

Services spontanés sécurisés pour l'informatique diffuse

Slim Trabelsi

▶ **To cite this version:**

Slim Trabelsi. Services spontanés sécurisés pour l'informatique diffuse. domain_other. Télécom ParisTech, 2008. English. NNT : . pastel-00004140

HAL Id: pastel-00004140
https://pastel.hal.science/pastel-00004140

Submitted on 10 Apr 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.





Services Spontanés Sécurisés Pour l'Informatique Diffuse

Thèse

Présentée pour obtenir le grade de docteur
de l'Ecole d'Ingénieur TELECOM
ParisTech
Spécialité : Informatique et réseaux

Par

Slim Trabelsi

Soutenue le 07 Juillet 2008 devant le jury composé de

Rapproteurs :	**Valérie Issarny (INRIA, LE Chesnay – France)** **Hossam Afifi (TELECOM SudParis – France)**
Examinateurs:	**Pascal Urien (TELECOM ParisTech – France)** **Roberto Di Pietro (Università di Roma – Italie)** **Jean-Christophe Pazzaglia (SAP Labs – France)**
Directeur de Thèse:	**Yves Roudier (Eurecom – France)**

Remerciements

Je dédie cette thèse à mes parents, à Olfa, et ainsi qu'à mon frère qui m'ont tout donné et qui m'ont apporté un soutien sans faille.

Je tiens à remercier tout d'abord Yves Roudier pour la confiance qu'il m'a accordée, ainsi que pour tous les conseils et toute l'aide apportée lors des mes années de thèse. Je remercie aussi Jean-Christophe Pazzaglia et Guillaume Urvoy-Keller pour leur aide si précieuse et leur bonne humeur permanente.

Une pensée particulière à tout le staff d'Eurecom, notamment aux doctorants : Marco, Alessandro, Emilie, Moritz, Federico, Remi, Selim, René et tant d'autres, avec qui j'ai passé des moments inoubliables, ainsi qu'au personnel administratif et tout particulièrement Gwenaëlle, Emilie, Carine, Patrice , Nadine, Christine…

Merci à tous.

Slim Trabelsi, Septembre 2008

Abstract

The pervasive computing paradigm assumes an essentially dynamic model of interaction between devices. That also motivates the need for mechanisms to discover previously unknown service at an early phase of such interactions. Whereas this assumption is at the heart of many pervasive computing protocols and systems, the necessity of securing service discovery, in particular privacy-wise, and the complexity of this task have been largely underestimated, if considered at all.
The goal of this thesis is to study the security requirements of service discovery protocols and to provide secure and reliable solutions to overcome threats and security lacks associated to service discovery standards. Three security solutions are proposed that address different deployments of service discovery. Architectures suitable for secure discovery and the protocols they require are introduced in the first part of this thesis. Decentralized architectures are addressed using an attribute based encryption scheme to restrict the access to discovery messages. An alternative proposal is presented for centralized architectures that makes use of a trusted registry in charge of enforcing discovery policies. Finally, a hybrid model is presented for large-scale deployments that relies on a peer to peer indexing system accessible through an anonymizing layer. The second part of this thesis details the effect of secure service discovery through a performance study approach. A mathematical performance model is designed in order to evaluate the robustness, availability, efficiency, and resource consumption of a service discovery system during its normal execution and under a denial of service attack. Finally, this thesis studies the security and trust issues related to the introduction of context awareness into pervasive computing systems and in particular, its impact on the service discovery mechanisms in terms of matching accuracy and flexibility with respect to the environment.

Résumé

Chapter I. Introduction

La notion d'informatique diffuse fut définie par Weiser en 1991 [WEI91] comme étant l'avenir de l'informatique dans lequel l'utilisateur pourra interagir numériquement avec son environnement (intelligent) de façon complètement transparente. L'environnement est assimilé à un ordinateur composé de plusieurs petits composants qui interagissent en permanence et offrant à l'utilisateur la possibilité d'accéder à des informations et des données. L'informatique diffuse devient donc accessible partout et à tout moment faisant ainsi partie de la vie quotidienne de l'être humain. Aujourd'hui l'informatique diffuse est devenue d'actualité et est réellement déployée dans certains réseaux (entreprises, aéroports, salles de conférences, maisons intelligentes …), et s'appuie généralement sur le concept du tout service, où les équipements intelligents sont accessible via une interface client service. Ce type de déploiement de systèmes est plus communément appelé Architecture Orientée Service (AOS), qui est l'une des techniques de déploiement de systèmes les plus répandues et les plus efficaces pour les systèmes distribués. Ce type d'architecture a été initialement développé pour faciliter l'accès dynamique aux applications déployées sur des équipements environnants. Combinée à la programmation orientée objet, l'AOS permet le déploiement d'applications distribuées pouvant communiquer de façon efficace et flexible via un réseau de données ou mobile, repoussant ainsi les limites des solutions centralisées. Le service est l'élément principal d'une AOS, il englobe un ensemble de fonctionnalités et de procédures accessibles via une interface standardisée. Cette interface permet aux utilisateurs ainsi qu'aux logiciels de s'interconnecter et de communiquer de façon optimale et flexible (adaptative). Ces services peuvent ainsi représenter tous les aspects applicatifs connus qui proposent des fonctionnalités tels que l'accès aux bases de données, gestion et traitement de données ou encore les transactions commerciales. Le consommateur de ces services est plus communément appelé client, et est représenté par un utilisateur humain ou logiciel. Ces deux acteurs peuvent communiquer via une interface standardisée sans pour autant partager la même plateforme d'implémentation ou de déploiement. IBM définit par exemple la notion de service dans une AOS comme étant une fonction applicative structurée et par conséquent comme un composant réutilisable pour n'importe quelle transaction commerciale.
Actuellement le paradigme d'AOS est accru par l'émergence de la technologie des Web Services. Cette technologie est totalement indépendante de l'environnement de programmation, ainsi les applications peuvent être développées en utilisant n'importe quel langage de programmation sans pour autant présenter un problème de communication. Pour faciliter la communication et l'interconnexion de ces différentes applications, des interfaces flexibles décrites à l'aide du langage balisé XML sont utilisées comme étant un pont de jonction des différentes applications. Ainsi chaque application peut interpréter cette interface XML et générer dynamiquement le squelette de la méthode utilisée pour interagir avec le service distant. Le standard utilisé généralement pour la description de ces interfaces d'accès aux services est appelé Web Service Description Language (WSDL). Le format des protocoles de communication est aussi standardisé via une enveloppe de messages XML appelée SOAP.

A. Découverte de services

Afin de gérer efficacement le déploiement et le fonctionnement des AOS, des techniques d'orchestration régissant toutes les étapes de mise en œuvre de ce genre d'architectures ont été développées. Ces techniques ont pour rôle d'identifier et localiser les différents composants du système et permettre de les manipuler de façon intelligente, comme par exemple la possibilité de composer et combiner plusieurs services en vue d'obtenir un service évolué. Parmi ces techniques d'orchestration, la découverte de services prend une place importante au sein de la pile AOS, surtout quand l'environnement de déploiement devient plus dynamique et plus ouvert. Ainsi, si on se place dans le cadre d'un intranet local mettant en jeu très peu d'acteurs, de simples services de nommage (à la CORBA) peuvent faire l'affaire. En revanche, dans le cadre d'un grand réseau distribué, il est évident que les services de découvertes doivent prendre en compte plusieurs paramètres, à savoir : l'existence de milliers voire de millions de services hétérogènes répartis à travers le globe, les limites de bandes passantes, la capacité du passage à l'échelle, et surtout la sécurité. Les protocoles de découverte de services sont différents des systèmes de nommage/adressage de type DNS car ils concernent plus la couche applicative et les aspects d'indexation logique complètement indépendants du système d'adressage et de routage niveau réseau. Les systèmes de découverte mettent en jeu deux ou trois éléments primordiaux, à savoir les clients ainsi que les serveurs, et dans certaines configurations la possibilité de rajouter des répertoires jouant le rôle d'annuaires de services. Les répertoires sont généralement publics et permettent aux serveurs d'enregistrer leurs services ainsi qu'aux clients de demander des services. Dans le cas où les seuls éléments du système sont les clients et les serveurs, la découverte de service est dite décentralisée. Dans le cas où le répertoire est utilisé lors de la découverte de services, le système est dit centralisé.

B. Contribution

Dans cette thèse je focalise mon étude sur la sécurisation et l'étude de performance des systèmes de découvertes de services appliqués aux services ubiquitaires dans les AOS. Cette étude commence par l'analyse des failles de sécurité, ainsi que des attaques possibles auxquelles sont exposés les systèmes de découvertes existants. Cette analyse a abouti à un modèle de sécurité détaillant les différents paramètres de sécurité qui doivent être pris en compte par les développeurs et les programmeurs afin de déployer un système de découverte sûr et robuste.

Partant de ce modèle de sécurité, j'ai proposé trois solutions sécurisées pour la découverte de services correspondant à des degrés divers d'organisation et d'échelle de l'infrastructure sur laquelle le mécanisme est déployé. Nous montrons tout d'abord comment le chiffrement est suffisant pour protéger des architectures décentralisées de type LAN ou WLAN en restreignant l'accès aux messages de découverte diffusés d'après une politique à base d'attributs. Nous proposons ensuite l'utilisation de politiques de découvertes comme concept essentiel à la sécurisation de la découverte de services dans les architectures centralisées qui s'appuient sur un registre comme tiers de confiance. Nous introduisons enfin une architecture pour le déploiement d'un mécanisme de découverte de services sécurisé à plus grande échelle s'appuyant sur un système d'indexation pair à pair accessible via une couche de routage anonyme. Dans une deuxième partie de la thèse, nous analysons l'efficacité des mécanismes de découverte de service sécurisée proposés par une étude de performance. Un modèle Markovien est construit afin de calculer différents paramètres de performances liées à la robustesse, la disponibilité, l'efficacité ou le coût en termes de ressources consommées lors de l'exécution d'un processus de découverte en charge normale aussi bien que soumis à une attaque de déni de service. Nous discutons finalement dans une dernière partie des problèmes

de sécurité et de confiance liés à l'introduction de sensibilité contextuelle dans les mécanismes de découverte de service.

Chapter II. Modèle d'attaque et de sécurité pour la découverte de services

Sécuriser les services ubiquitaires revient à protéger les informations de découvertes publiées par les utilisateurs et les services lors d'une exécution d'un processus de découverte dans un milieu hostile (non sûr). Afin de cibler ces informations sensibles et détecter les failles de sécurité possibles lors de l'exécution du protocole de découverte, nous avons décidé de mener une analyse non formelle de chaque étape d'exécution de l'algorithme et de souligner les vulnérabilités, d'imaginer les attaques possibles pour les exploiter et proposer des contre-mesures. Non avons donc commencé par énumérer une liste non exhaustive de ces failles et attaques.

A. Attaques et vulnérabilités

Vulnérabilités au niveau des messages

- Possibilité d'attaques de dénis de service contre les répertoires : l'attaquant peut générer de faux messages de publications de services afin de surcharger les répertoires jusqu'à la panne de ce service.
- Accès aux requêtes des clients : en accédant aux messages de requête clients, l'attaquant peut deviner les intentions de l'utilisateur ainsi que ses activités, son identité et ses ressources.
- Interception des requêtes des clients : afin de récupérer les informations personnelles décrites dans le point précédent, les attaquants peuvent se faire passer pour de faux répertoires et ainsi récupérer toutes les requêtes des clients.
- Modification ou suppression des requêtes des clients : si un attaquant arrive à compromettre un routeur, il a la possibilité d'intercepter les requêtes de clients afin de les modifier et les réutiliser à des fins illégales, ou bien tout simplement les supprimer afin d'isoler l'utilisateur.
- Rejouer des requêtes clients : il s'agit d'écouter le média de transmission des messages (dans un réseau sans fil non protégé), mémoriser les requêtes des clients, afin de les rejouer par la suite et ainsi se faire passer pour un client honnête.
- Rejouer des messages de publication de services : afin de donner de fausses informations à propos des services publiés, un attaquant a la possibilité de générer des messages de publication de faux services ou des services modifiés à partir de véritables messages récupérés du réseau. Ces fausses informations injectées dans le répertoire ont pour but de tromper voire pirater les clients à la recherche de véritables services.

Publication des services

- La publication d'un service sur un faux répertoire : un attaquant peut mettre en ligne de faux répertoires dans le but des récupérer et tromper tous les utilisateurs du service de découverte. Il peut ainsi créer un nœud absorbant.
- Effacement illégal de services déjà publiés : un attaquant peut se faire passer pour un service afin d'effacer les données publiées sur un répertoire.
- Publier de faux services : un attaquant peut publier de faux services qui ne correspondent pas à leurs descriptions.

B. Modèle de sécurité

A partir du modèle d'attaque décrit précédemment nous pouvons déduire un modèle de sécurité permettant de protéger les systèmes de découvertes contre ses éventuelles attaques.

- Authentification: l'objectif de base de la découverte est la communication entre différentes entités inconnues entre elles. Afin de pouvoir vérifier l'existence et l'honnêteté de ces éléments il est impératif que chaque entité puisse authentifier son vis-à-vis et ainsi vérifier l'origine des messages.
- Autorisation : on peut supposer que certains services puissent être restreints à un certain type de clients, ainsi que certaines requêtes puissent être destinées à certains répertoires ou services. Pour ce faire, des systèmes d'autorisation doivent être mis en œuvre pour octroyer certains privilèges (certificats, tickets, droits d'accès …) aux entités concernées.
- Protection des données privées : chaque message échangé lors de la découverte de service peut contenir des informations privées liées aux clients ainsi qu'aux services. Ces informations doivent absolument être protégées et restreintes d'accès aux éléments ayant les autorisations nécessaires pour y accéder.
- Confidentialité : afin de protéger l'accès aux messages de découverte ainsi qu'aux données privées, il existe des techniques qui assurent la confidentialité en cachant les données sensibles et en les divulguant uniquement aux entités ayant les autorisations nécessaires pour y accéder. Généralement, des techniques de chiffrement sont utilisées pour assurer la confidentialité des données.
- Contrôle d'accès : l'authentification des clients et des serveurs est assez problématique lors de la phase initiale de la découverte. Idéalement, un serveur ne devrait publier ses services restreints qu'aux clients ayant un profil ou des autorisations spécifiques, mais dans la pratique il est extrêmement difficile de mettre en œuvre cette restriction. Pour cette raison, il est impératif pour des serveurs de déployer des systèmes de contrôle d'accès aux profils des services proposés, afin de pouvoir filtrer efficacement les clients ayant le droit de les découvrir.

Chapter III. Sécuriser la découverte de services décentralisée

Lors d'une découverte de service décentralisée, les clients et les serveurs communiquent directement entre eux, sans aucun intermédiaire. Les serveurs contactent tous les clients du réseau en leur envoyant les profils des serveurs à publier, et les clients à l'écoute du réseau récupèrent les profils publiés et les mémorisent afin d'accéder aux services en cas de besoin. Si un nouveau client rejoint le système, il a la possibilité de contacter tous les serveurs afin de demander quelques services. La particularité de ce genre de configuration décentralisée est qu'il n'y a aucun intermédiaire de confiance qui pourra être en charge de la sécurisation de la découverte. Pour cette raison, chaque élément doit assurer lui-même la protection de sa découverte de service.

Afin de satisfaire les bases décrites dans le modèle de sécurité, comme la confidentialité, le contrôle d'accès ou la protection des données privées, il y a la possibilité, adoptée par certaines solutions, d'utiliser un système de chiffrement et d'authentification basé sur

l'infrastructure des clefs publiques (PKI). Ce type de système basé sur une distribution préliminaire de clefs publiques et privées ainsi qu'un serveur d'authentification déployé par l'autorité de certification permettant à chaque utilisateur de vérifier l'authenticité des clefs. Ce type d'infrastructure suppose la présence permanente d'une infrastructure de sécurité, ce qui entre en conflit avec le principe d'applications ubiquitaires décentralisées et mobiles qui ne peuvent pas forcément dépendre d'une infrastructure fixe. Ainsi, afin de se démarquer de cette contrainte tout en gardant les mêmes fonctionnalités associées au PKI, nous avons décidé d'utiliser une infrastructure de chiffrement appelée chiffrement basé sur les attributs, qui est une extension du chiffrement basé sur l'identité. L'avantage de cette nouvelle alternative est de permettre d'être indépendant d'une infrastructure d'authentification pour vérifier la validité des clefs de chiffrement/déchiffrement.

A. Chiffrement basé sur les attributs

Le concept du système de chiffrement basé sur les attributs est similaire au système de clefs asymétriques du PKI, à la seule différence que les clefs publiques ont un sens sémantique décrivant un ensemble d'attributs représentant une identité.

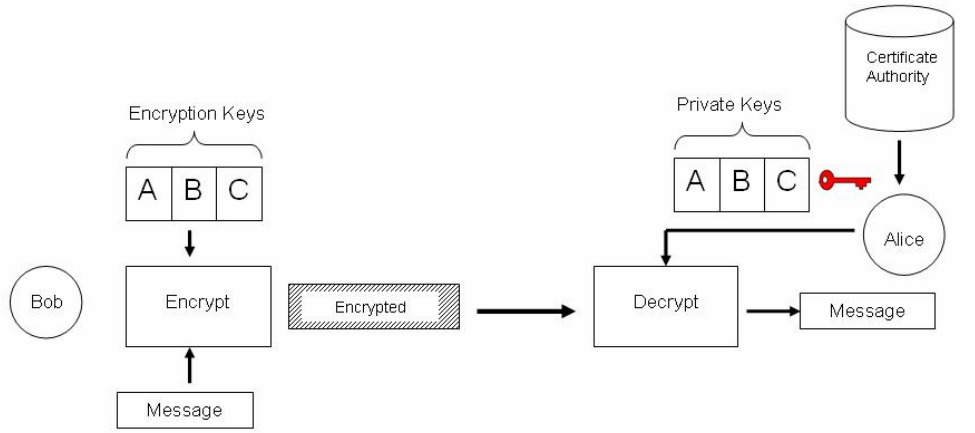


Figure 1 : Chiffrement basé sur les attributs

Comme la Figure 1 le montre l'utilisateur Bob qui veut envoyer un message chiffré à Alice, dont le profil est décrit par les attributs publics {A,B,C}, peut utiliser la combinaison de ces attributs afin de constituer la clef publique (unique) de chiffrement. Alice qui est propriétaire de ces attributs, et qui a un certificat démontrant cette propriété peut a tout moment générer les clefs privées liées à ses attributs afin de pouvoir les utiliser pour déchiffrer le message brouillé de Bob. Si elle y parvient, Bob a implicitement authentifié Alice, en restreignant l'accès à son message aux détenteurs des attributs {A,B,C}, protégeant ainsi les données privées qui peuvent être contenues dans son message.

B. Application du chiffrement basé sur les attributs aux protocoles de découverte de services

Nous avons choisi d'appliquer le chiffrement basé sur les attributs à tous les messages de découvertes qui doivent être sécurisés ; les requêtes des clients sont chiffrées en fonction des attributs correspondant aux services recherchés. Ainsi seuls les services authentifiés possédant les clefs privées valides correspondant à ces attributs sont capables de déchiffrer et répondre à la requête. Dans ce cas, le client est capable d'authentifier implicitement les serveurs

recherchés, vérifier la validité de leurs attributs, interdire l'accès de la requête aux entités non authentifiées et protéger les données privées contenues dans la requête.

```
<s:Body>
      <d:Probe>
        <d:Types>
          (Encrypt[Printer]{Printer|Eurecom})
        </d:Types>
      </d:Probe>
  </s:Body>
```

Figure 2: Requête client chiffrée

La Figure 2 représente une requête pour un service d'imprimante exprimée suivant le standard du protocole de découverte de Web Services WS-Discovery. Le corps de cette requête est chiffré en utilisant les attributs {printer, imprimante} comme clef publique. Ainsi uniquement les imprimantes certifiées du domaine Eurecom seront capables de déchiffrer la requête du client.

De la même façon, les serveurs proposant des services privés peuvent restreindre la découverte de ces services à certains clients ayant un profil bien déterminé. La même technique de chiffrement est utilisée pour brouiller le message d'annonce des services ainsi que les messages de réponse aux clients. Les attributs utilisés pour chiffrer le message correspondent aux attributs décrivant les profils des utilisateurs autorisés à découvrir le service.

```
<d:Hello ... >
<a:EndpointReference> Encrypt[EPR]{Professor,Eurecom} </a:EndpointReference>
      <d:Types> Encrypt[Color_Printer]{Professor,Eurecom} </d:Types>
      <d:Scopes> Encrypt[Eurecom_Printer]{Professor,Eurecom} </d:Scopes>
      <d:XAddrs> Encrypt[Colorprinter.eurecom.fr]{Professor,Eurecom} </d:XAddrs>
      <d:MetadataVersion>xs:unsignedInt</d:MetadataVersion>
...
</d:Hello>
```

Figure 3: Message de publication de service chiffré

La Figure 3 représente un message de publication de service d'imprimante couleur exprimée suivant le standard du protocole de découverte de Web Services WS-Discovery. Ce message est restreint aux professeurs certifiés par le domaine Eurecom.

C. Discussion

A ce niveau de la thèse, j'ai proposé une solution permettant de sécuriser la découverte de services dans une architecture décentralisée, où les éléments peuvent communiquer en toute sécurité sans aucun intermédiaire. En appliquant le chiffrement basé sur les attributs aux protocoles de découvertes de service, les contraintes de sécurité décrites au début de ce document sont satisfaites sans faire intervenir un tiers élément en charge de la sécurisation du processus, ce qui a pour avantage de permettre le déploiement de la découverte sécurisée dans un environnement ubiquitaire mobile.

Chapter IV. Sécuriser la découverte de service basée sur les répertoires

La solution décentralisée ne prend pas en compte l'utilisation de répertoires de découverte qui jouent le rôle d'intermédiaire entre les clients et les services. De plus, le fait de chiffrer les messages de découverte selon le procédé basé sur les attributs ne permet pas au répertoire de jouer son rôle d'annuaire, vu qu'il sera incapable de déchiffrer les messages brouillés. Par contre le fait d'utiliser un répertoire de confiance peut présenter un avantage pour sécuriser la découverte de services. Si les clients, ainsi que les services, arrivent à authentifier le répertoire, ce dernier pourra jouer le rôle d'intermédiaire de confiance sur lequel pourront se reposer ces éléments.

A. Politique de découverte de services

Afin de répondre aux contraintes énoncées par le modèle de sécurité, chaque utilisateur doit avoir la possibilité de décrire ses préférences, ainsi que les paramètres liés à la protection des messages de découverte. Côté client, l'utilisateur doit être en mesure de vérifier que les services qui correspondent à sa requête sont authentifiés et certifiés comme étant sûrs et honnêtes. Côte serveur, l'administrateur doit être capable de spécifier les restrictions et le contrôle d'accès liés à ses services. D'autres paramètres de sécurités tels que le chiffrement ou la protection des données privées doivent être définis. Pour se faire, nous avons décidé d'utiliser une politique de sécurité qui serait dédiée à la découverte de services (exemple voir figure 4).

```
<Policy PolicyId="Policy" RuleCombiningAlgId="permit-overrides">
   <Target>…</Target>
   <!--  Rule of Color Printer Discovery Action   -->
   <Rule RuleId="rgetPatient" Effect="Permit">
   <Target>
      <Subjects>…</Subjects>
      <Resources>Color Printer</Resources>
      <Actions>
         <Action>
         <ActionMatch MatchId="function:anyURI-equal">
         <AttributeValue   DataType="anyURI">Discover</AttributeValue>
         <ActionAttributeDesignator DataType=anyURI  AttributeId="action-id" />
         </ActionMatch>
         </Action>
      </Actions>
</Target>
   <!--  Check if the subject is a Professor from Eurecom   -->
   <Condition FunctionId="function:string-equal">
      <Apply FunctionId="function:string-one-and-only">
         <SubjectAttributeDesignator DataType="string"   AttributeId="SubjectRole"
/>
      </Apply>
      <AttributeValue DataType="string">Professor</AttributeValue>
       <AttributeValue DataType="string">Eurecom</AttributeValue>
      </Condition>
   </Rule>
</Policy>
```

Figure 4: Exemple de politique de découverte de services

Cette politique de sécurité est fournie par les clients et les serveurs puis est transmise au répertoire qui sera en charge de l'appliquer. Ainsi la fonction du répertoire n'est plus limitée au rôle de simple annuaire, mais de garant de la sécurité sur processus de découverte. Deux

nouvelles tâches lui sont assignées : l'interprétation et l'application des politiques de découverte, ainsi que l'authentification des éléments du système.

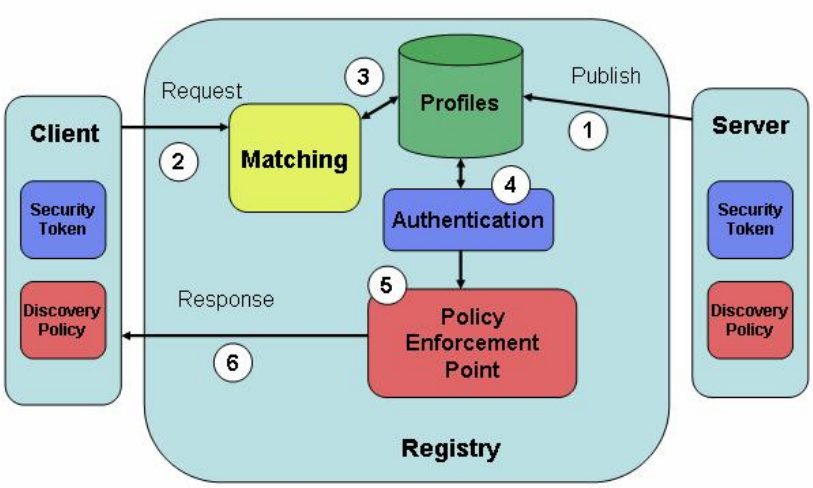


Figure 5: Architecture pour la découverte de services sécurisée basée sur les répertoires centralisés

La Figure 5 décrit l'ordre d'exécution du système de découverte de services basé sur les répertoires. Avant de commencer à échanger les messages de découvertes, les clients et les serveurs doivent d'abord joindre un répertoire, l'authentifier et établir une connexion sécurisée en utilisant des protocoles de chiffrements simples tels que SSL ou TLS. Ainsi tous les messages échangés par la suite seront chiffrés.

- Etape1 : le serveur publie ses services en envoyant au répertoire les profils des services à publier, les politiques de sécurité qui leur correspondent, et un certificat ou un ticket qui permettra au répertoire d'authentifier les services.
- Etape2 : le client envoie au répertoire une requête contenant les critères du service à rechercher, la politique de sécurité décrivant les critères que doivent satisfaire les services recherchés et un certificat permettant au répertoire d'authentifier le serveur en cas de besoin.
- Etape3 : le répertoire cherche une correspondance entre la requête des clients et les profils des services enregistrés.
- Etape4 : pour vérifier les politiques des clients et des serveurs, le répertoire vérifie les certificats fournis par ces deux éléments afin de les authentifier et appliquer les politiques de sécurité.
- Etape5 : après avoir authentifié les deux parties, le répertoire vérifie si les services sélectionnés répondent aux contraintes de sécurité imposés par la politique du client, ensuite vérifie si le profil du client correspond aux contraintes énoncées dans la politique des serveurs sélectionnés.
- Etape6 : après avoir vérifié les correspondances entre les politiques de sécurité et les profils des intervenants, le répertoire renvoie la réponse correspondant à la requête du client et qui satisfait toutes les contraintes de sécurité.

B. Intergiciel pour la découverte de services sécurisée

Les AOS actuelles ont pour avantage d'êtres flexibles et adaptables à toutes les plateformes de déploiement. La solution de sécurité proposée dans cette thèse doit donc être capable

d'avoir une interface assez flexible afin de permettre à toutes les applications ubiquitaires de bénéficier des fonctionnalités de la découverte de service sécurisée. Pour cette raison, nous avons décidé de fournir une couche intergicielle proposant des interfaces génériques accessibles via tous types de plateformes. Cette interface propose des méthodes permettant de :

- Déclarer les services en publiant leurs profils descriptifs.
- Exporter les politiques de sécurité liées à la découverte.
- Exporter les certificats d'authentification.

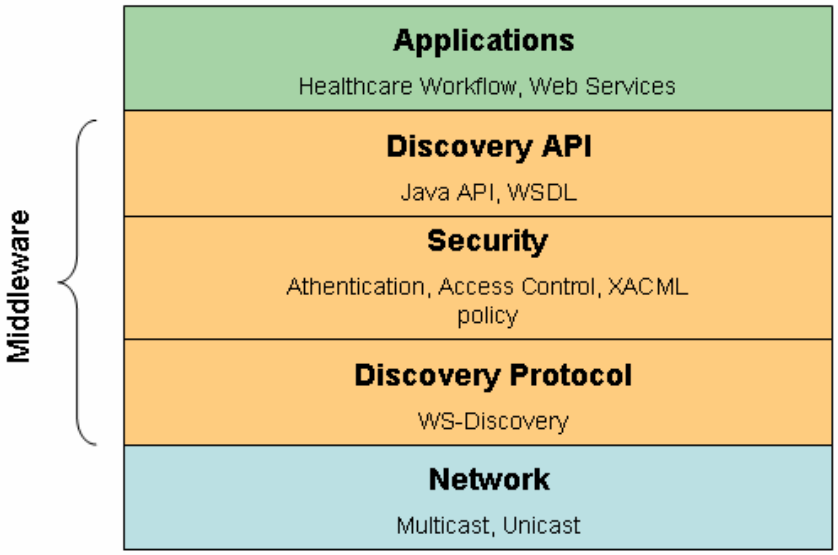


Figure 6: Pile intergicielle

La pile intergicielle proposée dans cette thèse se compose de trois couches :

- Couche du protocole de découverte de services : cette couche spécifie les formats de messages de découverte ainsi que le système de traitement des messages. Dans notre implémentation nous avons choisi de suivre les spécifications du protocole WS-Discovery qui a pour avantage d'implémenter les deux topologies de découverte (décentralisée et centralisée basée sur les répertoires). Les messages d'annonce sont envoyés via multicast et ceux de réponse via unicast.
- Couche de sécurité : cette couche fournit les méthodes utilisées pour authentifier les utilisateurs, interpréter et appliquer les politiques de sécurité.
- Couche interface d'accès : c'est une interface visible à tous les utilisateurs afin d'accéder aux fonctionnalités fournies par les autres couches.

C. Discussion

Dans cette partie de la thèse, j'ai introduit pour la première fois la notion de politique de découverte permettant à tous les utilisateurs de spécifier des politiques de sécurité afin de protéger la découverte de services. Cette politique permet d'exprimer tous les besoins spécifiés dans le modèle de sécurité de référence.

Chapter V. Passage à l'échelle pour la découverte de services sécurisée

A. Passage à l'échelle

Les solutions décrites précédemment sont limitées pour le passage à l'échelle. En effet, si le service requis par le client ne se trouve pas dans le même sous réseau ou dans le répertoire local, le processus de découverte est stoppé sans prendre en compte les services publiés dans d'autres réseaux et répertoires. De plus, dans le cas d'un système mettant en jeu plusieurs milliers de serveurs et clients, les solutions actuelles seraient vite surchargées et incapables de prendre en charge un tel nombre de participants. La solution la plus recommandée pour le passage à l'échelle préconise l'utilisation d'un système de répertoires distribués. La distribution des répertoires doit être régie selon une architecture de déploiement bien spécifique qui dépend du système d'indexation et de routage escompté. Trois architectures sont alors possibles (figure 7) :

- Plate : tous les répertoires sont interconnectés de façon non structurée et communiquent entre eux par broadcast. Il n'y a pas de stratégie d'indexation et de recherche particulière ; chaque nœud doit contacter tous les autres s'il veut publier un service ou en chercher un. Cette architecture offre un meilleur passage à l'échelle par rapport aux solutions simples, mais atteint vite ses limites dans le cas où des milliers de registres sont mis en jeu.
- Hiérarchique : la stratégie de déploiement correspond à un arbre binaire (ou n-aire) d'indexation. L'indexation et la recherche de services suit la logique de parcours de l'arbre, ainsi la complexité maximum d'une recherche est de l'ordre de o(log(n)). Comparé à l'architecture précédente, l'architecture hiérarchique offre un meilleur passage à l'échelle quelque soit le nombre de nœuds, mais elle peut se révéler coûteuse en terme de maintenance et surtout dans la gestion des pannes. En effet, si un nœud tombe, il est très difficile de récupérer les données maintenues, car il n'y a pas de système de duplication.
- P2P basé sur les tables de hachage distribuées (DHT) : cette solution a beaucoup de succès auprès des applications d'échange de fichiers. Chaque donnée dans le système a une valeur de hachage, chaque nœud est en charge de la mémorisation de plusieurs valeurs de hachages pointant vers les nœuds détenant les profils des services. Quand un nouveau profil de service est entré dans un nœud, ce dernier va générer une valeur de hachage correspondant au profil, puis va envoyer le lien de cette information à tous les nœuds en charge de cette valeur de hachage. Ainsi, s'il y a une nouvelle requête de service, le nœud relayant cette requête va calculer sa valeur de hachage et contacter tous les nœuds en charge de cette valeur afin de récupérer les coordonnées du nœud qui détient l'information. Cette solution, contrairement aux deux précédentes, passe très bien à l'échelle et ne requiert pas un système de maintenance très compliqué. Pour cette raison nous avons choisi d'adopter cette configuration.

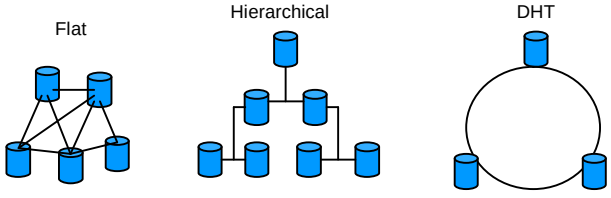


Figure 7 : Architectures pour les répertoires distribués

B. Découverte de services anonyme

Les mécanismes de sécurité décrits précédemment, utilisés dans le but de protéger la découverte de services, ne peuvent pas être appliqués tels quels pour une architecture de répertoires distribués. Le premier problème qui se pose concerne la confiance accordée aux répertoires ; il n'est pas possible de faire confiance à tous les répertoires y compris ceux qui ne sont pas connus. Le deuxième problème concerne l'utilisation du chiffrement basé sur les attributs ; si les messages de découverte sont chiffrés selon les profils des services recherchés, les répertoires ne pourront pas accéder aux messages afin de faire la correspondance avec les profils des services. Dans une solution précédente [TRA07] nous avons proposé de chiffrer certaines parties des messages et laisser les attributs de correspondance en clair, mais cette hypothèse allait à l'encontre du modèle de sécurité concernant la protection des données privées des clients et des services (qui peuvent être restreints). Afin de palier à cette faille, nous avons proposé de rendre les messages partiellement chiffrés anonymes ; ainsi les attributs privés qui sont en clair ne peuvent pas être reliés à leurs propriétaires et la correspondance données/utilisateurs sera impossible à établir. Différentes techniques d'anonymisation de messages peuvent être utilisées, on peut citer par exemple : les MIX [BER01] [KES98], les Crowds [REI98], ou bien les systèmes de routage basés sur les chiffrements co-centrique (onion routing) [KAT07].

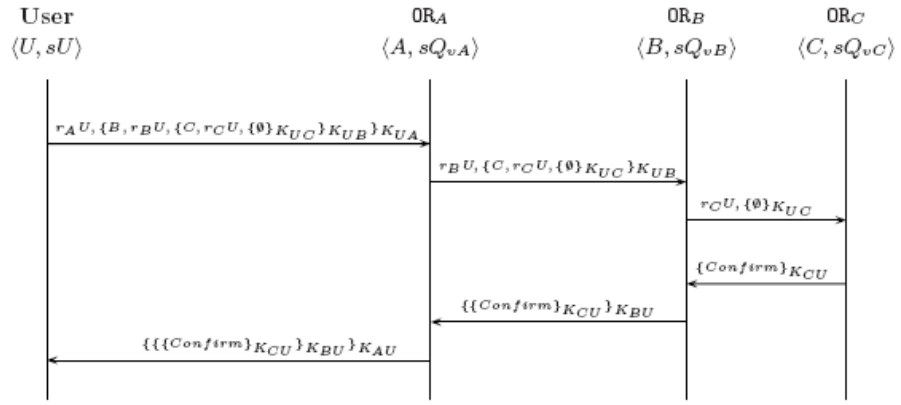


Figure 8: Routage anonyme basé sur le chiffrement co-centrique

Cette dernière technique (décrite en figure 8) permet notamment d'acheminer des messages anonymes dans les deux sens (envoi et réception). Ainsi, l'utilisateur voulant publier ou découvrir un service devra d'abord choisir un ensemble de nœuds intermédiaires jouant le rôle de relais vers le répertoire à contacter. L'envoyeur doit chiffrer le message successivement avec les clefs publiques chaque nœud du chemin de façon à ce que chaque participant ne puisse accéder qu'aux informations de routage du nœud précédent et du nœud suivant, à l'exception du répertoire qui aura accès aux informations de routage du nœud qui le précède et au message de l'utilisateur. Ainsi les nœuds intermédiaires ne connaissent pas le contenu du message, et le répertoire qui a accès au message ne connaitra pas l'expéditeur.

C. Description du protocole

Après avoir décrit les solutions envisageables pour le passage à l'échelle de la découverte de services et pour l'anonymisation de l'échange des messages de découverte, nous allons décrire l'architecture mettant en œuvre ces mécanismes au service de la découverte sécurisée (voir figure 9).

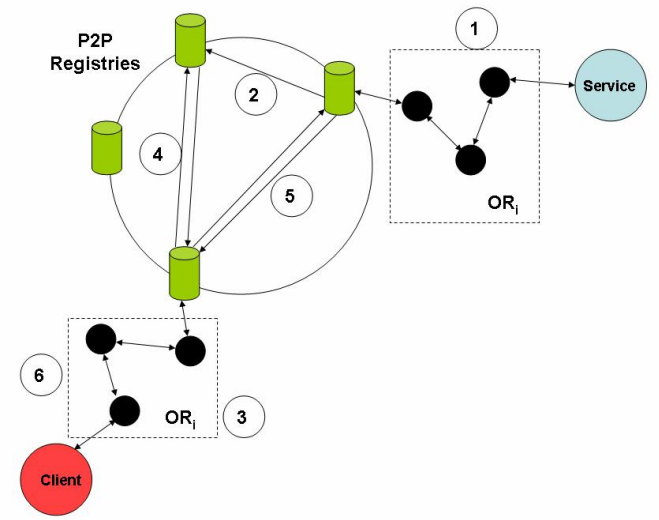


Figure 9: Architecture pour la découverte sécurisée de services à travers des répertoires distribués

- Initialisation : les clients et les services doivent d'abord choisir leurs chemins de routage des messages composés d'au moins trois nœuds intermédiaires.
- Etape1 : le serveur publie les profils de ses services en contactant le répertoire local de façon anonyme. Le message est relayé de façon anonyme à travers les nœuds intermédiaires jusqu'au répertoire qui une fois les données enregistrées renvoie un accusé de réception au serveur en utilisant le chemin anonyme inverse.
- Etape2 : le répertoire hache les valeurs des mots-clefs décrivant les services et envoie les clefs de hachages aux répertoires en charge du stockage de ces clefs.
- Etape3 : le client envoie sa requête de service au répertoire le plus proche, toujours de façon anonyme.
- Etape4 : le répertoire cherche localement les correspondances avec la requête. Si elles n'existent pas, les mots-clefs de la requête sont hachés et relayés vers le système pair à pair.
- Etape5 : le répertoire en charge des clefs de hachage correspondant aux attributs de la requête retourne le pointeur vers le répertoire en stockant les données des services recherchés.
- Etape6 : le répertoire local, après avoir récupéré les informations des services, retourne la réponse à l'utilisateur.

D. Discussion

Contrairement aux deux solutions précédentes, celle-ci permet une découverte sécurisée qui ne se limite pas aux réseaux locaux, mais qui s'étend à tous les autres réseaux et domaines. Cette solution passe mieux à l'échelle sans pour autant sacrifier la sécurité. De plus, le fait d'utiliser un système P2P offre une meilleure robustesse aux pannes et aux failles.

Chapter VI. Analyse de performance de la découverte de service sécurisée

Le déploiement d'un système de service de découverte peut se faire de plusieurs manières : centralisé, décentralisé, répertoires, ah-hoc, pair à pair … Le choix du type de déploiement

dépend de plusieurs facteurs comme le type et la taille du réseau, le nombre d'utilisateurs, la quantité de ressources qui sera mise à disposition, ainsi que l'autonomie énergétique des équipements à déployer. Ces facteurs doivent être connus par l'administrateur en charge du déploiement d'un système de découverte afin de proposer une configuration optimale. Il existe des outils de simulation et de modélisation permettant de prendre comme paramètres d'entrées ces facteurs environnementaux afin de fournir des résultats qualitatifs et quantifiables donnant des indications à l'administrateur afin qu'il aboutisse au meilleur choix de déploiement. Dans cette partie de la thèse nous avons développé un modèle de découverte de services prenant en compte les mécanismes de sécurité décrits dans les solutions centralisées et décentralisées. Notre modèle de performance, basé sur une représentation Markovienne (mathématique), a été validé par un simulateur décrivant le même système. Seuls les aspects applicatifs ont été pris en compte lors de la modélisation ; les contraintes réseaux ont été négligées vu qu'elles disposent déjà de plusieurs modèles de performance.

A. Modèle de découverte sécurisée

1. Modèle centralisé

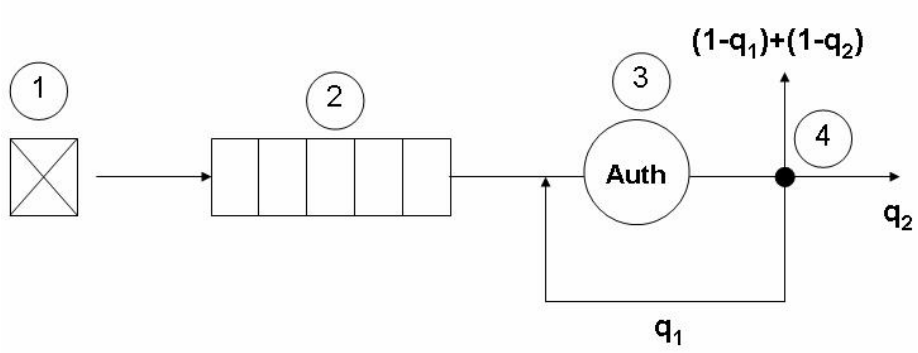


Figure 10 : Modèle centralisé

La figure 10 représente le traitement sécurisé d'un message de découverte de services dans un répertoire de confiance. Les arrivées des requêtes de services sont représentées par un générateur de variables aléatoires (1) qui simule les arrivées des requêtes selon un processus de Poisson. Une fois arrivées, les requêtes sont stockées dans la mémoire tampon du répertoire (2) en attendant d'être traitées. Le traitement de la requête consiste d'abord à authentifier les serveurs sélectionnés et vérifier la politique de sécurité du client (3), ensuite authentifier le client et vérifier la politique de sécurité des services (4). La probabilité q_1 correspond à la probabilité qu'un service corresponde à la politique du client. La probabilité q_2 correspond à la probabilité que le client corresponde à la politique du service.

2. Modèle décentralisé

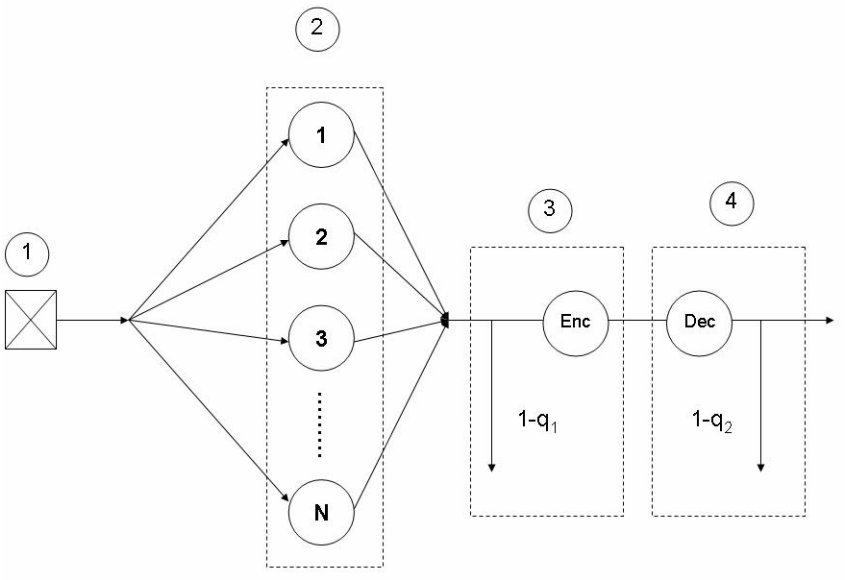


Figure 11: Modèle décentralisé

La figure 11 quant à elle, représente le traitement sécurisé d'une requête envoyée simultanément (via multicast) à tous les serveurs du système afin qu'ils puissent déchiffrer le message et traiter la requête. Tout comme dans le modèle centralisé, les arrivées des requêtes dans le système (1) sont supposées aléatoires et suivant une distribution de Poisson. Tous les serveurs ayant reçu la requête tentent de la déchiffrer (2) à l'aide des clefs privées dont ils sont propriétaires et qui correspondent aux attributs des profils de leur services. Une fois la requête déchiffrée avec une probabilité q_1, le service chiffre sa réponse pour la renvoyer au client (3). Le client doit alors essayer de déchiffrer la réponse du service à l'aide des clefs privées correspondant aux attributs de son profil.

3. Hypothèses du modèle

Afin que le comportement du système soit correctement quantifié et modélisé, il est nécessaire d'énoncer certaines hypothèses autour des différentes étapes par lesquelles passent les systèmes de découvertes. Ces hypothèses ont pour but de se rapprocher le plus possible du comportement réel d'un système, tout en simplifiant la modélisation et le calcul.

- Le temps de traitement d'une requête : afin de modéliser le temps de traitement d'une requête il est impératif de distinguer les deux types d'architectures. Pour l'architecture décentralisée, le temps de traitement consiste essentiellement en la durée de chiffrement et déchiffrement des messages. Suite aux mesures effectuées lors du chiffrement/déchiffrement des messages de découverte, nous avons observé une variabilité de la durée de traitement qui était indépendante de la longueur du message ou de la clef de chiffrement. Ceci nous a amené à supposer que le temps de traitement est très proche d'une distribution exponentielle. Nous avons donc décidé de modéliser le temps de chiffrement/déchiffrement d'une requête comme étant une variable aléatoire suivant une distribution exponentielle de moyenne $1/\mu$. Pour l'architecture centralisée, le temps de traitement consiste essentiellement en la durée d'authentification et du temps de vérification des politiques de sécurités. Nos mesures ont montré que la variation du temps de traitement était corrélée à la complexité de la politique de sécurité.
- Temps inter-arrivées des requêtes : les requêtes des clients arrivent au système suivant le processus de Poisson.

- Classe de trafic : dans le cas du modèle décentralisé on peut imaginer que certains services puissent être plus populaires que d'autres, ou que certains services sont dupliqués. Pour ces raisons il est important de distinguer différentes classes de trafic dont les probabilités q1 ne sont pas équivalentes et varient en fonction de la popularité des services ou du nombre de duplicatas.

B. Modèle Markovien

Les modèles mathématiques de performance permettent dans certains cas d'évaluer les performances d'un système en obtenant des résultats assez proches de ceux obtenus avec des simulateurs. L'avantage d'utiliser ces modèles mathématiques réside dans la durée de calcul : alors qu'une simulation d'un événement peut prendre plusieurs heures, voire quelques jours, les modèles analytiques eux ne prennent que quelques secondes, le temps de résoudre quelques systèmes d'équitations. De plus ces modèles sont très facilement extensibles et modifiables. Les chaînes de Markov constituent un outil fondamental pour modéliser les processus en théorie des files d'attente et en statistiques. Etant donné que les modèles de découvertes décrits précédemment sont représentés par un système de files d'attentes, les chaînes de Markov peuvent être utilisées pour l'étude de performance du processus de découverte de services. En mathématiques, une chaîne de Markov est un processus stochastique possédant la propriété markovienne. Dans un tel processus, la prédiction du futur à partir du présent ne nécessite pas la connaissance du passé.

1. Modèle Markovien centralisé

Pour chaque requête de découverte entrant dans le système, il peut y avoir deux cycles d'authentification et de vérification de politique de sécurité. Afin de prendre en compte ces deux cycles de traitement de la requête, nous avons décidé de représenter le processus de découverte en utilisant une chaine de Markov bidimensionnelle ; chaque dimension représente un cycle d'authentification (Voir figure 12).

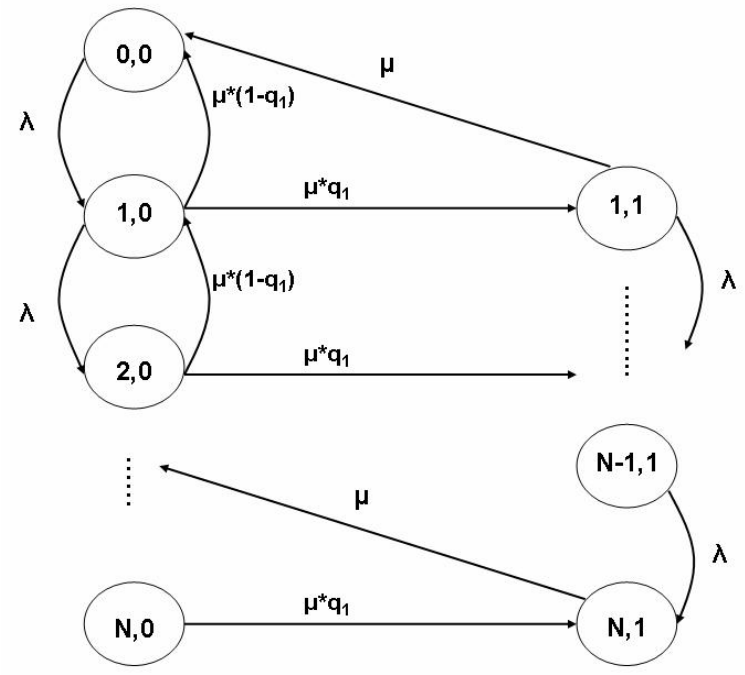


Figure 12 : Chaine de Markov pour la découverte centralisée

La première dimension de la chaîne de Markov (A) représente le nombre de requêtes stockées dans la mémoire tampon du répertoire, ainsi que la requête encours de traitement. La deuxième dimension (B) est un booléen qui indique le cycle de traitement de la requête ; 0 pour le premier cycle et 1 pour le deuxième. Les arrivées de requêtes sont représentées avec le taux d'arrivée λ. A chaque arrivée de client le paramètre (A) est incrémenté. Après le premier cycle de traitement, la requête passera au deuxième cycle avec une probabilité q_1, ou quittera le système avec une probabilité $(1-q_1)$. Si la requête se trouve au deuxième cycle (B=1) elle quittera le système après le traitement.

2. Modèle de Markov décentralisé

La représentation du système décentralisé a été divisée en deux chaines de Markov. La première (figure 13) représente le processus d'arrivées des requêtes avec un taux λ et les tentatives de déchiffrement des messages de la part des serveurs. La deuxième (figure 14) représente le processus de chiffrement de la réponse côté serveur et du déchiffrement de la réponse côté client.

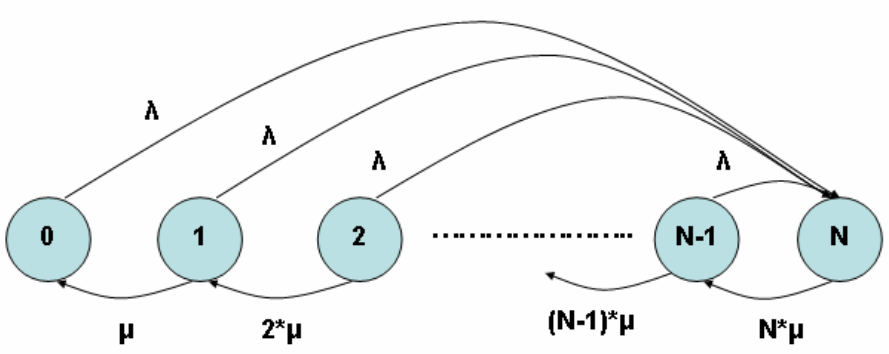


Figure 13: Chaine de Markov pour la découverte décentralisée

Dans la figure 13 chaque état représente le nombre de serveurs en cours de traitement (en train de déchiffrer une requête)

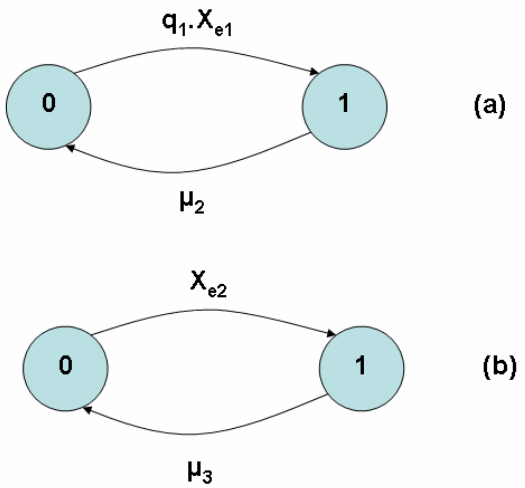


Figure 14 : Chaine de Markov pour la découverte décentralisée

Dans la figure 14, le paramètre X_{e1} représente le débit de sortie des services déchiffrant les messages. Le paramètre X_{e2} représente le débit de sortie du chiffrement des réponses par les services.

3. Probabilités d'authentification

Les probabilités q_1 et q_2 représentent, dans les deux modèles, les probabilités d'authentification (déchiffrement, authentification et correspondance avec les politiques de sécurité). Ces probabilités dépendent du nombre d'éléments dans le système et de la taille du vocabulaire sémantique du contexte applicatif. Le vocabulaire contient un ensemble de mots liés au domaine applicatif. Par exemple, le vocabulaire du domaine médical contient des mots comme : scanner, radiologie, chirurgie, dermatologie, cancérologie, médicament, pharmacie etc… alors que pour le domaine de la découverte de services, le vocabulaire contient des mots décrivant les services (imprimante, banque …) et les identités ou les rôles des utilisateurs (professeur, étudiant, chercheur …) participant à la découverte.
La probabilité de correspondance *P* d'un élément dans un vocabulaire est représentée par un groupe d'attributs *x* dans un système et est définie par la probabilité que ces attributs appartiennent au vocabulaire *V* :

$$P(x):\begin{cases}\dfrac{C}{V}; size(x)=1\\ \dfrac{C_C^x}{C_V^x}; size(x)>1\end{cases}$$

C. Simulateur et validation du modèle Markovien

Afin de vérifier la qualité et confirmer les résultats obtenus par le modèle Markovien, nous avons choisi de développer un simulateur décrivant les deux architectures de découvertes (centralisé et décentralisé). Si les résultats obtenus par le simulateur se révèlent être proches de ceux obtenus par le modèle mathématique nous pouvons donc admettre que ce dernier est valide et peut être utilisé pour l'étude de performance de la découverte de services sécurisée.
Le simulateur génère des événements liés au scénario décrit dans les deux modèles précédents. Les arrivées des requêtes ainsi que les temps de traitement sont générés aléatoirement à l'aide une librairie java appelée SSJ [SSJ] fournissant des générateurs de variables aléatoires. Nous avons donc comparé les résultats de certains paramètres de performance comme le taux de rejet (le rejet survient lorsque le système est plein et toutes les ressources sont occupées à traiter les requêtes) ou le taux d'utilisation des ressources.

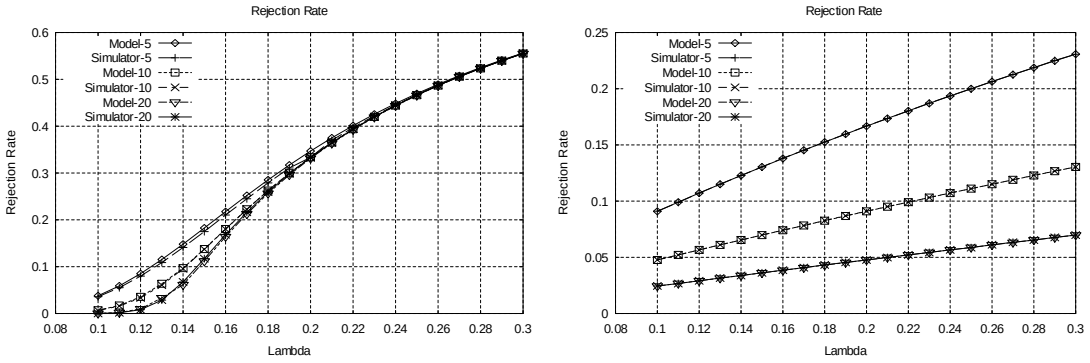


Figure 15 : Comparaison du taux de rejet

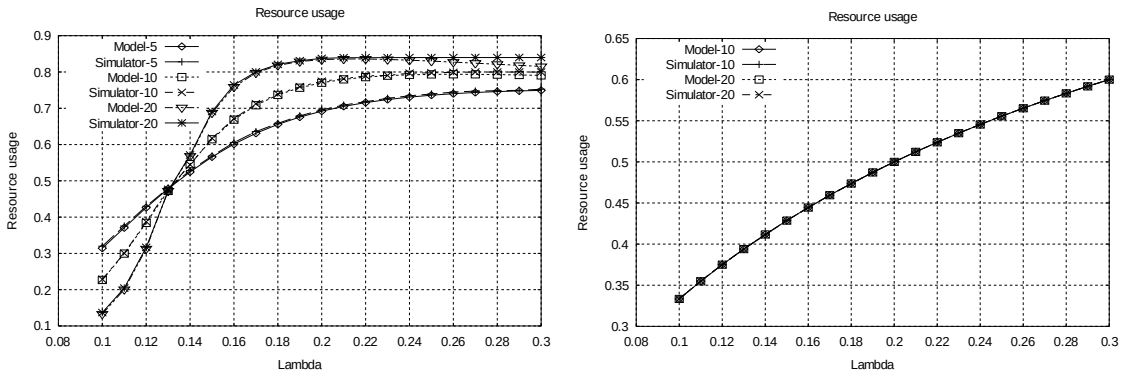


Figure 16 : Comparaison du taux d'utilisation des ressources

Les figures 15 et 16 montrent que les résultats obtenus grâce aux simulateurs sont très proches de ceux obtenus avec le modèle Markovien, ce qui prouve que les hypothèses choisies précédemment se rapprochent du fonctionnement réel du système. Grâce à cette validation nous sommes maintenant capables de calculer les paramètres de performance comme : le temps de séjour d'une requête, le nombre moyen de requêtes traitées, la disponibilité des ressources, la consommation en mémoire, le temps moyen d'activité des serveurs et des répertoires, le taux de succès des requêtes…

En analysant les paramètres de performances obtenus grâce au modèle, nous pouvons comparer les architectures centralisées et décentralisées afin de connaître les avantages et les inconvénients de chaque solution. Le tableau suivant résume partiellement certaines de ces différences.

Paramètres de performance	**Rejet**		**Nombre de requêtes**		**Temps de service**	
	Centralisé	Décentralisé	Centralisé	Décentralisé	Centralisé	Décentralisé
Augmentation de la taille du buffer	-	=	+	=	+	=
Augmentation de la probabilité de correspondance	=	=	-	=	+	=
Augmentation du nombres de serveurs dans le système	=	-	=	+	=	=

D. Evaluation de l'impact des attaques de dénis de service

La seule attaque envisageable contre les systèmes de découverte sécurisée présentés dans cette thèse est l'attaque de dénis de service (DdS) dirigée vers les serveurs et les répertoires. Il s'agit pour l'attaquant de générer de fausses requêtes et de les envoyer de façon intempestive vers les cibles à atteindre. Les victimes de cet envoi massif de fausses requêtes vont essayer de traiter tous les messages reçus jusqu'à arriver à saturation et ne plus être capable de fournir son service. Ce genre d'attaque affecte plus la solution décentralisée où les serveurs doivent d'abord déchiffrer les messages avant de les traiter ; le déchiffrement étant coûteux en termes de CPU.

Il existe deux types de sources d'attaques de DdS : la première, dite mono-source, dans laquelle l'attaquant génère les faux messages d'une même source en modifiant éventuellement les adresses IP pour leurrer le serveur. La deuxième, multi-sources, dans laquelle l'attaquant se sert de plusieurs ordinateurs zombis contrôlés à distance afin d'envoyer les fausses requêtes.

Afin de contrecarrer les attaques de type DdS, il existe des techniques appelées puzzles cryptographiques qui consistent à lancer un défi cryptographique à la source d'une requête : si celle-ci réussit à résoudre le calcul, sa requête sera traitée par le serveur. Ainsi les attaques DdS mono-sources sont éradiquées et les multi-sources fortement ralenties.

E. Modèle d'attaque

1. Modèle

Afin de connaître l'impact des attaques de DdS sur le système de découverte sécurisée, nous avons décidé d'étendre le modèle de performance décrit précédemment et d'y introduire un trafic supplémentaire représentant les fausses requêtes de découvertes générées par les attaquants. Le trafic corrompu est représenté par un taux d'arrivées variable de moyenne λ_{attack} et est mêlé au trafic normal. Dans cette étude nous allons comparer deux systèmes soumis à des attaques, l'un sera protégé et l'autre pas.

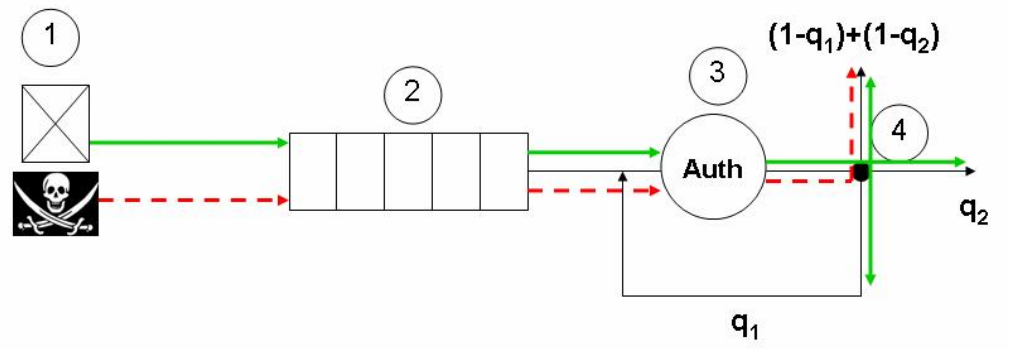


Figure 17 : Modèle d'attaque pour l'architecture centralisée

Dans le modèle centralisé (figure 17) les requêtes corrompues peuvent occuper de l'espace dans la zone tampon, mais sont rejetées du système dès que le répertoire procède au premier cycle d'authentification.

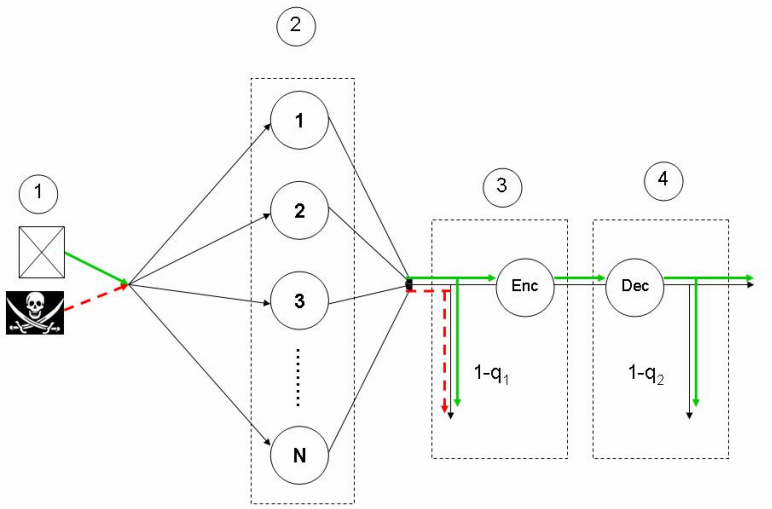


Figure 18: Modèle d'attaque pour l'architecture décentralisée

Dans le modèle décentralisé (figure 18) les requêtes corrompues sont rejetées du système juste après la tentative de déchiffrement par les serveurs.

Le trafic global du système prend en compte les deux types de trafics (normal et corrompu), ce qui va directement influer sur le calcul de la probabilité d'authentification à l'entrée du système (déchiffrement par les serveurs et premier cycle d'authentification du répertoire). La nouvelle formule décrivant la probabilité de d'authentification s'écrit :

$$q_{attack} = \frac{q_1 \cdot \lambda}{\lambda + \lambda_{attack}}$$

2. Impact d'une attaque de DdS sur les systèmes protégés et non protégés

Afin d'étudier l'impact des attaques de DdS sur la découverte de sécurité, nous avons comparé les paramètres de performances obtenus dans un système non protégé contre ces attaques et un système mettant en œuvre les puzzles cryptographiques.

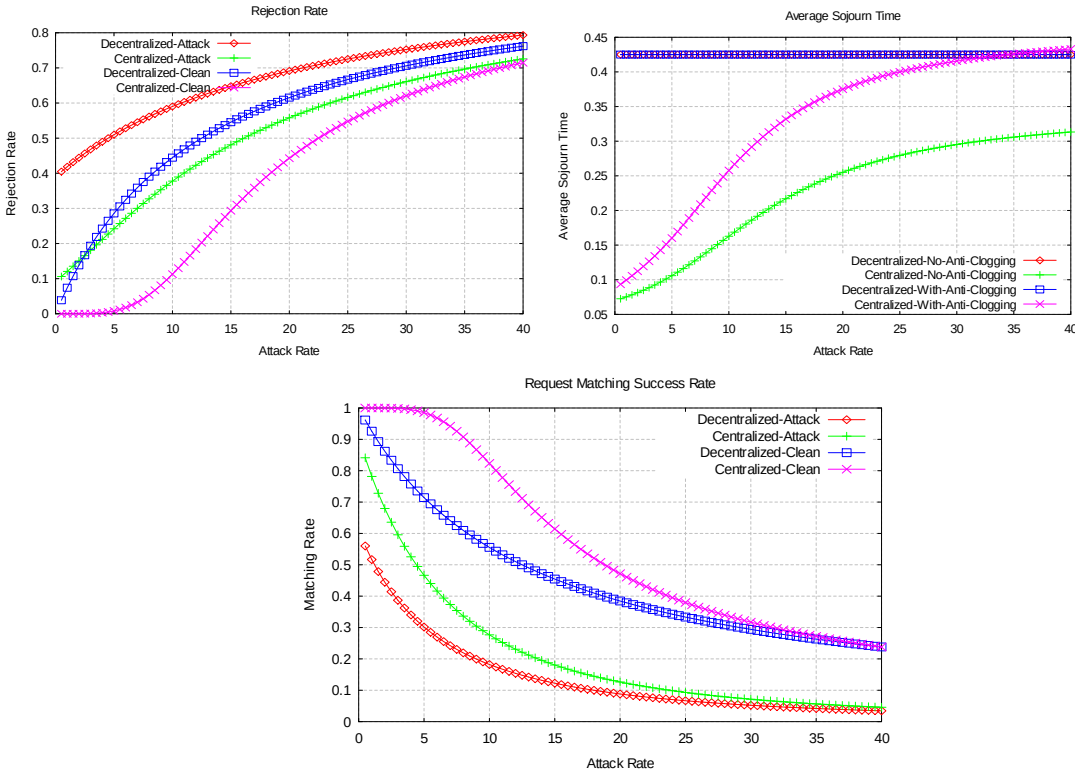


Figure 19: Comparaison des paramètres (rejet, temps de séjour, probabilité de correspondance) de performance lors d'une attaque de DdS

On remarque sur la figure 19 que les performances des systèmes non protégés se dégradent deux fois plus vite que les autres, sauf pour le temps de séjour d'une requête dans le système décentralisé qui, lui, reste inchangé à cause de la distribution des serveurs lors du déchiffrement.

Le modèle d'attaque nous permet donc d'estimer de façon analytique les dégâts que peuvent provoquer certaines attaques de DdS dirigées vers des systèmes non protégés. Ce modèle peut donc être utilisé comme extension de chaînes de Markov cachées servant à analyser le trafic et détecter les attaques de ce genre.

F. Discussion

Dans cette partie de la thèse nous avons proposé deux modèles analytiques pour évaluer l'impact de l'utilisation des mécanismes de sécurité dans les protocoles de découverte de services. Il s'agit à notre connaissance du premier modèle mathématique traitant cette problématique. Les résultats des paramètres de performance obtenus grâce à ce modèle sont très importants pour le choix de déploiement de l'une des architectures de découvertes possibles, à savoir centralisée ou décentralisée. Il est possible d'évaluer rapidement des paramètres comme la robustesse, l'efficacité, l'utilisation de la ressource, la disponibilité, la taille des messages ou le taux de traitement des messages. Ce modèle de performance a permis ensuite de développer un outil de modélisation des attaques de DdS permettant d'étudier le comportement du système de découverte face à ce genre d'attaques.

Chapter VII. Contexte dans les systèmes de découverte de services

Dans les systèmes ubiquitaires, l'utilisateur est entouré par un environnement intelligent dans lequel évoluent des équipements intelligents (RFID, capteurs …) de façon complètement transparente. Cet environnement est en évolution perpétuelle affectant de la même façon le comportement des utilisateurs et des applications qui lui appartiennent. Ces éléments doivent donc s'adapter dynamiquement à ces changements de façon à adapter leurs fonctionnalités et maintenir un échange d'informations correctes. Cette évolution dépend de plusieurs éléments dits contextuels. Le contexte joue donc un rôle primordial dans l'informatique diffuse.
L'utilisation des informations contextuelles dans la découverte de services permet d'affiner et d'adapter la recherche dynamique des éléments ubiquitaires. Le contexte est lié à toute information pouvant caractériser l'état dans lequel se trouve un élément. Les systèmes de découverte de services peuvent utiliser le contexte afin de fournir une correspondance requête/service plus précise. Les différentes approches d'intégration du contexte dans la découverte de service se sont pour la plupart limitées à l'exploitation de l'information contextuelle provenant des capteurs sans aucune réelle interprétation. Pour cette raison, nous avons proposé une véritable sémantique contextuelle permettant de raisonner correctement sur les informations fournies par les capteurs. Les informations contextuelles comme la localisation sont récoltées des capteurs, puis interprétées et traitées afin d'en dériver des informations plus riches et plus complexes telles que la proximité ou la distance. Nous avons de plus intégré ces informations aux politiques de découvertes décrites précédemment, afin de faire intervenir le changement de contexte dans la prise de décision lors de l'application de la sécurité durant le processus de découverte de services.

A. Représentation du contexte

Les ontologies sont utilisées pour la représentation et la gestion d'une base de connaissance ayant des propriétés taxonomiques communes. En plus de la classification et de la catégorisation des objets ontologiques, il est possible d'établir des relations logiques entre les différents éléments d'une même ontologie et entre les ontologies elles mêmes. Dans le domaine contextuel, les ontologies sont utilisées afin de classifier les différents aspects ontologiques selon leur nature (lieu, temps, température …). Il existe des langages dédiés aux ontologies qui permettent de les représenter et les manipuler selon des règles bien spécifiques. Le langage OWL (Web Ontology Language) permet de définir et représenter une ontologie selon un schéma bien défini (RDS). Il existe une ontologie définissant le domaine contextuel d'un utilisateur, décrite en langage OWL et appelée CoOL. CoOl propose quatre classes de contexte : l'utilisateur (son activité, son rôle, ses préférences), l'environnement (température,

lieux, distance…), la plateforme (matériel ou logiciel) et le service. La figure 20 décrit l'ontologie CoOl.

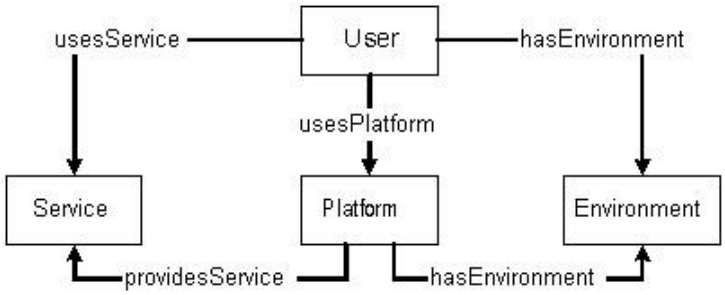


Figure 20: Ontologie CoOL

B. Raisonnement contextuel

Pour le raisonnement contextuel, nous distinguons deux approches complémentaires : le raisonnement ontologique ou le raisonnement basé sur les règles d'inférences. Le raisonnement ontologique a une portée assez limitée étant donné qu'elle se base sur des relations binaires entres les éléments. Pour cette raison, nous avons choisi d'utiliser des règles d'inférences via le moteur d'inférence Jess qui sera combiné à la description ontologique. La figure 21 décrit l'interaction entre le moteur d'inférence et les ontologies contextuelles.

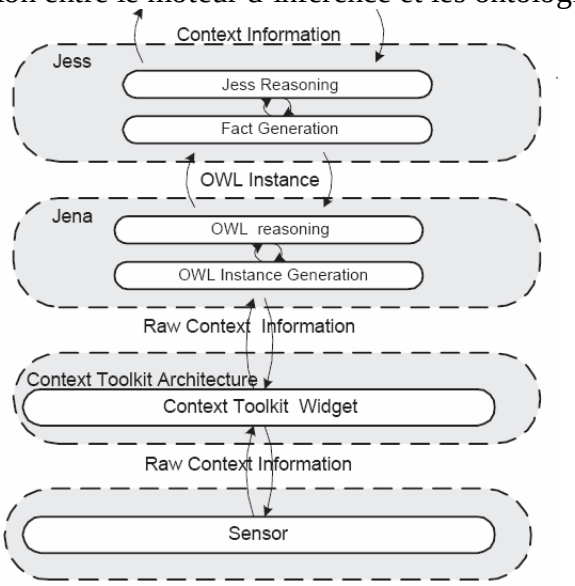


Figure 21: Raisonnement contextuel

C. Découverte de service contextuelle

Le module de raisonnement contextuel peut être rajouté à l'architecture de découverte sécurisée de telle façon que le module de gestion des politiques de découverte prenne aussi en charge les informations contextuelles. Dans ce cas, la politique pourra décrire des fonctions ainsi que des données liées au contexte. Ainsi la localisation d'un utilisateur vis-à-vis de celle d'un serveur mobile peut être considérée comme étant un critère de prise de décision lors du traitement d'une requête. La figure 21 décrit un exemple de politique de sécurité limitant la découverte des services médicaux à moins de 2000 mètres.

```
<Apply FunctionId="isCloseTo">
  <Apply FunctionId="findLocation">
    <SubjectAttributeDesignator
      DataType=GPSLocation AttributeId="SubjectLocation"/>
  </Apply>
 <AttributeValue
    DataType="integer">2000 meters</AttributeValue>
  <Apply FunctionId="string-one-and-only">
    <SubjectAttributeDesignator
       DataType=string AttributeId="PatientID" />
  </Apply>
</Apply>
```

Figure 21: Politique de découverte contextuelle

Chapter VIII. Conclusion

Dans cette thèse nous nous sommes focalisés sur les problèmes liés à la sécurité et à la performance des systèmes de découverte de services dans l'informatique diffuse. Nous avons proposé plusieurs mécanismes et solutions afin de combler les failles et sécuriser ces systèmes pour garantir un déploiement sûr et robuste.

Dans la première partie de ce travail de recherche nous avons présenté un état de l'art autour des différents protocoles de découverte de services. Nous avons ensuite analysé le mode de fonctionnement de ces protocoles afin d'y déceler les failles et les faiblesses de sécurité qui pourraient être utilisées par des attaquant afin de compromettre le bon fonctionnement de la découverte de services. A partir de cette analyse nous avons déduit un modèle de sécurité proposant des consolidations au niveau du protocole afin de protéger la découverte contre toutes les failles détectée dans le modèle d'attaque. La seconde partie de ce document est dédiée aux solutions de sécurité que j'ai proposées tout au long de ma thèse pour protéger la découverte de services dans les systèmes ubiquitaires. Nous avons proposé trois solutions : la première, dédiée aux architectures locales décentralisées, suggère de chiffrer tous les messages de découvertes selon une technique de chiffrement basée sur les attributs et permettant de restreindre l'accès des messages uniquement aux éléments authentifiés. La deuxième solution, consacrée aux architectures locales centralisée, propose d'utiliser des politiques de sécurité dédiées à la découverte de services. Ces politiques sont interprétées et appliquées par les registres centralises afin de garantir la sécurité en fonction des préférences des utilisateurs. Enfin la troisième solution, dédiée aux architectures du type internet, repose sur un système d'indexation pair à pair et un routage de messages anonyme.
Afin d'évaluer l'impact de l'utilisation de ces mécanismes de sécurité sur la performance des systèmes de découverte de services nous avons proposé un modèle analytique d'évaluation de performances. Ce modèle permettant d'évaluer plusieurs paramètres de performances du système et de comparer les résultats obtenus avec chaque type d'architecture. Une extension de ce model a permit d'aboutir à un modèle d'attaques permettant d'étudier l'impact des attaques de DdS sur les systèmes de découverte.
La dernière partie de mes travaux concerne l'utilisation des informations contextuelles, décrivant l'environnement des utilisateurs, dans les systèmes de découverte de services sécurisée les rendant ainsi plus adaptables à l'environnement dynamique des applications.

Table of Contents

Abstract....5
Table of Contents....31
List of Figures....35
List of Tables....37
List of Publications....38
Introduction....41
A. Pervasive and Ubiquitous Computing....42
B. Distributed Systems....44
C. Service Oriented Architecture (SOA)....44
D. Web Services....45
E. Peer to Peer Systems....46
F. Workflow Architecture....46
G. Security requirement in Pervasive Systems....46
H. Contributions....48
I. Outline....49
Chapter I. Service Discovery....51
A. Introduction....51
B. Definition....51
C. Service Discovery Components Design....52
D. Service Discovery Protocols....53
1. Salutation....54
2. Service Location Protocol (SLP)....54
3. Jini Lookup Service (JLS)....55
4. UDDI....55
5. UPnP....56
6. WS-Discovery....56
7. Service Discovery Protocol (SDP): Bluetooth....57
8. Service Discovery in Ad-Hoc Networks....58
E. Matching and Semantics....58
1. Matching....58
2. Ontology Based Service Discovery....59
F. Context Awareness and Service Discovery....59
G. Threats and Security Requirements....60
1. Threats and Attacks....60
2. Security Requirements for Service Discovery....64
H. Approaches Secure Service Discovery....66
1. Access Control on the Service Side....66
2. Registry-Based Architecture....66
3. Privacy Issues for the Service Discovery....67
4. Registry-less Architecture....67
Chapter II. Securing Decentralized Service Discovery....69
A. Introduction....69
B. Technical Background....69
1. Identity Based Encryption....69
2. Attribute Based Encryption....70
3. Attribute Based Algorithm....71
4. Private Key Generation: Online Vs Offline....71
C. Enabling Secure Service Discovery with Attribute Based Encryption....72

1. Introduction 72
2. Profiles and Attributes....72
3. Applying Attribute Based Encryption....73
D. Algorithms for Decentralized Secure Service Discovery System....75
E. Private Key Management....76
1. Requesting Private Keys from an Online PKG....76
2. Private Key Generation: Online Vs Offline....77
3. Key Revocation....78
F. Use Case Scenarios....78
G. Security Evaluation....80
1. Proof of Security....80
2. Security Analysis....82
H. Experimental Results....83
I. Alternative Solutions....83
1. Group Encryption....83
2. Policy Based Cryptography....84
J. Conclusion....84
Chapter III. Securing Registry-Based Service Discovery....86
A. Introduction....86
B. Technical Background....86
1. XACML....86
2. X.509 Attribute Certificate....87
C. Service Discovery Policy....87
1. Concept....87
2. Choosing a Service Discovery Policy....88
D. Architecture for a Registry-Based Secure Service Discovery....89
E. Algorithm for a Secure Centralized Service Discovery....90
F. Secure Service Discovery Middleware....91
1. Related Work....91
2. Middleware Stack....92
G. Security Evaluation....93
H. Measurement Results....94
I. Conclusion....95
Chapter IV. Secure Service Discovery with Distributed Registries....97
A. Introduction....97
B. Related Work....97
C. Technical Background....98
1. Onion Routing....99
2. Distributed Hash Tables (DHT)....100
D. Requirements....101
E. A Scalable Distributed Registry-Based Model....101
1. Indexing and Data Retrieval....101
2. Algorithms for inter-registry Indexing and Data Retrieval....102
F. Securing the Access to Distributed Registries....103
1. Need for Anonymity....103
2. Pairing-Based Onion Routing....104
3. Anonymizing Publish / Request Messages for the Service Discovery....104
G. Architecture for a Secure Distributed Registry-Based Service Discovery....106
H. Security Evaluation....107
I. Performance and Results....108

1. Pairing-Based Onion Routing Costs 108
2. Kademlia Request/Response Costs 108
J. Conclusion 109
Chapter V. A Performance Analysis of Secure Service Discovery Solutions 111
A. Introduction 111
B. Related Work 111
1. Matching Strategies 111
2. Fault Tolerance and Crash Robustness 112
3. Publishing and Retrieval Time 112
C. Modeling Secure Service Discovery 112
1. Centralized Discovery 112
2. Decentralized Discovery 113
3. System Model Assumptions 114
D. Markovian Model 114
1. Markovian Centralized Model 114
2. Markovian decentralized Model 116
E. Matching Probabilities 117
F. Model Validation 118
1. Java Simulator 118
2. Rejection Rate 119
3. Server and Resource Usage Rate 120
G. Performance Analysis 122
1. System Setup 122
2. Rejection Rate 123
3. Average Number of Users in the System 124
4. Service Time Duration of a Request in the System 125
5. Summary 127
H. Evaluation of the Impact of DoS Attacks on System Performances 128
1. Introduction 128
2. Attack Model 128
3. Impact of a DoS Attack for a Protected and non Protected System 129
4. Summary 131
I. Conclusion 131
Chapter VI. Context Awareness in Service Discovery 133
A. Definition 133
1. Data Modeling 133
2. Reasoning 134
3. Quality of Context 134
B. Context Awareness and Security 135
1. Context-Aware Access Control 135
2. Privacy and Context Awareness 136
3. Context-Aware Encryption 136
C. Context-Aware Security Policy 136
1. Introduction to Security Policies 136
2. Security Policy and Context-Awareness 137
3. Related Work 137
4. Context-Aware Security Policy Requirements 139
D. Securing Contextual Information 140
1. Confidentiality of context information 141
2. Integrity of context information: 141

3. Trustworthiness of Delivered Context Information 141
E. Context-Aware Security Policy for the Service Discovery 142
1. Context Information Representation 142
2. Reasoning about Context Information 143
3. Health Care Scenario 144
4. Performance and Results 145
F. Conclusion 145
Chapter VII. Conclusion and Perspectives 148
Bibliography 151
Annex 159
1. API description 159
1.1. General Features: 159
1.2. Interface Definition 160
1.2.1. Parameters 160
1.2.2. Methods 160
2. UML specifications 161
2.1. External API 161
2.2. Communication related data structures 161
2.3. Policy handling 162
2.4. Protocol implementation 163
3. WSDL interface specification 166
4. Installation and usage guidelines 168
4.1. Installation 168
4.2. Usage 169

List of Figures

Figure 1 :B2C E-Commerce Sales in Europe 42
Figure 2 : Moore's Laws for Hardware evolution....43
Figure 3 : SOAP Envelop 46
Figure 4 Adressed scopes 48
Figure 5 : Salutation architecture 54
Figure 6 :UDDI Architecture 55
Figure 7 : WS-Discovery Message Sequence 57
Figure 8 : SDP Architecture 57
Figure 9 : Top Level of the OWL-S Ontology 59
Figure 10 : Client request disclosure 61
Figure 11 :Client request interception 61
Figure 12 :Client request modification or dropping 62
Figure 13 : Publishing services to fake registries....62
Figure 14 : Services deregistered by an unauthorized party 63
Figure 15 : Identity Based Cryptosystem 70
Figure 16 : Attribute Based Encryption Cryptosystem 71
Figure 17 : Probe Message....73
Figure 18 : ProbeMatch Message....74
Figure 19 : Encrypted Probe Message....74
Figure 20 : Clear Probe Message 75
Figure 21 : Encrypted Hello Message 75
Figure 22 : Request Algorithm....76
Figure 23 : Response Algorithm 76
Figure 24 : Private key request format 77
Figure 25 :Private key response format....77
Figure 26 : Discovering shopping service in insecure mode 79
Figure 27 : Discovering restricted e-mail services....79
Figure 28 : Discovering restricted movie service with privacy protection....79
Figure 29 : PDP and PEP interaction in XACML 87
Figure 30 : Communicating Discovery Policy....88
Figure 31 : Discovery Policy sample using XACML 89
Figure 32 : Secure Registry-Based Architecture....89
Figure 33 : Request Algorithm....91
Figure 34 : Middleware Stack....93
Figure 35 : Onion Message Format 99
Figure 36 : Onion Routing Virtual Circuit 100
Figure 37 :DHT indexation....101
Figure 38 : Algorithm for node insertion....102
Figure 39 : Algorithm for Key insertion....103
Figure 40 : Pairing-based onion routing circuit 104
Figure 41 : Publish message structure....105
Figure 42 :Anonymizing Publish Messages 105
Figure 43 : Anonymising Request Messages....106
Figure 44 : Architecture for a Secure Distributed Registry-Based Service Discovery....106
Figure 45 : Lookup Latency for a Keyword in KAD....108
Figure 46 : Centralized Model 112
Figure 47 : Decentralized Model....114

Figure 48 :Centralized Markov Chain Model 115
Figure 49 : Decentralized Markov Chain Model 116
Figure 50 : a – Encryption Markov chain; b - Decryption Markov chain 117
Figure 51 : Rejection rate in a centralized architecture 119
Figure 52 : Rejection rate in a decentralized architecture 120
Figure 53 : Resource usage comparison in a centralized architecture 121
Figure 54 : Resource usage comparison in a decentralized architecture 122
Figure 55 : Rejection rate curves for the four test scenarios 124
Figure 56 : Average number of users in the system for the four test scenarios 125
Figure 57 : Service time duration of a request in the system for the four tests scenarios 127
Figure 58 : Attack model for a Centralized Architecture 128
Figure 59: Attack Model for a Decentralized Architecture 129
Figure 60 : Rejection rate 130
Figure 61: Total sojourn time of a request in the system 130
Figure 62 : Request successful matching probabilities 131
Figure 63 : Security Infrastructure 138
Figure 64 : Organization of CoOL 143
Figure 65 : Reasoning Module Architecture 143

List of Tables

Table 1 : Service discovery protocols Summary.....57
Table 2 : Service discovery threat model 64
Table 3 : measurement values.....83
Table 4 : measurement values.....94
Table 5 : Pairing-based onion routing time measurements 108
Table 6: Values of the input variables used in the tests.....122
Table 7 : Performance summary 127
Table 8 : Measurement values.....145

List of Publications

International Conferences
2008

[1] Trabelsi, Slim; Urvoy-Keller, Guillaume; Roudier, Yves; “A Performance Based Approach To Selecting A Secure Service Discovery Architecture” in Proceedings of Mobile and Wireless Networks Security workshop - MWNS 2008, joint with 7th IFIP International Conference on Networking 2008.
[2] Trabelsi, Slim; Urvoy-Keller, Guillaume; Roudier, Yves; “A Markovian Performance Model for Secure Service Discovery Systems” submitted The 15th International Conference on Analytical and Stochastic Modeling Techniques and Applications

2007

[1] Trabelsi, Slim; Roudier, Yves; ”A Security Oriented Middleware for Service Discovery” Programme-Initiative Réseaux Autonomes et Spontanés, 30 et 31 October 2007, Paris
[2] Trabelsi, Slim;Roudier, Yves; “Secure service publishing with untrusted registries Securing service discovery”, SECRYPT 2007, International conference on Security and Cryptography, July 28-31, 2007, Barcelona, Spain
[3] Trabelsi, Slim;Roudier, Yves;Pazzaglia, Jean-Christophe, “Discovery: Threats and solutions”, SAR-SSI 2007, 2nd Conference on Security in Network Architectures and Information Systems, 12-15 June 2007, Annecy, France
[4] Trabelsi, Slim;Gomez, Laurent;Roudier, Yves, “Context-aware security policy for the service discovery”, SSNDS'07, 3rd IEEE International Symposium on Security in Networks and Distributed Systems, May 21-23, 2007, Niagara Falls, Canada

2006

[1] Trabelsi, Slim;Pazzaglia, Jean-Christophe;Roudier, Yves, “Secure Web service discovery: overcoming challenges of ubiquitous computing” ECOWS 2006, 4th IEEE European Conference on Web Services, 4-6 December, 2006, Zurich, Switzerland. This paper obtained the Best Paper Award
[2] Trabelsi, Slim;Pazzaglia, Jean-Christophe;Roudier, Yves, “Enabling secure discovery in a pervasive environment”, SPC 2006, 3rd International Conference on Security in Pervasive Computing, April 18 - 21, 2006, York, UK - also published in LNCS Volume 3934 , pp 18-31

2005

[1] Baynat, Bruno; Boussetta, Khaled; Eisenmann, Pierre; Trabelsi, Slim, “Extended Erlang-B law for performance evaluation of radio resources sharing in GSM/(E)GPRS networks”, PIMRC 2005, 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications September 11 - 14, 2005, Berlin, Germany

Research Reports

[1] Trabelsi, Slim;Urvoy-Keller, Guillaume;Roudier, Yves “A Markovian performance model for secure service discovery systems” Rapport de recherche RR-08-214
[2] Trabelsi, Slim; Roudier, Yves;Pazzaglia, Jean-Christophe, “Service discovery: reviewing threats and security architectures” Rapport de recherche RR-07-197
[3] Trabelsi, Slim; Roudier, Yves; “Enabling secure discovery with attribute based encryption” Rapport de recherche RR-06-164

Book Chapters

[1] Pazzaglia, Jean-Christophe;Wrona, Konrad;Laube, Annett;Montagut, Frédéric;Gomez, Laurent;Roudier, Yves;Trabelsi, Slim; “Utilisation des informations contextuelles pour assurer la sécurité d'un processus collaboratif distribué : un exemple dans l'e-Santé”, Chapter 13 in "Informatique Diffuse", Observatoire Francais des Techniques Avancees, Collection Arago 31, Paris, Mai 2007, ISBN: 2-906028-17-7 , pp 345-370

Introduction

In the second half of the 20th century the computing devices were used to help human beings to process an incredible amount of information during short time periods. Computers were also used to automate industrial work, like automatically driven factory machines in heavy industries. The architectural design was primitive and restricted to a centralized deployment. Single machines and centralized processing centres were in charge of centralized data and information management and a single interface was offered to be accessed by high qualified users in order to administer the system.
With the emergence of personal micro-computers and the World Wide Web technology, the human / computer or computer / computer interaction has radically changed. Information access has become decentralized (we can access a huge wealth of knowledge from any computing machine); data processing is no longer restricted to big companies and industry; interfaces with machines becomes accessible for all users without any qualification; computers are not used only for work, but also for entertainment (Multimedia content, games, shopping …). "Information technology penetrates and changes job descriptions, life styles, and business relations" [HAN03]. The notion of a virtual world is no more limited to Science Fiction books and movies, but has turned into a reality. Men can have a virtual identity used in social network services, to apply for a job, or to make a business deal. Their personal web page or Blog becomes part of their personality. They can have a parallel social life via their computer, when they meet other people via chat-rooms, forums, web messengers. They can sell, buy, contact, create, vote, communicate … or live a double-life like in Second Life[1].
E-Business takes advantage of this new life style, improving efficiency and innovation for the market rules. A home can easily be changed into a shopping market, clients can buy from their office, and employees can remotely access their office desktop without leaving their homes. The evolution of this new business is extremely fast (about 25 % growth rate per year) as shown in a recent study led by eMarketer about the evolution of the B2C E-Business in Europe between 2006 and 2007 and predicting that the market will treble in size until 2011 (see Figure 1).

[1] http://secondlife.com/



Figure 1 :B2C E-Commerce Sales in Europe[2]

Besides the technological evolution, the decentralized and distributed architecture design played an important role in this revolution. The fact that various computing elements were deployed all over the world and could collaborate with each other, expanding space boundaries, raised the concept of tightly-coupled shared memory and multiprocessors providing means to share resources, data, and information, as is the case of printers, scanners, services. Peer to Peer architecture introduced a new concept of file sharing in which every user can exchange data with other users without any intermediation. This concept is also used for communication, telecommunications and distributed storage.

A. Pervasive and Ubiquitous Computing

According to Weiser "*Not only do books, magazines and newspapers convey written information, but so do street signs, billboards, shop signs and even graffiti. Candy wrappers are covered in writing. The constant background presence of these products of "literacy technology" does not require active attention, but the information to be conveyed is ready for use at a glance.*" [WEI91]. In other words, all the environmental elements can be assimilated to readable information and can be processed by a computing entity. The computer becomes omnipresent and plays an important part of our everyday life, becoming accessible to everyone, everywhere, at any time. The main idea of the pervasive computing is that the environment surrounding the user is a huge computing system composed of millions of small devices working together. Pervasive computing has four important characteristics:

- **Decentralization**: Huge computing systems are fragmented into a multitude of small devices processing the same data. Each of these devices is assigned a specific task and the end-result of these tasks is aggregated into a consistant result exploitable by the users or others machines. Decentralization improves the performance usage of the resources (load balancing, charge distribution, delegations …), availability and fault resistance (fault tolerance, recovering and replacing broken elements, etc.), the flexibility while accessing resources (interfaces are available everywhere).
- **Diversity**: The hardware evolution follows Moore's law formulations; the number of transistors integrated per circuit is rising, the cost per transistor is decreasing, computer consumption is decreasing, disk storage content is encasing for a decreasing volume (Figure 2)

[2] http://www.emarketer.com/

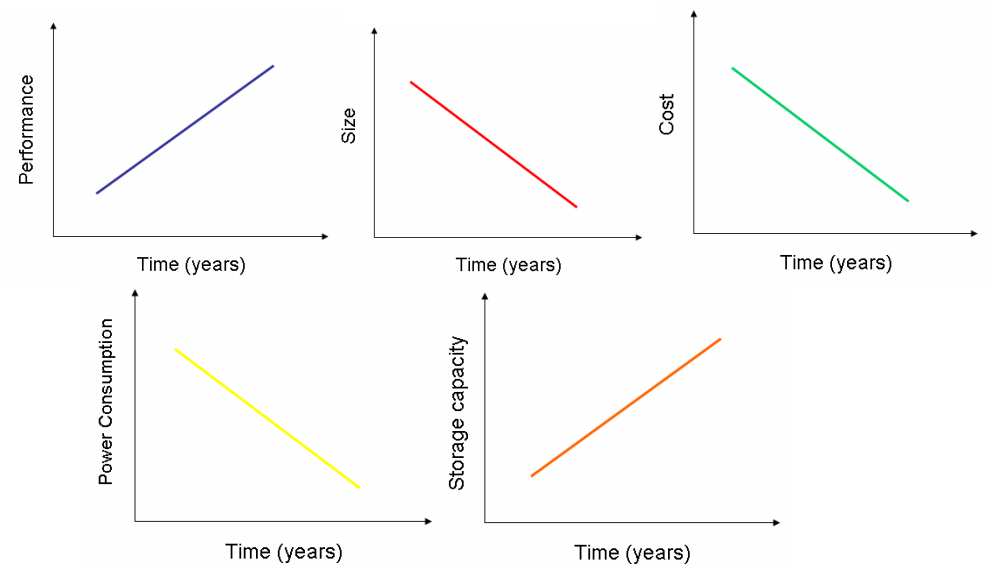


Figure 2 : Moore's Laws for Hardware evolution

This evolution makes computing systems small and cheap enough to be embedded, distributed, and deployed everywhere in space, in a huge quantity. Owning such equipments becomes accessible to all people independently of their budget (a personal PC is sold less than $200). For these reasons, users have a large choice to use pervasive devices in any circumstances. For example, a photographer can store his digital pictures in the local disk of his photograph, in a smart card, in a SD or XD disk, in a USB key, in his phone memory, in a CD, in a computer, in a remote storage container via the network, etc. Recent research is working on a collaborative P2P storage solution, whereby the user can store fragments of his data into his neighborhood disks. Access to the same data can be done via different equipment (PDA, Smart-Phones, laptops …) depending on the context of the user (home, office, travel, car …).

- **Mobility**: Wireless technology provided new network architectures with various access means: Infrared, Wifi, Bluetooth, GSM, GPRS, UMTS … The user can be permanently connected to a network (infrastructure of Ad-Hoc) via these different access media; he can move without interrupting the execution of an online service. The handover function facilitates a permanent connection while ensuring a transition between different wireless access media without interrupting the traffic. By means of a hybrid UMTS/Wifi card, the connection is automatically switched between these two networks depending on the connectivity (or QoS). Location techniques are also developed in order to retrieve equipment and services during the exploration of unknown environments.
- **Flexibility**: Ubiquitous applications are permanently communicating, though, initially, applications were developed under various technologies using different languages and running over different platforms an OS. In pervasive systems, applications are heterogeneous and completely independent; this is why developers provided tools and protocols to insure these interactions via user interfaces (user / machine) and middleware (machine / machine). The interface is a human understandable portal used by the user to interact with the application. The middleware is a mediation layer between different software components, which provides interoperability between those components. These interfaces are useful when an entity (human or software) wants to interact with a new unknown pervasive environment (set of applications) without any prior knowledge of the technology deployed and without installing

locally a foreign code imported from this environment. It is also useful to support the software evolution; when an application is updated or changed, other components must adapt themselves automatically to this change.

B. Distributed Systems

Tanenbaum [TAN02] defines the distributed systems as "*A collection of independent computers that appears to its users as a single coherent system*". In distributed systems, machines are usually autonomous though interconnected. Users can access these hosts via a single interface in order to interact with these machines. Users do not have any information about the machines with which they are interacting. The group of distributed systems is perceived as a black box by the human user. The main objectives of design in distributed systems are:

- **Connecting resources:** interconnection between machines and users via a network, and providing protocols to facilitate communication between different actors of the system.
- **Transparency:** as mentioned before, the system should be perceived as a black box by humans. Many parameters are hidden from the user, like data location and access means, the machine location, prospective replications, failure treatments, load balancing, concurrency with other users.
- **Scalability:** To avoid bottleneck problems and provide a solution capable to serve a huge number of requests at the same time, distributed systems can share the amount of tasks in order to balance the load of the system and permits a scalable solution.

C. Service Oriented Architecture (SOA)

The Service Oriented Architecture is one of the concrete technical deployment models of the distributed system paradigm combined with the object oriented programming concept, in the sense that entities are assimilated to remotely accessible distributed objects. The machines and applications in the distributed systems are subsumed by the notion of services (software agent abstraction). SOA was originally (in particular as envisioned in Jini [JIN]) intended to enable access to applications running on nearby devices in a dynamic fashion. This programming style promotes the use of loosely coupled and highly interoperable applications to overstep the limitations of traditional distributed component solutions (like CORBA [COR]). A Service, the building block of SOA solutions, encapsulates a set of related business functions within a container and gives access to these functions through standardized interfaces. SOA introduces a loosely coupled interaction model which serves as the basis to define protocols and procedures that enable an efficient interconnection between different applicative systems or software components. It consists mainly of services, which are applicative elements providing elaborate functions (database access, data processing, business logic…), and of clients that are requesting such services. These two types of players only rely on a standardized interface to communicate but do not necessarily share the same implementation platforms (programming language or OS). IBM's definition of services reads as follows: "*A service in SOA is an application function packaged as a reusable component for use in a business process. It either provides information or facilitates a change to business data from one valid and consistent state to another*"[3]. In this vision, the service is deployed to provide business functions that could be used by some application in order to support a business process.

[3] http://www.ibm.com/developerworks/library/ws-soaintro.html

The Client / Server relationship is central in SOA, where all the requesters (human or machines) are considered as clients, and all information providers are considered as servers offering services. Different interfaces are used to communicate, collaborate or to be composed independently of the technology used to develop services. In SOA, one usually distinguishes four entities:

- Service: software entity accessible via published interfaces
- Service provider (Server): entity that implements and deploys one or more services.
- Service requestor (Client): entity that needs to access a service (it could be a human user, or an application, or another service)
- Service broker (Registry, Repository): an entity playing the role of yellow pages in which servers can publish their services and clients can look for services. Services are published in order to be visible and accessible and Clients can use specific methods to retrieve these services.

D. Web Services

Currently, the SOA paradigm is largely promoted by the spread of Web Service technology. This technology proposes a neutral language and environmental programming model that provides flexible and adaptable interfaces. These interfaces describe a series of methods used to access services via an XML based descriptions and messages. Web Services overstep the limitations of traditional distributed component solutions (e.g., Jini [JIN], CORBA [COR]…) in that they increase distributed software dynamicity and flexibility thanks to the use of XML-based interfaces. Every computing entity able to process XML can generate the appropriate method stubs in order to access services. The XML language used to define the interface describing a Web Service is called Web Services Description Language (WSDL) [WSDL]. This language defines the interaction with the service, where it is located, its capabilities are (what the service can do), how to invoke it and input and output parameters. Usually, WSDL files describing services are published in repositories in order to be reached by prospective users.

The message exchange layer in Web Services technology is also based on an XML envelope called SOAP. SOAP is a simple and extensible protocols that allows applications to exchange XML coded messages over HTTP transport layer. Compared to other messages formats, SOAP provides a simple structure to incorporate useful information and meta-data that can be processed by applications. The SOAP envelope of a message has two parts (Figure 3):

- **SOAP Header**: It is an optional element in which management or security information is specified. It provides extra information about the application or about eventual preferences related to the message. For example it could contain encryption or authentication elements used to protect messages.
- **SOAP Body**: it is a mandatory element (payload) containing essential endpoint information about the message exchange. It contains information about namespaces, or a description of the remote methods used to access services.

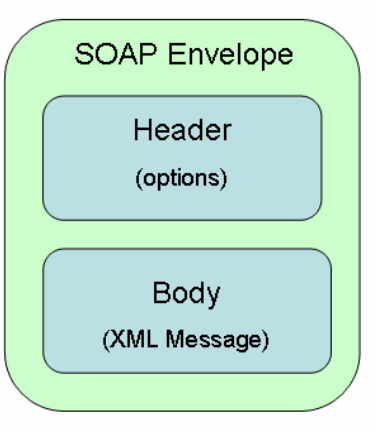


Figure 3 : SOAP Envelop

E. Peer to Peer Systems

The principle of Peer to Peer (P2P) systems is the data exchange between users without any mediation or any centralised entity to handle the data exchange. P2P architecture is actually one of the most scalable architectures developed in past years. This system relies on many distributed systems concepts like load balancing, redundancy, Distributed Hash Tables (DHT) for routing, overlay networks. Today, P2P networks are composed of millions of users (or hosts) called nodes, scattered all over the world and linked by the Internet. Users are supposed to permanently collaborate by sharing their resources and collaborating without any interest (actually protocols are tending to favour the user that is collaborating). Nodes are not necessary reliable, and there is not impact on the system if one of these nodes fails or disconnect. There are two main techniques to retrieve data: the first consists in sending requests to a random numbers of neighbour nodes; these nodes propagate the request with the same technique until obtaining a reference to the node keeping the data. It is called unstructured P2P system used notably in Gnutella [GNU]. The second technique, called structured technique, is based on hash tables maintained locally in every node. Data and nodes are identified by hash values that give an indication about the location of the peers holding the data. These hash values are stored in the hash tables. This system is called *Distributed Hash Tables (DHT) that is a look-up algorithm in a structured P2P system that uniquely maps a key k to a single peer p, and defines an overlay network substrate (X;U) through which queries for k are routed towards p.*[GAR04].

F. Workflow Architecture

Pervasive applications in some cases could be assimilated to participants in a business process collaborating according to specific business rules, defined to regulate the steps of a transactional process. Software entities called workflows are in charge of this regulation. The workflow is the automated version of a collaborative business process, where written procedures, tasks and interaction rules become managed and controlled by a computer. The workflow management coalition defines the automated workflow as "*Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal*"

G. Security requirement in Pervasive Systems

In pervasive computing, services are more open, accessible, distributed, and close to the user. This proximity introduces new threats and vulnerabilities for the systems. Before the emergence of ubiquitous systems, services were protected as citadels isolated from the foreign populations and accessible via small bridges secured by guards. Now, systems and services are assimilated to supermarkets open for everybody, in which merchandise is directly accessible to clients in a self-service way. Of course, such displayed goods are vulnerable and can be easily stolen, but new sophisticated guard systems appeared to protect the merchandise.

At present, the Internet is a market place with billions of costumers and e-business companies generating and exchanging trillions of dollars all over the world. It is evident that old security techniques, like access control lists or static firewalls, are no more sufficient to protect this business; this is why new techniques were developed to adapt the security requirements to the environment, to the application, to content of the exchanged data, etc …

These techniques usually concern fundamental security principles, like:

- **Authentication:** used to identify and authenticate an entity in order to verify its source or to estimate the trust level that it can be granted. The entity that would be authenticated has to provide a verifiable proof of its identity, like a password, a certified token or personal biometric information.
- **Confidentiality:** it concerns the protection of resources or information from unauthorized access. The main objective is to keep sensitive information secret. These information must be known only by a restricted number of users that share some specificities. For example, military secret documents must be accessible only to army officers of the same country. Confidentiality is not limited to data access protection, but it may also aim at hiding the very existence of the data. In fact, knowing that a data exists could be considered as a threat to the data itself and to the system. One well known mechanism used to preserve confidentiality is cryptography, which scrambles data and makes it unreadable or invisible for unauthorised entities.
- **Integrity:** integrity refers to trustworthiness regarding data or resources. It is usually phrased in terms of preventing improper or unauthorized change [BIS03]. Checking the integrity of a data is verifying if the information contained has not changed or been corrupted, and also verifying what the source of this data is. Integrity is used to prevent and detect any modification introduced on the content of a data.
- **Availability:** it is usually related to the reliability of the system, in the sense that a desired resource must remain available for use by providing a mechanism to keep the system running normally in any case. The attack that corrupts availability is called a Denial of Service (DoS) attack.
- **Privacy:** Westin defines privacy as "*the claim of individuals, groups and institutions to determine for themselves, when, how and to what extent information about them is communicated to the others*" [WES87]. During a message exchange, users and computing entities can exchange private data. The data must be protected against undue interference, illegal use and diffusion of such information. Some techniques were developed to safeguard personal privacy and limit the publication of identifiable data [FIS01] by providing anonymity, pseudonymity, unlinkability, unobservability, confidentiality, integrity and availability.
- **Access Control:** it is a restriction to access or performance of an action on some resources (objects) by the requester (subject) or the group to which it belongs. The subject has a collection of rights or authorizations that must be checked before access is granted to the requested resource. This restriction is applied on actions (read, write, modification, delete…) that can be performed by the subject on the object.

H. Contributions

In this thesis report we study orchestration mechanisms that enable the interconnection and the cooperation between pervasive systems described above. We particularly focus on the discovery and location mechanisms that are essential for any entity to join a pervasive system and exploit its functionalities in a correct manner. We first analyze the security problems and the potential threats related to these mechanisms that usually follow the security requirements related to pervasive systems in general. This security analysis provides a list of threats and the possible attacks that can be built against the data and resources of discovery mechanisms in pervasive systems.

Existing security solutions were proposed in the literature to overcome some of these security lacks but none of them take into account the new requirements imposed by pervasive environments in which users are mobile and application are flexible and dynamic. Privacy issues and user protection was also neglected in these studies.

For these reasons we propose in this thesis different security solutions, adapted to different network configurations and architectures, relying on contextual information related to the ambient environment then we proposed a performance evaluation of the overhead generated by the security techniques used in these solutions. Figure 4 illustrates the scopes of problems addressed in this thesis:

- **Security**: different security mechanisms like cryptography, security policy, anonymizing techniques are used and adapted to the discovery environment. These security solutions are deployed for centralized and decentralized architectures in local and global networks.
- **Context Awareness**: contextual information is added to the discovery mechanisms in order to add precision and flexibility in the security solutions.
- **Performance**: the impact and the costs related the security techniques used in our solutions are studied in order to evaluate the feasibility of a real deployment of such secure solutions. A dimensioning mathematical tool is proposed to study all the performance parameters of secure service discovery solutions.

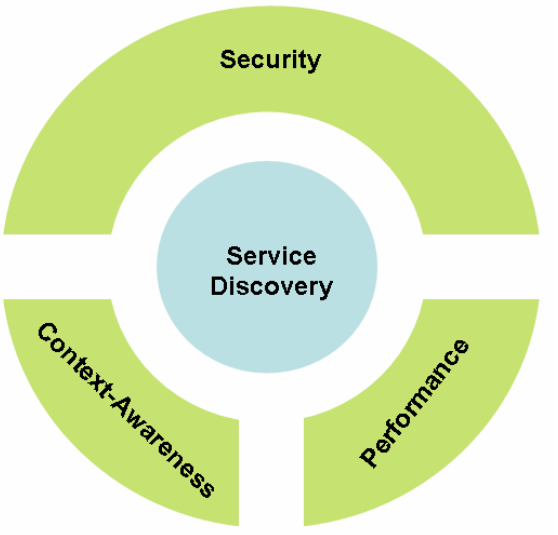


Figure 4 Adressed scopes

I. Outline

In the **first chapter** of this thesis we introduce and define the notion of service discovery technique in terms of design, protocol, and deployment. Then we present a threat model analysis describing lacks and possible attack observed on most of the existing protocols. From this threat model we derived the security requirements of service discovery mechanism.
The second part of the thesis covers three chapters, each one presenting a security solution proposed to protect and secure service discovery in pervasive systems; in the **second chapter** of the thesis we present a secure solution dedicated to decentralized architectures and using an attribute based encryption scheme to restrict the access to discovery messages. In the **third chapter** we describe another solution dedicated to centralized architectures that makes use of a trusted registry in charge of enforcing discovery policies. And the **fourth chapter** we propose a hybrid solution dedicated to large-scale deployments and relying on a peer to peer indexing system accessible through an anonymizing routing system.
The third part of this thesis is dedicated to the performance analysis of secure service discovery systems. For this performance study, detailed in **chapter 5**, we proposed a new mathematical performance model to evaluate the robustness, availability, efficiency, of our service discovery system during its normal execution and under a denial of service attack.
The last part of this document is dedicated to context awareness. In the **chapter 6** we demonstrate hwo contextual information adds more accuracy and flexibility in security mechanisms dedicated to service discovery.

Chapter I. Service Discovery

A. Introduction

The Service Oriented Architecture (SOA) paradigm was initially developed in the Jini [JIN] framework to address the specific requirements of pervasive computing software. In particular, SOA was originally intended to enable access to applications running on nearby devices in a dynamic fashion. This programming style promotes the use of loosely coupled and highly interoperable applications to overstep the limitations of traditional distributed component solutions (CORBA [COR], DCOM [DCO]). A Service, the building block of SOA solutions, is intended to encapsulate a set of related business functions within a container and enable access to these functions through standardized interfaces. In pervasive computing, context-awareness would typically be enabled through the access to a set of such services.
Orchestration techniques were developed in order to deploy a set of basic services into a more complex service. Even though static orchestration is frequently used, for example in Web Service based architectures, we contend that the quintessence of the SOA style, especially when used for pervasive computing software, lies in the dynamic composition of services. Such a dynamic composition obviously comes at a cost: being able to locate previously unknown services becomes mandatory.
Discovery therefore becomes of strategic importance in the SOA stack and this importance is growing proportionally with the dynamic aspect of the environment: while a typical intranet implementation may rely on a basic discovery strategy (e.g. naming service in CORBA) or may even not strictly require it (e.g. predefined set of known services), Internet-wide and, above all, pervasive applications face a set of challenges with respect to discovery. In such applications, the discovery strategy should cope with the heterogeneity of services and platforms from a technical perspective (e.g. take into account bandwidth, energy savings …), with the complex semantics of service descriptions (e.g. resorting to terminology- or ontology-based descriptions), with the scalability of the solution, and with the requirements for security and trust regarding the services discovered.

B. Definition

Communication devices in fixed networks like local LANs are at best traditionally assigned a static network configuration, at worst use DHCP to dynamically configure their IP address.

The DNS protocol is quite sufficient to find a host in such networks using its IP address or its domain name. With the emergence of new dynamic networks and services where devices are pervasive, the discovery techniques are being adapted in order to find mobile services rather than devices. Architectures in which deployed services are no more static and no more relying on a fixed topology. Wireless technology introduced a new dynamicity parameter for the service location; mobile services retrieval is no more relying on IP addresses routing tables (like in Bluetooth), but on other parameters and identifiers. This adaptation technique called service discovery addresses how to combine services as a logical layer in such systems, together with the specification of environmental constraints.
The main players of the discovery phase are: the service requester (client), which can be a human user or software and the service provider (server), which represents the entity providing one or multiple services that can be accessed by the clients. Two main discovery architectures can be adopted according to the network topology and the capacities of the computing devices:

- ***Centralized Discovery (registry-based)***: Centralized discovery approaches rely on a registry which plays the role of yellow pages, and which clients can refer to. The registry (or repository) is a database containing descriptions and references to some available services. Servers publish their services to a registry, while clients discover published services by requesting a registry. A service advertises its capabilities (a set of attributes describing the service) to the registry, which will store them for a certain amount of time. A client contacts the registry to find a service by sending a request containing service preferences, which the registry tries to match with the most suitable provider found from the stored advertisements. In that approach, registries have to be considered by the services and the clients as a third trusted party.
- ***Decentralized Discovery***: Limiting service discovery to registry supported architecture that many standards SOA based services have adopted in their implementations is reductive in terms of network architecture and equipments (e.g. need to deploy specific equipments like registries). An alternative approach to service discovery exists that relies on peer-to-peer advertisements between services and clients (point-to-point and point-to-multipoint). In such an approach, clients discover services by broadcasting their requests to their neighbourhood, and if one of the neighbours features the requested service, it will directly respond; the neighbour may otherwise forward a request to its own neighbourhood. This mechanism is used for instance by the P2P-based Web Service Discovery system (PWSD) [GAR04-2], which relies on the Chord P2P protocol to perform the service discovery over the internet.

C. Service Discovery Components Design

Service discovery is not limited to a matching procedure but involves a multitude of components that interact together to provide a coherent and efficient discovery service. These components can be deployed according to various design approaches depending on the environment and the technologies deployed by the system administrator. Zhu et al [ZHU05] provide a classification of these different components designs:

- ***Service and attribute naming***: the description contained in the service profiles published by the servers must specify a service name used semantically to characterize the published service (like printer, mail, bank, library etc …). Clients can use these names (also represented as a set of attributes) in their discovery queries in order to reach a particular service. These service attribute names can be expressed according to an infinite number of structures without limiting boundaries or particular description

logic; for this reason most of the discovery protocols propose to use template structures to define a naming format. Some other protocols offer predefined descriptions (Universal Unique Identifiers UUID) for some popular services. These solutions remain static and limited to well known identifiers and not really adapted to pervasive commuting systems.

- ***Communication method and diffusion protocols***: three communication methods can be used to exchange discovery messages: the unicast based method is easy to use and to manage, but limited to point to point between client – registry, client – service, or service – registry communication. Such a communication method requires a prior knowledge about the network addresses of the registries (fixed manually). The second method is Multicast-based and is used to avoid the manual configurations of unicast addresses by specifying a single multicast address to initiate the discovery (by multicasting publish or request messages) before switch to unicast. The last communication method is broadcast-based and has the same goal as multicast that frequently used in wireless ad-hoc networks. Compared with the multicast-based method, the broadcast aims of overloading the transmission links due to the large amount of packets used to disseminate the information.
- ***Query and registration methods***: in the announcement-based approach, clients and registries listen to a channel (multicast most of the time) and catch the announcements sent by the servers publishing their services. In this case, clients and registries can cache published information in order to access it faster when needed. In the query-based approach, clients (sometimes registries) directly query the available servers (and the registries holding information about services) without the need for any announcement.
- ***Service State***: published services have a maximal duration time while the services are not expired called lifetime. This information allows clients and registries to remove information about expired services. In this condition the discovery system has a soft-state. Sometimes, registries contact the concerned services to renew their announcement. Some other configurations, called hard-state, do not take into account lifetime and keep service information until the concerned sever unpublishes its services.
- ***Service selection***: after requesting services, a client receives a list of matched services; in this case the client is able to manually select his preferred services (more adapted for human users). If the client is automated, the selection mode can be employed to mechanically select the service verifying some specific criteria.
- ***Service invocation***: after selecting a service, clients can access it by remotely invoking the functions provided by the server. This invocation step involves a service network address, a communication mechanism and a specific application method. The network addressing element is responsible for locating the service end point in a network using an address resolution mechanism. The communication element is responsible for defining which communication mechanism is suitable for the service invocation, like RPC, SOAP/HTTP, TCP, UDP. Finally the invocation methods that are described via the interfaces provided by the services and described using description files like WSDL files. These description files are interpreted by the client in order to generate locally the appropriate methods used to interact with the service.

D. Service Discovery Protocols

The research community and industry elaborated numerous service discovery solutions and protocols initially dedicated to a particular context (frameworks, application, platform, OS,

network topology, programming language etc.), then adapted to generic applications and deployable for any context.
In this section, most of the well known discovery standards and protocols are described and compared.

1. Salutation

The Salutation discovery architecture is the product of an open source industrial consortium called Salutation Consortium (dissolved since June 2005). This solution provides a complete architecture (see Figure 5) dedicated to service discovery including standardized methods for applications and some particular entities called *Salutation Managers* (SLM). The SLM plays the role of a service broker in which servers could register their services capabilities and clients can ask for a service using a platform independent interface called SLM-API. One SLM is usually affected to a single network entity (LAN, WLAN, or sub-domain) but they are also reachable from other networks through a *Remote Procedure Call* (RPC). In order to harmonize the transport layers between different network entities, a Transport Manager ™ can be coupled with SLM to form a SLM-TI that offers the brokering capabilities of an SLM and the transport adaptation function of a TI. The responsibility of the SLM is not limited to service discovery but it also intermediates messaging by crating virtual pipes between client and services using the transport manager (TM). This functionality is called *Service Session*.

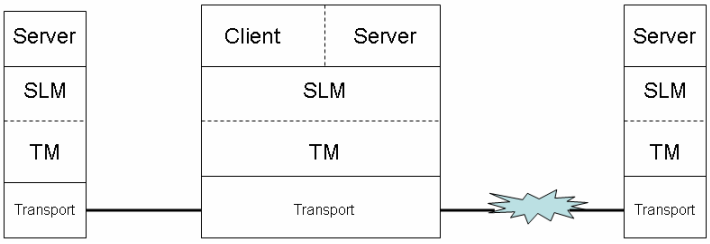


Figure 5 : Salutation architecture

2. Service Location Protocol (SLP)

Developed by an IETF working group, SLP provides a freely available framework allowing network application to discover networked services in enterprise TCP/IP networks. It provides a dynamic configuration mechanism for applications. Applications are related to clients looking for servers available in the local enterprise network. Centralized repositories are used for service publishing and client request matching.
The SLP architecture has three main components:

- The User Agent (UA) representing the clients that are looking for a service
- The Service agent (SA) that advertise the location and the characteristics of a service (service Type)
- The Directory Agent (DA) representing the centralized repository in charge of collecting information provided by the SA, caching service location and attribute information, and matching Service Type with the requests coming form the UA (*SrvRqst*).

UAs are able to directly send their request to SAs by multicasting a *SrvRqst* to the available SAs. If an SA is concerned by this request, it replies by a response message (*SrvRply*) to the UA via unicast. In order to use the DA to retrieve available SAs, the UA must discover the existence of the DA using one of the three DA discovery methods (static, active, and passive).

3. Jini Lookup Service (JLS)

In contrast with the previous discovery architectures, Jini defines a Java-based discovery architecture with a programming model exclusively exploitable through Java technology. Each Jini device is assumed to embed a Java Virtual Machine (JVM) running on it. In order to facilitate the access to the services, Jini relies on the Java *Remote Method Invocation* (RMI) in order to export device drivers to client applications.
The architecture is based on a central component called the *Jini Lookup Service* (JLS) that plays the role of a registry in which service providers can register their services by sending a *join* message. A timeout period called *lease* is applied to service registrations in order to eliminate the unavailable services that are still in the JLS. Clients can directly query the JLS about the available services. After selecting a service, the client is able to access it through an additional piece of code (driver) sent by the service provider during the registration step and retrieved from the JLS.

4. UDDI

The Universal Description Discovery and Interrogation (UDDI) specification is a pure registry based service discovery mechanism (see Figure 6) that provides elaborated classification and representation of metadata related to web services profiles. Conceptually, a service provider can publish three types of information into a UDDI registry:

- **White pages:** General contact information about a company providing the service, including business name, address, contact information, and unique identifiers.
- **Yellow pages:** Information describing a service using specific taxonomies and categorisation. This information allows others to discover your web service based upon its categorization (such as in the manufacturing or car sales business).
- **Green pages:** Technical information that describes the behaviour of the service and how to access it. This includes pointers to the grouping information of web services and where the web services are located.

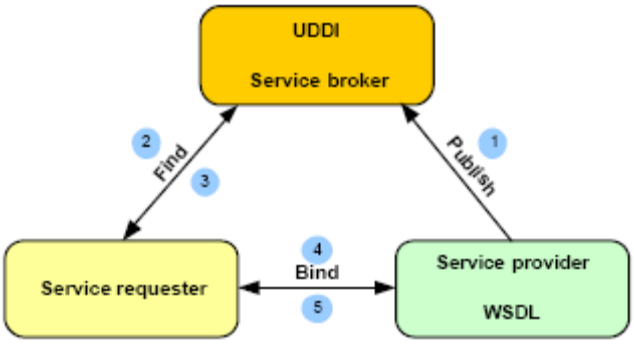


Figure 6 :UDDI Architecture [TSA03]

Version 3 of UDDI provides a new architecture design for the registries consisting in a classification depending on the sensitivity of the registry. We distinguish in this new architecture three categories of registries: the public, the private, and the shared repositories. In order to restrict the usage of the private and the shared registries, UDDI version 3 offers the possibility to use digital signatures. These additional functionalities are integrated to prevent fake service registration. Finally, only Web Services with signed digital tokens will be selected.

5. UPnP

Universal Plug and Play is a set of protocols issued by an industrial consortium led by Microsoft aiming at the interconnection, auto-configuration, advertisements, and discovery between clients, services, and devices belonging to the same network domain. Device and service profiles are represented in an XML format that facilitates the auto-configuration of the new devices joining the network. Unlike Jini and SLP, UPnP does not support registries; communication between devices and clients is always direct. This restricts UPnP deployement to small environments like home network systems.
UPnP offers a set of complementary functionalities: the first one is the addressing support via the Auto-IP protocol that automatically assigns IP addresses to the new entities joining the network. Its second functionality is the discovery by providing a basic protocol used to inquire about some information related to the requested devices. After getting preliminary information containing a URL (pointing to a XML-based description file), the third functionality will resolve this URL in order to download the XML file. The XML description file is then interpreted in order to exploit the control functions in charge of the interactions with the requested service. The control interaction will be transmitted via HTTP and wrapped using SOAP. The last functionality offered by UPnP and supported by GENA (General Event Notification Architecture) is related to the event notification used to send notification about the state variable changes of the system.

6. WS-Discovery

Web Services Dynamic Discovery (WS-Discovery) is a technical specification that defines a multicast discovery protocol to locate services connected to a network. Each service provider announces itself (by sending a ”Hello” message) through the multicast group to display the services that it can provide. Each user looking for a service propagates its query (by sending a ”Probe” message) through the multicast to which only concerned service must make a unicast response (by sending a ”Probe Match” message). The default matching attributes are the Type and the Scope of the service. Obviously, other attributes and metadata information can also be added.
Because the WS-Discovery protocol is based on multicast, the discovery scope may be restricted to local subnets. For this reason and in order to scale to a large number of endpoints, the specification defines multicast suppression behaviour if a discovery proxy (DP) is available on the network. By listening announcements, clients detect discovery proxies and switch to use a discovery proxy-based protocol. However, if a discovery proxy is unresponsive, clients revert to the decentralized protocol. These discovery proxies can communicate with each other in order to extend the discovery scope to the others subnets. This feature enables smooth migration from carefully managed to ad hoc networks. The WS-Discovery specification does not suggests securing the discovery process but it recommends the usage of a compact signature format to secure the exchanged messages. In this case, each entity has the possibility to verify the signature of the message sender. This signature protects from message modifications, replay, spoofing, etc. Signature verification is obviously insufficient to protect users (servers and clients) since a valid signature only assesses that the message content has not been altered without presuming of the level of trust of the issuer. Moreover, the content of the message is not confidential and there is no guarantee against the disclosure of private information. For example, a malicious server can publish fake services with a valid signature or listen to request messages in order to collect valuable information.

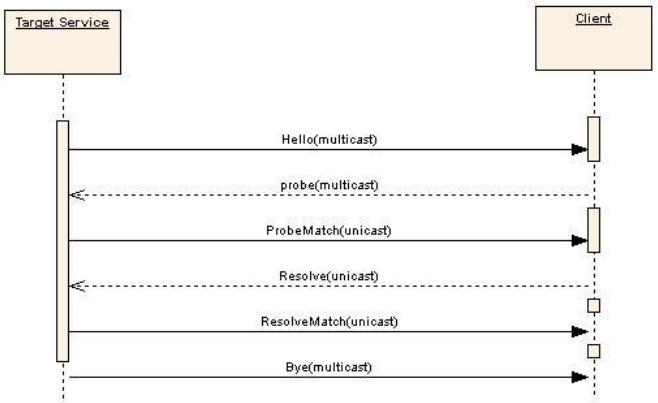


Figure 7 : WS-Discovery Message Sequence

7. Service Discovery Protocol (SDP): Bluetooth

SDP is a simple decentralized discovery protocol with minimal requirements for the transport layer (in case of application with Bluetooth, L2CAP is used as transport protocol). The request/response messages are called protocol data units (PDU) and are directly exchanged between clients and servers without the help of registries (see Figure 8). Each server maintains a service attribute record containing the following information: *ServiceRecordHandle*, *ServiceClassIDList*, *ServiceRecordState*, *ServiceID*, *ProtocolDescriptionList*, *BrowseGroupList*, *LanguageBaseAttributeIDList*, *ServiceInfoTimeToLive*, *ServiceAvaliability*, *BluetoothProfileDescriptorList*, *DocumentationURL*, *ClientExecutibleURL*, *IconURL*, *ServiceName*, *ServiceDescription*, *ProviderName*. These attributes are matched with the Client's request in order to send back a PDU response message containing the unique identifier of the requested services (UUDI) or an *SDP_ErrorResponse* in case of wrong matching. A UUID is a universally unique identifier that can be used by a client to retrieve and access the corresponding service.

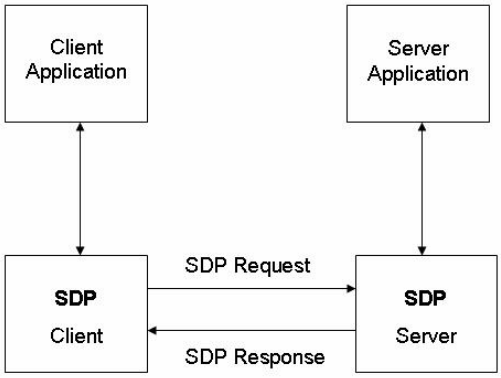


Figure 8 : SDP Architecture [GRA00]

	Type		Communication method		
	Centralized	**Decentralized**	**Unicast**	**Multicast**	**Broadcast**
SLP	Yes	Yes	No	Yes	No
UPnP	No	Yes	No	Yes	No
UDDI	Yes	No	Yes	No	No
Jini	Yes	Yes	Yes	Yes	No
SDP Bluetooth	No	Yes	No	No	Yes
WS-Discovery	Yes	Yes	Yes	Yes	No
Salutation	Yes	No	Yes	No	No

Table 1 : Service discovery protocols Summary

8. Service Discovery in Ad-Hoc Networks

The service discovery standards described above are not really adapted to Mobile Ad-hoc NETworks (MANET) because they systematically rely on a stable and fixed infrastructure that permits a traditional routing of discovery messages based on fixed ip addresses and quasi static routing tables. In Ad-hoc networks, users and nodes are systematically moving and the routing paths are permanently changing. For this reason service discovery is not limited to a simple match making but it concerns the service location. Service discovery in MANETs raises two crucial questions: does the service exists and how can we locate it?

[SAI05] propose a MANET architecture in which nodes are deployed around a directory backbone covering a neighborhood located within a fixed number of hops according to population density of the system. In order to simplify the indexation and retrieval in the directories Bloom filters [BLO70] and membership tests are used to locate the directories holding requested service descriptions. Compared to other existing MANET discovery protocols that rely on broadcast flooding this solution be more scalable and tends to minimize the traffic through the network with the restriction to selected trusted nodes. These play the role of directories that is not evident for a pure Ad-hoc network.

[YAN06] tries to enhance the efficiency of the post-query discovery protocol (relying on a multicast query and a unicast reply) by introducing the multicast reply with a response caching. All the nodes are supposed to cache the responses of a multicast request in order to decrease the total number of requests and also to decrease the complexity of the messages from $O(N^2)$ to $O(N)$. In order to ensure a reliable multicast routing, edge nodes are selected to forward messages to all neighbors. This solution makes two strong assumptions that limit the efficiency: first the lifetime of a service description is assumed to be long enough to be cached and reused. The second, the number of nodes must be limited to avoid overhead of the traffic because of systematic multicasting.

E. Matching and Semantics

1. Matching

Matching plays an essential role in the service discovery application in order to map requested elements (contained in a client query) and published elements (contained in a published service profile). In order to establish a degree of relationship between these two elements, we use mapping expressions. Mapping expressions are initially specified to find simple correspondence between two schema elements that could be more or less complicated. We noticed in the previous state of the art, related to some of the most popular service discovery mechanisms, that an important evolution occurred on the service description media. SLP, for instance, used textual description to publish service profiles and matchmaking process between client's request and service profiles, is limited to a string comparison. This solution is too restrictive and not enough dynamic for pervasive application and new generation services like web services (using WSDL representation). More recent discovery technologies like WS-Discovery or UDDI adopted more flexible media like the XML format to describe services by offering a better meta-data representation. Currently, semantic web technology is enhancing SOA by introducing new formalisms based on the ontological representation of the service description. This new representation enables a real semantic based discovery that provides more dynamicity to the orchestration techniques like service composition or dynamic binding; but it also introduces more complexity on the matching expressions due to the growth of the size of the representation schema and the number of matches to be performed.

Di Martino [MAR06] discussed the major matching approaches that could be adopted for a service discovery application. He proposed a classification following these criteria:

- *Instance* vs. *schema*: matching the data or the schema-level information.

- *Element* vs. *structure*: matching an element composed of a set of attributes or a combination of elements.
- *Language* vs. *constraints*: matching using a textual description of the elements or using constraints related to keys relationships.

This classification gives an overview of the different matching possibilities that could be adopted for service discovery depending on the application domain.

2. Ontology Based Service Discovery

A number of standards supporting these functionalities have been adopted recently like OWL-S [MAR03] (previously DAML-S) is an ontology (defined in chapter Chapter VI.A.1) described in OWL with three main components (see Figure 9); first, the service profile that describes the attributes of the published services, attributes that will be used for the matchmaking with the discovery requests. Second the service model describing the behavior of the service in terms of process invocation, composition, monitoring, and updating. Third, the service grounding that provides the technical specifications for the access to the service providing useful information related to the messaging format, the protocols and the details about the access methods and their variables (could be contained in a WSDL description file). Other technologies like METEOR-S [MET], WSMO [WSM], and SWSF [SWS] also made use of semantic web languages like RDF or OWL

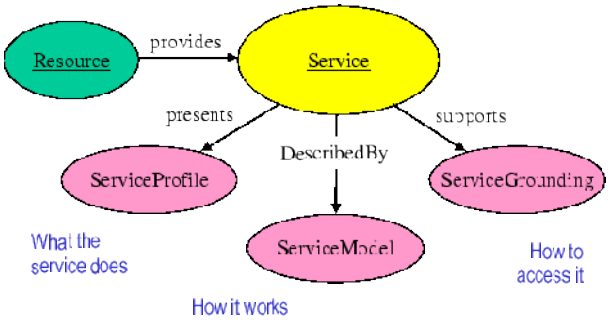


Figure 9 : Top Level of the OWL-S Ontology [MAR03]

F. Context Awareness and Service Discovery

The use of context represents a significant benefit to enable service discovery in the highly dynamic environments addressed in ubiquitous computing. Context or context information refers to any information that can be used to characterize the state of an entity (user, or software, or hardware component of a computing system) [DEY01]. The location of a service, obtained for instance through a GPS or WiFi-based location of the device on which a service is running, network bandwidth, or the security protocols enabled on some platform all may serve to characterize dynamic services and networks. Service discovery may obviously exploit context to achieve more precise matching in such environments. More importantly, such context information complements and provides more flexibility to the discovery policy specification. It in particular makes it possible to express fine grained discovery policies more closely following the constant changes of the environment and services.

[RAV06-2] exploited contextual information related to the network conditions: like the type of networks, supported protocols, number of active users, number of available services, etc. to select the most appropriate service instance and minimize the amount of generated traffic for the discovery. They demonstrate this performance gain by testing and comparing context-

aware service requests with classical requests and they observed a gain of number of exchanged messages and also in the processing and response time.
[DOU05] proposes an architecture for service discovery based on context-aware registries. In their system the authors provide a formal representation of the context related to services. The context is represented by a set of dimensions that takes discrete values over a specific domain that could be assimilated as a tree containing intermediate notes (context values) and leafs representing a contextual ID. Using this formal model combining the contextual information provided by services and clients, the context-aware discovery mechanism becomes more precise and improves the discovery performances. This model suffers from a lack of dynamicity concerning the context information values update.
[LEE03] presents a prototype of context-aware service discovery based on an extension of the Jini lookup service. Contrary to the previous solution, this one takes into account the dynamicity of the context value on the server side. During the registration, a service has to record two types of attributes: static attributes (does not change during the service lifetime) and dynamic attributes (contextual information change with the environment evolution). During the matchmaking process, the static attributes are first compared with the query in order to perform a static filter, and after that the discovery service evaluates the dynamic contextual information in order to perform a dynamic filter on the request.
[BRO04], [BRO04-2] propose to use an ontological description of the context information combined with an ontological description of the services (described in the previous section) in order to make the user query more information rich by increasing the precision of the matching results. Due to the high resource requirements of such ontological systems in terms of CPU consumption and memory space requirements, this solution is actually not adaptable for small mobile devices.

G. Threats and Security Requirements

1. Threats and Attacks

This section provides a list of threats and the possible attacks that can be built against the data and resources of service discovery players. Although this list is non-exhaustive most of the discovery protocols (centralized and decentralized) are studied and analyzed in order to find out weaknesses and vulnerabilities that could be exploited by attackers. For each threat, a possible countermeasure is proposed that can be applied in order to prevent the disruption of the service discovery service. The following description lists threats to the centralized and decentralized service discovery architectures together.

Protocol Messages and Entities

- The registry is not available (service-side): the attacker performs a Denial of Service attack by flooding registration messages. He intends to force the registry to consume its resources in such away as it can no longer provide its intended service. One of the possible countermeasures is to modify the protocol by adding anti-clogging messages, if message parsing is too costly, then blacklist originators of bogus messages.
- Client request disclosure (client-side): client intentions, activity, or identity may be revealed, directly or indirectly by his service lookup queries. An appropriate countermeasure consists in setting up secure channels (encryption). For instance, during an industrial social event, companies could provide Job services for people that want to apply for a job. Some companies by intercepting job lookup messages sent by users applying for a job are able to know people that may leave their company for another competitor company (see Figure 10)

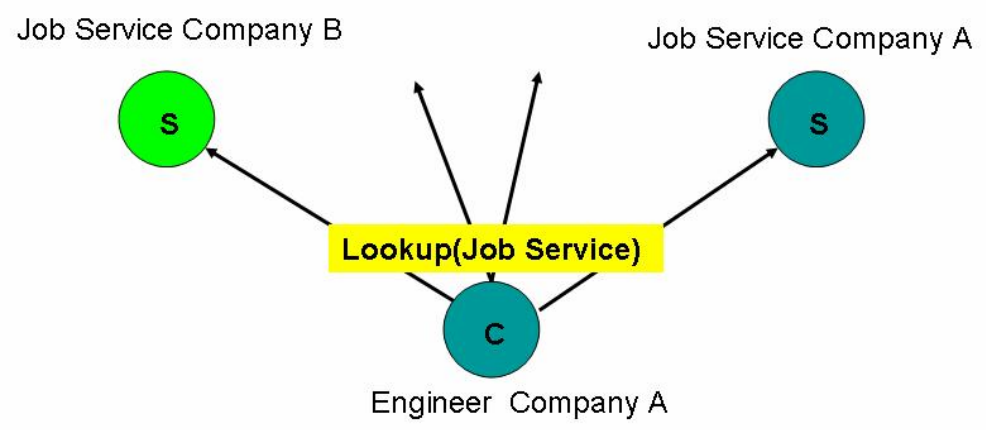


Figure 10 : Client request disclosure

- Interception of request (client-side): the discovery request reveals private information about service discovery clients. A possible attack consists in faking the identity of a registry that is known and trusted and forwarding to that registry. Registry certificate distribution might be an adapted countermeasure to prevent this type of attack. The process of distributing certificates of trusted registries should be protected during the configuration phase of the mobile device. A fake Bank Server could play a masquerade attack in order to obtain private banking information from users (see Figure 11).

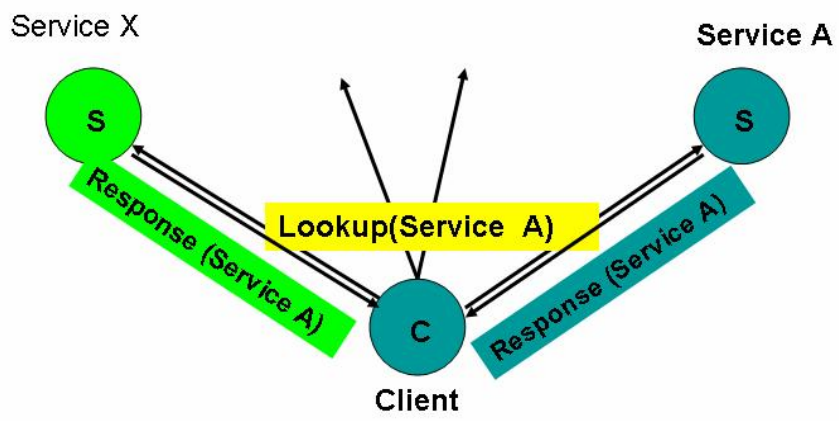


Figure 11 :Client request interception

- Message modification or drop (client side): if the attacker compromises a router from the network, he can intercept and modify or drop the client's lookup message to the registry (see Figure 12). The client should protect the message he sends with respect to its integrity and to the authentication of its origin, for instance with a signature or a message authentication code. A redundancy mechanism can be configured to guarantee the delivery of the messages in case of dropping.

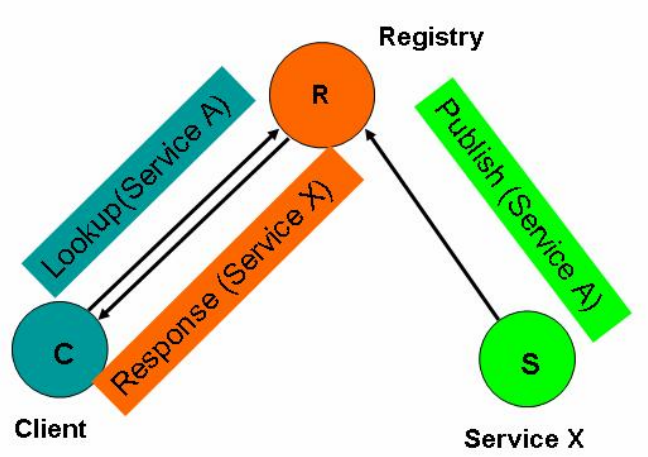


Figure 12 :Client request modification or dropping

- Replay of Request message DoS (client-side): the attack consists in replaying a lookup message coming from a legitimate client. A sequence number could be added to the message in order to drop the previously processed messages.
- Replay of registration message (registry-side): the attacker replays the registration message of a properly authenticated service in order to update the service profile with wrong information. A signed sequence number must be added to the registration message in order to take into account the processed messages and drop the relayed ones. An attacker could reuse a real banking service publish message in order to setup a fishing attack.

Service Registration (centralized architecture only)

- Registration to a malicious registry (server-side): an attacker might fake being a registry whose identity (and implicitly matching behavior) is known and trusted (see Figure 13). Subsequent attacks include preventing clients from matching the registered service. Registry authentication is one possible countermeasure. This can be achieved by ensuring a properly protected distribution of the certificates of trusted registries during the configuration of mobile device, which also requires an initial authentication phase during discovery; or by ensuring that registry keys are distributed to mobile devices and that communication with the registry is encrypted with that key.

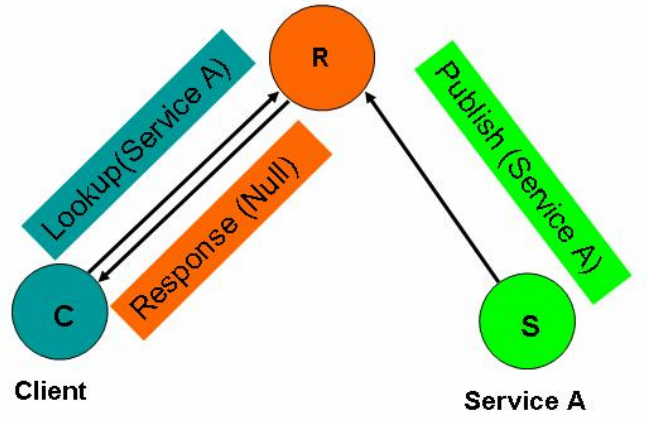


Figure 13 : Publishing services to fake registries

- A service can be deregistered by an unauthorized party (registry-side): this occurs when an attacker tries to dereference an active service from the registry which it

registered previously. The use of a nonce (e.g., sequence number) with a signature (MAC) by the registered service to certify the origin of a de-registration message constitutes a possible countermeasure to such attacks.

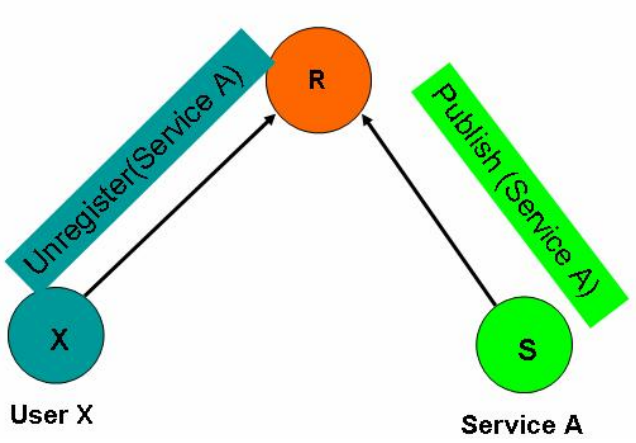


Figure 14 : Services deregistered by an unauthorized party

- Fake registration (registry-side): An attacker can send a fake registration message to the registry containing wrong information with fake attributes. To prevent this attack, the registry has to include a verification of the proper certification of attributes of registering services by appropriate authorities together with a proof of identity of the registering party (e.g., signature of registration request).

Matching process

- Client lookup disclosure (client-side): client intentions or activity might be disclosed if the matching process is open to all services registered. A service may have been established to gather statistics about users trying to access a certain profile of services. More dangerously, an attacker might try to get access to confidential information sent by the client at the access phase subsequent to service discovery. The countermeasure to this attack consists in restricting the services whose description matches the client lookup with additional constraints on some of their certified attributes. This specification can take the form of a policy submitted by the client together with his lookup request, and which may refer to the same or to attributes of the services different from those specified in the lookup request.
- Service discovered by unauthorized party (service-side): a typical example of this threat is the possibility for an attacker to determine the identity or content served by a service which wants to be seen or accessible only by a restricted set of other services (service trapping). The countermeasure consists in the delegation of a trusted (and authenticated) registry, the enforcement of a restrictive policy provided by the service that will allow the service discovery by authorized clients only. We also recommend the use of restrictive cryptographic mechanisms. This kind of threat can be extremely dangerous in case of industrial spying. A spy could discover all the interfaces and the architecture topology of an automated factory relying on web services for the workflow management of industrial machines.

	Threat	Attack	Countermeasure
Service Side	Non availability of the service	Brute force DoS attack	Anti-clogging mechanisms and IP blacklisting
	Registration to malicious	Fake registry that drops	Authenticating registries

	registry	or reuse illegally published information	
	Service disclosure	Intercepting restricted service profiles	Hide the service publishing and restrict the discovery to authorized users
Client Side	Request disclosure	Intercepting client's requests and deduce his intentions and his preferences	Protecting client's request from unauthorized access
	Request interception and modification	Spoofing attacks and reuse of clients requests for malicious purpose	Adding integrity mechanisms and checksums to requests
	Message drops	Packet interception and dropping. Adding a noise signal to physical communication layer	Redundancy mechanisms (used for UDP based protocols) and correcting code word mechanisms
	Request Replay	The attacker replays old requests	Sequence number and time stamp for messages
Registry Side	Non availability of the service	Brute force DoS attack	Anti-clogging mechanisms and IP blacklisting
	Replay registration messages	The attacker replays old registrations	Sequence number and time stamp for messages. Sender authentication
	Illegal deregistering services	The attacker fakes a deregistration message form a server to delete it from the registry	Message signature verification and the use of secret and unique publish ID or a nonce
	Fake registrations	Fishing attacks (registering fake services)	Service authentication

Table 2 : Service discovery threat model

2. Security Requirements for Service Discovery

Discovery is very often performed at the initiative of the service requester (e.g. lookup model) but can also be initiated by the service provider (e.g. advert model). The specificity of discovery is that these players, who may pertain to different administrative domains, are by definition initially unaware of their respective existence and security policies. The following requirements make it necessary to answer the relatively original threats described above:

- **Authentication**: the very objective of service discovery is to communicate with previously unknown entities that provide specific functionalities. Open discovery services therefore require that the first message sent (lookup or advert) be in clear, also meaning that the content of the message can be accessed. Without the means to authenticate clients and servers, service discovery makes the implementation of a man-in-the-middle attack possible [GHA04], a malicious entity being able to wrongly answer a discovery message. Registry based discovery schemes obviously make it much simpler than infrastructure-less ones to perform secure discoveries, since the registry is the only element which the client needs to identify and be identified to, yet at the price of additional infrastructure deployment requirements.
- **Authorization**: Some restricted services may not be discoverable by all clients. Only authorized clients are able to discover restricted services. This authorization must be provided by clients in order to prove their right to discover the service. Authorization can be materialized in security tokens, certificates, recommendations or access control lists providing necessary information about the rights of the client.

- **Privacy**: the discovery initiator takes a more important risk than the other party since he does not control the entities which will receive the discovery message, nor the potential usage of the information embedded in his request message. The information disclosed by client requests is likely to reveal a subset of the intentions of the service requester. An attacker may try to gather profiles of users of the service discovery mechanism based on the information carried by discovery messages as well as subsequent messages generated by the actual access to the service and display some more information (host name, Certificate, Credentials …). The correlation of such data with discovery related information is particularly worrying from a privacy protection perspective.
- **Confidentiality**: defined by the International Organization for Standardization (ISO) as "*ensuring that information is accessible only to those authorized to have access*": Confidentiality is strongly related to privacy; for instance when a data is labeled private, it also means confidential. Privacy is usually related to one entity (private information must be accessible only for the holder of this information) but confidentiality can concern a set of entities that share common particularities. A confidential document could be accessible for a group of users that share the same access rights (or roles). In service discovery, confidentiality is used to protect sensitive data (not merely private) contained in the discovery messages; it restricts access to these data to allowed users. Confidentiality protects from spoofing attacks, hides the intentions of a client contained in his service requests, protects from message modifications, and ensures the anonymity of the private services (by hiding their existence). Cryptographic techniques are usually applied to ensure confidentiality by ciphering clear data and making it accessible for users holding specific decryption keys.
- **Access control**: since client/service authentication is problematic in the initial discovery phase, traditional service oriented architectures do not support access control during discovery. Service providers would ideally advertise their services exclusively to authorized users, even though this objective is in practice difficult to achieve in a pervasive environment. Still, disclosing the description of a service to any requester potentially increases the risk that a malicious client or malware takes advantage of this knowledge and of the service vulnerabilities to gain unauthorized access.
- **Integrity**: The integrity of a data or a document concerns the detection (ultimately the correction) of possible errors or modification introduced on the data. In the previous threat model, we described an attack related to discovery message modification, alteration, deletion or replay. To protect discovery message from this kind of threat, an integrity mechanism must be applied to all messages to verify their correctness and be sure that sensitive information contained in discovery messages is original.
- **Accountability**: In case of malicious behaviour (like fake announcement sent by servers), clients or system administrators must be able keep a trace of this breach for an eventual appeal for a neutral judgment. These traces can take the form of logs or tokens.
- **Availability**: Availability: denial of service (DoS) is an attack against the availability of resources preventing the authorized access to a system resource or delaying system operations and functions. Openly exposing service descriptions during discovery enables attackers to exploit vulnerabilities by creating specially crafted messages for the server or by the registry. Notably, registries clearly constitute a single point of failure and therefore are particularly sensitive to brute force DoS attacks. Peer-to-peer

discovery on the contrary is expected to increase the availability of services on the whole.

H. Approaches Secure Service Discovery

1. Access Control on the Service Side

Initially in SOA systems the main security preoccupations were related to service access restrictions. For example, Universal Plug and Play (UPnP) Security [ELL03], [ELL02] and Bluetooth Security [BLU]. UPnP Security provides many authorization methods including access control lists, authorization servers, authorization certificates, and group definition certificates, and users are assumed to provide correct credentials. In the secure discovery mode of Bluetooth, services only respond to users that share a common secret. In this category, service providers may easily protect their privacy. If a user does not have the privilege to discover and access a service, then a service provider can remain silent. Users are obliged to expose in clear text their identities and service requests to service providers. Therefore, users may unnecessarily expose their sensitive information when service providers do not provide the requested services. Moreover, users need to memorize the relation among services, service providers, and credentials. Otherwise, they may not be able to supply correct credentials and, thus, they lose opportunities to access services.

2. Registry-Based Architecture

One of the first approaches dealing with secure service discovery was proposed by [CZE99]. This architecture relies on an additional component, called Service Discovery Service (SDS), which plays the role of a secure information repository (secure registry). This SDS helps clients and servers to set up a trust relationship and secure channels between each another: it provides authentication, access control, encryption, signature verification, and privacy protection using a PKI. The SDS multicasts its public key certificate in order to allow the encryption of the service publication and of the client request. A server also can restrict the discovery of its services to some specific clients by associating an access control list with the published service profile; a component of the SDS called the capability manager will use this list to enforce access control. The SDS, which has to be permanently available, and which has to decrypt all messages coming from clients and servers around therefore constitutes a single point of failure. It could create a bottleneck, and some malicious user could attack the SDS by sending fake encrypted messages. In order to encrypt the exchanged messages, the SDS uses a hybrid public/symmetric key system. Trust establishment between the SDS and other entities is limited to a simple verification of the SDS public certificate validity. This kind of infrastructure is heavy to manage and only based on certificate verification; in this case every user with a valid certificate is able to discover every existing service without any restriction. Contrary to our solution, clients and services do not have any possibility to define their own security preferences regarding discovery.

[ZHU03] also proposes an architecture for securing service discovery. In this work, components share a multicast address that will be used to bootstrap communication. Directories (i.e. registries) use this multicast address to periodically announce their unicast address and certificate. Proxies are used to protect the servers by handling the registration, authentication, authorization, and key management for them. Entities in the system set up a session using hybrid encryption. Due to the number of proxies (one by service) and the PKI infrastructure used to secure the communication, the model proposed is however likely to generate an important message overhead. Contrary to the claimed objective of this work to address pervasive systems, services are likely to be static, whereas the approach we advocate

only requires the local availability of a fixed registry (each client potentially being a server for other clients).

3. Privacy Issues for the Service Discovery

[ZHU04] addresses privacy protection aspects of the discovery process. The authors propose the use of Bloom filters to protect the client and server personal information set within a discovery request (identity, certificates, attributes…). Membership tests are performed between the directory and the client using generated Bloom filters in order to authenticate themselves. The participating entities must agree beforehand on specific hash functions in order to use these Bloom filters, yet this issue is not resolved but through a static agreement. The scope of the restrictions is very poor compared to our policy solution that provides an efficient semantic expressiveness used to define the security preferences of each entity.
Carminati [CAR05] also raised the question of privacy issues in Web Services with untrusted UDDI-based Discovery Agencies (equivalent to a foreign agency providing a registry service). After describing the privacy requirements related to the discovery mechanisms, they provide five UDDI-based registries scenarios (Internal enterprise application, Portal, Partner catalog, and e-Marketplace). For each scenario they proposed the application of three privacy enforcement strategies: Access-Control based solution using a third trusted party (a trusted UDDI registry) that is in charge of the access-control policy enforcement to the registry. Cryptographic-Based Solution, also relying on a trusted third party called encryption module that is in charge of encrypting sensitive data (XML encryption) according to a specific privacy policy provided by clients and services. Hash-Based solution where service providers publish hashed services in an untrusted registry.

4. Registry-less Architecture

Carminati's solution relies on a trusted third party or a trusted registry to secure the service discovery that is not adapted for a decentralized discovery configuration (like UPnP) that does not rely on a fixed infrastructure. For this reason a secure pervasive discovery protocol SPDP for infrastructure-less and in particular registry-less systems is proposed by Almenarez et al [AL03]. This protocol relies on a trust model to decide which devices may participate to the network and consequently to the discovery. The trust model is based on two trust degrees: direct trust or recommendation (a of web-of-trust model). This solution is still limited in terms of attacks because the setup step of the trust establishment procedure between new devices must be direct although with unknown ones.

Chapter II. Securing Decentralized Service Discovery

A. Introduction

In the previous section we provided a threat model followed by a set of security requirements that must be integrated in order to protect and secure service discovery mechanisms. An important distinction, to ensure an efficient secure system is to separate the solution related to a decentralized discovery from a centralized one then try to provide a hybrid solution able to adapt the security to every architectural configuration. Of course several solutions (described previously) provided interesting mechanisms to secure the discovery, but most of them are not adapted to pervasive environment for they require a fixed and static infrastructure.
In this chapter we present a security solution dedicated to a decentralized configuration (P2P communication) in which elements do not rely on a specific infrastructure or on a specific component (like a registry) to perform a service discovery. The discovery message exchange is performed in a P2P fashion. Each element relies only on it self. The challenge in this case of architecture consists in finding a way to establish a trust relationship between different actors of the system without requiring of a fixed trusted element that belongs to the environment (permanently in charge of establishing this trust relationship). The objective is to provide a simple mechanism that can be used by a new element of the system in order to perform a safe and secure discovery without having any a priory knowledge about the environment.

B. Technical Background

Before starting the description of our security solution we first choose to introduce some preliminary definitions related to the different technologies and building blocs used in order to develop these solutions.

1. Identity Based Encryption

The identity based encryption (IBE) scheme is an asymmetric encryption mechanism proposed by initially by Shamir [SHA84] and developed practically by Boneh and Franklin [BON01]. The principal advantage of this mechanism compared to other asymmetric key mechanisms like PKI; is the avoidance of the public key verification by a user that must verify the identity of a public certificate holder by contacting the certificate authority that issued this public token. In order to avoid this verification, the public key becomes

semantically dependent with the holder identity; compared to PKI for which the public encryption key has no semantic meaning (it is just a random value), in IBE the public key is semantically expressed as an identity (or unique identifier). In this case the IBE scheme allows to encrypt a message according to an identity in way such that only an entity holding the appropriate private key related to his identity is able to decrypt the message. For example Alice that wants to encrypt and send a message to bob can use bob's mail address as a public encryption key (bob@mail.com) and bob can use the private key related to this identifier in order to decrypt the message.

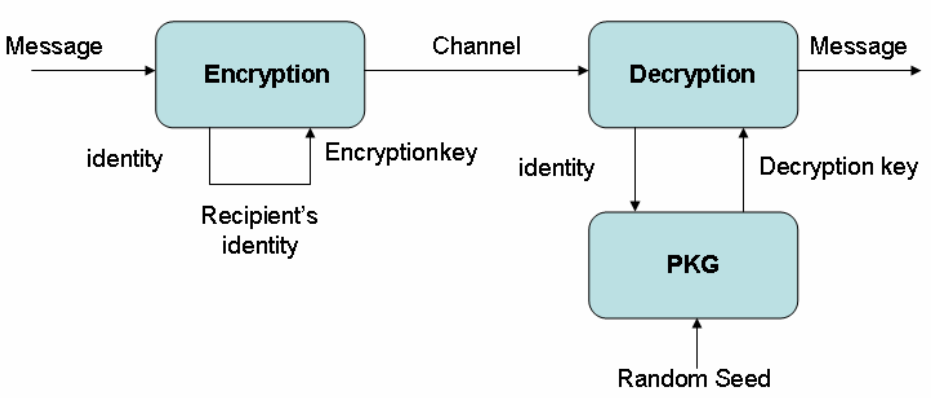


Figure 15 : Identity Based Cryptosystem

The private key related to an identity is generated by a private key generator (PKG) that belongs to the certificate authority. A shared secret called system parameters is used on both side to encrypt and decrypt the message. This secret is provided by the certificate authority.

2. Attribute Based Encryption

An identity can be described using a set of attributes. Like for example information contained in a passport (size, weight, birth date, birth city, nationality, biometric data, etc) putted together can describe the identity of the passport holder. For this reason Sahai and Waters [SAH05] proposed a new scheme called Fuzzy Identity Based Encryption (FIBE) that allows for a private to encrypt a message using a set of attribute ω', and decrypt it using a set of private keys related to a set of attributes ω if and only if $|\omega \cap \omega'| \geq d$ where d is fixed at the system setup. This concept is improved by Goyal et al [GOY06] in order to propose the Attribute Based Encryption (ABE) that provides the possibility for a user to encrypt messages according to a set of attributes with the possibility to make logical combinations between these attributes ("AND", "OR" operators). In order to decrypt such message, the user must use a correct combination of a set private keys related to the encryption attributes. Usually an indication about the combination must be sent in clear in order to combine the appropriate private keys. The attribute combination can be associated to an encryption policy. Pirretti et al [PIR06] implemented the ABE crypto system and provided a performance evaluation illustrating the impact of attribute combination (or policy complexity) used to encrypt and decrypt messages.

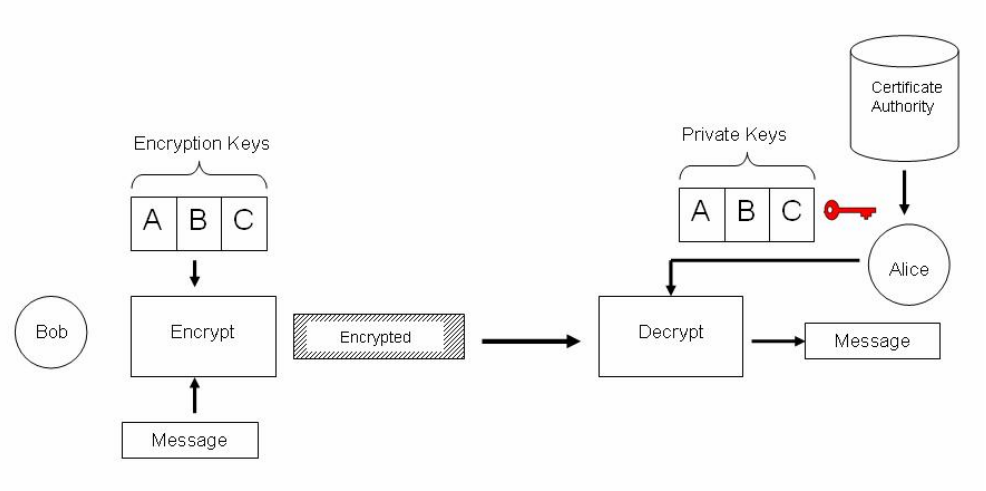


Figure 16 : Attribute Based Encryption Cryptosystem

In [GOY06] Private keys are identified by a tree-access structure in which each interior node of the tree is a threshold gate and the leaves are associated with attributes. A user will be able to decrypt a ciphertext with a given key if and only if there is an assignment of attributes from the ciphertexts to nodes of the tree such that the tree is satisfied.

3. Attribute Based Algorithm

[PIR06] provided an informal specification of the attribute based encryption system as a collection of four algorithms:

- **Setup(k)**: The Setup algorithm is run by the certification authority in order to create a new ABE system. Setup takes as input a threshold value, *k* and outputs a master key *MK* and a set of public parameters *PP*.
- **Key-Generator(Attributes,MK)**: The authority executes the Key-Generator algorithm in order to generate a new private key *PK*. The algorithm takes as input the user's identity, *Attributes*, as a set of strings representing a user's attributes and the master-key *MK* and outputs user's secret key *PK*.
- **Encrypt(M, Attributes ,PP)**: The Encrypt algorithm is run by a user to encrypt a message *M*, with a target set *Attributes*, and the public parameters. It outputs a ciphertext, *C*.
- **Decrypt(C, Attributes, PK)**: The Decrypt algorithm is run by a user with identity *Attributes* and private key *PK* to attempt to decrypt a ciphertext C that has been encrypted with *Attributes*.

4. Private Key Generation: Online Vs Offline

The private key generation is problematic for two reasons. First, how can the user contact the Private Key Generator (PKG) in a secure manner (secure key distribution)? Second, what are the appropriate keys required to decrypt the messages (key generation)? One of the interesting features of IBE and ABE systems is the choice of the key distribution mechanism, for which there exist three possible architectures:

- Online PKG: there exists a key server that is permanently running and reachable by the users in a secure manner. The user sends the server the requested attributes, and depending on his profile, the server generates and sends the private key corresponding to these attributes. It is a simple solution, yet inapplicable to ad-hoc environments non permanently connected to a network infrastructure.

- Offline PKG: the key server generates all the private key corresponding to the attributes contained in the user profile. These keys are stored into a protected directory in the user's laptop. The user will choose among his key collection the appropriate private key corresponding to the expected attributes of the recipient. This architecture makes it difficult however to change the system parameters (profiles, attributes, keys) or to manage the revocation of keys.
- Embedded PKG: the key server is implemented in a tamper-resistant hardware like a smartcard provided by the certification authority. This solution represents an ideal compromise between the two precedent solutions for ubiquitous computing applications. Using an embedded PKG the user that needs to get a private key does not have to contact in a secure manner a distant PKG. In fact in order to protect the sensitive information (private key, certificate) during the key exchange, the two involved entities have to setup a secure channel. The second advantage of using an embedded PKG, is the permanent availability, the user does not have to be permanently connected to a network to reach a distant PKG.

C. Enabling Secure Service Discovery with Attribute Based Encryption

1. Introduction

Service discovery is rendered necessary when clients need to locate services they can describe but that they do not necessarily know, thereby rendering PKI based solutions, which require a preliminary key distribution, awkward and contrived. In contrast, the new concept of Attribute Based Encryption, derived from Identity Based Encryption schemes, makes it possible to secure communications with unknown services based solely on their description, and in a peer-to-peer fashion, that is, without the introduction of any additional trusted third party like a registry. This technique is at the core of the mechanism we propose for securing the peer-to-peer discovery of services. This paper first reviews which security properties are expected from this architecture. It then goes on to detail how to integrate this mechanism within the WS-Discovery Web Service protocol.

2. Profiles and Attributes

In a SOA architecture, every entity (client/service) exposes some information about itself through one or more profiles. This information is useful for the users to distinguish between the different entities of the system. Just like an identity card contains particular characteristics of its owner like his name, his age, his home address, profiles characterize an entity through the enumeration of attributes. Profiles can be used by services in order to announce themselves and can be published in a public repository accessible to all users. During a service discovery process, the server publishes its service profile (service description). The attributes contained in this profile can be useful for the user to select the service he wants to contact. Depending on the technology used for the service deployment, profiles will take different forms: service description in the Web Service framework may for instance take either the form of a WSDL profile consisting in an XML-based file, or of a DAML-S profile made of an OWL-S based semantic web description. In WS-Discovery, the service profile is composed of two strings (Type and Scope). Similarly in CORBA, the description of the service is limited to a name within a context; this name is contained in a naming graph where each name is associated with a reference to the service. In Jini, the service registers its serialized proxy object together with a set of relevant attributes (Entry) that may be later use during discovery. The client profile can be described in a certificate that contains some indications about his identity and public key (X.509 Certificates) and also some other

attributes like the roles, the rights, or the delegations (Eureca [CRO05], X.509 Attribute certificates). As can be seen, all the attributes related to the description of users and services can be used to distinguish between the different entities involved in the system and also to improve the knowledge about the surrounding environment.

3. Applying Attribute Based Encryption

The goal of our solution is to protect the sensitive information contained within the WS-Discovery messages. To reach this objective, we applied ABE-mechanism to the principal messages exchanged during the discovery phase. The attributes used to encrypt the data are precisely the attributes enabling to describe a service:

- ***Type***: An identifier of the service endpoint (logical name describing the capability of the service. Ex: Printer, TV …)
- ***Scope***: An extensibility point that may be used to organize the services into logical groups (Ex: for the printer service the scope could be Color, Black & White …)

Assuming now that these two attributes identify the service, they can then be used to protect the client's probe messages by encrypting other attributes of the message[4]:

```
<s:Body ... >
      <d:Probe ... >
            <d:Types>
                ***Printer***
            </d:Types>
            <d:Scopes >
                **Eurecom**
            </d:Scopes >
            ...
      </d:Probe>
</s:Body>
```

Figure 17 : Probe Message

In this example, the Probe message (Figure 17) content could be self-encrypted using the attributes (Ex: $Encrypt[Attribute]_{Attribute}$) in order to hide the mandatory services' attributes requested by the user (the requested service type and the *EndpointReference* of the requester). This guarantee that only the service that holds the private keys corresponding to these attributes are able to decrypt and process the Probe message. Of course these private keys should be provided by a trusted PKG only to trusted services. The PKG should therefore verify the credentials exhibited by the service, this can done using existing PKI infrastructures and a specific X509v3 profile. The profile should be tuned to capture the attributes describing the service.
Now let's focus on the service's response, the *ProbeMatch* message (described in Figure 18) that must also be protected especially since the content of the message provides a set of attributes offering a precise description of the service (location, address, URI).

[4] In the messages, the s namespace refers to soap (http://www.w3.org/2003/05/soap-envelope) , the d namespace refers to WS-Discovery (http://schemas.xmlsoap.org/ws/2005/04/discovery/) and d namespace refers to WS-Addressing (http://schemas.xmlsoap.org/ws/2004/08/addressing).

```
[<d:ProbeMatch ... >
      <a:EndpointReference>
            <a:Address>
            uuid:98190dc2-0890-4ef8-
   ac9a-5940995e6119
     </a:Address>
      <a:EndpointReference>
      <d:Types>
            Printer
      </d:Types>
      <d:Scopes>
            Eurecom
      </d:Scopes>
      <d:XAddrs>
         http://printer.eurecom.fr/
      </d:XAddrs>
      <d:MetadataVersion>
                  75965
      </d:MetadataVersion>
...
</d:ProbeMatch>]
```

Figure 18 : ProbeMatch Message

All these attributes could be encrypted using a unique identifier of the user that requested the service. In order to avoid carrying extra information, the *endpoint-reference* information contained in the *ReplyTo* tag (Figure 18) of the Probe message's header, can be used as the identifier of the user. In this case the encryption action will be notated $Encrypt[ProbeMatch]_{Client_ID}$ and only the owner of the identity described in this endpoint-reference that holds the appropriate private key corresponding to this identifier is able to decrypt the Probe Match message. As described previously, this private key can be provided by a PKG relying on existing PKI infrastructure.

a) Securing Client Request

In the decentralized model, protecting private data contained in the request message is different from restricting the service discovery to some allowed clients. For this reason we separated the two security aspects of encrypting request messages and publish/Response messages.

Concerning the client's request XML message generation and processing functions, the WS-Discovery protocol was modified as follows:

- During the Probe message generation the clear text String contained in the Type tag is replaced by the encrypted text (Figure 19), then the protected probe message is broadcasted to the entire multicast group. In this example, the type Printer is encrypted.

```
<s:Body>
      <d:Probe>
        <d:Types>
          (Encrypt[Printer]{Printer|Eurecom})
        </d:Types>
      </d:Probe>
  </s:Body>
```

Figure 19 : Encrypted Probe Message

- All services listening to the multicast group will receive the message but only the intended recipients will be able to decrypt the content of the Type tag in order to process the received message. The encrypted text will be extracted from the Type tag by using the secret key corresponding to the appropriate type of service (Figure 20), thus making it possible to retrieve the clear text (Printer).

```
<d:ProbeMatch>
      <a:EndpointReference>
            <a:Address>
                  uuid:dc1c483f-8bc8-
48b9-9e34-e6546645c2ec
            </a:Address>
      </a:EndpointReference>
            <d:Types>
                  Printer
            </d:Types>
            <d:MetadataVersion>
                  1
            </d:MetadataVersion>
```

Figure 20 : Clear Probe Message

b) Securing Service Publish/Response

After receiving and processing the client request (*Probe* message) a server depending on his security requirements will encrypt his response message according the client's identity as described previously, or according to an access control policy to restrict the discovery to some allowed users. In the second case the service should encrypt his publish messages (*Hello*) or his response messages (*ProbeMatch*) using some attributes as public keys. These attributes correspond to the profile of users that must be allowed to discover his services.
Let's take the example of a server publishing a color printer service in a university, with a restriction concerning the users authorized to discover it. This restriction allows only professors from the university staff to discover the color printer service. In this case the *Hello* message will be modified (Figure 21) and all the date corresponding to the service profile will be encrypted according to the restriction policy:

```
<d:Hello ... >
<a:EndpointReference> Encrypt[EPR]{Professor,Eurecom} </a:EndpointReference>
      <d:Types> Encrypt[Color_Printer]{Professor,Eurecom} </d:Types>
      <d:Scopes> Encrypt[Eurecom_Printer]{Professor,Eurecom} </d:Scopes>
      <d:XAddrs> Encrypt[Colorprinter.eurecom.fr]{Professor,Eurecom} </d:XAddrs>
      <d:MetadataVersion>xs:unsignedInt</d:MetadataVersion>
...
</d:Hello>
```

Figure 21 : Encrypted Hello Message

The same kind of modification can be done in the service response message (*ProbeMatch*) in order to restrict the access to the color printer discovery for the user that are not professors of Eurecom university Staff.

D. Algorithms for Decentralized Secure Service Discovery System

The modified secure service discovery protocol is quite similar to the original one in terms of basic operations (Publish, Request, Response …) we some additional functionality related to the message encryption.
The request (lookup, *Probe*) algorithm Figure 22 describes the procedure used by the client to create a request message, to encrypt it and then send it:

Algorithm1: Request (*attributes*)

0. **Begin**
1. *RM* ← generate a Request message (*request*);
2. *EM* ← Encrypt the request message according with the appropriate attributes (*attributes*);
3. Senden encrypted message via multicast (*Multicast_Addr*);
4. **End**

Figure 22 : Request Algorithm

The response message (*ProbeMatch*) algorithm in Figure 23 describes the procedure used by a service that receives an encrypted message from a client. First the server has to decrypt the message, process, built the corresponding response, encrypt the response and send it back to the client. Concerning the encrypting arguments two cases are possible; the service is restricted, in this case the response is encrypted according the attributes related to the authorized users. The service is not restricted; in this case the response is encrypted according to the client identity.

Algorithm2: Response (*EM*)

0. **Begin**
1. *CM* = Decrypt (*EM*, Get_private_key(*KeyStore*);
2. **if** (*CM* != *null*)
3. *RM* = Generate_Response_Message (*Service_Profile*);
4. **if** (! *Service_Restricted*)
5. *RE* = Encrypt (*RM*, *CM.Sender_ID*);
6. **else**
7. *RE* = Encrypt (*RM*, *Client_attributes*) ;
8. **endif**
9. Send_Unicast (*RE*, *CM.Sender_Addr*);
10. **endif**
11. **End**

Figure 23 : Response Algorithm

E. Private Key Management

1. Requesting Private Keys from an Online PKG

The IETF[5] made some recommendation regarding to the private key request and transmission in a secure manner. Basically to obtain private keys, a client performs a http request to a remote server. The request must happen over a secure protocol. Both of the client and the PKG can use TLS. When requesting the URI the client must verify the server certificate and starts the keys transmission over a TLS secured connexion. The private key request can be structured using the following XML format:

```
<ibe:request xmlns:ibe="urn:ietf:params:xml:ns:ibe">
            <ibe:header>
               <ibe:client version="clientID"/>
            </ibe:header>
            <ibe:body>
            <ibe:keyRequest>
```

[5] The Internet Engineering Task Force (IETF) http://www.ietf.org/

```
                <ibe:algorithm>
                  <oid> algorithmOID </oid>
                </ibe:algorithm>
                <ibe:id>
                ibeIdentityInfo
                </ibe:id>
            </ibe:keyRequest>
            </ibe:body>
</ibe:request>
```

Figure 24 : Private key request format

The key server replies to the request with a protected http response. This response has a status code indicating success or no of the transaction. If the response contains a client error or server error status code, the client must abort the key request and fail. The private key response can be structured using the following XML format:

```
<ibe:response xmlns:ibe="urn:ietf:params:xml:ns:ibe">
            <ibe:responseType value="responseCode"/>
            <ibe:body>
               bodyTags
            </ibe:body>
 </ibe:response>
```

Figure 25 :Private key response format

The response code describing the type of response from the PKG is listed below:
100 KEY_FOLLOWS
101 RESERVED
201 FOLLOW_ENROLL_URI
300 SYSTEM_ERROR
301 INVALID_REQUEST
303 CLIENT_OBSOLETE
304 AUTHORIZATION DENIED

2. Private Key Generation: Online Vs Offline

The private key generation is problematic for two reasons. First, How can the user contact the Private Key Generator (PKG) in a secure manner (secure key distribution)? Second, what are the appropriate keys required to decrypt the messages (key generation)? One of the interesting features of IBE and ABE systems is the choice of the key distribution mechanism, for which there exist three possible architectures:

- **Online PKG**: there exists a key server that is permanently running and reachable by the users in a secure manner. The user sends the server the requested attributes, and depending on his profile, the server generates and sends the private key corresponding to these attributes. It is a simple solution, yet inapplicable to ad-hoc environments non permanently connected to a network infrastructure.
- **Offline PKG**: the key server generates all the private key corresponding to the attributes contained in the user profile. These keys are stored into a protected directory in the user's laptop. The user will choose among his key collection the appropriate private key corresponding to the expected attributes of the recipient. This architecture makes it difficult however to change the system parameters (profiles, attributes, keys …) or to manage the revocation of keys.

- **Embedded PKG**: the key server is implemented in a tamper-resistant hardware like a smartcard provided by the certification authority. This solution represents an ideal compromise between the two precedent solutions for ubiquitous computing applications. Using an embedded PKG the user that needs to get a private key does not have to contact in a secure manner a distant PKG. In fact in order to protect the sensitive information (private key, certificate …) during the key exchange, the two involved entities have to setup a secure channel. The second advantage of using an embedded PKG, is the permanent availability, the user does not have to be permanently connected to a network to reach a distant PKG. But the weakness of this solution concerns the Attribute certificate revocation management. Because of the off-line status of the PKG, it is not possible to verify if the attribute certificate provided by the key requester is revoked by the CA.

3. Key Revocation

Identity Based cryptosystems do not really have a key revocation mechanism: if a key is corrupted, all the entities related to the CA have to change the system parameters. The solution to limit the impact of this issue therefore consists in defining a key validity expiration date. Compared with public key certificate schemes in which the identifier is a random value revocable and replaceable over time, an IBE public key identifier is a unique name that cannot be revoked. In contrast, as proposed in [BON01], it can be extended with a key validity period (ex. time, date): we can for instance concatenate to this name with a variable value like a date, e.g., {Printer || 2008}, in which case only the node with private key related the attribute printer valid for the year 2008 can sign or decrypt messages.
The expiration-life solution is not efficient in case of explicit revocation (key corruption, Identity corruption, excluding or replacing nodes of the system) where keys are revocated before the expiration deadline. For this reason Hoeper et al. [HOE06] proposed a revocation method that relies on an additional data ν added to the public key and representing the version number of the issued public key. In this configuration the public key is built as:{ Attributes || Date || Version }
The version parameter is incremented every key renewal. For this solution authors proposed an alert system in charge of managing the events related to key corruptions, and informing the nodes of the system about the version incremental of each public key.

F. Use Case Scenarios

This section introduces a scenario that illustrates the application domains of such secure service discovery solution. Let us suppose that an airline company offers wireless services during flights (news, e-mail, movies, duty-free shopping …). Depending on the class of his seat, each passenger will have different access privileges to these services.
The shopping service would be accessible to all passengers without any restriction. Every passenger sending a service discovery request containing the { *shopping* } keyword with a laptop will receive a response containing the details of where and how to access the digital shopping mall (Figure 26). Of course, it is assumed that this service location needs not be protected from other passengers since it is publicly accessible. This does not preclude subsequent requirements for access control to the duty-free shopping service, for instance because one has to check the validity of the passenger's credit card number before agreeing to some transaction.



Figure 26 : Discovering shopping service in insecure mode

Passengers in business and first class may also get access to their e-mail. The response sent by the e-mail service will need to ABE-encrypted in order to restrict its access to business and first class passengers only (Figure 27). The response may contain credentials in order to enable the passengers to access the service. All other passengers should be unable to locate the service, much less gain access to it.



Figure 27 : Discovering restricted e-mail services

Passengers in first class may also request a premium movie service that lets them access to recent movies. The premium service should only be accessible in presence of an adult passenger, in particular in order to protect children against offensive or violent contents. A service discovery request for the premium movie should thus contain the requesting passenger's age for instance, yet such personal information should remain as confidential as possible. Encrypting the service discovery request to render it accessible by the premium service only would be enough to protect the passenger's privacy. However, to cope with requirements regarding access control to the service description, the discovery response will also need to be encrypted according to requester's age, so that the location of the movie service will be known only to adult passengers travelling in first class (Figure 28).

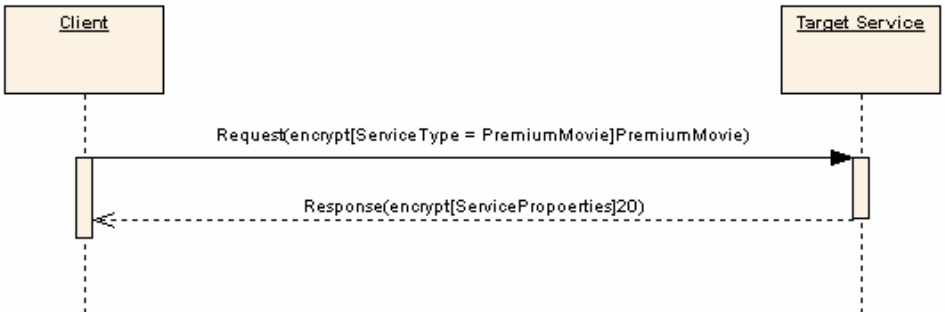


Figure 28 : Discovering restricted movie service with privacy protection

In all these examples, the assurance that some trusted authority did grant attributes to a service according to an agreed upon taxonomy is all a passenger needs to protect the privacy of his lookup messages. The same holds true with respect to the granting of attributes to

passengers. In all these scenarios, identities (or attributes) therefore are central to the referencing of services or passengers.

G. Security Evaluation

In this chapter we proposed a secure service discovery protocol relying on the ABE cryptosystem. The parameters of our security analysis are similar to the security properties offered by the mechanisms used in our solutions according to the Bilinear Diffie-Hellman (BDH) assumptions.

1. Proof of Security

Theorem : *if an attacker is able to break the ABE scheme, then a simulator can be constructed to play a decisional BDH game with a non-neglegible advantage*

Proof: supposing that exists a polynomial-time adversary *A* able to build attacks against the ABE scheme with an advantage ε. A simulator *B* playing a decisional BDH game with an advantage $\varepsilon/2$ can be executed with the following parameters:
A challenger able to set the groups G1 and G2 with a bilinear map *e* and a generator *g* is able to flip a fait binary coin μ. The challenger will set his parameters according to μ values for random {*w, x ,y ,z*} values:

$$\begin{cases}(W,X,Y,Z)=\left(g^{w},g^{x},g^{y},e(g,g)^{wxy}\right); \mu=0\\ (W,X,Y,Z)=\left(g^{w},g^{x},g^{y},e(g,g)^{z}\right); \mu\neq 0\end{cases}$$

We assume the universe *U* defined

Challenge execution:

Init: the adversary *A* chooses a set of attributes γ to be challenged upon.

Setup: the parameter $J=e(W,X)=e(g,g)^{wx}$. For all $i\in U$ we define a function T_i that chooses a random $r_i\in \mathsf{Z}_p$ is defined if $i\in\gamma$ and sets $T_i = g^{ri}$ for $t_i = r_i$; otherwise it shooses a random $\beta_i\in \mathsf{Z}_p$ and sets $T_i = g^{x\beta_i} = X^{\beta_i}$ for $t_i = x\beta_i$ then it gives the public parameters to *A*.

Phase 1: *A* makes requests for the keys corresponding to any access tree (defined in [GOY06]) structure Γ such that the challenge set γ does not satisfy Γ. Lets suppose that *A* can make a secret key request for an access structure Γ where $\Gamma(\gamma)=0$, in order to generate the private keys, *B* needs to assign a polynomial Q_a of degree d_a for every node in the access tree Γ.

Two procedures can be defined: *PS* that sets up sets up the polynomials for the nodes of an access sub-tree with satisfied root node. PU that sets up sets up the polynomials for the nodes of an access sub-tree with unsatisfied root node.

- *PS*(Γ_a; °; λ_a) where, $\Gamma_a(\gamma)=1$. The procedure takes an access tree Γ_a (with root node a) as input along with a set of attributes γ and an integer λ_a 2 Z_p.
  The function *PS* first sets up a polynomial q_a of degree d_a for the root node *a*. It sets $q_a(0)=\lambda_a$ and then sets rest of the points randomly to fix q_a. Now it sets polynomials for each child node *a'* of *a* by calling the procedure that could be written

PS($\Gamma_{a'}$; °; $q_a(index(a'))$). Notice that in this way, $q_{a'}(0) = q_a(index(a'))$ for each child node *a'* of *a*.

- *PU*(Γ_a; °; g^{λ_a}) where $\Gamma_a(\gamma) = 0$. The procedure takes an access tree Γ_a (with root node a) as input along with a set of attributes γ and an element $g^{\lambda_a} \in$ G1 (where λ_a 2 Z_p.)
  The function *PU* first sets up a polynomial q_a of degree d_a for the root node *a* such that $q_a(0) = \lambda_a$. Because $\Gamma_a(\gamma) = 0$, no more than d_a children of *a* are satisfied. Let $h_a \le d_a$ be the number of satisfied children of *a*. For each satisfied child *a'* of *a*, the procedure chooses a random point ¸$\lambda_{a'} \in Z_p$ and sets $q_a(index(a')) = \lambda_{a'}$. It then fixes the remaining $d_a - h_a$ points of q_a randomly to completely define q_a.

Now the algorithm recursively defines polynomials for the rest of the nodes in the tree as follows. For each child node *a'* of *a*, the algorithm calls:

- *PS*($\Gamma_{a'}$; °; $q_a(index(a'))$) if *a'* is a satisfied node. When $q_a(index(a'))$ is known
- *PU*(Γ_a; °; $g^{q_a(index(a'))}$), id *a'* is not a satisfied node. When only $g^{q_a(index(a'))}$ can be obtained by interpolation as only $g^{q_a(0)}$ is known. And $q_a(0) = q_a(index(a'))$ for each child node *a'* of *a*.

To give keys for the access tree Γ, simulator first runs *PU*(Γ, γ ,*A*) to define a polynomial q_a for each node *a* of Γ. When for each node *a* of Γ, q_a is known if *a* is satisfied; if *a* is not satisfied , then at least $g^{q_a(0)}$ is known. Furthermore $q_r(0) = w$.
Simulator now defines the polynomial $Q_a(\bullet) = xq_a(\bullet)$ for each node *a* of Γ. Where $b = Q_r(0) = wx$. The key corresponding to each leaf node is given using its polynomial as follows:
Let *i=attr(a)*

$$D_a = \begin{cases} g^{\frac{Q_a(0)}{t_i}} = g^{\frac{wq_a(0)}{r_i}} = X^{\frac{q_a(0)}{r_i}} ; att(a) \in \gamma \\ g^{\frac{Q_a(0)}{t_i}} = g^{\frac{wq_a(0)}{w\beta_i}} = g^{\frac{q_a(0)}{\beta_i}} ; otherwise \end{cases}$$

The simulator is now able to construct a private key for the access tree Γ. Furthermore, the distribution of the private key for Γ is identical to that in the original scheme.

Challenge: the adversary *A* submits two challenge messages m_0 and m_1 to the simulator. The simulator flips a fair binary coin v, and returns an encrypted messages m_v :

$$E = \left(\gamma, E' = m_v Z, \left\{E_i = C^{r_i}\right\}_{i \in \gamma}\right)$$

If $\mu = 0$ then $Z = e(g,g)^{wxy}$ and if s = y then $Y^s = \left(e(g,g)^{wx}\right)^y = e(g,g)^{wxy}$ and $E_i = \left(g^{r_i}\right)^y = Y^{r_i}$ in this case the encrypted message is a valid random encryption of m_v.

Otherwise if $\mu = 1$ then $Z = e(g,g)^z$ we have $E' = m_v e(g,g)^z$. Since z is random *E'* will be a random element of G2 from the adversaries view and the message contains no information about m_v.

2. Security Analysis

Assuming that the ABE cryptosystem proposed and analyzed in the security proof is secure and robust to attacks, we evaluate the robustness of our system.

Definition 1 spoofing attack is a situation in which one person successfully masquerades as another by falsifying data and in order to access to discovery messages

Property 1 each entity accessing to a protected message must be authenticated

Proof- To obtain an ABE private key necessary to decrypt protected messages, the recipient of an ABE-encrypted message provides the attribute certificate credential to a PKG and requests the private keys that correspond to its certified attributes. The PKG authenticate the client before issuing the key (Certificate verification and digest). The Authentication may either be done through the secure transport protocol (either http TLS or SSL). If a user *A* holding an X.509 attribute certificate describing attributes {*x*,*y*,*z*}, only the set of private keys {PK_x, PK_y, PK_z } will be generated by the PKG after the certificate digest.

Property 2 private data contained in client messages is protected against unauthorized access

Proof- The private data related to the clients; like the address, the identity, the intension, and the favourite services are encrypted and accessible only by the concerned services. According the ABE system only authenticated and authorized services can hold the corresponding private keys. Therefore users listening to communication issued by the clients are unable to decrypt exchanged messages.

Property 3 restricted services are invisible for non authorized users

Proof- Services can encrypt their service announcement and their responses according to chosen attributes. These attributes correspond to the profiles of the allowed users to discover the service. According the ABE system only authenticated and authorized users can hold the corresponding private keys. Therefore unauthorized users are unable de detect the existence of the restricted services. This is called implicit authentication.

Property 4 fake services are automatically detected and ignored

Proof- If a service is able to decrypt a client's request means that the appropriate private keys were used. In order to obtain private keys related to some attributes, a user has to authenticate to the Certification authority that verifies the capabilities of the services then generates the private keys related to these attributes. Let's suppose a malicious server *F* that sends a service response for every encrypted message sent in the network. If a client sends an encrypted request according to the attributes {*x*,*y*,*z*} only responses related to these attributes are accepted, the others are rejected.

H. Experimental Results

In order to evaluate the efficiency of this new secure service discovery model, we developed a Java implementation of the WS-Discovery protocol combined with Voltage IBE toolkit [VOL]. The Voltage IBE toolkit (C library) provides a high level interface for an easy integration with any application.

Our early experiments show that the application of the IBE mechanisms to the WS-Discovery protocol adds a negligible extra processing time with the following workstation specifications:

- Virtual Machine: VMware 1.0.1
- OS: Fedora Core 5 with a Linux 2.6.x kernel i686
- CPU: Mobile Intel ® Pentium ® 4 CPU 1.70 GH
- Physical memory 512 MB

Action	Time (ms)	Public Key size
Sending a Request	65,8	-
Processing a Request	100,2	-
Sending Response	2671,8	-
Processing a Response	54,2	-
Encryption Time	927	16
	961	64
	965	128
	968	256
	988	512
Decryption Time	859	16
	866	64
	874	128
	929	256
	955	512
IBE system setup	121	-

Table 3 : measurement values

If we suppose that the system setup step can be avoided (each has a local storage for the system parameter and the private key) the additional extra-time generated by the IBE application is approximately equal to the sum of the encryption / decryption process. This value does not affect the usual sequencing of the WS-Discovery message exchange protocol. This extra time is not problematic due the fact that WS-Discovery protocol uses the UDP messaging format imposing a retransmission delay of 5 seconds in order to ensure the good reception of a message in spite of some packet loss risks. This delay covers the extra time generated by the decryption process.

We also tested a 100% Java solution by integrating the Identity Based Encryption JCE Provider [DUF04] with our Java WS-Discovery Protocol implementation, but this library offers very bad performance processing time that is not compatible with the WS-Discovery message timings.

I. Alternative Solutions

1. Group Encryption

Using attributes based encryption means that for some elements sharing the same attributes may hold the same private keys related to these attributes. In this case one solution offering the same functionalities with improved performance results and management is the group encryption mechanism [KIA07] that allows a sender to encrypt a message and with a restriction for the receivers that would decrypt this message to be a member of the PKI group. As in a group signature, in a group encryption there can be an opening authority that when the

appropriate circumstances are triggered it can reveal the identity of the group member who is the recipient of the ciphertext. A group encryption provides “receiver anonymity” in the same way that a group signature provides “sender anonymity”. This solution could be interesting if the number of attributes of different services is quite reasonable in order to avoid the key management of hundreds of groups. The second limitation is related to the obligation to have an online certification authority and a key generator that is not always the case with the ABE solution. Finally, service privacy could be affected by the possibility for every user to access to the list of available services and this privacy violation does not correspond to our security requirement related to the possibility for a service to hide his existence.

2. Policy Based Cryptography

Another possibility that we can consider to optimize the credentials combination, is the usage of Policy Based Encryption mechanism [BAG05] (PBE). This mechanism is an extension of the ID-based cryptography that allows the encryption of a data according to a policy, and only the entity that fulfils this policy is able to decrypt this data. The advantage of using this mechanism is the possibility to use conjunctions and disjunctions to make an efficient combination of certificates and tokens used for encryption and decryption. Unfortunately, there is no available implementation of this cryptographic scheme that could be deployed in real systems.

J. Conclusion

This chapter discussed specific security and privacy issues of peer-to-peer service discovery mechanisms. A solution based on a particular type of Identity Based encryption called Attributed based Encryption was used to add security during the service discovery process by protecting the user’s requests and restricting the access to the discovery of a service. This solution does not need to rely on a trusted third party in order to perform the matchmaking function. Attribute Based Encryption makes it possible to both encrypt for and describe the semantics of a service, which is essential for accurately yet privately answering a request. Contrary to PKIs, IBE public keys (a string) identify services with human understandable semantics, even though they should be shared by all parties. Using ABE permits combination of different attributes to describe some complex service profiles or service types. The use of ABE may also prove beneficial in more open scenarios in which services are likely to be described using various knowledge representations, like for instance controlled vocabularies or ontologies. We are investigating how to combine reasoning about how service profiles match with such tools together with encryption, even though this may require tradeoffs regarding the privacy of service lookup.

Chapter III. Securing Registry-Based Service Discovery

A. Introduction

In the last section we described a secure solution dedicated to a decentralized discovery architecture. As seen previously in the service discovery introduction section, another kind of architecture can be used to deploys services, and this architecture is centralized (or registry based) and some standardized discovery protocols like UDDI are exclusively deployable on this kind of centralized configuration. With a registry based architecture the attribute based cryptographic solution described previously is not compatible and not adapted because it requires to hide important information needed by the registry in order to perform a matching between clients requests and published services profiles.
Basically the registry in a centralized architecture is required and always available. It also must be trusted by the clients and the servers, in the sense that the matchmaking procedure must be performed correctly, and the information published by the servers must be safely stored and not lost or corrupted. For this reason using a registry could represent an interesting occasion to point out a third trusted party that could be used to establish a trust relationship between different elements of the system.

B. Technical Background

1. XACML

XACML (eXtensible Access Control Markup Language) [XAC] is an OASIS policy standard used to express and perform access control. It includes an XML-like policy language and a query model for access control enforcement and decision making. The query model has a request response format executed between a Policy Decision Point (PDP) and the Policy Enforcement Point (PEP) that issues the request and handle responses (this interaction is described in Figure 29). XACML request consists of a triple {Subject, Resource, Action}. A Subject tends to gain access to a Resource (e.g. file, web service) in order to perform an Action (e.g. read/write, invoke a method). The Subject is characterized by a set of attributes (e.g role, location). Based on this triple {Subject, Resource, Action}, a rule-based access control policy is enforced. After decision making, a XACML response is sent back to the requestor (e.g. Permit, Deny, Intermediate or Not Applicable).

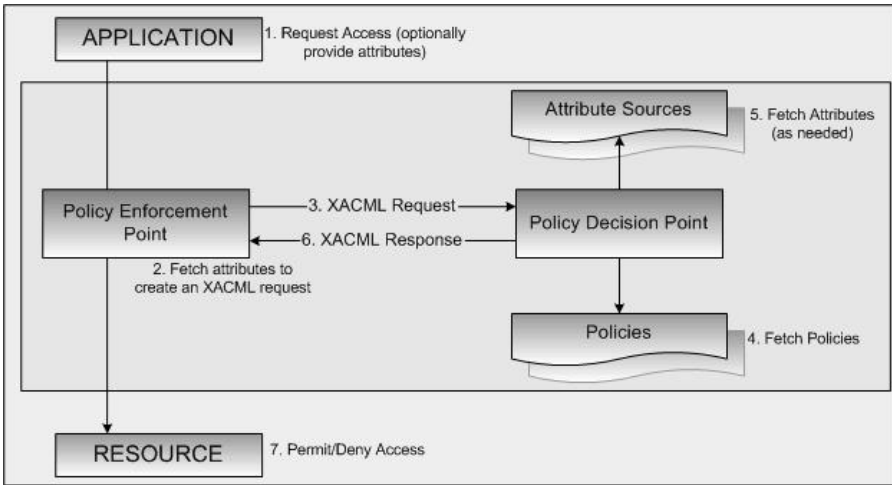


Figure 29 : PDP and PEP interaction in XACML [LOR03]

2. X.509 Attribute Certificate

An attribute certificate (AC) is a structure similar to a public key certificate (PKC); the main difference being that the AC contains no public key, but it may contains attributes describing group membership, role, security clearance, or other authorization information associated with the AC holder. The AC has a similar structure with a PKC; with standards fields like the version, the holder, the issuer, the signature the serial number, the validity and the issuer unique identifier. The additional field is the "attributes" field that gives information about the AC holder and its privileges. This field contains a sequence of attributes, and each attribute contains a set of values.

C. Service Discovery Policy

1. Concept

The security requirement section described previously makes it clear that clients should be able to find a service matching with their preferences, both in terms of service's characteristics and in terms of security and privacy requirements. These security requirements can be imposed respectively by the service and by the client. On the client side, the user should be sure that only services matching his preferences would be returned: from his point of view, trusting a service should therefore go beyond the simple authentication of the service provider and also encompass a complete certification process of the capabilities of the service. On the server side, the problem is quite similar since the server does not know the users that can potentially gain access to its service. They should therefore be accessible only to client they trust to access them according to a precise behavior guaranteed by some authority.

Assigning the responsibility to enforce such discovery policies to a trusted entity of the system is therefore critical to service discovery. To avoid raising the complexity of service discovery, we do not propose to add a new entity to the system together with a dedicated protocol, but rather to assign this task to the registry. The choice of the registry as being the trusted third party in charge of the policy enforcement is an absolute requirement in centralized approaches, since matching already implicitly is a trusted operation and policy enforcement and matching are closely tied together.

Discovery policies may be quite simple: the client or the service provides rules that describe who can access their respective profile based on some attributes. In this paper the discovery policy objective is twofold:

- Access Control: discovery constitutes a preliminary form of access control to services by restricting the clients which will be able to subsequently contact a service. The sensitive resource here is the service's profile that must be hidden to the non authorized users.
- Privacy Protection: the client can protect the private information he reveals for each lookup he performs (identity, intentions, favorite services …) from an uncontrolled disclosure.

As shown in Figure 30 the usual discovery messages (publish and lookup) should be accompanied by some credential (attribute certificate or signed token) in order to be authenticated by the registry, by a discovery policy that will be enforced by the registry in order to protect the entities according to their desires, the whole being secured using a signature based on the credential transmitted for instance.

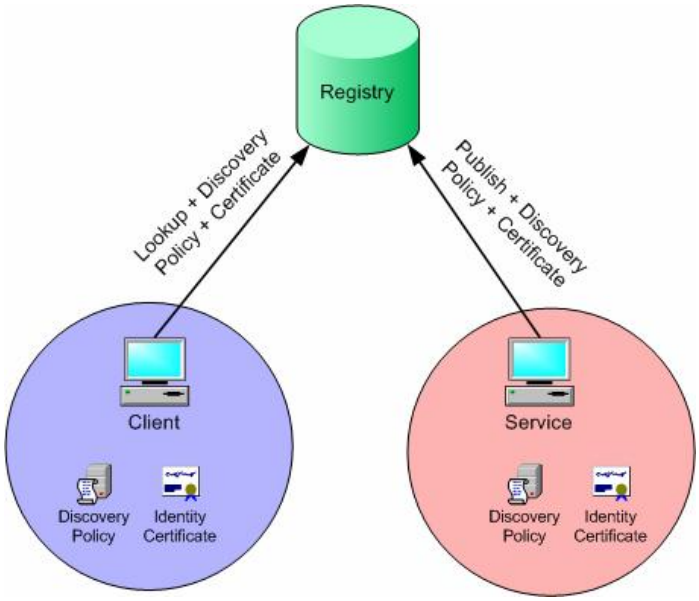


Figure 30 : Communicating Discovery Policy

2. Choosing a Service Discovery Policy

In order to avoid the complicacy of a new security policy language definition, we decided to reuse an existing one that will permit us to express our requirements related to the privacy and access control functionalities. Initially XACML offers a simple XML language traditionally used to express and enforce access control and role based access control (RBAC). It also supports extensions using the condition tags. These extensions specified as meta-data types and functions used to create predicates for conditions. Some of these functions are proposed in the XACML specification; like equality, arithmetic, string conversion, numeric data-type conversion, logical, numeric comparison, date and time arithmetic, non metric comparison, string, bag, regular expression, special match, and XPath functions. The programmer can also integrate new personal functions that refer to functions written in other programming languages. This wide expressiveness supported by XACML permit to extend the basic policy language and adapt it with the security requirements related to the service discovery (privacy, authentication, authorization, access control, recommendation).Using the XACML as a service discovery policy language, we can easily restrict the discovery of a service to some authorized clients. For example, we can allow the discovery of a Color printer service to users that have the role of *professor* in the *Eurecom* domain (see Figure 31).

```
<Policy PolicyId="Policy" RuleCombiningAlgId="permit-overrides">
   <Target>…</Target>
   <!--  Rule of Color Printer Discovery Action   -->
   <Rule RuleId="rgetPatient" Effect="Permit">
   <Target>
      <Subjects>…</Subjects>
      <Resources>Color Printer</Resources>
      <Actions>
         <Action>
         <ActionMatch MatchId="function:anyURI-equal">
         <AttributeValue   DataType="anyURI">Discover</AttributeValue>
         <ActionAttributeDesignator DataType=anyURI  AttributeId="action-id" />
         </ActionMatch>
         </Action>
      </Actions>
</Target>
   <!--  Check if the subject is a Professor from Eurecom   -->
   <Condition FunctionId="function:string-equal">
      <Apply FunctionId="function:string-one-and-only">
         <SubjectAttributeDesignator DataType="string"   AttributeId="SubjectRole"
/>
      </Apply>
      <AttributeValue DataType="string">Professor</AttributeValue>
       <AttributeValue DataType="string">Eurecom</AttributeValue>
      </Condition>
   </Rule>
</Policy>
```

Figure 31 : Discovery Policy sample using XACML

D. Architecture for a Registry-Based Secure Service Discovery

In this section we describe the different steps executed during a secure registry-based service discovery (see Figure 32). We also analyze and explain the working behavior of the system and how the different elements of the architecture can interact with each others. As we mentioned previously our system relies on a trusted registry that represents a secure intermediate between clients and services. In this case we suppose that clients and servers have an a priory knowledge about the registry information (location, address, public key …) in order to locate the appropriate trusted one and establish a secure interaction.

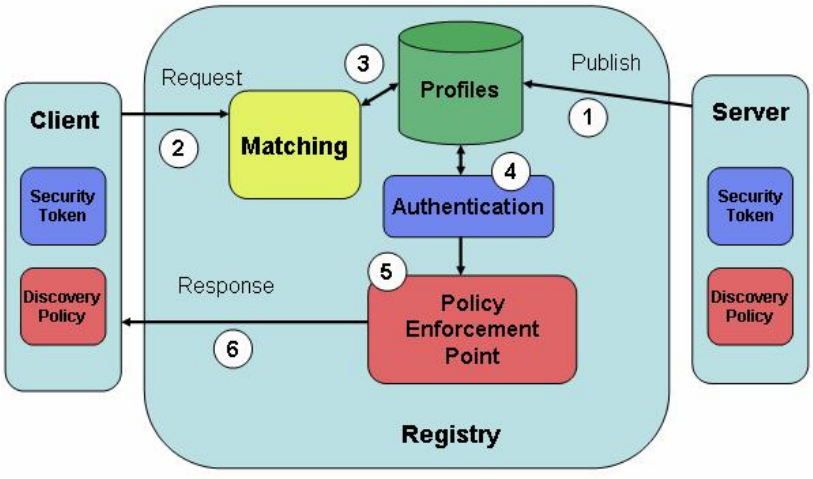


Figure 32 : Secure Registry-Based Architecture

- **Setup**: Client and services have to contact their default registry and establish an encrypted secure channel (using SSL/TLS) with the help of the Public Key of the registry. After establishing this communication channel the discovery process can safely begin. All the message exchange must be encrypted.

- **Step (1)** is initiated by the server in order to register its services by sending a publish message containing the description of its capabilities, its profile and some specific contextual information. The service has the possibility to add a discovery security policy that must be enforced by the registry in order to guarantee the privacy and the confidentiality of the proposed services. In order to be authenticated by the registry (to satisfy client's discovery policy) the server has the possibility to add a security tokens (like Attribute Certificate) during the registration process, this token could contain some certified information about the service (The owner, the company, the domain …). All the information provided by the servers is stored in the registry's database (Profiles).
- **Step (2)** is the Client's service lookup by sending a request message containing the service request. This request contains the attributes and the particularities of the wanted service. Like for the server, the client has the possibility to add its own security policy restricting the scope of the discovery to some specific services. The client could also add some information about its identity or its current context attribute values. Concerning the identity of the client it would be preferable to use a security token in order to be authenticated by the registry.
- The most important part of the secure service discovery process is the request matching and the policy enforcement. **The step (3)** performed by the registry consists in a request matching with the existing profiles contained in the profile database. If the query matches with one or more services, the registry verify if the there are policies related to the selected services or/and to the client's request. If these policies exist the registry starts with authenticating the client or the service **(4)** by checking the validity of the provided Attribute Certificates and extracting the values of these attributes (identity, role, domain, Qos …etc) in order to generate a XACML query related to these attributes. This query will be compared with the policy and the decision will be taken in order to give a response to the client **(5)**. If the request is accepted, the registry returns a response to the client by sending a response message containing the necessary information to joint the selected service **(6)**.

E. Algorithm for a Secure Centralized Service Discovery

The most important task of the system is assigned to the central registry that has to match the query of the client with the service profiles published locally, then verify the existence of the policies, authenticate the evolved parties, enforce the policies and finally send back a request. All these actions are described in the algorithm *Request_Process* in Figure 33.

Algorithm 3: Request_Process (*Client*)

```
0.  Begin
1.   | match client query with published service profiles;
2.   | while list of matched services ≠ ø then
3.   | |    get the next service recod of the list;
4.   | |    if Client and Service provide policies then
5.   | |     | Req_C ←build a Xacml request with Client properties;
6.   | |     | Req_S ←build a Xacml request with Service properties;
7.   | |     | match Req_C with Service policy;
8.   | |     | match Req_S with Client policy;
9.   | |     | if policies are satisfied then
10.  | |     | | send back a response to the client;
11.  | |     | endif
12.  | |    elseif only Service provides policy then
13.  | |     | Req_C ←build a Xacml request with Client properties;
```

```
14. | |     | match Req_C with Service policy;
15. | |     | if Service policy is satisfied then
16. | |     | | send back a response to the client;
17. | |     | endif
18. | |    elseif only Client provides policy then
19. | |     | Req_S ←build a Xacml request with Service properties;
20. | |     |  match Req_S with Client policy;
21. | |     | if Client policy is satisfied then
22. | |     | | send back a response to the client;
23. | |     | endif
24. | |    else
25. | |     | send back a response to the client;
26. | |    endif
27. | endwhile
28. End
```

Figure 33 : Request Algorithm

F. Secure Service Discovery Middleware

Discovery mechanisms are usually used to explore the environment and to adapt to the different interfaces provided to connect to relevant services. The definition of specific middleware is necessary in order to provide standardized interfaces facilitating the interconnection of mobile devices and services. Various solutions like J2EE, .Net, or CORBA have been developed providing a middleware abstraction over which one can deploy standardized discovery protocols such as WS-Discovery, UPnP, Jini, or SLP. However, these middleware have never specifically addressed security problems in the spontaneous networking framework, which usually rely on too static assumptions that hamper the dynamic deployment of services. In particular, an administrator has to know in advance users and services of the system in order to ensure their protection. In this section we aim to provide a flexible and adaptive middleware abstraction layer inspired from the architecture described in the section Chapter III.D that programmers might easily integrate via a plain Java API or through a Web Service interface.

1. Related Work

Several studies were published in the literature that proposes different middleware platforms to enable an efficient service discovery. This section provides an overview of various such platforms. To our knowledge however, no actual implementation of a security oriented service discovery middleware platform answering spontaneous networking challenges was ever described before.

[BIS06] introduces a service discovery middleware framework for an ad-hoc network of devices. This architecture relies on a hierarchy of distributed registries called “Service Repository”. In order to verify if all the published services are still “alive”, every registry polls the rest of the network and requests the other registries to report their local services. The authors compared the M2MI framework to Jini’s service discovery that is quite similar in its functionalities.

[SOM07] aims at answering the challenges of the service discovery in partially connected or disconnected mobile ad-hoc networks. It first lists the issues that must be addressed in order to overcome the problems related to mobility and to the lack of permanent connectivity (flexible addressing scheme, supporting asynchronous communications, content based management of messages, and taking into account special and temporal contextual properties of the services. The core framework of this proposal is based on the OSGi framework which the authors extended in order to describe local and remote services and to enable a proactive and reactive service discovery. Contextual properties are added to characterize mobile hosts.

With the emergence of pervasive computing environments, [CAM05] proposes a pervasive discovery policy associated with a service description language (GSDL). In terms of new challenges raised by discovery in such a programming paradigm, this paper only focuses on network transmissions minimization, on the decentralization of the discovery infrastructure, and on cooperation between different systems. The challenges described concerning service description are the need for simplicity to adapt to the limited power and computing capacity of the devices, the need for scalability, and the need to retain compatibility with all kinds of platforms. The discovery algorithms presented prioritize between existing services by warding the more static devices with more opportunities of answering requests. The GSDL description language, which can be considered as an extension of the WSDL language, provides a hierarchical description of the services for specifying the relationship between different services. This hierarchical categorization provides more scalability for service publication.
The security middleware solution from which we inspired most has been published in [JIA05] and addresses ubiquitous computing environments. The middleware provides a way for the users and the administrators to define security policies for context-aware trust management. Such a security policy was implemented by the authors. They define three kinds of policies: an authorization policy (access control policy), a delegation policy (that allows user to delegate his rights to another user), and an obligation policy that triggers actions in response to specific events in the system. The middleware consists of five components: a policy manager, used for the policy reasoning and enforcement; an object manager that provides a monitor control of the different objects involved in the system; a context manager, which collects and analyzes the contextual information from the sensors; finally an authentication manager, used to verify and manage the X.509 certificates provided by the users and the artifacts.

2. Middleware Stack

The heart of the platform is a middleware including a set of security services, notably for automatic SOAP traffic encryption and signature. Finally, applications are either written on top of the middleware or, in case of legacy applications, are adapted to use this middleware. Application-specific policies can be defined to configure the platform. The secure discovery middleware is intended to provide a secure and adaptable service discovery interface. This system could be used by service providers in order to securely publish their services by imposing their own security requirements. Users also have a way to discover and locate services without exposing their identities or attributes to unbeknownst and potentially untrusted entities, and of requesting that the services they are discovering adhere to specific security requirements.
The discovery service is accessible to client applications through a Web Services interface. This API gives access to records in a service registry that can be implemented in a centralized or distributed fashion. The current implementation supports the peer-to-peer WS-Discovery protocol that is nicely suited to spontaneous networking, even though some provision has been made in the middleware to integrate other types of directories, most notably UDDI based ones. This API is also accessible through Java methods. Using this interface, a server can:

- declare its services by publishing their profile;
- upload a security discovery policy for each of these services to the registry;
- upload certified credentials to the registry that will make it possible for the latter to authenticate the former.

Clients also use this interface in order to request for a specific service, described according to an extensible set of attributes. They can also upload a security discovery policy and certified credentials for authentication.

As can be readily seen from the above description, the registry is the decision (PDP) and enforcement point (PEP) for the discovery policy, two functions which both rely on the use of certified credentials. As a decision point however, the registry also relies on context evaluation, as described below.

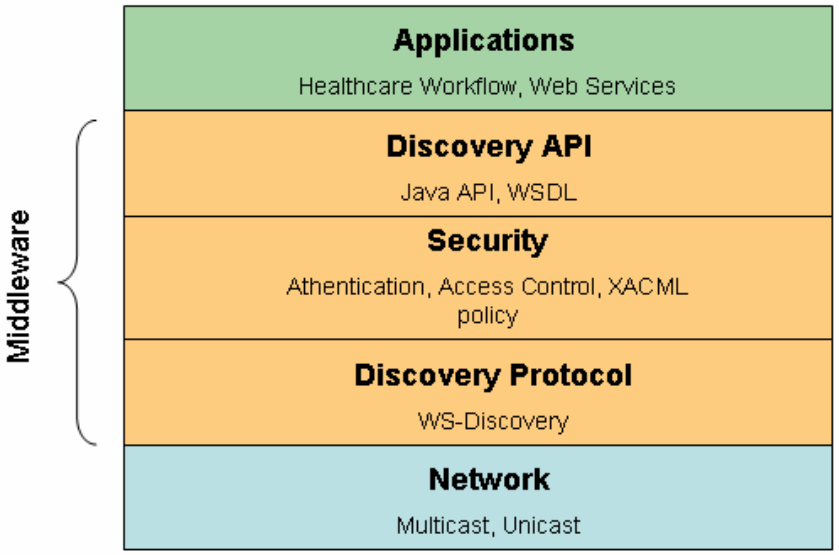


Figure 34 : Middleware Stack

The middleware stack shown in Figure 34 is composed of three layers:

- **Discovery Protocol Layer**: This layer specifies the message formats, the message sequencing, and the processing rules. Discovery follows the WS-Discovery protocol, which was slightly extended to accommodate an explicitly designated registry based on WS-Discovery proxies. Packets are routed via Multicast in case of a public announcement (messages from the service to groups of registries) or Unicast in case of a direct request (messages from the client to a registry). UDP datagrams are sufficient to guarantee a coherent discovery connection without setting up a TCP connection. A UDP message must be sent at least 3 times to guarantee its reception from the concerned party. Other discovery protocols dedicated to service discovery like UPnP, Jini, or SLP might be integrated if legacy systems are to be used.
- **Security Layer**: This part of the stack provides tools and methods used to: authenticate clients and services (identity and wrights verification), decide and enforce the security policy provided by clients and services, and finally perform an access control to restricted resources.
- **API Layer**: It represents the visible interface accessible to the user. It provides the information helpful for the users in order to access to the service discovery Middleware. Two interfaces, a Java library, and a WSDL description are provided (see the Annex).

G. Security Evaluation

In this chapter we proposed a secure service discovery protocol relying on a centralized trusted registry in charge of enforcing security discovery policies. In this section we present the secure properties of this protocol.

Property 1 it is impossible for a non authorized client to discover restricted services

Proof- Restricted services provide to the registry, during the registration, a security policy limiting the discovery of the services to clients that profile corresponds to the conditions expressed in the policy. If a client's request matches with a restricted service, the registry will authenticate the client, extract the attributes of the client and match it with the attributes specified in the service's policy. If the policy is not satisfied the registry will not respond to the client in order to specify that the service does not exist.

Property 2 it could impossible for a fake service to be discovered by clients

Proof- Client could attach to their service request a security policy to verify the authenticity of the selected services. If this request matches with a service the registry first authenticate the selected service and verifies if it is compliant with the conditions expressed by the client's policy. For example the matched service is a *printer* service the registry has to verify if the *printer* is one of the attributes of service's certificate, else no response will be sent to the client.

Property 3 communication between client / service and registries are secure

Proof- since SSL/TLS is systematically used between clients, service and registries, discovery messages are exchanged through a secure channel. The number of exchanged messages is not enough important to permit an attacker to break the encryption system.

H. Measurement Results

In order to evaluate the efficiency of our solution we extended the Java prototype of the WS-Discovery protocol with the XACML functionalities, and then we performed some measurements about time execution and memory consumption. For these experiments we used:

- OS: Fedora Core 5 with a Linux 2.6.x kernel i686
- CPU: Mobile Intel Pentium 4 CPU 1.70 GH
- Physical Memory 512 MB

In this table we provide all the measurement values related to each execution steps of the context aware policy based service discovery.

Actions	Time (ms)	Size (byte)
Sending Hello (Publish)	31	3963
Sending Probe (Request)	67	862
Service matching	370	-
Authentication	1572	-
Policy enforcement	862	-
Sending ProbeMatch (Response)	15	1622

Table 4 : measurement values

I. Conclusion

In this chapter we propose a policy-based solution to secure registry-based service discovery. For this solution we choose to take advantages of the status of central entity represented by the registry in a centralized architecture in order to cope with the identified threats in chapter Chapter I.G. The registry after being authenticated by clients and services plays the role of trusted discovery policy reasoning and enforcement point in charge of authenticating the elements that must be authenticated according to published policies. As proof of concept we implemented a prototype securing WS-Discovery protocol and relying on a discovery policy built as an extension of XACML access control policy. Our approach solves user's privacy and service access control by introducing a new security requirement expressiveness dedicated to the service discovery that efficiently supports trust establishment between different actors of the system.

Chapter IV. Secure Service Discovery with Distributed Registries

A. Introduction

The solutions described in sections Chapter II and Chapter III clearly do not scale to an internet wide service discovery in terms of lookup scope and protocol security; in a centralized architecture or in P2P one described previously, if the service does not exist locally, the client will stop sending its binding message or retry the binding process later. With a distributed policy-based architecture on the contrary, if the service is not found locally, the local registry can forward the query to other registries belonging to other domains and networks in order to extent the search scope. The success probability of a query is better with this distributed system, but it introduces more latency for the response. After a timeout the client may decide to drop responses from distant proxies. With the distributed solution we can avoid bottlenecks on the service side. In fact, with the decentralized solution, when a client multicasts an encrypted probe message, all the servers that are listening to the multicast channel will try to decrypt the message at the same time. During the decryption period if another client sends another request message, it could be dropped or cached until the end of the previous message decryption. This phenomenon generates a bottleneck on the service side that could be avoided with the registry-based solution. This gain of performance costs more complexity in the discovery protocol construction. With idea of distributed registries the secure service discovery is extended to other LANs and solves the bottleneck problem created with the decentralized solution. The essential challenge is to provide a scalable discovery system with respect to the security requirements defined at the beginning of this thesis.

B. Related Work

A distributed architecture for service discovery was proposed by Chakraborty et al. [CHA06] that aims to provide; first a local service discovery in P2P manner relying on advertisement between local peers, second a distributed group information system that enable an intra domain and scalable service discovery. In this solution services are categorized in hierarchical groups according to their capabilities. Each node of the system is in charge of one group. In case of mismatch with a client request (sent locally), the node's query is forwarded to the correct node in charge of the group to which the requested service belongs. Despite the claim of the authors concerning scalability, each node has to memorize the whole group hierarchy of

the system in order to route the request to the correct node, and this assumption is not realistic for a system involving millions of nodes deployed all over the world. The security aspects related to service discovery is out of scope of this study.
To our knowledge, the first study dealing about security and particularly privacy for distributed and scalable discovery architectures is proposed by Cardoso et al. [CAR07]. This solution is an extension of the MUSDAC [RAV06] middleware platform that enables an intra-domain and intra-network interaction between different service discovery protocols. In each local network a MUSDAC manager is deployed on top of the existing discovery protocols and provides a flexible interface to handle discovery and access requests of all the elements present in the network. This manager is connected to a bridge in order to expand the service discovery scope to other networks and domains. Extending the discovery scope to other domains raises new privacy issues regarding the dissemination of discovery information may contain private data that should be protected from the access of non trusted entities. Authors address this issue by proposing a trust model regulating the execution of the discovery process. Depending on the trust degree expressed by a user towards a foreign domain four security mechanisms can be gradually used to protect the service request and access.

C. Distributed Architectures for Service Discovery

The main motivation of this paper is to achieve scalable and distributed service discovery. Most of the scalability studies in the domain suggested the usage of distributed registries. Registry distribution must rely on a specific deployment architecture and specific indexation and routing mechanisms. Three architectures can be used to deploy distributed registries (see Figure 35):

- Flat: all the registries are interconnected and can communicate through broadcast or multicast. There is no specific indexation/retrieval strategy; in case of new service request addressed to one of the registries registry that does not store the relevant information about the requested service, this one will forward the request to the other registries in order to find a matching to the query. Such an architecture might work for a small number of registries (less than 100) but it becomes inefficient for a huge number of registries where the anarchic indexation/retrieval strategy generates an overhead in the network and a latency message delivery.
- Hierarchical: the deployment strategy of the registries corresponds to an n-ary or binary tree in which data is indexed following to a data structure distributed hierarchically through tree nodes. Indexation and retrieval operation will cost at the worst case log(n) operations. In terms of scalability and indexation performance this architecture overcomes the limitations of the flat architecture. However, in case of a registry failure or shutdown the recovery process can be very costly and requires a replication systems and an important signaling procedure. This could affect the service discovery availability and could lose some important data.
- P2P using DHT: an alternative architecture currently used in file sharing applications relies on Distributed Hash Tables (DHTs) for indexation and retrieval. Each peer in the system (a peer is a registry in our case) is in charge of indexing and maintaining the mapping from names to values. This indexation is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to an extremely large number of nodes while handling continual node arrivals, departures, and failures. This architecture enables to handle thousands of registries with millions of data entries.

DHT based systems are clearly the most appropriate solution for deploying a scalable and robust service discovery system and the rest of this paper discusses which security measures can be adapted to their use.

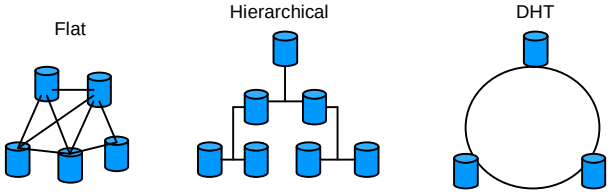


Figure 35 : Alternative architectures for distributed registries

D. Technical Background

1. Onion Routing

Onion routing [GOL96] is a scheme for anonymous communication in which users can communicate while hiding their identities from third parties. This approach is called Onion Routing, because it relies upon a layered object to direct the construction of an anonymous, bidirectional, real-time virtual circuit between two communicating parties, an initiator and responder. Onion Routing hides routing information through the routing of an encrypted data stream follow a path through intermediary nodes until the destination. To begin a session between an initiator and a responder, the initiator node identifies a series of routing nodes forming a path to the destination. The initiator constructs an onion message which encapsulates that path. Figure 36 illustrates an onion constructed by the initiator Node W for an anonymous route to the receiver Node Z through intermediate routing nodes X and Y. The initiator then sends the onion along that route to establish a virtual circuit between himself and the receiver Z.

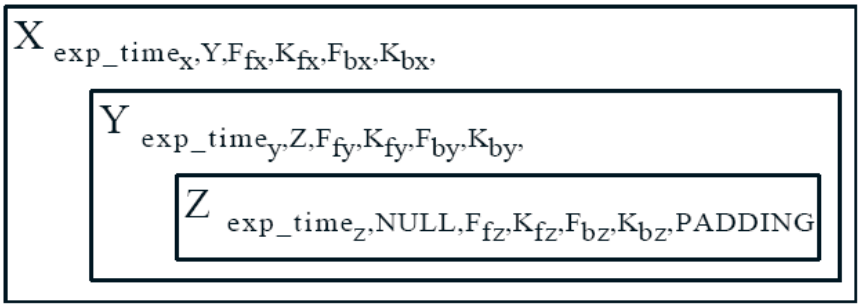


Figure 36 : Onion Message Format [GOL96]

The onion message structure is composed of a superposition of encrypted layers. The core of this onion contains the clear message to send. The basic structure of the onion is based on the route to the receiver that is chosen by the initial sender. Based on this route, the initiator encrypts first for the receiver, then for the preceding node on the route, and so on back to the first routing node to whom he will send the onion. When the onion is received, each node knows who sent him the onion and to whom he should pass the onion. But, he knows nothing about the other nodes, nor about how many there are in the chain or his place in it. The virtual circuit established between the node W and the node Z is described in the Figure 37.
The most famous anonymity software using this technique is TOR[6] (The Onion Router) that is originally sponsored by the US Naval Research Laboratory that actually becomes an open source project.

[6] http://www.torproject.org/

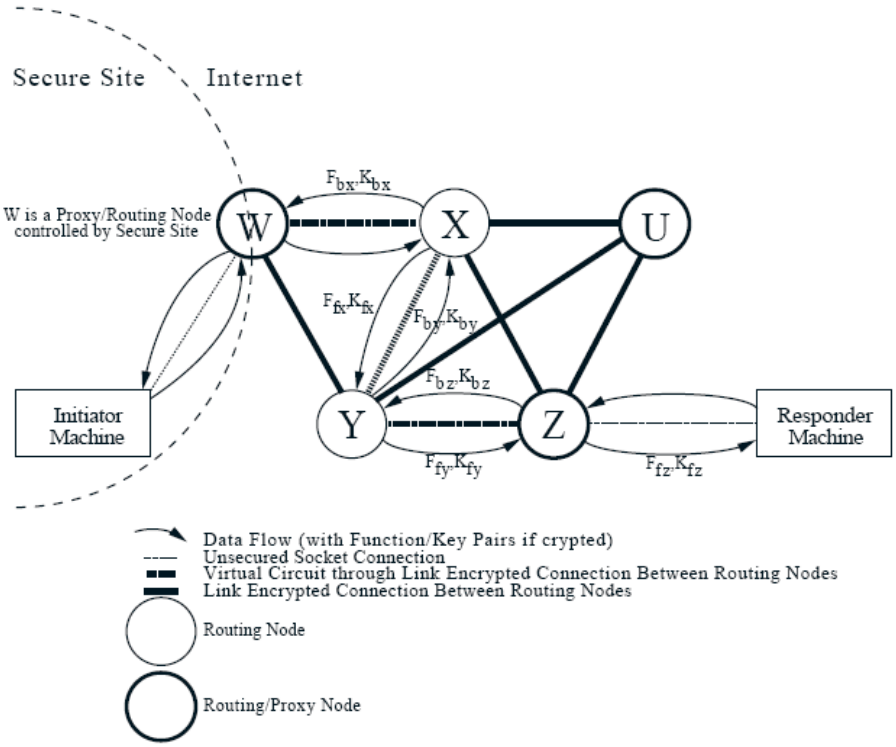


Figure 37 : Onion Routing Virtual Circuit [GOL96]

2. Distributed Hash Tables (DHT)

The distributed hash table (DHT) is an indexing and location system dedicated to P2P information storage. This distributed system provides the possibility for a user (node) to efficiently retrieve the value associated with a given name. DHT-based indexing systems provide interesting properties related to the scalability because it should efficiently work for millions of users and data. The behavior of the system is strongly dependent on the collaboration of nodes; each node with a fixed identifier is responsible for a range of key values representing a pointer to a stored element. Each stored element has an index value represented by a hash key, the whole key range are represented in a keyspace. The entire keyspace is partitioned among the participant nodes; each node is in charge of a partition. The whole nodes identifiers are virtually represented as a circle in which each node has a specific place. This virtual circle is used to facilitate the node position lookup. If a new element is added to the system, the name of this data is hashed, and then depending on the hash key value, the pointer to the data will be affected to the nodes in charge of the correspondent key range. In order to retrieve this data, a hash key is generated from the requested name, and depending on the value of the obtained key we can get an indication about the nodes in charge of maintaining the information related to the stored data, then the request is routed to the correspondent node that will give back the pointer to reach the queried element.

Figure 38 shows an example of an indexing circle containing 4 nodes sharing the entire keyspace. A new key injected to the system is assigned to the first node whose identifier is equal to or follows the hash value of the key. The key 2 is assigned to the node 6 and the key 9 to the node 12. If a request is concerning a key value 16 the request is routed to the node 20 that should holds the pointer to the requested element.

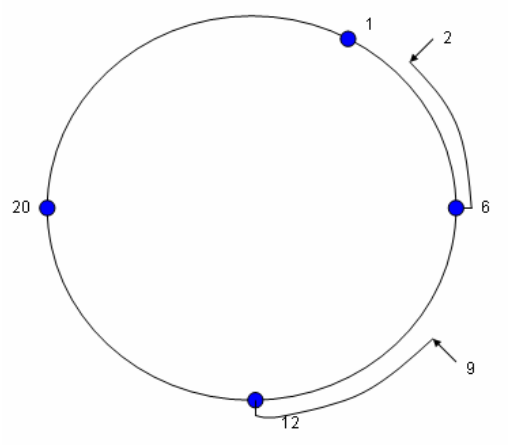


Figure 38 :DHT indexation

DHT based systems provide many interesting characteristics that can be exploited in service discovery mechanisms:

- Decentralization : many autonomous clients without a central control
- Scalability : the system has to adapt large extensions of peers
- Fault tolerance : the network must be resilient, especially in regard to stale peers
- Load balancing : routing messages have to be balanced to reduce the network overhead

E. Requirements

Scaling the discovery system does not mean scarifying its security. In the centralized registry based solution described in chapter Chapter III, as the registry (trusted third party) remains local it is in charge of establishing a secure channel between clients and servers by authenticating then establishing a trust relationship between different actors of the system. As soon as the discovery system deals with multiple distributed registries it becomes to be a hard task to establish a trust relationship with all these registries. For theses reasons the distributed discovery system must fulfill the following requirements:

- Scalability: no limitation related to the network size and the elements number.
- Efficient intra domain lookup: no restrictions for the request scope
- Efficient matching and indexation: avoiding collisions and bad matching performance values
- Privacy protection:
  - Protecting requests coming from clients: nobody must know who is looking for what
  - Protecting service publishing: restricted services must be protected against unauthorized discovery
- Authentication: implicit or explicit authentication for the trust establishment
- Access Control: only authorized entities must be able to discover restricted resources (requests and publications).

F. A Scalable Distributed Registry-Based Model

1. Indexing and Data Retrieval

The principal motivation of this solution is to make the service scalable and distributed. For this reason we selected the most scalable and the most reliable technology actually deployed

for the distributed indexation: the P2P indexation and retrieval system based on DHTs called Kademlia [MAY02] that provides a P2P storage and lookup protocol. This protocol is installed as an external interface for all the registries involved in the discovery system and scattered all over the world. Using this interface, active registries can permanently update information about the stored data like every P2P client using a file sharing system. In case of a new service publication, the registry stores the new entry locally. It then hashes the description name of the new entry, and finally sends the key (with a pointer to the service entry) to the appropriate registry in the indexing circle. If a client sends a service discovery request to the local registry, this one tries to find the service entry locally otherwise it hashes the query contained in the request and forward the request to the registry in charge of the hashed key value that points to the final registry holding the appropriate information.
For this purpose we reuse the same message format used in Kademlia:

- STORE: To publish a <hkey, value> pair, the registry locates the k closest nodes to the key and sends them STORE RPC messages.
- FIND_NODE: To retrieve the node in charge of the hkey corresponding to a published service the requesting registry sends a FIND_NODE message containing a triples (IP address, port, nodeID) for the contacts that it knows to be closest to the key.
- FIND_VALUE: If a corresponding value is present on the recipient node, the associated data is returned. Otherwise the RPC is equivalent to a FIND_NODE and a set of k triples is returned.

2. Algorithms for inter-registry Indexing and Data Retrieval

- **Inserting a new node**: Each registry/node maintains a routing table useful to locate other registries/nodes of the indexation circle. If a new node joined the system these routing tables must be updated. In practice the routing table used by a node does not point to all the nodes of the system (millions of nodes) but just to a few nodes representing a routing zone (similar to traditional routing tables for network routers). If a new node joined one of these routing zones, his positions must be added to the routing tables of the other nodes. The algorithm describing a new node insertion is described in Figure 39.

Algorithm 4: Insert Node()

```
0. Begin
1.  | if contact exists then
2.  |  | update contact information (IP address, keys, pointers);
3.  | else
4.  |  | create new entry to the table;
5.  |  | insert the node contact information (IP address, keys, pointers);
6.  | endif
7.  End
```

Figure 39 : Algorithm for node insertion

- **Publishing a new entry:** The registry receiving a new service publication will forward this new entry to the appropriate registry in the indexing circle. If the entry already exists the information will be updated. The algorithm describing a new key insertion is described in Figure 40.

Algorithm 5: Adding new entry (*key*)

```
0. Begin
1.  | if keyword had already sources then
2.  |  | create a new pointer for the existing keyword
3.  | else
4.  |  | Add new <keyword, pointer> to the local record
5.  | endif
6.  End
```

Figure 40 : Algorithm for Key insertion

G. Securing the Access to Distributed Registries

Now the indexation and retrieval part of the discovery system is scalable, the next objective concerns the protection of the users against potential threats that could affect publish/request steps of the discovery.

1. Need for Anonymity

In the chapter Chapter II and Chapter III discovery mechanisms are relying on cryptography, security policies dedicated to the discovery, and trusted registries. These security mechanisms are preventing against eventual threats and attacks but they are limited to a local application (LANs, PANs, home networks). Reusing the concept of trusted registries with thousands of registries across the world introduces management limitations regarding to discovery policies enforcement. Using Attribute based encryption to hide the content of exchanged messages makes it impossible for a non trusted registry to match between clients requests and services profiles. In [TRA07] we proposed a first solution that relies on attribute based encryption applied on messages exchanged between registries although with keeping a part of the messages in clear to enable matchmaking. In this solution it is possible for an attacker that analyses the traffic to get information about the client's intentions (who is looking for what) and also to retrace the addresses of the node publishing services. Globally this solution is scalable enough, but not sufficiently secure.

For this reason, we decided to improve this solution by adding anonymity for the senders in order to anonymously reach the registry without hiding matching elements. Numerous anonymity techniques exist to protect the anonymity of the senders. The easiest technique is the proxy-based solution, in which proxies are placed between the endpoint users and the rest of the networks in order to relay all the traffic issued by the users without showing the original address of the initiator. This technique requires the deployment of one proxy per user and does not protect against local traffic analysis that could be used to identify the initiator address. To overcome these limitations, another concept appeared called the "mix technique" that consists in creating a non direct path between the sender and the receiver in which a number of relays will exchange the initial message and each relay hides the information about the previous relay. With this configuration each node only knows the previous and the next relay, and the final receiver will not be able to retrace the route of the message in order to identify the sender. Variants of these concepts are Onion Routing (described previously), Chaum Mixes [CHA81], Web-Mixes [BER01], SG Mixes [KES98] and Crowds [REI98]. All of these solutions provide a scalable and efficient mechanism for anonymising a forward path between a sender and a receiver, but there are some limitations concerning the backward path for which these systems do not provide any particular protection. Kate et al. [KAT07], [KAT08] fixed this problem with their new version of the onion routing protocol called "pairing-based onion routing" in which they rely on pseudonyms to identify the nodes involved in order to facilitate a two way anonymous path construction. This protocol is described in the next

section. We chose to integrate this anonymity mechanism for a secure service discovery system relying on DHT-based registries.

2. Pairing-Based Onion Routing

This section describes how the pairing-based onion routing protocol works.

- **Pseudonyms and key agreements**: in order to protect the anonymity of users involved in the system, each node chooses for itself a pool of pseudonyms for which they will generate private keys. These pseudonyms will be announced to the other nodes of the systems. When a node A wants to contact another one B, A will use the pseudonym of B as a session key K_{BA} to encrypt the secure forward message. At the same time B using the pseudonym of A and his own private key to build the backward session key K_{BA} used to secure the backward path.
- **Circuit construction**: before staring the contribution to the routing system, a user has to create a set of routes (information provided by directory servers providing a list of available routes). After choosing a circuit, the user has to generate the appropriate session keys related to pseudonym of each node involved in the system in order to encrypt the onion message. If one of the involved nodes receives the message, he will decrypt his onion layer with his private key, then derives the backward session key, and forward the message to the next pseudonym. Figure 41 illustrates an example of the circuit construction.
-

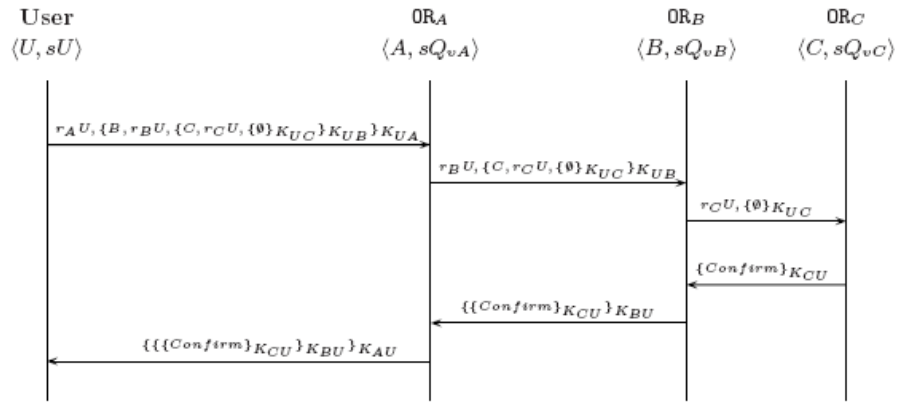


Figure 41 : Pairing-based onion routing circuit [KAT07]

3. Anonymizing Publish / Request Messages for the Service Discovery

In this section we describe in details the mechanisms used to protect service discovery message exchange with a distributed DHT-based registries that we propose.

- **Protecting the publish message**: let us assume that a server wants to publish a set of restricted services in some untrusted registries. First, this server will encrypt all the data related to the identity, the location and the methods provided in his services. In this case only encryption can hide this kind of information. An ABE encryption can be applied to this part of the publish message taking as an encryption key argument the profiles of the users that are allowed to decrypt the message and to discover the services. A part of the publish message must remain clear in order to enable an easy matching for the registry. In order to be authenticated by the client, the server has to sign the publish messages of his services using the private keys related to his service profiles. An illustration of a partially encrypted publish (WS-Discovery Hello

message) message restricted to users with role {professor} can be built as described in Figure 42.

```
<s:Envelope>
  <s:Header>Encrypt[Header]{Professor}
  </s:Header>
  <s:Body> <d:Hello>
    <a:EndpointReference>
 Encrypt[EndpointReference]{Professor}
    </a:EndpointReference>
    <d:Types>Printer</d:Types>
    <d:Scopes>University</d:Scopes>
    <d:XAddrs>
     Encrypt[XAddrs]{Professor}
    </d:XAddrs>
</d:Hello></s:Body></s:Envelope>
```

Figure 42 : Publish message structure

- **Anonymous publishing of services**: After securing the publish message the server has to send it to the local registry without giving information about the identity and the endpoint address of the services that could be deduced by correlating the service description (clear text in the message) and the address from which the message is sent. For this reason the server will select an onion routing path using the pseudonyms of the others nodes belonging to the same registry. This anonymous path is used to forward the message to the registry without any possibility for the registry to guess about the server address and the location of the restricted services. The anonym reverse path is used to send back a publication acknowledgement containing a unique identifier that could be used by the server in order to update his services or deregister it. Figure 43 details the message sequence to perform an anonymous service publication using $i > 3$ intermediary onion routing nodes.

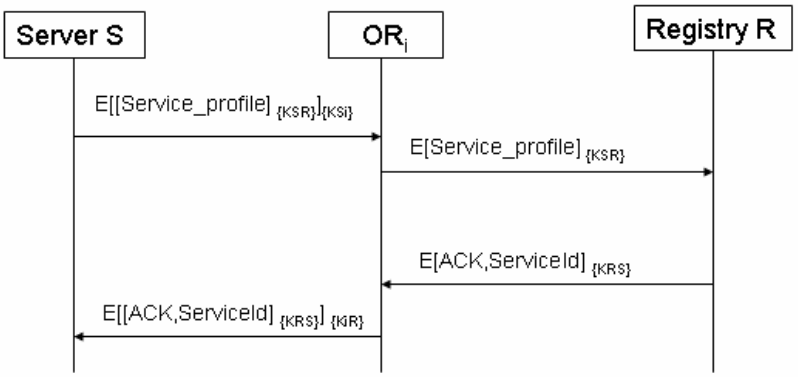


Figure 43 :Anonymizing Publish Messages

- **Protecting request message**: in order to authenticate the published services and verify the authenticity of the published services, a user can verify the ABE signature of a selected service by using the description attributes of the service as a key for signature verification. For this reason, the request message does not need to be encrypted. Only correlation between the requested service profile and the user's address must be prevented by making the request anonymous.
- **Anonymous service request**: the same anonymity method is used for the request and the publish actions. Before contacting the local registry, the request message must be routed through an onion mix to prevent any attempt of correlation between the

requested service and the requester identity. In this case the user has to choose a path to the registry according to the pairing-based onion routing protocol.

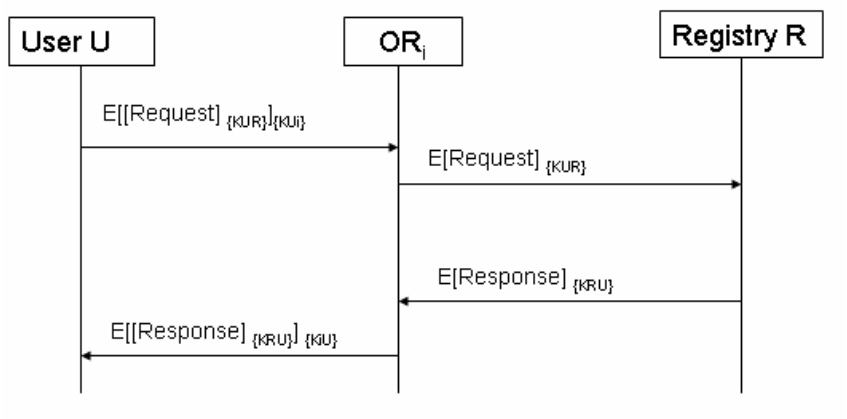


Figure 44 : Anonymising Request Messages

H. Architecture for a Secure Distributed Registry-Based Service Discovery

In this section we describe the different steps executed during a secure service discovery relying on untrusted distributed registries (see Figure 45). We also analyze and explain the behavior of the system and how the different elements of the architecture can interact with each others. As we explained previously our system relies on an important number of untrusted registries distributed all over world wide network. These registries are communicating through Kademlia for an optimal indexation and retrieval of services. Clients and services have to anonymize their requests before accessing to their local registry in charge of publishing and retrieving services. In this case we suppose that clients and servers have a prior knowledge about the location of the local registry.

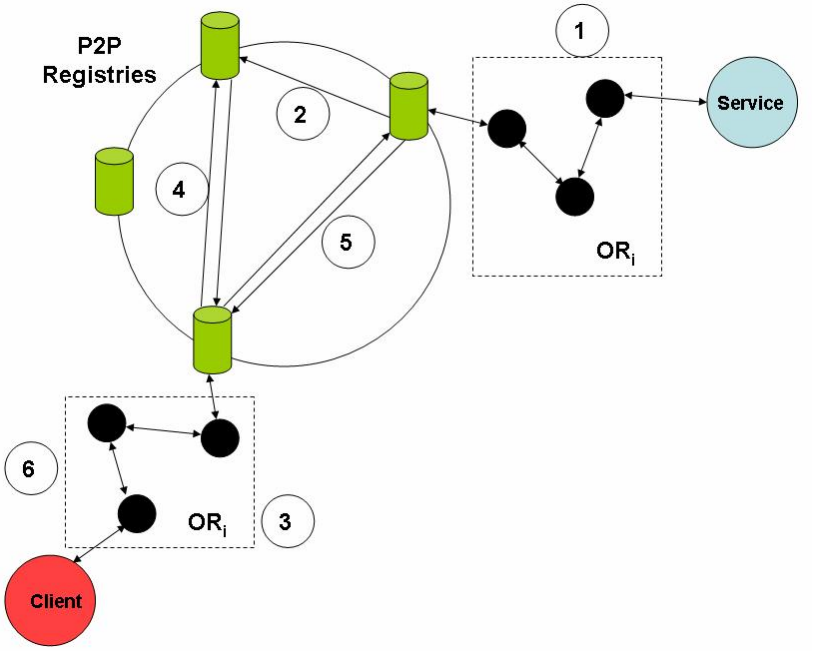


Figure 45 : Architecture for a Secure Distributed Registry-Based Service Discovery

- **Setup**: Clients and services have to create their list of anonymous routes by requesting the pseudonyms of the nodes depending on the same local registry.

- **Step (1)** the Server choose one or more anonymous path to the local registry then generate the publishing message according to the nodes pseudonyms of the chosen route. The publish message is routed anonymously to the local registry that decrypts the content and store the service entry locally. The registry sends back an acknowledgement and a unique ID through the same anonymous path.
- **Step (2)** after storing the service coordinates the registry hashes the service description attributes. The hash key obtained is published into the appropriate registry in charge of this hash key value.
- **Step (3)** the client choose one or more anonymous path to the local registry then generate the request message according to the nodes pseudonyms of the chosen route.
- **Step (4)** the registry matches locally the request with the published service profiles. If the service does not exists locally, the registry hashes the request attributes to obtain a key. The local registry contacts the remote registry in charge of this key value in order to get the location of the registries that store the information about the requested services.
- **Step (5)** the local registry contacts the remote registry holding information about the requested services. The remote registry matches locally the request and sends back a response containing the entry related to the requested service.
- **Step (6)** the local registry send back the response to the client using the same anonymous route initiated by the client.

I. Security Evaluation

In this chapter we propose evaluate the security requirements fulfilled by this security solution.

Theorem 1 it is impossible for a user intercepting a clear message in the P2P network to identify the service provider/ requester.

Proof- private information contained in clear exchanged messages are no more labeled as private since the data holder is completely anonymous. Due to the onion based anonymazing system deployed on the back-ends of the network, an attacker intercepting these data is not able to make a link between users involved in the system and the discovery data exchanged.

Theorem 2 Private services are not accessible for non authorized users.

Proof- a server publishing private services has the possibility to encrypt the service profile to be published using ABE encryptions scheme in order to restrict the access for the discovery of his service to authorized users. Only meta-date related to the matchmaking is kept in clear. Only user corresponding to the restricted profile and keeping the corresponding ABE private key of this profile are able to access to service description.

Theorem 3 Fake published services are avoided by users

Proof- after receiving a response related to his service request, a client will verify the ABE signature attached to this response and verify if it corresponds to the requested service profile. If it is not, the response is ignored by the client.

Theorem 4 Non-trusted registries are not able to identify messages sources and destinations

Proof- Since all the discovery messages are anonym the registry can not affect the privacy of the participants. Only anonym data is handled by the non-trusted registry.

J. Performance and Results

1. Pairing-Based Onion Routing Costs

The measurements values of the pairing-based routing mechanism are strongly dependent on the number *NO* of onion nodes involved in the anonym path. The user has to generate the keys of each node in the path, then after sending the message to be routed each node has to decrypt his dedicated part of the message then forward it to the next node in the path and those for forward and backward direction of the routing. Table 5 presents the measurements results obtained during tests gathered on a 3 GHz Pentium D.

Operation	Time	PB-OR	
		Sender	Onion Node
Pairing	2.9 ms	0	1
Multiplication in G	1.0 ms	*NO*	0
Exponentiation in G_T	0.2 ms	*NO*	0
Total Time (ms)		1.2 *NO*	2.9

Table 5 : Pairing-based onion routing time measurements

2. Kademlia Request/Response Costs

In order to evaluate the latency of a request response in the wide world Kademlia network we generated 500 requests containing words extracted from a dictionary and we evaluated the mean latency time between the moment when we sent a request and the moment when we received the related response. The average latency time obtained with this test is evaluated to 7.2851 s, representing the mean of the whole requests obtained and described in Figure 46.

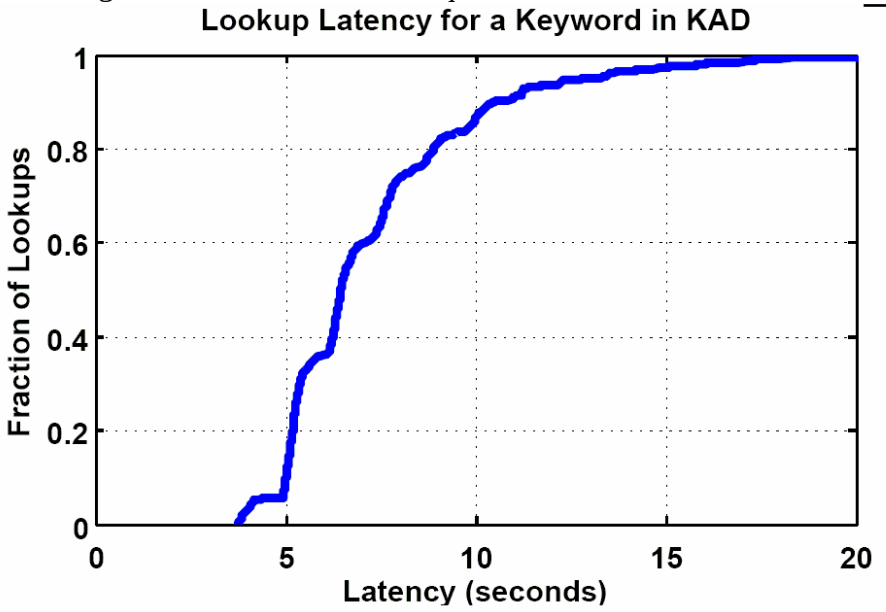


Figure 46 : Lookup Latency for a Keyword in KAD

K. Conclusion

In this chapter we proposed a scalable solution for securing service discovery without relying trusted registries. This solution is based on two topologies models: the first one related to the back-end access in which clients and servers are publishing and requesting services to the local registries in an anonymous way using onion based routing. The second topology model is related to the edge inter-domain routing over which all the local registries spread in the wide world are able to communicate and exchange discovery information using a P2P metadata exchange protocol (Kademlia) that enables a scalable and efficient indexation and retrieval of services profiles. Allying anonymous routing and P2P indexation this solution enable a real scalable and secure service discovery mechanism deployable every where independently from the domain and the network.

Chapter V. A Performance Analysis of Secure Service Discovery Solutions

A. Introduction

The deployment of ubiquitous computing systems and the trend towards Service Oriented Architectures will undoubtedly generalize the need for discovery mechanisms as essential components for locating ambient and location-based services. Service discovery in a network can be implemented in two manners, first using a decentralized architecture relying on point to point (broadcast) or point to multipoint (multicast) communication, and second using a centralized architecture based on an identified registry relied upon by users and servers to facilitate discovery request matching. The choice of an appropriate architecture to enable an efficient service discovery highly depends on the deployment environment (LAN, wireless or ad-hoc communications, Internet, VPN, etc.) and on parameters like the expected number of users and services, the type and amount of resources available (CPU, memory …), and the power consumption. This section introduces Markovian models that aim at assessing the impact at the application level of introducing security mechanisms, for both centralized and decentralized service discovery. Focusing on the application level, i.e., neglecting network artifacts such as delay or losses enables us to delineate network effects from the impact of security mechanisms in terms of processing overhead for the nodes in the system.

B. Related Work

These studies aim at getting a better understanding of the phenomena observed during discovery like message loss, faults, delays, or saturation, so as to select the most efficient service discovery mechanism for a given application.

1. Matching Strategies

[LUO04] and [BAR03] present simulation results and performance evaluation of several matching strategies called post-query strategies. They tested five strategies with two routing protocols (DSR and DSDV): the greedy strategy where all nodes broadcast queries and announcements in the network (flooding). The incremental strategy where each node broadcasts to a small set of other nodes. The uniform memoryless strategy announcements are sent to a random number of nodes. With memory strategy is similar to previous one but the nodes maintain a record about the visited nodes. This study compares the query success rate,

the number of transferred messages and the average waiting time. Still, this work focuses more on message propagation during discovery than the discovery process in itself.

2. Fault Tolerance and Crash Robustness

[DAB03] introduces a service discovery performance model that makes it possible to predict discovery service failure and overloading in real time. Authors propose to apply fault, crash and recovery rates during the simulation of a discovery process over centralized and decentralized architectures. These rates make it possible to observe which configuration is reliable and fault-tolerant. The results provided in this paper suggest that a decentralized architecture yields better robustness than a centralized one.

3. Publishing and Retrieval Time

[ABB07] focus their study on the performance evaluation of the service publish and retrieval time. Authors compared three discovery protocols: Bonjour [BON], Avahi and Free-Pastry and simulate their behavior. According to their results Bonjour provides a better performance for the registration and retrieval time.

C. Modeling Secure Service Discovery

1. Centralized Discovery

a) Description

Our security solution designed for a centralized configuration and described in chapter Chapter III relies on security policies provided by clients and services. Registries have to be considered by services and clients as a trusted third party whose role is no more limited to a basic matchmaker, but which evolves to a security guarantor. In this configuration, clients and services first establish a secure connection (e.g., SSL) with the registry to protect the confidentiality of the exchanged messages. Servers can restrict the discovery of their services to only certified users by specifying a security discovery policy to be enforced by the registry. Clients are also able to restrict the matching scope to some certified services by specifying a security discovery policy also enforced by the trusted registry. Both clients and servers have to provide credentials issued by a known authority that can be used by the registry to authenticate them during the policy verification phase.

b) Model

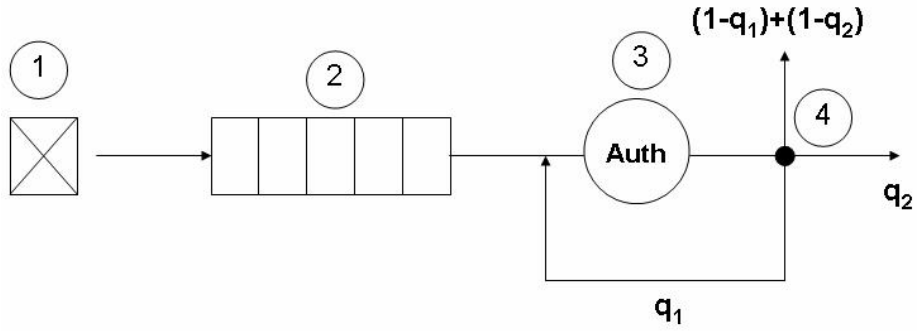


Figure 47 : Centralized Model

Figure 47 presents the processing phase of a secure client service request at the registry for a centralized configuration in case of a single-threaded registry, a sole thread is in charge of all processing steps.

The discovery process consists of these steps:

1. Client service discovery requests arrival: requests are assumed to be generated according to arrival process with a rate λ.
2. Buffering: The registry can temporarily store the requests to be processed by the central unit. Messages are served in a FIFO manner.
3. Request processing: the registry first matches a client request with the service profiles available locally. The matched service will be authenticated in order to verify its compliance with the security policy provided by the client. If the verification is successful, the registry also has to further authenticate the client in order to verify its compliance with the security policy provided by the service. The corresponding service time is a random variable with a mean value 1/μ.
4. Probabilistic decisions (acceptance or rejection): q1 is the probability that a service matches with a client request and also be compliant with its policy. q2 is the probability that a client be compliant with this service policy.

2. Decentralized Discovery

a) Description

The security solution proposed in the chapter Chapter II for a decentralized configuration relies on a particular usage of the Identity Based Encryption mechanism. The server advertises its service capabilities by multicasting its profile to the entire network. Clients can cache service information or ask for a specific service by multicasting its requests to all available servers and only concerned services will respond to him. With no possible reliance on any third party in ad-hoc configurations, clients and servers now must assure their own secure service discovery using a particular encryption scheme. Attribute Based Encryption (ABE) is adopted to make it possible for a server to encrypt its service description according to the restrictions imposed to users (i.e., only a class of users holding corresponding private keys will be able to access to services information). Clients also can use the same encryption mechanism in order to protect their request messages from unauthorized servers (i.e., only a class of servers is able to decrypt the request).

b) Model

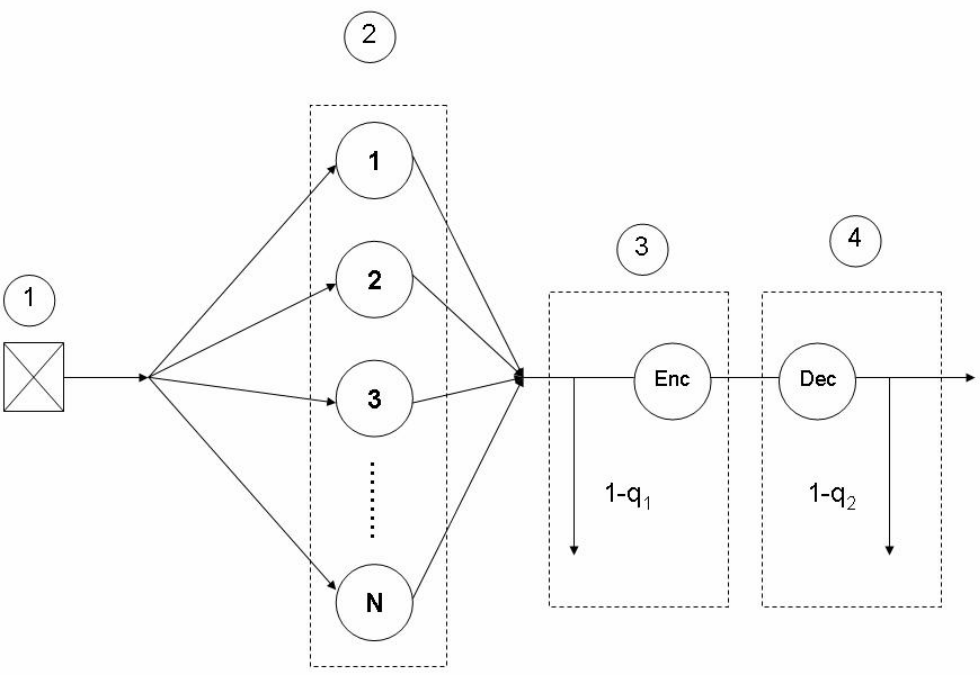


Figure 48 : Decentralized Model

Authentication and policy verification computing time are now replaced with encryption and decryption time. The fact that the request is routed using multicast adds complexity to the event handling. In a decentralized architecture, nodes usually have limited capacities as compared to a registry: For this reason, we considered in our model that servers do not buffer new requests when they are busy. In Figure 48, the execution takes this order:

1. Client service discovery request arrival: requests are generated according to an arrival process with rate λ.
2. Servers message processing: all the available servers are contacted by the client via multicast. Each of these servers has to decrypt the messages in order to authenticate and access to client's request. The time to decrypt is assumed to be a random variable with a mean value 1/μ1.
3. Service authentication: q1 is the probability to successfully decrypt a client request. In case of success the server has to encrypt the response message to the client.
4. Client authentication: q2 is the success decryption probability of a client.

3. System Model Assumptions

We make the following assumptions concerning the service demands and the processing for the service requesters and service providers.

- **Processing Time**. As described above, servers and clients must both perform some tasks during the discovery process. In the distributed configuration, processing time is essentially dedicated to encryption and decryption tasks. This processing time is variable for each message (message length, key length, padding size …), and also variable for the same message and the same encryption/decryption key. This variability is exemplified in [HEN05], in which we can observe an event independent duration time for Encryption/ Decryption actions. We model processing time in our decentralized model as an exponentially distributed random variable with mean 1/μ. In the centralized configuration we observed that the processing time is strongly correlated with the policy size (number of conditions/attributes to be checked) and does not vary for a given policy size. We assume enough diversity in the distribution of policy sizes to model the processing time as a random variable that we further assume to be exponentially distributed for mathematical tractability in our Markov models.
- **Inter-arrival Time**. We assume clients request to follow a Poisson process with rate λ.
- **Traffic class**. For a decentralized scenario, some servers could be more popular than others. In this case, the matching probability with the clients request is higher for popular services. To model this popularity, traffic classes could be used to distinguish between these services. The model described in this paper assumes that all services have the same popularity, or to put it differently, focuses on the performance of an average client.

D. Markovian Model

In this section we present Markovian models for the centralized and decentralized secure service discovery systems. We use networks of queues to model both approaches.

1. Markovian Centralized Model

For each request, the CPU of the registry is assumed to perform one or two authentication and policy verification cycles (since we assume that the registry is adopting a mono-threading

strategy for the request processing). The first cycle corresponds to service authentication, while the second one corresponds to client authentication. We model these two cycles using a bi-dimensional Markov chain (see Figure 49).
The first dimension of the Markov chain (A) represents the number of requests stored in the cache and the number of requests currently processed (0 or 1).The second dimension of the Markov chain (B) is a Boolean representing the request in the second processing cycle. If B = 1, the parameter A represents the number of requests in the cache. If B = 0, A represents the number of requests in the cache plus one request in the first cycle processing state. For instance, the left upper state in Figure 49 corresponds to A = 0 and B = 0.

a) Markov Chain

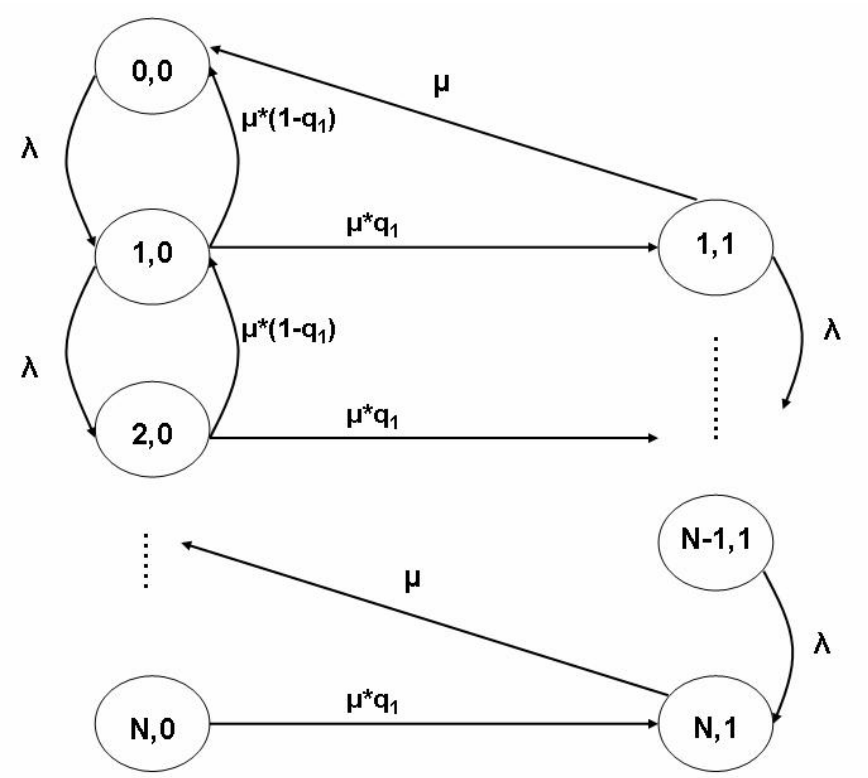


Figure 49 :Centralized Markov Chain Model

Figure 49 is the Markovian representation of the centralized system outlined in Figure 47. Client requests are entering the system according to a Poisson process with rate λ. The parameter A is the first to be incremented. After an authentication and verification first cycle (exponential with rate μ), the system moves to the second authentication and verification cycle with a probability q1 (B = 1) or the client is rejected with a probability (1-q1). If B = 1 and a new request reaches the registry, only A will be incremented.

b) Numerical Resolution

The bi-dimensional Markov chain described above is not easy to resolve using balance equations for the stationary distribution. We used a transition rate matrix and transition rate diagram to resolve numerically the system with the Gauss-Seidel method. The transition rate matrix *Q* is written by:

$$Q = \begin{bmatrix} -\sum_{j\neq 1} T_{ij} & T_{12} & \cdots\cdots & T_{1j} \\ T_{21} & -\sum_{j\neq 2} T_{2j} & \cdots\cdots & | \\ | & \cdots\cdots & \cdots\cdots & | \\ T_{i1} & \cdots\cdots & \cdots\cdots & | \end{bmatrix}$$

Where T_{ij} is the transition rate between state i and state j

Q can be decomposed as:

$Q = L + U - D$; where D is a diagonal matrix, L is the lower part of Q and U is the upper part of Q. In a stationary regime the steady state probability vector P can be written as follows:

$$P.Q = 0 \Rightarrow P.D = P.U + PL$$

$$\Rightarrow$$

$$P^{(n)} = \left(P^{(n-1)}L + P^{n}U\right)D^{-1} \qquad \textbf{Equation 1}$$

$$\Rightarrow$$

$$P_j^{(n)} = \frac{1}{\sum_{i\neq j} T_{ji}}\left[\sum_{i<j} P_i^{(n)} T_{ij} + \sum_{i>j} P_i^{(n-1)} T_{ij}\right]$$

The steady state vector P is used to calculate the different performance parameters of the system, including the rejection rate, the server usage rate, the mean number of users, the acceptance rate, the authentication rate, etc.

2. Markovian decentralized Model

Since we do not account for network effects, especially delay and losses, we assume that each client request reaches all available servers simultaneously, through some multicast communication scheme. For each request arrival the number of busy servers is equal to the total number of servers in the system. Each server independently processes the request. After decrypting the message, a server will generate a response, encrypt it, and send it to the client.

a) Markov Chain

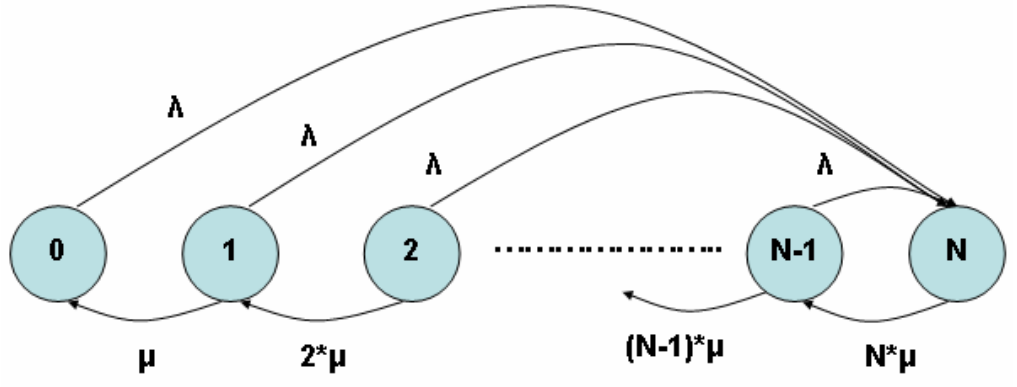


Figure 50 : Decentralized Markov Chain Model

The Markovian chain in Figure 50 represents parts 1 and 2 of the model described in Figure 48. Each state of this linear Markov chain represents the number of occupied servers. Part 3 and 4 of the Figure 48 are represented by the two states Markov chains in Figure 51.

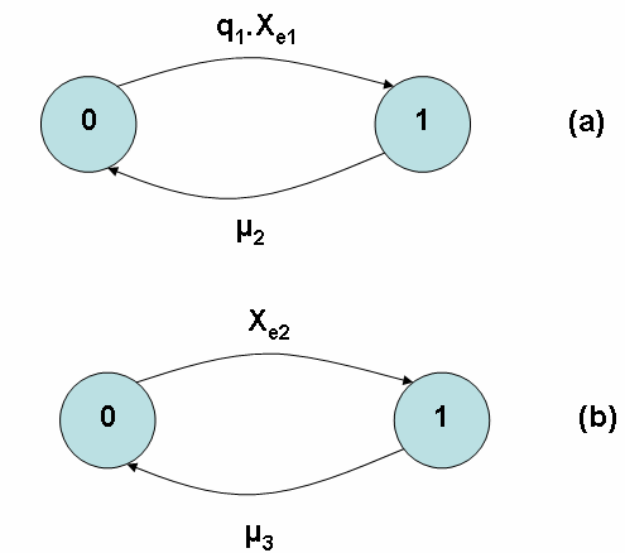


Figure 51 : a – Encryption Markov chain; b - Decryption Markov chain

The request arrival rate Xe_1 represents the output rate of the linear chain in Figure 50. The encryption Markov chain (Figure 51-a) is used to evaluate the impact of the server encrypting time on the system. Xe_2 represents the rate at which encrypted messages can be sent from servers. The decryption Markov Chain (Figure 51-b) is used to evaluate the impact of the decryption action performed by the client when it receives the encrypted response from the server.

b) Numerical Resolution

The Markov chain representing the system is linear. For this reason, it is easy to calculate the steady state probability vector P using balance equations:

$$\begin{cases} p(0)\lambda = p(1)\mu \\ p(1)(\lambda+\mu) = p(2).2\mu \\ \ldots\ldots\ldots \\ p(n-1)\big((n-1)\mu+\lambda\big) = p(n)n\mu \end{cases}$$

The steady state probability vector P can be written:

$$p(i) = \prod_{k=1}^{i}\left(\frac{(k-1)\mu+\lambda}{k\mu}\right).p(0)$$

With

$$\sum_{i=0}^{n} p(i) = 1 \Leftrightarrow p(0) = \frac{1}{\sum_{i=1}^{n}\prod_{k=1}^{i}\left(\frac{\lambda+(k-1)\mu}{k\mu}\right)+1} \qquad \textbf{Equation 2}$$

E. Matching Probabilities

The probabilities q1 and q2 described above represent the probability for a client or a service to obtain a successful matching (including authentication, access control, and decryption) with security policies protecting the access to resource profiles. This probability depends on the

number of elements of the systems and on the volume of vocabulary known by each element. The vocabulary volume is the amount of data knowledge related to a certain domain. For example, a subset of the medical vocabulary (scanner, radiology, dermatology, cardiology etc.) can be related to the services deployed inside a hospital building or the roles of users (surgeon, patient, etc.). In analogy to such concepts, we define a vocabulary as the global set of possible identities or roles in a system. The subset of this vocabulary is represented by the group of identities and roles existing in the system.
The probability to match an element (client or server), represented by a group of attributes *x* in a system, is defined by the probability *P* that these attributes belong to the subset vocabulary *C* part of the general vocabulary *V*:

$$P(x):\begin{cases}\dfrac{C}{V}; size(x)=1\\ \dfrac{C_C^x}{C_V^x}; size(x)>1\end{cases}$$ **Equation 3**

F. Model Validation

In this section, we consider several scenarios to compare the results obtained with our simulator (described below) and with the Markovian models presented in the previous section. For a complete validation, we have considered a large set of the system parameters by varying values like arrival rates, processing time, acceptance probabilities … These scenarios are not necessarily realistic but they aim to provide as complete as possible validation of our analytical models.

1. Java Simulator

In order to study the behavior of a system under various conditions, simulation is usually considered as a realistic solution to provide the expected performance measurements. A lot of network-oriented simulator tools are available but they are not really adapted to model security mechanisms (like encryption, authentication, access control…).For this reason we implemented our own event-driven simulator in Java using the SSJ [SSJ] Java library for stochastic simulation. This library provides methods for generating random variables, computing different measures related to probability distributions, performing goodness-of-fit tests.
The simulator is configured according to the models described in section Chapter V.C. The request source is represented by a generator that creates a new Client message structure entering the system every arrival time period (according to a Poisson process). In the centralized configuration the registry processor is represented by random variable generator that generates a uniform processing time values. In the decentralized configuration an exponential time generator is used for the same purpose. All the events of the simulator are collected by a scheduler that memorizes the arrival time of each client, the processing time of each request, the number of rejected requests, the number of successful matchings. All the data acquired with the scheduler are reused to compute the performance parameters described in the next section.
In order to verify the correctness of our self made simulator we compared our performance results with those obtained with the Java Modeling Tool (JMT) [BER07] that is a simulator for performance evaluation of queuing networks. We focus on the decentralized model that can be easily simulated using JMT unlike the centralized model for which the step 4 of the Figure

47 cannot be represented using the JMT simulator. This is one of the reasons that encourage us to develop our own simulator.

2. Rejection Rate

In Figure 52 and Figure 53, we compare the average rejection rate representing the probability for a client request to be rejected from a server before the processing phase. Rejection occurs when all the servers in the decentralized model are busy, i.e.,

$$R_d = P(N) \qquad \textbf{Equation 4}$$

And when N places of the cache of the registry in centralized model are occupied, i.e.,

$$R_c = P(N,0) + P(N,1) \qquad \textbf{Equation 5}$$

After setting the processing rate time μ to 0.2 (5 seconds on average to process a message), we varied the request arrival rate λ from 0.1 to 0.3 (10 seconds to 3.33 seconds of inter-arrival time) with an increment step of 0.01 in order to study different cases of system load: for instance, 0.1 corresponds to a light load while 0.3 corresponds to a heavy load. We also varied the number of servers and the buffer size at the registry (5, 10, and 20 places). The authentication probabilities (q_1 and q_2) are constant and equal to 0.5.

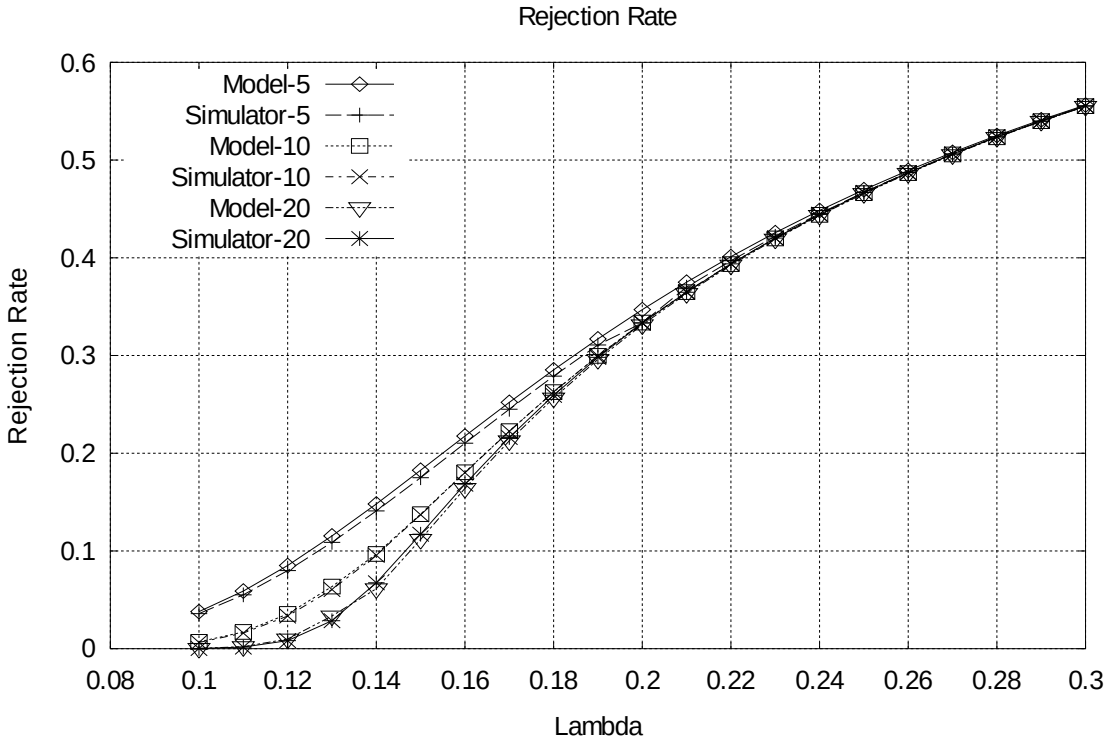


Figure 52 : Rejection rate in a centralized architecture

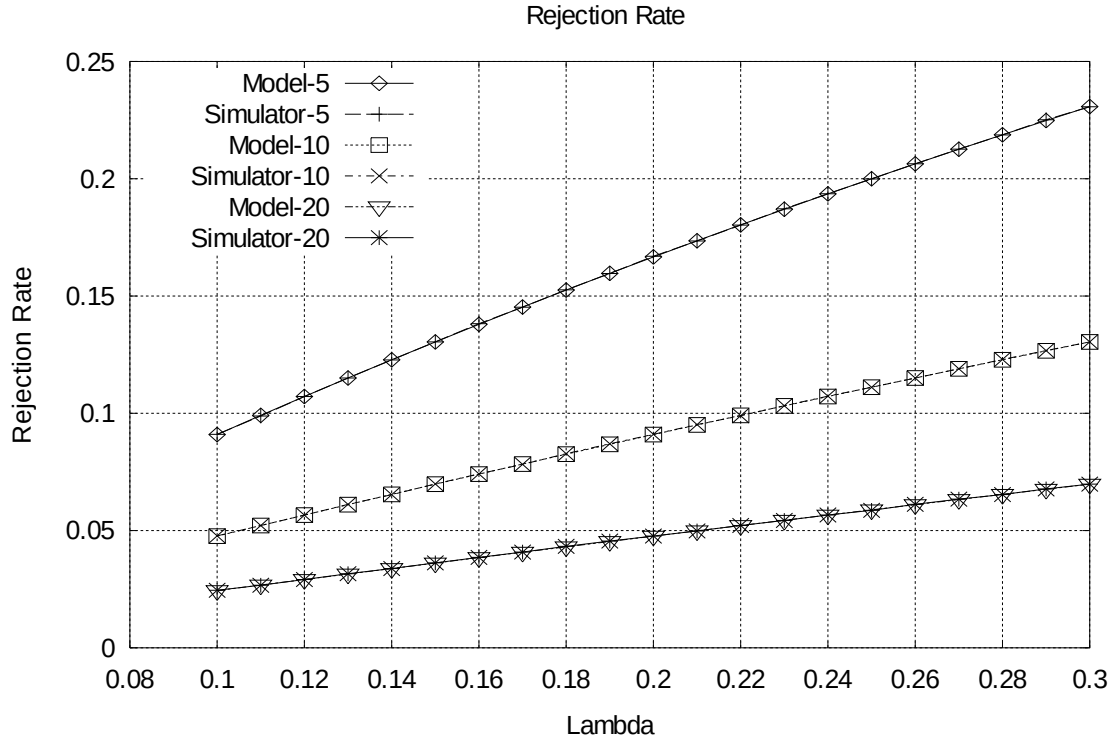


Figure 53 : Rejection rate in a decentralized architecture

We observe from Figure 52 and Figure 53 a perfect matching between the rejection rate measured by the simulator and the one computed by the Markovian model (the margin error is 0.07 %). For a distributed discovery model, the evolution of the rejection rate is linear: this behavior is due to the fact that for every sent request, all the servers become busy at the same time.

This means that a system administrator is able to predict in advance under which conditions his secure discovery system might become overloaded based on the behavior described through above and above, and which configuration is more suitable to ensure a better availability.

A straightforward observation of Figure 52 and Figure 53 could lead to the conclusion that rejection rate in centralized architecture is always higher the one in decentralized model. However this comparison is misleading as in reality a registry should be much more powerful than a server in a decentralized architecture, and action performed by registries are less expressive in terms of computing resources.

3. Server and Resource Usage Rate

Using the same scenario as in the previous section, we now provide a comparison between the usage rates of the servers for both architectures, in order to increase the accuracy of validation tests. To obtain a meaningful comparison between the distributed model (S servers where S > 1) and the centralized model (1 server but N slots in the queue), we focused on the resource usage time and not on the server usage time (the proportion of time the resources are busy). With decentralized discovery, this usage time is equal to:

$$U_d = \sum_{k>0}^{S} \frac{k.p(k)}{S}$$

Equation 6

where S is the number of servers in the distributed system. And for a decentralized architecture, with N the maximum capacity of the registry we obtain:

$$U_c = \sum_{k>0}^{N} \frac{k.p(k)}{N}$$ **Equation 7**

Figure 54 and Figure 55 illustrate a perfect matching between the resource usage rate measured by the simulator and the one computed by the Markovian model (with less than 0.05% of discrepancy). We notice that the usage rate in the decentralized model is independent from the resource size. This is due to the multicast allocation technique that balances the resource occupation. A system administrator should therefore be able to dimension the resources deployed in the system based on the behavior described in above and above. Buffer size can be optimally adjusted to the traffic in a centralized scenario and an optimal number of replicas of services (provided by the same server) can be deployed to ensure a good quality of service.

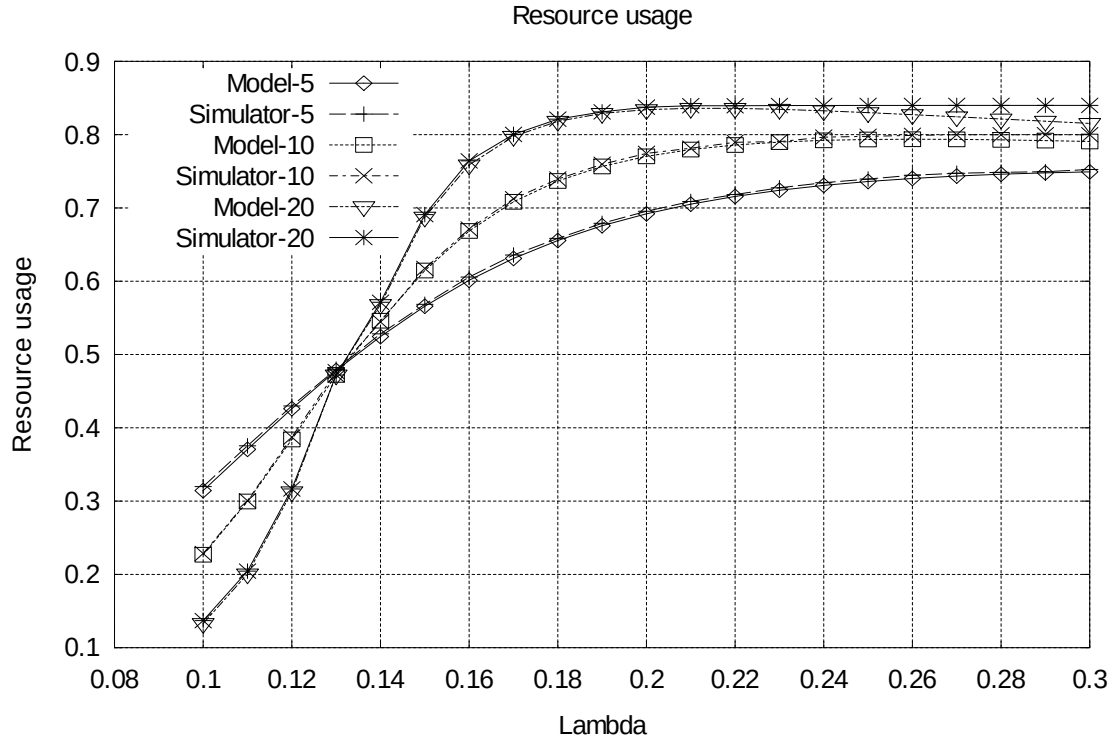


Figure 54 : Resource usage comparison in a centralized architecture

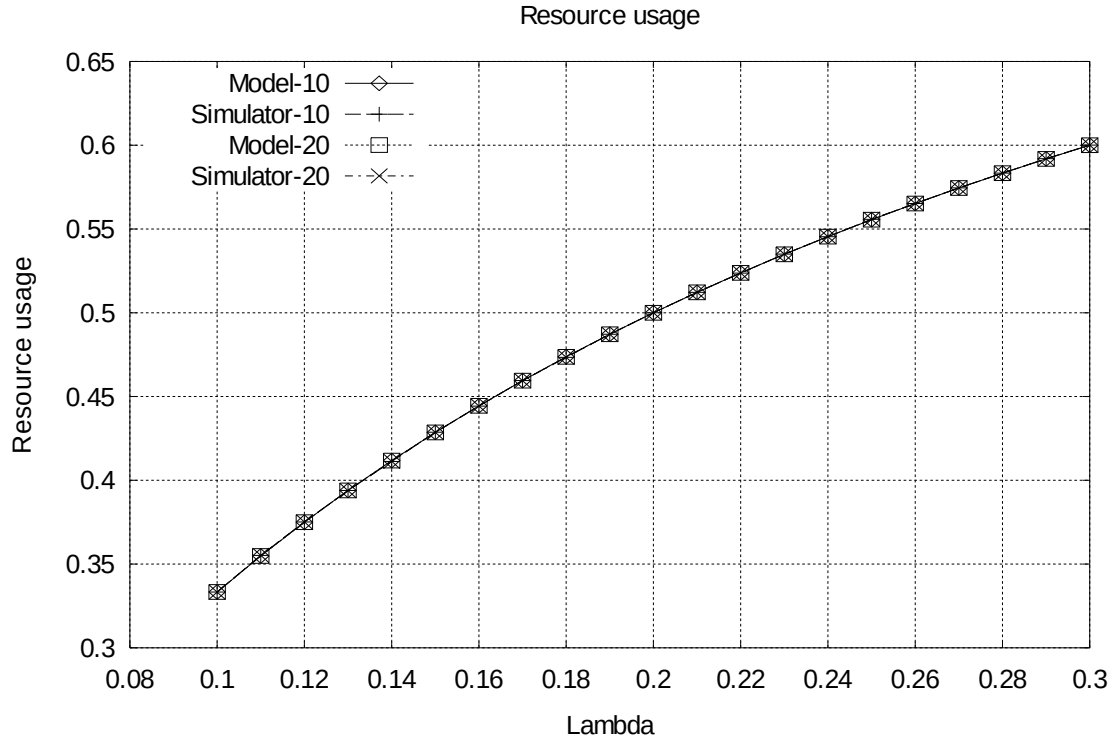


Figure 55 : Resource usage comparison in a decentralized architecture

We notice in the Figure 54 that the resource usage rate lower in 20 slots buffer when $\lambda < 0.13$ and becomes higher when $\lambda < 0.13$. This is however misleading as the number of free slots remains higher in any for a 20 slot buffer in any circumstances. For instance when $\lambda = 0.1$, in a 20 slot buffer, 17.6 places are free while 3.5 places are available in a 5 slot buffer. But if $\lambda =$ 0. 4 places remain free 3 in a 20 slot buffer as compared to 1.5 for a 5 slot buffer.

G. Performance Analysis

The performance study detailed below answers the following determinant questions for selecting one of the two secure solutions to service discovery: in which conditions is the request rejection rate better or worse? Which model is able to serve the largest number of clients? What is the fastest approach? What is the impact of a variable number of servers in the system? What is the impact of the matching probabilities on the performance?

1. System Setup

This relies on measurements obtained on real systems as published in previous studies.

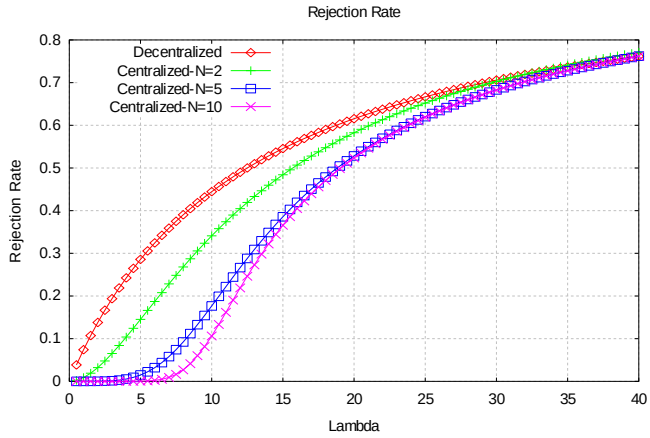
	Centr1	**Decentr1**	**Centra2**	**Decentr2**	**Centr3**	**Decentr3**	**Centr4**	**Decentr4**
λ	0.5 → 40	0.5 → 40	0.5 → 40	0.5 → 40	0.5 → 40	0.5 → 40	0.5 → 40	0.5 → 40
μ1	14.28	2.5	14.28	2.5	14.28	2.5	14.28	2.5
μ2	-	20	-	20	-	20	-	20
V	10	10	10	10	20	20	20	20
C	5	5	8	8	16	16	5	5
q1	0.5	0.5	0.8	0.8	0.8	0.8	0.25	0.25
Buffer Size	2-5-10	-	2-5-10	-	2-5-10	-	2-5-10	-

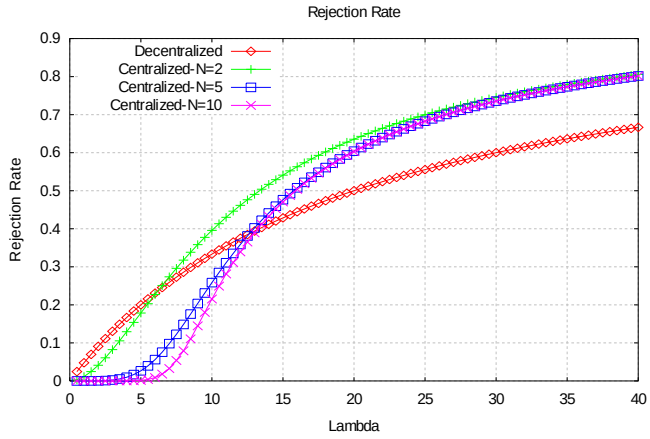
Table 6: Values of the input variables used in the tests

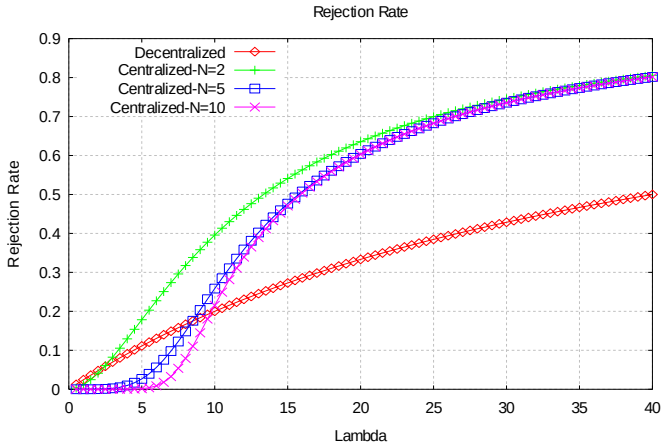
Four test scenarios are described in Table 6. The Attribute Based Encryption/Decryption duration are excerpted from [HEN05] according to set values for x={1,2,3}. We experimented with XACML policy reasoning and enforcement ourselves as detailed Table 4. The arrival rate of client requests, the number of services, the vocabulary size, and the matching probability q_1 are variable in our tests.

2. Rejection Rate

We compare the average rejection rate representing the probability for a client request to be rejected from a server before the processing phase. Rejection occurs when all the servers in the decentralized model are busy, and calculated according to the equation above. When N slots of the registry cache are occupied in the centralized model, the rejection rate is calculated according to the equation above.







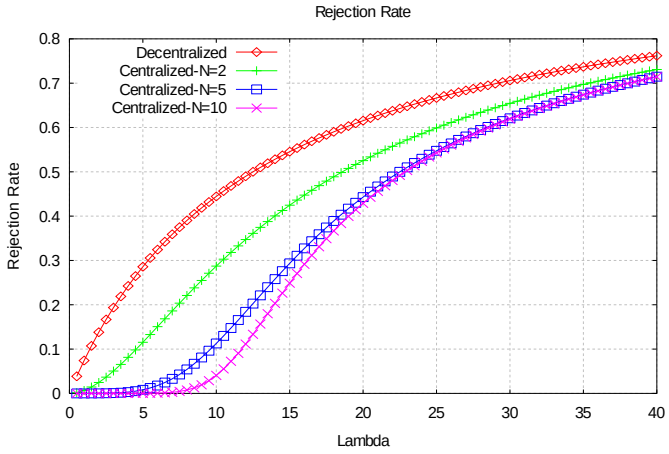


Figure 56 : Rejection rate curves for the four test scenarios

Figure 56 shows that the rejection rate due to a lack of resources is invariant for the centralized model in case of a fixed buffer size; in contrast, as the buffer size increases, the rejection rate reduces. Regarding the decentralized model, Figures 56-a, 56-b, and 56-c show that the rejection rate is strongly dependent on the number of servers deployed. As the number of servers increases, the rejection rate decreases. Figure 56-d shows that probability q1 does not affect the rejection rate for the decentralized model but clearly impacts the rejection rate for the centralized model. We can conclude that the decentralized system is more suitable in a system with a large number of servers.

3. Average Number of Users in the System

The average number of users Q present in the system is the temporal mean N(t) of the number of users observed in the system over period [0,T].

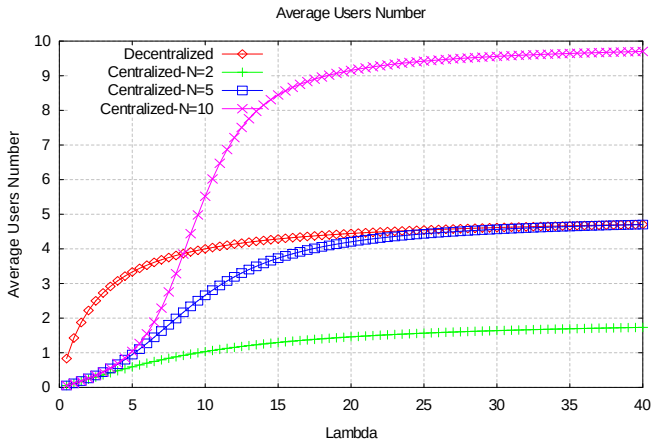
$$Q(T) = \frac{1}{T}\sum_{n} n \cdot T(n,T) \qquad \textbf{Equation 8}$$

In the centralized Markov chain model, Equation (4) can be written as:

$$Q_c = \sum_{n=1}^{Nbuffer} n \cdot \left(p(n,0) + p(n,1)\right) \qquad \textbf{Equation 9}$$

In the decentralized Model this equation becomes:

$$Q_d = \sum_{n=1}^{Nservers} n \cdot p(n) \qquad \textbf{Equation 10}$$



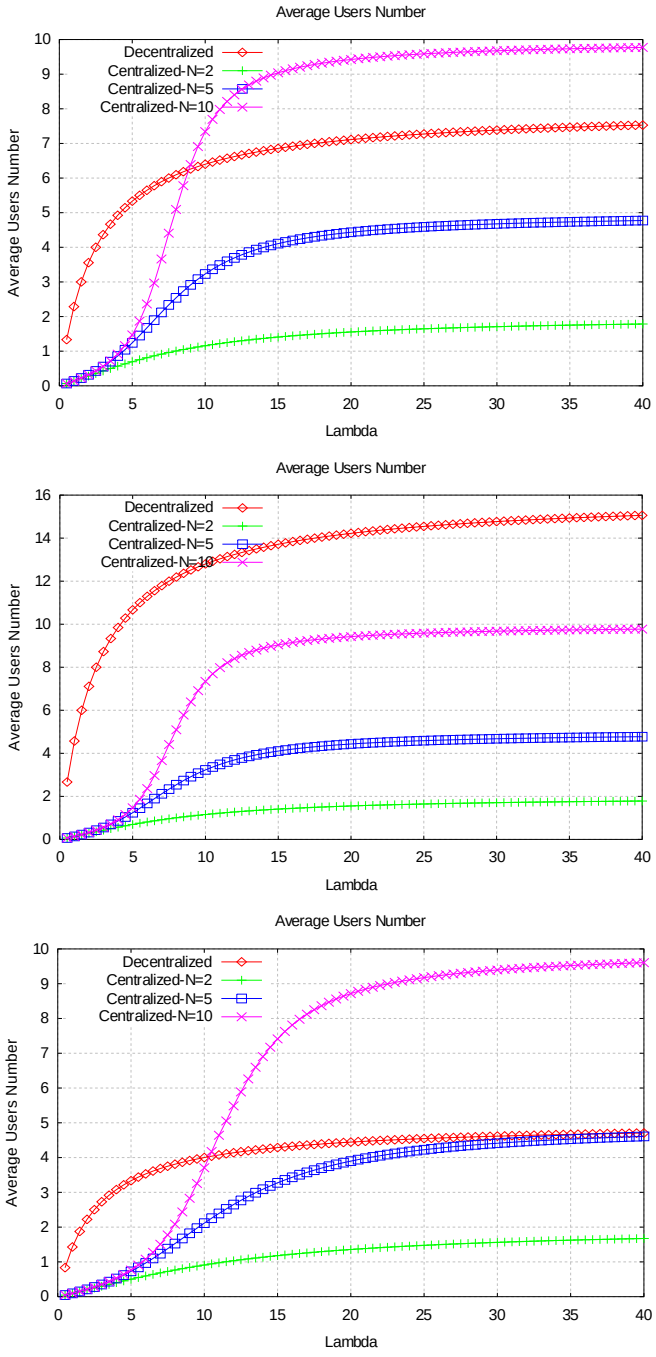


Figure 57 : Average number of users in the system for the four test scenarios

Figure 57 illustrates the capacity to serve requests. In the centralized system, it is proportional to the buffer size and to the matching probability (the bigger the matching rate, the longer requests stay in the system). In the decentralized system, the number of users served is proportional to the number of servers and the matching probability does not affect the number of users in the system.

4. Service Time Duration of a Request in the System

The lifetime R of a request in the system is the mean time spent by the requests accepted and processed during time period [0,T]. This rate can be computed using Little's Law that states that "*the long-term average number of customers in a stable system Q is equal to the long-*

term average arrival rate X multiplied by the long-term average time a customer spends in the system R" [LIT61]:

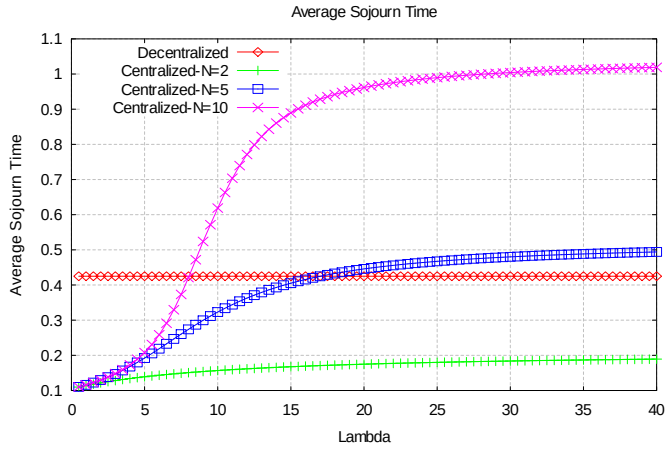
$$R = \frac{Q}{X}$$ **Equation 11**

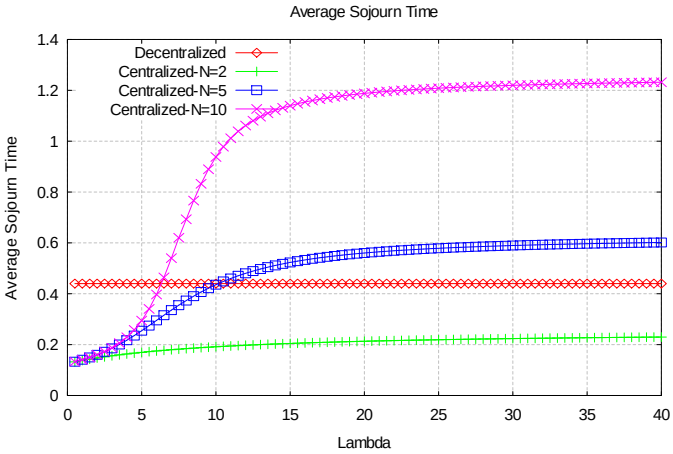
where X is the product of the probability of at least a user being served and of the Processing Rate. For the centralized model, the service time rate R is:

$$R_c = \frac{Q_c}{\sum_{n=1}^{Nbufeer} [p(n,0).(1-q1) + p(n,1)].\mu}$$ **Equation 12**

For the decentralized model, the service time rate R is:

$$R_d = \frac{Q_d}{\sum_{n=1}^{Nservers} \mu.p(n)}$$ **Equation 13**





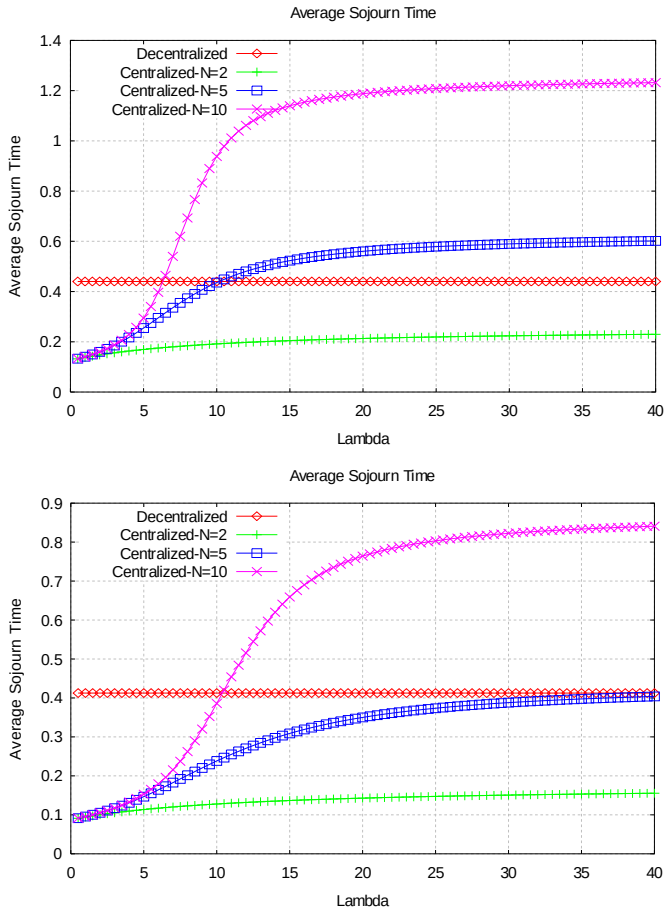


Figure 58 : Service time duration of a request in the system for the four tests scenarios

As depicted in Figure 58, the service time duration is strongly related to the matching probability in the centralized model. As this probability increases, the longer it takes to process the request. With a small buffer size, the system delivers a quicker response although with a high rejection rate. In contrast, the decentralized model exhibits a constant service time in every situation: this is due to the facts that tasks are distributed between servers.

5. Summary

Table 7 summarizes the tradeoffs from the performance study presented above: it lists the effects of a change in the infrastructure on the performance parameters that influence the quality of service of the service discovery functionality. One important result of this study is that no single approach can satisfy the requirements of all deployment scenarios.

Performance parameters:	**Reject**		**Number of users**		**Service Time**	
	Centralized	Decentralized	Centralized	Decentralized	Centralized	Decentralized
Increased Buffer Size	-	=	+	=	+	=
Increased Matching Probability	=	=	-	=	+	=
Increased Number of Servers	=	-	=	+	=	=

Table 7 : Performance summary (C : Centralized, D: Decentralized, + : increase, - : decrease, = : unchanged value)

H. Evaluation of the Impact of DoS Attacks on System Performances

1. Introduction

According to the security analysis of Chapter II.G and Chapter III.G, it is not possible for a malicious user observing the system and analysing all the exchanged discovery messages to build a software attack in order to corrupt the correct behaviour of a service discovery system. Software attacks usually relies on a partial or total knowledge of the message parameters like the service profile, the WSDL file, or the requested attributes in order to exploit these information for an illegal use. For instance accessing a WSDL file provides information about the service access functions and their parameters, with the possibility to detect holes in the service like buffer overflow risks. The unique attack that remains to the malicious user is the brute force DoS attack that consists in sending huge amounts of fake messages to the servers (or the clients) in order to force them to spend useless CPU time while processing bogus requests specially for a computing intensive operation based on cryptographic algorithms like the one implemented in the decentralized secure solution. DoS attacks persistently emulate several sources by either setting bogus source addresses in requests generated by a single intruder, or by having recourse to several sources as distributed intruders. The first approach used by DoS prevention methods consists of screening requests with bogus source identification. A second approach to DoS prevention consists in challenging the attacker with an anti-clogging mechanism [WAN03] in order to first add a processing cost to the requester (if the attacks are generated from a single source) and verify the validity of the requester (if the attacks are generated from distributed zombies machines).
In this section we present two DoS attack models for centralized and decentralised service discovery that analyses the performance impact of brute force DoS attacks on such discovery systems.

2. Attack Model

As detailed in the previous section, the attacks are limited to a brute force DoS attacks that consist of injecting fake messages on the system in order to perturb its normal behaviour. We compare two strategies: the first one we suppose that anti-clogging mechanisms are not activated and we measure the performance parameters of an increasing fake request rate, the second one we add a delay due to the anti-clogging mechanism also by increasing the fake request rate. The bogus traffic is modelled with an additional traffic class with a rate λ_{attack} melt to the clean one. This malicious traffic enters the system like other traffic, until the authentication phase in which all the corrupted messages will be dropped from the system. We try with this comparison to observe in which conditions the anti-clogging mechanism will be efficient in terms of performance measurements.

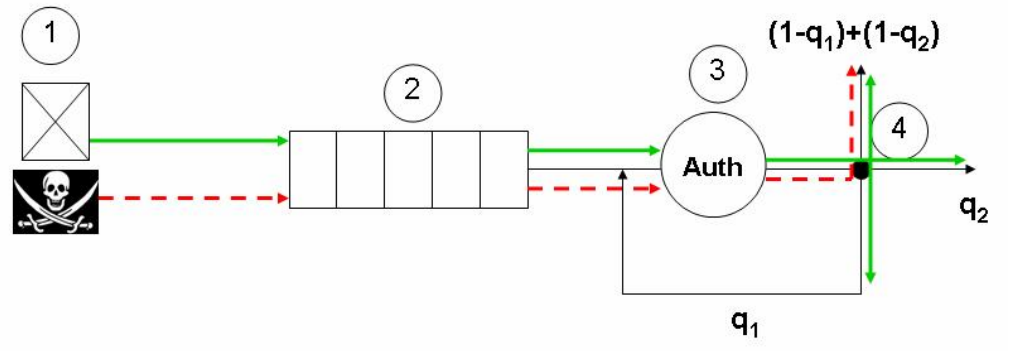


Figure 59 : Attack model for a Centralized Architecture

In the decentralized model (Figure 60) the corrupted traffic goes through all the available services before being dropped from the system.

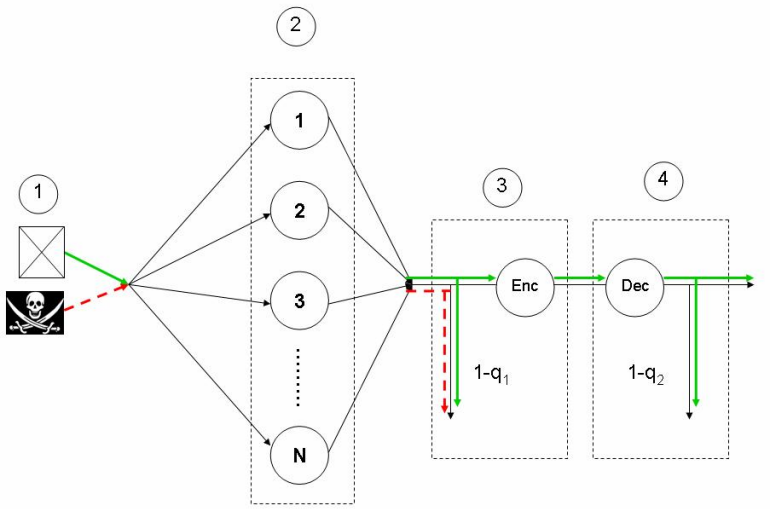


Figure 60: Attack Model for a Decentralized Architecture

The new traffic request rate generated by the attacker will impact on the global traffic request rate to be treated by the system and also on the positive matching probability of the requests. In the last section the matching provability q_1 as detailed in section Chapter V.E depends on the vocabulary size, the number of services, and the number of attributes related to the requested services but not on the request traffic rate. This is no more accurate with the addition of the new malicious traffic that will be automatically rejected during the matching process as described in Figure 59 and Figure 60 for this reason the matching probability that we note q_{attack} is dependent on the probability q_1 and the proportion of clean traffic compared to the whole traffic:

$$q_{attack} = \frac{q_1 \cdot \lambda}{\lambda + \lambda_{attack}}$$ **Equation 14**

3. Impact of a DoS Attack for a Protected and non Protected System

In order to study the impact of such DoS brute force attack we fixed the clean request traffic rate λ to 8 for both models. The processing time values are kept as the same as used in section Chapter V.G.1 for a non protected system, the number of servers and the buffer size (in the centralized model) are fixed to 5 with a vocabulary size of 10 and q_1 is equal to 0.5. We vary the attack rate λ_{attack} from 0.5 to 40 and finally we observe the variation of the rejection rate, the sojourn time and the success matching probability of a request. The use of anti-clogging mechanism like puzzle auction [WAN03] introduces an overhead for the request processing at the entry of the system estimated by Wang et al. [WAN03] to 4.6 ms that added to the processing time at the entry of the system.

We compare the performance parameters obtained for a protected system that uses an anti-clogging mechanism with those obtained with a non protected system.

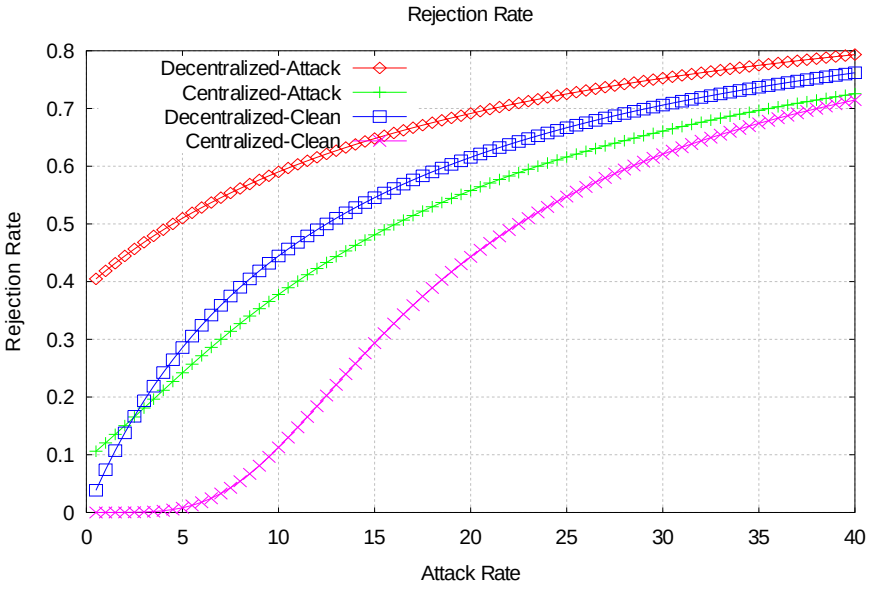


Figure 61 : Rejection rate

Figure 61 shows the impact of the malicious traffic on the rejection rate at the entry of the system that raises with the raise of the amount of corrupted requests.

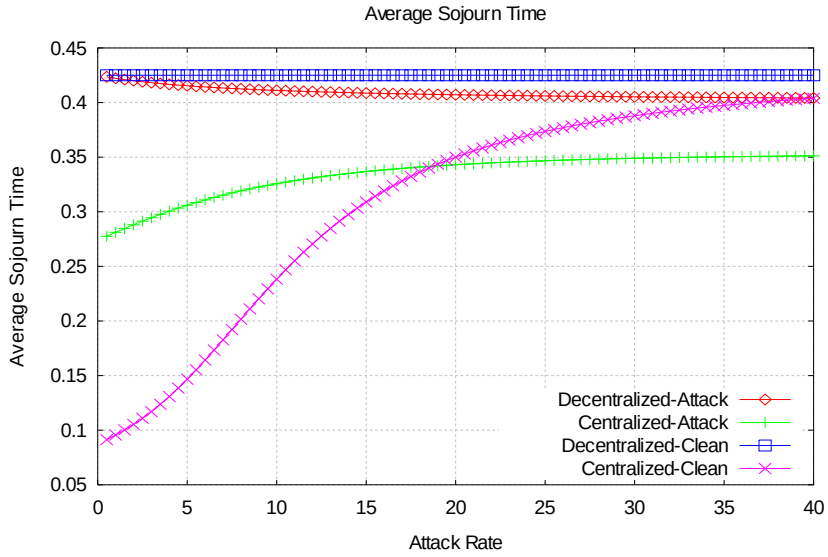


Figure 62: Total sojourn time of a request in the system

Figure 62 shows the impact of a DoS attack on the total sojourn time of a request in the system. We notice for the decentralized model that the sojourn time decreases when the attack rate increases and this is due to life time of a corrupted request that is less important than a life time of a clean request. The more there is fake requests on the system the more theses packets are rejected from the system after the authentication and the less the sojourn time is important. Concerning the centralized model when the malicious traffic is not important (less than 18) the sojourn time of the clean model is low, but when the malicious traffic raises, the number of fake requests rejected during the authentication raises and the growth of the average sojourn time slows down. This is the reason why the curves related to the centralised model intersect for a rate value equal to 18.

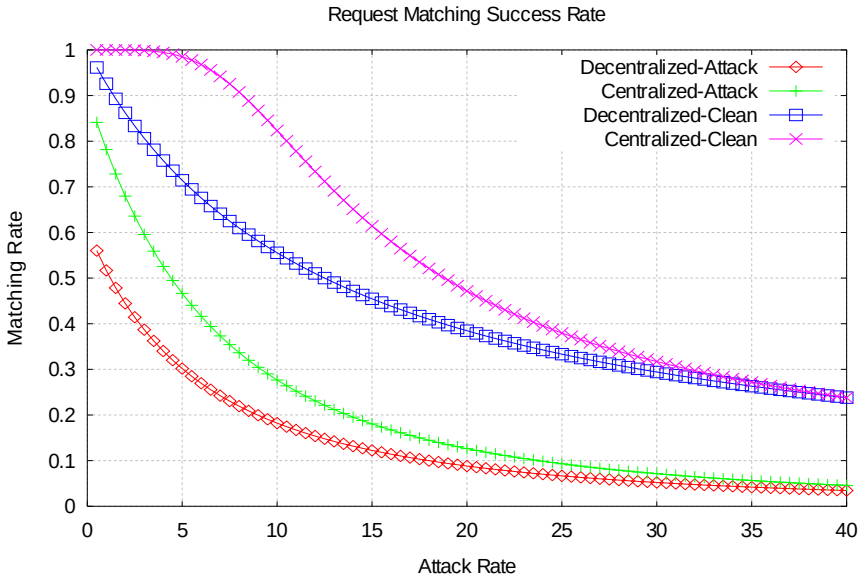


Figure 63 : Request successful matching probabilities

Figure 63 describes the variation of the successful matching probabilities for all the requests generated in the system. We notice that the impact of a DoS attack critically affects the matching probability with a difference of more than 22% between the different models. This is due to the impact of the corrupted traffic on the reject rate and on the matching probability described in above.

4. Summary

Adding a new traffic class representing corrupted request messages to the service discovery performance model enable us to predict the performance degradation of the system in case of DoS attack leaded against the service discovery system. The performance model presented in this thesis takes also into account the overhead generated by the deployment of anti-clogging mechanisms to counter DoS attacks.

I. Conclusion

In this chapter we introduced two analytical models to assess the impact of security mechanisms used in both centralized and decentralized secure service discovery architectures. This is the first such analytical study of this problem to our knowledge. Results provided by our Markovian models are extremely important to determine whether a centralized or decentralized strategy should be used to deploy services. They make it possible to undertake a systematic study of the robustness, efficiency, resource consumption, availability, message size, or acceptance rate in a SOA architecture, these performance parameters being easily computed thanks to the analytic approach. An important issue of this performance model was to extend the modelling tool in order to assess the impact of DoS attacks that could be addressed against the service discovery system. This extension permits to analyse the behaviour of the system in case of attacks and estimate the performance overhead generated if any anti-clogging mechanism is added to the system to counter these attacks.

Chapter VI. Context Awareness in Service Discovery

In Ubiquitous systems, the user is surrounded by an intelligent environment in which smart devices (RFID, sensors …) are integrated transparently. This environment is permanently changing and affecting the behaviour of users and applications which must dynamically adapt its functionalities in order to maintain interaction with the pervasive system. This adaptation is strongly dependent on the context of the environment, although users and application must be aware of this contextual information.

A. Definition

One of the first works dealing with context awareness in computing systems was carried out by Shillit [SHI94] that described contextual information as a combination of location information helping the software to adapt itself to the environment, a set of nearby objects and users, and the evolution of these elements over time. This definition is too restrictive to really describe contextual information, because it concerns a particular application related to location, time and objects. Dey [DEY01] proposes a better definition of the context awareness: "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*".

1. Data Modeling

In order to make the context information understandable by machines, some representation schemes are proposed. These representations use meta-tags that facilitate the automatic processing of the context types and their values without any human intervention. Gu et al [GU04] classify these context representation models into three categories:

- *Application oriented approach*: It is based on proprietary models that are strongly related to a specific application. These models lack formality and expressiveness because they use a local description. In these models, the application developer usually specifies his/her own model for the context representation. Nowadays, these purpose developers tend to use ad hoc XML vocabularies.
- *Model oriented approach*: This category of models is used a semi-formal approach rooted in existing and well accepted conceptual modeling techniques such as the

Entity-Relationship models or the UML diagrams. We can find in the literature some UML policy definitions [DUR03].

- *Ontology oriented approach*: Ontology is an information representation technique that refers to a particular subject of existence. It is a kind of a shared knowledge between different entities that have common understanding of some domains. Ontology is composed of a set of entities, relations, methods, rules, and axioms providing a kind of inference logic that enables an automated reasoning about this knowledge. In the service discovery systems, ontology is used as a service profile representation that provides richer semantic specifications. Such semantics enable a more flexible automation of service provision and use, and support the construction of more powerful tools and methodologies. W3C exerted lots of efforts to develop and deploy semantic web technology by encouraging new standards like Resource Description Framework (RDF[7]) that provides data model specifications and XML-based serialization syntax. The most popular ontology specification language proposed by W3C is called Web Ontology Language (OWL[8]) that enables the definition of domain knowledge structures and domain vocabularies. OWL is modeled through an object-oriented approach, and can be seen as an equivalent of description logic allowing a basic reasoning over structured data.

2. Reasoning

Reasoning mechanisms are often introduced in context aware applications to achieve adaptation. The content aware applications are adapted to achieve aggregation or translation of contextual information or to interpret the rules defined in the security policy. Concerning policy reasoning, the application developers will specify the interpretation of the set of rules according to their semantics. Obviously, the chosen data modeling approach might affect the underlying reasoning mechanisms. For example, OWL DL provides a built-in inference engine while other modeling approaches would require defining rules interpretation using different programming languages (Java, Prolog ….). Alternatively, the security module (which can be considered as a Java security library) may provide an API to develop the implementation of the security policy.

3. Quality of Context

The notion of Quality of Context (QoC) refers to the indicators values related to a category of contextual information. The QoC provides a logical quantification of contextual data in order to be objectively compared and evaluated. [SHE07] defies five QoC categories:

- **Precision**: describing the granularity with which the context information is described (Boolean, numeric, incremental without a numerical value, or weighted sets)
- **Freshness**: defines the time elapsed between the determination of the context value and its delivery to the system. A quantification metric value must be considered for every context type in order to evaluate the freshness of a data since its definition.
- **Temporal resolution**: describes the life time of contextual information. The value of the context can be considered as valid during a range of time depending on the type of the context. The location of a mobile car is no more valid after one or two minutes, but the temperature of a room can be correct after 30 minutes.

[7] http://www.w3.org/RDF/

[8] http://www.w3.org/TR/owl-features/

- **Spatial resolution**: defines the precision with which physical area an instance of context information is applicable. For example the GPS location of a set of people moving in the same room could me misleading.
- **Probability of correctness**: each sensor has an error range due to the natural imperfections of the measurements, this accuracy lack could raise if a large number of sensors are used, for this reason the context provider has to signal a probability of correctness value that must be considered by the user in order to evaluate the most realistic value of the context.

B. Context Awareness and Security

Pervasive computing offers users the possibility to interact with computers, devise physical spaces, and other users anytime and anywhere. In fact, in these ubiquitous environments, the applications must be as mobile as their users. They should be able to adapt to dynamic environments. These pervasive computing systems are able to capture, manage, and interpret the different context and situations; then, in a second stage, integrate these data to the applications. Due to the new vulnerabilities and exposures that are introduced by this context-aware architecture, their successful deployment will depend on the ability to secure them.
[CAM03] focuses on the user's interactions between the virtual and the physical worlds. In fact, pervasive computing environments often exploit physical location and other context information about users and resources to perform their services. For these reasons, environments become prone to more severe security threats that can affect users and equipments in the physical world and affect data and programs in the virtual world. Therefore, these new pervasive environments require both physical security (sensors devices protection) and digital security. Dynamicity and context-awareness flexibility for security mechanisms are studied in [COV02] by comparing the traditional security mechanisms (based on a relatively static access control decisions that do not change with context), with more pervasive environments security mechanisms offering better flexibility. The security architecture must also support mechanisms to securely collect contextual information that is used to enforce security policies. In particular, the architecture must account for the context in which requests are made (source of request, object requested, context information trust level, etc.). Patwardhan [PAT04] identifies context-aware application security risks as the basically malicious code; for example, using mobile devices as a carrier of malicious code especially for some applications that are capable of running on multiple platforms. For Julien [JUL04], security on context-aware environments concerns three types of threats: protecting mobile hosts from malicious agents, protecting agents from tempering hosts, and securing data (also contextual data).

1. Context-Aware Access Control

It is clear that the major part of the related work, concerning access control in context-aware security system, uses the Role Based Access Control RBAC [SAN96] for access control. In [CAM03], three types of roles (system, spaces and application roles) are defined. Access control policies are expressed in terms of space roles. The space administrator sets access control policies for resources within a particular space. When users enter a space, their system role is mapped into an appropriate space role. In [COV02], Covington suggests the Generalized RBAC [COV00], that is, an extension to the traditional RBAC that uniformly applies the concept of roles not only to subjects, but also to objects and system states (environments). But in [JUL04], there is a security agent that assumes the responsibility for mediating access to its data. To achieve a flexible access control, each agent specifies an individualized access control function (each agent can restrict access to its data for some agents) and each agent must provide credentials to identify himself.

2. Privacy and Context Awareness

In [ZUI04], [LAN02] and [JIA02], the main security threat in the context-aware environments is basically the violation of the user's privacy. In fact, contextual information is provided by many sensors that can be invisible to the users. It is obvious that these sensors, gathering information about people without being noticed, can be a threat to privacy.
To address the privacy problems in context-aware environments, in [CAM03], the authors use MIST [AL02]. The MIST infrastructure uses public key cryptography techniques to make communication impossible to trace by un-trusted third parties.
In [JIA02], every object, in an information space, is associated with a privacy tags. These tags are composed of three parts: A space handle that specifies which information spaces the object belongs to, a privacy policy that represents permissions specified by space owners for different types of operations, and a privacy property list describing the characteristics of an object (time life, representational accuracy, and capturing confidence).
For [LAN02] and [ZUI04], the privacy control architecture is based on the P3P [P3P] privacy policy standard (defined by W3C).

3. Context-Aware Encryption

Al-Muhtady et al. [AL06] defined a location-based encryption for publish/subscribe event systems. Here, a service provider encrypts its information with location-dependent encryption keys and makes the encrypted information publicly available. The network operator provides decryption keys to customers based on their current location. This approach provides location authentication, location-based access control models, and location based encryption. The major security breach of this solution is related to the secrecy of the decryption keys distributed to the users. Since these keys are not important for the users, nothing forbids them to publish these keys after the usage.

C. Context-Aware Security Policy

1. Introduction to Security Policies

A security policy defines the acceptable behaviors and the conditions of usage guaranteeing the protection of systems. Policies are rules governing the choices in behavior of a system, a service, an application or a user. A security policy is likely to be composed of a set of rules forbidding or allowing specific interactions with the system. The language used to define a policy can be informal, like human language as English, or it can be formal using a defined syntax associated with a specific semantics enabling complex processing (such as inference, reasoning, proof). Bishop [BIS03] defines security policy as a statement that partitions the states of the system into a set of authorized, or secure, states and a set of unauthorized or non-secure states. In particular, for Tonti et al [TON03] policy can be used to dynamically regulate the secure behavior of system components without changing code and without requesting the regular intervention of the administrator.
Each system has its own requirements concerning security aspects, and depending on these aspects; some kinds of security policies are used:

- **Authentication policy**: defines how systems can perform the authentication process. In the context-aware systems these policies define a set of contextual information that can be used as additional input of the authentication mechanisms.
- **Access control policy**: is used to grants or revokes the right to access some resources or to perform some actions.
- **Privacy policy**: it states the rules about how the private sensitive information concerning individuals is being collected; how the information being collected and

being used; how an individual can access his own data collected about him; how the individual can sign-out; and what security measures are being taken by the parties collecting the data.

- **Communication policy**: is used by the policy-based network management systems that specify the protocols, at the network layer, that must be used for the communication systems. Note that filters at the application layer are more related to the messaging policy.
- **Messaging policy**: defines which kind of message structure can be exchanged between the system entities. In the case of a WS web service case, a messaging policy can define the SOAP message structure.

2. Security Policy and Context-Awareness

Adding context information to security policies enables us to capture a new set of constraints introduced by the surrounding environment and correlated information as a new set of constraint parameters. This environment possibly changes and evolves during the execution of the application. Context awareness enables us to describe more accurate and dynamic stateful policy rules that take much more information into account. In fact, context information is strongly related to the application's environment, and this environment changes and evolves during the execution of the system. To add this notion of context in the security policies, we must be able to represent and model correctly the context information and also to reason about it.

3. Related Work

In pervasive computing environments, it is important to have a flexible and dynamic method for defining and managing security policies. In [KAG03], Kagal et al provided an overview of the dynamic policy definition languages. These languages (represented in a semantic language like RDF-S, DAML+OIL, OWL) are based on ontologies allowing a flexible and multi-domains application. The ontologies based language includes a representation of actions that allows more contextual information to be captured and allows for grater understanding and interpretation of contextual environment. Other solutions extend existing security policy languages like WS-security, XACML, and SAML in order to add the context reasoning. Classical access control policy rules are usually based on a set of constraints defined by a tuple <U, A, O>, which consists of three entities: the users U, the accessible object of the system O, and the actions A that can be performed. Context-aware systems may introduce a fourth entity representing the constraint C, which is likely to consist in context information. Based on this policy vision, Hu et al [GU04] propose an extensible context-aware access control policy based on a fine-grained definition of the context information. They proposed three definitions to model this contextual constraint C:

- The context type: defined as a property (a term used by the authors) related to every participant in a running application. It can be considered as a context classification depending on the application.
- The context constraint: defined as a regular expression based on logical operators that specify the contextual constraints C.
- The authorisation policy: defined as the representation of the tuple <U, A, O, C>.

Two algorithms are used for the dynamic context evaluation applied with a Web Service-based security infrastructure (see in Figure 64), where users can access data via a Web Service portal.

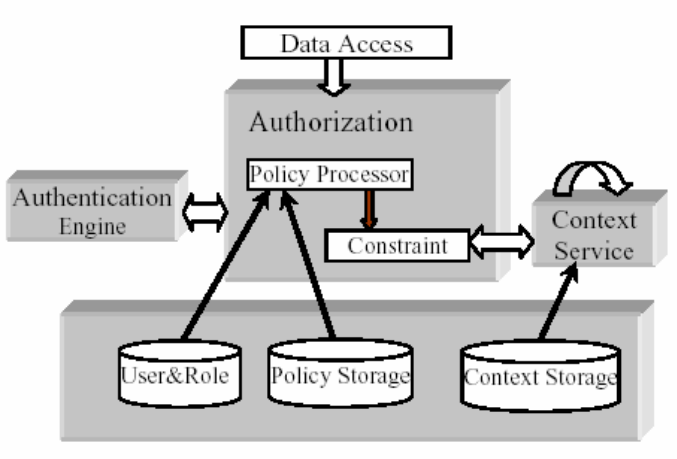


Figure 64 : Security Infrastructure [HU04]

This security infrastructure is based on static access control lists (ACL) for the access control mechanism, so that only known users can access the service; should there be a new user, the administrator must add manually the identity of the new member to the access list. Finally, this solution is original in so far as it uses context information in the security policy to add more precision and not dynamics (i.e. adjustment of the policy to the changing context) to their access control model. Covington et al [COV02],[COV00] define their own policy management tool for the policy specification and context representation. This tool uses a graphical representation of XML-encoded rules to specify access policies, role definitions and relationships. Their security policy is only focused on the access control policy; this is why their policy model is based on an extension to RBAC model called Generalized RBAC [COV00]. This access control model removes the subject centric limitation, by adding the possibility to define the policy from a subject-centric, object-centric, and environment-centric perspective. The notion of environment role allows the policy administrator to add roles to the surrounding environment of the system using a set of contextual information. For example, GRBAC enable to define an environment role corresponding to each day of the week. In his context-aware security system, Campbell et al [CAM03] defined a new implementation methodology for dynamic security policies [NAL02]. They defined the notion of dynamic policy as a program consisting of a set of guards and actions created and installed, on the fly, by the user or the administrator of the system. The originality of this development life-cycle is the last step (policy validation and testing). The final phase is composed of two steps: the first step, verification, performs a formal verification of the added proprieties and reasons formally about security guaranties (e.g. inconsistency); the second step, testing, checks the implementations of these policies to make sure they match with the specifications. This solution is not strictly speaking context-aware since the authors do not mention anything about the context modeling and reasoning. However, they propose to use this dynamic policy within a context-aware application. The application will be able to download and use the appropriate policy depending on the system situation; in other words, the dynamic policy offers a solution to adapt the system security to the environment.

Several other contributions to context-aware and policy-based security in the mobile networks can also be found in the literature. Recently, there has been special emphasis on the next generation networks (NGN). NGN represents the integration of the 3G networks (UMTS) with the WLAN. Jean et al [JEA03] propose a policy based context-aware service methodology; they choose a policy-based method because policies are ideal for context modeling. They purport to propose a solution to facilitate the provision of context-aware services in a secure manner by implementing a policy-based network management (PBNM). This PBNM focuses more on the security mechanisms in the network layer than on the

application layer. The authors applied their security policy on a network-centric context-aware service called TEANU (Transparent Enterprise Access for Nomadic Users). This service offers the user the possibility to access their workstation from anywhere, using the NGN technologies. In their scenario, the context-aware service uses the location of the user and the properties of the network to send the appropriate policy to the network domain (WLAN domain for example). Depending on the security policy, different strategies will be used to secure communication between the users and their office (for example IPsec VPN while on the WLAN). Context-aware security policy can also be used for mobile code verifications [HU04]. In fact, the applications and operating systems are nowadays very frequently updated, and these updates are often available online. Therefore, the user can download pieces of code (plug-in, patches) and will often execute them without proper verification.

4. Context-Aware Security Policy Requirements

The context-aware policies described in the previous section are usually defined for a particular application deployment (mobile code security, inter-network handover, personal network access control…). The security scope covered by these policies is limited to a specific behavior of the system and cannot be deployed for other systems including different applicative domains. This lack of flexibility is due missing of a general study about the requirements of a context-aware policy for pervasive systems. During my contribution to the Mosquito project [MOS] I was in charge of studying the requirements that must be satisfied by a generic context aware security policy.
Two policy application domains are identified:

a) Message Level Policy

Set of rules used to protect the confidentiality and the integrity of the exchanged messages in a pervasive system. These rules have to cover these specifications:

- **Confidentiality policy**: Depending on the message header (meta-data), the policy engine must decide which information must be encrypted and which mechanism must be used to perform this encryption. Confidentiality must also be adapted according to the contextual information related to the network or the domain. Depending on the network (local, or foreign) the confidentiality degree can vary. Or according to the wideband and the throughput offered in a system the policy will adapt the encryption strength and heaviness in order to maintain a minimum level of quality of service in the network.
- **Integrity policy**: To protect message against modifications the policy must define when and which mechanisms, like e.g. signatures or some message authentication code (MAC), are to be used to ensure the integrity of the message.
- **Messaging policy**: This kind of policy is used to fix some conditions and some rules concerning the messages exchanged between the different entities. For example, the messaging policy might suggest using a particular message format or header format between two entities. In some cases UPD port scanning must be ignored when IP source belongs to an untrusted domain and allowed if it belongs to local and trusted domains.

b) Application Level Policy

Each application can define the following policies that must to be respected by the rest of the entities of the system:

- **Trust policy**: This policy describes the constraints concerning the trust establishment between entities form the different administrative domains or from the same administrative domains. These constraints concerns trust establishment mechanisms like signatures, recommendation, certification or reputation. Location information could be used for a trust establishment between two close entities in case of data exchange between two PDA of two users staying behind, and refusing all the connexions coming from distant PDA.
- **Access Control policy**: Access control policies can be used either to control the access to the resources and the methods offered by the applications deployed within a pervasive environment. Each application offers one or more services and resources. For example a GPS service is in charge of acquiring context information and offering some functionality to use this information. These resources must be protected against non authorised access and security defines which entity can access to which resource and which functionality.
- **Federation policy**: A federation policy enables users to co-operate through exchanging data and/or programs. Two types of policies can be differentiated that are suitable for a federation. The device-federation policy describes when and how devices can federate. This is complemented by the delegation policy which describes when and to whom a user can delegate rights or application tasks. The policies can be "system-wide" if deployed in a pervasive framework, but can always be overwritten by some application specific policies that are deployed with the application.
- **Service Discovery policy**: This policy is applied to perform a secure service discovery request and also to protect the service registration. In this policy, we have two aspects: first the client discovery policy with which a client can define a set of constraints concerning his request; these constraints can concern confidentiality or access control. We can take the example of a client that requests only services that are signed by some certification authorities. Second the service publication policy, used by a server that wants to restrict the rights to discover its service to a predefined set of users. Location information could be a restriction for the service discovery or the request delivery in case of local service discovery.
- **Privacy policy**: used to protect the personal and private information of the users. It defines what information could be considered as private and also specifies who is allowed to access, read, write, modify, store or publish such data. This kind of actions that could be performed over sensitive private data could be restricted with temporal and geographical conditions. For example confidential documents of a company that should be displayed only in the restricted area.
- **Code verification policy**: This policy is used to verify the correctness and the safety of an imported code. These foreign codes can represent a threat to the system for two reasons: if the administrator uses a non self-made code (for example downloaded from a non trusted source) this code can be malicious and if an administrator develops his own patch code with some security vulnerabilities.

D. Securing Contextual Information

In this thesis, considering contextual discovery policy, we expose ourselves to threats at discovery policy enforcement and decision point. Thus, discovery services strongly rely on context. It raises security and trust issues regarding context acquired from sensor networks [ROM02].

1. Confidentiality of context information

Considering user's context information, user should be able to protect personal information such as his health status, or medical history. Hong proposes an architecture so-called Confab in order to provide privacy in ubiquitous computing [HON04]. Confab framework has been designed for protecting a user's location information in ubiquitous systems. It is based on an analysis of privacy needs for end-users and application developers. The main idea is that personal information are captured, stored and processed as much as possible on the user's device. Users can apply security control on their choice on those data. Context views have been introduced by Shankar [SHA02]. Context view is a fraction of an entity's context that is relevant to the application. In order to protect context information confidentiality, delivered context information can be controlled in context view. Bussard et al proposed security mechanisms for protecting privacy of context information [BUS04].

2. Integrity of context information:

This focuses on guaranteeing that the provided context information has not been corrupted by a third party. Hash functions or public key digital signatures can provide context integrity.

- **Hash function**: We consider two kind of hash function: un-keyed and keyed hash functions. In the first case, un-keyed hash functions, such as MD5 or SHA-1, provide a very low level of data integrity with respect to the use of context in application adaptation. With keyed hash function, such as MAC, we have the issue of the establishment of pre-shared key between the context information providers and the context-aware systems.
- **Public key digital signature**: the PKI-based approach might not be always suitable for context-aware systems, especially in distributed systems including low-cost sensors.

3. Trustworthiness of Delivered Context Information

In the scope of context-aware system, it is important to distinguish between two trust aspects: the trust of a context information provider that is supposed to be reliant and honest, like for example a national forecast provider. And trustworthiness of delivered context information related to the correctness and the quality of the delivered context information. A certified national forecast provider is trusted and honest, but weather information could be misconceived. In this case we can evaluate the trust about delivered context information by computing the distance between it and the real context using some existing methods:

- **Statistical analysis of context information**: the idea is to perform statistical analysis of the value of context information in order to detect unreliable ones. A simple example is to take an average of temperature values provided by different thermometer in the same room. Such naïve approach raises the following issue: if the temperature values collected are (10; 10; 11; 50) we get an average of over 20. It is obvious that the last value is a wrong one, and the temperature in the room should be 10. A simple solution is to use median that detects that value 50 is out of the scope, and eliminate the context information provider delivering this temperature value as an unreliable provider. Of course, real-life implementation requires use of much more sophisticated statistical tools.
- **Distributed reputation network**: [NEI07] developed a trust model for context aware provisioning based on recommendations and reputation. This trust model is used as an input parameter for a new formalism combining different trust aspects (identity, privacy, and context provisioning) in order to evaluate the resulting trust users have in a context-aware service. Usually the trustworthiness is based on previous experience

with the same party. The problem is that often there has not been any direct interaction before. In this case, an entity can establish trust in its communication partner based on the latter's reputation [GAN04]. Reputation is based on the collection of evidence of good and bad behavior. In the case where we use sensors in order to identify and authenticate users, their reputation depend, e.g., on whether the sensors input led to correct authentication or to a violation of a security policy. Reliability of context information is computed based on the previous experiences with a context information provider. The mathematical foundations for reputation management are rooted in statistics and probability. Trust context views [ROB03] have been introduced by Robinson in order to establish a degree and state of trust within a certain interactive context. This approach doesn't consider the authentication of context information provider.

- **Confidence value**: in the scope of Gaia architecture, Al-Muhtadi et al. [AL03] define a notion of confidence values in context-aware authentication. Gaia architecture establishes a confidence value based on the authentication devices and protocol used. For example, iris recognition would have a better confidence value than a fingerprint authentication device. The use of challenge-response protocol would be considered with a strong confidence value than a user's identity sent in clear text. Above considerations about security and trust are relevant for discovery and acquisition of context information. Reasoning about context information raises another security challenge. Indeed, reasoning about context information often deals with integrating several pieces of context information into another piece of context information. For example, from a patient's heart rate and blood pressure, we can evaluate his health condition. Assuming that patient's heart rate and blood pressure has different level of trust and security, the question is about the level of trust and security about the patient's health condition.

E. Context-Aware Security Policy for the Service Discovery

In the section Chapter I.F we studied the impact and the requirements of the context awareness for the service discovery and the benefits of the integration of the contextual information in order to provide a fine grained discovery policy. The existing approaches to the introduction of context-awareness for service discovery have so far only exploited raw context directly acquired from sensors (e.g. GPS location, remaining battery). While this may indeed enhance service discovery with basic context-awareness, the use of sensor context information is however too restrictive to define a discovery policy. Instead, we introduce semantically-rich context information, thereby supporting context reasoning: raw contextual data that are gathered from sensors, like the location, can therefore be further processed to derive complex information, such as proximity. De facto, we already improve the flexibility of context-aware discovery policies whose expressive power extends to more complex contexts. Context reasoning may also take place during the enforcement of discovery policies, and make it easier to combine context information coming from different sources.

1. Context Information Representation

As defined previously ontologies deal with knowledge management and object taxonomy based on their properties. Once objects are classified and characterized, an ontology makes it possible to specify relationships between those objects. In the scope of context-aware systems, context ontology classifies context information and establishes relationships (e.g. similarity relationship) between it [BRO04], [LEE04] and [BER05]. The use of a context ontology language provides a well defined syntax, semantics, and efficient support for reasoning about context information [LIS03]. CoOL, developed in the scope of CoDAMos project [BER05] was

selected for this purpose: it proposes four classes of context: user (e.g., user's activity, role, preferences), environment (e.g., ambient temperature, location), platform (e.g., hardware, software) and service (e.g., service description, data-flow, supported protocol) [PRE03]. Figure 65 provides an overview of the upper CoOL context ontology. With respect to context representation, CoOL is expressed in Web Ontology Language (OWL). We choose to take advantage from this rich ontology in order to express our requirements in terms of contextual representation. In our implementation context information are acquired via context toolkit widget [SAL99]. A Widget is a Java interface to a sensor.

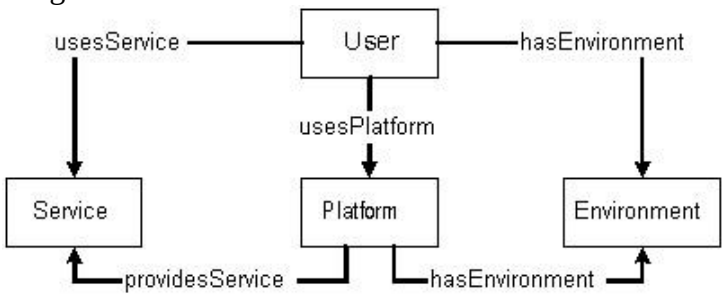


Figure 65 : Organization of CoOL [BER05]

2. Reasoning about Context Information

For reasoning about context information, we distinguish two complementary approaches for reasoning about context information: ontology and inference rule based reasoning. As described in the previous section, ontology supports relationship definition between context information. Based on those relationships, ontology eases reasoning about context.

Nevertheless the expressiveness power of OWL can be quickly restrictive as soon as we try to target more complex reasoning about context. Due to the restriction of OWL-DL, ontology based reasoning is limited to binary relationship between two context notions. For this reason, we can not quantify relationship in OWL-DL. For example, proximity relationship can be established in context ontology, but it can be quantify with respect to the distance between the users. For those reasons, we propose to use inference rule-based reasoning engine such as Jess [JES], in combination to ontology-based reasoning. Once ontology-based identified relationships between context, inference rule tends to cope with ontology-based reasoning by evaluating and quantifying those relationships. In our use case, proximity between doctor and patient has to be evaluated. An overview of our architecture for reasoning about context information is given in Figure 66.

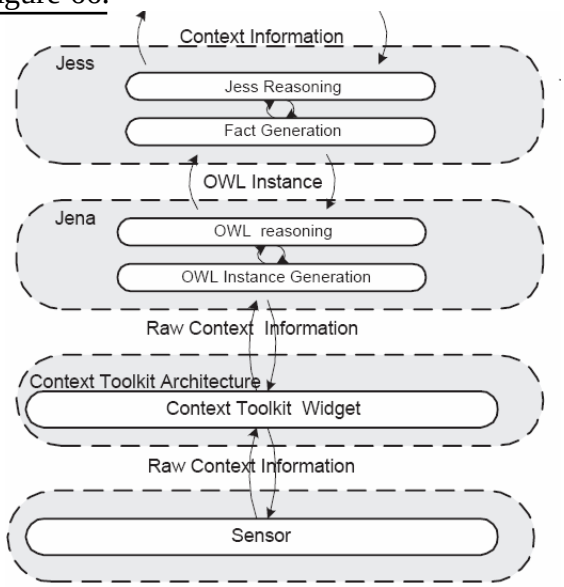


Figure 66 : Reasoning Module Architecture

3. Health Care Scenario

In order to provide a practical example of the usage of contextual information for the service discovery we can consider an emergency scenario in which in case of danger a patient cans discovery available physicians in his surrounding environment, according to two conditions: the authentication, and the physical proximity location of the physician. These conditions are expressed using the discovery policy defined previously.
The GPS location of the client and the physician are represented using the CoOL ontology as follows

```
<cool:GPSLocationElement xmlns:cool="http://discovery.service.com/cool#">
   <cool:Altitude>0.0</cool:Altitude>
   <cool:Latitude_Degree>44</cool:Latitude_Degree>
   <cool:Latitude_Minute>4</cool:Latitude_Minute>
   <cool:Latitude_Second>28.206604</cool:Latitude_Second>
   <cool:Longitude_Degree>123</cool:Longitude_Degree>
   <cool:Longitude_Minute>7</cool:Longitude_Minute>
   <cool:Longitude_Second>44.2411</cool:Longitude_Second>
</cool:GPSLocationElement>
```

Using this representation we can reason about the notion of proximity by defining a function for inference rule engine Jess that evaluated the proximity between the physician and the patient. This function *isCloseTo* is expressed as follow:

```
(defrule isCloseTo
    ;; LOOK FOR A PHYSICIAN's LOCATION ;;
    …
    ;; LOOK FOR A PATIENT's LOCATION ;;
    …
    ;; GET LATITUDE, LONGITUDE AND ALTITUDE OF  PHYSICIAN's LOCATION ;;
    …
    ;; GET LATITUDE, LONGITUDE AND ALTITUDE OF  PATIENT's LOCATION ;;
    …
    ;; TEST IF THEIR DISTANCE IS HIGHER THAN 20 ;;
    (test (< (distanceInM ?PhysicianLocation ?PatientLocation)20))

    =>

    ;; ASSERT THE RELATIONSHIP BETWEEN THE PHYSICIAN AND PATIENT ;;

    (assert
    (triple
    (p"http://www.owl-ontologies.com/unnamed.owl#isCloseTo")
    (s ?PatientLocation)
    (o ?PhysicianLocation)
    )
    )
)
```

This function is used for a discovery policy definition in which the patient will restrict the discovery and the access to his personal medical data module to authorized physicians present nearby. This function can be expressed with the XACML-based policy as follow:

```
<Apply FunctionId="isCloseTo">
  <Apply FunctionId="findLocation">
    <SubjectAttributeDesignator
      DataType=GPSLocation AttributeId="SubjectLocation"/>
  </Apply>
 <AttributeValue
    DataType="integer">2000 meters</AttributeValue>
  <Apply FunctionId="string-one-and-only">
    <SubjectAttributeDesignator
       DataType=string AttributeId="PatientID" />
  </Apply>
```

```
</Apply>
```

4. Performance and Results

In order to evaluate the efficiency of our solution we extended the Java prototype of the WS-Discovery protocol with the XACML functionalities, then we performed some measurements about time execution and memory consumption. For these experiments we used:

- OS: Fedora Core 5 with a Linux 2.6.x kernel i686
- CPU: Mobile Intel Pentium 4 CPU 1.70 GH
- Physical Memory 512 MB

In this table we provide all the measurement values related to each execution steps of the context aware policy based service discovery.

Actions	Time (ms)	Size (byte)
Sending Hello (Publish)	31	3963
Sending Probe (Request)	67	862
Service matching	370	-
Authentication	1572	-
Context Reasoning	4005	76000
Policy enforcement	862	-
Sending ProbeMatch (Response)	15	1622

Table 8 : Measurement values

The context reasoning step is composed of a sequence of actions. First OWL reasoning generates 231 rules and 66 OWL instances, which represents around 19 Kb. It takes 1302 ms for reasoning about the similarity between patient's pulse and heart rate. Then inference rule-based reasoning, the translation of OWL instances to facts takes 2453 ms and generates 335 facts, which represents 57 kb. Finally, inference about patient's unconsciousness costs 190 ms, whereas proximity evaluation takes 60 ms. In conclusion, OWL-reasoning is less consuming in term of memory usage. But it is slower than inference-rule reasoning. Overall, reasoning about proximity or patient's health condition is performed in less than 4 seconds, with 80 kb.

F. Conclusion

In this chapter we introduced the notion of context awareness in pervasive computing systems and particularly the impact of the contextual information in service discovery mechanisms in terms of matching accuracy and flexibility to the environment. We demonstrate how context awareness could be incorporated in some security mechanisms and like security policies and more particularly service discovery policy described in section Chapter III.C for which we propose a context-aware extension applicable with secure registry-based service discovery. In this solution we tend to cope with the identified threats in chapter Chapter I.G by defining a secure registry solution relying on discovery policy. We motivate the use of secure and trusted context-information in order to adapt the security policy enforcement with dynamic environment. As proof of concept we implemented a prototype securing WS-Discovery protocol and relying on a context-aware extension of XACML access control policy. Our

approach solves user's privacy and service access control by introducing context-aware access control for discovery service and efficiently supports trust establishment between different actors of the system.

Chapter VII. Conclusion and Perspectives

This thesis deals about security issues related to the service discovery in pervasive computing. Various approaches and techniques are tackled with the wish to overcome the most important threats and weakness of such systems independently from the deployment architecture.

The first part of this study introduces the notion of service discovery in pervasive computing systems. Different architectures and deployment strategies related to this domain are detailed and organized according to the applicative environment. An overview of the existing service discovery protocols and standards are technically described and compared in order to get an idea about the different strategies and techniques adopted and deployed in real pervasive computing systems. This specifications analysis permits to detect security holes, weaknesses and vulnerabilities related to the protocols that could be exploited by attackers and provides a detailed threat model describing the potential attacks and the countermeasures that could be adopted to overcome these vulnerabilities. The required security aspects of service discovery to mitigate the threats are identified in security requirement list that should be followed by an administrator to design a secure service discovery protocol.

The second part of this thesis describes the different solutions proposed in order to overcome the security challenges described in the security requirement list for any architecture and deployment scheme. This first solution relies on the usage of Attribute Based Encryption scheme to secure the discovery messages by encrypting sensitive and private data contained in these messages and permitting to clients and services to restrict the scope of the discovery to only trusted and authenticated elements. This solution does not rely on a trusted third party except a unique certificate authority issuing encryption/decryption keys. The second solution exposes a registry-based solution for securing service discovery mechanisms that tend to overcome threats and attacks related to registry-based environments. In addition to PKI-based message encryption mechanisms used to hide discovery message, this solution introduces the new concept of service discovery policy that is used by clients and serves in order to express their security and privacy requirements during the service discovery process. The registry dedicated usually to the matchmaking between published services and clients requests has a new function of trusted third party in charge of authenticating discovery elements and enforcing the discovery policies provided by the users. Both of these two solutions are dedicated to small networks (like PANs, LANs , WLANs, and home network) and it is difficult to expend it to wide networks with millions of elements without scarifying matching efficiency and security. For this reason a third scalable and secure solution is proposed to ensure a secure service discovery independently from the network size and the number of element involved in the system. This solution relies on anonymization mechanisms used by clients and servers to reach securely reach local registries that in case of mismatch with users requests will expand the discovery to other registries spread in the world wide network using a P2P communication system for indexing and retrieval.

The third part of this dissertation presents a performance study that analyses the impact of the deployment of such security mechanisms in service discovery protocols. For this part of the thesis we developed a mathematical model validated by simulations that permits to calculate the different performance parameters of the secure systems in order to undertake a systematic study of the robustness in case of Dos attacks, the efficiency for extreme environmental condition, and resource consumption. This model makes it possible for a system administrator that is deploying a secure service discovery system to choose the appropriate security solution

depending on the network topology and the environmental profile of the area in which he planned to deployed it. He can also adjust the needed resources to ensure a robust and stable system deployment.

The last part of this thesis focuses on the impact of the contextual information related to the surrounding environment in the behavior of users and applications in pervasive computing systems. We discussed how contextual information can be used as a dynamic parameter for the security mechanisms and the service discovery systems by incorporating a contextual reasoning in our security discovery policy.

An interesting issue need further research is the conception of a new policy language dedicated to service discovery. This language should describe all the identified functionalities that are useful to perform a secure service discovery including authentication, encryption, signature verification, reputation, recommendation, negotiation etc…
Since we supposed that users are not really mobile in the system, an interesting extension of the performance discovery model could be a mobility model evaluating the discovery system when users are joining or leaving a service discovery enabled area.

Bibliography

[ABB07] H. Abbes, C. Cérin, J-C. Dubacq, M. Jemni, “Performance Analysis of Publish/Subscribe Systems”, Rapport Interne LIPN (15/05/2007), 2007

[AL02] J. Al-Muhtadi, R. Campbell, A. Kapadia, D. Mickunas, and S. Yi, "Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments," presented at International Conference of Distributed Computing Systems (ICDCS 2002), Vienna, Austria, 2002.

[AL03] J. Al-Muhtadi, A. Ranganathan, R. Campbell and M. D. Mickunas, "Cerberus: A Context-Aware Security Scheme for Smart Spaces," in the Proceedings of the First IEEE Annual Conference on Pervasive Computing and Communications (PerCom 2003), pp. 489-496, Fort Worth, Texas, March 26, 2003

[AL06] J. Al-Muhtadi, R. Hill, R. Campbell, and D. Mickunas, "Context and Location-Aware Encryption for Pervasive Computing Environments" in 3rd IEEE International Workshop on Pervasive Computing and Communication Security (PerSec), 2006 at IEEE PerCom 2006

[ALM03] F. Almenarez, C. Campo: “SPDP: A Secure Service Discovery Protocol for Ad-hoc Networks”, In 9th Open European Summer School and IFIP Workshop on Next Generation Networks, Budapest, 2003.

[BAG05] W. Bagga, R. Molva, “Policy-based cryptography and applications”, FC' 2005, 9th International Conference on Financial Cryptography and Data Security, 28 February-03 March 2005, Roseau, The Commonwealth of Dominica - Also published in LNCS Volume 3570

[BAR03] M. Barbeau, E. Kranakis, “Modeling and Performance Analysis of Service Discovery Strategies in Ad Hoc Networks”, International Conference on Wireless Networks pp. 44-50, 2003

[BER01] O. Berthold, H. Federrath, and S. Köpsell. "Web MIXes: A System for Anonymous and Unobservable Internet Access," H. Federrath, editor, Designing Privacy Enhancing Technologies, LNCS 2009, pp 115-129, 2001.

[BER05] J. Van den Bergh and K. Coninx, “Towards integrated design of context-sensitive interactive systems”, Third IEEE International Conference on Pervasive Computing and Communications Workshops, 2005.

[BER07] M.Bertoli, G.Casale, G.Serazzi, “The JMT Simulator for Performance Evaluation of Non-Product-Form Queueing Networks”, SCS Annual Simulation Symposium 2007.

[BIS03] M. Bishop, “Computer Security: Art and Science”, Addison Wesley Professional, 2003

[BIS06] H. P. Bischof, J. V. Donaldo, “M2MI Service Discovery Middleware Framework”, Proceedings of the International Conference on Pervasive Systems and Computing (PSC 2006), Las Vegas, Nevada, USA , 2006

[BLO70] H. B. Bloom, "Space/time trade-offs in hash coding with allowable errors", Communications of the ACM 13 : pp. 422–426, 1970

[BLU] “Bluetooth Security,” white paper, Bluetooth SIG Security Expert Group, 2002.

[BON] Apple’s Bonjour http://www.apple.com/macosx/technology/bonjour.html

[BON01] D. Boneh and M. Franklin, “Identity-based encryption from the weil pairing“, Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, pages 213–229. Springer-Verlag, 2001.

[BRO04] T. Broens and al, “Context-aware, ontology-based, service discovery”, 2nd Symposium on Ambient Intelligence, 2004

[BRO04-2] Broens, T.H.F., “Context-aware, Ontology based, Semantic Service Discovery”, Master thesis, University of Twente, the Netherlands, 2004

[BUS04] L. Bussard L., Roudier Y., “Untraceable secret credentials: Trust establishment with privacy,” in PERCOMMW’04. Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.

[CAM03] R Campbell, J Al-Muhtadi, P Naldrug, G Sampemane, M D Mickunas “Towards Security and Privacy for Pervasive Computing”, Software Security - Theories and Systems, Springer Berlin / Heidelberg, pp. 77-82, 2003

[CAM05] C. Campo, M. Munoz, J.C Perea, A. Mann, C. Garcia-Rubio, “PDP and GSDL: a new service discovery middleware to support spontaneous interactions in pervasive systems”, in proceedings of Pervasive Computing and Communications Workshops, 2005.

[CAR05] Carminati, B., Ferrari, E., Hung, P.C.K, “Exploring Privacy Issues in Web Services Discovery Agencies”, IEEE Security and Privacy .Volume 3, Issue 5, 2005.

[CAR07] R. S. Cardoso, P-G. Raverdy, V. Issarny. "A Privacy-Aware Service Discovery Middleware for Pervasive Environments", In Proceedings of IFIPTM 2007 Joint iTrust and PST Conferences on Privacy, Trust Management and Security. 2007.

[CHA81] D. Chaum, “Untraceable Electronic Mail, Return address, and Digital Pseudonyms”, Communications of the ACM 24/2, pp. 84-88, 1981.

[CHA06] D. Chakraborty, A. Joshi, Y. Yesha, T. Finin, , "Toward Distributed Service Discovery in Pervasive Computing Environments", Article, IEEE Transactions on Mobile Computing, pp. 97- 112, 2006.

[COR] CORBA, http://www.corba.org/

[COV00] M J. Covington, M J. Moyer, M Ahamad “Generalized Role-Based Access Control for Securing Future Applications” In 23rd National Information Systems Security Conference, Baltimore, MD, October 2000

[COV02] M J. Covington, M Ahamad, S Srinivasan “A Security Architecture for Context-Aware Emerging Applications”, Proceedings of the 18th Annual Computer Security Applications Conference, December 9-13, 2002 Las Vegas, Nevada, 2002

[COV04] M. J. Covington, M. Ahamad, I. Essa, and H. Venkateswaran. Parameterized Authentication. In Proceedings of 9th European Symposium on Research in Computer Security (ESORICS 2004), pages 276–292, September 2004

[CRO05] S. Crosta, J C. Pazzaglia, H. Schottle, Modelling and Securing European Justice Workflows”,6th Information Security Solutions Europe (ISSE 2005) Security Conference - Budapest, Hungary, 2005.

[CZE99] S.E. Czerwinski et al, “An Architecture for a Secure Service Discovery Service” , In Proceedings of MobiCom '99, Seattle, WA, August 1999.

[DAB03] C. Dabrowski, K.L. Mills, A.L. Rukhin, “Performance of Service-Discovery Architectures in Response to Node Failures”. Software Engineering Research and Practice, pp. 95-104, 2003

[DAV02] N. Davies and H.W. Gellersens. “Beyond prototypes: Challenges in deploying ubiquitous systems”. IEEE Pervasive Computing, 1(2):26--35, 2002.

[DCO] http://msdn2.microsoft.com/en-us/library/ms809340.aspx

[DEY01] A. K. Dey, “Understanding and using context,” Personal and Ubiquitous Computing Journal, vol. 5(1), pp. 4–7, 2001.

[DOU05] C. Doulkeridis, N. Loutas, and M. Vazirgiannis, “A System Architecture for Context-Aware Service Discovery”, International Workshop on Context for Web Service, 2005.

[DUF04] A. Duffy and T. Dowling, “An Object Oriented Approach to an Identity Based Encryption Cryptosystem”, 8th IASTED International Conference on Software Engineering and Applications, 2004.

[DUR03] F Dur´an, J Herrador, and A Vallecillo “Using UML and Maude for Writing and Reasoning about ODP Policies”, Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks, POLICY 2003, Lake Como, Italy, 2003

[ELL02] C. Ellison, “Home Network Security,” Intel Technology J., vol. 6, pp. 37-48, 2002.

[ELL03] C. Ellison, “UPnP Security Ceremonies” V1.0, Intel Co., Oct. 2003.

[FIS01] S. Fisher-Hubner, “IT-Security and Privacy”, Srpinger-Verlag Berlin Heidelberg 2001

[GAN04] S. Ganeriwal and M. B. Srivastava, “Reputationbased framework for high integrity sensor networks,” in SASN ’04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks. ACM Press, pp. 66–77, 2004.

[GAR04] Luis Garcés-Erice, “A Hierarchical P2P Network :Design and Applications”, 2004

[GAR04-2] O. Garofalakis et al “Web Service Discovery Mechanisms: Looking for a Needle in a Haystack?” 15th ACM Conference on Hypertext and Hypermedia (Hypertext 2004), Santa Cruz, USA, 2004

[GHA04] M. Ghader et al “Secure resource and service discovery in personal networks” Wireless World Research Forum Meeting #12, Canada, 4-5 Nov 2004

[GNU] Gnutella http://www.gnutella.com

[GOL96] D. Goldschlag, M. Reed, and P. Syverson, “Hiding routing information”. In First International Workshop on Information Hiding, pp. 137–150, (1996).

[GOY06] V Goyal, et al, “Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data”, Proceedings of 13th ACM Conference on Computer and Communications Security (CCS 2006), Alexandria, USA, October 2006

[GRA00] Gryazin, Eugene A., “Service Discovery in Bluetooth”, Bluetooth Technology and Utilization Department of Computer Science, Helsinki University of Technology, Finland, presented Nov. 9, 2000.

[GU04] T Gu, X H Wang, H K Pung, D Q Zhang “An Ontology-based Context Model in Intelligent Environments“Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference,. San Diego, California, USA, pp. 270-275, 2004.

[HAN03] U. Hansmann, L. Merk, M. S. Nicklous, and T. Stober, “Pervasive Computing”, Springer-Verlag, 2003.

[HEN05] U. Hengartner, and P. Steenkiste, “Exploiting Hierarchical Identity-Based Encryption for Access Control to Pervasive Computing Information”. Proc. of First IEEE/CreateNet

International Conference on Security and Privacy for Emerging Areas in Communication Networks, Athens, Greece, pp. 384-393, 2005.

[HOE06] K. Hoeper, G. Gong, “Key Revocation for Identity-Based Schemes in Mobile Ad Hoc Networks”, in Proceedings of Ad-Hoc, Mobile, and Wireless Networks, Springer Berlin / Heidelberg, Volume 4104/2006, pp. 224-237, 2006.

[HON04] J. I. Hong and J. A. Landay, “An architecture for privacy-sensitive ubiquitous computing,” in MobiSYS ’04: Proceedings of the 2nd international conference on Mobile systems, applications, and services. ACM Press, pp. 177–189, 2004.

[HU04] J Hu, A C Weaver “A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications” First Workshop on Pervasive Privacy Security, Privacy, and Trust (PSPT2004), Boston, MA, USA, 2004.

[JEA03] K Jean, K Yang, and A Galis “A Policy Based Context-aware Service for Next Generation Networks” 8th London Communication Symposium, London, UK, 2003

[JES] Jess. Rule engine for java. http://herzberg.ca.sandia.gov/jess/

[JIA02] X Jiang, J A Landay “Modeling Privacy Control in Context-Aware systems” IEEE Pervasive Computing , Volume 1 , Issue 3 , pp. 59 – 63, 2002.

[JIA05] Z. Jiang, K. Lee, S. Kim, H. Bae, S.Kim, S. Kang, “Design of a security Management Middleware in ubiquitous Computing Environment”, In Proceedings of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2005), Dalian, China, 2005.

[JIN] SUN Microsystems, Jini Specifications, http://java.sun.com/products/jini/

[JUL04] C. Julien, J. Payton, and G. C. Roman “Context-Sensitive Access Control for Open Mobile Agent Systems”, in Proceedings of the 3rd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'2004), Edinburgh, Scotland (UK), May, pp. 42-48, 2004.

[KAG03] L. Kagal, T. Finin and A. Joshi “A Policy Language for a Pervasive Computing Environment”, Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks, POLICY 2003, Lake Como, Italy, 2003.

[KAT07] A. Kate, G. Zaverucha, and I. Goldberg. “Pairing-Based Onion Routing”, in proceedings of 7th Privacy Enhancing Technologies Symposium (PETS 2007), 2007.

[KAT08] Aniket Kate, Greg Zaverucha, and Ian Goldberg , “Pairing-Based Onion Routing with Improved Forward Secrecy”, Cryptology ePrint Archive, Report 2008/080, February 2008.

[KES98] D. Kesdogan, J. Egner, and R. Büschkes. "Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System," Information Hiding 1998, LNCS 1525, pp 83-98, Springer Heidelberg, 1998.

[KIA07] A. Kiayias, Y. Tsiounis, M. Yung: “Group Encryption”. In proceedings of 13th Annual International Conference on the Theory and Application of Cryptology & Information Security ASIACRYPT 2007: pp.181-199, 2007.

[LAN02] M Langheinrich “A Privacy Awareness System for Ubiquitous Computing Environments”, Proceedings of the 4th international conference on Ubiquitous Computing , Göteborg, Sweden, pp. 237 - 245,2002.

[LEE03] C. Lee and S. Helal, “Context Attributes: An Approach to Enable Context-awareness for Service Discovery”, in Proceedings of the Symposium on Application and the Internet (SAINT), 2003.

[LEE04] Y. Lee, S. A. Chun, and J. Geller. Web-based semantic pervasive computing services"". IEEE Computational Intelligence Bulletin, 4(2):4–15, 2004.

[LIS03] R. Liscano and A. Ghavam, "Context Awareness and Service Discovery for Spontaneous Networking", School of Information and Technology and Engineering (SITE), University of Ottawa, 2003

[LIT61] Little, J. D. C.: A Proof of the Queueing Formula L = λ W" Operations Research, 9, pp. 383-387, 1961.

[LOR03] M. Lorch, S. Proctor, R. Lepro, D. Kafura and S. Shah, "First Experiences Using XACML for Access Control in Distributed Systems", Presented at the ACM Workshop on XML Security, Fairfax, VA, USA, October 2003

[LUO04] H. L. Luo, M. Barbeau, "Performance Evaluation of Service Discovery Strategies in Ad Hoc Networks". In proceedings of the Second Annual Conference on Communication Networks and Services Research pp. 61-68, 2004.

[MAR03] D. Martin, OWL-S 1.0 Release, http://www.daml.org/services/owl-s/1.0/owl-s.html

[MAR06] B. Di Martino, "An Ontology Matching Approach to Semantic Web Services Discovery", Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops, Springer Berlin / Heidelberg, pp. 550-558, 2006.

[MAY02] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric". In Proceedings of the first International Workshop on Peer to Peer Systems IPTPS, Cambridge, MA, USA, 2002.

[MET] http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s

[MOS] European ist project MOSQUITO. http://www.mosquito-online.org.

[NAL02] P Naldurg, R H. Campbell, and M. D Mickunas "Developing Dynamic Security Policies" 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), San Francisco, CA, USA, 29-31 May 2002.

[NEI07] R. Neisse, M. Wegdam, M.van Sinderen, G. Lenzini, "Trust Management Model and Architecture forContext-Aware Service Platforms", In Proceedings of the 2nd International Symposium on Information Security (IS07), 2007.

[P3P] W3C, The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, available on http://www.w3.org/TR/P3P/ , April 2002.

[PAT04] A. Patwardhan, V. Korolev, L. Kagal and A. Joshi "Enforcing policies in Pervasive Environments", The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004, Boston, Massachusetts, USA, 2004.

[PIR06] M. Pirretti, P. Traynor, P.McDaniel, and B. Waters, "Secure Attribute-Based Systems", Proceedings of ACM Conference on Computer and Communications Security (CCS '06), Alexandria, VA, November 2006.

[PRE03] D. Preuveneers and J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers and K. De Bosschere. "Towards an extensible context ontology for ambient intelligence. In Second European Symposium on Ambient Intelligence, volume 3295 of LNCS, , Eindhoven, The Netherlands, pp. 148 – 159, 2004.

[RAV06] P-G. Raverdy, V. Issarny, R. Chibout, A. de La Chapelle. "A Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments", In Proceedings of

MOBIQUITOUS – The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services. San Jose, CA, USA, 2006,.

[RAV06-2] P.-G. Raverdy, O. Riva, R. Chibout, A. de La Chapelle and V. Issarny, "Efficient Context-aware Service Discovery in Multi- Protocol Pervasive Environments", In Proc. of IEEE Intl. Conference on Mobile Data Management (MDM), Tokyo,Japan, May 2006.

[REI98] M. Reiter, A. Rubin. Crowds: "Anonymity for Web Transactions," ACM Trans. on Information and Systems Security, pp 66-92, 1 (1) 1998.

[ROB03] P. Robinson and M. Beigl. “Trust Context Spaces: An Infrastructure for Pervasive Security”. In the First International Conference on Security in Pervasive Computing, 2003.

[ROM02] K. Romer, O. Kasten, F. Mattern, “Middleware Challenges for Wireless Sensor Networks”, ACM Mobile Computing and Communication Review, Vol. 6, No. 4, pp. 59-61, October 2002

[SAI05] F. Sailhan, V. Issarny, “Scalable Service Discovery for MANET”. In Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom'2005). March 2005.

[SAH05] A. Sahai and B. Waters, “Fuzzy Identity-Based Encryption”, Advances in Cryptology-Eurocrypt'05.LNCS 3494, pp. 457-473, Springer, 2005.

[SAL99] D. Salber, A.K. Dey and G.D. Abowd, "The Context Toolkit: Aiding the development of context-enabled applications", in Proceedings of the SIGCHI conference on Human factors in computing systems, (1999), pp. 434-441.

[SAN96] R. Sandhu, E. Coyne, H. Fienstein, and C. Youman, "Role Based Access Control Models," in IEEE Computer, vol. 29, 1996.

[SHA84] A. Shamir, “Identity-based cryptosystems and signature schemes", in Advances in Cryptology Crypto '84, Lecture Notes in Computer Science, Vol. 196, Springer-Verlag, pp. 47-53, 1984

[SHA02] N. Shankar and D. Bafanz, “Enabling secure ad-hoc communication using context-aware security services,” in UNBICOMP 02: Workshop on Security in Ubiquitous Computing, 2002.

[SHE07] K. Sheikh, M. Wegdam, M. van Sinderen, “Middleware Support for Quality of Context in Pervasive Context-Aware Systems”, In: Proceedings of the Fourth IEEE International Workshop on Middleware Support for Pervasive Computing (PerWare'07), co-located with Fifth Annual Conference on Pervasive Computing and Communications (PerCom 2007), 2007.

[SHI94] Schilit, B., Theimer, M. Disseminating Active Map Information to Mobile Hosts. IEEE Network, pp. 22-32, 1994.

[SOM07] N. Le Sommer, “A Framework for Service Provision in Intermittently Connected Mobile Ad hoc Networks”, In Proceedings of the 8th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM 2007), Helsinki, Finland, 2007.

[SSJ] P. L'Ecuyer, L. Meliani, J. G. Vaucher, “SSJ: a framework for stochastic simulation in Java”. Winter Simulation Conference, 2002.

[SWS] http://www.daml.org/services/swsf/1.0

[TAN02] A. S. Tanenbaum, M. Van Steen, “Disctributed Systems: Principles and Paradigms”, Prentice Hall, 2002.

[TON03] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, A. Uszok, “Semantic Web Languages for Policy Representation and Reasoning: A comparison of KAoS, Rei, and Ponder” International Semantic Web Conference (ISWC 03). Sanibel Island, Florida.

[TRA07] S. Trabelsi and Y. Roudier, "Secure service publishing with untrusted registries: Securing service discovery", SECRYPT 2007, International conference on Security and Cryptography, July 28-31, 2007, Barcelona, Spain

[TSA03] W.T. Tsai, R. Paul, Z. Cao, L. Yu, A Saimi, B. Xiao, "Verification of Web services using an enhanced UDDI server", Proceedings of the 18th International Workshop on Object-Oriented Real-Time Dependable Systems, Anacapri (Capri Island), Italy, 2003.

[VOL] Voltage IBE Toolkit http://www.voltage.com/ibe_dev/

[WAN03] X. F. Wang, M.K Reiter, "Defending against denial-of-service attacks with puzzle auctions", in proceeding Symposium on of Security and Privacy, USA, (2003).

[WEI91] M. Weiser, "The Computer of the 21st Century," Scientific American, vol. 265, no. 3, Sept. pp. 66–75, 1991.

[WES87] A. Westin, "Privacy and Freedom", New York, 1987

[WSDL] WSDL specifications http://www.w3.org/TR/wsdl

[WSM] http://www.wsmo.org

[XAC] OASIS, XACML, http://www.oasis-open.org/committees/xacml/

[YAN06] Y. Yang, H.Hassanein, A. Mawji, "Efficient Service Discovery forWireless Mobile Ad Hoc Networks", in proceedings of IEEE International Conference on Computer Systems and Applications, 2006.

[ZHU03] F. Zhu, M. Mutka, and L. Ni, "Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services", in Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (Percom'03). IEEE Computer Society, pp. 235–242, 2003.

[ZHU04] F. Zhu, M. Mutka, L. Ni "Prudent exposure: A private and user centric service discovery protocol" Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom'04) Orlando, USA, 2004

[ZHU05] F. Zhu, M. Mutka, L. Ni, "Service Discovery in Pervasive Computing Environments", Proceedings of Pervasive Computing, IEEE, Vol. 4, No. 4, pp. 81-90. 2005.

[ZUI04] M. Zuidweg, J. G. P. Filho, M. Van Sinderen, "Using P3P in a web services-based context-aware application platform" Proceedings of the 6th international conference on Electronic commerce table of contents Delft, The Netherlands, pp. 376 – 381, 2004.

Annex

In this annex document we present the specifications of the secure service discovered implementation that we developed during my thesis work. This specification contains a detailed API description, the diagram classes describing the developed components, the sequence diagram describing the interactions between different elements of the system and finally a WSDL description file providing information about the web service interface of our secure service discovery system.

1. API description

The token supports dynamic discovery: when it is plugged into a device, it announces itself as a discovery proxy, registering some personal services. This interface is mainly based on WS-Discovery.

1.1. General Features:

The token acts both as a Target Service, announcing itself as a Discovery Proxy and as a Discovery Proxy:

- IP multicast messages supported:
  - Sending:
    - Hello (SOAP/UDP)
      - Both Types DiscoveryProxy and TargetService
      - Hello messages sent periodically
- IP multicast messages not yet supported:
  - Receiving:
    - Probe
    - Resolve
- Messages over HTTP
  - Sending:
    - Bye
    - ProbeMatch
    - ResolveMatch

With each matching endpoint reference, the associated metadata includes the credentials to access to the service

  - Receiving:
    - Probe
      - Scope rules support to be defined
    - Resolve

The token does not support compact signature.
Secure registration of target services:

- Through HTTPS, with client authentication
- Registration messages from Liberty Alliance Discovery Specification 2.0
  - Receiving
    - Modify.

The ModifyDiscovery operation allows multiple insertions and removals to be made in a single request. The operation is atomic; the insertions and removals either all succeed or all

fail. Each entry is defined as an End Point Reference (EPR) according to WS-Addressing specification.

- Sending
  - ModifyResponse

The message ModifyResponse includes a status response and the new ids of the eventual inserted entries.

1.2. Interface Definition

1.2.1. Parameters

- ServiceProfile: A structure describing the profile of a service. This structure is composed in on seven optional and extendible attributes that are :
  - Type: describes the type of the offered service (printer, file sharing, weather, …)
  - Owner: represents the identity of the user that proposes the service.
  - Domain: an indication about the domain in witch belongs the service
  - Organisation: the name of the organisation (or the business) that proposes the service
  - LifeTime: an indication about the life time of the service after this delay the service is no more available
  - Qos: We can add an indication about the quality of service provided by the service.
- EndPointReference: A structure containing the information that are useful to join the service and composed on two attributes:
  - Address: also called UUID that represents an unique identifier, but this address does not provide an indication about the end point of the service.
  - ReferenceProperties: provides an URL or a routable address where we can access to the service

- PublishType: a structure that contains a service profile and an EndPointReference used to register a service
- LookupResponseType: structure that contains a service profile and an EndPointReference used as a response to the service requester query
- ServiceID: it is an identifier affected by the registry to the services to distinguish between them, this identifier is used by the service in order to update or unregister itself.
- ClientPolicy / ServicePolicy: contains a metadata representing the client (server) policy

1.2.2. Methods

- Publish: Method used for the service publication
  - Input parameter : PublishType
  - Output parameter: serviceID
- Lookup: Method used by the client to discover to make a service discovery request
  - Input parameter : ServiceProfile
  - Output parameter: an array of LookupResponseType
- Unregister: Method used by the service to unregister from the registry

- Input parameter : ServiceID
- SetServicePolicy / SetClientPolicy: Methods to set security policies
  - Input parameter : ServicePolicy / ClientPolicy

2. UML specifications

The discovery mechanism code has been developed in Java and is organized along several sub-packages.

2.1. External API

The JavaAPI sub-package contains classes providing the Java mapping of the WSDL interface. These classes are organized as follows (see Figure 67):

- The Discovery Java interface provides the operations necessary to access a registry
- Data structures are provided to store the service profile, for the registration of a service as well as for its lookup (incomplete profile in this case)

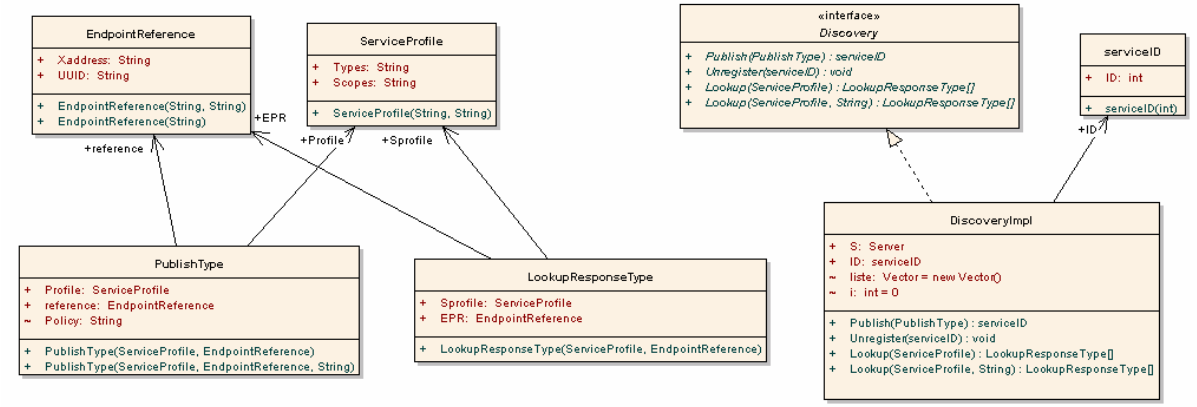


Figure 67: Java API to Service Discovery (UML class diagram)

2.2. Communication related data structures

Data structures related to communications and protocol are located in two sub-packages:

- The Data sub-package contains data structures made to handle the sending and reception of WS-Discovery messages
- The Messages sub-package (see Figure 68) describes the structure of WS-Discovery messages

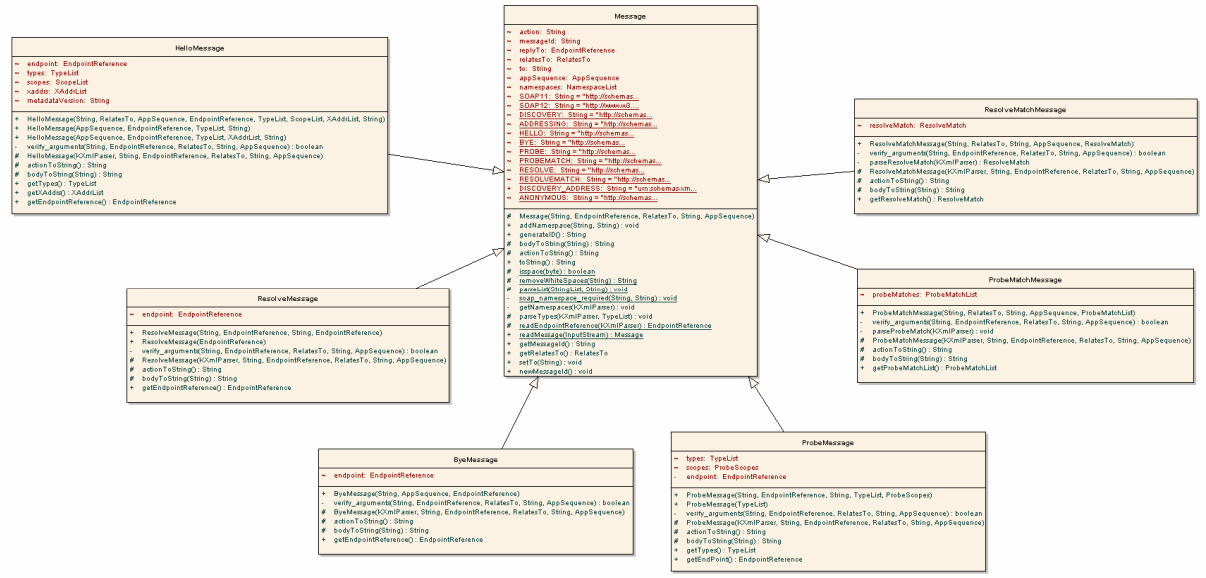


Figure 68: Messages sub-package (UML class diagram)

2.3. Policy handling

Policy specifications are addressed in the Policy sub-package (see Figure 69). It describes:

- the discovery policy decision point interface; the PDP functionality is called by the MOSQUITO registry (Proxy) following the process described in Figure 70;
- helper classes to construct:
  - the access control policy to the service description (used by service providers;
  - a XACML request from the MOSQUITO registry PEP to the PDP (this structure is necessary since the client request, being a discovery, does not contain an actual reference to a managed resource)

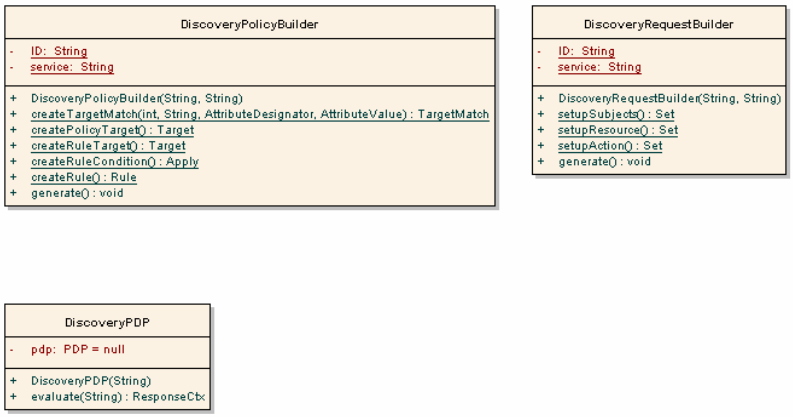


Figure 69: Policy sub-package (UML class diagram)

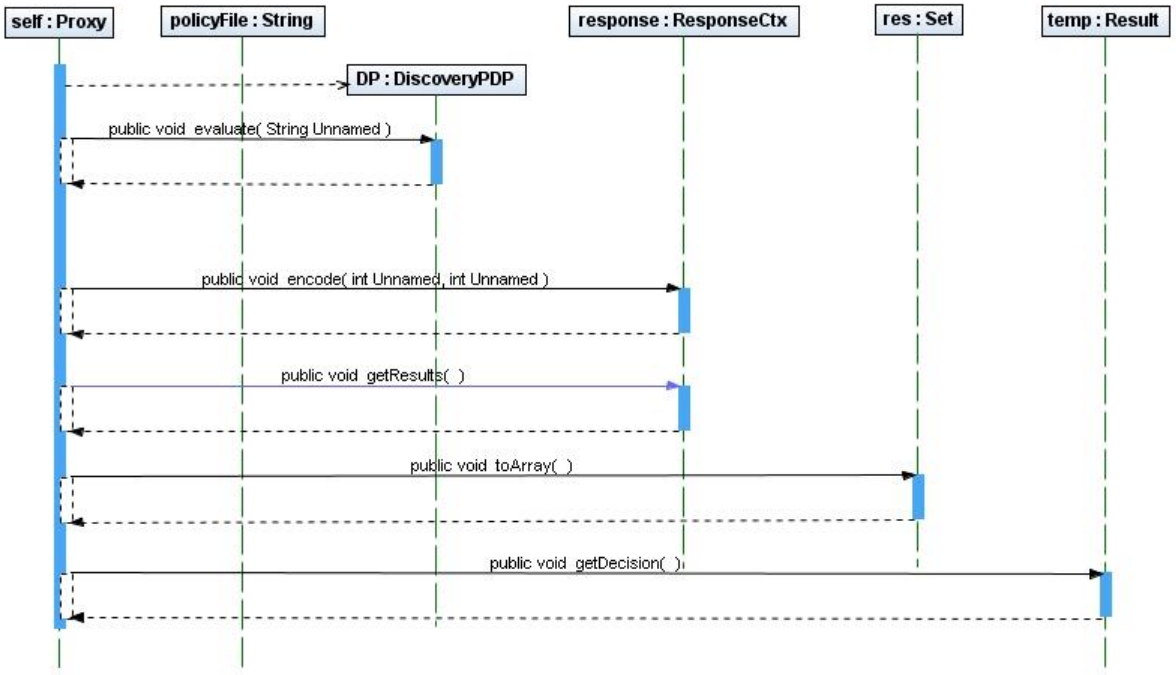


Figure 70: registry - Policy enforcement process (UML sequence diagram)

2.4. Protocol implementation

The Discovery sub-package (see Figure 72) contains classes that implement the WS-Discovery protocol and its extensions. The most important classes in this sub-package are as follows:

- The Proxy class implements a registry: this means it is both a regular WS-Discovery proxy and a (trusted) discovery policy enforcement point. Figure 73 describes how the service policy is retrieved and stored with the service description. It also describe the more complex process of client probe request retrieval and processing (matching and discovery policy enforcement).
- The GenericClient class is used as a launcher for clients to retrieve information from looked up services (opening a unicast socket to a specified registry, generating and sending probe messages, and handling probe match messages)
- Service launchers are provided by classes Server (multicast-able service) and UnicastServer (unicast case only – necessarily registered with a registry). The use of such a launcher together with the construction of a policy is illustrated by the sequence diagram of Figure 71.

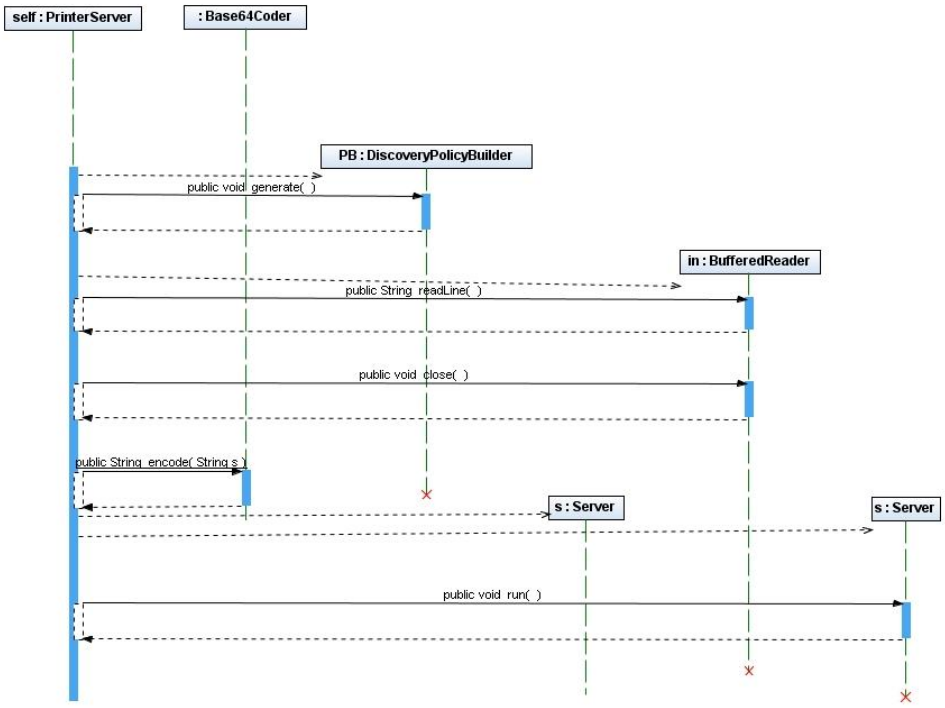


Figure 71: Use of a service launcher (UML sequence diagram)

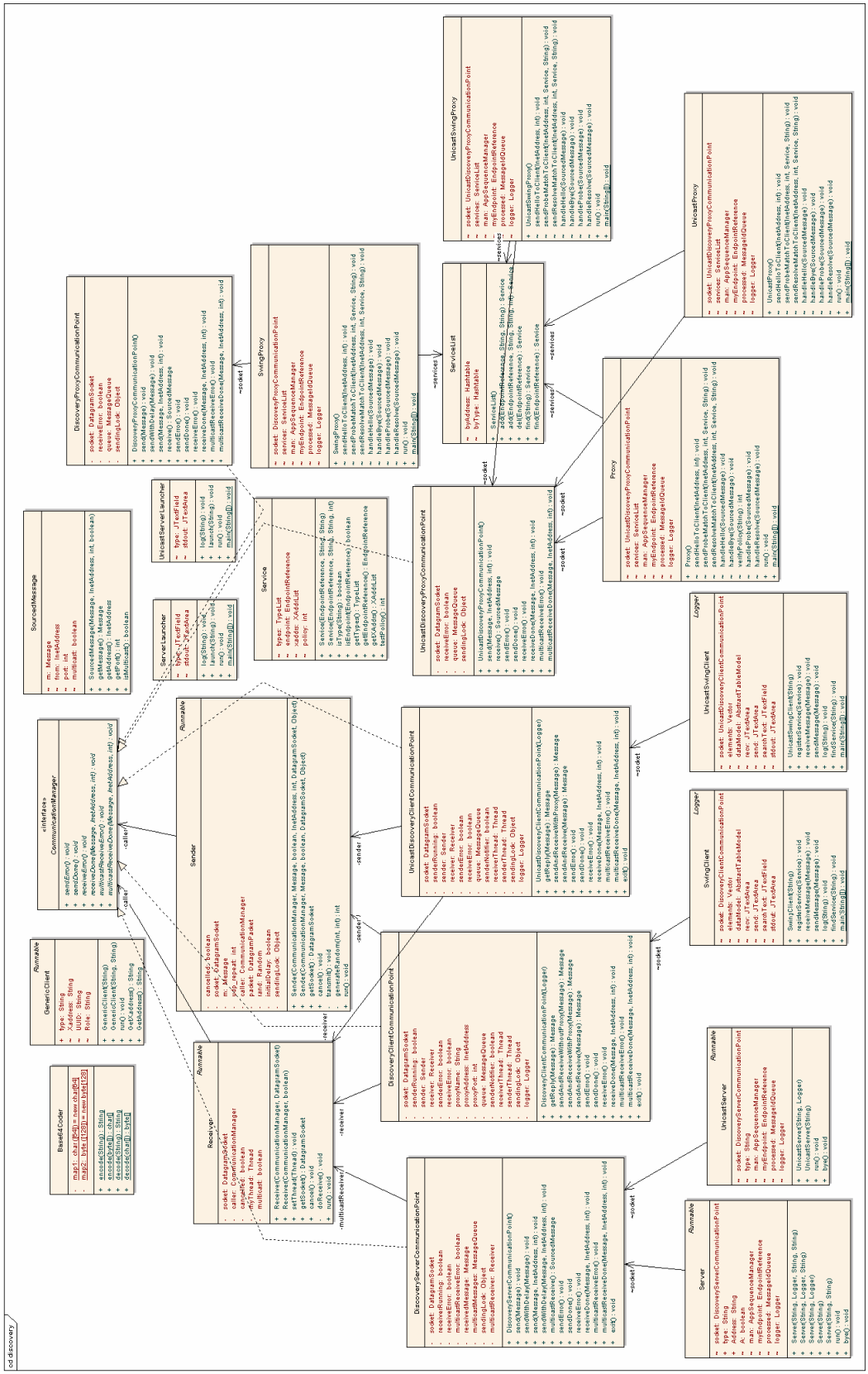


Figure 72: Discovery sub-package – protocol implementation (UML class diagram)

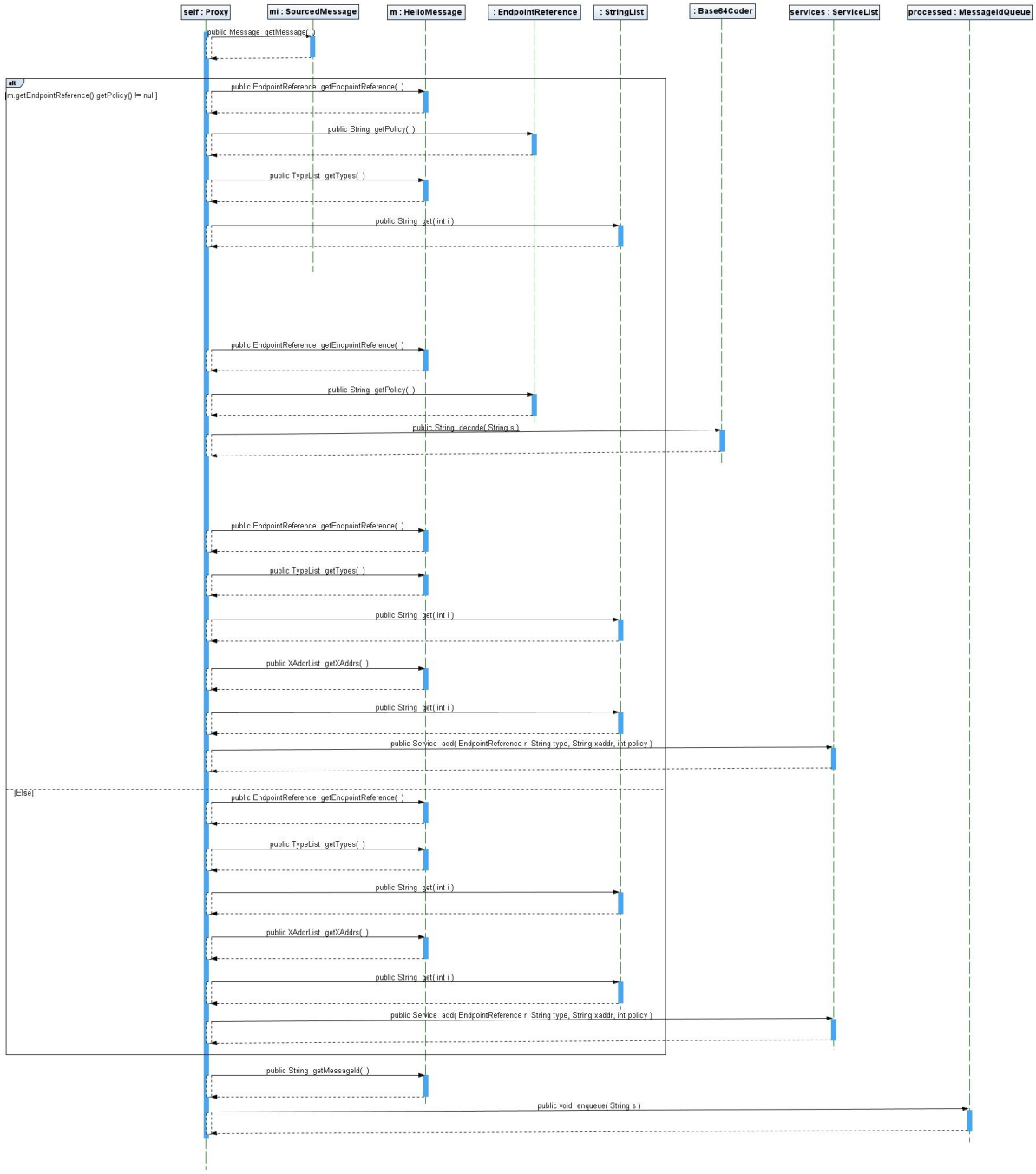


Figure 73: MOSQUITO registry - Service policy retrieval and storage (UML sequence diagram)

3. WSDL interface specification

The discovery mechanism exposes the LAN registry interface to potential clients and services that respectively want to send lookup or registration requests. The corresponding interface description is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://JavaAPI" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://JavaAPI" xmlns:intf="http://JavaAPI"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<wsdl:types>
 <schema targetNamespace="http://JavaAPI" xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <complexType name="ArrayOfLookupResponseType">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="impl:LookupResponseType[]"/>
    </restriction>
   </complexContent>
  </complexType>
 </schema>
</wsdl:types>

  <wsdl:message name="PublishRequest">
     <wsdl:part name="in0" type="xsd:anyType"/>
  </wsdl:message>

  <wsdl:message name="UnregisterResponse">
  </wsdl:message>

  <wsdl:message name="LookupResponse">
     <wsdl:part name="LookupReturn" type="impl:ArrayOfLookupResponseType"/>
  </wsdl:message>

  <wsdl:message name="UnregisterRequest">
     <wsdl:part name="in0" type="impl:serviceID"/>
  </wsdl:message>

  <wsdl:message name="LookupResponse1">
     <wsdl:part name="LookupReturn" type="impl:ArrayOfLookupResponseType"/>
  </wsdl:message>

  <wsdl:message name="LookupRequest1">
     <wsdl:part name="in0" type="impl:ServiceProfile"/>
  </wsdl:message>

  <wsdl:message name="LookupRequest">
     <wsdl:part name="in0" type="xsd:anyType"/>
     <wsdl:part name="in1" type="soapenc:string"/>
  </wsdl:message>

  <wsdl:message name="PublishResponse">
     <wsdl:part name="PublishReturn" type="xsd:anyType"/>
  </wsdl:message>

  <wsdl:portType name="Discovery">
     <wsdl:operation name="Publish" parameterOrder="in0">
        <wsdl:input message="impl:PublishRequest" name="PublishRequest"/>
        <wsdl:output message="impl:PublishResponse" name="PublishResponse"/>
     </wsdl:operation>
     <wsdl:operation name="Unregister" parameterOrder="in0">
        <wsdl:input message="impl:UnregisterRequest" name="UnregisterRequest"/>
        <wsdl:output message="impl:UnregisterResponse" name="UnregisterResponse"/>
     </wsdl:operation>
     <wsdl:operation name="Lookup" parameterOrder="in0 in1">
        <wsdl:input message="impl:LookupRequest" name="LookupRequest"/>
        <wsdl:output message="impl:LookupResponse" name="LookupResponse"/>
     </wsdl:operation>
     <wsdl:operation name="Lookup" parameterOrder="in0">
        <wsdl:input message="impl:LookupRequest1" name="LookupRequest1"/>
        <wsdl:output message="impl:LookupResponse1" name="LookupResponse1"/>
     </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="mosquitoSoapBinding" type="impl:Discovery">
     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
     <wsdl:operation name="Publish">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="PublishRequest">
           <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                          namespace="http://JavaAPI" use="encoded"/>
        </wsdl:input>
        <wsdl:output name="PublishResponse">
           <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                          namespace="http://JavaAPI" use="encoded"/>
        </wsdl:output>
     </wsdl:operation>
     <wsdl:operation name="Unregister">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="UnregisterRequest">
           <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

```
                        namespace="http://JavaAPI" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="UnregisterResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://JavaAPI" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="Lookup">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="LookupRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://JavaAPI" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="LookupResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://JavaAPI" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="Lookup">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="LookupRequest1">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://JavaAPI" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="LookupResponse1">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://JavaAPI" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="DiscoveryService">
      <wsdl:port binding="impl:mosquitoSoapBinding" name="mosquito">
         <wsdlsoap:address location="http://localhost:8080/mosquito"/>
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

4. Installation and usage guidelines

4.1. Installation

In order to use the Secure Service discovery application, you have to install

- the sunxacml-1.2 binary archive[9]
- the Kxml2 binary archive[10]

and add the following JAR files into your classpath: home_root\sunxacml-1.2\lib\samples.jar, home_root\sunxacml-1.2\lib\sunxacml.jar, home_root\sunxacml-1.2\lib\sunxacml-debug.jar, home_root\kxml2\kxml2-2.2.2.jar, home_root\kxml2\xmlpull_1_1_3_4b.jar

In order to externalise some parameters we used three parameter files that must be copied on the project root:

- *proxy.properties*: intalled on the proxy side and contains two parameters:
  - Proxy.policyPath : used to set the path directory where the proxy could store the security policies of the servers
  - Proxy.requestPath: used to set the path directory where the proxy could store the policy requests related to the clients service query
- *policyGenerator.properties*: installed on the server side, it contains one parameter:
  - DiscoveryPolicyBuilder.policyPath: this path is used as a policy store of the service. When a server generates un new service policy, it will be stored there. Then when a service will publish its policy it will extract the file from this

[9] http://sourceforge.net/project/showfiles.php?group_id=73884&package_id=74038&release_id=253638
[10] http://sourceforge.net/project/showfiles.php?group_id=9157&package_id=58653

directory path.

Unicast.txt: installed on the client and service sides and contains the IP address of the proxy (this will enable the unicast communication discovery)

4.2. Usage

To get a first impression of the library usage:

- Run a proxy object (the registry starts)
- Run class test (from the fr/eurecom/mosquito/test directory)

If something does not work, check that the content of class test code has correct parameters (e.g. at least one service started, then the client run).

To get a better understanding of the discovery protocol (or to demonstrate the messages that are exchanged), protocol loggers for the client and registry were developed and can be launched to see the messages exchanged between parties. In particular, this makes it possible to visualize message duplication because of multicast sending. One should simply start one of the following classes: SwingClient, UnicastSwingClient, SwingProxy, or UnicastSwingProxy (see Figure 72) depending on the version of the extended WS-Discovery protocol used.