Context-Sensitive Security in a Pervasive Environment

Research Report

Charles-Eric Pigeot, Yann Gripay, Jean-Marc Pierson, and Vasile-Marian Scuturici
INSA de Lyon
Laboratoire LIRIS
e-mail : { Charles-Eric.Pigeot, Yann Gripay, Jean-Marc.Pierson, Marian.Scuturici }@insa-lyon.fr

Abstract

Ubiquitous systems enable us to have an overview of what digital environments will look like in the future. The opportunities given by the 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 his 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 ubiquitous environments, integrated with an architecture, namely PerSE, in which privacy is a main concern and in which it 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 the devices surrounding the user must bring him useful services, depending on his needs, in a reactive way (after the user has expressed his needs), or in a proactive way (anticipating his needs). We believe that those two characteristics are essential to pervasive environments, as well as invisibility (the user must not be aware of the interactions between the devices) and non-intrusiveness in his personal life.

However, each user might want to control precisely how he interacts with his environment, *i.e.* which services or data he wants to share and in which context he wants this. To this end, he must have the possibility to define different context-aware access authorizations on his data: For example, a user may want to give access to his data only if he remains in a specific room of a building, and only to the users who are located in his visually accessible neighbourhood. This simple example may be much more complicated, but this kind of scenario is very likely to occur with the development of pervasive systems. A security and privacy system for pervasive environments must then enable the user to answer these 3 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 must be integrated between the different devices, but also into the devices to control the access to the data and services hosted locally. Moreover, the security integration must 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 context of the user into account in a satisfying way, though it must 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 the 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 ubiquitous 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 the related works. 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 works, and Section 7 discusses the contributions and future works and improvements.

2. Related Work

Discretionary Access Control (DAC) and Mandatory Access Control (MAC) were amongst the first access control solutions. They were quickly replaced by Role Base Access Control (RBAC) [4],[5]. In RBAC, roles are assigned to users, and the roles have permissions on objects. RBAC is particularly well adapted to organizations like hospitals, enterprise, 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 [4],[5]:

- $RBAC_0$: The basic model with users, roles and permissions
- $RBAC_1$: $RBAC_0$ with role hierarchies
- $RBAC_2$: $RBAC_0$ with constraints on roles, users, permissions
- $RBAC_3$: $RBAC_1 + RBAC_2$

Although these 4 models have proven their efficiency and simplified greatly the security management for structured organizations, it is obvious that RBAC, and its extensions developed to improve the model, like CBAC [6], an access control mechanism in which general associations between users and permissions are specified by the rules (or constraints) governing the access rights of each user, do not address all the issues related to pervasive and ubiquitous 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.* presents Temporal-RBAC [7] which addresses temporal needs on Role Based Access Control with the support of periodic role enabling and disabling and the introduction of time in RBAC model. This model was generalized by Joshi et. al [8] with GTRBAC (Generalized Temporal RBAC). The GTRBAC model includes a set of language constructs for the specification of various temporal constraints on roles, including constraints on their activation as well as on their enabling time, user role assignment and role permission assignment.

Time management and dynamicity is an important feature for ubiquitous 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 [12] with environment roles and GRBAC (Generalized RBAC) [13]. The authors proposed a generalization of the RBAC model that allows administrators to specify environment and context constraints through a new type of role, environment role. Those roles are based on context conditions as constraints, for their activation for instance. GRBAC also introduces Subject Roles and Object Roles and context information is used to make the access decision. While this model seems interesting by the introduction of context-aware roles and enable simple context-aware policies definition, the use of the context data is very limited, and the formalization and definition of the context are not satisfying, thus policies based on context can not express more complex aspects of context data.

Zhang and Parashar [9] present DRBAC, for Dynamic RBAC, which tries to address the dynamic access control needs for pervasive applications. Again, the authors add the context to the RBAC model, and the context data are collected by a "Context Agent". In DRBAC, roles change as the context changes, and each user has a context agent which detects context change. These changes trigger transitions between the roles. This model does not address important issues about dynamic and distributed access control, the main issue of this model being that a Central Authority is needed to manage the role hierarchy and the transitions between the roles. This centralization is not adapted to very distributed environments, like pervasive environments.

Another extension to RBAC was proposed to address context issues, OrBAC [14] and later Multi-OrBAC [11]. OrBAC introduces context as a new entity to specify the circumstances in which the organisation grants permissions on objects. However the use of the context of the user in his environment, is limited and unclear, whereas it should be a central point, especially in pervasive environments. The authors do not provide any information on how they gather context information, which context information is used, and they do provide a context model reusable. Furthermore, like RBAC, this model is well adapted to organizations, hence the application to pervasive environment is very limited. That's why the authors extended their model to Multi-OrBAC, meant to address multi organization issues. In Multi-OrBAC, each role and permission is valid in a specific organization. This model is more adapted to distributed and heterogeneous systems, although it still does not provide a satisfying context-aware access control model, since the context part is the same as in OrBAC.

An promising work for context aware access control for distributed Healthcare applications is presented by Hu and Weaver [21]. The authors provide good and useful definitions and formalization of the context, but their model is not far from DBAC or OrBAC: Rules consist of permissions on objects to users in a specific context. The implementation of their access policy use WS-Policy. 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 [10], for Context Sensitive RBAC, which enable RBAC to enforce security policies dependent on the context of the operation attempted, the user and the object. However, the authors do not provide any satisfying context model, and the 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 of access control for pervasive environments, but most of them don't solve all the problems. Other works propose an integrated and secured architecture for pervasive environments in which privacy is the main requirement.

Langheinrich [1] describes a secured pervasive architecture named pawS, in which devices of the environment announce user data collection to a privacy assistant carried by the user on his mobile device. If the two devices can't agree on the negotiation about the data collection (the user preferences are described in a privacy policy), the user declines the usage of the service. The privacy policy of the user is described with P3P [20], a labeling protocol from the Web. While this approach is centred around privacy, access control is very limited, and the description of the user privacy policy becomes very hard if the number of entity increases.

An other approach is proposed by Hong and Landay with their solution, Confab [2]. Confab is an infrastructure for facilitating the development of privacy-sensitive ubiquitous computing applications. The authors gathered requirements for Confab through an analysis of privacy needs for both end-users and application developers. Confab provides several customizable privacy mechanisms as well as a framework to extend privacy functionality. levels and privacy needs. This infrastructure enables the administrator to define metadata on the data to protect, metadata related to the privacy: number of utilisation, time of life etc. This approach deals only with the use of personal data, but not the access control at all. Moreover, context awareness is not addressed here.

The Daidalos approach [3] is based on virtual identities of the user, each identity containing a subset of the user data. Identities are changed and generated upon the context, and when two entities (user and service for instance) want to cooperate, they first need to agree on the data to exchange, as in pawS [1]. If they agree, a new identity is generated or a satisfying existing one is used. This architecture handles context, but the privacy preferences are defined in a static way, and can not be changed easily.

Our policies must 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 [18], WS-Policy [17], and SAML [19]. SAML defines an XML framework for exchanging authentication and authorization information for securing Web services, and relies on third-party authorities for provision of "assertions" containing such information. However, SAML itself is not designed to provide support for specifying authorization policies. XACML is an XML framework for specifying context-aware access control policies for Web-based resources.. WS-Policy is used to describe the 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 indeed be used to define an access control policy.

Herzberg *et al* propose TPL [16], Trust Policy Language, a XML-based language to define policies using well formed XML document. The main purpose of the Trust Policy Language (TPL) is to map entities to roles, using well defined logical rules described in XML. Finally, Netegrity [15] has proposed S2ML, a security services markup language that provides mechanisms for describing security models with XML and for sharing security information about transactions and end users between companies.

Those approach are promising and some of them tend to become standards in security policy definition, in particular XACML and WS-Policy. They enable standardization in policies definition, and they are very powerful and expressive.

We need 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. Indeed, WS-Policy was too complex and too expressive for our needs and for our simple rule-based system, as it is first designed for Web Services policies. We could have used XACML, which is very close to our XML syntax and process, but the XML files generated with XACML are much heavier than with our XML files, because it is more powerful. As resources are limited in pervasive environments, we chose to make our XML syntax as simplest as possible. Anyway, the translation of our rules to XACML rules is very easy and can be done quickly if there is a strong need of interoperability.

We decided to propose to the user a declarative language to define his rules, because we do believe that even if XML is really simple compared to other languages, it is far from being natural to a basic user. Thus, we chose to define very simple but powerful declarative languages to this end. We'll see in the following sections our security model, and how we used this model to create a secured infrastructure for pervasive architectures, infrastructure in which security and privacy is handled at multiple levels and in which access control is totally context-aware.

3. Theoretical Framework and Definitions

Here we present our definitions and framework for our model.

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

Base: A meta-services running on a pervasive device, enabling it to share his local services and resources (local-context). Each base is in charge of communications with the 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 picture, a file etc. A service is hosted on a base. A resource can belong to a user or to a base.

Entity: We define an entity as a generic term for either a base, a resource, or a user, *i.e.* the 3 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, whatever their nature might be (base, user, service).

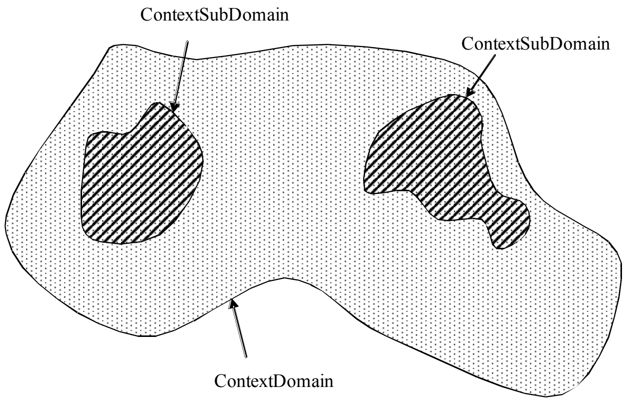

Figure 1 : Context Domain and Subdomains

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

Context SubDomain: A set of states included in the *ContextDomain.* (*ContextSubDomain* ⊆ *ContextDomain*)

Rule support: the context sub domain used to define the rule.

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, and this action has to be realized in a certain context. A communication rule can be represented by the tuple:

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

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

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

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 the rule support of r_i, then:

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

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 the rule support of r_i, then:

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

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 profile of communication and resource access rule defined by a user to protect his 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.

For conflicts between the rules: 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 2 actions are applicable in the mean time, then the action α_1 will be applied.

For conflicts between the profiles: We define the function Priority Φ:

$$\Phi : P_C \rightarrow [0, 1]$$
$$p_c \rightarrow k$$

The value of this function for a profile represent 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 3 modules, is integrated in an existing pervasive environment named PerSE (Pervasive Service Environment) [22] 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 is the one which decides which policy to use, depending on the context at the moment of the request. Indeed, we believe that a strong requirement for a security system for a pervasive environment is the context-awareness. The context of the user is the basis for every ubiquitous environment, and we can easily understand that a user might want to change his privacy policy in different contexts. That's why we introduced in our model, and in our policy, the concept of context. We'll see later how context is central in our infrastructure.

Moreover, we would like to create totally proactive security policies, that is to say security policies that can be defined by the system without any intervention of the user. However, it is very difficult to design such policies, so we designed context-aware policies, that, once defined by the user, adapt themselves to the context, *i.e.* where context is central and controls their applicability. By defining context-aware policies, we then reduce the interaction with the user.

Invisibility, one of the most fundamental characteristics of pervasive environments, makes difficult for user to evaluate the concepts of privacy and security in these environments.

Some works have tried to study the perception of the user regarding security and privacy threats in ubiquitous environments. Beckwith [24] 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 place and 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.

Key *et. al* also studied this aspect of pervasive environments [25], and their works clearly reveal the fact that the quantity of personal information given in response to a request depends both on the identity of the emitter of the request and on the context in which the user stands. Again, all the users do not define context in the same way, and each user has his own parameters to define it: One will prefer to use his location, another will use the temperature of the room etc. The conclusions are nearly the same in other similar studies [26], [27].

For these reasons, we decided to give 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 he will use and which constraints he 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 represents our vision of a pervasive environment, user-oriented and in which the user has access to services on the different surrounding devices by expressing an intention. Moreover, this platform has to be 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 his local services and resources (local context). The PerSE Base is in charge of communications with other bases, in order to run in a smart and optimized way distributed services.

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

In order to respond to user needs, a modelling of his intention is necessary. The PsaQL language [22] enables the user (or an application) to express his intention (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.

The PerSE architecture (Figure 2) is composed of 3 layers, corresponding to the 3 main functionalities of the Base : Communication, Environment and Action. Between and within these layers, we integrated our security infrastructure, composed of 3 modules.

4.2.1 The 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 the 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 the user who wants to interact with the base, especially to start *partial actions*. The Messenger is in charge of the communications between the bases, communications based on specific messages.

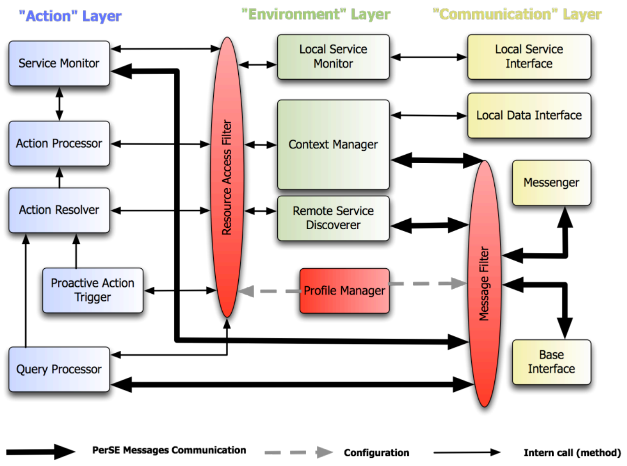

Figure 2: The PerSE architecture

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 3). 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.

By using these information stored in the message structure, the Message Filter can decide to stop the message or to let it pass. 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 Enforcement Point (PEP) and a Policy Decision Point (PDP), since it renders authorization decision, and performs access control. We'll see in the next sections how the user can define such policies.

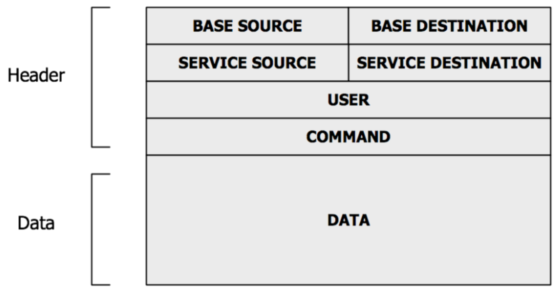

Figure 3: The PerSE Message structure

4.2.2 *The Environment Layer*

The Environment Layer manages the local knowledge of the base on its environment. The Local Service Monitor is the module which manages all the local services available. 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 services available on distant bases.

This layer manages and has the 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 since it makes decision on access control.

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 context of the user, 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 cannot be optional.

4.2.3 *The Action Layer*

The Action Layer is intended to gather the request from the users or applications, and to execute actions to answer to these requests. The Query processor receives PerSE messages, containing the 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 an 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 the services.

4.3 Rule-based Communication and Access Control

The user is the key actor of our security system. He is to define the security policies that will be applied on his 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. These languages are interpreted by the system in a XML-based language used to represent the rules and readable by the policies enforcers. They can also be translated easily in a standard language for privacy policies, like XACML [18] or WS-Policy [17] for improved interoperability.

The Communication Rules syntax is described by the following BNF grammar (Figure 4):

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

Figure 4: Communication Rule BNF grammar

Here are a few examples of Communication rules based on this grammar (Figure 5):

```
    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
```

Figure 5: Examples of Communication Rules

The first rule specifies that every incoming message using the protocol TCP should be blocked if the context defined as "neighbourhood" is valid (see part. 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 etc.

The action part of the rule describes which action to execute on the incoming or outgoing message: Allow, to let pass the message, deny, to stop the message and to notify the sender of this failure, and drop, to 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. The user can specify to do actions on communication using a certain type of network protocol, like udp, tcp etc.

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 entity (the groups are defined by the user simply by gathering entity). To each known entity(base, user, service) is assigned a unique identifier. This identifier is used for example by the user who wants to log in on a common base, and each message sent during this user session will have the user field filled in with the user identifier. The first three fields are filled with the base, service or user sending the message. The message is composed by the Messenger module of the PerSE architecture, and the fields are filled in just before the message is sent. In the communication rule, the user specifies the sender and receiver identifier to filter. We'll see later how the incoming message is analyzed to get those data stored in the fields of the message. However, at 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 and receiver part specify a group instead of a single entity, then the system has to determine to which group the sender of the message belong before making his decision.

These two parts offer the user the opportunity to filter communications between two distinct entities, and then to define a precise security policy.

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 from the rule, the rule is applicable whatever the context may be. In this case, the rule is very similar to an simple firewall rule, and the Message Filter becomes nothing more than a firewall. If the context part is defined, then the Message Filter can be compared to a context-aware firewall.

As we can see, these rules are expressive, and they enable the administrator to express and to define easily 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 makes easier for the administrator to express his preferences.

In the same way we defined a grammar to express communication rules, we defined another one to describe Resource Access Rules (Figure 6), which are higher level rules used to control the access to local resources (data, services...).

The Resource Access Rule defines permissions for entities or group of entities on a resource or a group of resources in certain contexts.

When a request from a user comes at the base, and after the Message Filtered has decided whether the message is allowed to pass or not, the Resource Access Filter, depending on the resource access rules, provides the response 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 6: Resource Access Rule grammar

In the examples (Figure 7), the first rules specifies that the service identified ad "Se-098" has the permission to modify and delete the resource "img18.jpg" if the context defined as "neighbourhood" is valid. We can see that this language is high level and 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
```

Figure 7: Example of Resource Access Rules

4.4 Context and Profiles

A strong requirement we had identified clearly before the conception of our model was the context awareness. In a pervasive environment, a security policy defined by the user must depends on the context of the user, and on the information of the context that seems important to him among the huge quantities of contextual data a pervasive environment can gather.

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

To define a context, we created a simple declarative language (Figure 8), 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 he 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 the 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 his 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, are high-level expressions of contextual parameters), more understandable for a basic user.

```
<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 8: Context definition grammar

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.

The definition of a context uses many parameters, called contextual parameters, which correspond to a type of information on the context. These parameters are taken from the context of the 2 entities implied in the request: The caller base and the local base. For instance, the contextual parameters used to define a context can be the temperature, the lightning, the location,... The user can specify a relation that links the parameter to the value : equal to, superior to, inferior 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 9: Example of a context definition

In the example (Figure 9), the user defined a context named “trusty”, and the context will be “trusty” when the location of the caller base is the room name “room 203”, and when the trust mark (which we consider as a parameter of the context we can calculate locally depending on an history of the interaction with the entity), is superior or equal to 0.7. The location is a parameter from the context of the local base, and the trust mark is a parameter of the caller base, that is what is specified in the definition. This is only an example among the numerous contexts the user is able to define with this syntax.

As we saw, a rule is valid in a context, or in a limited number of contexts, as specified with the last parameter, which correspond to the name of the context defined by the user.

All the rules applicable in the same context are gathered in profiles. Hence, a profile is a set of rules with the same support, and corresponds to a precise context. In other words, a profile is a contextual security policy. A potential problem that can occur is that at the time of a request, more than one context are valid, that is to say that the contexts of the caller base and the local base correspond to many contexts defined by the user. At this moment, more than one security policies are applicable for the request, and some conflicts can 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 time *t*, then 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 first defined.

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 defines himself 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.

In the implementation section, we'll see how and why we used the *perl* language to implement those context definition and make them efficient and usable by our system.

Below is the algorithm (Figure 10) used to resolve conflicts between communication rules and profiles:

```
// $P_g$ : Set of applicable communication profiles at the moment of the request
// Request: incoming message containing information about the sender of the message and the // request
// decided_rule: rule that will be applied to the incoming request

CommunicationRule_choice (in : $P_g$,  in : Request, out : decided_rule)
Begin
        ApplicableRules  tab;   // structure to store for each profile the
                                // most prioritary rule
        Communication_rule decided_rule;

        for each p in $P_g$
        CommunicationRule priority_rule = p.firstRule    // for the current
                                // profile, the most prioritary rule.
        for each $r_c$ in p
            // we check if the rule is applicable depending  on the request
        // sender and recipient
                if  $r_c.\sigma$  = Request.sender and $r_c.\delta$  = Request.destination then
                        // checking of the rule priority
                        if $r_c.\alpha$ > priority_rule.$\alpha$  then
                                priority_rule = $r_c$
                        end if
                end if
        tab.add ($r_c$, p.name) // adding of the rule and the profile
                        // corresponding
        priority_profile = tab.firstElement
        // we run through the vector of profile and we determine which
        // profile is most prioritary with the $\Phi$ value for each profile. The
        // rule applied will be the one of the most prioritary profile
        for each element in tab
                if  $\Phi$(element.p)  >  $\Phi$ (priority_profile)  then
                        priority_profile = element
                end if
        decided_rule = priority_profile.$r_c$;
        return decided_rule ;
end
```

Figure 10: Communication Rule choice algorithm

Let N be the number of profile applicable for a request, and p the average number of rules per profile. Then the algorithm has a complexity of N*p + N. The memory occupation of such an algorithm is about the size of the profiles, which itself depends on the number of rules. The other data structure we use do not cost much memory.

4.5 Internal Functioning

We describe here the internal functioning of our 3 security modules of our infrastructure.

The internal functioning of the Message Filter is quite simple (Figure 11). When an incoming message arrives at the Message Filter, from the Messenger or the Base Interface, the encryption / decryption module deciphers the message if it is coded. We discuss this aspect in our discussion part. Then the message is transmitted to the Header Decomposer, which decompose the header of the message to extract the useful information about the sender and the recipient: the base, the user or the service. These information are then transmitted to the Profile Processor, which is the main sub module of the Message Filter. It is responsible for deciding whether the message must be blocked or not. He asks the Profile Manager the security policies that are applicable at the time of the request. With the policy enforcement, it applies the algorithm to resolve the different conflicts (if there is more than one policy, or if there is more than one rule applicable in a policy), and then decides, using the information of the header, to let pass the message or not. If the communication is denied, the Deny Notifyier is in charge of notifying the failure of the communication to the sender.

For an outgoing message, the process is reversed but similar.

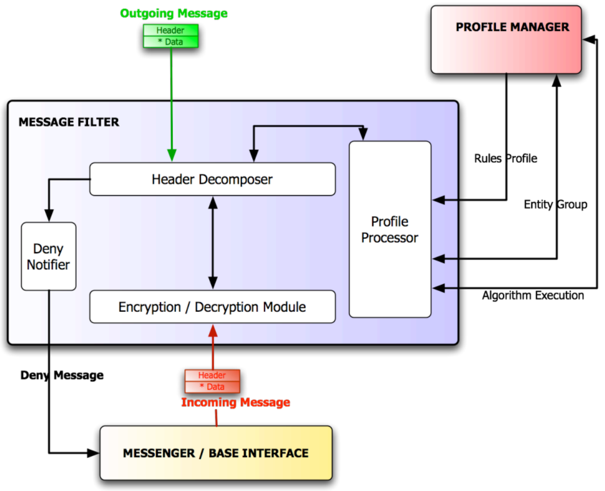

Figure 11: Message Filter

3 kinds of request can arrive to the Resource Access Filter from the Action Layer (Figure 12) :

- Context request, that is to say request asking for local or distant data, or user data stored on the base.
- Service listing request, when a distant module, base, service or user needs to know which services are available locally
- Service execution request, to ask for a service to be launched

The different requests arrive at the Request Dispatcher, and, depending on their nature, are redirected on the 3 sub modules in charge of the request treatment: The Context Provider, for the context requests, the Service Provider, for the service execution requests, and the Service Listing, for the service listing requests. Then, these 3 components call the Profile Processor, which has the same role and the same functioning as in the Message Filter. When the Profile Processor knows which policies to enforce, it then decides if the action demanded is authorized or not.

If the request is authorized, the 3 submodules ask to the corresponding components of the Environment Layer to gather the data requested or to execute the service demanded.

Context events, which are a special type of communication, are also filtered. They aims at keeping informed the Proactivity module of the Action Layer, of important events or changes in the context. The Context Event Transmitter handles these alerts, and transmit them if it is authorized.

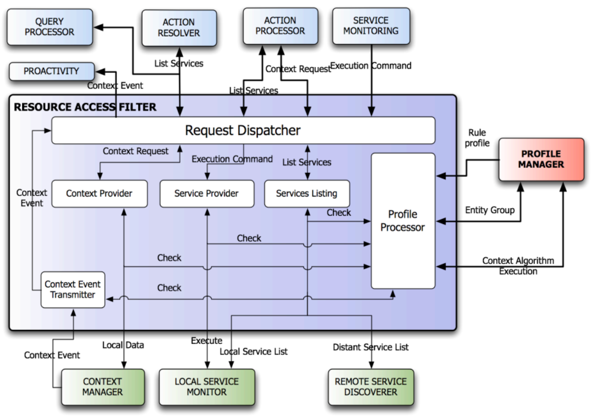

Figure 12: Resource Access Filter

The Profile Manager (Figure 13) is the main component of the security infrastructure, since it controls and provides the security policies that can be applied at the time of a request.

The Profile Communication handles all the communications between the Profile Manager and the two filters. It receives the requests asking for the policies to apply, and send the references to the existing policies to use, when the request has been treated. The contexts defined by the user are implemented as *perl* functions, and these functions are maintained by the Function Manager, which both registers the functions and knows the descriptions of these functions (input parameters etc), necessary to the *perl* interpreter situated in the Profile Decider.

When the Profile Decider receives the request notifications, it demands the existing *perl* functions, with the meta data on these functions, to the Function Manager, and the perl interpreter executes these functions. To do so, contextual data are needed about the entities who take part in the transaction. These contextual data are provided by the Entities Context Manager, which aims at maintaining a database of every contextual data available on the entities around and on the user. To this end, it is in constant communication with the Context Manager of the Environment Layer of the PerSE architecture. The Profile Decider is then able to tell which context, then which profile of rules, is applicable. More than one context can be valid at once, then all the valid profiles are transmitted and the two filters will decide which context has the priority using the algorithm described earlier. The Entities Manager manages the different groups defined by the user, and answers the request from the two filter to know in which group is an entity.

An history is updated for each request, containing information on the request and its response. These data can be used and analyzed with data mining algorithms, to find some useful information to enhance the performances on the response. This analysis has not been currently implemented but the logging is.

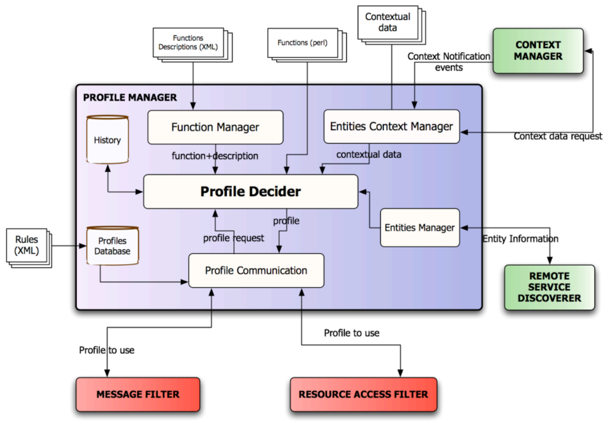

Figure 13: Profile Manager

4.6 Summary

To make easier the understanding of our solution, we propose an outline (Figure 14) 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 the Profile Manager the profiles (the security policies) to enforce (2). The Profile Manager asks his internal modules to gather information on the 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, enforcing the given policies and applying the conflicts resolution algorithms (5).

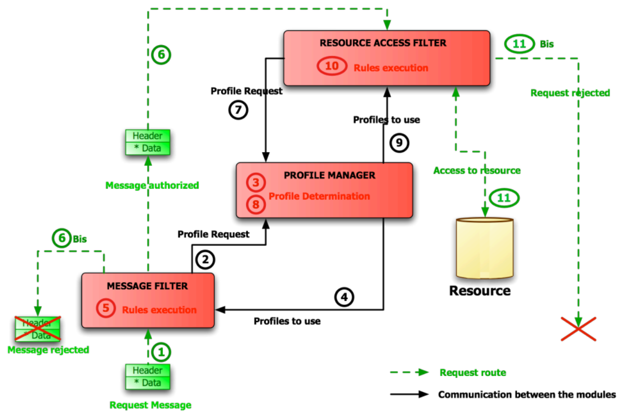

Figure 14: Request treatment process

If the message is not blocked (6 bis), then the message is authorized to enter the base (6), and the Resource Access Filter acts 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 the filter the profiles of rules to use (9). The Resource Access Filter enforces the policies (10) and gives the access to the resource (if it is a resource request, or gives the services list if it is a listing request, or executes the service if it 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 want to protect his resources and give access to some resources only if specific conditions on the context are fulfilled.

The scenario is simple (Figure 15): A laptop E_0, on which a PerSE base is installed, can share a video sequence with the 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 the users equipped with a PDA and situated in the same room, since 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 the users with a PDA and situated in the room 502 are authorized to execute it.

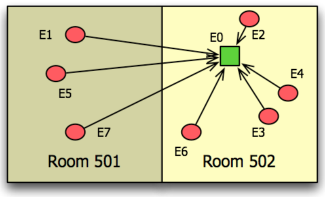

Figure 15: Use Case Scenario

The administrator has defined two types of policies on his base: communication policies, and resource access policies. These policies consist of rules, gathered in profiles (a set of rules for which the context of application is the same).

Amongst all the communication rules defined, some do concern the users E_1,... E_7 (Figure 16).

```
...
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
...
```

Figure 16 : Communication rules defined by the user

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 (Figure 17) are defined to give authorizations on *ShareVideo* to entities in a precise context.

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

Figure 17 : Resource Access rule defined by the user

With the definition of these rules, the user has defined the context "neighbourhood_PDA" (Figure 18).

```
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"
```

Figure 18 : "neighbourhood_PDA" context definition

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 if some rules are defined without the contextual parameter, that is to say if rules applicable whatever the context would be are defined. If so, the Message Filter can enforce this policy without asking the Profile Manager which contextual policy to use. In the communication rules defined by the user, the messages from the 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 19).

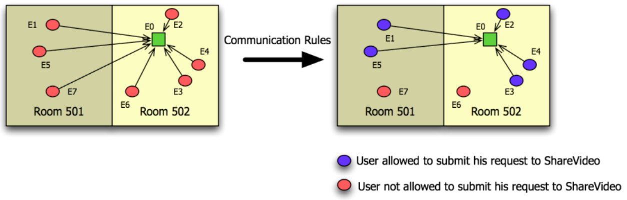
Communication Rules
E1
E0
E2
E4
E5
E7
E3
E6
Room 501
Room 502
User allowed to submit his request to ShareVideo
User not allowed to submit his request to ShareVideo

Figure 19: First filtering : Communication

The Resource Access Filter then receives each execution request from the 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 the 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 profile "neighbourhood_PDA" are enforced, and the users are given the authorization to execute the service *ShareVideo*. On the contrary, for the user Us-E1, the context "neighbourhood_PDA" is not true, so the rules defined in this profile are not enforced, and since no other rule can give the authorization to Us-E1 to execute the service, the request of Us-E1 is not allowed (Figure 20).

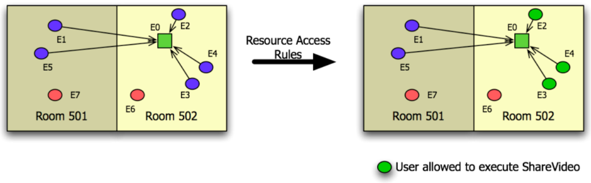

Figure 20: Second filtering: Resource access control

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

6. Implementation, Evaluation and Results

Resources in pervasive environments are limited, and our two main priorities for this evaluation were the efficiency in terms of response time and memory occupation, and the 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. We used our own XML based language to describe the interpreted and executable rules. In order to interpret the rules expressed with the defined grammar, we used a light XML parser, called *tinyXML*.

```
  <communication_rule id="1">
          <action>deny</action>
        <communication>
  incoming_perse_messages
        </communication>
          <protocol>tcp</protocol>
          <sender>all</sender>
          <receiver>localhost</receiver>
  </communication_rule>
```

Figure 21: XML-translated communication rule

A profile of rules is a XML file (Figure 21), 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.

The definitions of the context 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, 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, is a description of the *perl* function, needed by the system to gather the parameters needed by the function, which are context data. This description also contains the path of the *perl* file, the return values etc.

The memory occupation of our system is not a real issue, since it won't exceed 400 KB in most of the cases.

The experiments have been done on a Powerbook 1,5 GHz Power PC G4, with 512 MB RAM.

To evaluate execution time, we divided our experiments into two parts: The first part was about rule interpretation in function of the number of rules involved, and the second part was the evaluation of a request response time. In a real environment, the first part, that is to say the rule and context interpretation, would be realized once during the initialisation of the base, whereas the second part, the response to request, would occur at anytime.

The Figure 22 shows the results of the rule interpretation, in function of the number of rules. The 4 curves corresponds to the number of profiles in which the rules are distributed. Rule interpretation is time-consuming, and depending on the number of profiles, it can evolve in an polynomial way.

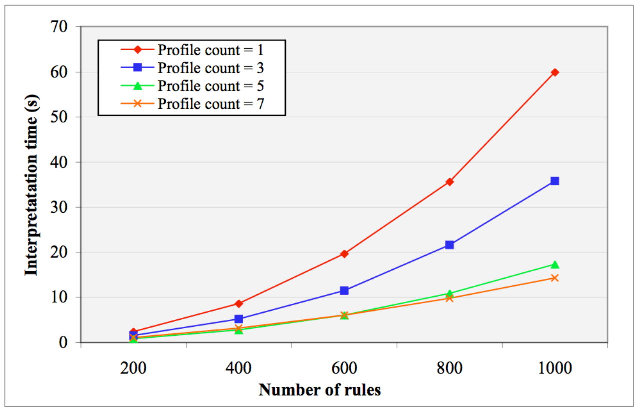

Figure 22: Rule interpretation time evaluation

We observe through these experimental results that, the interpretation of 1000 rules can take more than a minute if the rules belong to a single profile. But this time is greatly reduced as the number of profiles increases, to reach a time of about 20 seconds with 5 or 7 profiles. Hence, if the user defines more than one context or profile, which is likely to happen in a pervasive environment, the interpretation time is very acceptable.

Anyway, we conducted the evaluation with a large number of rules. In a real scenario, we can assume that a simple user would not define more than 200 rules, with 3 or 5 contexts, since a policy with more than 200 rules would quickly become hard to maintain and to understand for a simple user. With these assumptions, the interpretation time is very low, about 2 or 3 seconds. Moreover, as we said, the loading of all the rules is meant to happen once at the initialisation of the base. Once done, the files are stored in the mobile device memory, so the interpretation times given are very acceptable when done only once.

The second evaluation we made was on the response time of a request (Figure 23). The response time 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.

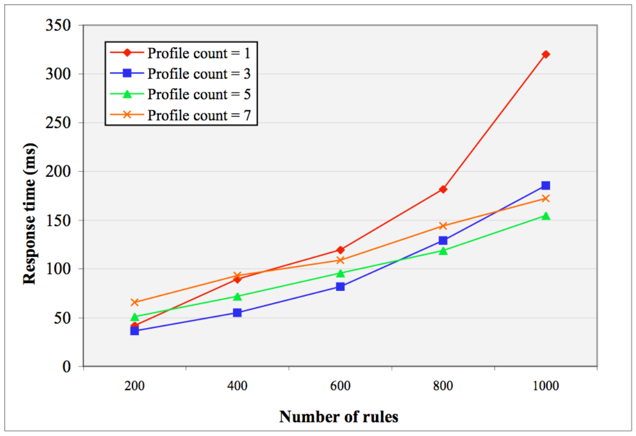

Figure 23: Request response time evaluation

These evaluations show that the interpretation and 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, and we believe that we could obtain better results with more time on the development.

7. Contributions Summary, Discussion and Future Works

We discuss here the main contributions and issues brought by our approach. Our solution enables a user to define precisely his privacy and security policy in a pervasive environment.

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 the incoming and outgoing communication of the device. 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 the request.

Another main contribution is the context-aware security policies we introduced. Actually, two aspects of these policies are innovative and interesting. The first one is the context-awareness of the rules of the policies. This context-awareness is essential for a security policy in a pervasive environment, since the context is the base of those environments. As we saw in the related works, context-awareness in security and access-control is an issue rarely addressed, and when the context is taken into account in the policies, it is not satisfying and simple for the user. The second important aspect is the languages and grammar we introduced for the user to let him define his rules very easily. Furthermore, we introduced an other simple language to enable him to express what a context is, and which parameters to take into account in the definition of a context. The user is then able to express in which context a rule is applicable, and then to define a simple context-aware security policy.

Future works will integrate our security infrastructure in a wide PerSE environment, with many bases interacting and communicating with each other. Our model has been implemented in a PerSE Base, but we will realize evaluations with more bases to have a precise overview of the real performances. This integration will enable us to have a fully working and secured pervasive environment, which we will continuously enhance by the introduction of new services for the user.

One of the main issue in our evaluation is that we only assess the request response time, provided the Profile Manager has access very quickly 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, or sensors, or servers, to obtain the data, and we couldn't evaluate and take this major aspect into account in our evaluation. The evaluation times could be then strongly increased, depending on the data needed, on the availability of these data, and on the communication time between the requester and the provider of those.

An issue we have to work on is the integration of encryption/ deciphering algorithm in the Message Filter to enable a user to encrypt his communication and the message filter to decipher the incoming or outgoing communication. To this end, we have to study the encryption algorithm and mechanisms (public or private or hybrid cryptography, session keys, static keys etc..) that fit the best to the needs of pervasive environments, as we know that resources are very limited are that most of the encryption algorithm are costly.

To conclude with the technical issues, we did not used and developed the History module in our implementation. This module could be used to optimize the response time of our system, as it would act as a cache, and would be interrogated before trying to resolve the request, to know if the results of the request are not already known. With adequate and efficient algorithms, we could obtain significantly reduced response time. This aspect will be treated in future studies.

About the approach itself, our system focus on the access control to the data and the protection of the user data, but we do not propose any solution for the use of the data, once the access has been granted. In future works, we will experiment such approaches, like the use of metadata in [2], to specify properties on the data itself, like its number of use allowed, its lifetime, etc.

Moreover, we have to explore means to gather data in groups. Permissions would then be granted on groups of data instead of a single data as we do on our rules. This improvement would make the resource access policy specification easier and faster. The criteria on which the resources would be gathered are yet to define, and we will study a means to gather the resources automatically instead of manually by the user himself. A semantic gathering would be promising, since it would help to formulate and treat more complex and semantic requests.

As we saw in the presentation of the 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 of the message, 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, since 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 [28], 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 the user. Indeed, access to the device, then to its resources, are enforced using access rules defined by the user himself. This possible fastidious task is simplified thanks to declarative languages with which he expresses intuitively his 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.

The 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] Langheinrich M., *Personal Privacy in Ubiquitous Computing : Tools and System Support*, Ph.D. thesis, University of Bielefeld, 2005.

[2] Hong J., Landay J.A., *An architecture for privacy-sensitive ubiquitous computing.* In Proceedings of MobiSYS '04 : pages 177–189. ACM Press, 2004.

[3] Clarke J., Neubauer M., Hauser C., *Security and privacy in a pervasive world – The Daidalos approach*. Eurescom Mess@ge magazine, Issue 2/2005, page 8, 2005

[4] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security (TISSEC), 4(3):224-274, 2001

[5] R. S. Sandhu, et al. "Role-Based Access Control Models", IEEE Computer 29(2): 38-47, IEEE Press, 1996

[6] Wee-Yeh Tan: Constraints-based Access Control. DBSec 2001: 31-44

[7] Bertino, E., Bonatti, P. A., & Ferrari, E. (2001). *TRBAC: A Temporal Role-based Access Control Model*, ACM Transactions on Information and System Security, 4(3), 191-233.

[8] J. B. D. Joshi, E. Bertino, U. Latif & A. Ghafoor, *Generalized temporal role based access control model* (GTRBAC) (Part I) - specification and modeling. Technical report, CERIAS TR 2001-47, Purdue University, USA, 2001.

[9] G. Zhang and M. Parashar. Context-aware dynamic access control for pervasive computing. In 2004

[10] A. Kumar, N. Kamik, and G. Chafle. Context Sensitivity in Role-based Access Control, ACM SIGOPS Operating Systems Review, pp. 53-66, July, 2002.

[11] A. Abou El Kalam, Y. Deswarte "Multi-OrBAC: un modèle de contrôle d'accès pour les systèmes multi-organisationnels" 3rd *Security of Information Systems (SSI)*, Seignosse - Landes, France, 6-9 juin 2006.

[12] M. J. Covington, W. Long, S. Srinivasan, A. Dey, M. Ahamad, and G. Abowd. *Securing context-aware applications using environment roles*. In Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, May 2001.

[13] M. J. Covington, M. J. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications. In 23rd National Information Systems Security Conference, Baltimore, MD, October 2000

[14] A. Abou El Kalam, R. Elbaida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, G. Trouessin "ORBAC : un modèle de contrôle d'accès basé sur les organisations", Cahiers francophones de la recherche en sécurité de l'information, Numéro II, 1er trimestre 2003, pp30-43

[15] Netegrity. S2ML : The XML standard for describing and sharing security services on the internet. Technical report, 2001.

[16] A. Herzberg, J. Mihaeli, Y. Mass, D. Naor, and Y. Ravid. *Access Control Meets Public Key Infrastructure*, Or: Assigning Roles to Strangers. In IEEE Symposium on Security and Privacy, Oakland, CA, May 2000.

[17] WS-Policy, http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf

[18] XACML (eXtensible Access Control Markup Language), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[19] SAML, (Security Assertion Markup Language), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=securit y

[20] Cranor L, Langheinrich M, Marchiori M, Presler-Marshall M, Reagle J, The platform for privacy preferences 1.0 (P3P1.0) specification. MIT/World Wide Web Consortium, available at http://www.w3.org/TR/P3P

[21] J. Hu, A. C. Weaver, "Dynamic, Context-aware Security Infrastructure for Distributed Healthcare Applications", Proceedings of First Workshop on Pervasive Security, Privacy and Trust (PSPT), August 26, 2004.

[22] Bihler P., Brunie L., Scuturici V.M.: *Modelling User Intention in Pervasive Service Environments*, EUC 2005, Japan, Dec 2005, LNCS 3824 Springer, pp. 977-986

[23] Gripay Y., Pierson J-M., Pigeot C-E., Scuturici V.M.: *Une architecture pervasive sécurisée: PerSE*,

[24] R. Beckwith. *Designing for ubiquity: The perception of privacy*. Pervasive Computing, 2(2):40--46, April-June 2003.

[25] A.K. Dey, S. Lederer, J. Mankoff, *Who Wants to Know What When? Privacy Preference Determinants in Ubiquitous Computing* , Privacy Preference Determinants in Ubiquitous Computing. in Extended Abstracts of CHI 2003, ACM Conference on Human Factors in Computing Systems. 2003. Fort Lauderdale, FL

[26] Jiang, Hong, Landay, Socially-Based Modeling of Privacy in Ubiquitous Computing, UbiComp 2002, Springer LNCS 2498, pp 176-193, 2002

[27] M. Langheinrich, *Personal Privacy in Ubiquitous Computing :Tools and System Support*, Ph.D. thesis, University of Bielefeld, 2005.

[28] R. Saadi, J-M. Pierson, L.Brunie, *Distrust Certification Model for Large Access In Pervasive Enviroment*, Journal of Pervasive Computing and Communications. 2005.