Une architecture pervasive sécurisée : PerSE

Yann GRIPAY, Jean-Marc PIERSON, Charles-Eric PIGEOT, Vasile-Marian SCUTURICI
Laboratoire LIRIS – UMR 5205, INSA de Lyon
7 avenue Jean Capelle, 69621 Villeurbanne cedex
{ Yann.Gripay, Jean-Marc.Pierson, Charles-Eric.Pigeot, Marian.Scuturici }@insa-lyon.fr

RESUME
Les opportunités ouvertes par les systèmes pervasifs sont extrêmement nombreuses et prometteuses. Du point de vue de l'utilisateur et de son interaction avec son environnement, la sécurité est un point fondamental, à la fois pour la confidentialité et l'intégrité de ses données ou pour la sureté de ses équipements. Une large adoption des systèmes pervasifs ne peut se faire sans une approche intégrée de la sécurité. Nous proposons l'architecture pervasive PerSE pour laquelle nous détaillons ici les différents mécanismes de sécurité mis en œuvre sous la forme de règles d'accès à l'information, aux services et aux équipements.

Mots clefs
Sécurité, Environnement Pervasif, Informatique Ubiquitaire

ABSTRACT
The opportunities given by the pervasive systems are manifold and very promising. From the user point of view and his interaction with his environment, the security is a key point, in terms of privacy and integrity or security of his equipments. A wide adoption of pervasive systems will not be possible without an integrated approach to security. We propose a pervasive architecture, namely PerSE, and we focus in this article on the different levels of rule based security mechanisms developed.

Categories and Subject Descriptors
C.2.0 [**Computer Systems Organization**]: Security and protection; C.2.4 [**Computer Systems Organization**]: Distributed systems; D.4.4 [**Operating Systems**]: Communications Management – Network communication;

General Terms
Design, Security

Keywords
Security, Pervasive Environment, Ubiquitous Computing

1. INTRODUCTION
La sécurité dans les environnements pervasifs est un facteur clé pour l'acceptation des technologies apparaissant dans ces environnements. L'omniprésence des dispositifs autour de l'utilisateur doit lui apporter des services utiles et pertinents en fonction de ses besoins, de manière réactive (après avoir exprimé une intention) ou de manière proactive (anticipation des besoins).

UbiMob'06, September 5-8, 2006, Paris, France.

Toutefois, chaque utilisateur veut pouvoir contrôler la manière avec laquelle il interagit avec son environnement, en particulier quels services ou données il est prêt à partager avec lui. En effet, il doit pouvoir décider différents niveaux d'autorisation d'accès à son environnement : par exemple, quelles données sont extrêmement confidentielles et ne devraient transiter que sur une liaison sécurisée, quelles données sont assez peu confidentielles pour pouvoir être lues par n'importe quel service ou personne alentour, quels sont les droits d'usage des services gérés sur un dispositif, … Toutes sortes d'autorisations personnalisées doivent pouvoir être mises en place dans une architecture pervasive, en fonction du contexte d'utilisation et de la confiance dans son environnement. La sécurité doit donc se trouver d'une part à l'interface entre les dispositifs pervasifs eux-mêmes, et également à l'intérieur des dispositifs pour contrôler l'accès aux données et services hébergés. De plus, l'ajout de sécurité dans le système pervasif ne peut se faire au détriment des performances, et doit rester la moins intrusive possible.

La plupart des approches actuelles d'environnement pervasifs n'abordent pas le problème de la sécurité au cœur de l'architecture. Cette dernière se trouve généralement à la périphérie de l'architecture globale, de manière auxiliaire, ou se limite souvent à une autorisation d'accès sommaire au dispositif pervasif. Le projet Aura [2] se concentre sur la modélisation des tâches des utilisateurs pour créer un système proactif. 3PC [3] permet aux applications pervasives de s'adapter à l'environnement, notamment grâce à un système de composants distribués spécifiés par contrat, de même que one.world [4], qui met en place un framework simplifiant la migration et la composition d'applications, ainsi que le partage de données.

Certaines approches proposent cependant des architectures pervasives sécurisées, mais deviennent très limitatives et ne permettent que peu de choses dans l'environnement pervasif. Les interactions se limitent alors à des échanges de données entre dispositifs, comme dans l'architecture pawS [7] ou Confab [8]. La solution Daidalos [9] fait, elle, appel à une tierce partie de confiance pour garantir la confiance de certaines entités, ce qui pose certains problèmes dans un environnement pervasif très dynamique où une tierce partie n'est pas forcément présente.

Par rapport à des modèles de contrôle d'accès discrétionnaires, notre travail permet de prendre fortement en compte le contexte de l'utilisateur. Les modèles RBAC ou ORBAC [10], basés sur les rôles et/ou les organisations, sont eux destinés à des structures prédéfinies, dans lesquelles la dynamicité n'est pas importante, et ne prennent que peu en compte le contexte de l'utilisateur dans son environnement.

Aussi, nous pensons qu'une approche intégrée prenant en compte de multiples aspects de sécurité dès la conception d'une architecture pervasive est une solution élégante et performante.

Le travail présenté dans cet article a été soutenu par le Conseil Scientifique de l'INSA de LYON.

Nous illustrons dans cet article l'intégration des problématiques de sécurité dans la plateforme PerSE (Pervasive Service Environment) [1].

Le reste de cet article est organisé comme suit : la partie 2 présente brièvement l'architecture d'environnement pervasif PerSE. La partie 3 s'intéresse plus particulièrement aux modules liés à la sécurité dans l'architecture. La partie 4 illustre l'utilisation de la plateforme et des modules de sécurité sur un exemple. La partie 5 conclue l'article et ouvre les perspectives.

2. ENVIRONNEMENT PERVASIF A BASE DE SERVICES

2.1 Environnement PerSE

Nous avons modélisé et développé notre vision d'environnement pervasif, se comportant d'une manière non-intrusive et permettant des actions réactives et proactives. Le résultat se concrétise par une plateforme nommée PerSE.

Pour intégrer un environnement PerSE, chaque équipement exécute un méta-service, appelé Base, lui permettant de partager ses services locaux et son contexte local. La Base PerSE se charge des communications avec les autres Bases afin d'exécuter intelligemment des services répartis, de manière transparente et adaptée, dans le but de répondre aux besoins des utilisateurs.

Un environnement PerSE comprend ainsi plusieurs Bases autonomes capables de se découvrir et d'échanger des messages à travers différents canaux de communication (LAN, WiFi, Bluetooth, …) disponibles sur les équipements.

Pour répondre aux besoins des utilisateurs, une modélisation de leurs intentions est nécessaire. Le langage PsaQL [1] permet à l'utilisateur (ou à une application intermédiaire) d'exprimer son intention sous forme d'une *action partielle*, définissant les services que l'utilisateur souhaite utiliser et potentiellement leur localisation. La Base PerSE doit alors interpréter cette action pour construire, à partir de sa connaissance de l'environnement, une *action complète*, c'est-à-dire un graphe connecté de services interopérables pouvant être exécuté.

2.2 Base PerSE

Afin de répondre aux besoins de l'environnement PerSE, une Base PerSE remplit plusieurs fonctionnalités réparties en trois niveaux : *Communication*, *Environment*, *Action* (voir Figure 1). Afin de prendre en compte les besoins de sécurité, des modules spécifiques ont également été intégrés (en gris sur la figure). La communication entre modules se fait soit de manière directe (appel de méthodes), soit par échange de messages PerSE asynchrones (voir la section 3.1 pour plus de détails).

2.2.1 Niveau « Communication »

Le niveau *Communication* comprend les interfaces entre la Base PerSE et l'environnement informatique (réseaux, système de fichiers, applications, …). Les modules *Local Data Interface* et *Local Service Interface* permettent d'abstraire les appels au système d'exploitation afin d'unifier l'accès aux données locales (capteurs, fichiers) et aux services locaux (services Web, programmes, ...)

Le module *Base Interface* est le point d'entrée local pour les utilisateurs (ou les applications utilisatrices) voulant utiliser les fonctionnalités de la Base, notamment pour démarrer des *actions partielles* en utilisant PsaQL. Le module *Messenger* est chargé des communications entre les Bases de l'environnement PerSE (découverte et échange de messages). Ces deux modules, connectés avec le monde extérieur, envoient et reçoivent (à travers le *Message Filter*, pour des raisons de sécurité, voir partie 3.1), des messages PerSE vers/depuis les autres modules.

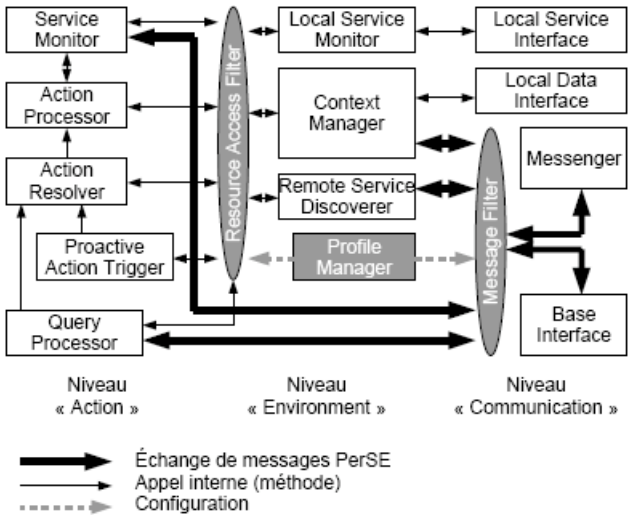

Figure 1 : Architecture globale d'une Base PerSE

2.2.2 Niveau « Environment »

Le niveau *Environment* gère les ressources locales de la Base, ainsi que ses connaissances sur l'environnement PerSE. Le *Local Service Monitor* permet de gérer, de manière unifiée, l'ensemble des services locaux disponibles. Le *Context Manager* se charge à la fois de l'accès au contexte local et de l'accès distant (sous forme de requêtes) au contexte des autres Bases, afin d'offrir aux autres modules une vision unifiée et la plus complète possible du contexte de l'environnement. Le *Remote Service Discoverer* soumet des requêtes aux autres Bases pour maintenir localement un annuaire des services distants disponibles dans l'environnement.

Du point de vue sécurité, l'accès aux modules de ce niveau se fait par l'intermédiaire du module *Resource Access Filter,* commandé par le *Profile Manager*, afin de filtrer les informations disponibles au niveau *Action*.

2.2.3 Niveau « Action »

Le niveau *Action* se charge d'une part de recueillir les requêtes venant des autres Bases ou du module *Base Interface*, d'autre part d'exécuter des actions. Le *Query Processor* réceptionne des messages PerSE encapsulant des requêtes et y répond en recherchant les informations demandées dans le niveau *Environment* (liste de services, accès au contexte). Il reçoit aussi des *actions partielles*, déclenchant alors une nouvelle action. Le module *Proactive Action Trigger*, qui surveille le contexte et maintient un historique des actions réalisées, peut également générer des *actions partielles* de manière proactive.

Les *actions partielles* sont alors transmises à l'*Action Resolver* pour être transformées, en fonction du contexte et des services disponibles, en *actions complètes* pouvant être exécutées par l'*Action Processor*. Le *Service Monitor* est utilisé pour commander l'exécution de services, en local et à distance, et surveille leur statut.

Les modules de sécurité sont décrits dans la section suivante.

3. MODELE DE SECURITE DANS PERSE

La sécurité et la protection des ressources (nous appelons par ressource toute donnée ou service liée à une entité) est l'un des objectifs majeurs de l'architecture PerSE. Nous avons donc intégré dès la conception de l'architecture elle-même des modules dédiés à la sécurité. Ainsi 3 modules sont dédiés à la sécurité : le *Message Filter*, le *Resource Access Filter*, et le *Profile Manager*.

3.1 Le *Message Filter*

Le *Message Filter* agit comme un filtre sur les communications entrantes et sortantes, communications basées sur des messages asynchrones. Un message PerSE possède la structure suivante :

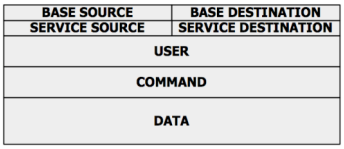

Figure 2 Structure d'un message PerSE

Il est composé de deux parties principales : l'entête et les données. L'entête est composée elle-même de 4 couches : les bases source et destinataire du message, les services sources et destinataire, l'utilisateur à l'initiative du message (s'il existe), et un champ commande, qui définit le type du message : requête, réponse à une requête, découverte, etc.

L'entête d'un message contient donc les informations nécessaires à son identification et à son filtrage. Le *Message Filter,* par l'application de **règles de communication**, autorise ou non le passage de ces messages. C'est donc un premier Policy Decision Point (PDP) pour l'architecture, mais aussi un Policy Enforcement Point (PEP) au niveau des communications. Une règle de communication permet d'effectuer 3 actions sur un message provenant d'une entité : accepter (allow), refuser et notifier du refus (deny), ou refuser et ne pas notifier (drop). Une règle de communication est de la forme :

```
DO action ON MESSAGES FROM entity
action = {allow, deny, drop}
entity = {user X, base X, service X}
```

Ces règles représentent le premier niveau de sécurité mais aussi le plus simple.

3.2 Le *Resource Access Filter*

Le second niveau de sécurité se situe au niveau de l'accès aux ressources. Le *Resource Access Filter* permet de contrôler les interactions entre les différentes entités (nous appelons par entité une base PerSE, un service, ou un utilisateur) et les ressources (données ou services). Ici encore, ce contrôle est réalisé à base de **règles**, regroupées sous forme de **profils**. À un moment donné, un profil est appliqué sur la base, et réglemente l'accès aux ressources. Le *Resource Access Filter* est donc un second PDP, mais aussi un PEP car il accède et fournit les données si la requête d'accès est autorisée.

Une **règle d'accès aux ressources** définit, pour un groupe d'entités, le droit d'effectuer une action sur des groupes de données (modification, suppression, etc.) ou de services (exécution, découverte, monitoring, communication entre services, etc). Un **groupe d'entités** est défini par l'utilisateur ou l'administrateur de la base, et regroupe des entités proches du point de vue de la confiance et de la sécurité (par exemple, un groupe « anonyme » regroupera les entités que l'utilisateur ne connaît pas et donc en qui il ne peut avoir confiance). Nous verrons plus loin la forme exacte d'une règle.

3.3 Le *Profile Manager*

Le profil à utiliser est défini par le *Profile Manager*, qui représente véritablement le coeur de la sécurité de l'architecture PerSE. Ce module permet de déterminer quelles règles s'appliquent lors de la réception d'une requête, et ce en fonction du contexte qui entoure l'utilisateur. C'est le Policy Administration Point (PAP) de l'architecture, car il décide de la politique de sécurité à appliquer.

Nous introduisons la notion d'**espaces contextuels**, sous-ensembles de l'espace environnant, et définis par des informations sur le contexte de cet espace. Ces espaces contextuels sont référencés et stockés dans le *Contextual Space Manager* (voir Figure 3 : Structure du Profile Manager). Dans chaque espace contextuel s'applique un **algorithme de détermination des règles**, transmis par *l'Algorithm Manager*, qui gère les différents algorithmes existants, puis exécuté par le *Rule Decider*. Chaque algorithme est basé sur des paramètres contextuels différents (la confiance que l'on a dans les entités présentes alentour, la localisation de l'utilisateur ou encore la température actuelle, etc.). Ce contexte constitue **le support des règles,** c'est à dire l'ensemble des paramètres contextuels et leur valeur, pour lesquels la règle est valide. Nous pouvons maintenant définir une règle d'accès aux ressources. Une règle est de la forme :

```
group_x CAN DO action ON resource IN ctxt_a
```

Les règles font intervenir 4 paramètres : le groupe de l'entité (*group_x*), l'action, la ressource sur laquelle s'effectue l'action, et le support des règles, c'est à dire l'espace contextuel dans lequel la règle est valide (*ctxt_a*). Nous verrons lors de l'étude de cas comment modéliser et traduire ce contexte.

De nombreuses études ([5], [6]) sur la perception de la sécurité par les utilisateurs dans un environnement pervasif ont montré que celle-ci varie énormément selon l'utilisateur, et donc chaque personne possède ses propres critères d'évaluation de la confiance d'un lieu. Nous offrons donc à l'utilisateur la possibilité de définir lui-même de nouveaux espaces contextuels et de nouveaux algorithmes selon les paramètres qu'il souhaite utiliser dans un espace précis pour déterminer les règles et donc la politique de sécurité associée. Cet ajout est réalisé par *l'Algorithm Manager*, qui mettra à disposition le nouvel algorithme en fonction du besoin. Pour cela, l'utilisateur dispose d'une liste des données contextuelles à sa disposition et sur lesquelles il peut se baser pour définir ses algorithmes.

En l'absence d'algorithmes ajoutés par l'utilisateur, c'est un algorithme par défaut qui est utilisé, basé sur la localisation de l'utilisateur et son voisinage proche, lointain, et inconnu. Il est même possible de composer les algorithmes pour répondre à des besoins plus précis, par exemple à la fois sur la localisation et la température.

Les règles à appliquer, groupées sous forme de profils, sont transmises aux *Message Filter* et *Resource Access Filter* qui les utiliseront.

Enfin, un historique permet de stocker les différentes situations rencontrées, ainsi que l'algorithme utilisé et les profils de règles déduites de l'algorithme, ceci afin d'éviter, pour une situation déjà

rencontrée plusieurs fois, de re-exécuter les algorithmes et demander des données, ce qui peut parfois se révéler coûteux.

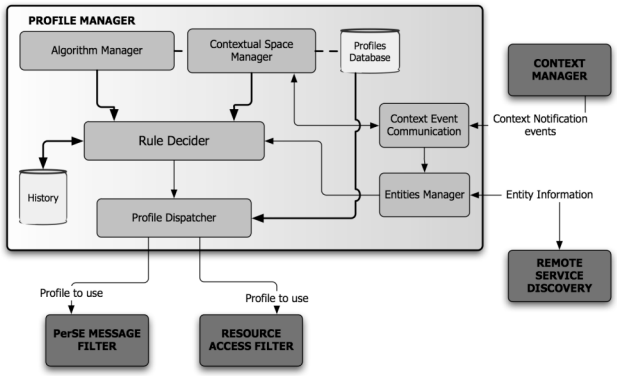

Figure 3 : Structure du Profile Manager

Concernant les règles utilisées dans notre modèle, l'objectif à moyen terme est de fournir pour l'utilisateur un langage de description simple, basé sur une grammaire décrite en BNF. Ce langage permettra ensuite d'interpréter et d'implémenter les règles dans un standard reconnu comme XACML, dont les règles basées sur le triplet "Sujet-Ressource-Action" correspondent à notre sémantique.

4. ETUDE DE CAS

Dans notre scénario, un ordinateur E_0 contenant une base PerSE partage une séquence vidéo en utilisant le service *ShareVideo*. La base E_0 se situe dans une pièce d'un bâtiment. D'autres entités (E_1 jusqu'à E_5) souhaitent accéder à la base E_0 pour utiliser le service *ShareVideo*. Mais l'administrateur de la base E_0 à mis une restriction sur l'utilisation de ce service : seules les bases situées dans la même pièce ont le droit. Cette restriction est exprimée de la façon suivante :

```
GROUP anonymous CAN execute ON ShareVideo IN
ctxt_visible = "neighbourhood"
```

Cette règle se traduit littéralement par : « le groupe anonyme, donc tout le monde, peut exécuter le service *ShareVideo* si et seulement si le support de règle *ctxt_visible* est *neighbourhood*, c'est à dire si l'entité est dans le voisinage proche ». Le support de règle *ctxt_visible* est défini par l'algorithme suivant :

```
ctxt_visible(LocalBase, CallerBase):
IF LocalBase.Context.Localization.Name =
CallerBase.Context.Localization.Name
THEN ctxt_visible = "neighbourhood"
ELSE ctxt_visible = "unknown"
```

Un support de règle a deux paramètres : la base recevant une requête (*LocalBase*) et la base faisant appel à une fonctionnalité (CallerBase). Chaque objet base offre un contexte, organisé d'une manière spécifique en PerSE.

Dans notre scénario, une fois que la requête est arrivée, la règle s'applique et l'algorithme détermine la valeur de *ctxt_visible*. Les entités situées dans la même salle que la base E_0 auront donc le droit d'exécuter le service *ShareVideo*, les autres non (Figure 4). Ceci n'est bien sûr qu'un exemple trivial des nombreuses possibilités de cette modélisation qui permet en réalité de prendre en compte bon nombre de paramètres et de conditions pour la sécurité des données.

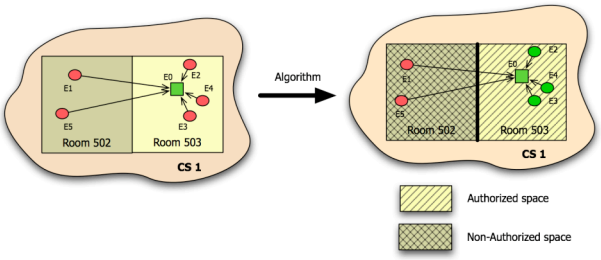

Figure 4 : Détermination des règles d'accès aux ressources

5. CONCLUSION

Nous avons présenté dans cet article l'architecture d'un environnement pervasif où la sécurité joue un rôle de premier ordre. Les différents filtres utilisent le contexte pour la définition des règles de communication et d'accès aux ressources. Nous pensons ainsi minimiser l'interaction entre un utilisateur et son environnement pour des actions concernant la sécurité des données ou les droits d'utilisation des différents services.

L'utilisation de cette architecture dans un certain nombre d'exemples nous a permis d'accroître la satisfaction d'un utilisateur par rapport à une architecture sans mécanisme de sécurité contextuel. Nous envisageons d'approfondir encore plus la relation entre le contexte et les règles de filtrage (pour les communications et l'accès aux ressources), et mettons l'accent sur l'amélioration de la performance d'un tel système.

6. BIBLIOGRAPHIE

[1] 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

[2] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P., *Project Aura: Toward Distraction-Free Pervasive Computing,* IEEE Pervasive Computing, April-June 2002

[3] Becker C. *et al., PCOM - A Component System for Pervasive Computing*, IEEE PerCom'04, p. 67, 2004

[4] Grimm R. *et al*., *System Support for Pervasive Applications*, ACM Transactions on Computer Systems, Vol. 22, No. 4, November 2004, Pages 421–486

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

[6] Dey A.K., *Understanding and using context, Personal and Ubiquitous Computing*, 5(1):4—7, February 2001, Springer.

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

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

[9] 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

[10] Abou El Kalam A. *et al*., *Organization Based Access Control*. In *Policy'2003*, Como, Italie, June 2003.