Université Saâd DAHLAB de Blida

Faculté des Sciences

Département : Informatique

Mémoire Présenté par :

El-bey Amina et Hassein-Bey Lamia Safia

En vue d'obtenir le diplôme de Master

Domaine : mathématique et informatique

Spécialité : informatique

Option : ingénierie du logiciel

Sujet : Conception et réalisation d'un composant de sécurité

Sous la direction du promoteur

Mr. Djamal BENNOUAR

2011/2012

Remerciements

Tout d'abord nous remercions Dieu pour nous avoir guidés vers le bon chemin de la lumière et du savoir, pour nous avoir donné du courage et de la volonté afin de pouvoir réaliser ce mémoire.

Nous remercions notre promoteur Mr Djamal Bennouar pour le temps qu'il nous a consacré et les précieux conseils qu'il nous a prodigué tout au long de notre travail.

Nous remercions notre encadreur Mlle Faiza Farhi pour avoir bien voulu nous encadrer et orienter dans l'élaboration de ce travail.

Nous remercions Mr N.Idirene pour son aide dans le domaine de la sécurité informatique.

Nous remercions laboratoire Fundapel de nous avoir ouvert ces portes et plus précisément l'équipe de « physiques appliqué au micro et nano system ».

Nous remercions tous les professeurs pour nous avoir formés depuis notre premier cycle d'étude, jusqu'à la fin, ainsi les membres du jury qui ont accepté de lire et évaluer notre travail.

Enfin, nous adressons nos vifs remerciements à toutes les personnes qui de près ou de loin nous ont suivi et aidés à la réalisation de ce travail.

Dédicace

Je dédie ce modeste travail à mes chers parents, pour leurs soutiens et encouragement.

A la mémoire de ma sœur Fattouma.

A mes adorables frères, sœurs pour leurs patiences.

A mes amies Hadjer, Meriem et Lamia

A toute ma grande famille, et toutes les personnes qui me connaissent de près ou de loin.

Amina

Je dédie ce modeste travail à mes chers parents, pour leurs soutiens et encouragement.

A mes adorables frères, sœurs pour leurs patiences.

A mes amies Meriem et Amina

A toute ma grande famille, et toutes les personnes qui me connaissent de près ou de loin.

Lamia

Résumé :

La réalisation de n'importe quelle application logicielle nécessite de prendre en considération l'aspect de la sécurité. Au niveau de chaque application, il y a souvent la re-conception et réalisation de cet aspect technique, ce qui le rend car une fois réalisé, fortement lié à l'application et non réutilisable.

Les approches à composant apportent une réponse intéressante au problème de la réutilisation. C'est dans ce contexte que s'inscrit ce travail qui vise la conception et la réalisation d'un composant de sécurité permettant l'authentification des acteurs d'une application, la gestion très fine des droits d'accès, l'aide à la mise en place de solution de sécurité particulière, la signature et l'authentification de documents numérique. Ce composant sera destiné à être intégrée dans les applications ordinaires orientée objet ou dans les applications orientées composant.

Mots clés : sécurité, authentification, droits d'accès, Orienté composant, composant, réutilisation, architecture logiciel, Langage de description d'architecture ADL.

Abstract:

The realization of any software application requires taking into consideration the aspect of security. In each application, there is often re-design and production of this technical aspect, making it because once realized, strongly linked to the application and not reusable.

Component approach provides an interesting answer to the problem of reuse. It is within this context that this work involves the design and implementation of a safety component to authenticate application actors, the very precise management of access rights, aid to the establishment of special security solution, the signing and authentication of digital records. This component is intended to be integrated in applications in ordinary object-oriented or component-oriented applications.

Keywords: security, authentification, access rights, component-oriented, component, reuse, software architecture, Architecture Description Language (ADL).

ملخص:

لانجاز أي برنامج يتطلب الأخذ الاعتبار جوانب الأمن ,وفي كل مرة يعاد دراسة ,تصميم وتنفيذ لهذا الجانب التقني لان بعد اتمام انجازه يكون تطبيقه مرتبط بقوة مع هذا البرنامج.

مناهج المركبات تقديم إجابة مثيرة للاهتمام لإشكالية إعادة الاستخدام , ومن ضمن هذا السياق ينطوي هذا العمل على التصميم والتنفيذ مركب امن للمصادقة على هوية مستعملين البرنامج , وإدارة دقيقة للغاية لحقوق الوصول لمختلف وظائف البرنامج , والمساعدات للإنشاء حل أمني خاص , والتوقيع والمصادقة على السجلات الرقمية , ويهدف هذا المكون إلى دمجه في التطبيقات العادية اي التطبيقات الموجهة للكائن أو الموجهة للمركب.

الكلمات المصنفة : البرمجيات المركبة,القطع البرمجية , الواصل , المنافذ , اللغة الفعلية.

Sommaire

Liste des figures :

Chapitre 4 « Réalisation et test »

Introduction générale

La sécurité en général, l'authentification et le contrôle d'accès en particulier, représentent des aspects techniques que nécessitent les diverses applications pour leur mise en œuvre. Nous remarquons que cet aspect est à chaque fois reconçu et réalisé lors de la conception et la réalisation de chaque application. Malheureusement, ces conceptions et réalisations de l'aspect sécurité sont très liées à la logique des logiciels à sécuriser et sont souvent réalisées pour couvrir des aspects bien particuliers nécessités pour la sécurité des applications. Ainsi, ces réalisations se trouvent confrontées à un problème crucial de réutilisation. Il faut aussi noter le fait important, que lors de la conception et la réalisation d'une application, l'aspect de sécurité est souvent pris en charge par des spécialistes du métier de l'application et non pas de spécialiste en sécurité d'applications informatique. La conséquence de cette situation est que l'aspect sécurité réalisé ne sera pas d'une grande qualité et d'une grande fiabilité.

Dans ce même ordre d'idée, ceux qui ont plus de chances et un budget assez suffisant, engagent des spécialistes pour réaliser cet aspect de sécurité. Cependant, une fois réalisé, malgré sa fiabilité et sa grande qualité, l'aspect sécurité se trouvera fortement lié à l'application, ce qui entraînera la grande difficulté de sa réutilisation et nécessitera ainsi de le reconsidérer pour chaque application.

Cette situation pourrait être expliquée par le fait, qu'actuellement la plupart des processus de conception se basent sur le concept d'objet qui ne favorise pas la séparation nette des préoccupations. Même si parfois au niveau conceptuel, le processus préconise la nécessité de la séparation des préoccupations, au niveau formalisation de la conception (en UML par exemple) et au niveau technologique (langage de programmation), les approches permettant de séparer et tisser correctement ces préoccupations (approches orientées aspects) ne sont pas mise en œuvre pour obtenir le système final. Ceci pourrait être du au degré de maturité de

ces approches et aussi à la difficulté de changer les mentalités des équipes qui ont travaillé assez longtemps dans le contexte de traditions difficilement remplaçable.

Les premières approches introduites pour la réutilisation d'aspects techniques important tels que la persistance, le traçage, la gestion des transactions, et la sécurité sont les approches à composant dit industriel tels que les Entreprise Java Bean et les Corba Component Model. Les composants industriels sont en réalité des objets qui sont instanciés dans des conteneurs par des fabriques. Les conteneurs sont souvent appelés des serveurs d'application. L'exécution des composants industriels se fait dans le contexte du conteneur. Les aspects techniques tels que la sécurité sont alors fournis par le conteneur. Ainsi, les concepteurs d'applications à base de composants industriels, concentreront leurs efforts sur la mise en place de composants métiers qui pourront tirer profit des aspects technique de haute qualité fournis par les conteneurs.

Dans les approches à composant industriels, la réutilisation des aspects technique ne peut se faire que dans le contexte des conteneurs spécifique et d'une forme de déploiement de l'application (ou architecture de déploiement) bien précise. Ainsi si une application a été réalisé dans le contexte du serveur JBOSS, il n'est pas sur qu'elle sera facilement transportable sur un serveur JONAS, malgré le fait que ces deux serveurs se disent compatible avec la norme J2EE. La forme de déploiement des applications utilisant les composant industriel est particulière et est dite architecture à 4 niveaux : navigateur web, serveur web, serveur d'application, serveur de base de données.

Dans la réalité, les applications ne sont pas toutes des applications qui exploitent les serveurs d'applications. De ce fait pour assurer une réutilisation plus efficace des aspects technique, il est nécessaire de rechercher une approche qui devrait être indépendante de la forme de déploiement de l'application.

Actuellement, l'approche qui semble être la plus indépendante de la forme de déploiement et de la technologie d'implémentation est l'architecture logicielle. L'architecture logicielle est une approche à composant. La conception se fait par assemblage de composant. Un composant possède un modèle de base qui

permet d'indiquer comment un composant doit être vu par ses utilisateurs. Le modèle de composant peut aller encore plus loin en fixant une style de conception de sa vue interne. Dans ce cas nous parlons souvent d'architecture, ou de configuration ou de composant composite.

En architecture logicielle, notamment en Architecture logicielle orientée aspect, un composant est par définition une entité réutilisable, dont la conception est totalement indépendante de l'application dans lequel il sera utilisé. Parfois nous parlons même de composant inconscient de son environnement (obliviousness component).

Un composant exprime de manière explicite des interfaces fournies et des interfaces requises. Les interfaces fournies sont implémentées par le composant lui-même. Les interfaces requises représentent un ensemble de fonctionnalité réalisé par d'autres composants. La connexion des interfaces requises d'un composant aux interfaces fourni d'un autre composant permettra de faire fonctionner le premier composant.

Objectifs :

L'objectif de notre travail se concentre sur la réalisation d'un composant de sécurité fortement autonome, indépendant de toute application et facilement réutilisable. Le composant à concevoir et à réaliser doit être conforme au modèle de composant utilisé en Architecture Logicielle. Ainsi, ce composant devra explicitement exposer une interface qui indiquera les services fournis par le composant et les services requis.

La sécurité étant un domaine assez vaste, le composant à réaliser devra supporter dans sa première version, les fonctionnalités les plus usuelles en termes de sécurité notamment l'authentification et la gestion des droits d'accès. Le composant devra aussi fournir des facilité pour aider à mette sur pied des politiques de sécurité spécifique, notamment dans le contexte de la généralisation d'internet et le passage au tout numérique. C'est ainsi qu'il devra fournir des facilités permettant de réduire les accès douteux aux sites web, et des fonctionnalités de signature de document et d'authentification de document numérique.

Organisation du mémoire :

Le présent mémoire est structuré en cinq chapitres comme suit :

Le chapitre 1 : « sécurité informatique », dans lequel on aborde les définitions fondamentales des notions et des mécanismes de la sécurité.

Le chapitre 2 : « architecture logicielle », représente une étude détaillée de l'architecture logicielle.

Le chapitre 3 : « analyse des besoins», comprend l'étude de toutes les fonctionnalités du composant à réaliser.

Le chapitre 4 : « conception », il représente la conception des différents besoins déterminées dans le chapitre précédant.

Le chapitre 5 : « réalisation et test », il définit les différentes techniques utilisées pour implémenter le composant de sécurité ainsi leur tester dans une autre application.

Chapitre 1 :

Sécurité informatique

I. Introduction :

Les enjeux de la sécurité des systèmes d'information, avec l'omniprésence de l'informatique, la mise en ligne des systèmes et l'augmentation du nombre d'utilisateurs, sont devenus considérables.

Le contrôle d'accès, ou autorisation, c'est-à-dire le mécanisme qui définit et impose ce qu'il est permis et interdit de faire est un outil technique et organisationnel incontournable lorsqu'on envisage de garantir la sécurité d'un système. Ce domaine de recherche traite de multiples problématiques relatives à la notion de droit dans un système : structuration, formalisation, sémantique, stockage, représentation ou encore vérification et raisonnement.

Dans ce présent chapitre nous découvrons les problématiques de la sécurité, plus précisément l'authentification et le contrôle des droits d'accès dans les systèmes d'information. Nous présentons également les objectifs, nos motivations.

II. Concepts de la sécurité :

1) Exigences fondamentales :

Les principes de la qualité incitent les concepteurs, développeurs et autres acteurs à prendre en compte les propriétés non fonctionnelles dans le développement des systèmes d'information. Une de ces propriétés est la sécurité, la propriété qui permet aux utilisateurs d'avoir une confiance justifiée dans le système et dans les services qu'il délivre. Cette notion dépasse largement le cadre des systèmes d'information informatisés, et couvre des domaines allant de la sécurité physique à la sécurité logique.

La propriété de sécurité peut être analysée selon plusieurs sous-propriétés :

A. **Disponibilité** : désigne le fait que les données considérées sont accessibles au moment voulu par les utilisateurs autorisés.

B. **Intégrité** : consiste à déterminer si les données n'ont pas été altérées durant la communication (de manière fortuite ou intentionnelle).

C. **Confidentialité** : consistant à assurer que seules les personnes autorisées aient accès aux ressources échangées.

D. **Authentifications :** L'authentification a pour but de vérifier l'identité dont une entité (personne ou machine) se réclame. L'authentification est toujours précédée ou combinée avec une identification qui permet à cette entité de se faire reconnaître du système par un élément dont on l'a doté : un identifiant.

E. **Autorisation** : synonyme de contrôle d'accès, définit et impose ce qu'il est permis et interdit de faire.

F. **Non répudiation** : les données publiées de façon authentique sont certifiées, et leur auteur ne peut pas nier les avoirs publiées, il en assume la responsabilité. [LC05]

2) Etude des risques :

Les coûts d'un problème informatique peuvent être élevés et ceux de la sécurité le sont aussi. Il est nécessaire de réaliser une analyse de risque en prenant soin d'identifier les problèmes potentiels avec les solutions avec les coûts associés. L'ensemble des solutions retenues doit être organisé sous forme d'une politique de sécurité cohérente, fonction du niveau de tolérance au risque. On obtient ainsi la liste de ce qui doit être protégé.

Voici quelques éléments pouvant servir de base à une étude de risque:

A. Deni de service :

Le "Denial-of-service" ou déni de service est une attaque très évoluée visant à rendre muette une machine en la submergeant de trafic inutile. Il peut y avoir plusieurs machines à l'origine de cette attaque qui vise à anéantir des

serveurs, des sous-réseaux, etc. D'autre part, elle reste très difficile à contrer ou à éviter.

D'une manière générale, on parle de déni de service quand une personne ou une organisation est privée d'un service utilisant des ressources qu'elle est en droit d'avoir en temps normal. On trouvera par exemple des dénis de service touchant le service de courrier électronique, d'accès à Internet, de ressources partagées. [DS02]

B. Analyse passive du trafic :

Ce type d'attaque permet d'obtenir un certain nombre d'informations aussi bien sur la partie authentification de la communication que sur la partie interactive. Ces attaques requièrent la possibilité d'écouter le trafic réseau entre un ou plusieurs serveurs et clients SSH.

Au niveau de la partie authentification, il est possible de déterminer les méthodes d'authentification utilisées, les longueurs du nom de l'utilisateur et de son mot de passe, le nombre de clés RSA autorisées, ainsi que les succès, les échecs et les refus des authentifications RSA.

Avec cette information il est possible de déterminer lorsque l'utilisateur tape une commande ou lorsqu'il entre un mot de passe. La différenciation entre ces deux moments. [DE10]

C. Attaque par rejeu :

Les attaques par rejeu de session sont relativement simples si l'attaquant à la possibilité de voler le cookie de session. En effet, une des attaques les plus en vogue du moment se nomme " XSS " ou attaque de Cross Site Scripting. Beaucoup de personnes qui travaillent au sein d'une cellule sécurité connaissent ce fléau qui touche un très grand nombre de sites web. Le principe est que le pirate s'attaque à l'utilisateur plutôt qu'à l'application.

La mise en œuvre est relativement simple. L'attaque consiste à trouver un paramètre non filtré (c'est-à-dire non contrôlé par le serveur web) afin d'injecter un code Javascript. Une URL sera spécialement conçue par

l'attaquant puis envoyée à la victime pour que ce dernier suive le lien malveillant. [DE10]

D. Attaque par l'homme au milieu :

L'attaque Man In The Middle - ou homme du milieu (HDM)- consiste pour un tiers à s'interposer dans une communication sans que les parties concernées n'en aient conscience. Il existe en réalité plusieurs méthodes d'attaque dites Man In The Middle.

En cryptographie, il s'agirait ainsi pour l'assaillant de parvenir à lire, adresser et modifier des messages chiffrés entre deux parties de manière transparente pour ces deux dernières. Pour cela, il devra préalablement écouter le trafic réseau et intercepter les paquets IP.

La technique man in the middle peut par conséquent s'appliquer dans un système de chiffrement par clefs publiques, ainsi que dans le protocole d'échange de clefs Diffie-Hellman lorsque celui-ci est utilisé sans authentification. [DE10]

E. Tromperie :

L'attaque par tromperie est une attaque plus sophistiquée que l'attaque par écoute du réseau : une personne ou une machine malveillante se fait passer pour un « vrai » client auprès du fournisseur de service. La machine malveillante interroge le fournisseur avec les informations d'un « vrai » client et récupère des informations, qui peuvent être confidentielles. La Figure 1.1 synthétise le fonctionnement de cette attaque.

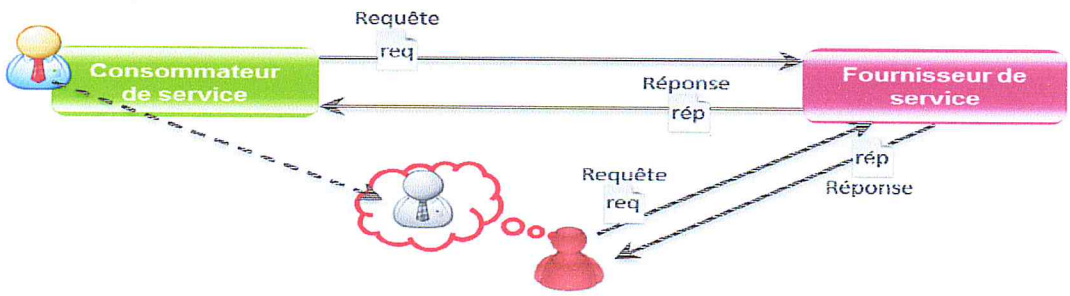


Figure 1.1 : Tromperie sur l'identité du client

Pour cette attaque, la machine malveillante émet des messages en volant l'identité d'un «vrai » client. Les messages sont donc échangés entre le fournisseur et la machine malveillante sans que le fournisseur se rende compte de la supercherie. Ce type d'attaque peut se faire après avoir écouté longtemps le réseau pour utiliser les informations récupérées ou en piratant les informations du client sur sa machine.

Pour que le fournisseur de service soit sûr de l'identité de son client, il faut mettre en place un système d'authentification qui permette de reconnaître les « vrais » clients des pirates. En plus de ce système d'authentification, il faut que les informations d'authentification ainsi que les autres informations échangées restent confidentielles entre le client et le fournisseur pour qu'il n'y ait pas de vol ni d'écoute par une machine malveillante.

F. Détournement d'informations :

Le détournement d'informations est une attaque qui a des effets différents par rapport aux deux attaques présentées précédemment. En effet, les informations du consommateur sont détournées par une personne ou une machine malintentionnée et elles sont modifiées à des fins malveillantes.

Le principe de cette attaque est d'altérer les données du consommateur afin d'endommager le service du fournisseur. La personne ou la machine malintentionnée intercepte un message d'un « vrai » client s'adressant au fournisseur. Elle modifie le message initial en y insérant du code malveillant.

Le message est ensuite envoyé au fournisseur de service qui l'interprète comme un message provenant du client qu'il connaît. Le code ajouté peut, par exemple, être un virus qui endommage le fournisseur de service. La Figure 1.2 illustre le scénario de cette attaque

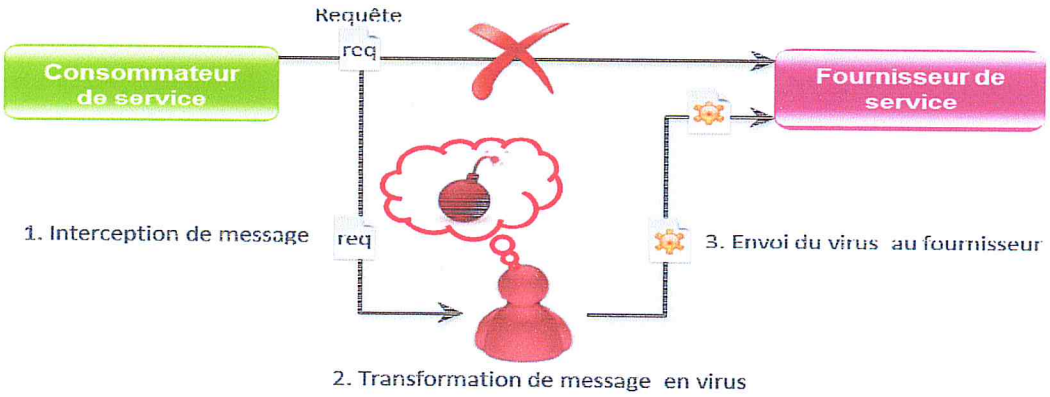


Figure 1.2 : Détournement d'informations

Cette attaque a pour conséquence immédiate de faire perdre les informations contenues dans le message du client. Dans un deuxième temps, elle endommage le serveur qui est alors dans l'incapacité de continuer à répondre correctement aux autres requêtes.

Pour lutter contre ce type d'attaque, il faut que les informations qui circulent restent intègres, c'est-à-dire qu'il faut pouvoir s'assurer que les informations émises n'ont pas été modifiées ou détruites jusqu'à leur réception. Une pratique courante est de chiffrer les informations et d'ajouter une somme de contrôle.

G. Injection SQL :

L'attaque par *injection SQL* vise les sites Web qui proposent des transactions mal construites dont les résultats sont emmagasinés dans une base de données relationnelle.

Elle consiste en ceci : SQL est un langage qui permet d'interroger et de mettre à jour une base de données relationnelle ; les requêtes sont soumises au moteur de la base en format texte, sans être compilées. Une requête typique est construite à partir de champs de formulaire remplis par l'internaute. Si l'auteur du site a été paresseux, il aura construit ses requêtes en insérant directement les textes rédigés par l'internaute, sans en contrôler la longueur ni le format ni le contenu.

Ainsi, un utilisateur malveillant informé de cette faille (ou qui la soupçonnerait) peut confectionner un texte tel qu'une fois incorporé à une requête SQL il ait des effets indésirables sur la base de données, par exemple en y insérant directement des ordres du langage, de telle sorte qu'ils soient interprétés. En voici un exemple :

L'instruction suivante construit directement à partir du nom introduit par l'utilisateur une requête SQL innocente, qui extrait de la base des utilisateurs tous les enregistrements qui concernent celui-là en particulier :

```
requete := "SELECT * FROM clients
WHERE nom = '" + nom_client + "';"
```

Soit un attaquant informé de cette faille, qui au lieu d'entrer dans le formulaire un nom valide, introduit la chaîne de caractères suivante :

*x'; DROP TABLE clients ; SELECT * FROM data WHERE nom LIKE %" as "nom_client*

La requête sera :

```
SELECT * FROM clients WHERE nom = 'x'; DROP TABLE clients;
SELECT * FROM secrets WHERE nom LIKE '%';
```

Avec, comme résultat, la destruction pure et simple de la table clients et un accès imprévu à la table secrets, dont le nom suggère qu'elle n'est pas destinée à être lue par les internautes.

La parade à ce type d'attaque consiste essentiellement à écrire des programmes moins naïfs, qui vérifient les données introduites par les utilisateurs avant de les utiliser, et en particulier qui éliminent les caractères qui ont une valeur sémantique spéciale pour SQL. Cette recommandation vaut d'ailleurs pour *tous* les programmes. [LC05]

H. Attaque des mots de passes :

Un mot de passe fort est un mot de passe qui est difficile à retrouver, même à l'aide d'outils automatisés. La force d'un mot de passe dépend de sa longueur et du nombre de possibilités existantes pour chaque caractère le composant. En effet, un mot de passe constitué de minuscules, de majuscules, de caractères spéciaux et de chiffres est techniquement plus

difficile à découvrir qu'un mot de passe constitué uniquement de minuscules.

- ***Attaques par force brute :***

Cette attaque consiste à tester toutes les combinaisons possibles d'un mot de passe. Plus il existe de combinaisons possibles pour former un mot de passe, plus le temps moyen nécessaire pour retrouver ce mot de passe sera long.

Un mot de passe fort, d'une longueur minimale de dix caractères et constitué d'au moins trois des quatre groupes de caractères énoncés ci-dessus (minuscules, majuscules, caractères spéciaux et chiffres), ne pourra être découvert par cette attaque dans un temps raisonnable, avec les moyens dont on dispose au moment de la rédaction de cette note d'information.

- ***Attaques par dictionnaires :***

Cette attaque consiste à tester une série de mots issus d'un dictionnaire. Il existe toutes sortes de dictionnaires disponibles sur l'Internet pouvant être utilisés pour ce type d'attaque (dictionnaire des prénoms, dictionnaire des noms d'auteurs, dictionnaire des marques commerciales...). En utilisant un mot de passe n'ayant aucune signification cette attaque ne donnera aucun résultat.

Cependant, plusieurs règles de transformation des mots du dictionnaire sont utilisées par les outils automatisés pour augmenter le nombre de combinaisons possibles. Citons par exemple :

- le remplacement d'un ou de plusieurs caractères du mot du dictionnaire par une majuscule (**bUreAU**).
- le remplacement de certains caractères par des chiffres comme par exemple le **S** en **5** (**mai5on**).
- l'ajout d'un chiffre au début ou à la fin d'un mot (**arbre9**).
- l'ajout des mots de passe déjà découverts.

Il est possible d'utiliser les dictionnaires pré calculés contenant une liste de mots de passe et leur empreinte associée. Même si cette possibilité accélère le temps nécessaire pour retrouver un mot de passe, elle nécessite une place plus importante en mémoire.

La solution idéale pour un individu malintentionné qui souhaiterait retrouver des mots de passe le plus rapidement possible serait d'avoir une liste exhaustive de tous les mots de passe possibles et de leur empreinte associée. Un tel dictionnaire n'est pas envisageable car il nécessiterait une place en mémoire bien trop importante. Cependant il est possible d'utiliser les attaques par compromis temps/mémoire.

- **Attaques par compromis temps/mémoire :**

Les attaques par compromis temps/mémoire sont des solutions intermédiaires permettant de retrouver un mot de passe plus rapidement qu'avec une attaque par force brute et avec moins de mémoire qu'en utilisant une attaque par dictionnaire. Ces compromis sont réalisés à partir de chaînes construites à l'aide de fonctions de hachage et de fonctions de réduction. Pour retrouver un mot de passe, il faudra d'abord retrouver à quelle chaîne appartient l'empreinte recherchée. Une fois que la chaîne aura été retrouvée il sera alors facile de retrouver le mot de passe, à partir du début de cette chaîne.

- **Attaques indirectes**

D'autres attaques assez connues car très pratiquées (en particulier le filoutage et les logiciels de captures des frappes au clavier) consistent non pas à déterminer le mot de passe par une recherche technique mais à le capturer au moment où il est saisi, ou encore à se le faire communiquer en usant de supercherie. Face à ces attaques, la qualité (ou « force ») du mot de passe doit être complétées par des mesures organisationnelles essentielles :

- procédures robustes d'ouverture de compte (initialisation et première fourniture du mot de passe) ;

- sensibilisation et bonne information des utilisateurs afin qu'ils détectent les tentatives pour leur soutirer leur mot de passe ;
- procédures robustes de réinitialisation en cas d'oubli ou perte du mot de passe par un utilisateur.
- ne pas réutiliser les mots de passe et en particulier, pas d'utilisation pour une application peu protégée du même mot de passe que pour une application sensible. [DT12]

III. Les certificats numériques et la sécurité :

1) Définition d'un certificat numérique :

Un certificat est un document électronique utilisé pour identifier un individu, un serveur, une entreprise ou toute autre entité et pour associer une clef publique à cette identité. Tout comme un permis de conduire, un passeport, ou tout autre moyen d'identification personnelle couramment utilisé, un certificat fournit généralement une preuve reconnue de l'identité de la personne. La cryptographie à clef publique utilise les certificats pour éviter les problèmes d'usurpation d'identité.

Les certificats fonctionnent sur les mêmes principes que ces différents documents d'identité. Les autorités de certification (AC ou CA) sont des entités qui valident les identités et émettent les certificats. Elles peuvent être des tierces-parties indépendantes ou des organisations possédant leur propre serveur d'émission de certificats (tel que *Red Hat Certificate System*). Les méthodes utilisées pour valider une identité varient selon les politiques d'émission d'une AC donnée — tout comme les méthodes de validation des autres formulaires d'identification varient selon les organismes d'émission et leurs domaines d'application. En général, avant d'émettre un certificat, une AC doit utiliser ses procédures de vérification publiées pour un type de certificat afin de s'assurer que l'entité demandant le certificat est bien celle qu'elle prétend être.

Le certificat émis par l'AC lie une clef publique particulière au nom de l'entité qu'il identifie (tel qu'un nom d'employé ou de serveur). Les certificats aident à prévenir l'utilisation de fausses clefs publiques pour l'usurpation d'identité. Seule la clef publique certifiée dans le certificat fonctionnera avec la clef privée correspondante possédée par l'entité identifiée par le certificat.

En plus de la clef publique, un certificat contient toujours le nom de l'entité qu'il identifie, une date d'expiration, le nom de son AC émettrice, un numéro de série et d'éventuelles autres informations utiles. Plus important, un certificat contient toujours la signature numérique de l'AC émettrice. La signature numérique de l'AC émettrice permet au certificat de fonctionner comme une *lettre d'introduction* pour les utilisateurs qui connaissent l'AC et lui font confiance mais qui ne connaissent pas l'entité identifiée par le certificat. [NT11]

2) L'utilisation des certificats numérique :

A. La signature numérique :

Elle permet de garantir l'identité de l'émetteur ainsi que l'intégrité des données. Elle met en œuvre une fonction de hachage non réversible combiné à un algorithme de cryptage asymétrique (paire de clé publique et clé privé). Elle permet d'éviter qu'un utilisateur envoie un message en usurpant l'identité d'un autre utilisateur.

En théorie la signature d'un message consiste à envoyer 2 informations :

1. le message en clair
2. le même message signé avec la clé privé de l'auteur du message.

Ensuite le destinataire :

- décrypte le message signé à l'aide de la clé publique de l'émetteur
- vérifie si le résultat correspond au message en clair.

En pratique, ce principe n'est pas vraiment satisfaisant car :

- **il faut signer le message dans sa totalité** pour l'authentifier, ce qui peut être très consommateur en temps CPU et en débit réseau lors du transfert du message signé. Pour résoudre ce problème, on remplace le message signé par un **condensé de message**, toujours signé, mais plus petit, et tout aussi unique.
- **une signature garantit une provenance mais pas une identité**. Pour résoudre ce problème, on utilise des certificats, où la clé publique d'un émetteur est certifiée par celle d'un autre, qui est elle-même certifiée par un autre, etc. jusqu'à parvenir à la clé publique d'un auteur dont l'identité est incontestable (une Autorité de Certification).

En pratique donc, l'envoi d'un message signé (mais non crypté) consiste à :

1. générer un condensé du message qu'on souhaite envoyer
2. crypter ce condensé de message avec sa clé privée
3. envoyer le condensé crypté + le message en clair

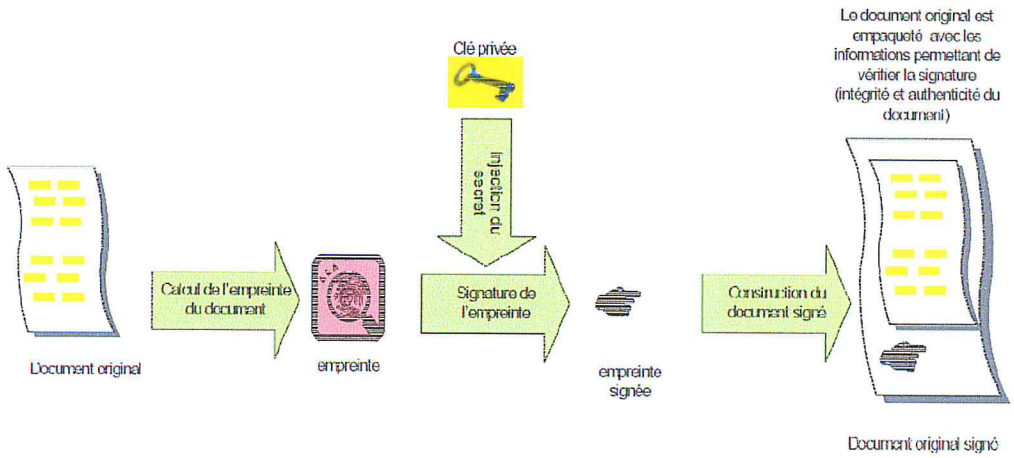


Figure 1.3: Mécanisme de signature à clé asymétrique

Pour vérifier si le message est bien celui de l'émetteur, le récepteur :

1. décrypte le condensé avec la clé publique de l'émetteur supposé
2. vérifie si le condensé obtenu est égal au condensé du message en clair

Un algorithme de signature est donc toujours composé d'algorithmes de condensé de message (fonctions de hachage tel que MD5, SHA-1…) et de cryptage Asymétrique tel que RSA. [SE04]

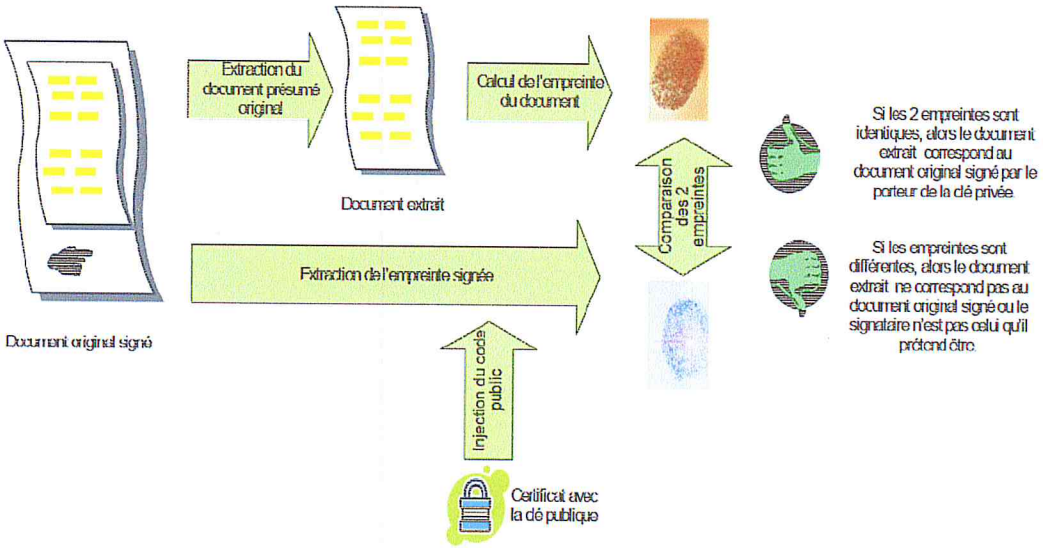


Figure 1.4: Mécanisme de vérification de signature

IV. Les mécanismes d'authentification :

L'authentification peut se faire de multiples manières:

1) Mots de passe :

Presque tous les serveurs permettent l'authentification cliente à l'aide d'un nom, ou pseudonyme, et d'un mot de passe. Par exemple, un serveur pour demander à un utilisateur de fournir un nom et un mot de passe avant de donner des droits d'accès à certaines parties du serveur. Le serveur maintient une liste des noms et des mots de passe ; si un nom particulier est dans cette liste, et que l'utilisateur fournit le bon mot de passe, le serveur donne des droits d'accès. [NT11]

La Figure 1.5 montre les étapes basiques mise en œuvre dans l'authentification d'un client à l'aide d'un nom et d'un mot de passe. Cette figure suppose que :

- L'utilisateur a déjà décidé de faire confiance au serveur, sans authentification ou sur la base d'une authentification de serveur via SSL.
- L'utilisateur a demandé une ressource contrôlée par le serveur.
- Le serveur demande une authentification client avant de donner les droits d'accès aux ressources demandées.

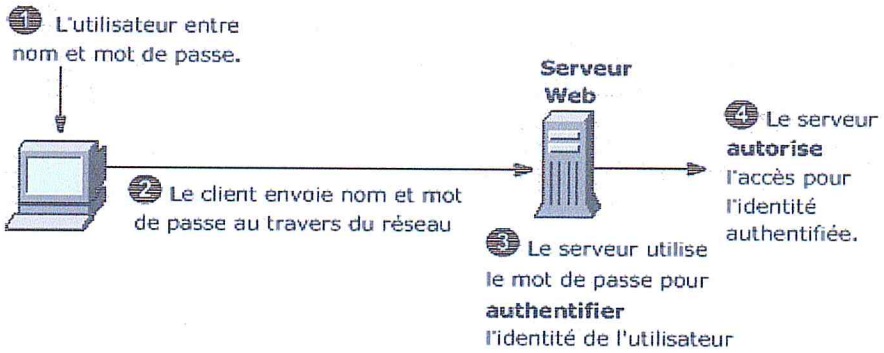


Figure 1.5 : Authentification par mot de passe

Voici les étapes décrites dans la figure 1.5 :

1. En réponse à une demande d'authentification de la part du serveur, le client affiche une boîte de dialogue demandant le nom de l'utilisateur et son mot de passe pour accéder à ce serveur. L'utilisateur doit fournir séparément un nom et un mot de passe pour chaque nouveau serveur qu'il désire utiliser pendant sa session de travail.
2. Le client envoie le nom et le mot de passe par le réseau, en clair ou par une connexion SSL chiffrée.
3. Le serveur vérifie le nom et le mot de passe dans sa base de données locale et, s'ils correspondent, il les accepte comme preuves authentifiant l'identité de l'utilisateur.
4. Le serveur détermine si l'utilisateur est autorisé à accéder aux ressources demandées, et si oui, autorise le client à y accéder.

Avec cet arrangement, l'utilisateur doit fournir un mot de passe pour chaque serveur, et l'administrateur doit conserver les noms et les mots de passe de chaque utilisateur, généralement sur des serveurs distincts.

Une implémentation propre ne mémorise pas les mots de passe en texte simple. À la place, il concatène le mot de passe avec une valeur aléatoire propre à chaque utilisateur (également appelée « *salt* ») et mémorise la valeur hachée du résultat avec le « *salt* ». Ceci rend plus difficile des attaques de force brute.

Comme expliqué dans la section suivante, un des avantages de l'authentification par certificat est qu'elle peut être utilisée pour remplacer les trois premières étapes décrites à la figure 1.5avec un mécanisme qui permet à l'utilisateur de fournir un seul mot de passe (qui n'est pas transmis à travers le réseau) et permet à l'administrateur de centraliser le contrôle de l'authentification des utilisateurs.

2) Certificats de clés publiques :

Les certificats de clés publiques sont l'une des techniques d'authentification les plus usitées à ce jour, certes loin derrière les mots de passe.

La figure 1.5 décrit le fonctionnement d'une authentification client à l'aide des certificats et du protocole SSL. Pour authentifier un utilisateur auprès d'un serveur, Le client signe numériquement des données générées aléatoirement et envoie à la fois ces données signées et le certificat sur le réseau. Pour les besoins de cette discussion, la signature numérique associée aux données signées peut être considérée comme une preuve fournie par le client au serveur. Le serveur authentifie l'identité de l'utilisateur en se basant sur la force de cette preuve. [NT11]

Comme pour la figure 1.4, la figure 1.5 suppose que l'utilisateur a déjà décidé de faire confiance dans le serveur et qu'il a demandé une ressource, et que le serveur a demandé une authentification client lors du processus d'évaluation des droits à accéder à la ressource demandée.

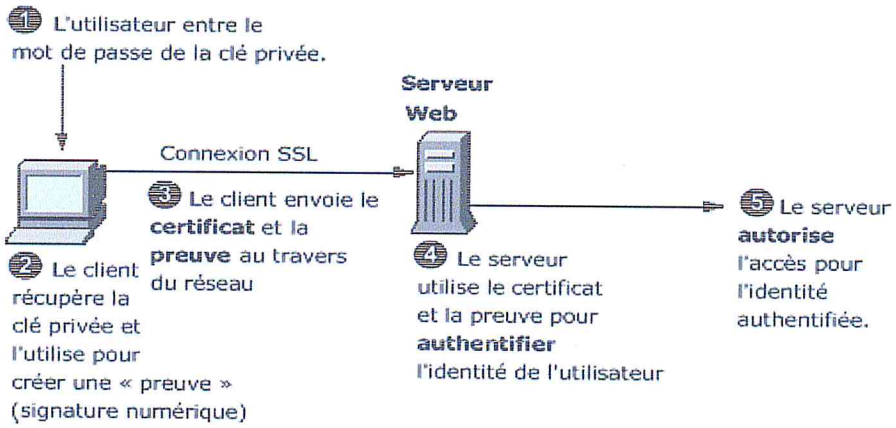


Figure 1.6 : Authentification par certificat à clé publique

Contrairement au processus décrit à la figure 1.5, celui de la figure 1.6 nécessite d'utiliser SSL. La figure 1.6 suppose également que le client possède un certificat valide qui peut être utilisé pour l'identifier auprès du serveur. L'authentification par certificat est généralement considérée comme préférable à l'authentification par mot de passe car elle est basée sur ce que l'utilisateur a (la clef privée) aussi bien que sur ce que l'utilisateur sait (le mot de passe qui protège cette clef privée). Cependant, il est important de remarquer que ces deux affirmations ne sont vraies que si aucune personne non autorisée n'a accès à l'ordinateur de l'utilisateur ou a son mot de passe, si le mot de passe de la base de données des clefs privées du logiciel client a été défini, et si le logiciel est paramétré pour demander le mot de passe à intervalles raisonnablement fréquents.

Voici les étapes décrites dans la figure 1.6 :

1. Le logiciel client, tel que le navigateur, maintient une base de données des clefs privées correspondantes aux clefs publiques publiées avec tous les certificats émis pour ce client. Le client demande le mot de passe de cette base de données la première fois qu'il a besoin d'y accéder lors d'une session donnée, par exemple, la première fois que l'utilisateur essaie d'accéder à un serveur SSL qui requiert une authentification par certificat. Après avoir renseigné une première fois ce mot de passe, l'utilisateur n'en a plus besoin pour la durée de la session, même en accédant à d'autres serveurs SSL.

2. Le client débloque la base de données des clefs privées, récupère la clef privée du certificat de l'utilisateur et utilise cette clef privée pour signer numériquement des données générées aléatoirement dans ce but en se basant sur des entrées du client et du serveur. Ces données et la signature numérique constituent une « preuve » de la validité de la clef privée. La signature numérique peut uniquement être créée avec la clef privée et peut être validée par la clef privée associée aux données signées, ce qui est réservé à la session SSL.
3. Le client envoie le certificat de l'utilisateur et la *preuve* (les données générées aléatoirement signées numériquement) par le réseau.
4. Le serveur utilise le certificat et la *preuve* pour authentifier l'identité de l'utilisateur

À ce moment, le serveur peut éventuellement exécuter des tâches d'authentification supplémentaires, comme vérifier si le certificat présenté par le client est stocké dans l'entrée de l'utilisateur d'un annuaire LDAP. Le serveur continu alors à évaluer si l'utilisateur identifié est autorisé ou non à accéder à la ressource demandée. Ce processus d'évaluation peut employer une variété de mécanismes standards d'autorisation, en utilisant éventuellement des informations présentes dans un annuaire LDAP, des bases de données d'entreprises, etc. Si le résultat de l'évaluation est positif, le serveur autorise le client à accéder à la ressource demandée.

Comme on peut le voir en comparant les figures 1.5 et 1.6, les certificats remplacent la portion de l'authentification correspondant à l'interaction entre le client et le serveur. Plutôt que de demander à l'utilisateur d'envoyer des mots de passe par le réseau à longueur de journée, l'ouverture de session unique demande une seule fois à l'utilisateur de saisir son mot de passe de base de données de clefs privée, sans l'envoyer par le réseau. Pour la suite de la session, le client présente le certificat de l'utilisateur pour authentifier l'utilisateur auprès de chaque serveur auquel il se connecte. Les mécanismes d'autorisation existants basés sur l'authentification de l'identité du l'utilisateur ne sont pas concernés.

3) Les cartes à puce :

Les cartes à puce sont aujourd'hui considérées comme des ressources sûres du fait de leur structure interne et des validations qu'elles subissent tout au long de leur cycle de vie. Il est donc possible de les utiliser comme éléments de base pour sécuriser différents types d'environnements : accès physiques à des locaux, transactions bancaires, systèmes embarqués...etc.

L'authentification par carte à puce consiste à utiliser des cartes d'identités virtuelles (Certificats délivrés par des autorités de certifications), combinée à des cartes à puce multi-applications à système d'exploitation ouvert. Pour augmenter davantage la sécurité, nous pouvons utiliser des méthodes biométriques d'authentification telles que la reconnaissance vocale et/ou digitale. **[TE06]**

4) Authentification unique :

L'authentification unique (en anglais single sign-on ou SSO) donne accès à toutes les applications d'un utilisateur avec un seul moyen d'authentification. Cela peut être par exemple un mot de passe, une clé USB, ou votre doigt si vous disposez d'un lecteur biométrique…

Typiquement, un SSO nécessite d'installer sur PC un logiciel discret qui renseignera les mots de passe des applications. Mais cela n'est même pas nécessaire si on se contente de lancer des applications Web ou si on travaille en mode 'client léger'. Dans ce cas, le logiciel de SSO réside sur un serveur et renseigne les mots de passe à distance.

Le SSO simplifie la vie des utilisateurs, car un seul mot de passe suffit. Mais le SSO permet également d'augmenter la sécurité et de réduire les coûts, notamment de help desk. **[NT11]**

V. Besoins du contrôle d'accès :

Les recherches sur le contrôle d'accès s'attachent à de multiples problèmes relatifs à la notion de droit dans un système : organisation, formalisation, sémantique, architecture.

La famille des modèles de contrôle d'accès basés sur les rôles, appelés généralement contrôle d'accès à rôles pour Role Based Access Control (RBAC), ont été proposés pour la première fois.

La famille des modèles RBAC présente une nouvelle organisation des droits centrée sur le concept de rôle. Cette notion a été introduite dans le contrôle d'accès pour simplifier l'administration des droits des grands systèmes. Depuis, les principes fondateurs des modèles RBAC ont largement été répandus et adaptés pour aboutir à la définition de nouveaux modèles raffinés, intégrants des notions de plus en plus complexes. De nombreuses propositions ont été faites, que d'aucuns appelleraient les modèles x-Based Access Control, comme les notions de treillis (Lattice-BAC), de temps (Temporal-RBAC, Generalized-Temporal-RBAC), d'équipes (Team-BAC), de tâches (Task-BAC), d'organisations (Organization-BAC) ou encore de contextes (Context-BAC) ont été proposées pour structurer les droits.

La volonté sous-jacente à cette profusion de modèles de contrôle d'accès est de proposer une organisation des droits qui permette de traduire au mieux les « politiques de sécurité », c'est-à-dire l'ensemble des règlements qui régissent la façon dont sont protégées les ressources d'un système. Ces politiques sont exprimées en langue naturelle au début du cycle de développement du contrôle d'accès. Il faut ensuite traduire ces règlements dans les systèmes grâce à des modèles de contrôle d'accès qui soient :

– suffisamment expressifs, pour permettre l'expression de règlements complexes et représenter fidèlement la structure et le fonctionnement des organisations,

– décidables, pour évaluer les autorisations à partir des règlements,

– faciles d'administration, les modèles doivent fournir un ensemble de primitives et de fonctionnalités simplifiant les activités des administrateurs.

La mise en place d'une politique de contrôle d'accès dans un système est un chantier important pour une organisation, à la croisée de l'informatique, de la gestion des ressources humaines et de la qualité. **[AA03]**

VI. Conclusion :

La sécurité du système d'information n'est pas et ne peut pas être contenue dans un dispositif ni dans un ensemble de dispositifs, qu'elle ne peut pas non plus être contenue dans les limites temporelles d'un projet, mais qu'elle est un processus ou, si l'on veut, une activité. Nous entendons par là que les ingénieurs de sécurité du SI doivent se consacrer à cette activité, pas forcément à plein temps, mais en permanence, sur plusieurs fronts : veille scientifique et technologique, surveillance des journaux d'événements, audit des infrastructures et des applications, sensibilisation et formation des utilisateurs, expérimentation de nouveaux outils et de nouveaux usages, c'est ainsi que La démarche de sécurité doit être toujours active.

Chapitre 2:

Architecture Logicielle

I. Introduction :

La programmation orienté objet représente un progrès important en termes de qualité de programmation et de lisibilité de code mais ne permet pas encore une distinction claire entre le code qui forme la préoccupation centrale de l'application (le code métier) et le code d'interaction avec le système. Ainsi, certains objets ne peuvent être réutilisés à cause de leur forte dépendance avec le système ou de leur forte dépendance avec le reste de l'application.

Aujourd'hui, l'approche architecture logicielle comme nouvelle discipline du génie logiciel, est largement acceptée comme voie prometteuse pour la production de logiciels de haute qualité.

Dans ce chapitre nous étudions les caractéristiques de la définition d'une architecture logicielle à base de composants en s'appuyant sur différents travaux autour des langages de description d'architecture.

II. L'architecture logicielle :

L'ingénierie des logiciels à base de composant se réduit à un assemblage de composants prédéfinis. Chaque composant joue un rôle spécifique dans le système. Il définit clairement les services qu'il offre et les services requis pour accomplir sa fonction. Le développement de logiciel basé composant augmente considérablement la fiabilité des systèmes puisque leur construction est basée sur des composants testés et certifiés. De même, les coûts du développement sont généralement réduits car une partie des composants sont simplement réutilisés. L'étape de maintenance se réduit, quant à elle, à un remplacement de composants, ce qui facilite sa tâche et favorise l'évolution du système. [MH08]

III. Les concepts de bases de l'architecture logicielle :

1) Le composant :

Il n'existe pas une définition consensuelle de ce qu'est un composant. Cependant une des définitions les plus souvent citées dans la littérature est celle que donne Szyperski [Szy98] :

"Un composant logiciel est une unité binaire de composition ayant des interfaces spécifiées de façon contractuelle et possédant uniquement des dépendances de contexte explicites. Un composant peut être déployé de manière indépendante et est sujet à composition par des tierces."

L'intérêt de cette définition est qu'elle contient plusieurs concepts importants qui caractérisent un composant

Une autre définition donnée par [BCS02] :

"Les composants sont des structures de 'run-time', ce qui veut dire qu'elles se manifestent lors de l'exécution d'un système".

Alors que dans [MN97] il est définit par :

"Une entité généralement statique qui est nécessaire au moment de la construction d'un système et qui n'existe pas forcément au moment de l'exécution "

On termine par la définition donnée par Hans-Gerhar Gross[SG96] :

"Un composant logiciel est une pièce indépendante qui fournit un ensemble de fonctions permettant l'accès aux services grâce à des interfaces"

En résumant, un composant logiciel est une entité responsable de la réalisation d'une (ou plusieurs) fonctionnalité(s) bien précise(s) dans une architecture à un certain niveau d'abstraction. C'est une entité qui fournit des fonctionnalités de calcul et de stockage. Il interagit avec les autres composants pour réaliser un ou plusieurs objectifs d'une architecture. [DB09]

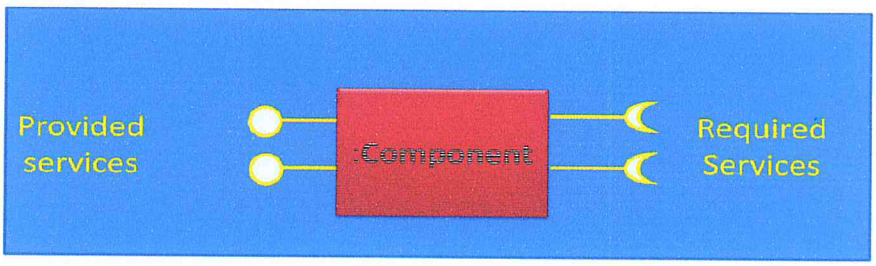


Figure 2.1 : Représentation UML d'un composant logicielle

2) Les connecteurs :

L'interconnexion entre les composants peut se faire par l'intermédiaire de connecteurs. Un connecteur modélise une interaction entre les composants dans une architecture logicielle comme le montre la figure 2.1.

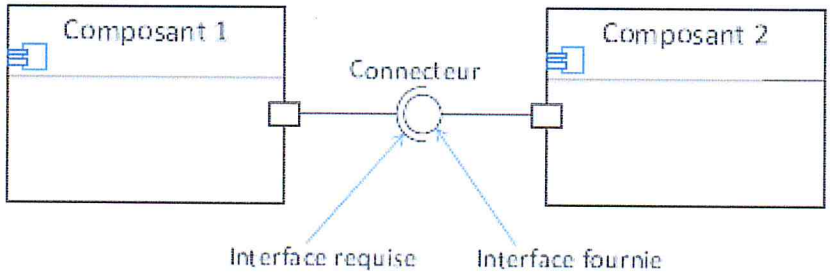


Figure 2.2 : Eléments d'une configuration architecturale

Le rôle primordial des connecteurs est d'assurer une communication correcte entre les composants. Un connecteur doit être indépendant de la logique de l'application. Ce dernier aspect ouvre la voie vers les possibilités de changements statiques ou dynamiques des connecteurs sans aucun impact sur la fonctionnalité de l'architecture du logiciel. **[DB09]**

Un connecteur peut être un connecteur d'assemblage ou de délégation.

- ✓ **Un connecteur d'assemblage** permet de connecter un composant qui fournie des services à un composant qui les utilise, on peut en déduire une relation d'*utilisation*.
- ✓ **Un connecteur de délégation** permet de déléguer la réalisation ou le requiert d'un service à un de ses sous-composants, on en déduit une relation de *composition* entre les participants.

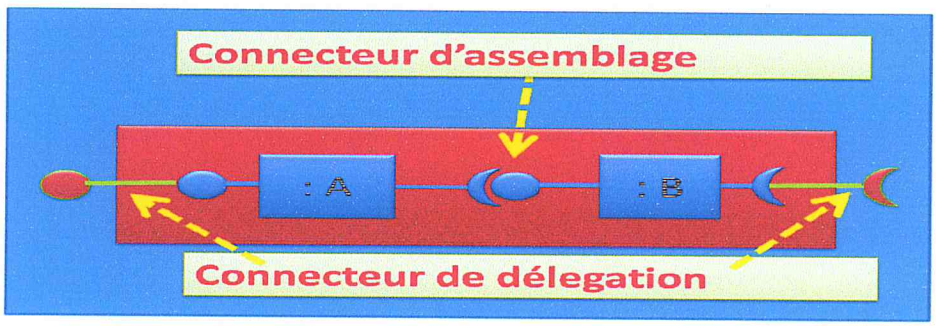


Figure 2.3: types de connecteurs d'un composant

3) Interface :

C'est le point de communication qui permet d'interagir avec l'environnement. Pour un composant, il existe deux types d'interfaces comme le montre la figure 2.3 :

– **Les interfaces fournies** : décrivent les services offerts par le composant.

– **Les interfaces requises** : décrivent les services que les autres composants doivent fournir pour le bon fonctionnement du composant dans un environnement particulier.

4) Configuration :

Une configuration est la description de l'ensemble des composants logiciels nécessaires pour le fonctionnement d'une application ainsi que celle de leurs communications. **[IA10]**

Elle représente en fait une instance possible d'un style architectural. Plus précisément, une configuration décrit les instances de composants intervenants et les relations qu'elles entretiennent entre elles.

IV. Cycle de vie de développement :

La construction d'applications à base de composants, qui est schématisée dans la figure 2.4, se décompose en trois étapes dont deux sont fondamentales : le développement de composants et la composition (ou assemblage) ; la troisième étape concerne les activités qui ont lieu après le développement et l'assemblage, et incluent notamment le déploiement des applications et de leurs composants, ainsi que l'exécution et l'administration des applications. Bien que les étapes fondamentales de l'approche à composants concernent le développement de composants ou de compositions, cette approche ne propose pas de moyens d'analyse permettant de modéliser un problème d'une façon particulière comme dans l'approche orientée objet; cette approche est plutôt focalisée dans les activités qui ont lieu après l'analyse. Chacune des trois étapes est constituée à son tour par un certain nombre d'activités pouvant être effectuées par des acteurs avec des compétences différentes, et ces trois étapes sont séparées entre elles par des activités de livraison. **[HC04]**

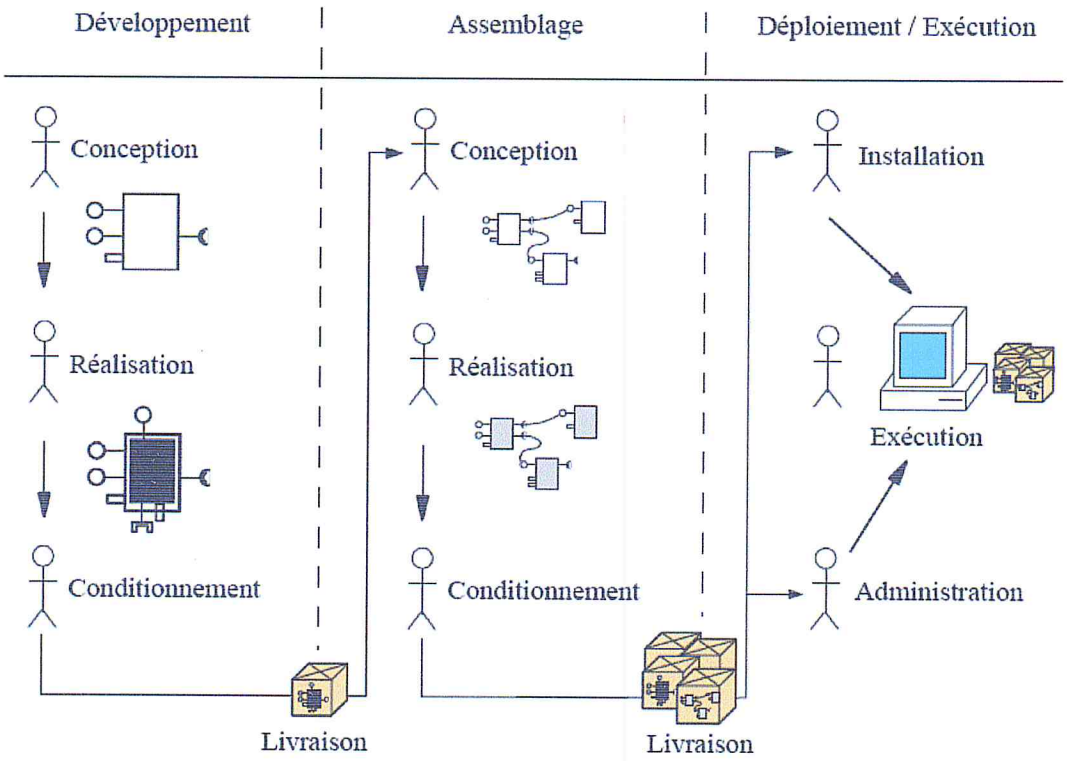


Figure 2.4 : Cycle de vie de développement simplifié

1) Développement :

Le développement de composants inclut les activités de conception, implémentation et conditionnement des composants destinés à être assemblés dans différentes applications.

- ✓ **conception** : la conception de composants inclut la spécification de la *vue externe* d'une classe de composant ainsi que de son comportement.
- ✓ **réalisation** : la réalisation de composants consiste à la création de l'implémentation de la classe de composant qui implémente les vues externe et interne. Plusieurs implémentations peuvent être réalisées pour une même vue externe.
- ✓ **conditionnement** : lors du conditionnement, une classe de composant est introduite dans un paquetage de composant qui permet de réaliser la livraison et le déploiement indépendants.

2) Assemblage :

L'assemblage (ou composition) de composants inclut les activités de conception, de réalisation et de conditionnement d'un assemblage qui peut représenter une partie ou bien la totalité d'une application. L'étape d'assemblage suppose l'existence de composants développés auparavant qui sont utilisés par les acteurs de cette étape.

- ✓ **conception** : La conception d'un assemblage est réalisée en étudiant la manière de composer un ensemble de composants pré-existants pour créer une composition représentant soit une partie ou bien la totalité d'une application, c'est-à-dire une architecture.
- ✓ **réalisation** : La réalisation d'un assemblage consiste à l'écriture du code permettant de réaliser la composition des instances de composants.
- ✓ **conditionnement** : Lors du conditionnement, un assemblage est introduit dans un *paquetage d'assemblage* qui peut inclure les composants employés dans l'assemblage ainsi que des ressources propres à l'assemblage telles que des fichiers de configuration ou des fichiers binaires (librairies, images, sons, etc...).

3) Déploiement, Exécution et Administration :

Les étapes postérieures au développement et à l'assemblage sont les suivantes :

- ✓ **déploiement** : lors de l'installation, des paquetages contenant des compositions et des composants sont installés et des activités de configuration peuvent être réalisées.
- ✓ **exécution** : l'application est exécutée. A ce moment là, l'environnement d'exécution est actif.
- ✓ **administration** : l'administration inclut l'application de mises à jour ainsi que la désinstallation d'une application et de ses composants. **[HC04]**

V. Environnement d'exécution :

Un modèle à composants, qui définit la structure des composants et des compositions, est associé à un environnement d'exécution qui fournit du support aux applications construites à partir du modèle lors de l'exécution. L'environnement d'exécution est parfois comparé à un mini-système d'exploitation car il est chargé de

gérer des aspects divers tels que le cycle de vie des instances ou bien les propriétés non-fonctionnelles.

L'environnement d'exécution est typiquement constitué par le conteneur ainsi qu'une infrastructure sous-jacente (par exemple un intergiciel, ou une machine virtuelle).

L'environnement d'exécution peut aussi fournir des mécanismes d'introspection permettant de réaliser des analyses de la structure des composants ou même de l'application lors de l'exécution. **[HC04]**

VI. Les modèles à base de composant :

1. Le standards UML2.0:

Les ADLs ont pendant longtemps souffert d'être une approche très académique, peu utilisée par le milieu industriel du génie logiciel. Ce manque d'utilisation a fait apparaitre UML2.0.

Après le succès de la première version d'UML, la deuxième version notée UML 2.0 introduit la notion de diagramme d'architecture, aussi appelée diagramme de structure composite. UML propose une notation unique pour couvrir toutes les phases du développement logiciel. Avec l'ajout de ce diagramme, UML veut combler une des lacunes de la première version en permettant la spécification de l'architecture d'une application.

Un composant UML 2.0 est une entité modulaire et réutilisable fournissant et requérant des interfaces qui peuvent être potentiellement exposées par l'intermédiaire de ports. Un connecteur est une entité qui relie des ports ou des interfaces de composant. Dans un diagramme d'architecture, une application ou un composant est constitué de l'assemblage d'autres composants par l'intermédiaire de ports interconnectés.

2. Le modèle a composant *Fractal* :

Fractal est un modèle plus léger et plus proche des concepts des langages de programmation. Le modèle de composant Fractal réalisé par France Telecom R&D et par l'INRIA. Fractal vise à autoriser une définition, une configuration et une

reconfiguration dynamique d'une architecture à base de composants, ainsi qu'une séparation claire des préoccupations fonctionnelles et non fonctionnelles.

Un composant Fractal est formé d'une membrane et d'un contenu. *La membrane* définit, par l'intermédiaire d'un ensemble d'interfaces, les services requis et fournis par le composant. *Le contenu* d'un composant Fractal permet de différencier deux sortes de composant : les primitifs et les composites.

Le contenu d'un primitif identifie un module logiciel (classe, ensemble de fonctions, *etc.*) permettant de réaliser les services du composant primitif. Le contenu d'un composite définit un assemblage de composants permettant de mettre en œuvre les services du composite. **[OB05]**

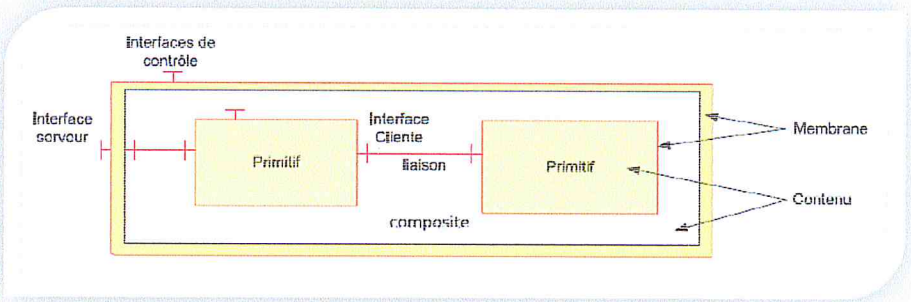


Figure 2.5: modèle de composant fractal

Fractal met à disposition un ensemble d'interfaces de contrôle qui sont la gestion du cycle de vie (*life-cyclecontroler*), du nom (*name-controller*), des liaisons (*binding-controler*), des attributs (*attribute-controler*), des parents (*super-controller*) et du contenu (*content-controler*) pour les composites. Le modèle étant extensible, il est possible de définir ses propres interfaces de contrôle.

Une interface Fractal métier est du type client ou serveur. Une interface serveur identifie les services offerts par un composant alors qu'une interface client spécifie les fonctionnalités qu'un composant requiert pour son fonctionnement.

Fractal ne possède pas de notion de connecteur explicite avec une sémantique de processus.

Cependant, il utilise la notion de liaison pour spécifier les interactions entre composants. Une liaison Fractal est définie comme un lien orienté entre une interface client et une interface serveur. Ce lien permet aux composants d'interagir.

VII. Les langages de description d'architecture :

1) L'ADL Darwin :

Darwin a été développé au Distributed Software Engineering Group de l'Imperial Collège de Londre. Il propose un modèle de composant pour la construction d'applications distribuées.

Comme pour la plupart des ADLs, un composant est une instance définie par une interface qui décrit les services qu'il fournit et qu'il requiert, cette dernier représente le port de communication, et une signature pour décrire la signature de la fonction du composant qui lui permettre de fonctionner et de s'intégrer dans une architecture. **[OB05]**

2) ArchJava :

ArchJava se situe à la frontière entre le langage de description d'architectures et les modèles de construction d'applications à composants. Il a été créé à l'université de Washington par Jonathan Aldrich et Craig Chambers. ArchJava a pour objectif d'améliorer la compréhension des programmes, de garantir l'architecture de l'application, de permettre une meilleure évolutivité des applications. **[OB05]**

ArchJava étend le langage Java afin d'incorporer les éléments d'architectures à l'intérieur du code. Un travail important a été réalisé sur le compilateur ArchJava afin de respecter l'intégrité des communications entre les composants. La communication entre les composants se fait à l'aide d'un appel de méthode.

3) L'ADL Fractal :

En amont du modèle de composant Fractal, il est possible de décrire l'architecture d'une application à base de composants à l'aide de l'ADL Fractal. Basé sur XML, ce langage définit une syntaxe abstraite indépendante de tout langage de programmation pour la description de l'architecture en termes de composants, d'interfaces, de liaisons et d'attributs.

L'ADL Fractal, est un ADL extensible. Il permet d'intégrer facilement de nouveaux concepts au niveau de la description de l'architecture en modifiant

juste le descripteur XML ceci est charger par l'usine fractal pour faire la liaison entre les différents composants et sous-composants.

L'Usine fractal est un composant présent dans chaque implémentation dont le rôle est de lire le descripteur et faire la liaison des différents composants. **[OB05]**

4) L'ADL Assembler4J :

L'assembleur est un outil développé pour permettre l'assemblage et la configuration dynamiques des composants dans une application, selon une description fournie.

Lorsqu'un composant est instancié, sa partie contrôle fait appel à l'assembleur pour charger le fichier descripteur propre au composant, ce dernier configure les dépendances de ce composant.

Les composants primitifs sont implémentés par des classes java implémentant les interfaces fournies par le composant. Pour éliminer les dépendances vers d'autres composants, le code source d'une telle classe ne contient aucune référence directe vers un type de composant externe. Au lieu de ça, il utilise des références sous forme d'interfaces, qui sont définies conceptuellement en tant qu'interfaces requises.

Comme la vue interne de ces composants est inaccessible, ces composants n'ont pas de fichier de description d'architecture.

Les composants composites sont implémentés sous forme de package contenant une classe principale (contrôleur), des classes représentant les composants primitifs et souvent d'autres composants composites. Ces composants ont chacun un fichier descripteur d'architecture à fin de les configurer. La classe principale étend la classe Controller et implémente l'interface Binding pour pouvoir utiliser l'API de l'assembleur. Elle implémente aussi les interfaces fournies par le composite puis les délègue aux sous composants chargés de les réaliser.

La délégation est un comportement lié à l'interception des appels entrants et de les rediriger vers le sous composant approprié. La délégation est assurée par l'implémentation de l'interface Binding, dans nos exemples et

notre application nous avons opté pour que la classe principale du composite soit la classe qui l'implémente.

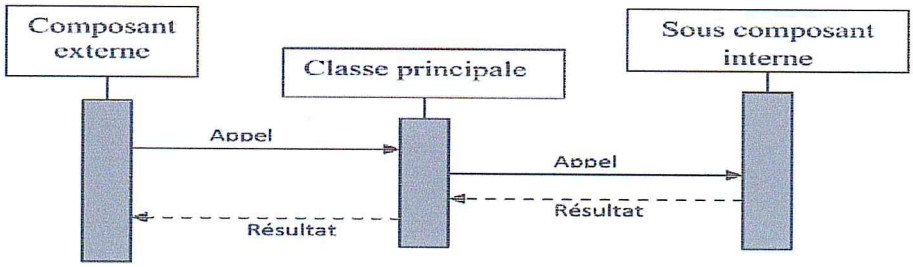


Figure 2.6 Diagramme de séquence montrant la délégation

Le contrôleur est un objet qui est chargé de traduire la configuration fournie dans le fichier descripteur en une connexion concrète entre des sous composants déterminés. [BM11]

VIII. La programmation orienté aspect :

La Programmation Orienté Aspect(POA) se propose de pallier aux faiblesses des paradigmes classiques (programmation modulaire, POO, programmation par composant) en introduisant un autre type d'organisation au sein des programmes. Elle permet de séparer les aspects qui se retrouvent de manière transverse dans différentes parties d'une application (*cross-cutting concerns*) permettant ainsi de minimiser le code répétitif qui devient complexe et peu maintenable.

POA s'utilise souvent pour écrire des Logs, mesurer les performances ou pour la gestion de transactions. [YN09]

IX. Conclusion :

Aujourd'hui l'architecture logicielle est reconnue comme une étape fondamentale du développement logiciel, elle apparaît dans tous les projets sérieux. Mais elle reste un domaine récent, il n'existe pas de représentation standard, et aussi il existe peu d'aide pour la conception architecturale ainsi que les méthodes de vérification/validation sont immatures.

Chapitre 3 :

Analyse des besoins

I. Introduction :

Dans ce présent chapitre nous essayerons de déterminer toutes les fonctionnalités du composant à réaliser, pour que tout concepteur d'application puisse exploiter par la suite. Ces fonctionnalités représenteront la vue externe du composant. Nous utiliserons les diagrammes de cas d'utilisations d'UML pour illustrer le besoins de ces fonctionnalités

II. Généralités :

Le composant est amené à être instancié dans le contexte d'une application qui désire exploiter les services de sécurité du composant à réaliser. Le composant n'est pas sensé interagir avec un acteur humain à travers une interface IHM. IL est indépendant de tout IHM. Dans notre cas, notre composant interagit avec les autres composants d'une application. Les acteurs de notre composant pourraient être les autres composants si la conception de l'application est une conception par assemblage de composant selon une approche d'architecture logicielle. Les acteurs pourraient être les méthodes de classes si l'application est orientée objet. Elle peut aussi être une page JSP dans le cas d'une application web basée sur la technologie Java. Dans la suite nous utiliserons « application » pour designer ces acteurs.

III. Interaction Application Composant :

L'interaction du composant avec l'application est plutôt une interaction de type requête réponse. La requête active le service et la réponse rend les résultats du service, si de tels résultats sont nécessaires.

A travers ces requêtes l'application demande conseil au composant de sécurité. A titre d'exemple une méthode pourrait demander si celui qui désire la mettre en œuvre (un utilisateur humain ou un utilisateur système, ceci dépend de l'application) en a le droit ou non. Le composant de sécurité en se basant sur les informations préalablement fournies par l'application et la politique de sécurité

que l'application a indiqué de suivre de répondre par oui (il a le droit d'exécuter) ou non (il n a pas le droit).

Dans une autre situation, une application pourrait demander si une opération déclenchée par un de ses acteurs a le droit d'arriver à une information, la lire ou la modifier. Le composant en se basant sur l'information fourni au préalable par l'application pourrait répondre par l'affirmative ou non.

Cependant les requêtes réponse ne sont pas sans état. Le composant de sécurité pourrait maintenir un état dans lequel s'inscrivent un ensemble de requêtes réponses. C'est ce qu'on appelle souvent le concept de session. Une session doit être explicitement ouverte par l'utilisateur (ici l'application). La session pourra par la suite maintenir des informations diverses définies par l'utilisateur. La session doit par la suite être fermée explicitement ou automatiquement (dépassement de délai). Il est clair qu'une session est associée à un et un seul utilisateur.

1er résultat :

De la discussion précédente, il ressort les conclusions suivantes :

- ✓ Le composant de sécurité donnera ses conseils en se basant sur l'information donnée par l'application.
- ✓ Les informations que fournit l'application sont de trois grandes natures :
  1) Les opérations à sécuriser
  2) L'information à sécuriser
  3) Les acteurs qui veulent accéder et/ou informations

IV. Fonctionnalités générales du composant :

Une application peut être vue comme étant un ensemble de fonctionnalités complexes. Dans la pratique une application possède des utilisateurs, en général des humains. Selon leur position dans l'entreprise, ces utilisateurs ont des privilèges différents. Souvent il y a deux catégories

d'utilisateurs : L'administrateur du système et les autres utilisateurs. L'administrateur du système possède en général tous les privilèges et peut réaliser n'importe quelle action dans un système. Les autres utilisateurs peuvent accéder à un sous ensemble bien précis de fonctionnalité et à un sous ensemble particulier de données. La gestion de ces droits d'accès se base fondamentalement sur une étape bien précise qui est l'authentification.

Une application doit reconnaitre (authentifier) un utilisateur. Ça sera à la base de cette reconnaissance que l'application peut déterminer les limites de l'espace dans lequel pourrait évoluer l'utilisateur. Ainsi une organisation des utilisateurs s'impose. L'organisation des utilisateurs impose au préalable une déclaration des utilisateurs ainsi que leur droit d'accès. La déclaration des utilisateurs, et de manière plus générale la gestion des utilisateurs et de leurs droits d'accès est une opération qui est du domaine exclusif de l'administrateur.

En général, l'administrateur d'une application est le premier utilisateur à définir. Ceci se fait souvent lors de l'installation d'une application et son initialisation.

L'organisation des utilisateurs et la définition de leurs droits d'accès sont en fait des actions liées à l'aspect sécurité dans une application. Elles doivent en réalité être prises en charge par le composant de sécurité

Le composant de sécurité devra ainsi supporter les opérations globales suivantes :

- ✓ La gestion des utilisateurs
- ✓ L'authentification des utilisateurs
- ✓ La gestion des droits d'accès.
- ✓ L'association des droits d'accès aux utilisateurs

Parfois, pour une même action, le nombre d'opérations de vérification des droits d'accès et le nombre de règles spécifiques à respecter est différent pour un même utilisateur. Ces règles définissent ce qu'on appelle souvent le niveau de sécurité. Le niveau de sécurité pourrait restreindre ou étendre le champ d'action d'un utilisateur.

Dans certains contexte, notamment celui de l'internet, plusieurs aspect de sécurité spécifique sont apparu. A titre d'exemple, l'opération d'authentification d'une application peut faire face à ce qui est communément appelé un robot1. Pour face diverses techniques ont été mise en place. Les techniques les plus connus sont l'isolation pour une période de la source d'où vient la menace (adresse IP par exemple) et l'utilisation de Captcha. Dans ce contexte le composant pourrait apporter ses conseils dans la mise en place de ces techniques.

Le composant ne pourra pas mettre en œuvre tous ces techniques vu que c'est un composant indépendant de toute IHM et même de toute application. Il pourra par contre contribuer de manière efficace.

Ainsi les autres fonctionnalités du composant de sécurité sont

- ✓ La gestion des niveaux de sécurité
- ✓ La mise à disposition de l'application de divers services d'aide à la mise en place de solutions de sécurités spécifique
- ✓ La gestion des utilisateurs et les informations d'authentification des utilisateurs.
- ✓ Le composant doit suivre un modèle de contrôle d'accès comme RBAC et avoir des solutions pour surpasser les attaques comme l'injection SQL.

Qu'est ce qu'un utilisateur :

Un utilisateur est représenté par un ensemble d'information qui permet d'une part de l'authentifier par le système et d'autre part de l'identifier en tant qu'élément du monde réel. La présence d'un nombre plus ou moins important d'information concernant un utilisateur dépendra en réalité de la politique de sécurité de l'application et de son niveau de sécurité. A titre d'exemple, selon le niveau de sécurité et la politique de sécurité, l'authentification pourrait être légère ou très fine, Une authentification légère se suffit d'un nom d'utilisateur et d'un mot de passe. Une

1 Une application qui essaie de refaire ce que l'acteur humain fait, notamment durant la phase d'authentification

authentification fine pourrait nécessiter en plus un lieu bien précis (la console d'un ordinateur, une position géographique), une identification précise de la machines cliente (adresse IP), une information biométrique etc.…

V. Diagramme de cas d'utilisation :

L'utilisation des diagrammes de cas d'utilisations nécessitent la détermination des acteurs du système suivis des cas d'utilisations.

1) Diagramme de cas d'utilisation générale :

Dans ce diagramme on définit la fonctionnalité globale de notre composant de sécurité.

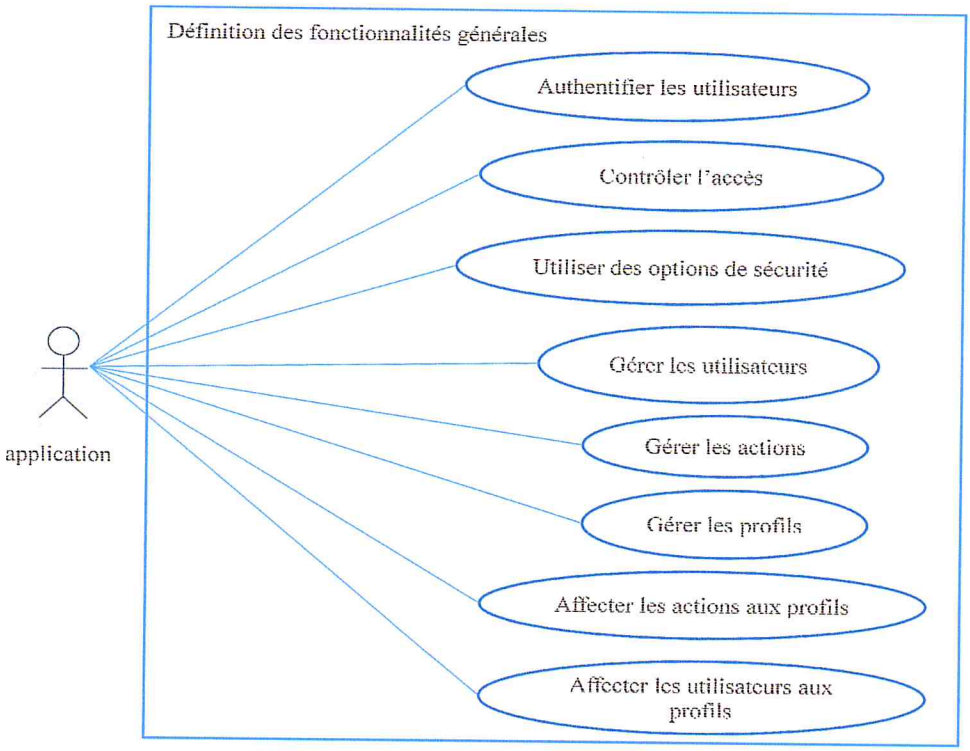


***Figure 3.1** : Diagramme de cas d'utilisation générale*

Voici une explication de chaque cas motionné dont ce diagramme :

- **Authentifier les utilisateurs** : permet d'authentifier les utilisateurs d'un système en utilisant plusieurs manières (par certificat, par mot de passe).

- **Contrôler l'accès** : permet de contrôler l'accès des utilisateurs aux informations et aux opérations.
- **Utiliser des options de sécurité** : permet de distinguer les différentes options de sécurité que notre composant va implémenter parmi eux gestion des Captcha, gestion des sessions.
- **Gérer utilisateurs, Gérer actions, Gérer profiles** : gère les utilisateurs d'un système, ainsi ses profils et les actions qui constitue chaque profil.
- **Affecter les actions aux profils** : ce cas permet d'affecter un ensemble d'actions à un profil.
- **Affecter les utilisateurs aux profils** : ce cas permet d'affecter un utilisateur à un profil.

2) Diagrammes de cas d'utilisation détaillés :

Dans cette partie nous étudions chaque cas d'utilisation présenté dans la figure 3.1

A. Authentifier les utilisateurs :

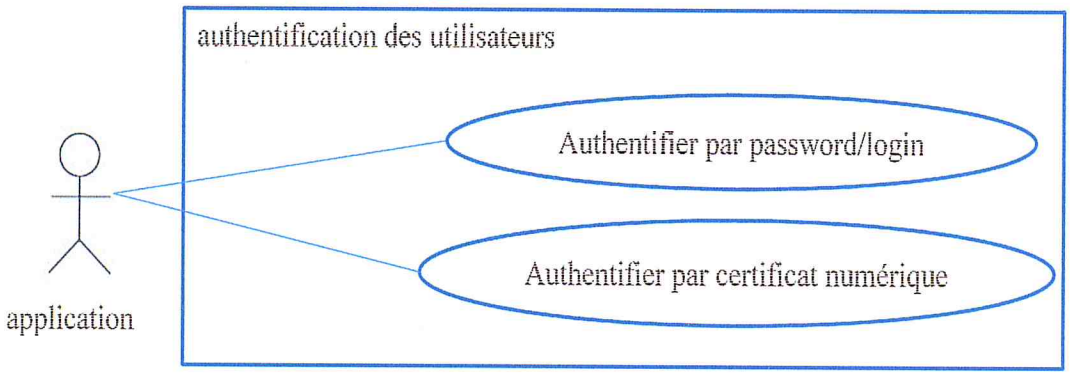


***Figure3.2**: Diagramme de cas d'utilisation « authentification des utilisateurs »*

L'authentification des utilisateurs se fait soit par le nom d'utilisateur et le mot de passe, soit par certificat numérique, cette authentification a un grand niveau de sécurité par rapport à la première.

B. Contrôler l'accès :

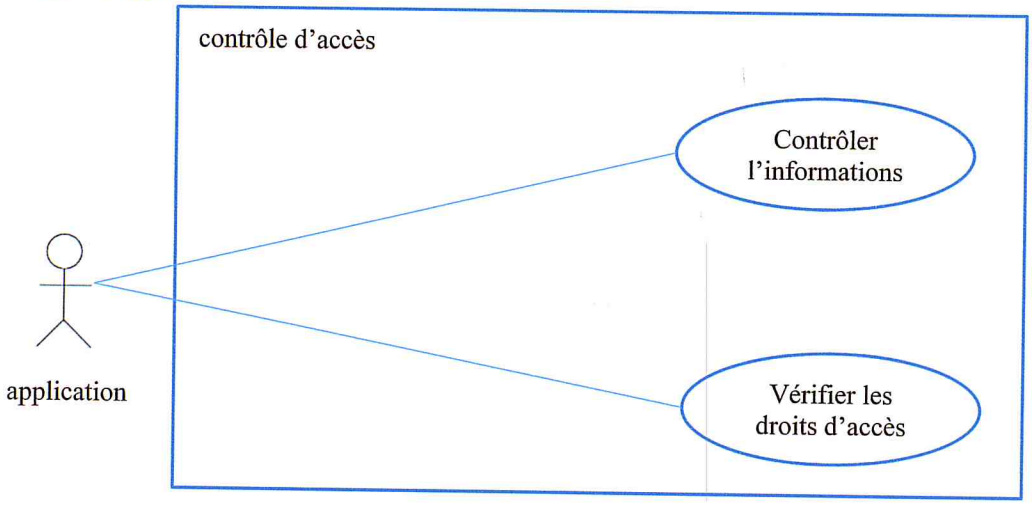


***Figure3.3**: Diagramme de cas d'utilisation « contrôle d'accès »*

- **Contrôler l'information** : ce cas permet de l'imiter l'accès a l'information que sa soit en lecture ou en écriture.
- **Vérifier les droits d'accès** : permet de vérifier si un certain utilisateur a le droit d'accéder à des informations ou d'effectuer certaines opérations.

C. Utiliser des options de sécurité :

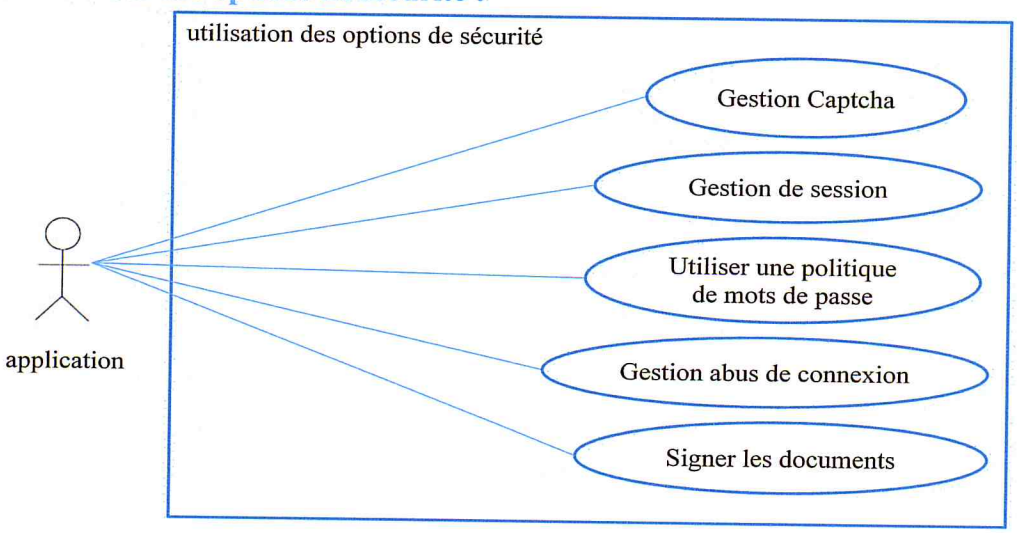


***Figure3.4**: Diagramme de cas d'utilisation « utilisation des options de sécurité »*

Voici une explication de chaque cas motionné dont ce diagramme :

- **Gestion captcha** : permet de manipuler des captcha (crée, supprimer, vérifier).
- **Gestion de session** : permet de manipuler des sessions de travail d'utilisateur (crée session, supprimer session, bloquer session).
- **Utiliser une politique de mot de passe** : oblige à suivre un certain format de mot de passe, par exemple :le mot de passe doit contenir des chiffre ,des majuscules, caractères spéciaux.
- **Gestion abus de connexion** : permet de bloquer les abus de connexion en faisant appel aux Captcha.
- **Signer les documents** : permet de signer un document pour assurer son intégrité et authenticité.

D. Gestion des utilisateurs :

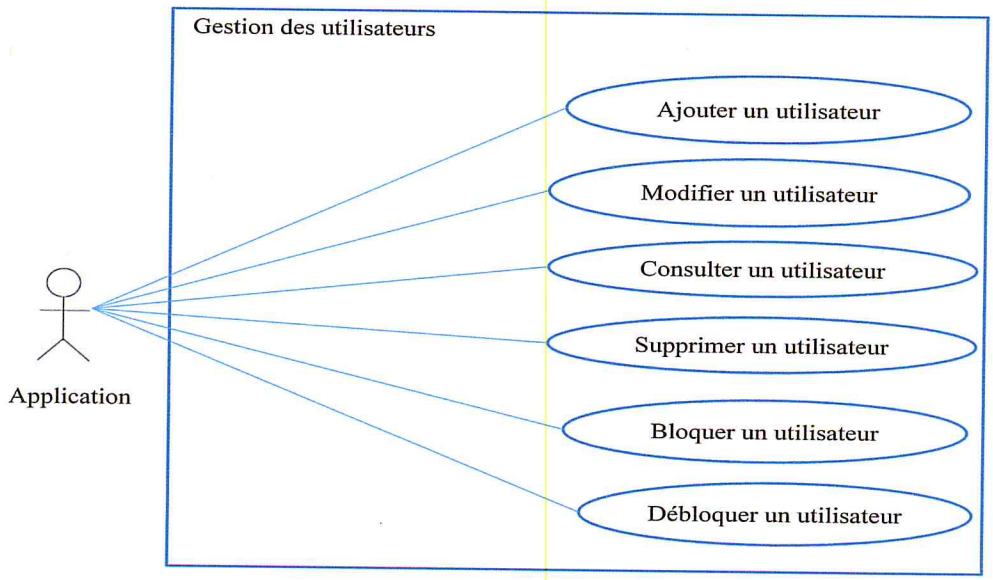


***Figure3.5**: Diagramme de cas d'utilisation « Gestion des utilisateurs »*

Les cas « ajouter un utilisateur », « modifier un utilisateur », «consulter un utilisateur » et « supprimer un utilisateur » permet dont cet ordre de faire des opérations d'ajout, modification, consultation et suppression sur les informations d'un utilisateur.

Alors que « bloquer un utilisateur » et « débloquer utilisateur » permet de bloquer et débloquer un utilisateur par des désistions venant de l'administrateur.

E. Gestion des actions :

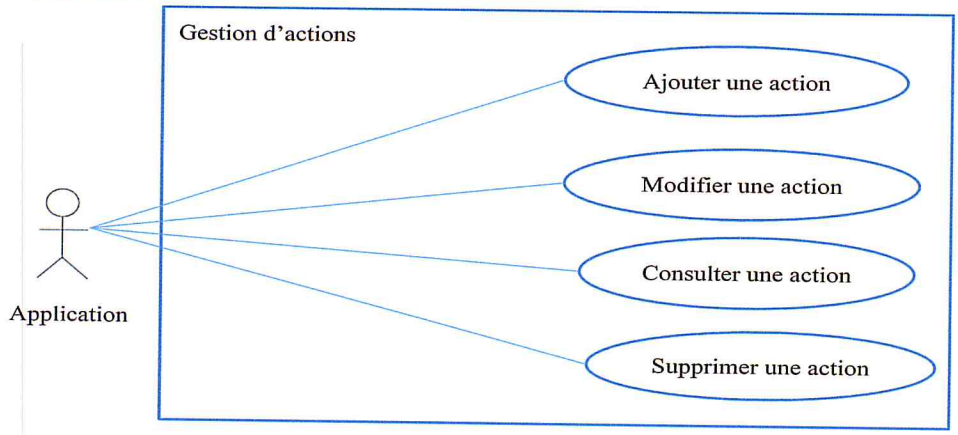


***Figure3.6** : Diagramme de cas d'utilisation « Gestion d'actions »*

Les cas « ajouter un action », « modifier un action », «consulter un action » et « supprimer un action» permet dont cet ordre de faire des opérations d'ajout, modification, consultation et suppression sur les actions.

F. Gérer les profils :

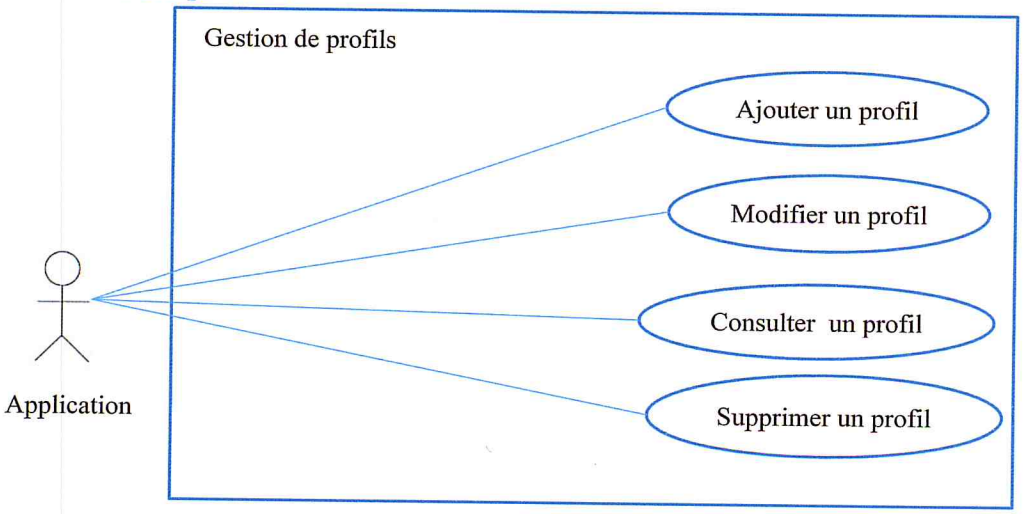


***Figure3.7** : Diagramme de cas d'utilisation « Gestion de profils»*

Les cas « ajouter un profile », « modifier un profile», «consulter un profile» et « supprimer un profile» permettent dont cet ordre de faire des opérations d'ajout, modification, consultation et suppression sur les profils.

VI. La vue externe du composant de sécurité :

Notre composant de sécurité a besoin d'un certificat numérique pour authentifier les utilisateurs, ainsi que des documents PDF pour les signer numériquement, de même il fournie plusieurs services.

On résume ces services requis et fournis dans la figure suivante :

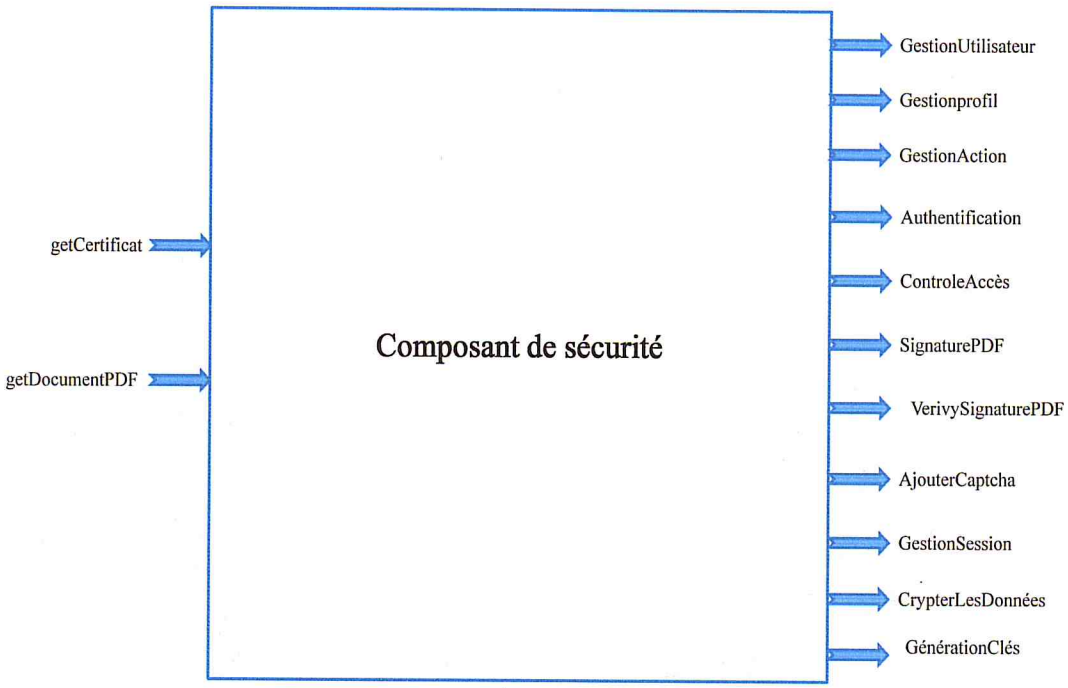


***Figure3.8**: vue externe du composant de sécurité.*

VII. Conclusion :

Dans cette étape qui est l'analyse des besoins nous avons déterminé les fonctionnalités globales de notre composant ainsi que ses services fournis et requis. Dans le prochain chapitre nous allons présenter toutes les étapes nécessaires pour la conception de notre composant.

Chapitre 4:

Conception

I. Introduction :

Nous présenterons dans ce chapitre les divers aspects liés à la conception de notre composant qui devra répondre aux objectifs définis dans le chapitre précédent.

Nous commencerons tout d'abord par introduire la forme générale du modèle de composant que nous utilisons dans notre conception. Ce modèle simple s'inspire du modèle de composant de l'approche IASA et de FRACTAL.

II. Modèle de composant :

Le modèle de composant que nous utilisons utilise une organisation bien précise de la vue interne. Cette vue distingue de manière explicite entre deux aspects liées au composant : L'aspect construction (mise en place et initialisation) du composant et l'aspect métier. Le métier est représenté par toutes les fonctionnalités qui seront accessibles une fois le composant mis en place (ou instancié). De ce fait la vue interne du composant est divisée en 2 partie : La partie Contrôle et la partie métier (Figure 4.1). Dans la partie contrôle nous trouverons tous les composants qui agissent dans le cadre de la construction et l'initialisation du composant. L'initialisation du composant peut comprendre l'instanciation des divers composants formant la partie métier.

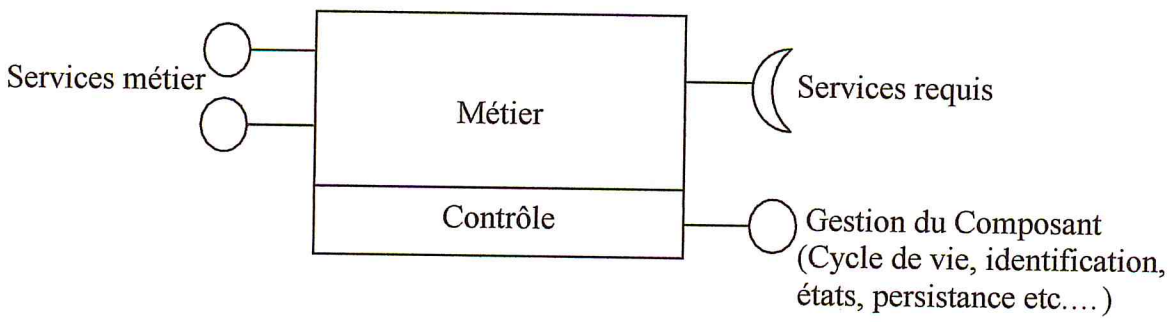


***Figure 4.1** : Modèle de composant utilisé*

III. Identification d'un composant :

En général, un composant, comme un objet, possède un cycle de vie inférieure à l'application. Il est instancié puis détruit dans le contexte de l'exécution d'une même application. L'identification de ces composants à durée de vie limitée par l'application est souvent réalisée à l'aide de référence directe (nom de variable) ou indirecte (pointeurs en C++, référence Java) par l'application durant l'exécution du programme. Cependant, dans certaines situations, les composants ont une durée de vie plus grande que l'application dans laquelle ils sont utilisés. Ces composant disposent d'états ou gèrent des données qu'il est nécessaire de retrouver lors du relancement de l'application ayant instancié ces composants. Ainsi, la terminaison d'une application ne veut nullement dire qu'un composant de cette catégorie a disparu. Ce type de composant possède des états qui doivent être restauré une fois le composant instancié dans une application. Plus encore, un même composant pourrait être partagé entre plusieurs applications.

De ce fait, l'identité du composant ne doit pas être une référence du programme mais un nom à travers lequel une application le retrouve lors de son instanciation. Ainsi un composant doit posséder un nom qui l'identifie de manière unique dans le monde réel

IV. Contrôle de l'instanciation :

Un composant ayant une identité particulière peut être instancié par diverses applications. Dans certains cas, pour protéger l'information que gère ce composant, il n'est pas recommandé que n'importe quelle application puisse l'utiliser. Ainsi il est nécessaire de faire un contrôle lors de l'instanciation du composant. Ce contrôle permettra de retrouver les applications ayant le droit de l'instancier. La technique la plus usuelle et efficace pour faire ce contrôle est la technique d'authentification de l'application désirant instancier le composant. Ainsi un nom d'utilisateur et un mot de passe serait requis pour instancier le composant. En général le mot de passe et le nom utilisé coïncident souvent avec

le nom d'utilisateur et le mot de passe de l'administrateur de l'application dans laquelle le composant a été instancié.

V. Scénario d'instanciation du composant :

Nous distinguons deux grands scénarios :

- Instanciation initiale du composant
- Instanciation ordinaire

L'instanciation initiale représente la création effective du composant. Durant cette création il est nécessaire de spécifier le nom du composant qui devra être unique dans le monde ou évolue le composant. C'est durant cette création que les informations d'authentification de l'application sont définies soit de manière explicite ou implicite. Dans la définition explicite l'application doit indiquer le nom et le mot de passe qui seront utilisé pour instancier le composant par la suite. Dans la définition implicite, le composant utilise des valeurs par défaut qui sont « adminuser » et « adminpass ». Ces valeurs nécessitent une opération de modification de l'appart de l'application instanciée.

Dans l'instanciation ordinaire, un nom de composant existant doit être indiqué ainsi qu'un nom d'utilisateur et un mot de passe dans le cas ou le composant ne doit être instancié que par les ayant droits. L'interface de contrôle présente ainsi les opérations suivantes indiquées sur le schéma de la fig. L'écriture Java de cette interface est reportée par la figure 2

```
Interface  IControleCmp {
                build(String  cmpName)  /* Comoosant persistant
                build (String cmpName, String adminUser, Strin adminPass)
/* Composant persistant avec authentification
                build()  /*Non persistant
                initialize()
                destroy()

                clone(String cmpName)
```

VI. Architecture interne du composant de sécurité :

Notre composant de sécurité se décompose en sous composants, et chaque sous composant possède des interfaces requises et fournies. Les dépendances entre composants permettent de mettre en évidence la réutilisation de composants.

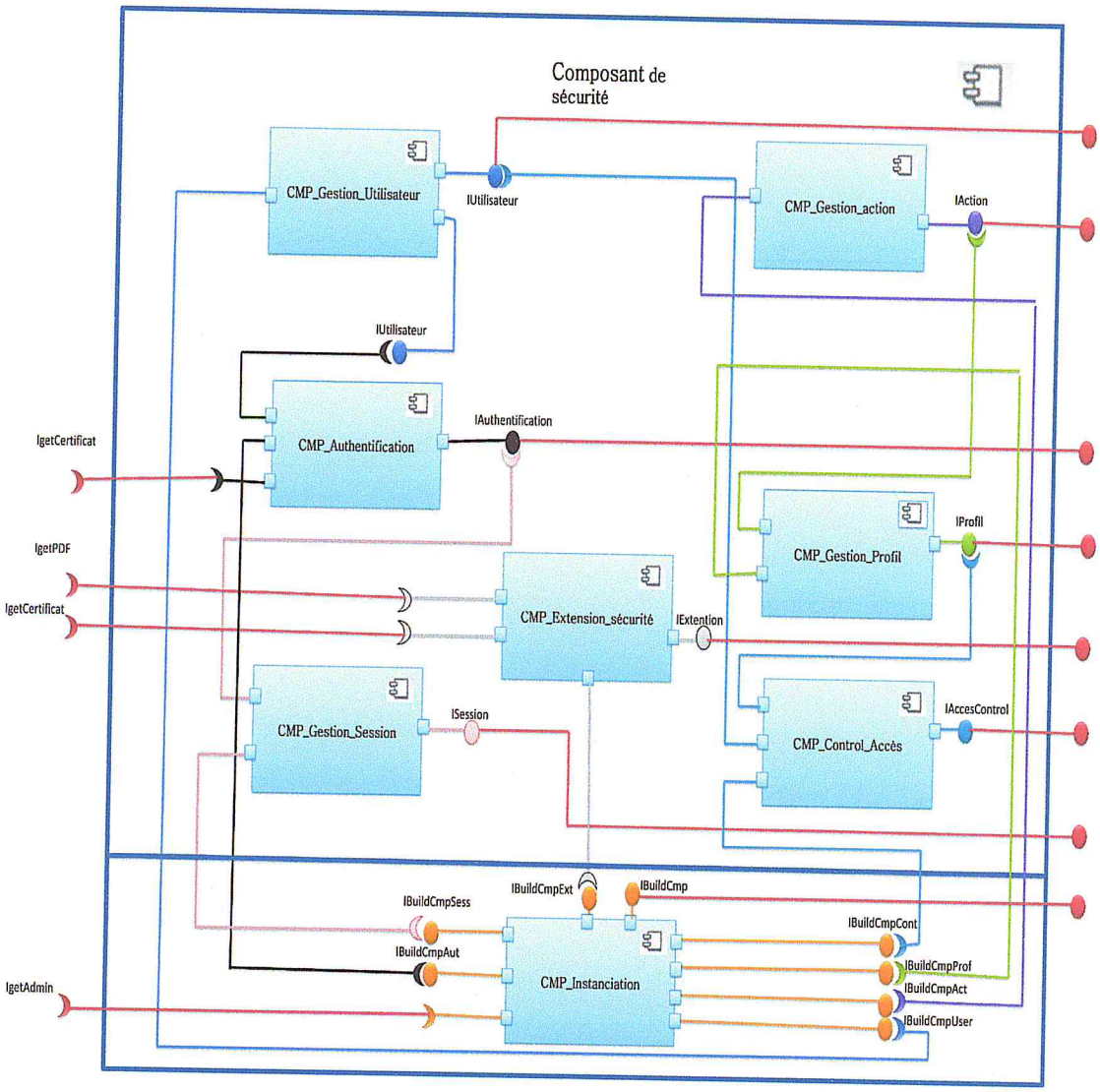


Figure 4.2 : *Diagramme du composant général*

Pour ce qui suit nous représentons pour chaque sous composant sa vue dynamique sous forme des diagrammes d'activités en détaillant toutes les interfaces fournies pas ces composants, ainsi que sa vue statique sous forme de diagramme de classes inclus dans un package pour montrer que les sous composants sont organisés en packages.

VII. Etude détaillé des sous composants :

1) Le composant d'instanciation:

Ce composant se trouve dans la partie contrôle du composant de sécurité, il prend en charge l'instanciation et le contrôle lors de l'instanciation du composant, il fournie l'interface suivante :

A. Instancier le composant de sécurité :

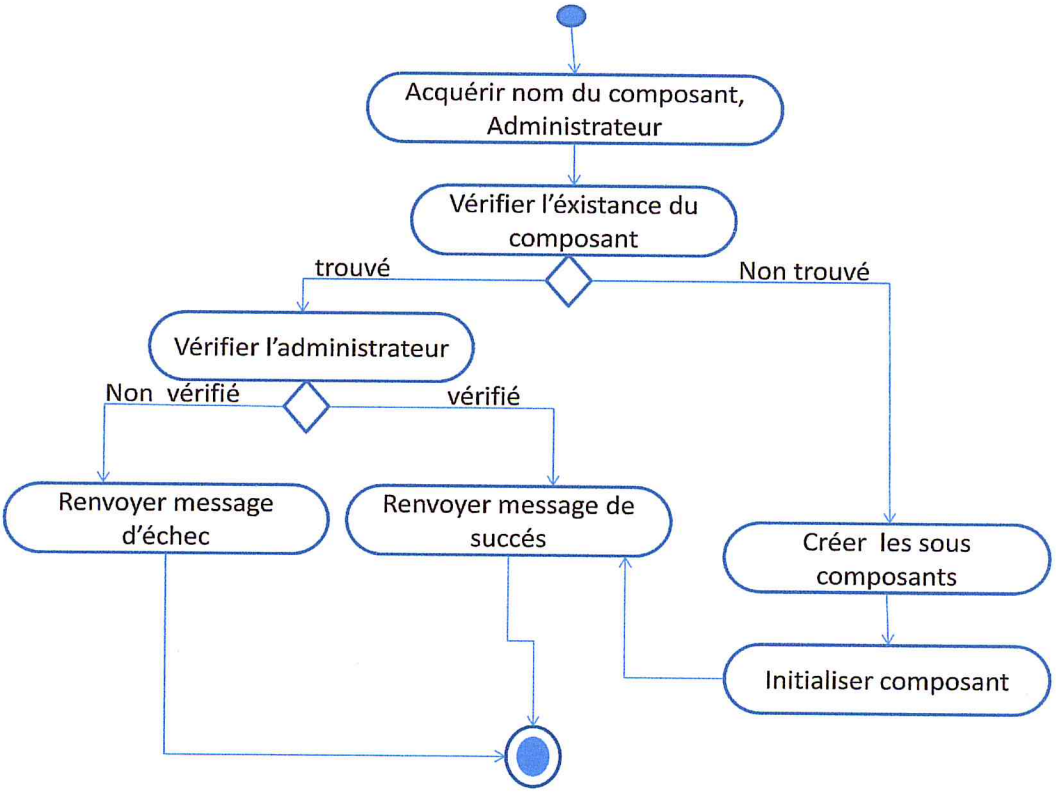


Figure 4.3 : *Diagramme d'activité « instancier le composant »*

➢ **Diagramme de classes :**

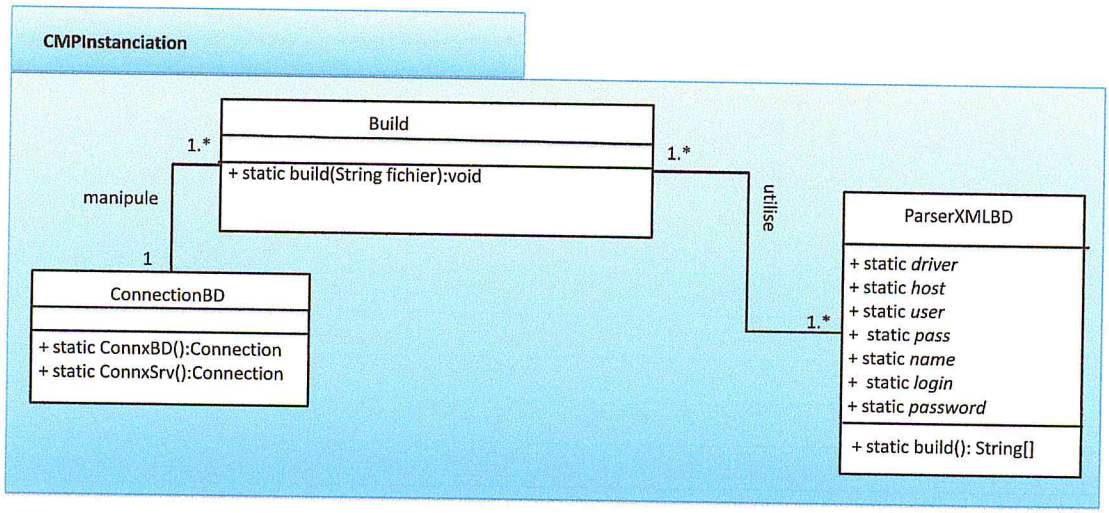


Figure 4.4 : *Diagramme de classes « instanciation du composant »*

2) Le composant gestion d'utilisateur :

Le composant (CMP_Gestion_Utilisateur), permet de gérer tous les utilisateurs d'un système, il fourni les interfaces suivantes :

A. Ajouter un utilisateur :

Cet interface permet d'ajouter un nouveau utilisateur on passant en premier par vérifier si cet utilisateur existe déjà, s'il n'existe pas ces informations seront ajoutées sinon un message d'erreur sera renvoyer.

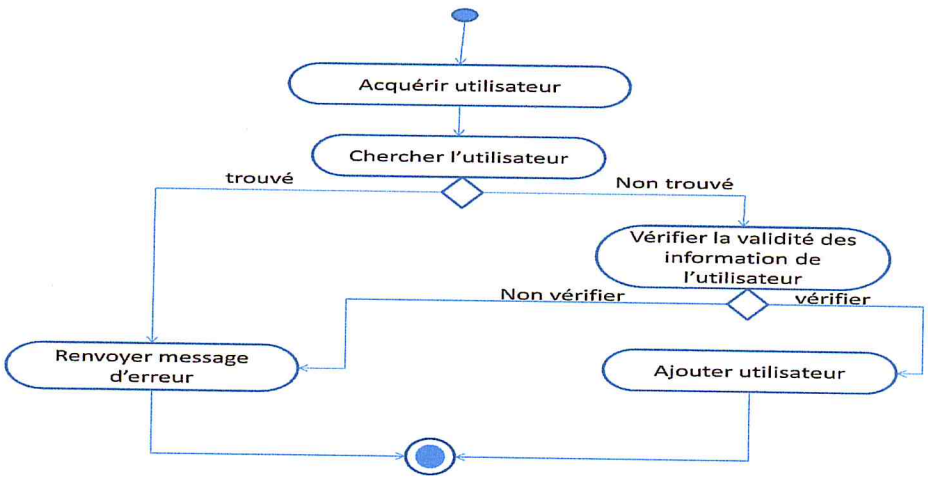


Figure 4.5 : *Diagramme d'activité « ajouter un utilisateur »*

B. Chercher un utilisateur :

Cet interface permet de retourner les informations d'un utilisateur bien précis on le cherchant par sont login si il n'existe pas un message d'erreur sera renvoyer.

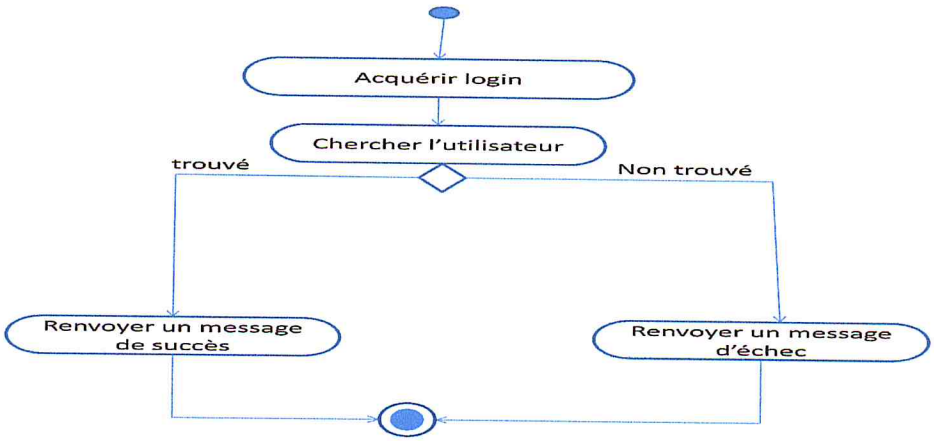


Figure 4.6 : *Diagramme d'activité « chercher un utilisateur »*

C. Modifier un utilisateur :

Cet interface permet de modifier les informations d'un utilisateur cherché par login puis renvoyer un message de succès on cas d'échec de recherche un message d'erreur est renvoyer.

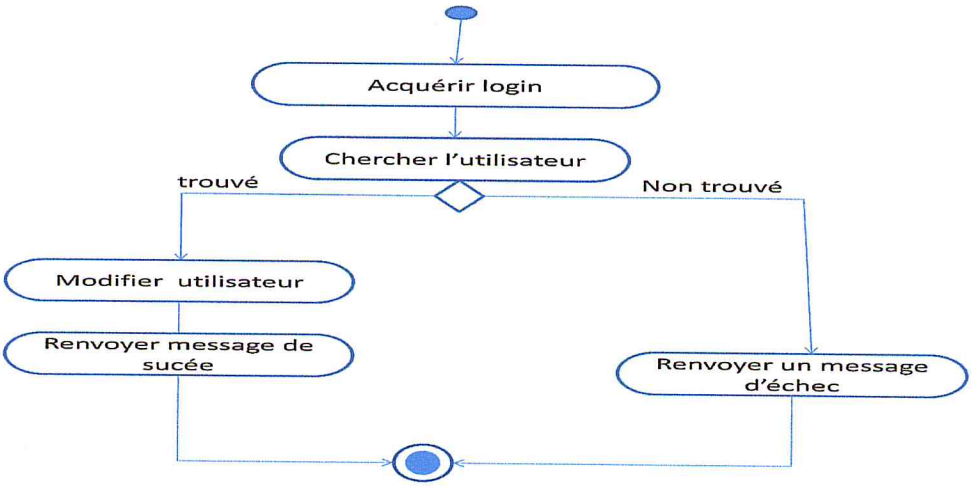


Figure 4.7: *Diagramme d'activité « modifier un utilisateur »*

D. Supprimer un utilisateur :

Cette interface permet de supprimer un utilisateur du système si ce dernier existe.

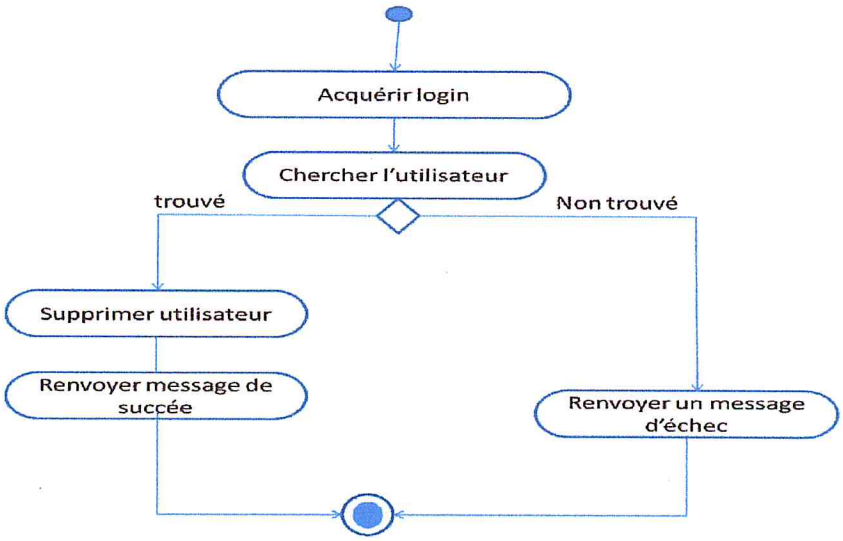


Figure 4.8 : *Diagramme d'activité « supprimer un utilisateur »*

E. Bloquer un utilisateur :

Cet interface permet de bloquer un utilisateur suite à des décisions différentes venant de l'administrateur en commençant par une recherche de l'utilisateur puis transformer sont état à l'état bloqué s'il est active sinon renvoyer un message d'échec.

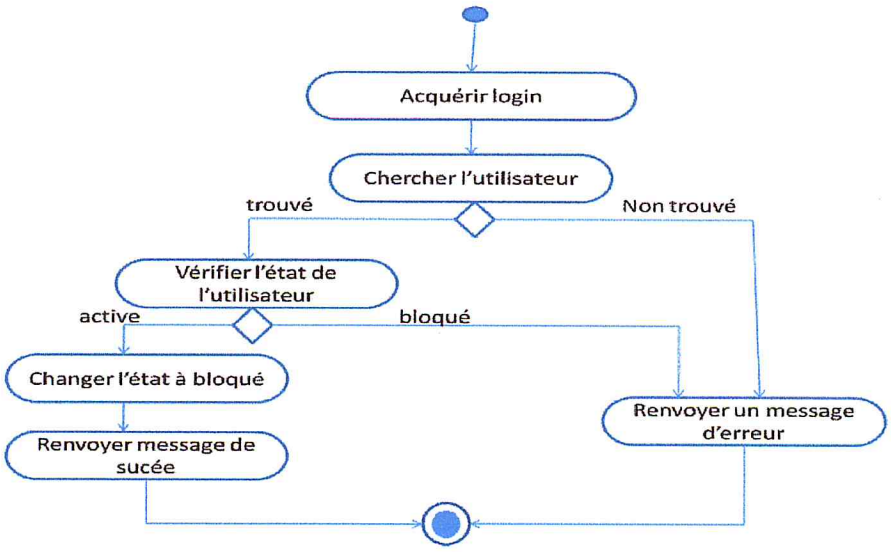


Figure 4.9 : *Diagramme d'activité « bloquer un utilisateur »*

F. Débloquer un utilisateur :

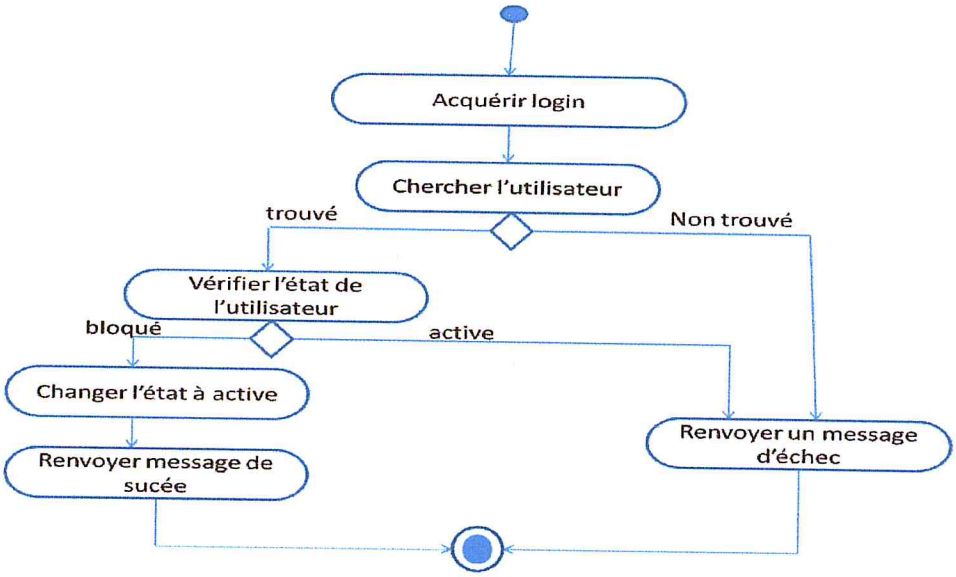


Figure 4.10 : *Diagramme d'activité « débloquer un utilisateur »*

G. Assigner un utilisateur à un profil :

Cet interface permet d'accorder à un utilisateur un profil en cherchant d'abord si l'utilisateur existe puis si le profil existe si le profile n'existe pas elle invoque l'opération qui le fait puis termine le processus d'affectation de profil et envoie un message de succès sinon un message d'échec est envoyer.

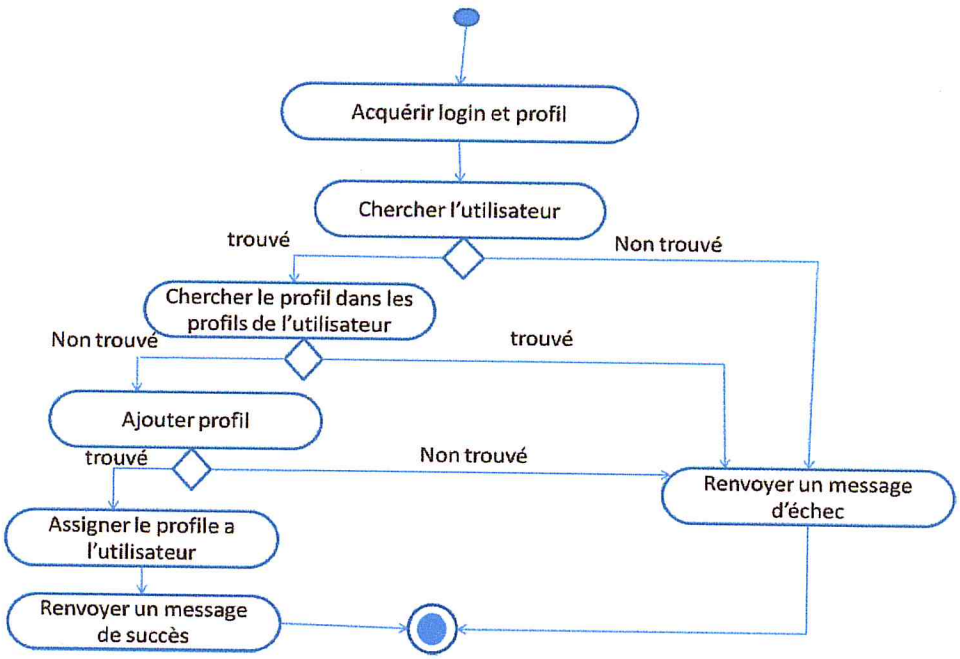


Figure 4.11 : *Diagramme d'activité « assigner un utilisateur à un profil »*

➢ **Diagramme de classes :**

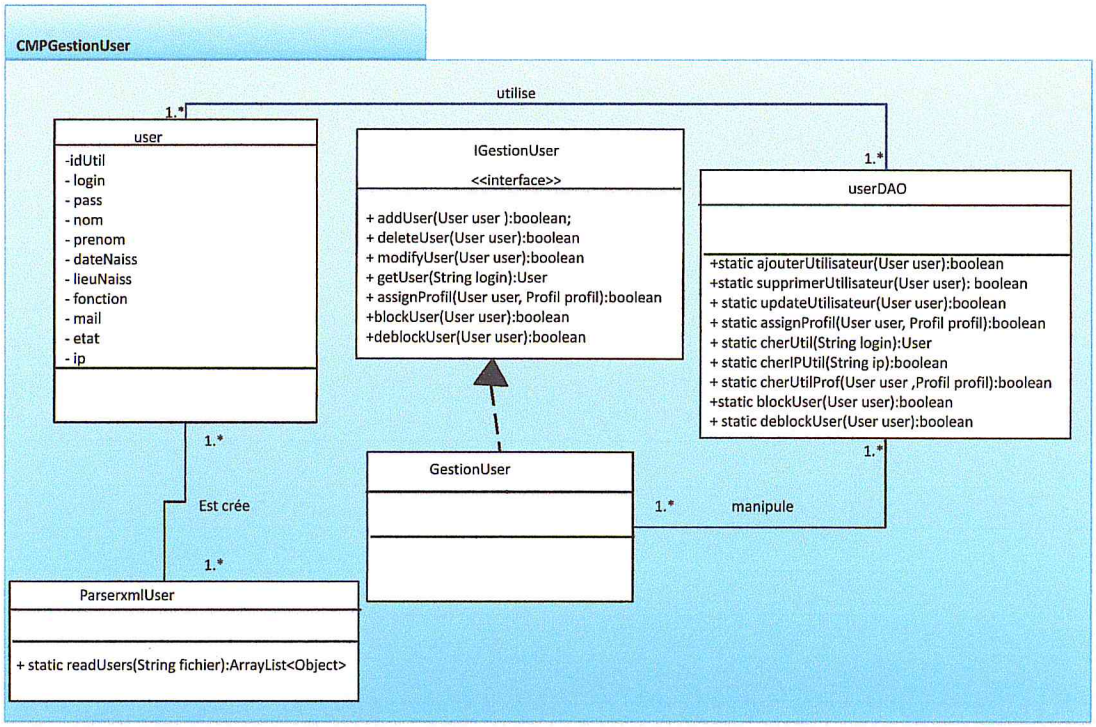


Figure 4.12 : *Diagramme de classe « gestion utilisateur »*

3) Le composant gestion de profil :

Ce composant permet de gérer tous les profils manipulés par l'application, il fournie les interfaces suivantes :

A. Ajouter un profil :

Cet interface permet d'ajouter un nouveau profile après avoir vérifié qu'il n'existe pas déjà.

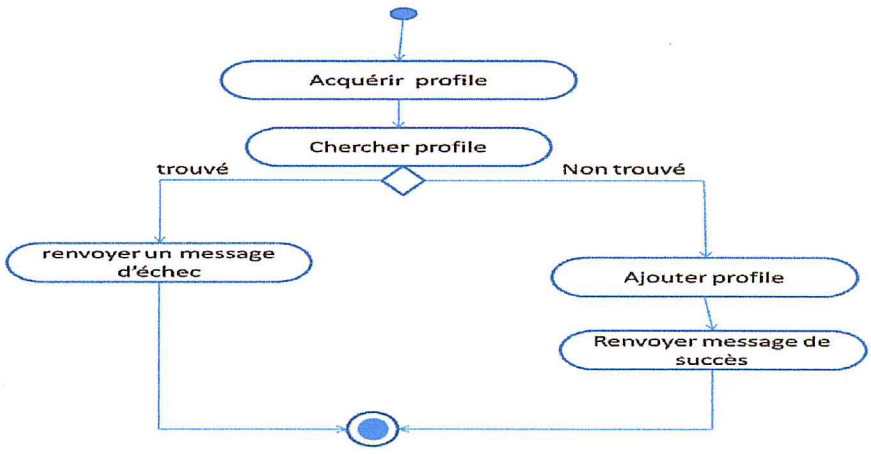


Figure 4.13 : *Diagramme d'activité « ajouter un profil »*

B. Chercher un profil :

Cette interface permet de chercher un profil parmi les profils définis par l'application.

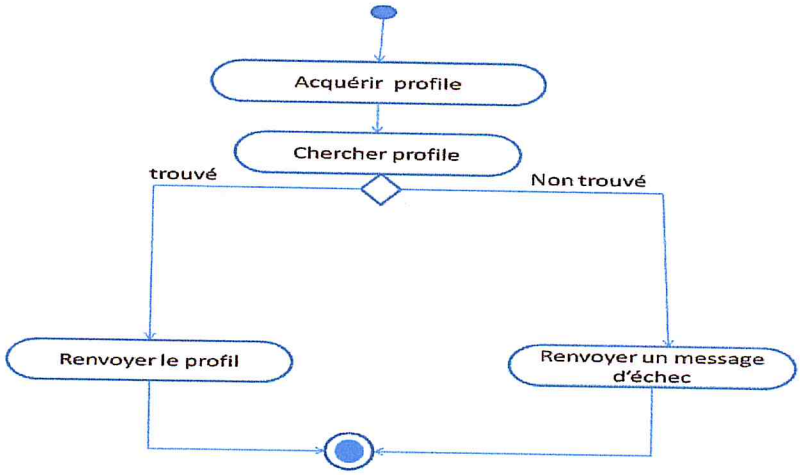


Figure 4.14 : *Diagramme d'activité « chercher un profil »*

C. Modifier un profil :

Cette interface permet de modifier un profil s'il existe déjà sinon un message d'échec sera renvoyé.

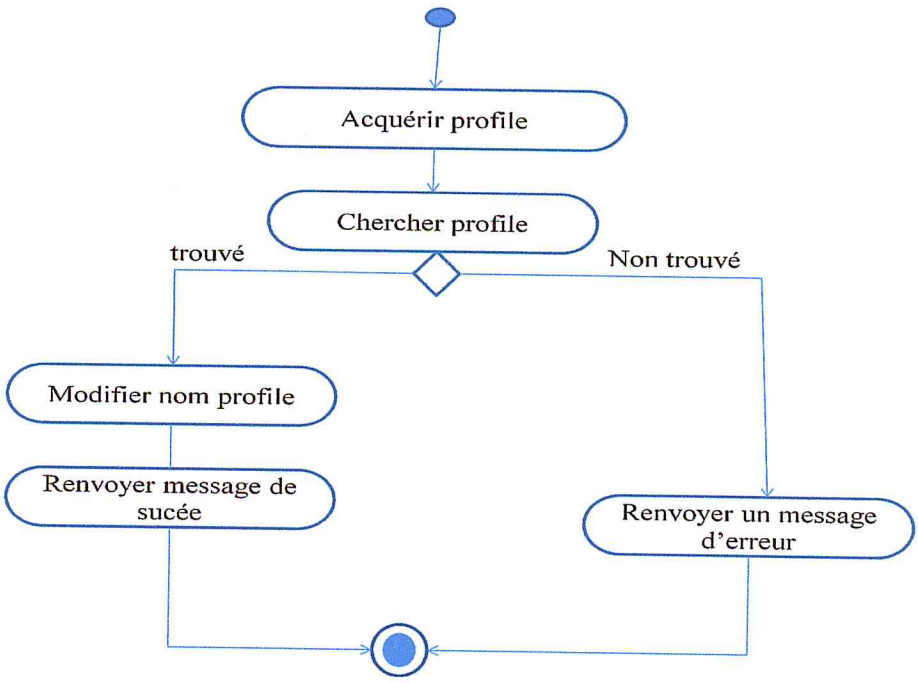


Figure 4.15 : *Diagramme d'activité « modifier un profil »*

D. Supprimer un profil :

Cet interface permet de supprimer un profil puis renvoyer un message de succès de suppression ou d'échec si la recherche du profile a échoué.

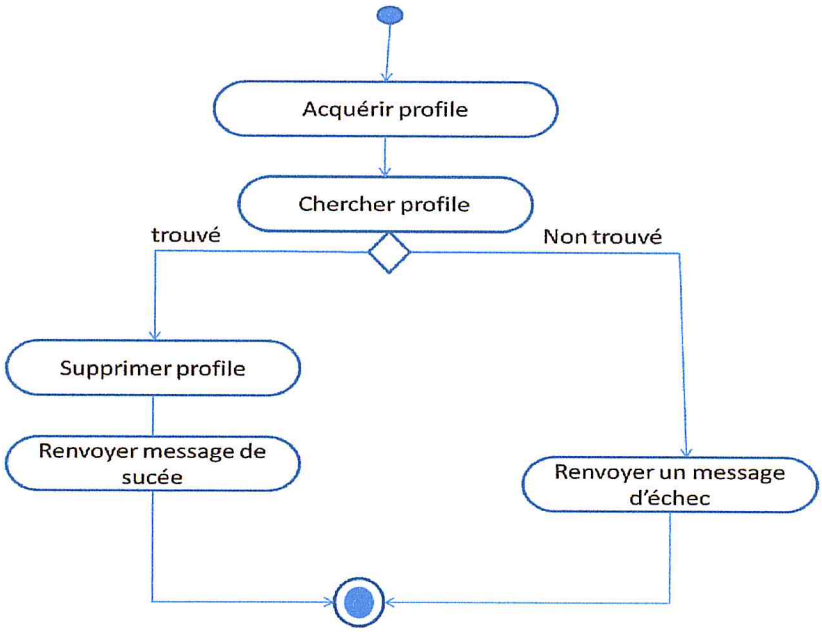


Figure 4.16 : *Diagramme d'activité « supprimer un profil »*

E. Assigner une action à un profil :

Cet interface permet d'accorder à un profil des actions en cherchant d'abord si le profil existe puis si l'action existe si l'action n'existe pas elle invoque l'opération qui le fait puis termine le processus d'affectation d'action et envoie un message de succès sinon un message d'échec sera renvoyé.

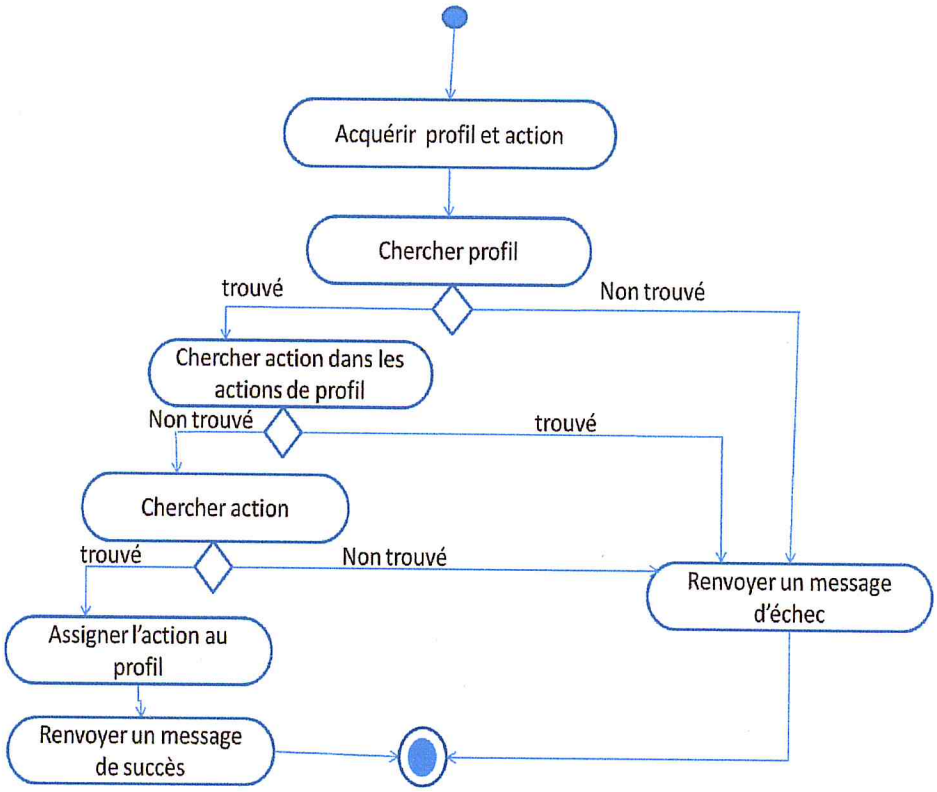


Figure 4.17 : *Diagramme d'activité « assigner une action à un profil»*

- **Diagramme de classes :**

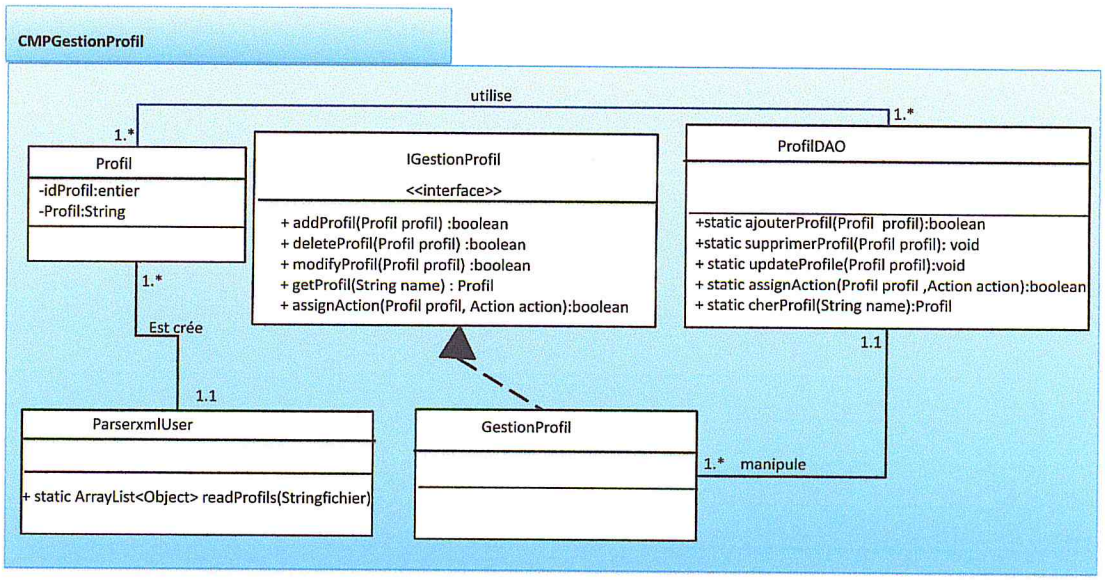


Figure 4.18 : *Diagramme de classes « gestion profil »*

4) Le composant gestion d'action :

Ce composant permet de gérer toutes les actions qui constituent les profils d'utilisateurs, il fournie les interfaces suivantes :

A. Ajouter une action :

Cet interface permet d'ajouter une nouvelle action en vérifiant si cette action n'existe pas déjà, sinon un message d'échec sera renvoyer.

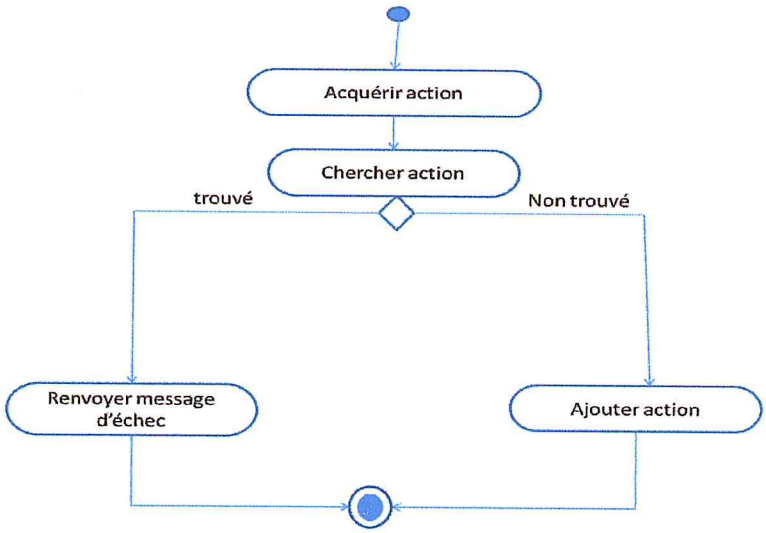


Figure 4.19 : *Diagramme d'activité « ajouter une action »*

B. Chercher une action :

Cette interface permet de retourner les informations d'une action bien précise en la cherchant par son identifiant et si il n'existe pas un message d'échec sera renvoyé.

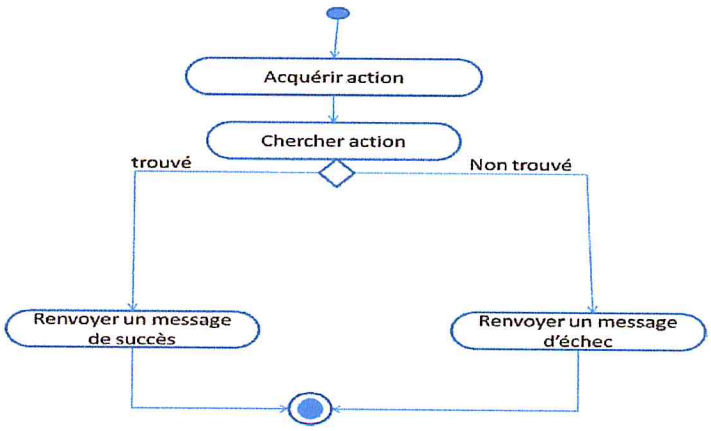


Figure 4.20 : *Diagramme d'activité « chercher une action »*

C. Modifier une action :

Cette interface permet de modifier les détailles d'une action chercher par sont identifiant puis renvoyer un message de succès on cas d'échec de recherche, un message d'erreur est renvoyer.

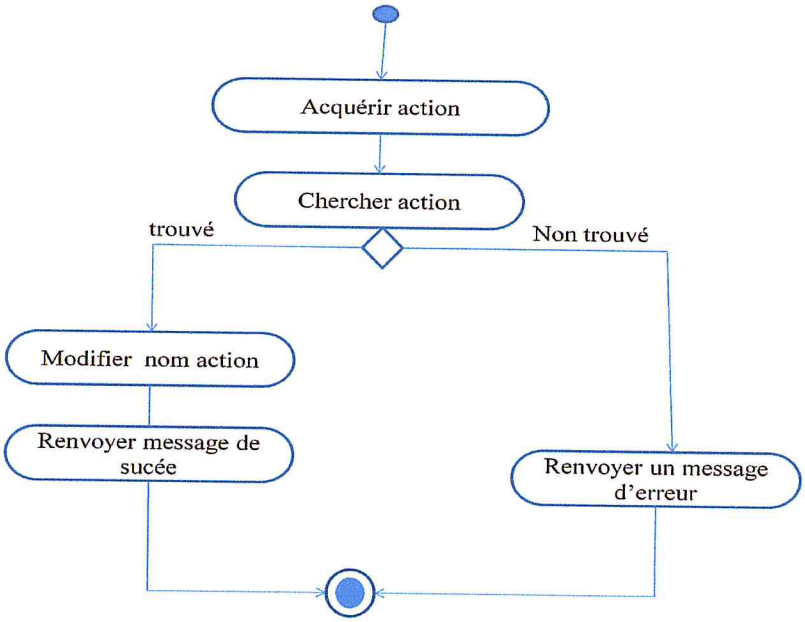


Figure 4.21: *Diagramme d'activité « modifier une action »*

D. Supprimer une action :

Cette interface permet de supprimer une action puis renvoyer un message de succès de suppression ou d'échec si la recherche de l'action a échoué.

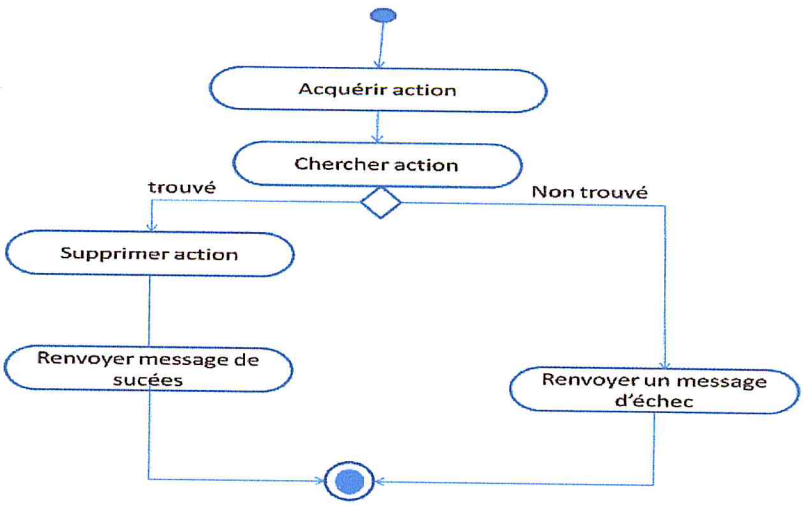


Figure 4.22 : *Diagramme d'activité « supprimer une action »*

➢ **Diagramme de classes :**

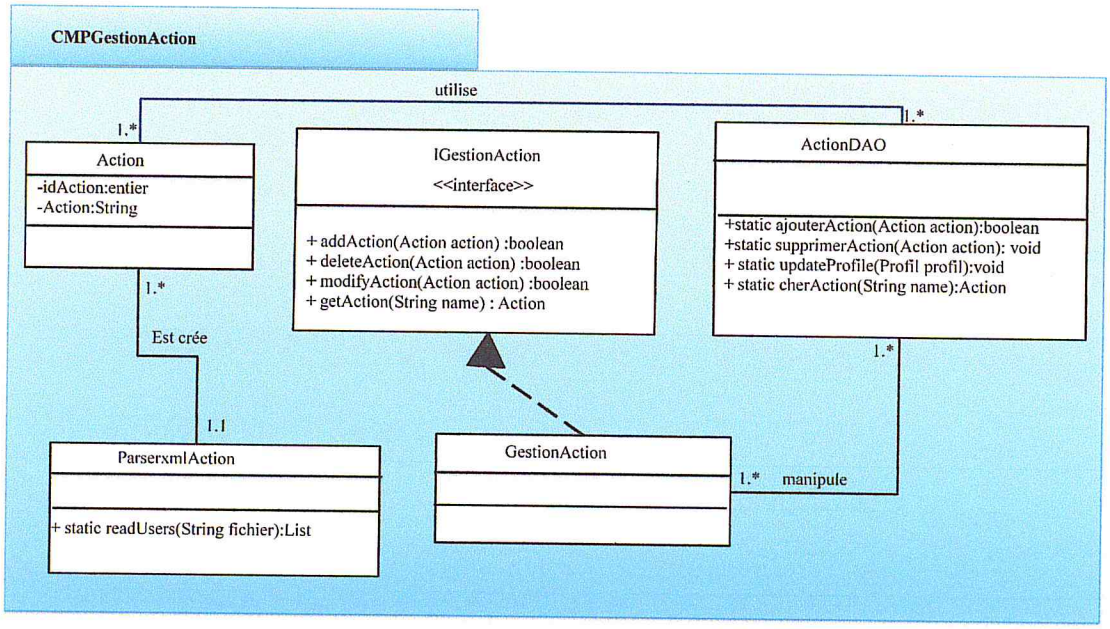


Figure 4.23 : *Diagramme de classes « gestion profil »*

5) Le composant gestion de session :

Ce composant gère les sessions d'utilisateurs après l'authentification.

A. Créer une session :

Cette interface permet de créer une session d'utilisateur, en ajoutant comme variable de session le login.

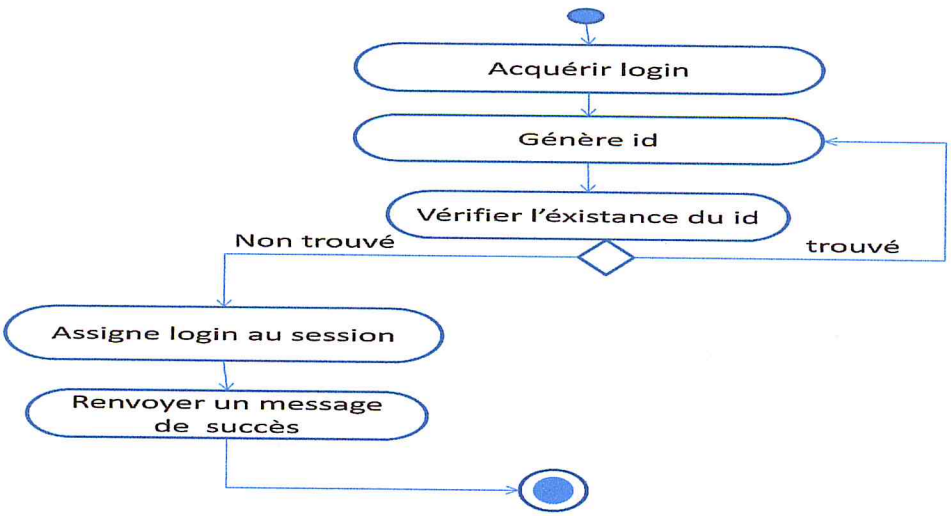


Figure 4.24 : *Diagramme d'activité « créer une session »*

B. Chercher une session :

Cette interface permet de chercher une session d'utilisateur par le login qui représente une variable de session.

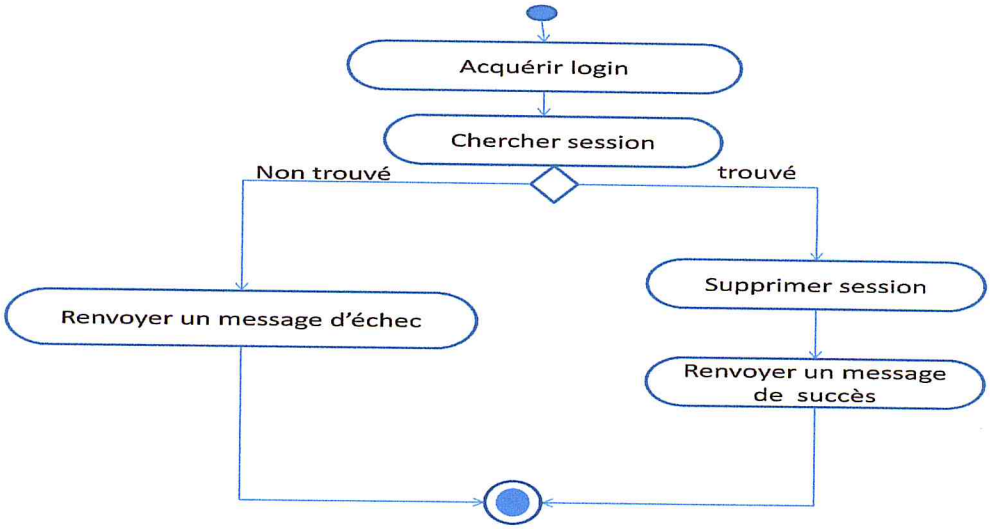


Figure 4.25 : *Diagramme d'activité « chercher une session »*

C. Supprimer une session :

Cette interface permet de supprimer une session si elle existe, sinon un message d'échec sera renvoyé.

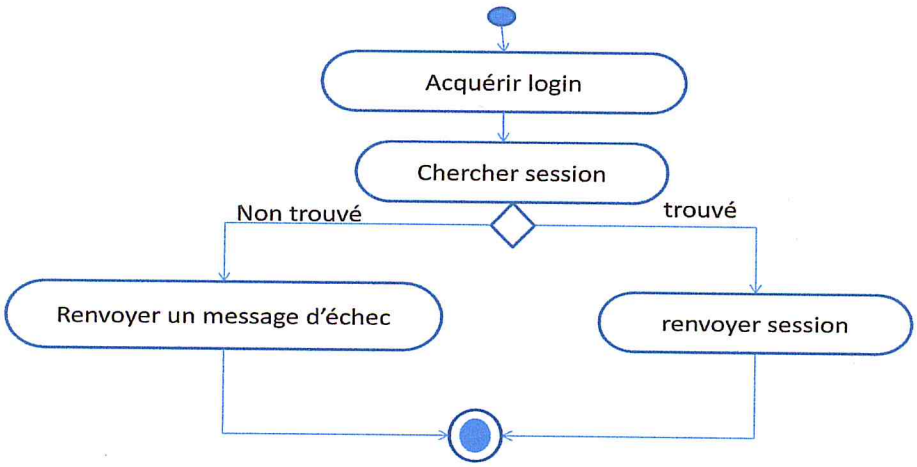


Figure 4.26 : *Diagramme d'activité « supprimer une session»*

➢ **Diagramme de classes :**

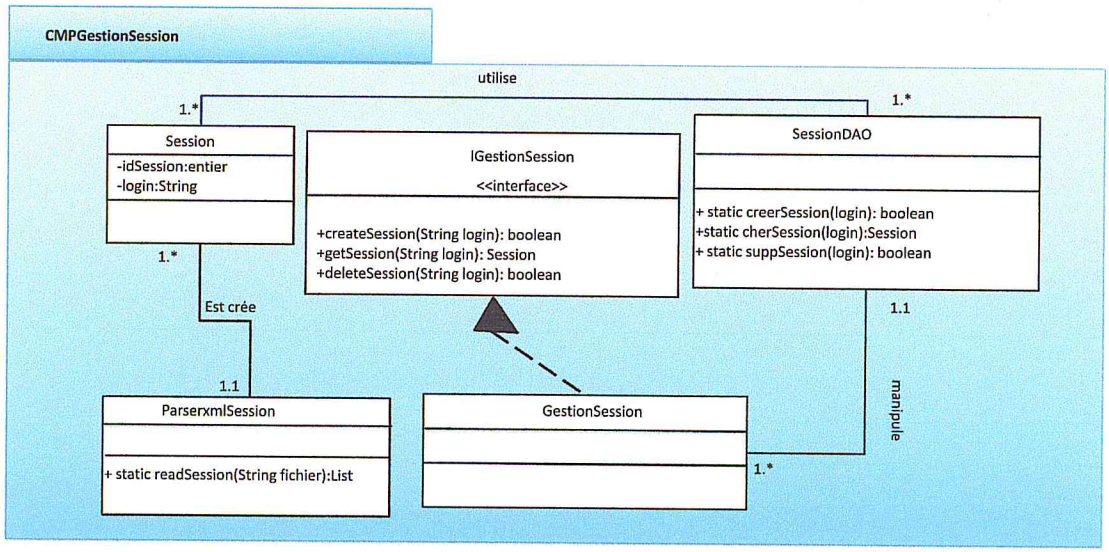


Figure 4.27 : *Diagramme de classes « gestion session »*

6) Le composant d'authentification :

Ce composant fournie deux types d'authentification selon les besoins et le niveau de sécurité de l'application à sécuriser :

A. Authentifier par login/password :

Cette interface permet de faire l'authentification avec mot de passe et login d'un utilisateur qui sont stockés dont la base de donnes.

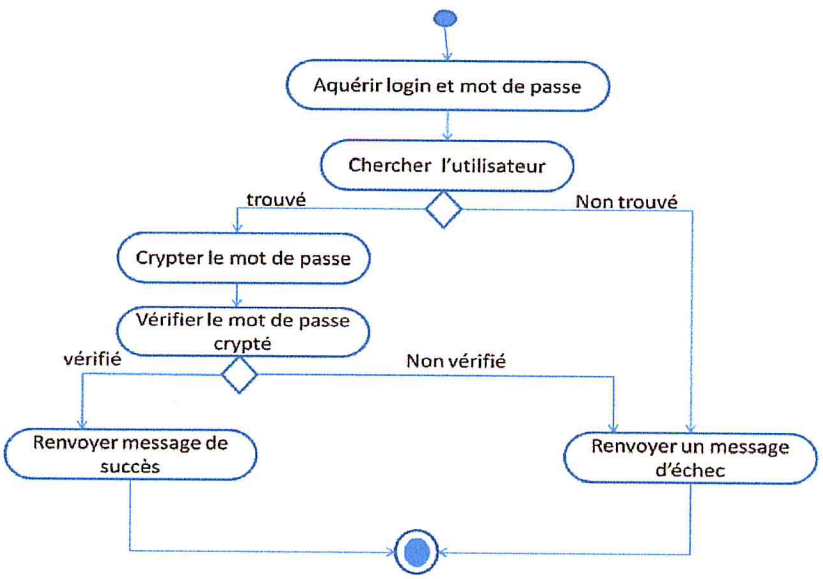


Figure 4.28 : *Diagramme d'activité « authentifier par login/password »*

B. Authentifier par certificat :

Cette interface permet d'authentifier les utilisateurs par leurs certificats numériques.

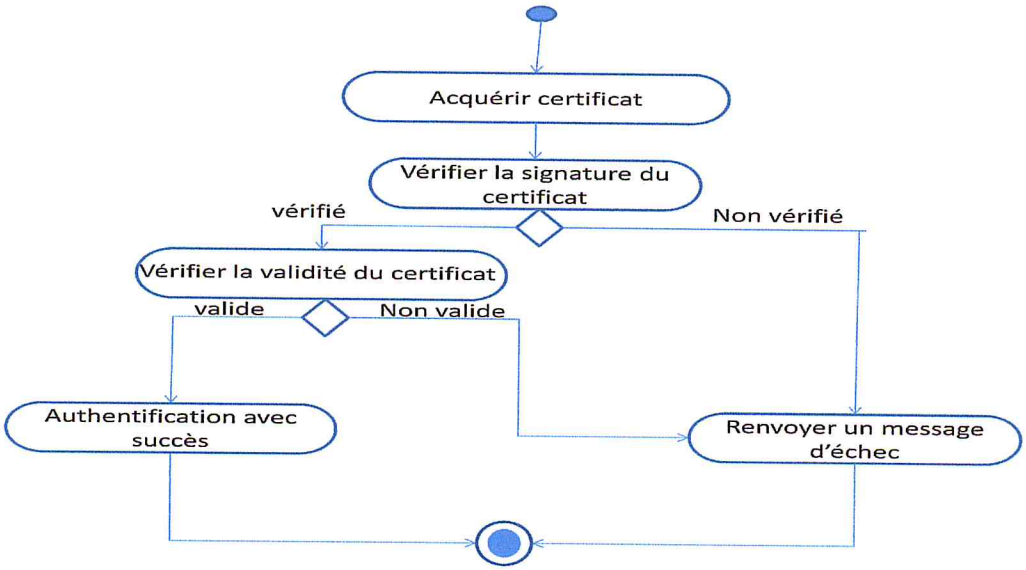


Figure 4.29: *Diagramme d'activité « authentifier par certificat »*

➢ **Diagramme de classes :**

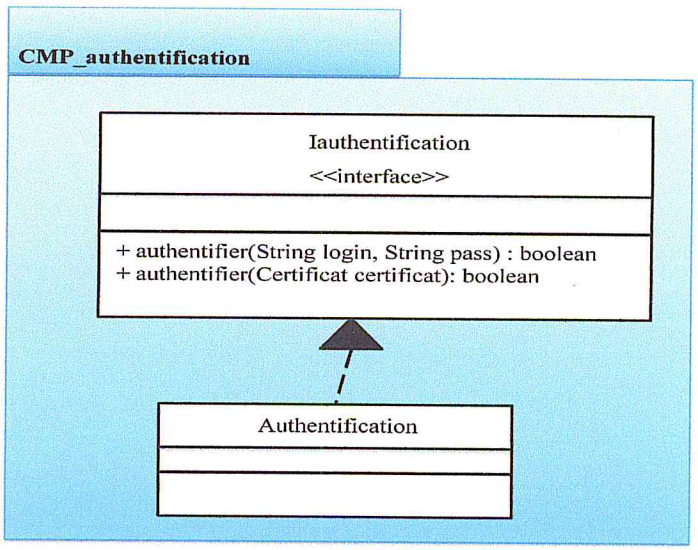


Figure 4.30 : *Diagramme de classes « authentifier par certificat »*

7) Le composant de contrôle d'accès :

Ce composant assure la gestion très fine des droits d'accès, il fournie les interfaces suivantes :

A. Accéder à une action :

Cette interface permet de vérifier le droit d'accès d'un utilisateur sur une action qui veut faire sur le système.

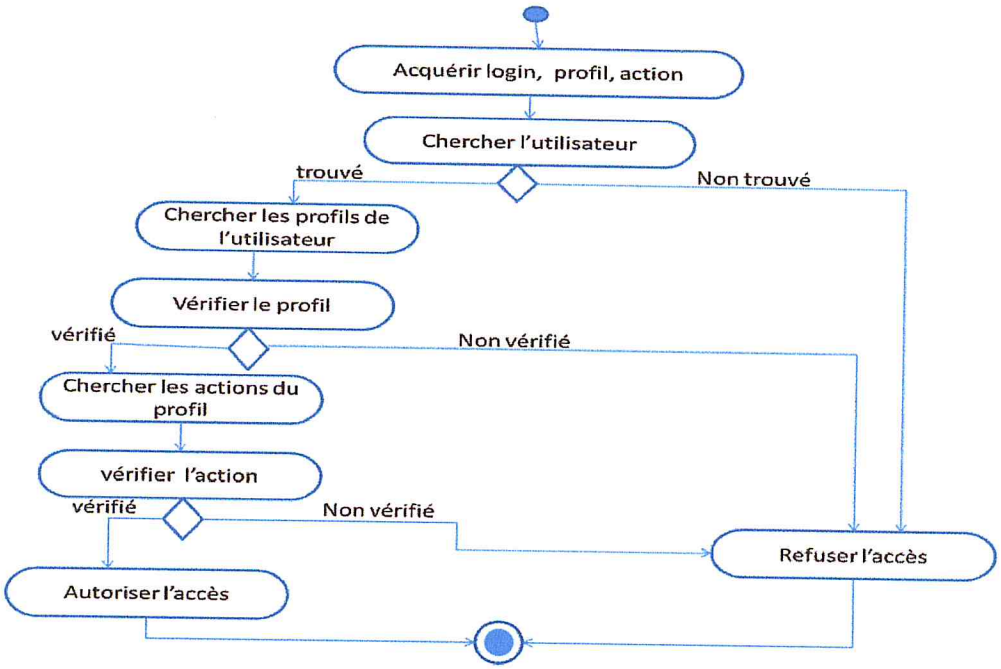


Figure 4.31 : *Diagramme d'activité « accéder à une action »*

B. Contrôler l'accès aux informations :

Cette interface permet de contrôler l'accès aux informations, c'est à dire, pour permettre à un utilisateur de consulter ou modifier une donnée il faut vérifier la chaine d'affectation d'une information à un utilisateur puis cet utilisateur avec le type d'accès (lecture ou écriture).

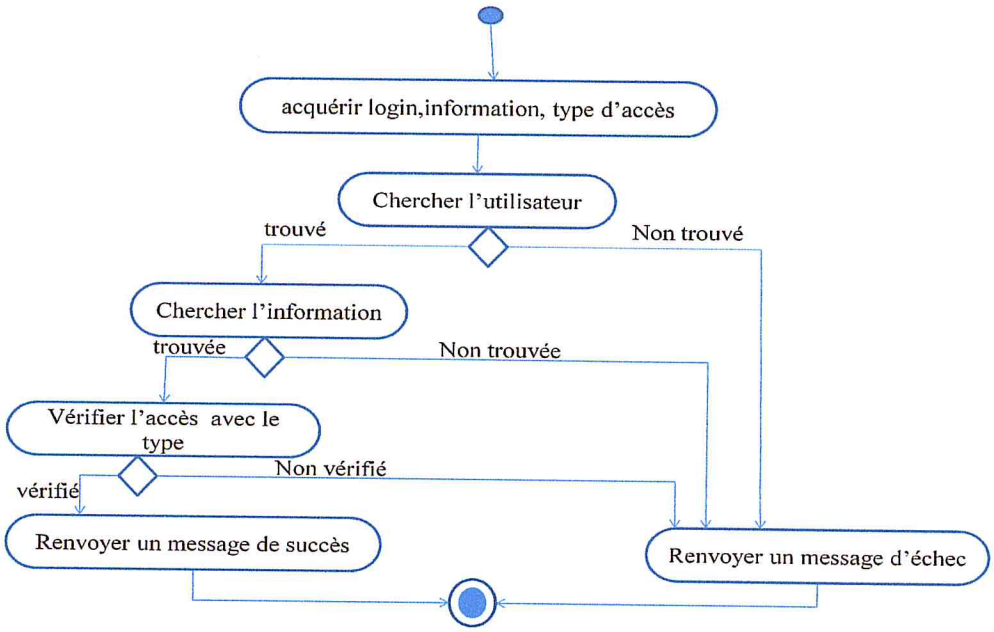


Figure 4.32 : *Diagramme d'activité « contrôler l'accès à l'information »*

C. Ajouter une information :

Cette interface permet d'ajouter une information après avoir tester si elle n'existe pas si non un message d'échec est renvoyé.

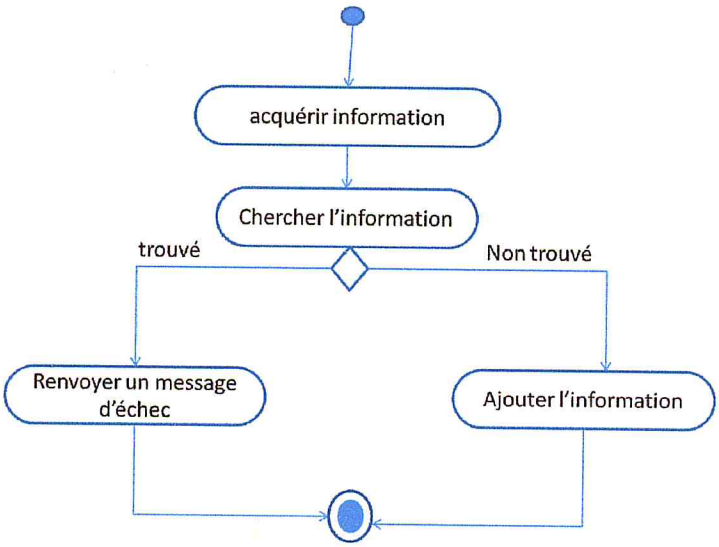


Figure 4.33 : *Diagramme d'activité « ajouter une information »*

D. Autoriser l'accès à une information:

Cet interface permet d'affecter le droit d'accès avec le type (lecture, écriture) d'une information a un utilisateur après avoir vérifier si l'information existe puis

si l'utilisateur existe puis son accès avec le type s'il est pas vérifié l'information sera ajouter a l'utilisateur si non échec.

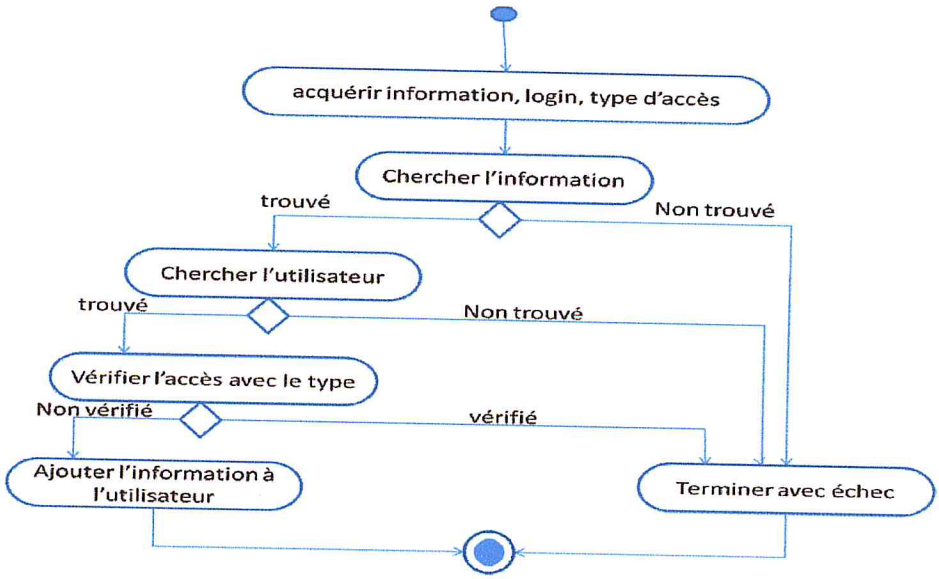


Figure 4.34 : *Diagramme d'activité « contrôler l'accès aux informations »*

➢ **Diagramme de classes :**

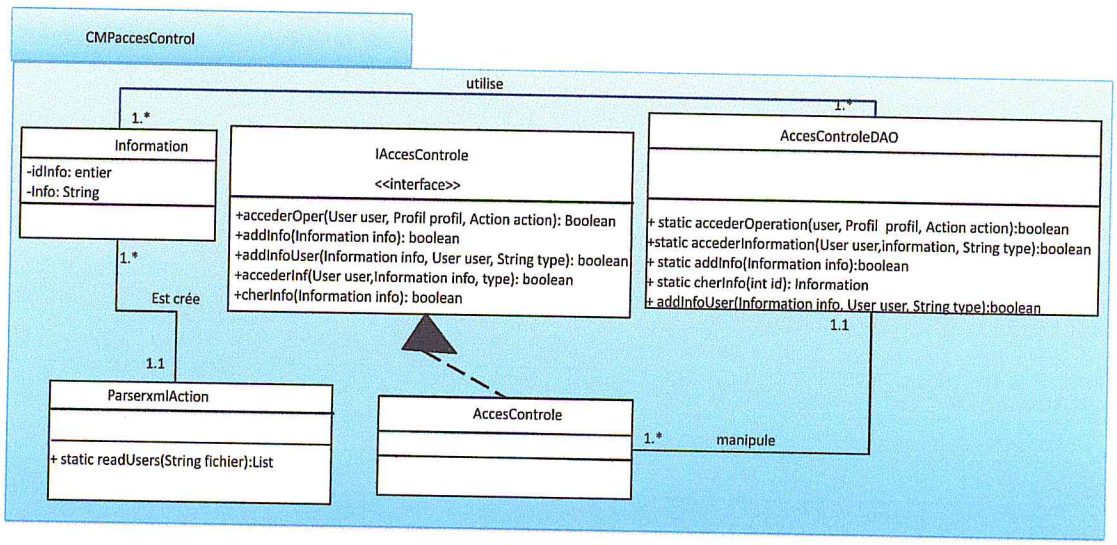


Figure 4.35 : *Diagramme de classes « contrôler l'accès aux informations »*

8) Le composant d'extension de sécurité :

Ce composant fournis des services de sécurité qui sont nécessaire pour le bon fonctionnement de n'importe quelle application, il fournie les interfaces suivantes :

A. Générer l'image du Captcha :

Cette interface permet de générer une image de Captcha en sauvegardant dans la base de données un identifiant qui correspond à cet image ainsi le code contenu dans cette dernière.

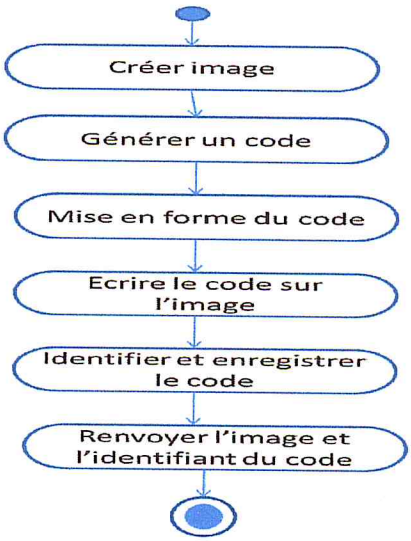


Figure 4.36 : *Diagramme d'activité « générer image Captcha »*

B. Valider un Captcha :

Cet interface permet de vérifier le captcha en vérifiant le code reçu avec l'identifiant, s'il lui correspond le captcha sera validé sinon le renvoie d'un message d'échec sera fait.

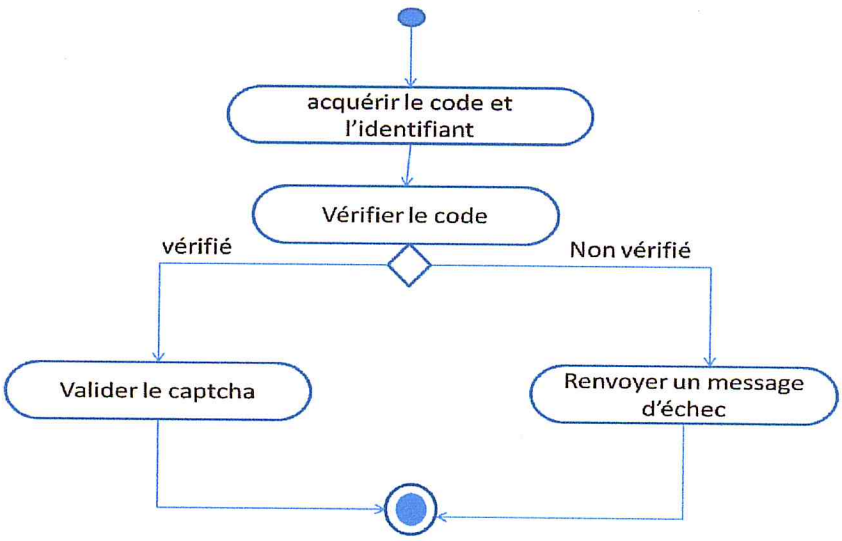


Figure 4.37 : *Diagramme d'activité « valider un Captcha »*

C. Crypter MD5 :

Cette interface permet de crypter n'importe quelle clé en utilisant la fonction de hachage MD5, comme par exemple crypter les mots de passes des utilisateurs.

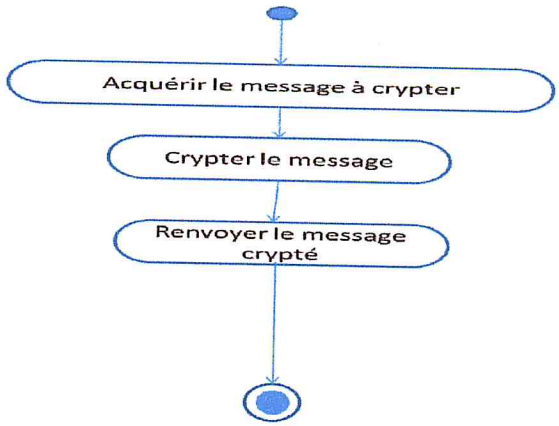


Figure 4.38: *Diagramme d'activité « crypter MD5 »*

D. Signer un document PDF:

Cette interface permet de signer un document PDF par un certificat numérique.

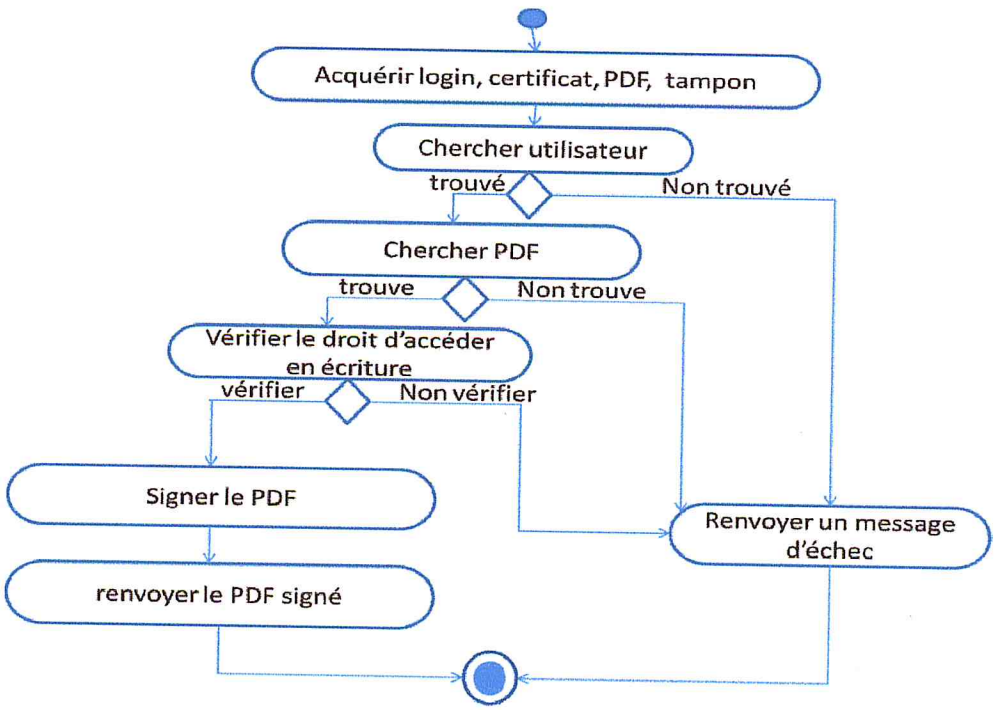


Figure 4.39 : *Diagramme d'activité « signer un document PDF »*

E. Valider un document PDF :

Cette interface permet de vérifier la validité d'un document PDF signé numériquement par un certificat à clé publique.

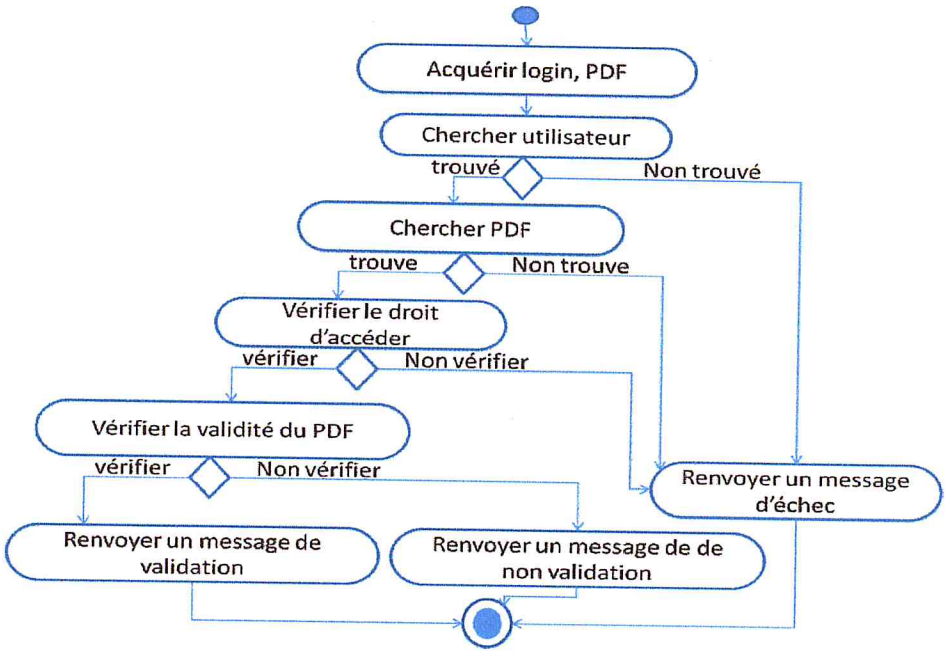


Figure 4.40: *Diagramme d'activité « valider un document PDF »*

➢ **Diagramme de classes :**

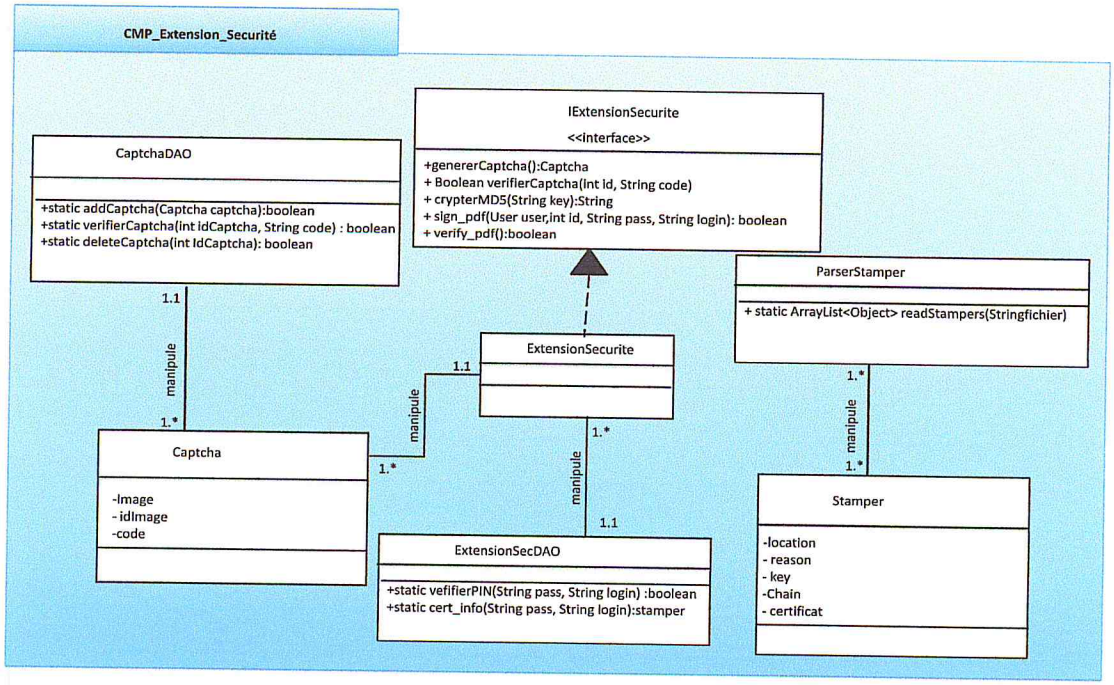


Figure 4.41 : *Diagramme de classes « extension de sécurité »*

VIII. Conclusion :

Dans ce présent chapitre nous avons présenté le diagramme du composant, les diagrammes d'activités de toutes les interfaces fournis par notre composant, et ainsi que les diagrammes de classes pour décrire la structure statique du composant. Dans le chapitre suivant nous présenterons l'implémentation du composant de sécurité ainsi la façon de l'intégrer dans une application.

Chapitre 5:

Réalisation et test

I. Introduction :

Dans notre projet nous avons conçus et réaliser un composant de sécurité fortement autonome.

Dans ce présent chapitre nous allons donner un bref aperçu sur l'application web que nous avons développés pour l'intégration et le test de notre composant, ainsi sa validation dans un langage de description d'architecture.

Pour l'implémentation de notre composant nous avons utilisé Java comme langage de programmation, et MySQL comme serveur de base de données.

II. Outils utilisés :

Pour la perfection de la sécurité du composant nous avons utilisé les outils suivants :

1) Certificats à clé publique :

Il est utilisé Pour l'authentification des utilisateurs et la signature des documents, il existe plusieurs types de certificats numériques, nous avons utilisé le type suivant :

X.509 est le format de certificats à clé publique le plus utilisé, il se base sur l'utilisation des autorités de certification. Un certificat X.509 est utilisé pour lier une clé publique à un individu ou à une entité, il est digitalement signé par l'émetteur du certificat (CA) qui assure le lien entre la clé publique et le détenteur (sujet) du certificat. **[CS03]**

Les fichiers de certificats X509 possèdent les extensions suivantes : .CER /.DER /.PEM/.P7B /.P7C /.PFX /.P12.

- La spécification .p12 fait partie des **.P12- PKCS#12,** nous avons choisis ce type de certificat car c'est le format communément utilisé pour stocker un certificat et sa clé privée associée dans un fichier protégé en confidentialité et intégrité par un mot de passe. Ce format est utilisé par Mozilla et Internet

Explorer/Outlook pour importer et exporter un certificat avec sa clé privée associée.

2) L'API iText:

iText est une API java libre / open source, elle a été développé pour permettre aux développeurs de faire ce qui suit:

- Servir des documents PDF à un navigateur.
- Manipuler des fichiers PDF.
- Ajouter une signature numérique à un fichier PDF.
- à l'aide de Bouncy Castle qui est une bibliothèque de cryptographie libre et open-source, Bouncy Castle n'est pas installé de base sur les plateformes java. **[SC12]**

3) OpenSSL :

OpenSSL est une boite à outils cryptographiques implémentant les protocoles SSL et TLS qui offre :

1. une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS.

2. une commande en ligne (openssl) permettant

   - ✓ la création de clés RSA, DSA (signature).
   - ✓ la création de certificats X509.
   - ✓ le calcul d'empreintes (MD5, SHA, RIPEMD160, ...)
   - ✓ le chiffrement et déchiffrement (DES, IDEA, RC2, RC4, Blowfish,..)
   - ✓ la réalisation de tests de clients et serveurs SSL/TLS.
   - ✓ la signature et le chiffrement de courriers (S/MIME)

4) Le Microsoft Management Console (souvent abrégé en MMC) :

Est une composante du system d'exploitation Windows qui fournit aux administrateurs de système une interface flexible à travers laquelle ils peuvent configurer leur système et en surveiller le fonctionnement.

A travers cette console on peut accéder au magasin de certificats de windows où se trouve l'autorité de certification racine de confiance et les certificats des personnels. C'est par cette console qu'on a rajouté le certificat du CA et les certificats client du test pour qu'elle soit valide.

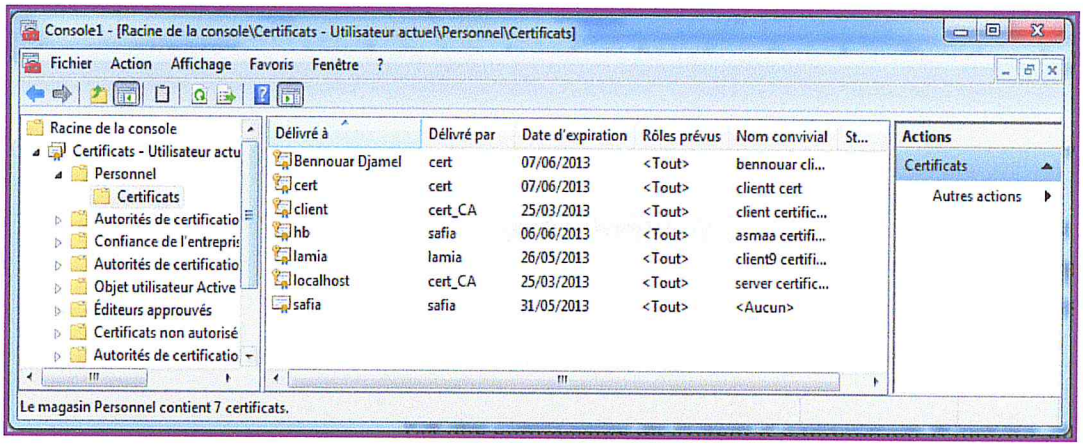


***Figure 5.1** : Accès au magasin des certificats du personnel*

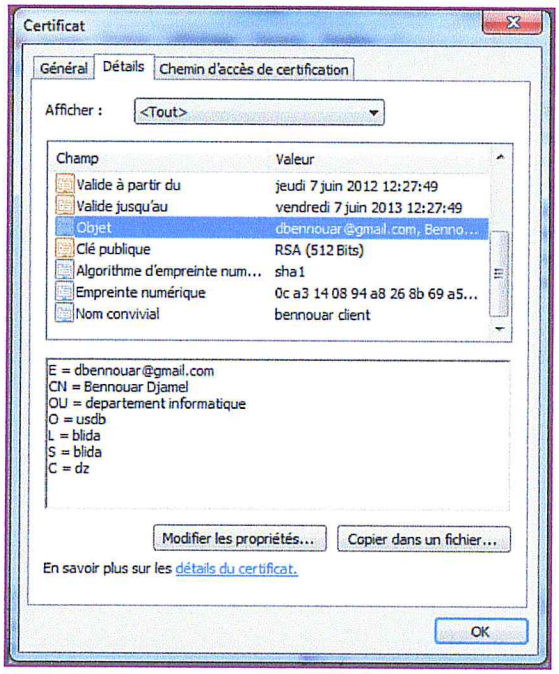


***Figure 5.2** : Détail du certificat « Bennouar Djamel .P12»*

5) L'API JDOM :

JDOM est une API open source Java dont le but est de représenter et manipuler un document XML de manière intuitive pour un développeur Java sans requérir une connaissance pointue de XML, donc but de JDOM

n'est pas de définir un nouveau type de parseur mais de faciliter la manipulation au sens large de document XML : lecture d'un document, représentation sous forme d'arborescence, manipulation de cet arbre, définition d'un nouveau document, ... etc.

Dans le rôle de manipulation sous forme d'arbre, JDOM possède moins de fonctionnalités que DOM mais en contre partie il offre une plus grande facilité pour répondre aux cas les plus classiques d'utilisation. **[NC12]**

III. Description de l'implémentation du composant :

Dans l'implémentation de notre composant (composite) nous avons choisis de l'organiser en packages, chaque package représente un sous composant (primitif) indépendant des autres.

Chaque composant primitif est constitué d'une interface qui représente le point d'accès aux services fournis par le composant, cette dernière est implémentée par une autre classe, c'est le principe de l'inversion de contrôle, L'idée principale est que les classe dépendre d'interfaces, et non les classes dépendre les uns des autres. Les classes dérivent des interfaces. L'inversion de contrôle les rend donc indépendants entre eux. L'utilisation d'une interface permet de s'abstraire de la source de données. Ainsi les composants ou les services, qui sont des composants distants, forment des entités autonomes et réutilisables sans avoir modifier l'ensemble des codes sources.

Nous avons utilisé aussi les notions des objets beans et DAO pour bien organiser le code et pour bien séparer l'accès à la base de données du traitement métier.

Dans un contexte générale, notre composant interagit avec le monde externe par les appelles de méthodes, ces méthodes fournies par le composant utilise parfois des fichiers XML comme paramètres pour faciliter les échanges avec l'externe.

Pour que les données contenues dans les fichiers XML soient exploitable par le composant nous avons développé un parseur XML en utilisant l'API JDOM

La figure suivante représente un exemple d'un fichier XML utilisé obligatoirement pour instancier le composant de sécurité:

```
<?xml version="1.0" encoding="UTF-8"?>
<connections>
        <DataBase>
                <name>E-commerce</name>
                <drivers>com.mysql.jdbc.Driver</drivers>
                <host>jdbc:mysql://127.0.0.1/</host>
                <user>root</user>
                <password>root</password>
        </DataBase>

        <admin>
                <login>amina</login>
                <pass>pass modifié</pass>
        </admin>
</connections>
```

Information sur de base de données

Administrateur du composant

***Figure5.3** : fichier XML « instance.xml »*

IV. Intégration du composant dans une application web :

Pour tester le bon fonctionnement du composant, nous avons développé une application web java E-commerce en suivant une architecture **MVC** (Modèle-Vue-Contrôleur), et comme conteneur de servlet nous avons utilisé **Apache Tomcat.**

Avant l'intégration de notre composant (comme jar), l'application fonctionne sans utiliser aucun aspect de sécurité, par exemple l'accès aux services est direct sans aucune demande d'authentification.

Après avoir instancier le composant de sécurité par l'application E-commerce avec des paramètres par défauts, l'application peut utiliser les services fournis par notre composant en faisant des appels méthodes aux interfaces.

1) Présentation de E-commerce après l'intégration du composant :

➢ **Page d'accueil :**

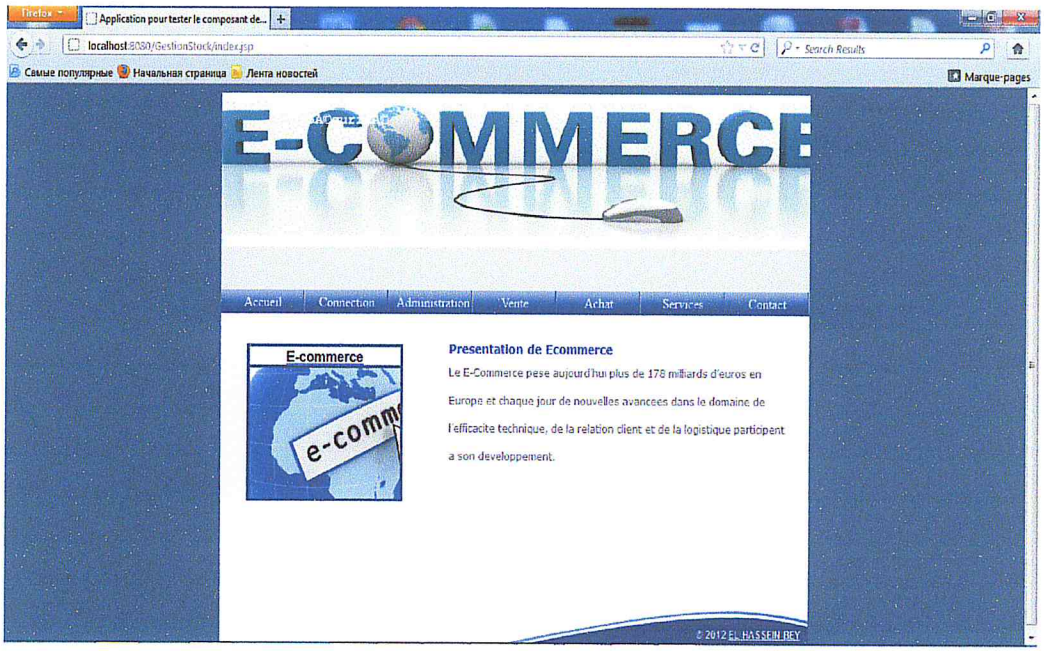
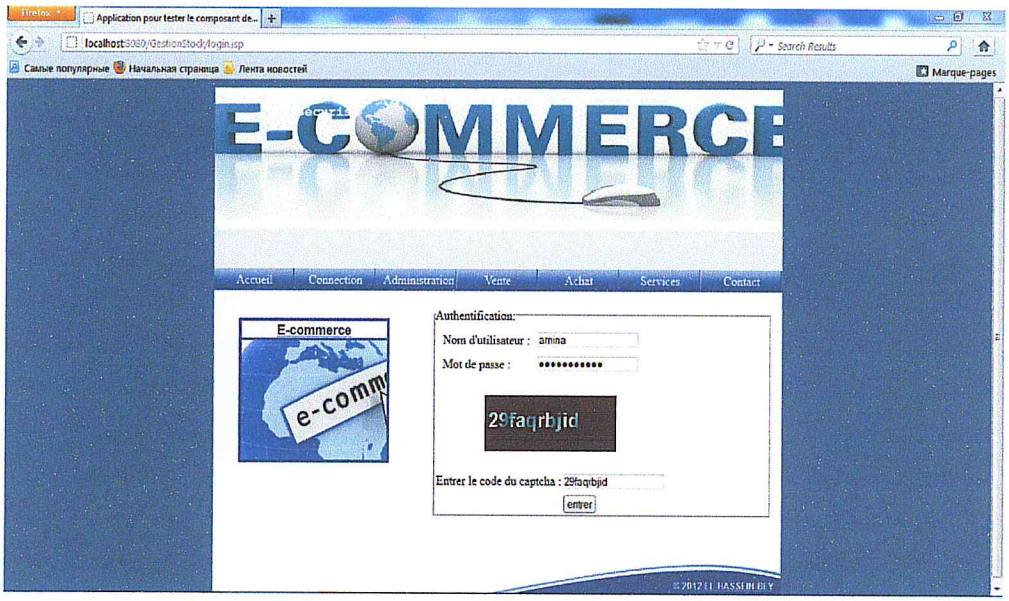


***Figure 5.4** : page d'accueil de E-commerce*

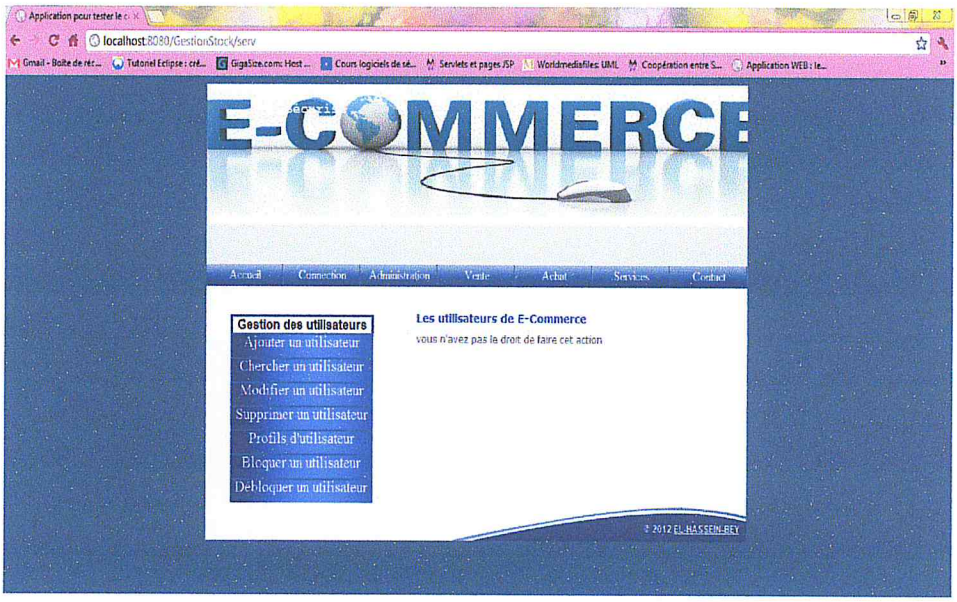
➢ **Authentification :**

Dans cette page, après que l'utilisateur remplis le formulaire, l'application fait un appel au composant pour vérifier que les informations entrés sont correctes, si c'est le cas une session s'ouvre.



***Figur5.5** : authentification des utilisateurs*

➢ **Accès aux services après l'authentification :**



***Figur5.6** : vérification des droits d'accès*

Cette page affiche le résultat retourné par le composant après la vérification des droits d'accès

➢ **Gestion des utilisateurs :**

***Figure 5.7** : gestion des utilisateurs de E-commerce*

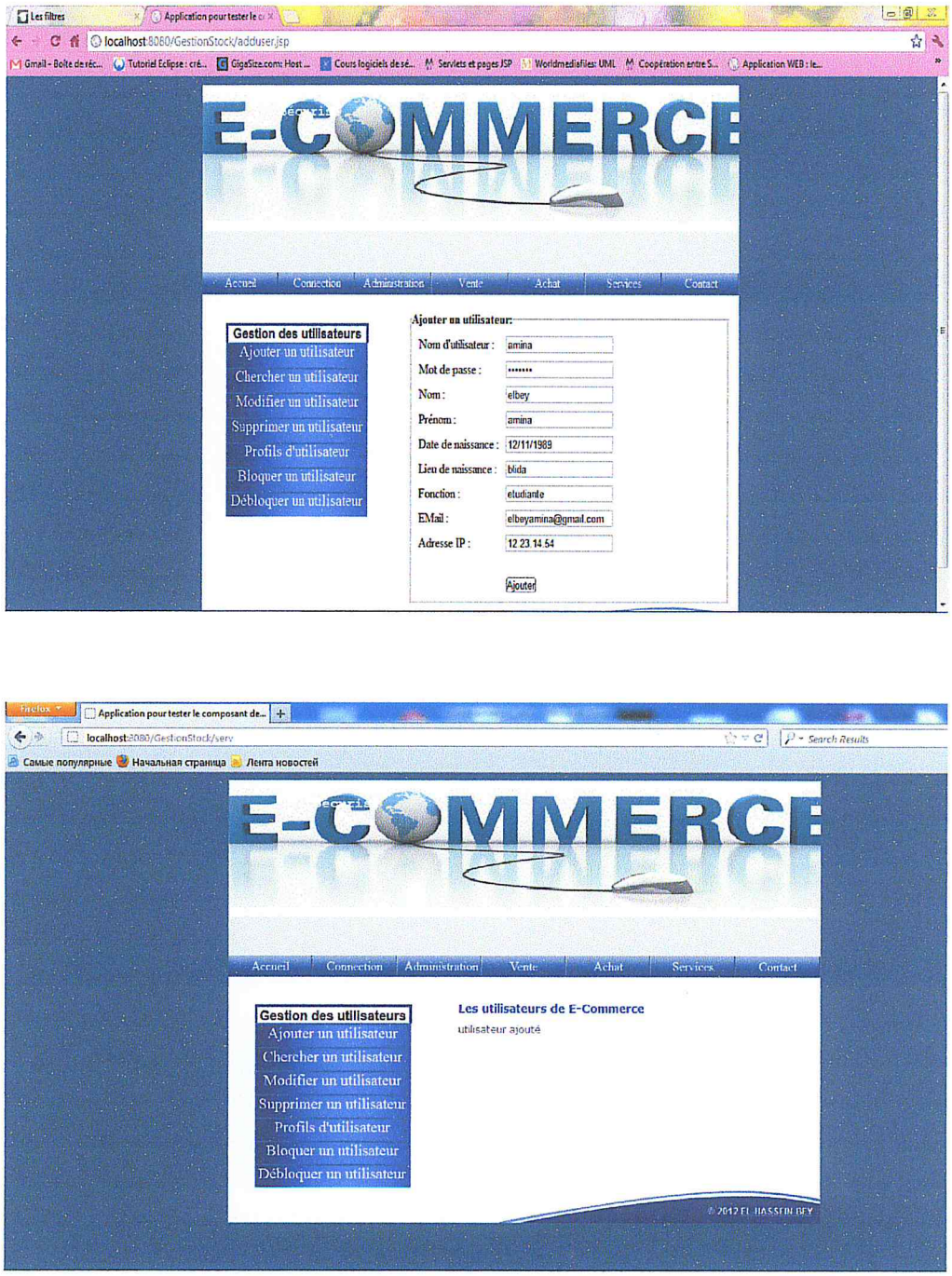


***Figure 5.8** : Ajout d'un utilisateur*

➢ **Signature d'un document PDF :**

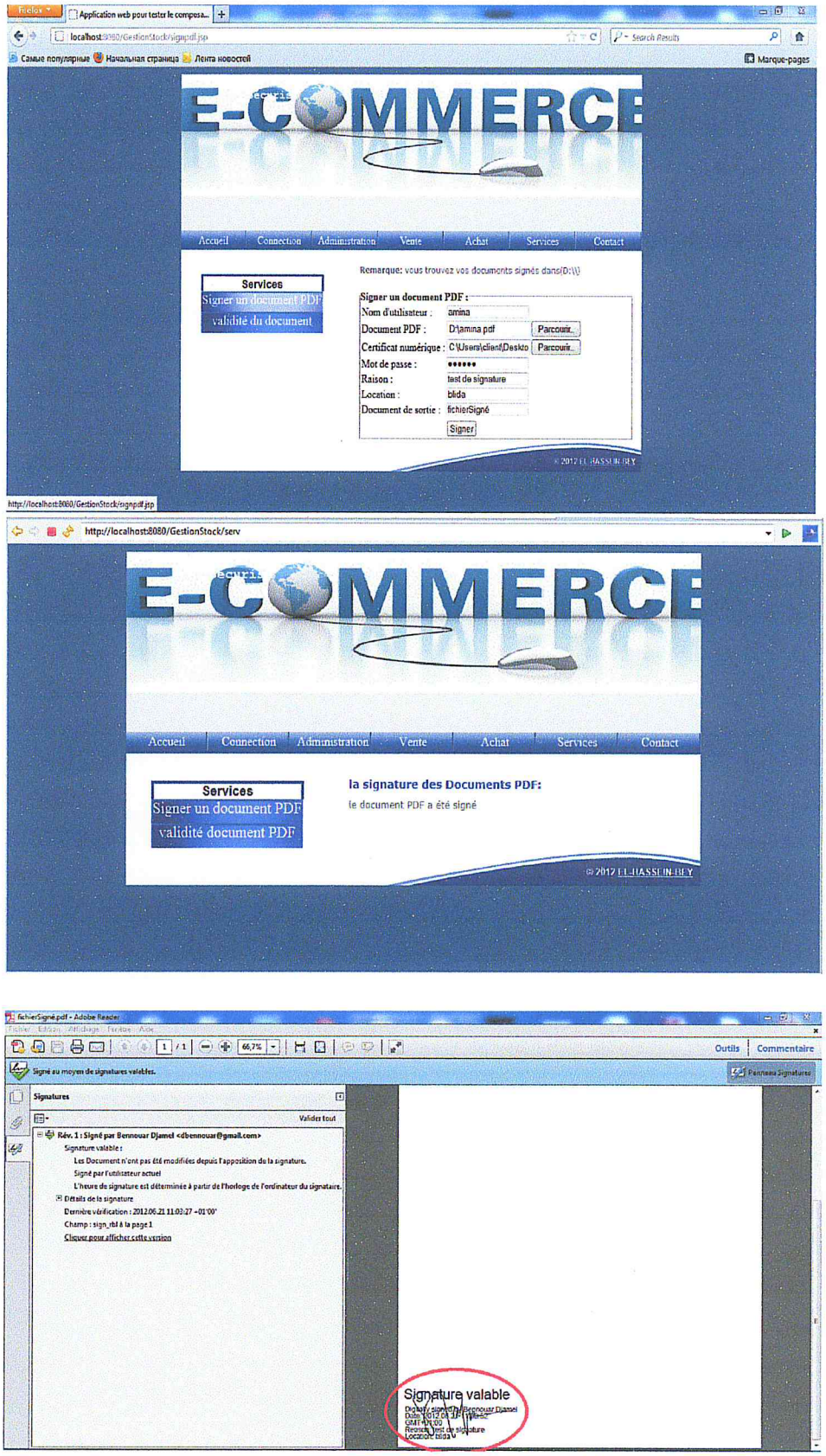


***Figure 5.9**: signature d'un document PDF*

➢ **Authentification d'un document PDF :**

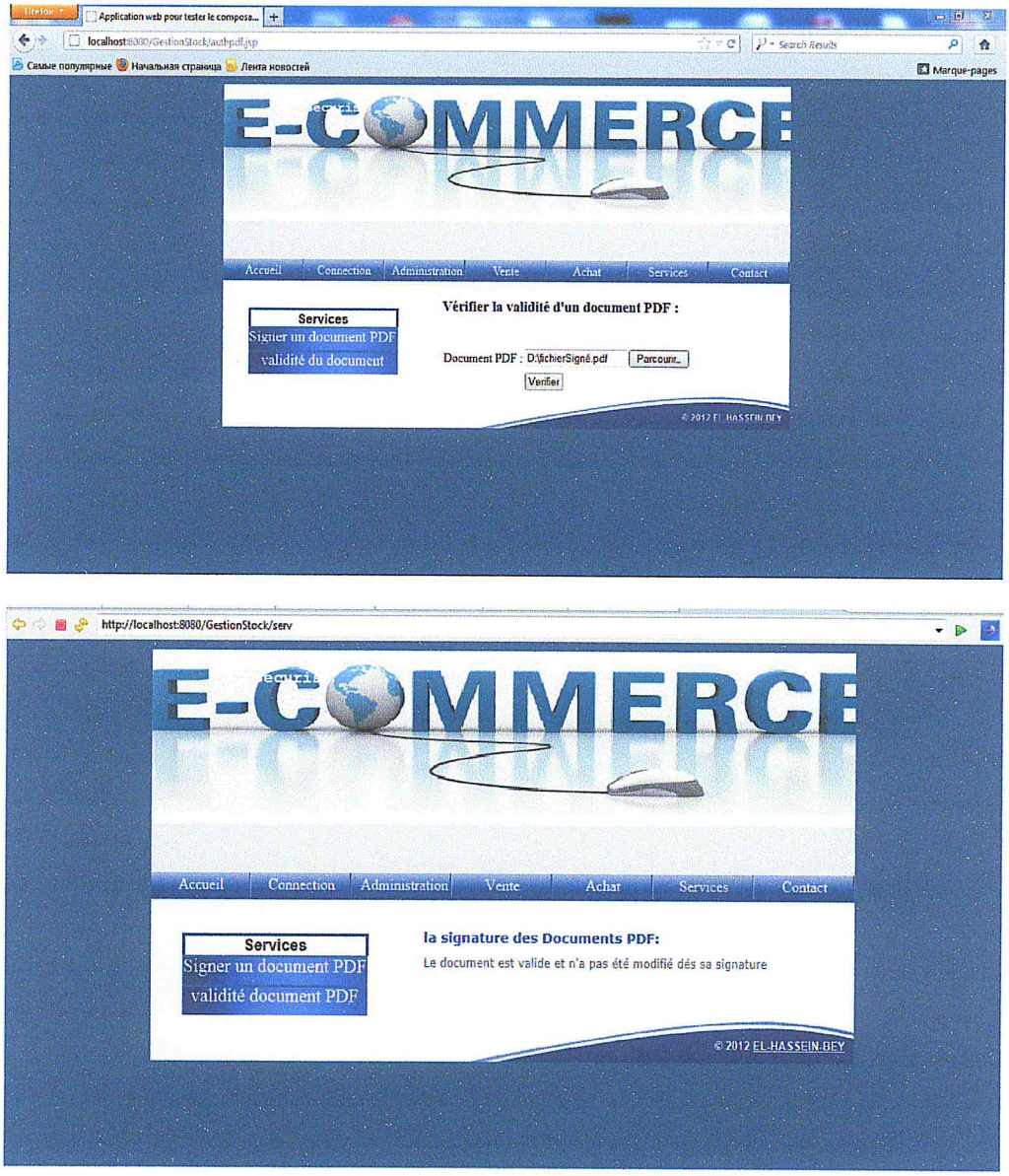


***Figure 5.10** : Authentification d'un document PDF*

V. Validation du composant dans un contexte ADL :

1) ADL Assembler4J :

Pour valider notre composant dans l'ADL Assebler4J, nous avons choisis deux sous composants : Authentification et Gestion des utilisateurs, car une fois testé avec succès nous pouvons généraliser sur le reste des sous composants.

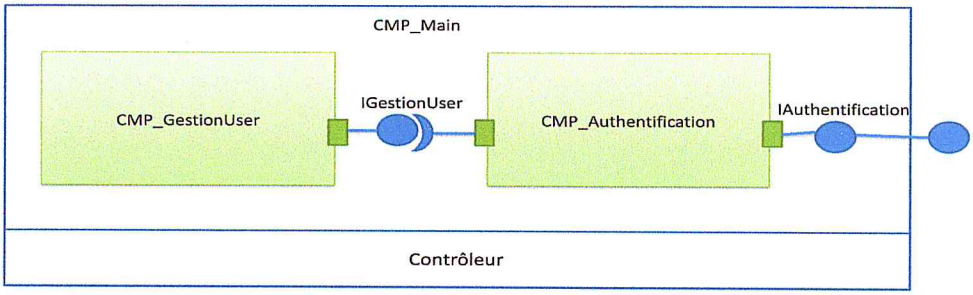


***Figure 5.11** : Exemple d'assemblage de composants*

Dans la représentation précédente le composant CMP_Main est Constitué de deux parties :

- Partie Contrôle : *Main*
- Partie métier : Constituée de deux composants primitifs Authentification et Gestion des utilisateurs.

Pour mettre en place le composant de test, on crée un fichier descripteur écrit en XML. Ce fichier décrit la vue interne du composite. Il s'agit d'une description détaillée du composant, elle est utilisée par ***l'Assembleur*** comme référence pour configurer dynamiquement l'architecture d'un composant lors de sa création.

```
<?xml version="1.0" encoding="UTF-8"?>

<tns:definition name="cmpMain" type="composite"
        xmlns:tns="http://www.example.org/definition"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.org/definition definition.xsd ">

        <interface name="iauth" role="server" signature="IAuthentification" />          } Delegation

        <component name="authentification" type="primitive">                           } Composant d'authentification
                <interface name="iauth" role="server" signature="IAuthentification" />
                <interface name="iuser" role="client" signature="IGesionUser" />
                <content class="cmpSec.authentification.Authentification" type=""/>
        </component>

        <component name="gestionUser" type="primitive">                                } Composant gestion des utilisateurs
                <interface name="iuser" role="server" signature="IGestionUser" />
                <content class="cmpSec.gestionUser.GestionUser" type=""/>
        </component>


        <binding client="cmpMain.iauth" server="authentification.iauth" />             } Binding
        <binding client="authentification.iuser" server="gestionUser.iuser" />
</tns:definition>
```

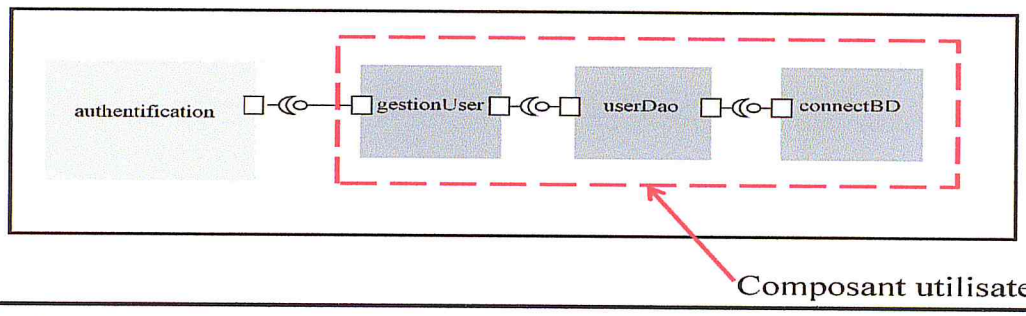
Figure 5.12 : Descripteur XML de composant CmpMain

2) ADL ArchJava :

Nous avons aussi validé notre composant dans le contexte de l'ADL ArchJava, le test concerne juste les deux sous composants utilisateurs et authentification pour pouvoir accomplir l'opération d'authentification d'un utilisateur. Pour mettre en place des composants archjava, nous avons suivis les étapes suivantes :

1) Il faut avoir une version du JDK 1.5 ou supérieur, télécharger ArchJava et faire la configuration adéquate.
2) Créer les sous composant **connect.archj**, **userDao.archj**, **gestionUser.archj** et **authentification.archj** puis les compiler en ligne de commandes et en résultat nous allons avoir des fichiers avec des extensions java.
3) En fin compiler et exécuter ces fichiers résultants.

Voici la représentation des composants testés :




```
public component class Main{
  private final connectBD connect= new connectBD ();
  private final gestionUser gu=new gestionUser();
  private final userDao a=new userDao();
  private final authentification auth=new authentification();

 connect a.in, connect.out;
 connect gu.in,a.out;
 connect gu.out,auth.in;

  public void run(){ auth.run("lamia"); }

  public static void main(String[] arg){new Main().run();
        }
}
```

Instanciation des composants

Définition de la connexion entre les composants

Définition de la fonction principale

***Figure 5.13** : Validation dans l'ADL ArchJava*

Ce fichier « **Main.archj** » permet de définir les connexions entre les différents composants et invoquer la fonction d'authentification.

VI. Conclusion :

Dans ce chapitre nous avons testé le composant de sécurité dans une application web ensuite dans un ADL, pour monter qu'il est indépendant du monde externe et qu'il peut être intégré dans n'importe quelle application java.

Conclusion générale

Nous avons présenté dans ce travail, la réalisation d'un composant de sécurité fortement autonome, indépendant de toute application et facilement réutilisable. Le composant conçu et réaliser est conforme au modèle de composant utilisé en Architecture Logicielle.

Ainsi, ce composant expose des interfaces qui indiquent les services fournis par le composant et les services requis, Nous avons utilisé un modèle de composant qui s'inspire des modèles des deux approches IASA et FRACTAL.

Notre composant généralisé peut être mis dans n'importe quel contexte d'ADL d'où nous avons montré sa validité avec l'ADL Assembler4J et ArchJava.

Ce projet nous a permis d'améliorer nos connaissances en matière d'architecture logicielle, de développement d'applications web avec Java et de manipulation des certificats numériques ; il nous a permis aussi d'acquérir une certaine expérience dans le domaine de l'orienté aspect. Nous espérons que notre travail aura été à la hauteur de vos espérances.

Perspective :

Pour manque de temps nous avons fait une authentification par certificat d'où on peut étendre cet authentification à une authentification par carte à puce c'est à dire de conserve le certificat dans une carte à puce et ça donnera plus de sécurité et une meilleur portabilité.

Un autre perspectif est la détermination d'une technique de connexion des composants basée sur les concepts de la programmation orientée aspect.

Annexe

I. Le modèle à composant IASA :

Un composant IASA (Integrated Approach to Software Architecture) est vu de l'extérieur comme une boite noire qui communique avec le monde externe grâce à des ***Ports***, qui définissent les services qu'il peut offrir ou requérir. La vue interne d'un composant primitif est inaccessible, tandis que celle d'un composant composite est bien définie, elle consiste en trois parties : *Partie Opérationnelle*, Partie *Aspect et Partie Contrôle.*

- **Partie Contrôle** : a pour rôle de charger et configurer les composants internes et satisfaire la demande des composants externes, en délégant ces demandes à un composant interne.

- **Partie Aspect** : C'est les fonctionnalités techniques, qui ne font par partie de la logique du métier de l'application.
- **Partie Opérationnelle** : Regroupe les composants métier (Opérations principales pour le fonctionnement de l'application). **[BM11]**

II. Le protocole SSL

Le protocole *Secure Sockets Layer* (SSL) est un ensemble de règles gouvernant l'authentification serveur, l'authentification client et les communications encryptées entre des serveurs et des clients. SSL est largement utilisé sur Internet, particulièrement pour les interactions mettant en œuvre l'échange d'informations confidentielles telles que les numéros de cartes de crédit.

SSL requiert un certificat SSL serveur, au minimum. Comme partie du processus de négociation, le serveur présente son certificat au client afin d'authentifier son identité. Le processus d'authentification utilise le chiffrement par clef publique et les signatures numériques pour confirmer que ce serveur est bien celui-ci qu'il prétend être. Une fois le serveur authentifié,

le client et le serveur utilisent des techniques de chiffrement à clefs symétriques, ce qui est rapide, pour chiffrer toutes les informations qu'ils échangent pour le reste de la session et pour détecter toutes tentatives d'altération des données qui peuvent arriver.

Les serveurs peuvent éventuellement être configurés pour demander l'authentification client aussi bien que l'authentification serveur. Dans ce cas, après le succès de l'authentification serveur, le client doit à son tour présenter son certificat au serveur afin d'authentifier son identité avant qu'une connexion SSL ne puisse s'établir.

III. Types de certificats

Cinq types de certificats sont couramment utilisés:

a) Certificats de client SSL :

Utilisés pour identifier des clients auprès de serveurs via SSL (authentification client). Généralement, l'identité du client est présumée être la même que celle d'un être humain, tel qu'un employé dans une entreprise. Les certificats d'un client SSL peuvent également être utilisés pour la signature de formulaires et comme composante d'une solution de l'ouverture de session unique.

Exemples

Une banque donne un certificat client SSL à l'un de ses usagers qui lui permet de s'identifier auprès du serveur de la banque et d'accéder à ses comptes. Une compagnie peut donner un certificat client SSL à l'un de ses nouveaux employés qui lui permet de s'identifier auprès du serveur de l'entreprise et d'obtenir accès aux ressources disponibles sur ce serveur.

b) Certificats de serveur SSL :

Utilisé pour identifier les serveurs auprès des clients via SSL (authentification serveur). L'authentification serveur peut être utilisée avec ou sans authentification client. L'authentification serveur est obligatoire lors de l'établissement d'une connexion SSL chiffrée. Pour plus d'informations.

Exemple

Les sites internet de commerce électronique (communément appelé e-commerce) supportent habituellement l'authentification serveur par certificat, au minimum, pour établir une session SSL chiffrée et assure les clients qu'ils traitent avec un site identifié comme étant celui d'une entreprise donnée. La session SSL assure que les informations personnelles renseignées par le client et transmises par le réseau, telles que son numéro de carte de crédit, ne seront pas aisément interceptées.

c) Certificats S/MIME :

Utilisés pour signer et chiffrer les courriels. Comme pour les certificats client SSL, l'identité du client est généralement présumé être la même que celle d'un être humain, tel qu'un employé d'une entreprise. Un certificat unique peut être utilisé comme certificat S/MIME et comme certificat SSL. Les certificats S/MIME peuvent également être utilisés pour la signature de formulaires et comme composante d'une solution de l'ouverture de session unique.

Exemples

Une entreprise déploie des certificats combinés S/MIME et SSL dans l'unique but d'authentifier l'identité des employés, permettant ainsi la signature de messages et l'authentification de client SSL, mais pas le chiffrage des messages. Une autre entreprise émet des certificats S/MIME uniquement dans le but de signer et de chiffrer les messages de natures financière ou légale qu'elle envoie.

d) Certificats de signature d'objet :

Utilisés pour identifier les signataires de code Java, de scripts JavaScript, ou d'autres fichiers signés. Pour plus d'informations.

Exemple

Une entreprise de logiciel signe les logiciels qu'elle distribue par Internet pour fournir l'assurance à ses utilisateurs que le logiciel est un produit

légitime. L'utilisation des certificats et des signatures numériques de cette façon peut également servir pour que les utilisateurs identifient et contrôlent l'accès à leurs ordinateurs des logiciels téléchargés.

e) Certificats d'AC :

Utilisés pour identifier les autorités de certification (AC). Les logiciels client et serveur utilisent les certificats d'AC pour déterminer quelles autres certifications peuvent être de confiance. Pour plus d'informations.

Exemple

Les certificats d'AC stockés dans *Communicator* déterminent quels autres certificats peuvent être utilisés pour l'authentification. Un administrateur peut implémenter certains aspects de la politique de sécurité de son entreprise en contrôlant les certificats d'AC stockés dans les *Communicator* de chaque utilisateur.

IV. Vérification d'une chaîne de certificat

La vérification de la chaîne de certificats est le processus permettant de s'assurer que la chaîne de certificats donnée est bien formée, proprement signée et sécurisée. Les logiciels Red Hat utilisent la procédure suivante pour former et vérifier une chaîne de certificats, en commençant par le certificat présenté pour l'authentification :

1. La période de validité du certificat est vérifiée par rapport à la date actuelle fournie par l'horloge système du vérificateur.
2. Le certificat de l'émetteur est localisé. La source peut être une base de certificats locale du vérificateur (du client ou du serveur) ou une chaîne de certificats fournit par le sujet (par exemple, par une connexion SSL).
3. La signature du certificat est vérifiée à l'aide de la clef publique du certificat de l'émetteur.
4. Si le certificat de l'émetteur est présent dans les certificats de confiance du vérificateur, la vérification s'arrête avec succès à cette étape. Autrement, le certificat de l'émetteur est vérifié pour être certain qu'il contient les indications appropriées

concernant les AC subordonnées dans l'extension du type de certificat de Red Hat, et la chaîne de vérification recommence depuis l'étape 1, mais avec le nouveau certificat. La figure 8 présente un exemple de ce processus.

Glossaire

SSH	*Secure SHell*. C'est un protocole qui permet de faire des connexions sécurisées
RSA	Le RSA est l'abréviation de Rivest, Shamir et Adleman ces inventeurs en 1978. C'est l'exemple le plus courant de cryptographie asymétrique, toujours considéré comme sur, avec la technologie actuelle, pour des clés suffisament grosses (1024, 2048 voire 4096 bits).
URL	En anglais Uniform Ressource et en français localisateur uniforme de ressource
SQL	Structured Query Language est un langage informatique servant à effectuer des opérations sur des bases de données.
AC	Autorité de certification ou en anglais CA certification authority
CPU	Central processing unit
MD5	Message Digest 5
SHA-1	Secure Hash Algorithm
SSL	Secure Sockets Layer
LDAP	Lightweight Directory Access Protocol, en français Protocole d'accès aux annuaires léger
SSO	Authentification unique en anglais Single Sign-On
USB	Universal Serial Bus en français Bus universel en série
IASA	(Integrated Approach to Software Architecture) est vu de l'extérieur comme une boite noire qui communique avec le monde externe grâce à des ***Ports***, qui définissent les services qu'il peut offrir ou requérir.
UML	Unified Modeling Language ou en français langage de modélisation unifié
IHM	interaction homme-machine
PDF	Portable Document Format
INRIA	Institut national de recherche en informatique et en automatique
France Telecom R&D	Le centre de recherche de la société France Télécom. Il était précédemment appelé CNET ou Centre national d'études des télécommunications.
ADL	Langage de description d'architecture
XML	Extensible Markup Language en français langage de balisage extensible
.J2EE	Java Enterprise Edition, ou Java EE (anciennement J2EE)
TLS	Transport Layer Security
DAO	Data access object
.P12-PKCS#12	Public Key Cryptography Standards qui ont été développés et publiés par RSA, c'est un standard de syntaxe d'information personnelle, Définit un format de fichier généralement utilisé pour stocker la clé privée et le certificat de clé publique correspondant en les protégeant par un mot de passe .

Bibliographie

[AA03] Anas Abou El Kalam, thèse de doctorat « Modèles et politiques de sécurité pour les domaines de la santé et des affaires sociales », Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique, 2003.

[BCS02] E. Bruneton, T. Coupaye, et J.B. Stefani, "*The Fractal Composition Framework Version 1.0.*", Object Web Consortium, Juillet 2002.

[BM11] Abdelhakim BAOUYA et Mohamed MAHDJOUB, « Conception Orientée Aspect et par composants d'une application d'E-Gouvernement », thèse de master, 2011.

[CS03] Carine Saab, « Conception d'objet XML pour la mise en oeuvre des services de sécurité comme alternative aux certificats », diplôme d'études approfondies, université libanaise et université saint Joseph ,2003.

[DT12] Division assistance Technique, « Recommandations de sécurité relatives aux mots de passe », http://www.certa.ssi.gouv.fr/site/CERTA-2005-INF-001/, 2012

[DS02] Le déni de service (DoS), http://www.securiteinfo.com/ attaques/hacking /dos.shtml , 2002

[DE10] David Endler, "Brute-Force Exploitation of Web Application Session IDs"http://www.securityfocus.com/advisories/7143, 2010.

[DB09] Djamal Bennouar, « Une approche intégrée pour L'architecture logicielle », thèse de doctorat, L'école nationale supérieur d'informatique d'alger, 2009.

[HC04] Humberto Cervantes, « Vers un modèle à composant orienté service pour supporter la disponibilité dynamique», thèse de doctorat, Université Joseph Fourier - Grenoble, 2004.

[IA10] Imen Amouri Loulou, « P/S-COM : une méthode formelle d'aide à la conception correcte des architectures logicielles Publier/Souscrire », thèse de doctorat, 2010.

[LC05] Laurent Bloch et Christophe Wolfhugel, « Sécurité informatique principes et méthodes », ÉDITIONS EYROLLES61, 2005

[MH08] Mehdi Hariati, « Les Composants Logiciels et La SéparationAvancée des Préoccupations : Vers une nouvelle Approche de Combinaison », thèse de doctorat, Laboratoire LRI, Université Badji Mokhtar-Annaba BP 12, 23000 Annaba ALGERIE, 2008

[MN97] T. Meijler et O. Nierstrasz. Beyond, "Components. Dans *Cooperative Information Systems : Current Trends and Directions*", Academic Press, 1997.

[NT11] Nteray, « certificat et authentification », https://developer.mozilla.org/index.php?title=fr/Introduction_%C3%A0_la_cryptographie_%C3%A0_clef_publique/Certificats_et_authentification, 2011.

[NC12] Nicolas CYNOBER, « Manipuler des données XML avec java et JDOM» , http://cynober .developpez.com /tutoriel/java/xml/jdom/, 2012

[OB05] Olivier Barais, « Construire et Maitriser l'Evolution d'une Architecture Logicielle µa base de Composants », Thèse de doctorat, l'université des Sciences et Technologies de Lille, 2005

[SE04] « Signature electronique », http://www.javarome.free.fr/net/Signature%20électronique%20-%20Javarome.htm , 2004

[Szy98] C. Szyperski, "*Component software: beyond object-oriented programming*", ACM Press/Addison-Wesley Publishing , 1998.

[SG96] Shaw M. and Garlan D, "Software Architecture- Perspectives on an Emerging Discipline", Prentice Hall. ISBN: 0-13-182957-2,1996.

[SC12] Sarl Clevacti, PKCS, http://www.techno-science.net /? Onglet =glossaire &definition=10986, 2012

[TE06] Tewfiq.ElMaliki, Professeur au Laboratoire de télécommunications, Ecole d'Ingénieurs de Genève, « Plate-forme sécurisée d'un réseau de cartes d'identités virtuelles à l'aide de carte à puce », 2006, page 38.

[YN09] Yann Arthur Nicolas, Spring Garden, www. developpez.com, 2009.