "neighbourhood_PDA":

```
CONTEXT neighbourhood_PDA
  WITH PRIORITY 0.5
  USING caller_and_local_base
  IS DEFINED BY
    location  OF callerbase IS equal to 'room 502' AND
    device OF callerbase IS equal to 'PDA'
```

In the use case, when the requests from the different users E_i arrive at the base E_0, encapsulated in a message, the Message Filter checks its rules, that are not context-dependent in this use case. The Message Filter can then enforce this policy without asking the Profile Manager. In the communication rules defined by the user, the messages from entities E_6 and E_7 are blocked. So the Message Filter blocks every communication from those users. The other users are allowed to submit their execution request to the service (Figure 9).

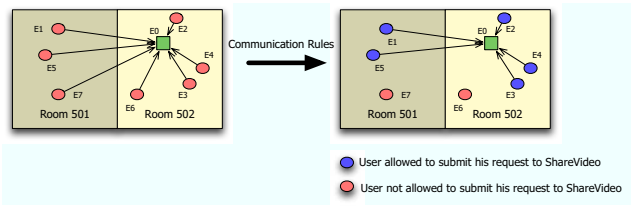

Figure 9. First filtering: Communication

The Resource Access Filter then receives each execution request from users Us-E1 to Us-E5. For each request, the Resource Access Filter asks the Profile Manager to determine the profile(s) to use, that is to say to determine which context(s) is valid. For users Us-E2, Us-E3, US-E4, situated in the Room 502 and equipped with a PDA, the context "neighbourhood_PDA" is true, so the rules of the corresponding profile are enforced, and users are given the authorization to execute the service ShareVideo. On the contrary, for user Us-E1, the context "neighbourhood_PDA" is not true, so rules defined in this profile are not enforced. Since no other rule can give the authorization to Us-E1, her request is rejected (Figure 10).

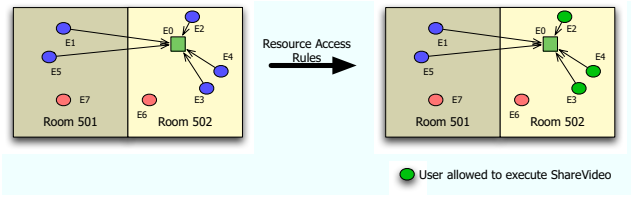

Figure 10. Second filtering: Resource Access

This is a simple example of the definition and application of a context-aware security policy, that enables a user to give access to her resources using contextual data.

6. Implementation, Evaluation and Results

Resources in pervasive environments are limited, and our two main priorities for this evaluation were efficiency in

terms of response time, and scalability of the rule-based policy definition and execution.

We implemented our infrastructure in C++. Our three components (Message Filter, Resource Access Filter, and Profile Manager) are composed of ten classes, and the compiled code occupies no more than 100 KB. A profile of rules is a XML file, and its size depends on the number of rules defined by the user. Typically, a 100-rules file is about 20 KB. In our tests, we will assume that a normal user will not define more than 1000 communication and resource access rules, divided into less than ten profiles. Each profile is divided into two files, one for communication rules, the other for resource access rules. We use other XML files too, to describe the profiles with metadata, but they each take about 4 KB only.

Context definitions are interpreted in *perl* language. We chose *perl* because we wanted an interpretable language, simple but efficient. A context definition is translated into a *perl* function and executed by a *perl* interpreter situated in the Profile Manager. The *perl* interpreter is a set of C functions, integrated and freely available in every *perl* distribution. Among the other XML files we mentioned earlier, one file is the description of the *perl* function, needed by the system to gather the parameters used by the function, which are context data. This description also contains the path of the *perl* file, the return values, *etc.*

The experiments have been done on a Powerbook 1.5 GHz Power PC G4, with 512 MB RAM. The response time (Figure 11) depends both on the number of rules and the number of profiles to which the rules belong. Even with 1000 rules and one profile, the worst case, the response time does not exceed 350 ms. For a nominal case, with 200 rules and 3 or 5 profiles, the response time is less than 50 ms, and even less with a 100 rules policy. The response time depends also on the position in the XML file of the resource access rule that grants the permission to the entity of the request. The response time will be lower if the rule is at the beginning of the file, whereas the position of the communication rule that allows the message or not does not influence our performances because our algorithm runs through all the communication rules.

This evaluation shows that the response time of our infrastructure enable the scalability of our system without a great loss of performances. We didn't optimize the algorithms or the code during the implementation, so better results can be obtained by spending more time on prototype development.

7. Contribution Summary, Discussion and Future Works

In this Section, we discuss here the main contributions and issues brought by our approach. Our solution enables a user to define precisely her privacy and security policy in a pervasive environment.

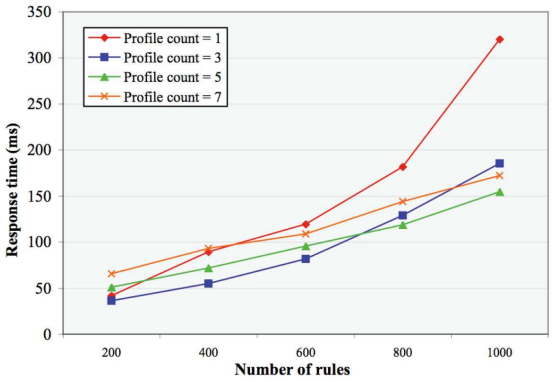

Figure 11. Request response time evaluation

The first main contribution is the infrastructure we propose, based on modular components, and which can be fully integrated in a broader pervasive architecture. This infrastructure guarantees the security and the privacy at different levels of the device: the first security level, the Message Filter, acts as a firewall on incoming and outgoing communication; the second security level, the Resource Access Filter, controls the access to the different resources present on the base. These two components are controlled by the Profile Manager, that decides which policy the two filters must enforce at the moment of a request.

Another main contribution is the context-aware security policies we introduced. As we saw in the related work, context-awareness in security and access-control for pervasive environment is complex to handle and rarely user-centered. That's why we introduce the languages and grammar for users to define their rules in a natural way, including their own perception of context. A user is then able to express a simple context-aware security policy.

A future work will integrate our security infrastructure in a wide PerSE environment, with many bases interacting and communicating with each other. Evaluations with more bases will give a precise overview of performances in real environments. This integration will lead to a fully working and secured pervasive environment, which will be continuously enhanced by the introduction of new services for users.

One of the main issue in our evaluation is context data gathering: we only assess the request response time, provided the Profile Manager has immediate access to all the context data it needs. However, in a real pervasive environment, these data are not always available easily and costlessly. Sometimes, a device has to ask another device (sensor, server) to obtain a data, but we didn't evaluate and take this major aspect into account.

An issue we have to work on is the integration of en-

cryption/deciphering algorithms in the Message Filter to enable secured communications between bases. To this end, we need to choose encryption algorithms and mechanisms (public, private or hybrid cryptography, session keys, static keys, *etc.*) that fit best to the constraints of pervasive environments, as most of encryption algorithms are expensive in term of resource.

As we saw in the presentation of communication rules, we trust the incoming message, that is to say we consider that the fields of the message which contains information about the sender are true, *i.e.* we do not make any authentication before an interaction with an entity. This strong hypothesis can not be made in a real environment, and it represents a real issue, as authentication in pervasive environment is difficult. We explore some ways to integrate authentication before the communication is established, like those works on authentication in pervasive environments [16], with digital certificates and local certification authorities, and trust propagation between these authorities.

8. Conclusion

We presented in this article a comprehensive framework for security and privacy in a pervasive environment. We first define a generic theoretical framework for context-aware access control. Our approach is based on a two-level control to the personal device of a user. Indeed, access to the device, then to its resources, are enforced using access rules defined by the user herself. This possible fastidious task is simplified thanks to declarative languages with which she expresses intuitively her wills. The originality of our approach relies in these possibilities to define precisely and easily security policies useful in a pervasive environment, thanks to a strong theoretical background and a large expressiveness of the rules definition languages.

Our approach has been actually successfully integrated in a pervasive environment, and evaluated both in terms of memory and computing consumption, proving its competitiveness and usability in a real environment.

References

[1] R. Beckwith. Designing for ubiquity: The perception of privacy. *IEEE Pervasive Computing*, 2(2):40–46, 2003.

[2] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

[3] P. Bihler, V.-M. Scuturici, and L. Brunie. Expressing and Interpreting User Intention in Pervasive Service Environments. *Journal of Digital Information Management*, 4(2):102–106, 2006.

[4] J. Clarke, M. Neubauer, and C. Hauser. Security and Privacy in a pervasive world - The Daidalos approach. *Eurescom Mess@ge magazine*, 2:8, 2005.

[5] M. Covington, M. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications. In *Proceedings of the 23rd National Information Systems Security Conference*, 2000.

[6] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 10–20, 2001.

[7] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[8] Y. Gripay, J.-M. Pierson, C.-E. Pigeot, and V.-M. Scuturici. Une Architecture Pervasive Sécurisée : PerSE. In *Proceedings of 3e Journées Francophones Mobilité et Ubiquité (UbiMob'06)*, pages 147–150, 2006.

[9] J. I. Hong and J. A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189, New York, NY, USA, 2004. ACM Press.

[10] J. Hu and A. C. Weaver. Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. In *Proceedings of the First Workshop on Pervasive Security, Privacy and Trust (PSPT)*, 2004.

[11] X. Jiang, J. Hong, and J. Landay. Socially-based modeling of privacy in ubiquitous computing. In *Proceedings of Ubicomp 2002*, pages 176–193, 2002.

[12] A. Kumar, N. Karnik, and G. Chafle. Context sensitivity in role-based access control. *SIGOPS Oper. Syst. Rev.*, 36(3):53–66, 2002.

[13] M. Langheinrich. *Personal Privacy in Ubiquitous Computing – Tools and System Support*. PhD thesis, ETH Zurich, Zurich, Switzerland, May 2005.

[14] U. Latif, J. B. D. Joshi, E. Bertino, and A. Ghafoor. A Generalized Temporal Role-Based Access Control Model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, 2005.

[15] S. Lederer, J. Mankoff, and A. K. Dey. Who wants to know what when? privacy preference determinants in ubiquitous computing. In *CHI '03: ACM Conference on Human Factors in Computing Systems*, pages 724–725, 2003.

[16] R. Saadi, J.-M. Pierson, and L. Brunie. Distrust certification model for large access in pervasive environment. *Journal of Pervasive Computing and Communications*, 2005.

[17] SAML. Security Assertion Markup Language. http://www.oasis-open.org/committees/security.

[18] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[19] WS-Policy. Web Services Policy Framework. http://specs.xmlsoap.org/ws/2004/09/policy/.

[20] XACML. eXtensible Access Control Markup Language. http://www.oasis-open.org/committees/xacml.

[21] G. Zhang and M. Parashar. Context-Aware Dynamic Access Control for Pervasive Applications. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2004)*, 2004.