IdMe : un socle pour la gestion d'identités en entreprise

Frantz Fischer

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

PARIS

MÉMOIRE
présenté en vue d'obtenir
le DIPLÔME d'INGÉNIEUR CNAM
SPÉCIALITÉ : Informatique

OPTION : Architecture et ingénierie des systèmes et des logiciels

par

FISCHER Frantz

IdMe – Un socle pour la gestion d'identités en entreprise

Soutenu le 27 juin 2011

JURY

PRÉSIDENT : POLLET Yann
MEMBRES : BARRÉ Sylvain
DOUIN Jean-Michel
LIGNELET Patrice
SCHÖNEBERGER Olivier

Remerciements

Je remercie Nicolas Pioch pour m'avoir montré la voie et m'avoir aiguillé vers le CNAM.

Mes remerciements vont également à Jean-Louis Dewez, Pascal Graffion, et Marc Le Bihan, pour m'avoir encouragé à développer ce sujet de mémoire.

Merci à mes tuteurs Jean-Michel Douin et Olivier Schöneberger sans qui je n'aurai pu réaliser ce projet.

Je remercie donc le jury dans son ensemble pour le temps passé sur ce mémoire, et notamment Sylvain Barré qui n'a pas hésité à parcourir une grande distance afin d'être présent.

Enfin, je remercie particulièrement ma femme Anastasia qui a su me remettre dans le droit chemin lorsque c'était nécessaire et m'encourager sans relâche tout au long de mon parcours CNAM.

Abréviations

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CNIL	Commission Nationale de l'Informatique et des Libertés
CRM	Customer Relationship Management
CSS	Cascading Style Sheet
DAC	Discretionary Access Control
DAO	Data Access Object
EJB	Enterprise Java Bean
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management
IAS	International Accounting Standards
IDE	Integrated Development Environment
IDP	Identity Provider
IP	Internet Protocol
IHM	Interface Homme Machine
LSF	Loi sur la Sécurité Financière
JAAS	Java Authentication and Authorization Service
JDBC	Java Database Connectivity
JDK	Java Development Kit
JEE	Java Enterprise Edition
JPA	Java Persistence API
JPQL	Java Persistence Query Language
JSF	Java Server Faces
JSP	Java Server
JTA	Java Transaction API
LCEN	Loi pour la Confiance dans l'Economie Numérique
LDAP	Lightweight Data Access Protocol
LOLF	Loi Organique des Lois de Finance
MCO	Maintien en Condition Opérationnelle
MOA	Maîtrise d'ouvrage
MOE	Maîtrise d'œuvre

MVC	Model View Controller
NOS	Network Operating Systems
ORM	Object Relational Mapping
OS	Operating System
PC	Personal Computer
PDA	Personal Digital Assistant
PKI	Public Key Infrastructure
POC	Proof Of Concept
POJO	Plain Old Java Object
RBAC	Role Based Access Control
SAML	Security Assertion Markup Language
SEC	Securities and Exchange Commission
SI	Système d'Information
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SP	Service Provider
SQL	Structured Query Language
SSL	Secure Socket Layer
SSO	Single Sign On
TCP	Transfer Control Protocol
UC	Use Case
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Location
USB	Universal Serial Bus
WSDL	Web Service Description Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSLT	Extensible Style sheet Language Transformations

Sommaire

1. Introduction

Oracle a jusqu'à présent proposé des solutions issues d'agrégation de produits via l'acquisition de sociétés tierces. N'ayant pas développé de solution de gestion d'identité à partir de zéro, Oracle a donc décidé de développer un produit sur une durée de 6 à 9 mois en ayant une seule personne à disposition, afin d'évaluer la charge globale de travail nécessaire à la réalisation d'un tel outil.

Cet outil devait comporter des fonctionnalités basiques de gestion des utilisateurs, de leurs droits, et de leur cycle de vie. Il devait permettre l'administration et l'alimentation aval de ces mêmes droits sur des plateformes telles que des annuaires, des bases relationnelles, et des fichiers plats. De plus, une synchronisation d'informations utilisateurs entre 2 plateformes distantes devait être possible mais rester basique. D'autres fonctionnalités mineures pouvaient être ultérieurement précisées dans l'expression des besoins en fonction de l'évaluation de la charge de travail.

Ce mémoire présentera donc dans un premier temps la question de la gestion des identités en entreprise, son but, son intérêt, et ses concepts.

Enfin, la réalisation du projet sera abordée via ses différentes étapes en respectant l'ordre chronologique, à savoir : l'expression des besoins, l'analyse, la conception, l'architecture et les technologies utilisées, et enfin la réalisation.

2. Introduction à la gestion d'identité

La gestion des identités et des droits d'accès (souvent connu sous l'acronyme IAM, pour Identity and Access Management) est un maillon clé dans la chaîne de sécurité des organisations. Elle permet de renforcer le niveau de sécurité général en garantissant la cohérence dans l'attribution des droits d'accès aux ressources hétérogènes du système d'information.

La gestion des identités et des droits d'accès est également devenue l'un des moyens majeurs permettant de répondre aux exigences réglementaires de plus en plus fréquentes concernant la traçabilité. C'est aussi un moyen d'optimiser l'administration des droits.

Mais qu'entend-on exactement par gestion des identités et des droits d'accès ?

Selon www.clusif.asso.fr, la gestion des identités consiste à gérer le cycle de vie des

personnes (embauche, promotion, mutation, départ, etc.) au sein de la société et les impacts induits sur le système d'information (création de Comptes utilisateurs, attribution de Profils utilisateurs, mise en œuvre du contrôle d'accès, etc.). Cette gestion des identités doit pouvoir être faite d'un point de vue fonctionnel par des non informaticiens (exemple : Ressources Humaines, Maîtrise d'ouvrage, l'utilisateur lui-même) et d'un point de vue technique par des informaticiens (exemple : administrateur, Maîtrise d'œuvre). La solution de gestion d'identités doit être une solution globale sur la base d'une infrastructure centralisée avec une gestion fonctionnelle distribuée et qui intègre les fonctionnalités suivantes :

- la gestion du référentiel central des utilisateurs (alimentation à partir de référentiels utilisateurs sources)
- la gestion du référentiel central des ressources concernées par la gestion des droits d'accès
- la gestion des habilitations (gestion des Profils, Rôles, gestion des utilisateurs, workflow ou processus métier)
- le provisioning (synchronisation des référentiels cibles de sécurité)
- l'administration décentralisée
- l'auto administration, gestion par les utilisateurs des mots de passe et des données privées
- l'audit et le reporting (gestion des rapports)
- le contrôle d'accès (authentification, autorisation), voir [3]

Ce document vous permettra de vous familiariser avec les concepts et la modélisation utilisés dans une démarche de gestion des identités. Il a également pour but de vous aider à mener à bien un projet de gestion des identités et vous guider dans vos choix d'architecture.

2.1. Etat des lieux

Aujourd'hui, la multiplication et la diversité de systèmes de contrôle d'accès liés aux systèmes d'exploitation et aux applications est devenue particulièrement contre-productive. En effet, chaque « système » (OS, NOS, messagerie, groupware, applications métiers, ERP, CRM, etc.) est protégé par une procédure de contrôle d'accès spécifique. De fait, chaque fonction à réaliser peut nécessiter un code d'accès et des droits associés. Cette multiplicité de contrôles d'accès est source de confusion pour l'utilisateur qui, par exemple, perd ou oublie ses mots de passe. Il lui reste alors deux solutions : soit il sollicite le service de helpdesk, au risque de l'engorger en lui faisant perdre trop de temps à réinitialiser très souvent des mots de

passe, soit il note lesdits codes sur un support à sa portée pour ne plus les oublier ! Bien entendu, aucune de ces solutions n'est satisfaisante...

D'une manière générale, chaque système d'exploitation et/ou application est géré par un administrateur unique; de fait, toute vision globale est impossible. Dans ce cas, l'administration autonome de chaque système est particulièrement source d'erreurs, de vulnérabilités et de perte de temps. Par exemple, ne pas avoir la vision globale sur les droits de l'ensemble des utilisateurs peut engendrer des problèmes de responsabilité (devenus importants dans le cadre des nouvelles lois ou réglementations), tel qu'un acheteur qui validerait sa propre commande.

Généralement, les nouveaux arrivants se voient attribuer plus ou moins rapidement certaines ressources (un bureau, un téléphone, un badge d'accès, un PC, etc.), mais ne peuvent pas travailler, faute de droits d'accès. Ceci est notamment dû au fait que le délai d'attribution des ressources et des droits est trop long, que les circuits d'attribution sont trop « lourds », etc. Ils doivent même souvent se débrouiller seuls : trouver le bon responsable pour l'accès à telle base de données, à telle application, à l'Intranet, etc. La liste des interlocuteurs est à la mesure de la complexité du Système d'Information.

D'autre part, on est rarement certain d'avoir supprimé tous les droits dont disposait un employé partant ou d'avoir mis à jour les droits d'un employé en cas de changement de fonction. Le Système d'Information regorge souvent de comptes dits « fantômes » (dormants et/ou périmés).

De plus, les comptes techniques génériques, installés par défaut, par les systèmes d'exploitation et/ou les applications, ne sont pas toujours modifiés, voire supprimés, induisant d'autres failles. Ceci étant d'autant plus grave que ces mots de passe sont facilement accessibles sur Internet... De même, certaines personnes utilisent, lorsqu'elles arrivent dans l'entreprise, des comptes utilisateurs « génériques » et partagés, simplement du fait de la durée importante de création de leur propre compte.

Enfin, l'audit et la traçabilité sont souvent les parents pauvres de la mise en œuvre des droits d'accès des utilisateurs. Pourtant, de plus en plus, les entreprises doivent respecter des normes, des lois et/ou des réglementations strictes en matière de politique de contrôle interne.

2.1.1. Illustration

Si nous prenons l'exemple du référencement d'un nouvel utilisateur dans un Système d'Information, nous pouvons identifier les actions suivantes :

- embauche dans une entreprise

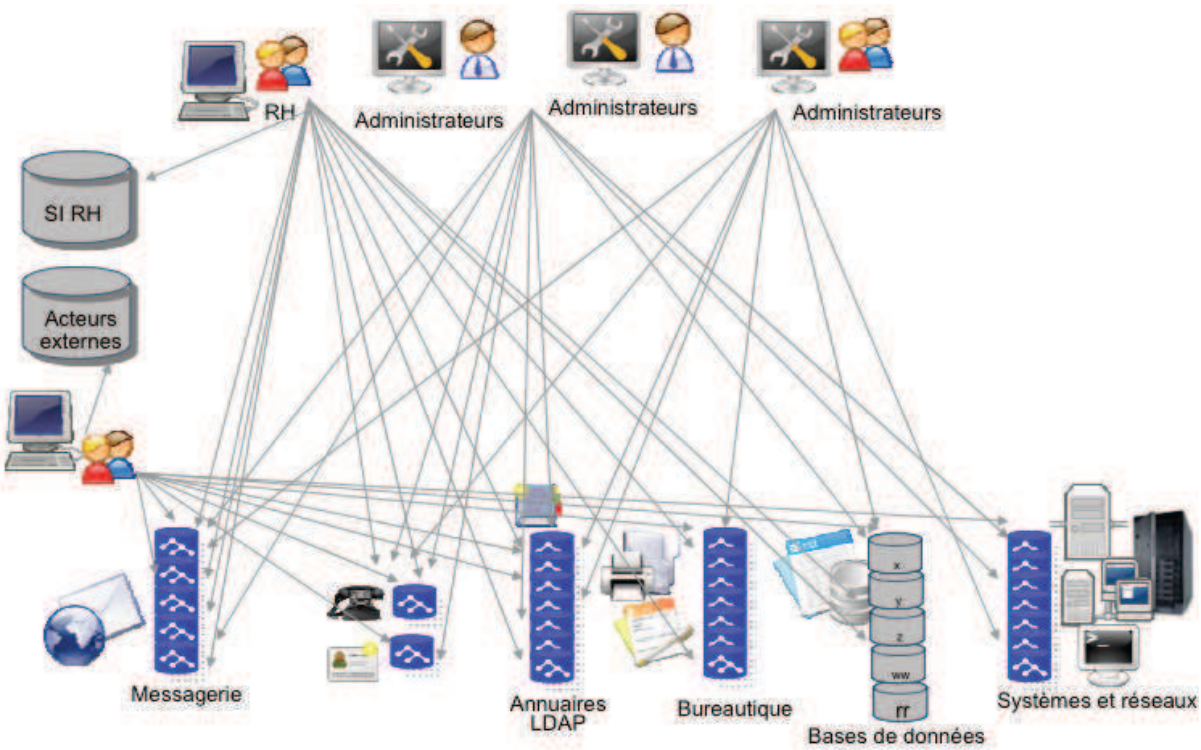

Figure 1 – Existant « standard » de la gestion des identités et des droits d'accès

- référencement dans le système de gestion de la paie
- attribution d'un bureau : référencement dans la base du service logistique
- attribution d'un numéro de téléphone : référencement dans la base téléphonique
- attribution d'un ordinateur, d'un identifiant et d'un mot de passe pour accéder au réseau : référencement dans le système bureautique
- attribution d'un badge pour l'accès aux locaux : référencement dans le système de gestion des badges
- droits d'accès à un restaurant d'entreprise : référencement dans la base du restaurant d'entreprise
- droits d'accès sur une application : référencement dans la base de l'application
- etc.

Dans la majorité des entreprises, ces opérations font appel à des annuaires qui ne sont ni compatibles entre eux, ni synchronisés (cf. Figure 1). Ainsi pour un nouvel utilisateur il faut saisir plusieurs fois les mêmes informations dans des systèmes différents par des personnes différentes et il en va de même en cas de modification d'une information. Cette mise à jour est parfois très longue ou que partiellement réalisée.

2.2. Justifications d'un projet de gestion des identités et des droits d'accès

Comme nous l'avons vu au chapitre précédent, l'absence de gestion globale des identités et des droits d'accès peut générer de nombreux problèmes, parmi lesquels :

- la perte de productivité due aux délais d'obtention des droits d'accès
- une charge importante d'administration (multiplication des administrateurs, réinitialisation des mots de passe, etc.)
- l'impossibilité de tracer les actions d'administration des droits et d'en contrôler la cohérence et la pertinence
- la difficulté d'auditer les accès aux ressources
- des entorses au principe de séparation des tâches
- le non respect des contraintes légales et/ou réglementaires (par exemple au travers d'un mauvais paramétrage des règles de gestion)

La justification d'un projet de gestion des identités et des droits d'accès reposera sur les

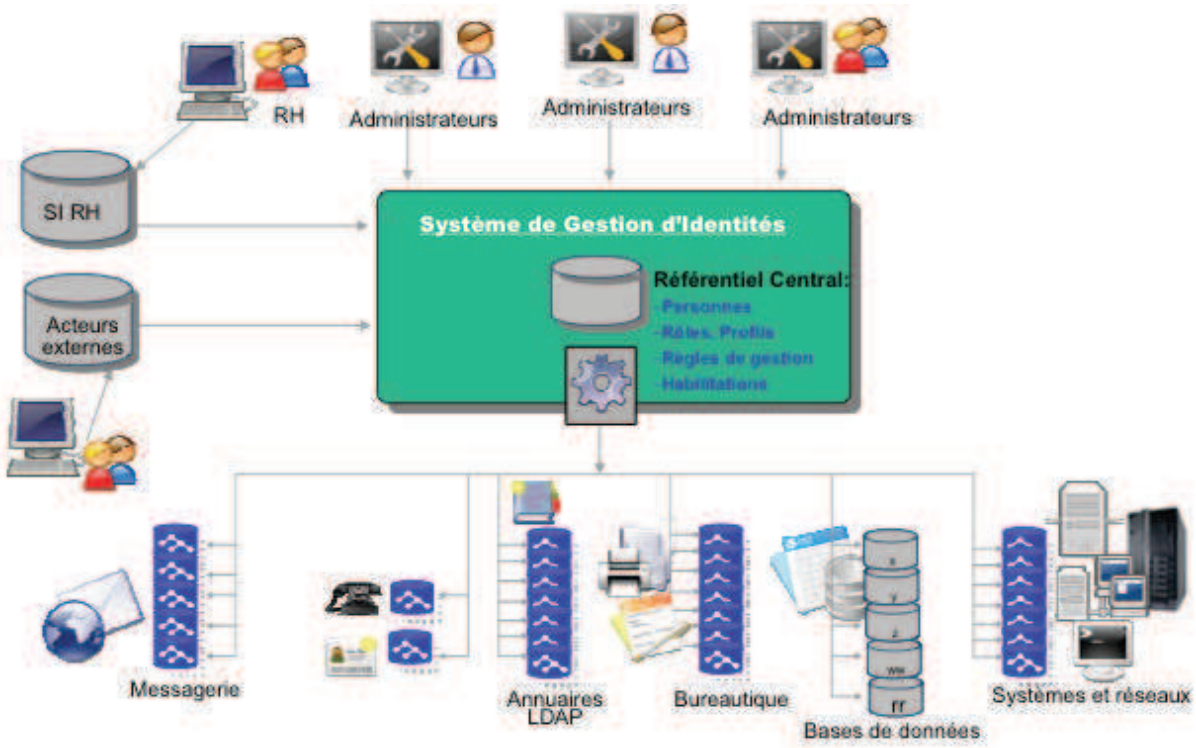

Figure 2 – Flux de mise à jour après la mise en place d'une gestion centralisée

améliorations suivantes :

- garantie de traçabilité et de la possibilité d'auditer afin de répondre aux obligations légales et/ou réglementaires
- repositionnement des « propriétaires fonctionnels » au centre du débat
- réduction des coûts d'administration
- amélioration de l'efficacité et de la réactivité
- amélioration de la sécurité (adéquation des droits aux besoins métier)

L'ensemble des flux présentés dans le chapitre précédent va pouvoir être représenté de la manière suivante après la mise en place d'une gestion centralisée des identités (cf Figure 2). Les informations sont mises à jour dans le référentiel central qui alimente ensuite automatiquement les annuaires ou bases de sécurité des différents environnements.

2.2.1. Garantie de traçabilité et d'auditabilité

Les principales lois et réglementations impliquant le SI et ayant des impacts directs sur les aspects traitant de la sécurité (en particulier traçabilité et auditabilité) sont présentées dans le chapitre – « aspects juridiques ».

D'une manière générale, ces lois et règlements « imposent » au Système d'Information des exigences de :

- continuité d'activité
- de séparation des tâches : par exemple, une même personne ne doit pas à la fois commander une fourniture ou prestation et valider sa réception
- de traçabilité et d'auditabilité : permettant de valider « qui a fait quoi » au sein du système d'information, et « qui a habilité qui »
- de respect de la vie privée. Déroger à ces exigences peut entraîner un risque juridique pour les responsables de l'entreprise

2.2.2. Réduction des coûts d'administration

Un système de gestion des identités et des droits d'accès permet d'alléger la charge de travail de l'équipe de « support informatique » (administration, help desk). Cet allègement résulte d'une part de l'automatisation de tâches de gestion de comptes (réduction du nombre d'administrateurs) et d'autre part de la diminution du nombre d'appels d'utilisateurs (perte ou oubli de nombreux mots de passe, relance de demandes d'accès, etc.).

Le système de gestion des identités peut permettre aux utilisateurs la gestion directe de certains aspects de leur profil (par exemple le mot de passe, l'adresse, les numéros de

téléphone, etc.).

2.2.3. Amélioration de l'efficacité et de la réactivité

Un système de gestion des identités et des droits d'accès permet de réduire le nombre d'interventions humaines par une automatisation de la propagation des droits sur les différents environnements concernés. La conséquence est à la fois une réduction des délais de mise à disposition des droits d'accès et une réduction des sources d'erreur (prise en compte systématique de tous les besoins liés à l'activité de l'utilisateur, garantie de cohérence dans les droits attribués).

Les gains générés concernent à la fois les utilisateurs internes (gain de productivité) et externes (amélioration de la qualité du service et de l'image de l'entreprise).

Sur un autre plan, lors d'une fusion ou d'une acquisition, il faut fournir le plus rapidement possible un accès aisé aux ressources rassemblées d'entreprises auparavant autonomes. Là encore, une solution de gestion des identités et des droits d'accès aidera à relever ce défi au travers d'un service d'intégration des informations multi plates-formes permettant de connecter les systèmes de chaque entreprise à la plupart des systèmes (nouveaux ou préexistants) de la nouvelle entité.

2.2.4. Amélioration de la sécurité

Un système de gestion des identités et des droits d'accès permet de renforcer la sécurité. Une telle approche conduit à établir des liens entre toutes les applications, bases de données et annuaires en s'appuyant sur des notions de rôle et de profil. Cette solution offre un point unique de gestion des règles de sécurité pour l'ensemble des systèmes concernés. Elle permet de créer simplement des règles d'accès et de sécurité, en cohérence avec la Politique de Sécurité des Systèmes d'Information et les besoins métier, puis de les propager automatiquement à tous les systèmes de l'entreprise.

La gestion centralisée des identités permet d'éliminer une source considérable d'erreurs d'administration pouvant causer des failles de sécurité d'accès au SI de l'entreprise. Elle permet également de résilier complètement et immédiatement les droits d'accès sur l'ensemble des systèmes lorsque des salariés ou personnels extérieurs quittent l'entreprise ou changent d'affectation et supprimer ainsi les comptes « fantômes ».

En mettant en place des processus maîtrisés d'habilitation, le système permet d'impliquer les responsables métiers dans le circuit d'habilitation et de ne plus laisser au seul administrateur technique la maîtrise des droits d'accès.

2.3. Concepts et modélisation

2.3.1. Les concepts

Ce chapitre introduit les concepts et les objets manipulés selon www.clusif.asso.fr et [1], avant de décrire les mécanismes de gestion des identités et des habilitations. ***En présence de nombreux modèles théoriques, et en l'absence de normalisation, la terminologie utilisée dans ce chapitre est issue de retours d'expérience.***

2.3.1.1. Personne (utilisateur / acteur)

Désigne une personne physique : les employés d'entreprise, les prestataires, les partenaires et les clients de l'entreprise qui, de par leur fonction, exercent une activité ayant vocation à leur permettre de bénéficier des applications et des ressources mises à disposition par l'entreprise.

2.3.1.2. Compte

À chaque personne peuvent être associés des comptes d'accès aux différents systèmes et applications. Le compte est défini par l'identifiant d'accès, un mot de passe (ou un authentifiant d'une autre nature), et plusieurs attributs supplémentaires en fonction de l'environnement dans lequel il est crée comme : la politique de mot de passe associée, l'accès externe autorisé ou non, l'état du compte, les modes d'authentification autorisés etc.

Il existe quatre types de comptes :

- le **compte global**. Ce compte, unique (à un utilisateur correspond un seul compte) identifie une personne dans le référentiel central de gestion des habilitations et est utilisé par tous les processus d'attribution des droits.
- le **compte utilisateur**. Ce compte donne l'accès à un utilisateur dans un environnement particulier auquel cet utilisateur est habilité. Chaque compte utilisateur est obligatoirement associé à une personne (et son identifiant unique). Sa création / suppression et la cohérence des informations associées est maintenue automatiquement par le système de gestion des habilitations en fonction des profils métiers attribués à la personne.
- le **compte d'administration**. Ce compte donne l'accès à un administrateur dans un environnement particulier. Ce compte n'est pas associé à une personne. Il ne correspond donc à aucune entrée dans le référentiel central. ***Leur usage doit être limité aux actes d'administration techniques des environnements et des applications***

dans les environnements où ces tâches ne peuvent pas être effectuées via les rôles d'administration. Exemple : Compte « Root » d'Unix. Les procédures mises en œuvre doivent garantir la traçabilité et l'auditabilité des personnes physiques auxquelles ces comptes administrateurs ont été autorisés d'emploi. Un changement d'affectation doit être associé à une procédure de changement des mots de passe.

- le **compte « de service fonctionnel ou technique »**. Ces comptes sont utilisés par les composants d'un système pour accéder aux services applicatifs et/ou données d'un autre système.

2.3.1.3. Rôle

Un rôle définit les permissions nécessaires à l'utilisation des objets (applications et/ou des ressources). Le rôle applicatif est un ensemble de droits propres à une seule fonction dans une application. Par exemple : le droit d'usage d'un jeu d'écrans et de menus correspondant à une fonction dans l'application.

Une habilitation donne à un utilisateur un ensemble de permissions dans une application. Elle est attribuée en fonction du poste opérationnel au sein de l'organisation et non à titre individuel. C'est le poste opérationnel qui détermine les rôles et les périmètres nécessaires.

2.3.1.4. Profil

Un profil fonctionnel regroupe un ensemble de rôles nécessaires à l'exécution d'une fonction métier. Ce profil peut également être vu comme un package de rôles applicatifs ou un niveau supérieur dans la hiérarchie des rôles. Un utilisateur peut avoir un ou plusieurs profils fonctionnels.

2.3.1.5. Le poste opérationnel

Le poste opérationnel (position de travail) correspond à une fonction métier exercée au sein d'un élément de structure (service, département, ...). Un poste opérationnel est toujours défini au sein d'un et un seul élément de structure. Le responsable de structure indique les postes qui lui sont attribués.

2.3.1.6. Groupe

Les utilisateurs peuvent être regroupés, dans le référentiel central, en groupes statiques ou dynamiques. Ces groupes sont utilisés pour faciliter la gestion en masse des habilitations.

2.3.1.7. Le périmètre

Le périmètre est utilisé par les applications et/ou systèmes pour affiner le contrôle

d'autorisation qu'ils réalisent. Le périmètre peut avoir trois types différents (temporel, géographique et fonctionnel) et être associé à : une personne, un compte (uniquement dans le cas de limitation des autorisations d'accès à des ressources d'un environnement), un rôle.

2.3.1.8. Périmètre temporel

Le périmètre temporel permet de restreindre les possibilités d'accès d'un utilisateur dans le temps. Plusieurs types de restrictions sont possibles : période (Date début – Date fin), plage horaire (Heure début – Heure fin), calendrier (date définie). Ces limitations sont appliquées par les différents systèmes d'autorisation en fonction de la capacité du système à gérer ce type de restriction. Le périmètre temporel peut être associé à : une personne, un rôle, une ressource.

Un périmètre temporel associé à une personne limite son accès à l'ensemble des ressources et applications en empêchant l'utilisateur d'établir une session en dehors de périodes autorisées.

2.3.1.9. Périmètre géographique

Le périmètre géographique permet de restreindre les possibilités d'accès d'un utilisateur en fonction du lieu à partir duquel il accède au SI. Plusieurs types de restrictions sont possibles : un lieu, un réseau précis, un ou plusieurs poste(s) de travail.

2.3.1.10. Périmètre fonctionnel

On désigne sous ce terme les limitations imposées par le programme de contrôle d'une application. Transmis à l'application lors de l'appel des transactions associées au rôle, il permet de gérer la sécurité applicative : le programme autorisera ou non certains traitements en fonction des données qui lui seront communiquées par le système d'habilitation (identifiant acteur, poste de travail, éventuellement lieu de présence ou d'affectation d'utilisateur, etc.). Le périmètre fonctionnel peut être associé à un rôle.

2.3.1.11. Mode d'authentification

Certaines applications critiques imposent un mode d'authentification particulier (authentification forte). Le système d'authentification fournit (dans le contexte de sécurité) l'information indiquant le mode d'authentification utilisé lors de l'ouverture de la session. Cette information est exploitée par le système de contrôle d'accès et/ou l'application. Le mode d'authentification peut être associé à un rôle ou à une ressource.

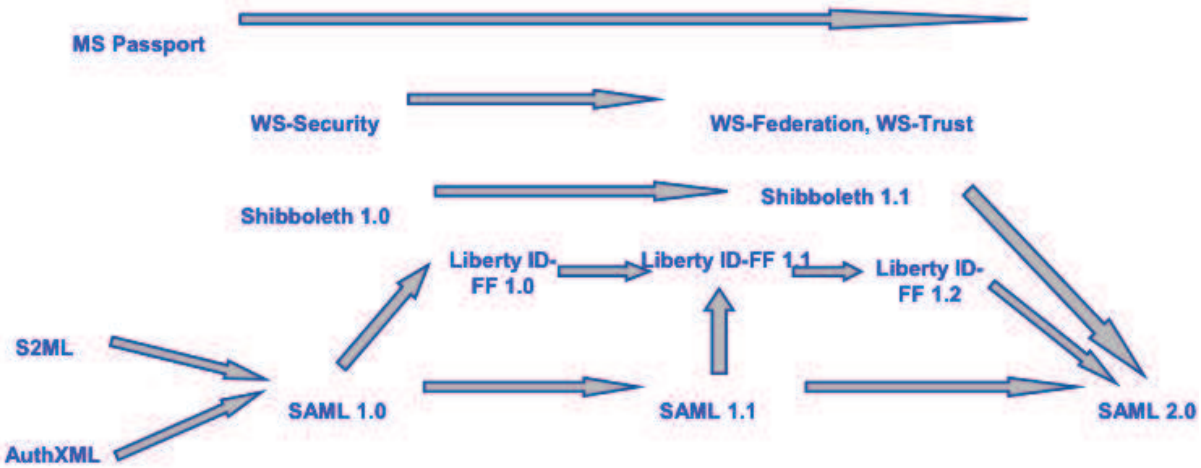

Figure 3 – Les « efforts » de normalisation

2.3.1.12. Ressource

La ressource est définie par : un libellé, les modes d'authentification nécessaires pour y accéder, les périmètres d'accès autorisés, les rôles ou les groupes de comptes autorisés sur cette ressource. Par ailleurs, l'autorisation d'accès du rôle ou des groupes de comptes peut être suspendue à tout moment si besoin. Une ressource désigne par exemple une ou des adresses IP de serveurs, une ou des URL, une commande de lancement d'une application, etc. À une ressource peuvent être associés : un périmètre temporel (durée de validité), un état (actif, suspendu). Elle est utilisée lors de la connexion des utilisateurs au réseau.

2.3.1.13. Environnement du SI

Le terme d'environnement du SI désigne un ensemble de ressources et de processus (système, sous-système, application, etc.) dont les droits d'accès sont gérés par un système de contrôle unique et autonome et administré par l'entité responsable.

2.3.1.14. Environnement externe

Le terme d'environnement externe désigne un ensemble de ressources et de processus (système, sous-système, application, etc.) gérés par des tiers et dont les droits d'accès sont gérés par un système de contrôle autonome et administrés par l'organisme tiers. Dans certains cas le contrôle d'accès peut s'appuyer sur les informations de sécurité fournies par le SI de l'entreprise (identité et/ou rôle).

2.3.2. Fédération d'identités

Il est impossible d'aborder la fédération des identités d'une manière exhaustive dans ce document dédié à la gestion des identités d'un organisme et/ou d'une entreprise. C'est un sujet très vaste qui a fait l'objet de multiples travaux.

Nous nous limiterons par conséquent à une description générale en indiquant de quelle manière le système interne et autonome peut s'ouvrir à l'extérieur et collaborer avec des systèmes tiers.

La fédération consiste à faire communiquer plusieurs systèmes de gestion des identités, afin d'éviter de constituer une solution centralisée, tout en assurant des services d'authentification unique, d'échanges d'attributs et de droits utilisateurs entre les différents sites auxquels ils ont accès. À l'heure actuelle les solutions sont concentrées sur les technologies Web mais des travaux sont en cours pour étendre leur champ d'application. Le diagramme suivant (Figure 3) présente la multiplicité des différentes initiatives et la

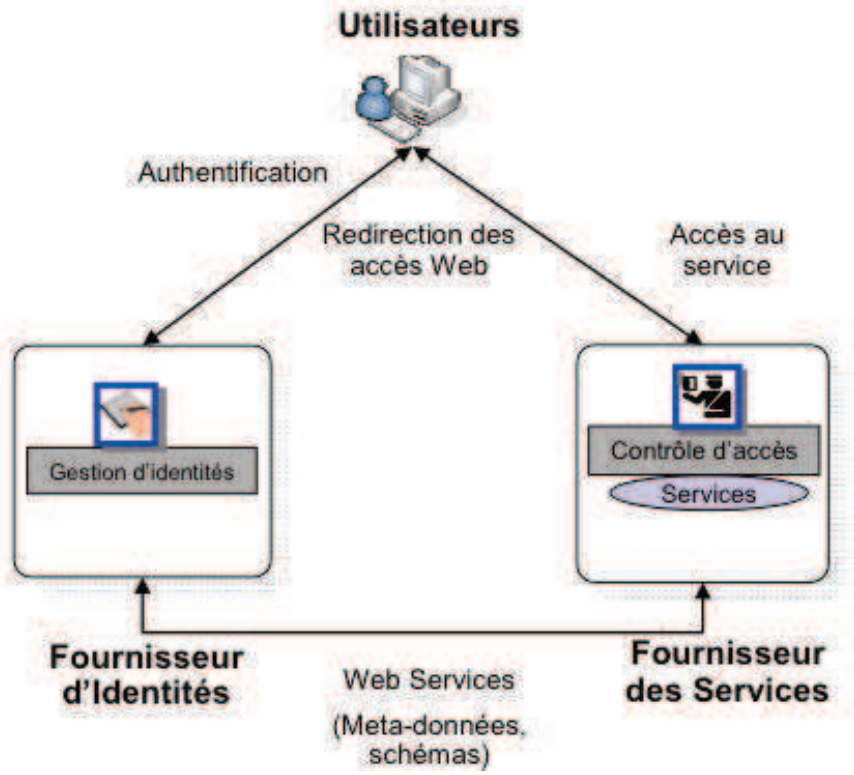

Figure 4 – Principe d'architecture de la fédération d'identités

convergence de tous ces travaux vers la normalisation autour de SAML 2.0.

2.3.2.1. SAML

SAML (Security Assertion Markup Language) a été initialement conçu pour permettre, entre autres, la délégation d'authentification. C'est devenu un standard OASIS en 2002. Il s'agit d'un ensemble de spécifications qui définit comment des services peuvent échanger des assertions de sécurité (authentification, autorisation, attributs), indépendamment des technologies utilisées par chacun de ces services. SAML s'appuie sur des standards pré existants (XML, SSL, etc.) et a été conçu avec suffisamment d'abstraction pour rendre inter opérables des systèmes hétérogènes et s'articuler au mieux avec d'autres mécanismes de gestion d'identités.

2.3.2.2. Les concepts de fédération

Les approches de centralisation et de fédération sont parfaitement complémentaires :

- la gestion des identités centralisée est une première étape de rationalisation des informations au sein de l'entreprise/organisme
- la fédération des identités répond aux besoins d'intégration des services d'identités entre différentes organisations (métiers, partenaires, fournisseurs, clients, etc.)

Les solutions de fédération d'identités s'appuient sur quelques concepts tels que :

L'identité fédérée *:*

- est un ensemble d'attributs fédérés, c'est-à-dire d'informations relatives à l'identité provenant de différentes sources et pouvant être mises en commun
- apporte des gains fonctionnels :
  - pour l'utilisateur : la possibilité de partager son identité entre différents systèmes
  - pour l'entreprise : la possibilité de s'associer avec d'autres partenaires au sein d'une fédération, afin que les identités d'un domaine puissent donner accès aux services d'un autre sans être obligé de mettre en œuvre une gestion lourde (et souvent presque impossible) de gestion des identités des utilisateurs des partenaires.

La répartition des responsabilités

La propagation d'identités et la fédération inter partenaires s'appuient sur une organisation tri partite (cf. Figure 4) :

- un fournisseur d'identité (Identity Provider ou IDP) : chargé d'authentifier l'utilisateur et de gérer son identité (enregistrement, provisioning, comptes et de mots de passe)

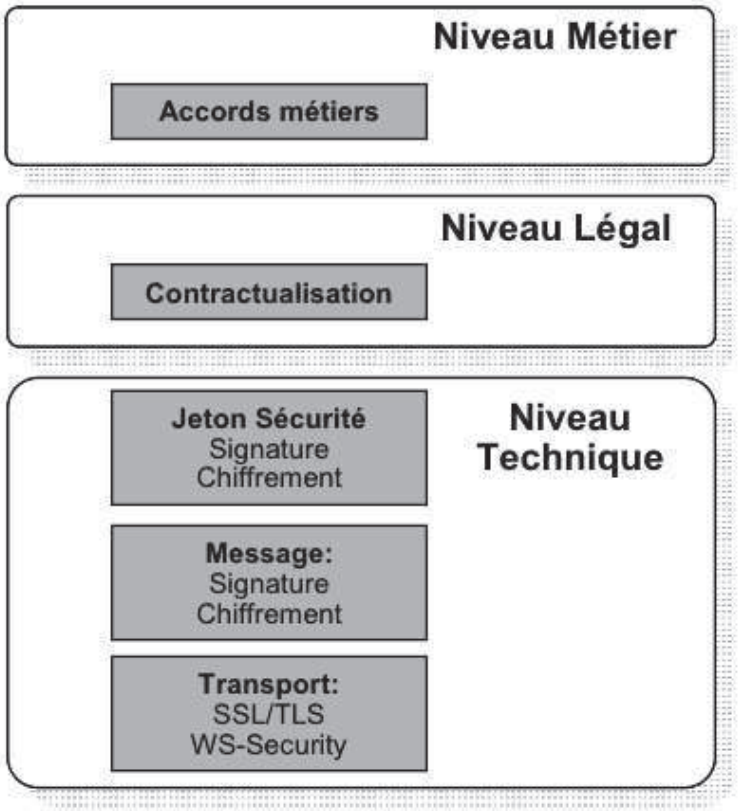

Figure 5 – Principes des relations de confiance

- un fournisseur de services (Service Provider ou SP) : chargé de lui fournir des services en fonction de ses habilitations sur la base des informations fournies par le fournisseur d'identités à qui il fait confiance
- l'utilisateur

Confiance entre les partenaires

Tous les mécanismes de fédération d'identités se basent sur le principe fondamental d'existence d'une relation de confiance entre les partenaires qui ont décidé de collaborer. Il est primordial que ces relations de confiance soient établies et gérés sur trois niveaux :

- **métier** où seront définis les services concernés par la fédération, les engagements de qualité de service et les conditions de mise en œuvre. En particulier le fournisseur de service pourra exiger une garantie de fiabilité et de niveau d'authentification de la part du fournisseur d'identité
- **légal** où seront formalisées et contractualisées les exigences métier et définis les moyens et les procédures de résolution de cas de litiges
- **technique** où seront définis les moyens techniques de mise en œuvre des liens sécurisés entre les sites, les formats de jetons de sécurité et les processus de validation de l'authenticité des informations échangées. (cf. figure 5)

La gestion d'identités fédérées ou Federated Identity Management (FIM)

- est une façon standardisée de gérer de bout en bout le cycle de vie des identités, au sein de l'entreprise et entre les entreprises
- elle permet :
    - d'étendre les pratiques de gestion des identités de l'entreprise
    - de simplifier la gestion des identités au-delà des frontières de l'entreprise
    - de faciliter l'intégration métier des partenaires via des relations de confiance et le partage d'information dans un environnement sécurisé

La gestion FIM se base sur les définitions suivantes :

- le mapping d'identités/attributs définit les attributs d'identités à partager et leur correspondance chez les différents partenaires
- la gestion/provisioning des comptes définit les processus de gestion d'information d'identités :
    - les informations à partager

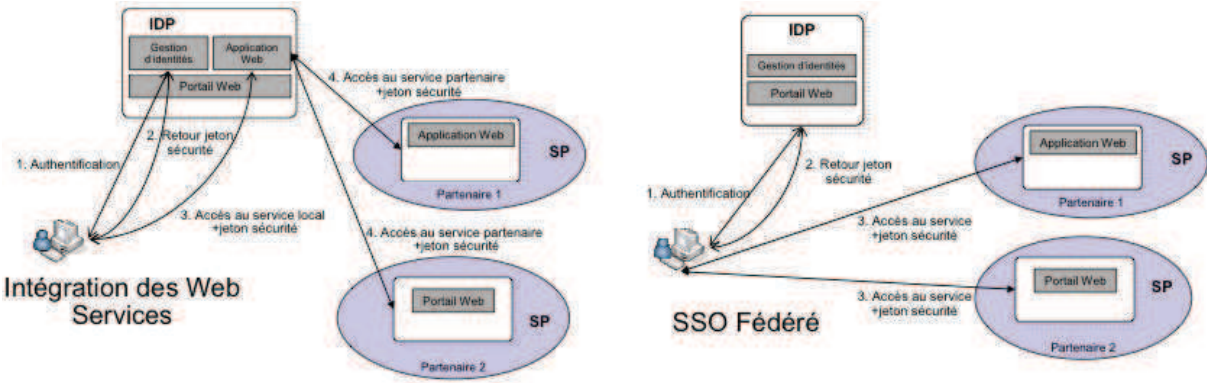

Figure 6 – Exemples de mise en œuvre de fédération des identités

    - les informations que l'utilisateur peut gérer
    - les informations qui peuvent être provisionnées automatiquement
- la jonction des comptes (Account linking) soit :
    - définit les processus de création et de suppression d'un lien entre un compte d'identité unique et les comptes de service chez les fournisseurs de service
    - établit les liens entre les différents comptes de la même personne sur les divers sites des partenaires
- la confiance définit les processus et les moyens utilisés pour assurer une communication sécurisée sur plusieurs niveaux : connexions/transport, messages, jeton

La gestion FIM met en œuvre les services suivants (cf. figure 6) :

- le Web SSO inter domaines ou SSO Fédéré – Accès interactif et sécurisé aux applications des partenaires qui met en œuvre les techniques de :
    - Push SSO : l'utilisateur accède d'abord au site de son IDP et est redirigé ensuite vers le SP
    - Pull SSO : l'utilisateur accède au site du SP et est redirigé vers l'IDP si non authentifié
    - SLO – Single Log Out : déconnexion de l'ensemble de sites ou l'utilisateur est autorisé d'accès
    - WAYF : Le WAYF (pour Where Are You From? « d'où êtes-vous ? ») est un service optionnel dont le but est d'orienter l'utilisateur vers son IDP, sécurité de Web Services – Communication sécurisée entre les applications conforme aux spécifications de OASIS « Web services security specifications »
- gestion du cycle de vie d'identité - federated provisioning
    - Account linking à l'initiative de l'IDP
    - Account linking à l'initiative du SP
    - Provisioning du SP en temps réel (durant la phase de SSO)
    - Provisioning du SP « a priori » (durant la phase de gestion d'identités IDP)

2.3.3. Modèle RBAC

Le modèle de sécurité RBAC (Role Based Access Control) est principalement issu d'Internet afin de prendre en compte des applications déployées sur de vastes organisations ou des applications inter organisations (Extranet par exemple). Ce modèle permet en particulier de simplifier l'administration des droits et de prendre en compte la délégation de

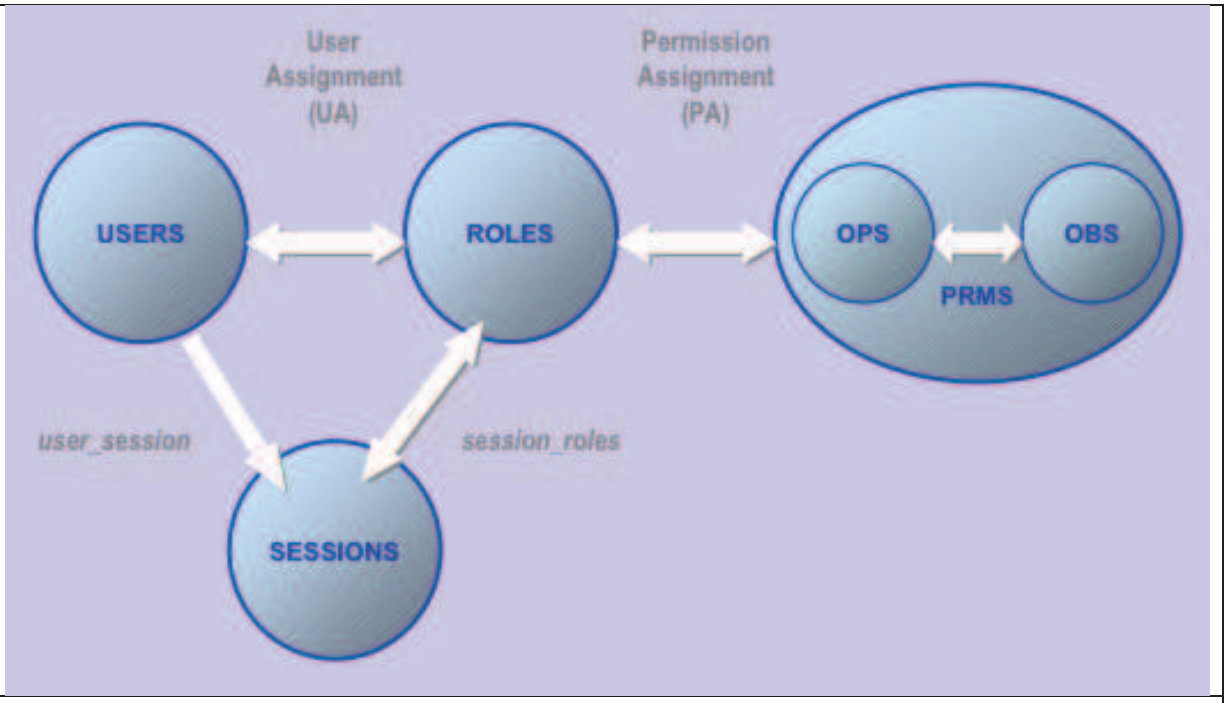

Figure 7 – Modèle de base RBAC

l'administration.

Les concepts du RBAC (http://csrc.nist.gov/rbac/) ont servi de base à la norme établie par American National Standard et référencée sous le N°:ANSI INCITS 359-2004 (approuvée le 19 Février 2004).

Ce modèle tend à se généraliser dans l'industrie et un nombre croissant de produits supportent un modèle d'habilitation "orienté rôles".

Le modèle RBAC se distingue du modèle DAC (Discretionary Access Control) popularisé par Unix. Le modèle DAC est centré sur les ressources physiques (fichier, exécutable, etc.) et identifie un propriétaire ainsi que des groupes d'utilisateurs ayant des droits sur la ressource (lecture, écriture, etc.).

Le modèle RBAC modélise des fonctions métiers plutôt que des accès à des ressources informatiques. Un rôle correspond à une fonction au sein d'une organisation. Le principe de base du RBAC est que deux utilisateurs ayant les mêmes rôles ont les mêmes droits sur le système.

L'administration des rôles est ainsi facilement compréhensible par des administrateurs métiers et peut être déléguée. Les associations entre les rôles et les ressources physiques sont modélisées séparément par les concepteurs d'application et maîtrise d'ouvrage métier.

2.3.3.1. Modèle de base RBAC

Le standard propose un modèle de base (Core RBAC) ainsi que les extensions présentées dans les sous-chapitres suivants.

Les concepts manipulés par le modèle RBAC (cf. figure 7) sont les suivants :

- USERS (Utilisateurs) : comptes permettant aux utilisateurs de se connecter au système
- ROLES (Rôles) : fonctions métiers dans des organisations ou des périmètres donnés
- OBS (Objets) : objets informatiques à protéger
- OPS (Opérations) : opérations possibles sur les objets
- PRMS (Permissions) : autorisation d'effectuer l'opération X sur l'objet Y
- SESSIONS (Sessions) : session temporelle, chaque session est associée à un utilisateur pour une période de temps limitée

Principe de privilège minimum

Le fonctionnement de ce modèle et l'intégrité du système sont garantis si l'attribution des permissions respecte le principe de privilège minimum. Ce principe exige que l'utilisateur ne dispose pas de plus de droits que nécessaire à son travail. Ce qui implique, que les

permissions affectées à un rôle constituent le strict minimum nécessaire à l'accomplissement des tâches relatives à ce rôle.

2.3.3.2. Le modèle RBAC hiérarchique

Le modèle RBAC hiérarchique (Hierarchical RBAC) ajoute au modèle de base le support de hiérarchie des rôles. La hiérarchie établit les liens de parenté entre plusieurs niveaux des rôles et permet aux rôles « parents » de disposer des permissions attribuées aux rôles « enfants ». Le standard admet deux types de hiérarchies :

- le **modèle hiérarchique général** (General Hierarchical RBAC) : cette variante établie des relations multiples entre plusieurs « parents » et « enfants »
- le **modèle hiérarchique limité** (Limited Hierarchical RBAC) : cette version limite la relation à une simple structure d'arborescence. Ce qui veut dire qu'un rôle ne peut avoir qu'un seul « parent »

2.3.3.3. Le modèle RBAC avec contraintes

Le modèle RBAC avec contraintes (Constrained RBAC) ajoute au modèle la contrainte de séparation des pouvoirs. Cette contrainte permet d'inclure dans le modèle la gestion de conflits d'intérêts et assurer que les utilisateurs bénéficieront des permissions selon la politique définie par l'organisation, et ne pourront pas abuser de cumul non contrôlé de droits. On distingue la séparation Statique des Pouvoirs (SSD - Static Separation of Duty Relations) et la séparation Dynamique des Pouvoirs (DSD - Dynamic Separation of Duty Relations).

2.3.4. Autres modèles

D'autres modèles de sécurité existent, plus orientés ressources. Ces modèles tels que DAC et MAC sont très utilisés dans certains contextes (militaire) mais sont moins adaptés que le modèle RBAC à l'approche actuelle de la gestion des identités.

2.3.5. Modèle fonctionnel et de données

2.3.5.1. Principe de fonctionnement

La gestion des habilitations est basée sur les principes suivants :

- tout accès au Système d'Information est conditionné par une authentification et une autorisation. *L'authentification peut être déléguée à un système tiers de confiance*.
- tout acteur du système est déclaré d'une manière unique dans un référentiel central de sécurité en tant que personne physique et peut disposer de comptes dans différents environnements et de permissions dans les applications en fonction des habilitations

accordées. *La cohérence de ces informations est maintenue automatiquement par le système de gestion des habilitations*.

- **toute habilitation est attribuée en fonction du poste opérationnel au sein de l'organisation et non à titre individuel. Le poste opérationnel détermine les rôles et les périmètres nécessaires.** *Le poste opérationnel (position de travail) correspond à une fonction métier exercée au sein d'un élément de structure*.
- un utilisateur peut avoir un ou plusieurs profils. L'attribution se fait en fonction du poste opérationnel de l'utilisateur.
- un profil regroupe un ensemble de rôles nécessaires à l'exécution d'une fonction métier,
- un rôle définit les permissions nécessaires à l'utilisation d'une application ou des ressources.
- les définitions des rôles, profils ainsi que l'attribution des profils aux personnes se font via un outil central qui propage ensuite les informations nécessaires vers toutes les applications et environnements concernés. L'association personne / rôle applicatif (ou profil applicatif) est gérée hors application, lorsque l'application l'autorise. *Cela permet d'éviter d'avoir à développer une gestion des droits des utilisateurs dans l'application*.
- l'association personne / rôle applicatif est gérée de façon statique et explicite dans l'annuaire de sécurité (plutôt que de façon dynamique à base de règles de gestion organisationnelles). L'évaluation des droits, par le système de contrôle d'accès, doit être basée pour l'essentiel sur la consultation de ce rôle applicatif explicite (pas d'interprétation de règles complexes dans le processus d'autorisation).
- un contrôle permanent de l'adéquation des profils attribués aux rôles effectifs des personnes.
- la définition des droits d'accès aux ressources ou aux groupes de ressources est administrée dans les environnements cibles via les outils natifs de ces environnements. *Les administrateurs locaux continuent de gérer les ressources elles-mêmes et leurs associations avec les habilitations*.
- le contrôle d'accès lui-même est assuré soit par les composants du socle technique (cas de ressources et applications Web) soit par les mécanismes natifs des environnements (OS, NOS, messagerie, groupware, etc.) soit par les applications.

Il est à noter donc que les informations nécessaires à la gestion des habilitations sont distribuées dans différents référentiels. On distingue :

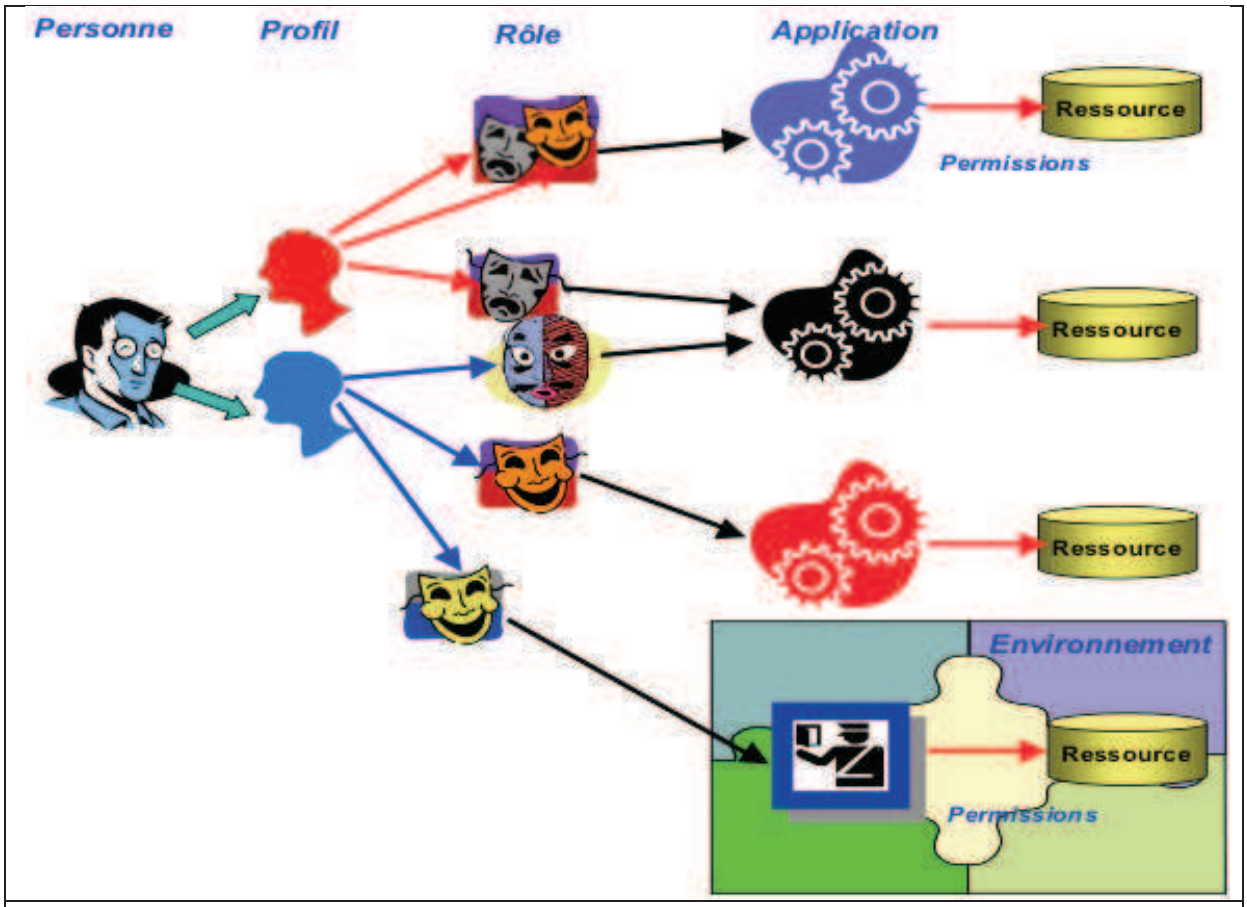

Figure 8 – Modélisation des droits utilisateurs sur les Ressources

- le **référentiel des habilitations**, référentiel mis à jour lors de la gestion des utilisateurs, des habilitations et des mots de passe.
- les **annuaires de sécurité**, qui ne contiennent qu'une partie des données du référentiel des habilitations (par exemple, les profils n'y figurent pas) et sont utilisés par les services d'authentification et d'autorisation, tels que : systèmes d'exploitation, sous-systèmes de gestion des droits, LDAP, NOS (Network Operating Systems) et systèmes de partage des fichiers, messageries et systèmes collaboratifs.

2.3.5.2. Modèle de données

Le système de gestion centralisée d'habilitations doit permettre de modéliser les droits utilisateurs sur les ressources, (y compris les applications et les environnements). Cette modélisation passe par l'utilisation de rôles applicatifs, profils utilisateurs.

Cette modélisation doit permettre de fournir une visibilité « métier » des utilisateurs (le travail quotidien des utilisateurs). Cette visibilité sera fournie au travers des profils utilisateurs. La Figure 8, explicite la modélisation des droits utilisateurs sur les ressources.

Un service, ou une application, est décomposé en fonctions élémentaires appelées « ressources ». Un rôle applicatif est un ensemble de droits sur un ensemble de ressources :

- si l'autorisation a été entièrement déléguée (externalisée) par l'application, le rôle applicatif constitue simplement un droit d'accès aux ressources (typiquement droit d'accès à des URL Web)
- si le système central se contente de fournir le rôle applicatif au service, c'est au service d'interpréter le sens du rôle applicatif. Des exemples typiques de rôle applicatifs sont : administrateur, administrateur délégué, utilisateur standard, etc. Un rôle applicatif ne peut cibler les ressources que d'un seul service.

Un profil utilisateur est un ensemble de rôles applicatifs. Un profil utilisateur se doit de décrire l'activité de l'utilisateur. Ce profil utilisateur permettra de lister l'ensemble des services que doit utiliser un utilisateur pour effectuer son travail quotidien. Afin d'éviter de nouvelles saisies inutiles, on pourra ajouter des profils utilisateurs dits de « factorisation » :

- Profil « général » : décrit l'accès standard messagerie, pages jaunes, pages blanches, etc.
- Profil « métier » : décrit l'ensemble des services accédés en standard par une personne appartenant à un métier.

Des contraintes de non cumul de pouvoir peuvent être associées aux rôles ou aux profils.

Ainsi, une personne pourra être associée à plusieurs profils utilisateur : profil général, profil métier(s).

2.4. Les processus et les acteurs

2.4.1. Les acteurs

Les principaux acteurs sont :

- l'utilisateur
- l'administrateur du workflow
- l'administrateur système et/ou services
- l'administrateur de mots de passe
- l'approbateur métier
- l'approbateur technique
- le propriétaire fonctionnel
- les gestionnaires des ressources humaines internes et externes (DRH, gestionnaires de contrats, etc.)
- l'auditeur
- le RSSI

L'administrateur a pour responsabilité de gérer les profils correspondant aux fonctions rencontrées dans l'entreprise.

2.4.2. Les processus

Les principaux processus sont :

- la gestion des mots de passe
- l'allocation des ressources (provisioning)
- l'administration et l'audit
- la gestion des processus
- la gestion des habilitations
- le contrôle d'accès
- la gestion personnelle (mots de passe, demandes d'accès, etc.)
- la réconciliation (association des comptes et des personnes, recherche de comptes orphelins, alertes et/ou correction des écarts)

2.5. Les fonctions et les services

2.5.1. Gestion des identités et des habilitations

La gestion centralisée des personnes et de leurs comptes d'accès sur toutes les

applications et les environnements interfacés avec le système central d'habilitation fournit aux administrateurs une vue unifiée de l'ensemble des utilisateurs et de leurs attributs de sécurité.
La gestion des habilitations doit permettre de gérer le contenu du référentiel de sécurité. Elle se décline sous plusieurs aspects :

- la **gestion de personnes** (création, modification, suppression des comptes utilisateurs du référentiel central), grâce à :
  - une alimentation automatique du référentiel central à partir de référentiels tiers (fichiers du personnel, référentiels acteurs, structures, etc.) incluant utilisateurs internes et si nécessaire externes et partenaires, etc.
  - une alimentation manuelle par des administrateurs centraux ou délégués
  - la **déclaration des gestionnaires** : désignation des gestionnaires fonctionnels qui interviennent sur la gestion fonctionnelle des personnes
- la **déclaration des domaines d'administration** : périmètres de travail fonctionnel définissant un ensemble de populations géré par un gestionnaire. Ce périmètre pourra être de type géographique, métier, organisationnel. Dans certains cas, les périmètres de travail des gestionnaires peuvent se recouper.

  L'administration du référentiel de sécurité, qui se fait via :
  - la définition de niveau hiérarchique d'administration. Un niveau est attribué à chaque administrateur pour chaque processus d'alimentation
  - la mise en place d'une fonction permettant à chaque administrateur de déléguer simplement tout ou partie de ses droits à un ou plusieurs autres administrateurs
  - la déclaration des ressources applicatives ou services qui permet à un nouveau service ou application d'être déclaré dans le référentiel de façon à bénéficier du contrôle d'accès sur les ressources de l'application/service
  - la déclaration des rôles, profils, groupes et périmètres. Cette déclaration permet de créer des profils et des rôles afin de permettre ensuite l'automatisation de la création et de la modification des comptes utilisateurs sur les systèmes et applications cibles

    L'administrateur a pour responsabilité de créer les profils correspondant aux fonctions rencontrées dans l'entreprise. A chaque profil correspond un ensemble de rôles définissant la manière dont les comptes doivent être créés sur les plates-formes cibles.

    Toutes les déclarations peuvent être effectuées de manière interactive, mais également avec des moyens permettant de rejouer cette déclaration d'une plate-

forme de test/intégration/pré production vers une plate-forme de production sans risque d'erreur (par exemple, possibilité d'export de la déclaration)

- la **mise en œuvre des règles de gestion de mots de passe** définies pour chaque système et application cible (longueur, durée de validité, etc.) et respectant la politique de sécurité
- création d'un système de « workflow » pour la gestion d'attribution, selon les règles préétablies, des profils et des rôles aux personnes (utilisateurs). Les principales fonctions du workflow sont :
  - l'affectation d'un profil à un utilisateur
  - la suspension ou suppression d'un utilisateur
  - l'activation d'un utilisateur
  - le changement de profil (changement des habilitations)
  - la réinitialisation / changement d'un mot de passe sur l'ensemble des comptes propres à l'utilisateur
  - la délégation des droits opérationnels ou d'administration
  - la modification de la configuration du système de workflow

2.5.2. Alimentation aval (Provisioning)

Selon [4] le système de gestion des habilitations est le point central d'alimentation des comptes (identifiant, mot de passe, autres attributs) pour des environnements techniques et des applications du Système d'Information. Dans certains cas (comme l'initialisation), des informations pourront être remontées des environnements.

Le provisioning permet d'alimenter les référentiels utilisateurs des services cibles à partir du référentiel central mais aussi de détecter des modifications des référentiels. Le provisioning permet de répondre aux besoins suivants :

- association d'une fiche personne à un ensemble de comptes
- calcul automatique de certains attributs des comptes à partir d'autres comptes, ou des attributs de la fiche de personne
- pour les attributs ne pouvant être calculés automatiquement, et étant propriété du référentiel cible, leurs valeurs devront être remontées du référentiel cible pour consolidation dans le référentiel central
- toute modification dans le référentiel central sur un compte doit être répercutée sur le référentiel cible associé au compte
- toute modification effectuée en direct sur un référentiel cible doit être remontée sur le

référentiel central ; si la modification est autorisée, celle-ci est consolidée au sein du référentiel central, et répercutée si nécessaire sur les autres référentiels cibles (changement de mot de passe, changement d'adresse e-mail) ; si la modification est non autorisée, la modification effectuée sur le référentiel cible doit être écrasée avec l'ancienne valeur stockée dans le référentiel central (comme toute autre action de provisioning, cette action devra être tracée)

- tout compte sur un référentiel cible non rattaché à une personne doit pouvoir être détecté
- tout compte dont un attribut ne respecte pas la règle de création doit pouvoir être détecté ; toutefois, la règle de création doit pouvoir accepter les exceptions de sorte à pouvoir prendre en compte l'historique des comptes actuels.

2.5.3. Le service de changement et de synchronisation des mots de passe

Ce service permet de garantir la conformité à la politique de gestion des mots de passe en vigueur. La conformité est vérifiée en amont et propagée ensuite. De cette façon les mots de passe de tous les environnements respectent la même politique générale.

Il est possible également d'intercepter les demandes de changement des mots de passe initiées dans certains environnements. Dans ce cas le nouveau mot de passe est dirigé vers le service central qui le distribue selon le mode standard.

Par le même mécanisme, l'administrateur peut également forcer un nouveau mot de passe pour un utilisateur. Ce mot de passe doit être obligatoirement changé à la première connexion. Le système offre un service du changement global de mot de passe accessible depuis le réseau de l'entreprise et un service de réinitialisation via une interface spécifique (réinitialisation du mot de passe à l'aide de challenges (questions/réponses) statiques ou dynamiques) ou via un Help Desk.

Ce service permet à l'utilisateur de demander en une opération le changement de mot de passe dans tous les environnements auxquels il peut accéder en fonction de son métier dans l'entreprise.

2.5.4. Processus et mécanismes de contrôle d'accès et d'évaluation des droits

Tout accès au Système d'Information est conditionné par une authentification et une autorisation.

2.5.4.1. Authentification

Le système de gestion des identités offre un service d'authentification pour les applications et les environnements SI. Une requête contrôle, dans l'annuaire de sécurité,

l'existence du compte de la personne, le mot de passe (ou les crédentiels d'autre nature) et le statut du compte.

Le service d'authentification peut être fourni par plusieurs composants différents en fonction d'environnement ou d'application. La cohérence des tous les annuaires de sécurité est assurée par le système central de gestion des identités. De plus, en complément de celui-ci, le service d'authentification permet principalement :

- de réaliser une fonction de SSO (Single Sign On) pour améliorer le confort des utilisateurs
- de coopérer avec un ou des systèmes d'authentification forte (PKI, Biométrie, Carte à puce, Clé USB, Token, etc.)
- de coopérer avec un, ou des systèmes d'authentification tierce pour déléguer l'authentification à un système tiers de confiance
- de pouvoir imposer en fonction de la ressource, du périmètre temporel et géographique, un mode d'authentification pour cette personne

2.5.4.2. Autorisation

Après la phase d'authentification des utilisateurs, le système pourra autoriser ces utilisateurs sur l'environnement technique ou l'application auquel ils tentent d'accéder par le contrôle de leurs droits d'accès. Le service d'autorisation est chargé d'évaluer les droits effectifs sur la base des informations (identité et preuves d'authenticité) fournies par le service d'authentification.

Le contrôle d'accès lui-même est assuré soit par les composants du socle technique (cas de ressources et applications Web) soit par les mécanismes natifs des environnements (OS, NOS, messagerie, groupware, etc.) soit par les applications.

Les modes d'autorisation supportés peuvent être classés de la manière suivante:

- **autorisation en entrée du réseau** : le système de gestion des identités permet ou non l'accès (l'ouverture de session) à l'infrastructure d'accès à l'ensemble de services supportés en amont de l'application. L'évaluation se fait en fonction de l'utilisateur, du périmètre temporel, périmètre géographique et de la typologie de sa tentative d'accès. Le service d'autorisation d'entrée réseau est chargé également de gérer les connexions des utilisateurs une fois le contrôle d'autorisation réalisé. Des paramètres de connexion (timer de déconnexion après inactivité, durée maximum de connexion...) spécifiques seront contrôlés en fonction de l'utilisateur, de la ressource à laquelle il accède, du périmètre temporel, géographique et/ou de sa typologie d'accès.

- **autorisation de type Rôle / Ressource/ Application** : autorisation de type simple : accès ou non au service (en fonction du rôle de l'utilisateur, du périmètre temporel, périmètre géographique et du mode d'authentification de sa tentative d'accès). Les ressources du service cible sont modélisées dans le référentiel et c'est le système de gestion des identités qui permet ou non l'accès à ces ressources (filtrage des URL des pages Web statiques ou dynamiques). Dans le cas d'applications capables d'appliquer le modèle RBAC (comme les rôles J2EE) le système de gestion des identités positionne les rôles et les droits effectifs sont évalués par l'application elle-même.
- **autorisation Environnement**: D'une manière générale, les autorisations des environnements non Web sont validées par les mécanismes natifs déjà présents. Ainsi ce sont les OS, NOS, messagerie, groupware qui contrôlent les droits d'accès à leurs propres ressources (système de fichiers, imprimantes, transactions, etc.) sur la base des informations détenues dans leurs structures de sécurité internes. Ces informations sont mises à jour, d'une manière cohérente avec les rôles des utilisateurs, par le service de gestion centrale des habilitations. La définition des droits d'accès aux ressources ou aux groupes de ressources en fonction des rôles (modélisées en tant que groupes) est administrée dans les environnements cibles via les outils natifs de ces environnements.
- **autorisations externes :** Les systèmes externes partenaires peuvent autoriser l'accès à leurs ressources tout en déléguant au SI d'entreprise l'authentification de la personne et le processus de détermination du rôle actif de la session en cours. Le système de l'entreprise prend en charge l'authentification de la personne et la recherche du rôle actif de l'application demandée. Ce rôle et l'identité sont communiqués au système externe par un jeton via un canal sécurisé. Les modalités et les mécanismes précis peuvent varier en fonction du système du partenaire. Ces mécanismes feront souvent appel aux solutions de fédération d'identités telles que décrites précédemment.

2.6. Démarche

La mise en œuvre d'un système de gestion des identités s'inscrit globalement dans un cadre classique d'un projet informatique. Toutefois, par son caractère transverse à la fois sur le plan fonctionnel et technique, ce type de projet impose une démarche adaptée dont les grandes lignes sont présentées dans ce chapitre.

Les facteurs clés critiques du succès d'un projet de gestion des identités sont d'abord d'ordres fonctionnels et organisationnels :

- identification des référentiels sources de données personnelles pour les différentes

populations (internes, externes, clients, etc.). A noter que ces référentiels doivent être fiables

- définition des profils métiers et des rôles liés aux outils nécessaires à l'exécution des tâches associées et indépendants d'organisation hiérarchique
- désignation ou mise en place d'un organisme (comité inter métiers, organisation du travail, direction des risques, etc.) responsable de la validation de définition des rôles, des profils et des règles de leur affectation
- définition des règles de séparation des pouvoirs (conformité aux réglementations du secteur d'activité)
- identification et formalisation des processus de gestion du cycle de vie des droits (arrivée, départ, mutation d'une personne), des règles d'approbation, etc.
- engagement fort de la Direction Générale indispensable pour le pilotage d'un projet transverse

Il est à noter que certaines normes ou codes de bonnes pratiques (tels que l'ISO 17799, etc.) abordent, voire justifient la gestion des identités et des droits d'accès.

2.6.1. Définition du périmètre

Le périmètre d'un projet de gestion des identités et de contrôle d'accès pour une organisation donnée doit prendre en compte :

- les activités métier et leurs propres règles de sécurité
- les différentes populations concernées (personnel interne, prestataires, fournisseurs, visiteurs, intérimaires, auditeurs, etc.)
- les processus de définition et d'attribution des droits
- les ressources prises en compte :
    - ressources à protéger (locaux, serveurs, composants techniques, applications, bases de données, documents, etc.)
    - ressources attribuées (PC, téléphone, PDA, carte à puce, etc.)

Un périmètre plus restreint peut être envisagé dans une première phase du projet pour en faciliter la mise en œuvre progressive (par exemple en considérant comme prioritaires des domaines métiers ou des populations).

2.6.2. Les acteurs du projet

Un projet de gestion des identités et de contrôle d'accès est transversal et concerne de nombreux acteurs internes, voire externes.

2.6.2.1. Principaux acteurs du projet

Le sponsor

C'est une personne qui dispose d'un haut niveau de décision (idéalement, il fait partie de la Direction Générale) et soutient le projet au travers :

- d'une communication adaptée à l'ensemble des parties prenantes du projet
- de validation stratégique tout au long du projet

Le Maître d'Ouvrage du projet

C'est le commanditaire du projet. Il peut s'agir des Ressources Humaines, d'une Direction Métier ou de la Direction des Systèmes d'Information.

Il est possible que ce soit la même personne que le sponsor.

L'équipe projet (Maître d'œuvre du projet)

Elle doit associer des compétences métier, en organisation, en architecture de Système d'Information et en sécurité.

Les propriétaires fonctionnels

Ils sont considérés comme les propriétaires des ressources et des processus à protéger. Ils définissent et valident les profils, les rôles et les droits associés ainsi que leurs modalités d'attribution. Cette définition exige une vision globale et une acceptation générale. Il est donc nécessaire qu'une entité réunissant tous les métiers de l'entreprise ou de l'organisme et disposant de l'autorité nécessaire pour entériner les choix soit chargée de piloter cette activité. Il peut être nécessaire de la créer pour le besoin du projet si aucune entité existante ne dispose de cette capacité. Comme mentionné en introduction de ce chapitre, c'est un facteur clé de réussite d'un projet de gestion des identités.

Les gestionnaires du personnel interne et des intervenants ou utilisateurs externes

Étant responsables des référentiels de données personnelles, ils participent à la définition et à la validation des interfaces avec leurs systèmes.

La Direction des Systèmes d'Information

Elle a en charge la gestion des différents systèmes et contribue à la conception des solutions et à leur intégration.

Le RSSI

Le RSSI vérifie les solutions en s'assurant que le niveau de sécurité est conforme aux exigences de l'entreprise. Une fois le projet terminé (voire en cours de projet), il audite et valide la bonne mise en œuvre du système de gestion des identités et des droits d'accès.

2.6.2.2. Autres contributeurs au projet

L'organisation

Lorsqu'il existe, le service de l'organisation valide les circuits d'attribution. Dans le cas contraire, cette validation est effectuée par les Directions Fonctionnelles.

Le service juridique

Il vérifie la conformité des solutions aux exigences légales et valide les clauses de partage des responsabilités des différents acteurs externes.

Les prestataires en cas d'externalisation de services

Leurs engagements doivent intégrer les nouvelles règles et dispositifs de gestion des droits. Dans certains cas il peut arriver que les contrats d'externalisation doivent être modifiés puisque les domaines de responsabilité changent. Le prestataire pourra, par exemple, être toujours en charge de l'infrastructure mais ne maîtrisera plus le processus de gestion des droits, qui ne sera plus sous sa responsabilité, mais qui sera opéré par le système central.

Les partenaires

En cas d'ouverture des Systèmes d'Information aux partenaires (clients, fournisseurs, etc.), ceux-ci sont associés à la définition des règles d'accès et des interfaces.

2.6.3. Les étapes standard

Le système de gestion des identités apporte le service de gestion centralisé et, en principe, ne doit pas modifier le fonctionnement des systèmes et/ou applications qui seront provisionnés. La phase d'étude de l'existant doit donc être la plus complète possible, pour permettre une intégration avec un impact minimal sur le Système d'Information en place.

L'analyse critique de l'existant devra prendre en compte les domaines suivants :

- analyse de l'organisation :
  - recensement des utilisateurs et identification des référentiels sources d'approvisionnement des données personnelles
  - identification des hiérarchies et de la réparation géographique
  - identification des mouvements
- analyse métier :
  - recensement des métiers et des populations concernées
  - identification des applications et des règles de gestion des droits
  - identification des profils existants et des processus de demandes, d'approbation et de déploiement
- analyse technique :
  - analyse du mode de gestion des droits dans les applications / systèmes
  - analyse de structure des données des annuaires et des bases de sécurité
  - analyse de l'infrastructure : réseau, systèmes, technologies exploitées (annuaires, SGBD, etc.)
- prise en compte des contraintes de sécurité, de qualité de service, de continuité

Les données collectées permettront de démarrer les phases suivantes de conception et de spécifications.

2.6.3.1. La conception fonctionnelle

- définition du modèle des données et d'habilitation :
  - règles d'identification de personnes
  - rôles (restrictions d'accès aux fonctions)
  - profils (regroupements et mise en cohérence des rôles)
  - périmètres (restrictions d'accès aux données)
  - etc.
- définition des règles d'alimentation à partir des référentiels
- définition des processus (workflow) de gestion du cycle de vie d'une identité et d'habilitation, tels que :
  - demandes et approbations des droits
  - création, suppression, suspension/révocation des comptes
  - gestion des approbateurs et des administrateurs
- définition des processus pour les rôles et des profils (définition, validations, évolutions)

- définition des processus d'audit et de reporting (génération de rapports)
- définition des règles de provisioning et de réconciliation des systèmes cibles
- spécification des interfaces de connexion aux systèmes cibles
- validation des aspects juridiques et réglementaires

2.6.3.2. La définition de l'architecture technique et le choix des outils

L'élaboration de l'architecture technique, dont un modèle type est présenté dans le prochain chapitre, doit prendre en compte, en particulier, les aspects d'intégration aux différents systèmes cibles de contrôle d'accès.

La solution technique de gestion d'identités peut être construite sur la base d'un développement mais il faut savoir que dans ce domaine il existe, sur ce segment du marché, plusieurs produits performants qui permettent de satisfaire pratiquement tous les besoins fonctionnels.

Le choix d'un tel outil passe par les étapes classiques d'expression des besoins, de rédaction du cahier des charges et d'évaluation des produits. Il est également vivement recommandé de valider ce choix par le prototypage (création d'un POC, pour Proof Of Concept). Les principaux produits sont arrivés à une certaine maturité, mais la diversité des schémas d'annuaires cibles et des processus de gestion impose cette étape pour valider l'exploitabilité du produit dans un contexte particulier. D'une manière générale, les éditeurs et/ou intégrateurs se plient de bonne grâce à ce type de demande.

2.6.3.3. La réalisation et la mise en service

La réalisation est pilotée selon la méthodologie standard des projets informatiques.
Lors de la mise en service, quelques phases critiques doivent faire l'objet d'une attention particulière :

- l'alimentation initiale de la solution par les identités en provenance des référentiels d'utilisateurs. La qualité des données d'entrée est primordiale pour le bon démarrage. Elle devra être validée d'une manière précise avec les «propriétaires» des référentiels sources.
- la réconciliation initiale, phase critique par excellence. Elle permet d'aligner la vision centrale de la cible élaborée par la solution de gestion d'identités avec les données déjà présentes dans chacune des cibles originelles. D'une manière systématique ce processus permet de constater de nombreux écarts et de les résoudre en appliquant les règles établies préalablement. Les erreurs dans ce traitement peuvent potentiellement perturber fortement la production. Ces processus doivent donc être répétés et validés

minutieusement, d'autant plus qu'ils devront s'exécuter sur une période la plus courte possible, de faible activité et de stabilité des informations incluses dans les annuaires.

- la conduite du changement. L'introduction d'une solution de gestion d'identités produit nécessairement des changements dans plusieurs modes opératoires et bouscule les habitudes des utilisateurs et/ou des administrateurs. C'est pourquoi, si l'on veut que le projet soit une complète réussite, elle devra être accompagnée d'une gestion de changement rigoureuse avec un support spécifique sur la période du déploiement.

2.6.3.4. Le maintien en condition opérationnelle (MCO)

Comme tout projet informatique, le système de gestion des identités et des droits d'accès doit, une fois son intégration terminée, être maintenu en condition opérationnelle. Pour ce faire, il est nécessaire de prendre en compte les évolutions des applications et des référentiels. De plus, l'outil mis en œuvre peut nécessiter d'éventuelles mises à jour fournies par l'éditeur. En particulier, les correctifs de sécurité seront systématiquement pris en compte.

2.6.4. Prise en compte d'un périmètre étendu

L'extension du périmètre de gestion des identités peut se traduire par différents cas d'ouverture du système vers l'extérieur :

- gestion d'identités des partenaires (entreprises clientes ou fournisseurs). Cette intégration peut s'effectuer selon deux approches :
    - référencement et intégration directe des utilisateurs désignés par le partenaire dans le système interne. Cette solution n'induit pas de changements majeurs dans le fonctionnement global. Les comptes des partenaires sont vus comme les comptes internes. Certaines procédures d'inscription, d'approbation et le nommage des comptes peuvent demander des adaptations. En contrepartie le partenaire est obligé de gérer ces utilisateurs dans son propre système et dans le système tiers. En effet, les procédures d'authentification et d'accès des deux systèmes restent totalement indépendantes
    - intégration de la gestion d'identités du partenaire via la fédération d'identités. Les principes de ce genre de solution ont été présentés dans les chapitres précédents. Sa mise en place demande des travaux d'adaptation d'architecture mais apporte la simplification des procédures de gestion et d'accès ainsi qu'une garantie de cohérence des informations de sécurité
- gestion d'identités des clients individuels. L'intégration de cette population est plus difficile. Souvent elle est prise en charge par des solutions spécifiques ou des systèmes

dédiés. Leur complexité se situe sur des plans différents. La gestion des utilisateurs internes concerne les populations moins nombreuses mais ayant l'accès à des systèmes multiples et des profils complexes tandis que la gestion des droits des clients individuels peut s'appliquer à des populations importantes mais avec des profils beaucoup plus simples ou portés par des systèmes intégrés. Cette démarche intéressera donc en premier lieu les entreprises / organismes ayant à gérer en direct des clients qui bénéficient d'une riche palette de services offerts par des architectures complexes et hétérogènes ainsi que les fournisseurs mettant en œuvre des services réalisés en collaboration avec plusieurs partenaires. Dans ce dernier cas de figure, ce sont les architectures du type « fédération d'identités » qui présentent le plus d'intérêt.

2.7. Aspects juridiques

L'aspect juridique de l'identité numérique [2] et celui de la gestion des identités nous amènent à présenter dans ce chapitre plusieurs textes importants, français ou internationaux, qui contribuent à justifier la mise en œuvre d'un système de gestion des identités. Le point commun à ces textes récents est l'exigence de traçabilité, exigence qui ne saurait être satisfaite sans une identification efficace des utilisateurs des Systèmes d'Information. Du fait que la gestion d'identités traite nécessairement certaines données nominatives, les textes relatifs à l'informatique et aux libertés revêtent également une importance particulière.

2.7.1. La loi Sarbanes-Oxley

Votée par le Congrès américain en juillet 2002 suite aux scandales ENRON et WORLDCOM, la loi Sarbanes-Oxley implique que les présidents des entreprises cotées aux États-Unis ou qui empruntent sur le marché des États-Unis et leurs filiales, y compris à l'étranger, certifient leurs comptes auprès de la Securities and Exchange Commission (SEC). Elle est guidée par trois grands principes :

- exactitude et accessibilité de l'information
- responsabilité des gestionnaires
- indépendance des auditeurs

La loi vise à augmenter la responsabilité des dirigeants et à mieux protéger les investisseurs. Elle a une portée extraterritoriale dans la mesure où les entreprises européennes doivent se soumettre à cette loi à partir du moment où elles sont cotées aux États-Unis. Les Systèmes d'Information sont impliqués :

- dans l'utilisation de l'informatique comme outil de gestion et de contrôle financier
- dans l'obligation instituée d'assurer la sécurité de ce même système

Le principal objectif est de renforcer le contrôle interne et de fournir des contrôles protégeant l'information contre toute utilisation, divulgation ou modification non autorisée, et contre tout dommage ou perte. Dans les Systèmes d'Information, cet objectif est atteint à l'aide de contrôle d'accès logique assurant l'accès aux systèmes, données et programmes aux seuls utilisateurs autorisés et à la traçabilité de ces accès. Cette activité de contrôle comporte 22 éléments différents, depuis les pare-feux jusqu'à la protection contre les virus et la réaction face à un incident en passant par la gestion, l'authentification et l'autorisation des utilisateurs.

2.7.2. La réforme Bâle 2

La réforme Bâle II du ratio de solvabilité bancaire s'inscrit dans une démarche mondiale de réglementation de la profession bancaire remontant à la fin des années 80, dont l'objectif premier est de prévenir les faillites.

Au delà de la dimension financière qui est le calcul des fonds propres à allouer, Bâle II prend en compte et place ses exigences sur les systèmes de notation et de surveillance. Bien plus, et c'est l'aspect le plus novateur, la réforme ne se limite plus aux seuls risques financiers « classiques », comme le risque de crédit ou les risques de marché (risque de change, risque de taux, etc.), mais couvre aussi le « Risque Opérationnel ».

Le Risque Opérationnel se définit comme le risque de pertes résultant de carences ou de défaillances attribuables à des procédures, personnels et systèmes internes ou à des événements extérieurs. Les Risques Opérationnels incluent notamment :

- les risques relatifs à la sécurité des biens et des personnes (incendie, inondation, tremblement de terre, attaque physique, sabotage, vol et fraude).
- les risques informatiques, liés aux développements et à la maintenance des programmes, aux traitements et à l'utilisation des services de télécommunications. Cette catégorie inclut en particulier le risque lié aux défauts de conception ou de réalisation d'une application, les incidents d'exploitation dans les systèmes de production, les accès non autorisés et les erreurs de traitement, ainsi que les pertes ou altérations accidentelles des données transmises et les défaillances dans la conservation de ces données.
- les risques de gestion interne, liés au fonctionnement interne de la banque, incluant les erreurs dans les traitements administratifs et comptables des opérations, les erreurs de conception ou de mise en place de nouveaux produits ou projets, la malveillance

interne, les risques légaux, réglementaires ou déontologiques, les risques en matière de ressources humaines, de sous-traitance et de communication externe.

Les accords Bâle 2 conduisent à renforcer les contrôles de processus et à garantir la transparence dans les opérations financières. La gestion rigoureuse des identités et l'efficacité des contrôles d'accès contribuent de manière importante à ces objectifs.

2.7.3. La réforme Solvency 2

La réforme Solvency 2 pour les compagnies d'assurance est comparable à la réforme Bâle 2 pour les établissements financiers. Son objectif est de fournir un cadre d'évaluation de la solvabilité de ces établissements afin d'optimiser leurs fonds propres, sur la base de leurs risques réels et de critères qualitatifs.

L'entrée en vigueur était initialement prévue pour 2010 – 2011 et devra finalement être appliquée au 31 octobre 2012 selon fr.wikipedia.org et www.ffsa.fr

2.7.4. La loi sur la sécurité financière - LSF

La loi n°2003-706 du 1er août 2003 de sécurité financière constitue la réponse du législateur français à la crise de confiance que connaissent les marchés financiers depuis le début des années 2000 suite aux scandales et aux faillites qui ont secoué les marchés.
Elle comporte des dispositions de nature financière mais également de nombreuses mesures intéressant les sociétés. Ces mesures tendent notamment à renforcer le contrôle des comptes et la transparence des entreprises.

La loi corrige ou adapte un certain nombre d'articles du code de commerce ou du code monétaire et financier.

2.7.5. Loi pour la confiance dans l'économie numérique – LEN ou LCEN

Cette loi du 21 juin 2004 a pour but de favoriser la confiance dans l'utilisation d'Internet en France. Elle aborde les principaux points suivants :

- la communication au public par voie électronique
- la responsabilisation et les contraintes sur les pratiques commerciales
- la réglementation envers les hébergeurs et les prestataires
- l'équilibre entre les droits de l'expression et la garantie des droits de la personne
- les dispositions relatives à la sphère publique et au développement des technologies de l'information et de la communication

2.7.6. Loi organique des lois de finance - LOLF

La loi organique des lois de finances définit une nouvelle architecture du budget de l'état permettant un meilleur contrôle de son utilisation. La LOLF prévoit pour chaque action, la définition d'objectifs et d'indicateurs de contrôle et que chaque responsable rende des compte sur ses résultats.

Cette approche nécessite une révision des méthodes de travail et de nouvelles procédures associées à des exigences de traçabilité des opérations.

2.7.7. La norme IAS 39 (International Accounting Standards)

En juillet 2002, un règlement européen a entériné la décision de la Commission Européenne d'imposer à toutes les sociétés européennes cotées (y compris les banques et les sociétés d'assurance) l'élaboration de leurs états financiers consolidés conformément aux normes comptables IAS.

En norme comptable IAS, l'information financière ne repose plus sur la notion de coût historique mais celle de la « juste valeur ». L'application de ces normes et particulièrement l'IAS 39 doit permettre :

- plus de transparence financière
- une comparabilité des états comptables
- une amélioration de la qualité de l'information plus économique

L'application de la norme IAS39 va avoir des conséquences fortes sur les systèmes informatiques, principalement sur la protection et l'intégrité des données.

2.7.8. CNIL

La loi « informatique et libertés » du 6 février 1978 a marqué une étape importante en France, et même en Europe, en réglementant la création et l'utilisation de fichiers contenant des données nominatives. Cette loi définit des obligations de protection et de déclaration des données personnelles et de leurs traitements à la CNIL, Commission Nationale de l'Informatique et des Libertés.

La loi de 1978 a été enrichie par la loi n° 2004-801 du 6 août 2004 relative à la protection des personnes physiques à l'égard des traitements de données à caractère personnel. Cette nouvelle loi attribue des pouvoirs de sanction à la CNIL.

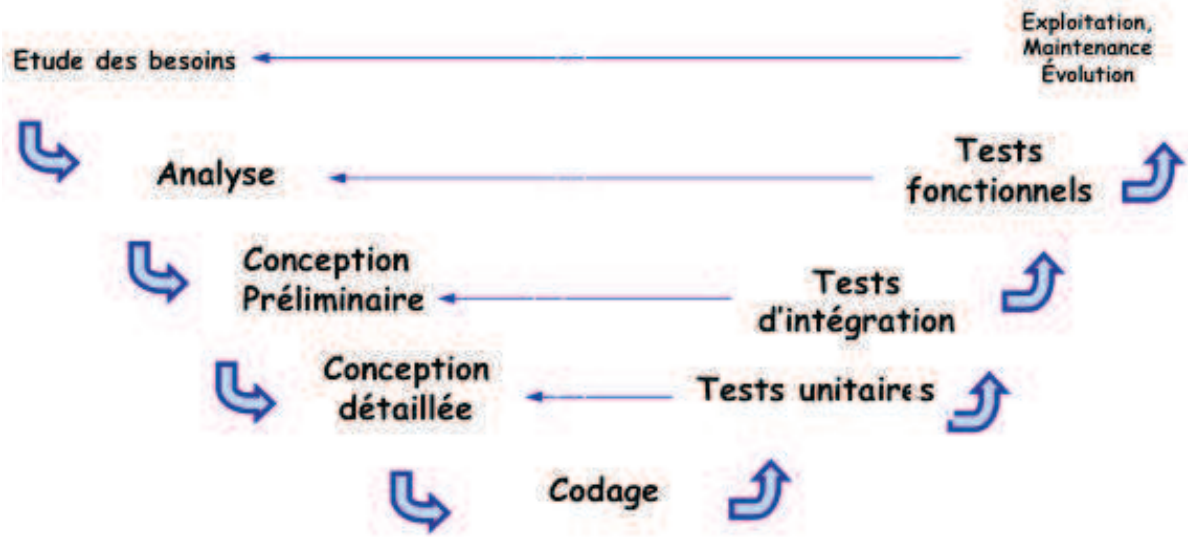

Figure 9 – Le cycle en V du développement logiciel

3. IdMe dans la gestion d'identité

3.1. Périmètre

IdMe reprend partiellement les éléments de l'introduction. En effet, il n'est pas possible en quelques mois d'intégrer la totalité des fonctionnalités du marché. J'ai pour cela choisi un périmètre restreint comportant des éléments parmi les suivants :

- gestion des utilisateurs
- gestion des cycles de vie
- gestion de l'alimentation aval des comptes utilisateurs (provisioning)
- gestion de la synchronisation de données utilisateurs
- synchronisation de mots de passe
- gestion du SSO (Single Sign On)
- gestion de l'authentification et de l'autorisation applicative

Les fonctionnalités sont détaillées dans l'expression des besoins.

3.2. Méthode utilisée

Pour ce développement, j'ai utilisé le cycle de développement en V (cf. figure 9). La moitié gauche de ce cycle a été par ailleurs réalisée à l'aide de la méthode Arrington [5] et du cours introductif du CNAM issue de java.cnam.fr, tout en s'inspirant de certaines idées évoquées dans un autre ouvrage de modélisation UML [6].

La méthode Arrington permet donc de découper en phase le recueil des besoins, leur interprétation, et leur implémentation. Trois documents ont donc été produits :

- Expression des besoins
- Analyse
- Conception

Les trois documents sont donc basés principalement sur UML. Par ailleurs, l'utilisation d'outils pour la modélisation UML conjointement aux ouvrages précités n'est pas suffisante. En effet, l'ouvrage sur la méthode Arrington [6] décrit la façon dont on doit modéliser autour du langage UML, mais ne décrit pas ce dernier, ni l'utilisation des différents diagrammes. Il faut donc s'appuyer sur un ouvrage de référence tel que celui

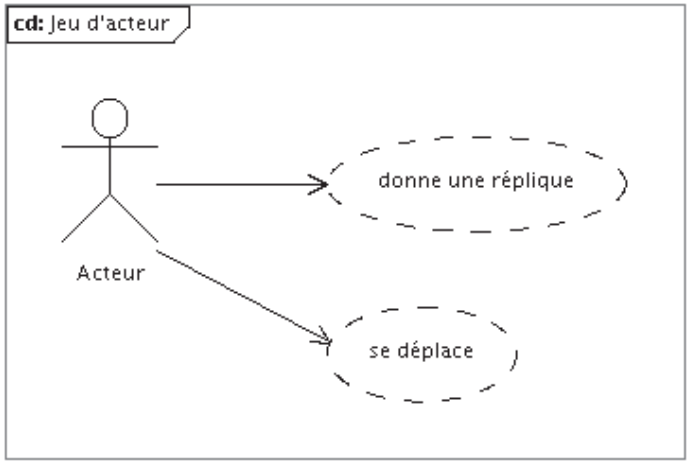

Figure 10 – Exemple de diagramme de cas d'utilisation UML

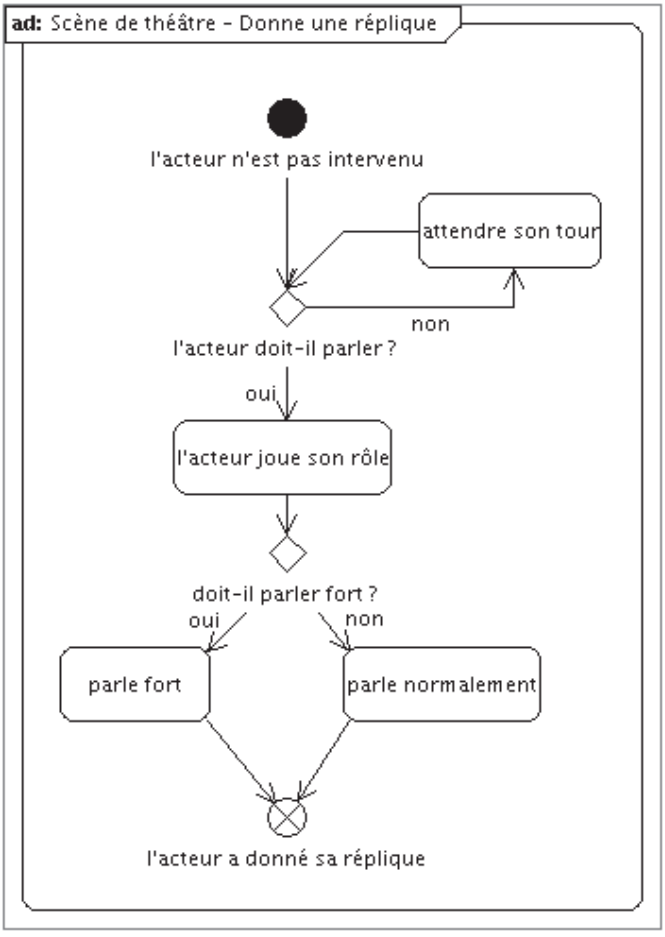

Figure 11 – Exemple de diagramme d'activité UML

sur la modélisation UML de P.A. Muller et N. Gaertner [7]. Ceci permet d'exprimer avec précision le sens de chaque interaction. Les différents types de diagrammes, leur utilité, les types de flèches, etc., y sont tous décrits. La figure 10 montre un diagramme de cas d'utilisation décrivant un acteur et les actions qu'il peut effectuer. La figure 11 quant à elle, décrit un diagramme d'activité détaillant un cas d'utilisation de cet acteur.

4. Expression des besoins

4.1. Document

L'expression des besoins est le document qui permet à la maîtrise d'ouvrage (MOA) de recueillir les besoins des clients et de les mettre en forme conjointement avec la maîtrise d'œuvre (MOE). Etant donné que je cumulais les fonctions de MOA et de MOE, il était assez délicat d'assumer les deux rôles en même temps. Ceci est comparable à une partie d'échec contre soi-même ou bien être juge et partie. Néanmoins, je me suis déjà trouvé professionnellement dans cette situation et cela ne m'a pas empêché de mener à bien ma mission.

L'expression des besoins, selon [5] et java.cnam.fr nous permet de :

- Définir les fonctionnalités du système du point de vue des utilisateurs
- Délimiter le système
  - ce qui est extérieur et qui communique avec le système
  - ce qui est interne au système
- Donner une description cohérente de toutes les vues que l'on peut avoir du système

Pour cela on effectue le processus suivant :

- Interviewer/écouter les clients
  - Experts du domaine
  - Utilisateurs finals
  - Production de notes de meeting
- Trouver les Acteurs
- Trouver les cas d'utilisation (Use Case)
- Spécifier/détailler chaque UC
- Faire valider par le client (presqu'automatique dans le cas présent)

On retrouve donc dans ce document, des diagrammes [6] principalement généraux et fonctionnels : diagramme de cas d'utilisation, diagramme d'activité et diagramme d'état.

4.2. Contexte

IdMe est un projet de gestion des identités. Il vise à réduire les coûts d'exploitation des bases utilisateurs, à rationaliser les accès au système d'information, et à fournir une piste d'audit. Il pourra par la suite servir de « colonne vertébrale » en fournissant un annuaire des accédants et en différentiant les types d'utilisateurs (cycles de vie). On pourra alors s'en servir pour contrôler les accès, voir dématérialiser les demandes d'habilitations par l'implémentation de workflow (non traité dans le projet actuel) qui piloteront l'outil précité.

Il existe 2 types de synchronisation :

- la centralisation de l'administration de données réparties sur des systèmes distants, via des rôles applicatifs. Cette première fonctionnalité déporte l'administration de certaines données d'un système distant vers IdMe. Par exemple, un administrateur d'annuaire pourrait arrêter d'administrer un ou plusieurs attributs donnant des droits sur une application, en délégant cette tâche à l'outil IdMe et à ses administrateurs via l'attribution de rôles applicatifs. Il s'agit donc d'un transfert de données entre IdMe et les systèmes distants, l'outil agissant comme un maître sur les systèmes distants.
- la synchronisation de données utilisateurs entre les systèmes distants à intervalles réguliers. Cette synchronisation pourra se faire directement au niveau des données en définissant des règles de correspondance entre une source et une cible. Les systèmes distants pourront notifier IdMe de changement de données. Cette seconde fonctionnalité est beaucoup moins critique, est destinée par exemple à la synchronisation d'adresses, de numéros de téléphone, et ne nécessite pas la génération de rapports spécialisés.

Les principaux thèmes de ce projet seront les suivants :

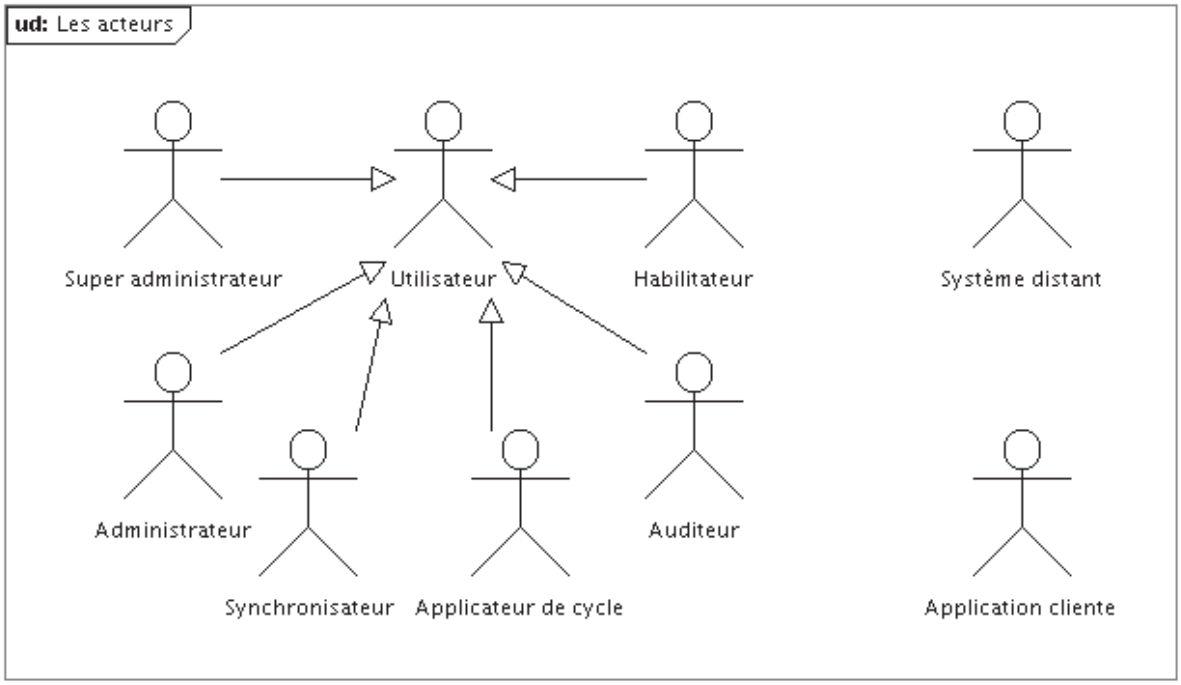

Figure 12 – Les acteurs de l'expression des besoins

- synchronisation des comptes entre bases prédéfinies au sein de l'outil : par exemple entre un annuaire LDAP et une base relationnelle via des correspondances prédéfinies
- création d'un compte dans la base de référence de l'outil : c'est ce compte qui nous permettra de gérer une fiche utilisateur de manière globale
- gestion du cycle utilisateur : automatisation des désactivations, des suppressions, des envois de mail de relance lorsqu'un compte arrive à échéance, d'actions de synchronisations (actions correspondantes sur les bases distantes).
- Gestion de règles de synchronisation : il s'agît de règles donnant les correspondances entre bases utilisateurs. Par exemple une règle A pourrait contenir un compte oracle, un compte Active Directory, et un compte Unix. Le fait d'affecter cette règle à un utilisateur par le biais d'un rôle ou d'un autre mécanisme créerait automatiquement les comptes associés. Ceci bien sûr à condition que des règles de nommages aient été définies à un moment donné.
- service d'authentification applicative et de SSO
- Il devra être possible de vérifier qu'un utilisateur a ou n'a pas un ou plusieurs comptes sur une ou plusieurs bases, en fonction des règles des rôles définies. Ceci devra pouvoir être effectué au moins une fois par jour.
- la cible UNIX n'est pas prioritaire et ne sera pas implémentée si le temps alloué n'est pas suffisant.
- Les cibles pour les systèmes distant seront LDAP (SUN au minimum), MySQL, et fichier plat.
- Les utilisateurs devront pouvoir mettre certaines données à jour via leur navigateur.
- Les mises à jour des bases distantes devront être prises en compte par l'outil.
- Dans un premier temps, on ne s'occupera pas des mécanismes d'authentification par certificat, token card, … Le périmètre n'ira donc pas au-delà du login et du mot de passe.

4.3. Les acteurs

Les acteurs (cf. figure 12) sont des entités externes au système et qui interagissent avec lui. Ils peuvent être des humains, des entités physiques, des sous-systèmes, etc.

4.3.1. L'utilisateur

Il se connecte au système d'information et utilise les comptes gérés par IdMe. Il peut aussi se connecter à IdMe pour mettre à jour un groupe de données définies par les administrateurs. Par exemple, le nom de la personne, son mot de passe, etc. Cet acteur regroupe tous les acteurs susceptibles de se loguer sur l'outil.

4.3.2. Le super administrateur

Cet utilisateur est particulier, il a tous les droits et ne peut pas être restreint dans son périmètre, ses droits son inaltérables. Son rôle est de pouvoir débloquer les situations particulières, ainsi que de créer des administrateurs. Il a une fiche de renseignement contenant le strict minimum et notamment un mot de passe. Il n'est pas inclus dans le fonctionnement des synchronisations (rôles, mot de passe, attributs, etc.) avec des systèmes distants. Il est créé automatiquement s'il n'existe pas. Il peut être utilisé comme administrateur mais ce n'est pas son but principal.

4.3.3. L'administrateur

Il s'agit d'un utilisateur privilégié nommé par le super administrateur ou par un autre administrateur. C'est lui qui fait vivre le référentiel des accédants en gérant les utilisateurs, les règles, les cycles de vie, etc.

4.3.4. Les systèmes distants

Il s'agît des bases de comptes distantes capables de notifier leur état (LDAP, SQL, fichier plat), et de recevoir des mises à jour du moteur.

4.3.5. Habilitateur

Il est responsable de l'attribution et du retrait des rôles applicatifs entre IdMe et les systèmes distants, de la vérification de leur cohérence avec les systèmes distants, ainsi que de la mise en conformité de ces derniers et de la génération de rapports. Ces habilitations pourront intervenir ponctuellement ou à intervalle régulier, pendant les heures de faible charge, la nuit idéalement.

4.3.6. Synchronisateur

Il s'occupe de la synchronisation des données autres que celles gérées par l'habilitateur, entre les systèmes distants et IdMe. Il gère aussi la synchronisation d'un

système distant à l'autre. Ces synchronisations pourront intervenir ponctuellement ou à intervalle régulier, pendant les heures de faible charge, la nuit idéalement.

4.3.7. Applicateur de cycle

Il est responsable de l'application des cycles de vie : activation d'un utilisateur à une date donnée, inactivation, suppression, envoie de mails d'alerte. Ces applications pourront intervenir ponctuellement ou à intervalle régulier, pendant les heures de faible charge, la nuit idéalement.

4.3.8. Auditeur

Il s'occupe de la vérification des habilitations et en a une vue complète.

4.3.9. Les applications distantes

Elles sont clientes des services d'authentifications et de SSO. Elles permettent à leurs utilisateur de bénéficier d'un service centralisé d'authentification, d'autorisation en évaluant les droits disponibles pour chaque utilisateur, et de SSO en s'authentifiant de manière unique pour toutes les applications inscrites.

4.4. Les cas d'utilisation

Les cas d'utilisation ont été regroupés en trois catégories :

- critiques : les cas qui devaient être implémentés obligatoirement
- importants : les cas qui doivent être implémentés en priorité, suite à l'implémentation des cas critiques
- utiles : les cas à implémenter s'il reste du temps

Au final, tous les cas ont été implémentés, en respectant l'ordre précité. Chacun d'entre eux est spécifié à l'aide d'un diagramme de cas d'utilisation regroupant les cas par thème, et parfois d'un diagramme d'activité et/ou d'état. Un cas est systématiquement décrit par une fiche donnant les informations suivantes :

- nom du cas
- résumé de la fonction du cas
- les acteurs impliqués dans le cas
- les pré conditions du cas : ce qui doit être présent, l'état d'un utilisateur par exemple (actif ou non)
- description détaillée du cas : son scénario

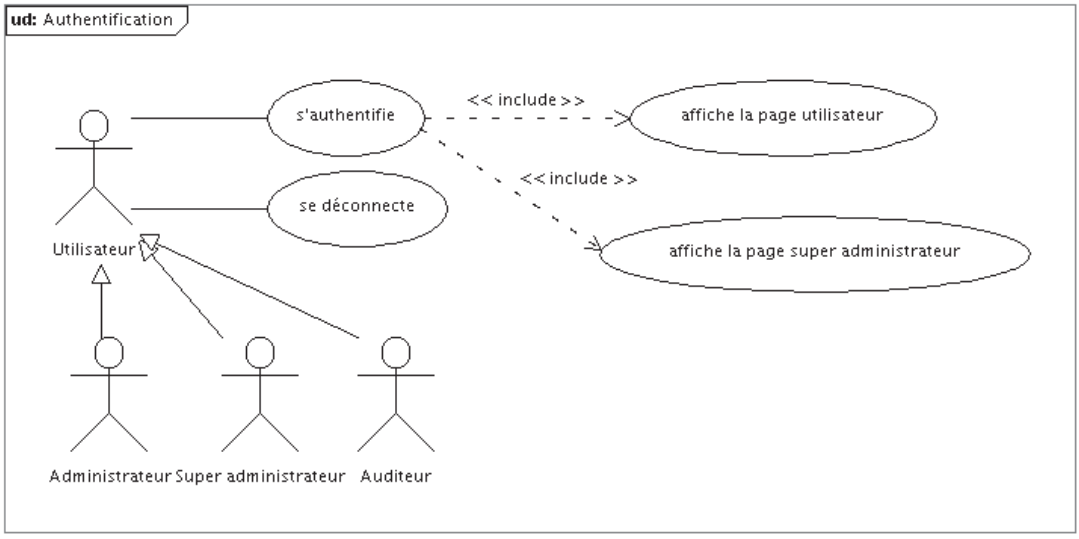

Figure 13 – Authentification d'un utilisateur

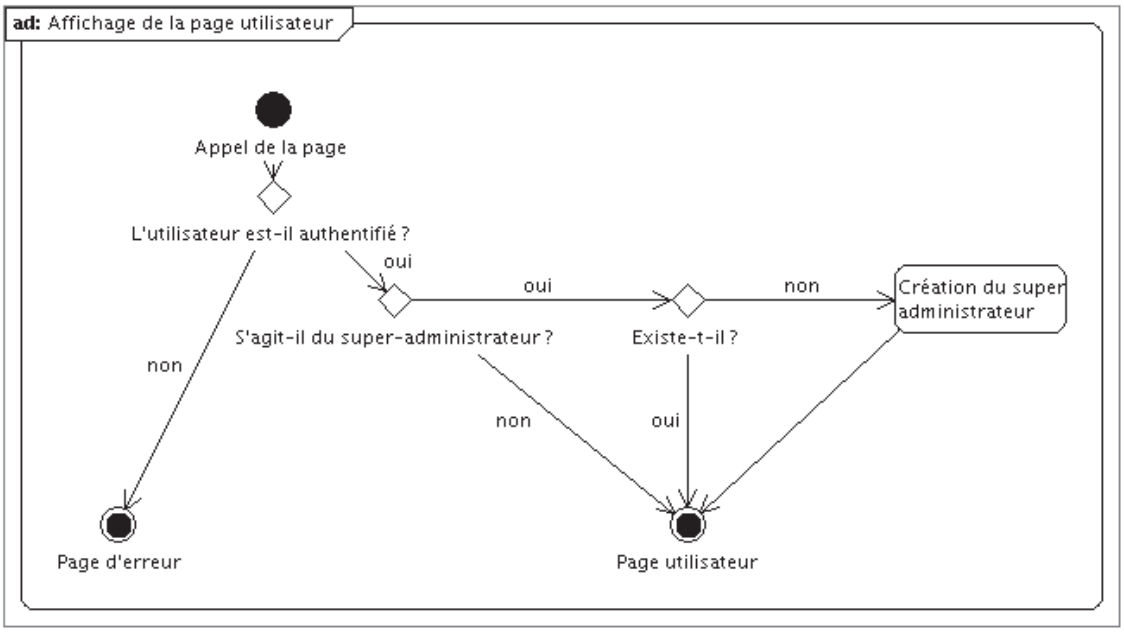

Figure 14 – Affichage de la page utilisateur

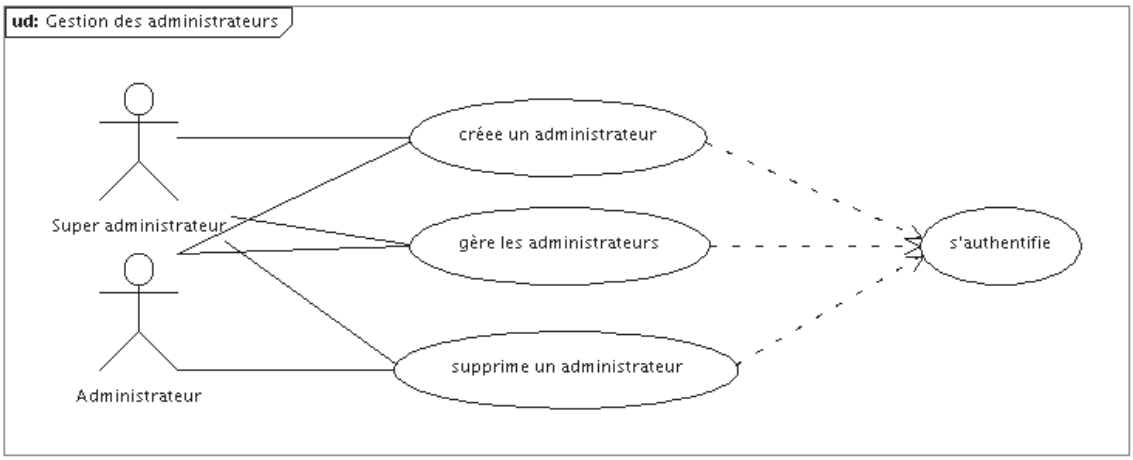

Figure 15 – Gestion des administrateurs

- exceptions : en cas d'erreur que doit-on faire ?
- post conditions : l'état après l'exécution du cas
- remarques : pour ajouter certains commentaires relatifs à des cas particuliers

Par soucis de clarté, nous ne présenterons ici que le nom et description de chacun des cas.

4.4.1. Cas critiques

4.4.1.1. Authentification

L'authentification correspond à l'identification d'un utilisateur auprès d'IdMe. Elle regroupe plusieurs cas d'utilisation (cf. figure 13) :

- **s'authentifie** : il décrit le scénario d'un acteur arrivant sur une page d'accueil en fournissant son identifiant et son mot de passe. Si l'identifiant et le mot de passe sont corrects, le client est connecté au système. Dans le cas contraire on gère l'exception.
- **se déconnecte** : l'acteur clique sur le bouton de déconnection, sa session est détruite, il retourne sur la page d'accueil
- **affiche la page utilisateur** : la page de l'utilisateur est affichée avec ses informations, ainsi que les actions disponibles en fonction de ses droits.

Ce dernier cas d'utilisation prend aussi en charge le fait que le super administrateur soit créé automatiquement (cf. figure 14).

4.4.1.2. Gestion des administrateurs

La gestion des administrateurs (cf. figure 15) fait partie des opérations de configuration et permet de nommer ou révoquer des administrateurs. Ces opérations sont toutes tracées. Les cas suivants décrivent la gestion de ces administrateurs :

- **crée un administrateur** : l'utilisateur connecté sélectionne un utilisateur parmi la population d'utilisateurs et le nomme administrateur.
- **gère les administrateurs** : l'utilisateur connecté sélectionne un utilisateur. Il valide son action et une confirmation est affichée.
- **supprime un administrateur** : l'utilisateur connecté sélectionne un administrateur et valide son action. Une confirmation de suppression est ensuite affichée.

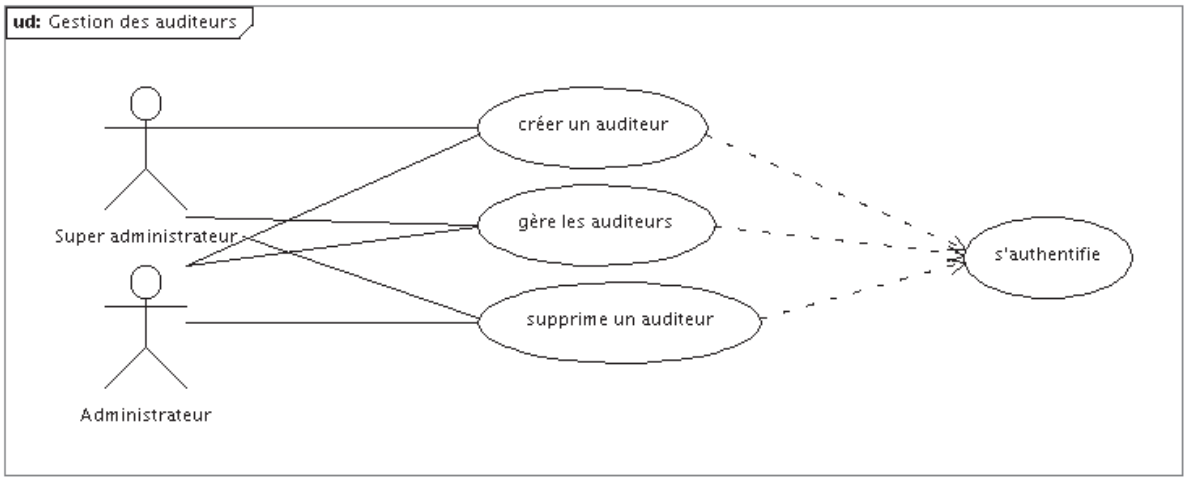

Figure 16 – Gestion des auditeurs

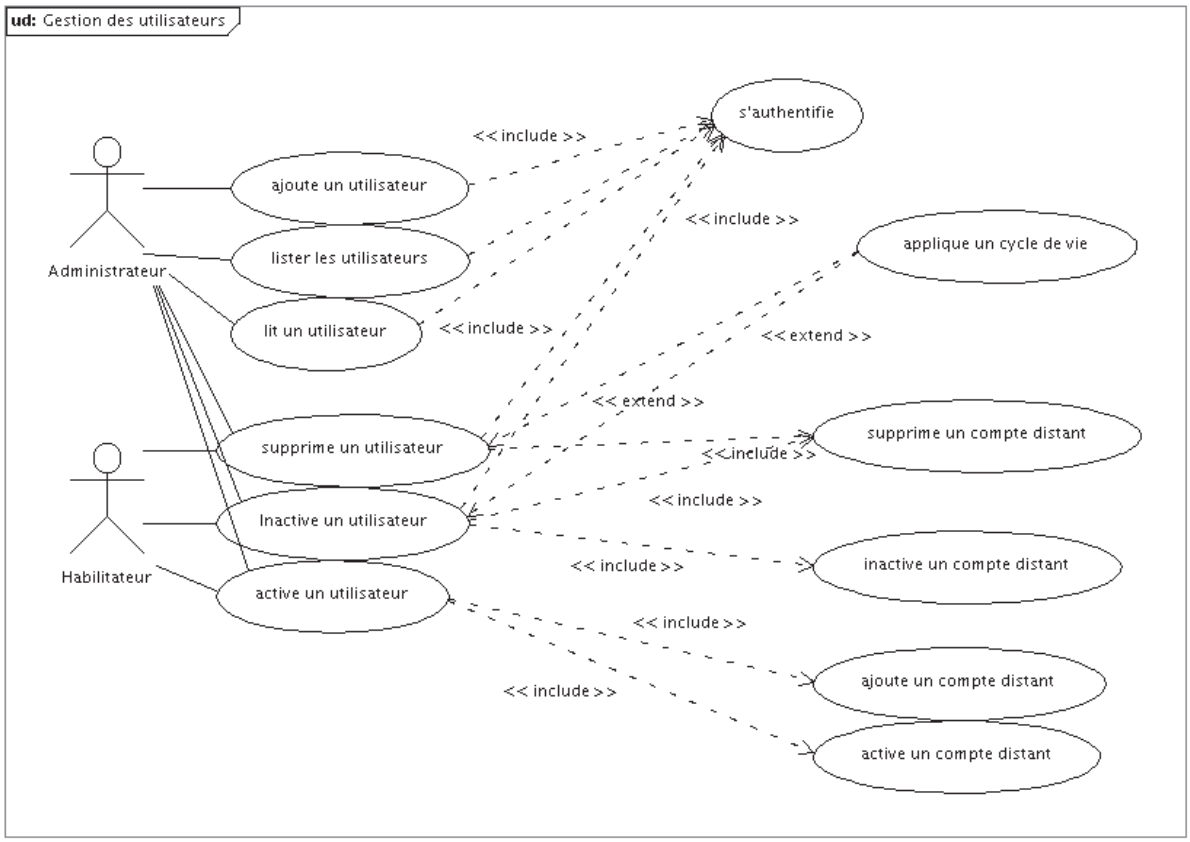

Figure 17 – Gestion des utilisateurs

4.4.1.3. Gestion des auditeurs

La gestion des auditeurs (cf. figure 16) fait partie des opérations de configuration et permet de nommer ou révoquer des auditeurs. Ces opérations sont tracées. Les cas d'utilisations sont pratiquement identiques à ceux des administrateurs et ne seront donc pas détaillés.

4.4.1.4. Gestion des utilisateurs

La gestion des utilisateurs représente les actions qui peuvent être effectuées sur les utilisateurs (cf. figure 17). On y retrouve la création, la suppression, ainsi que les changements d'état tel que inactivation et activation. Certains cas incluent d'autres cas indépendants qui détaillent des sous-parties du produit. Par exemple, on notera que les cas « inactive un utilisateur » et « active un utilisateur » incluent les cas « supprime un compte distant » ou encore « active un compte distant ». Ceci signifie donc que l'exécution du cas « active un utilisateur » pourra engendrer une activation ou une création de compte, mais en aucun cas ceci ne sera obligatoire. En effet, si un compte existe déjà, il est inutile de le créer, il en va de même pour l'activation. On retrouve donc les cas suivants au sein de la gestion des utilisateurs :

- **ajoute un utilisateur** : l'administrateur sélectionne l'action de création. Il remplit une fiche de création contenant les renseignements sur l'utilisateur. Le login est soit entré manuellement par l'administrateur. Il valide ensuite sa saisie. Si une des valeurs saisies n'est pas correcte, l'administrateur est prévenu, sinon l'utilisateur est créé et un message de confirmation est affiché. L'action est loguée si c'est indiqué dans la configuration des logs.
- **lit un utilisateur** : l'administrateur sélectionne l'action de listing des utilisateurs. Il clique sur l'utilisateur recherché. La page de l'utilisateur recherché est alors affichée.
- **lister les utilisateurs** : l'administrateur sélectionne l'action de listing des utilisateurs. La liste de tous les utilisateurs est affichée. Elle pourra être organisée par pages de résultat.

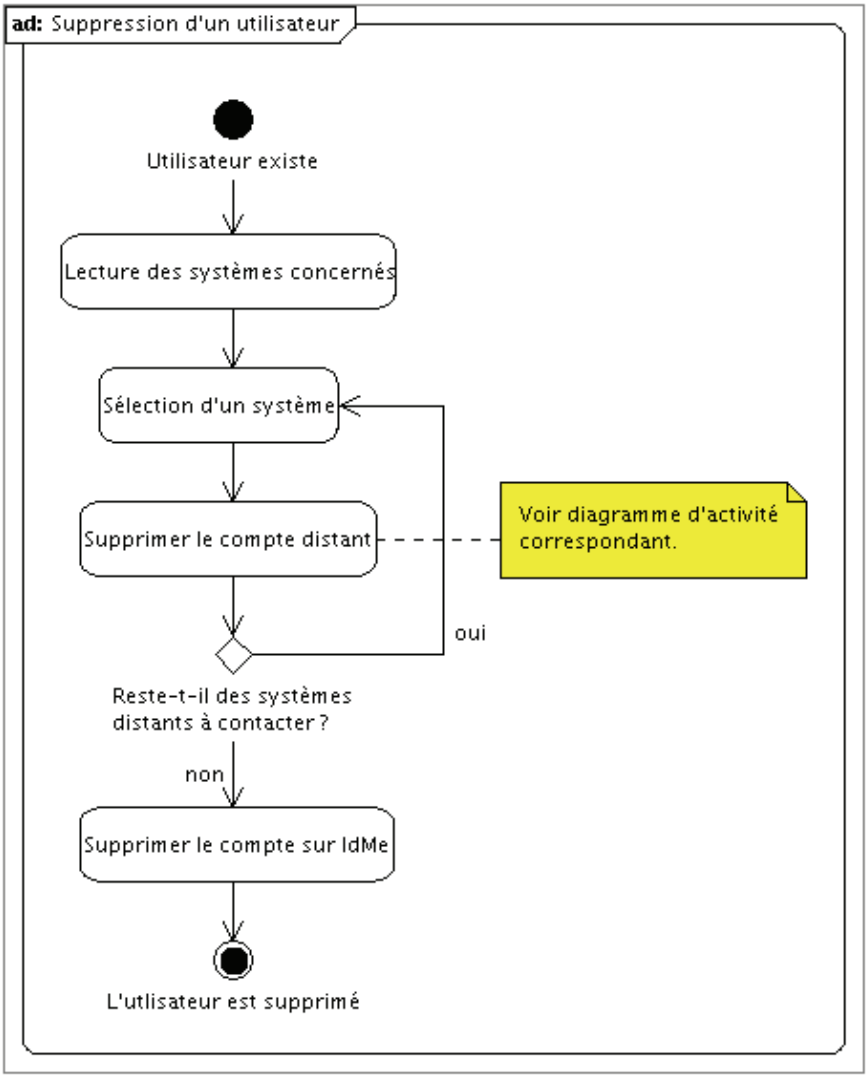

Figure 18 – Suppression d'un utilisateur

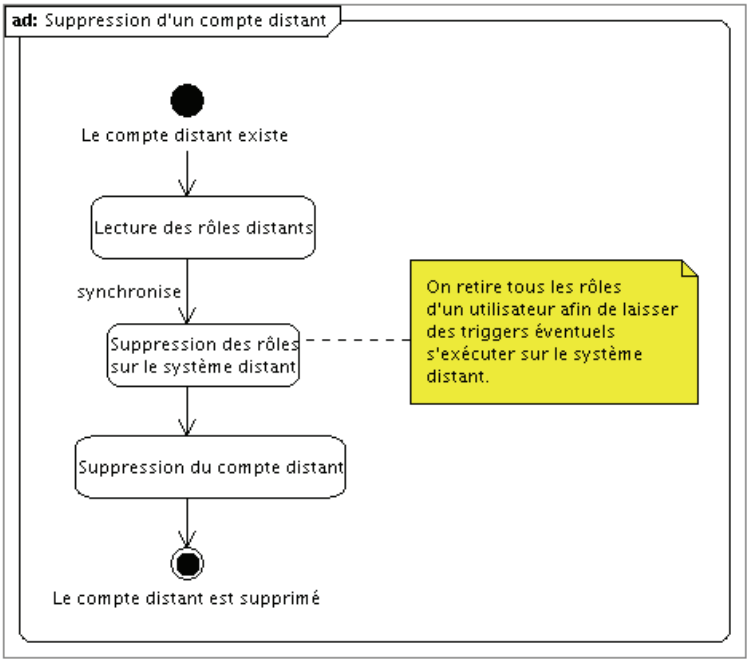

Figure 19 - Suppression d'un compte distant

- **supprime un utilisateur** (cf. figure 18) : l'administrateur ou l'habilitateur sélectionne l'action de suppression ainsi que l'utilisateur. Une demande de confirmation est ensuite affichée. Si l'administrateur valide, l'utilisateur est supprimé et tous ses rôles son retirés (cf. UC retirer un rôle). Un message confirmant le bon déroulement s'affiche ensuite. L'action est loguée si c'est indiqué dans la configuration des logs. Par ailleurs, l'action est manuelle s'il s'agît d'un administrateur. Elle est automatique s'il s'agît d'IdMe en étant déclenchée par l'application d'un cycle de vie (cf UC applique un cycle de vie).
- **ajoute un compte distant** : cette action est indirecte et résulte de l'action de l'ajout, cas « ajoute un utilisateur ». Un compte est créé sur le système distant en prenant en considération les données de configuration du moteur et les données de l'utilisateur.
- **supprime un compte distant** (cf. figure 19) : cette action est indirecte et résulte de l'action de suppression UC « supprime un utilisateur ». Un compte du système distant correspondant à un compte utilisateur d'IdMe est donc supprimé en prenant en compte l'identifiant de l'utilisateur.
- **active un compte distant** : cette action est indirecte et résulte de l'action d'ajout, cf. cas « active un utilisateur ». Un compte du système distant correspondant à un compte utilisateur d'IdMe est activé en prenant en compte l'identifiant de l'utilisateur. Ce compte devra au préalable exister, sinon l'action n'aura aucun effet.
- **inactive un compte distant** : cette action est indirecte et résulte de l'action de suppression, cas « inactive un utilisateur ». Un compte du système distant correspondant à un compte utilisateur d'IdMe est inactivé en prenant en compte l'identifiant de l'utilisateur. Ce compte devra au préalable exister, sinon l'action n'aura aucun effet.

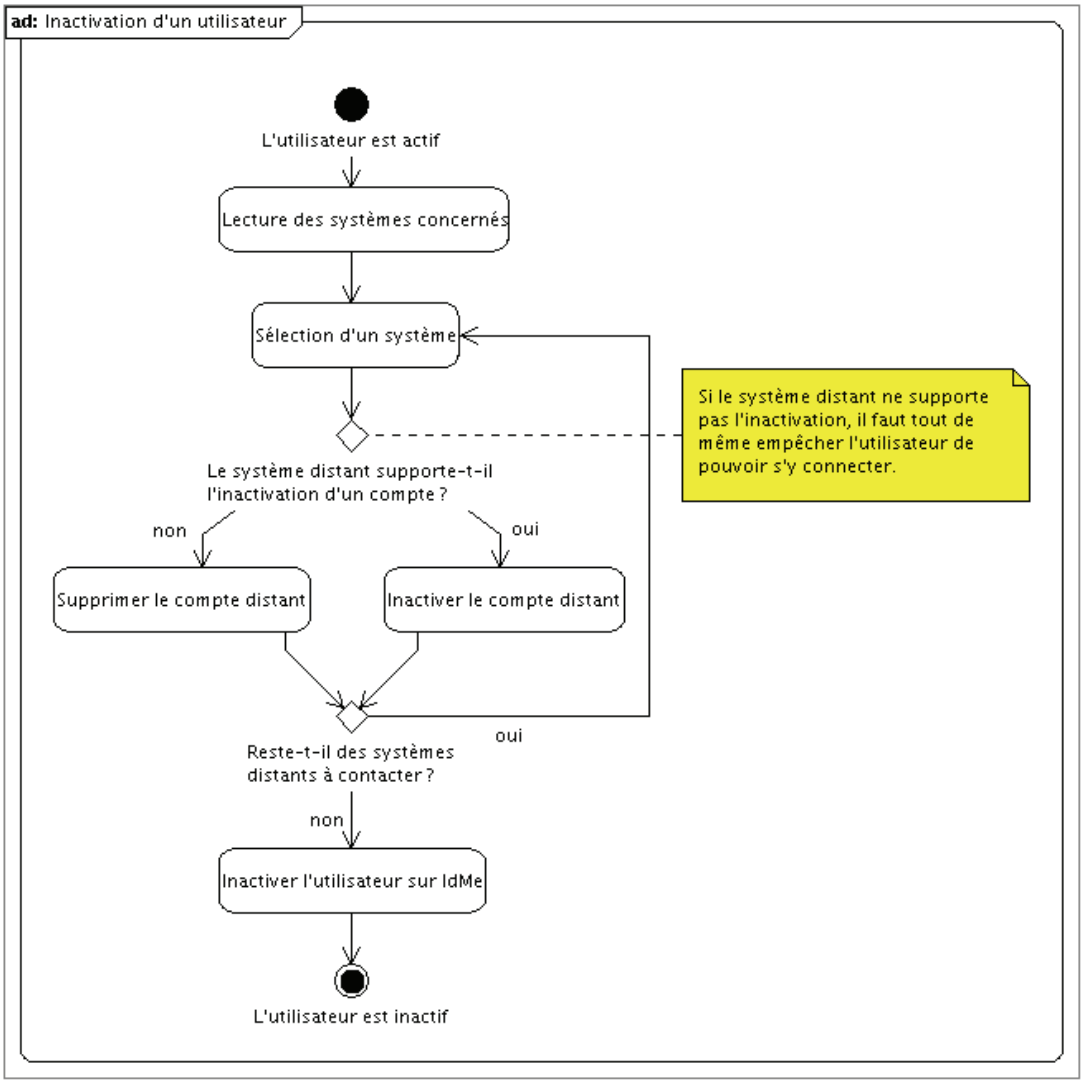

Figure 20 - Inactivation d'un utilisateur

- **inactive un utilisateur** (cf. figure 20) : l'administrateur ou l'habilitateur sélectionne l'action d'inactivation/activation et l'utilisateur. Une demande de confirmation est ensuite affichée indiquant le nouvel état de l'utilisateur. Si l'administrateur valide, l'utilisateur est supprimé et un message confirmant le bon déroulement s'affiche. L'action est loguée si c'est indiqué dans la configuration des logs.

Un utilisateur peut donc être dans plusieurs états. Il débute sa vie en tant que brouillon, état dans lequel on ne peut lui attribuer des droits. Sa suppression n'est pas tracée et peut être effectuée manuellement.

Une fois que l'utilisateur quitte son statut de brouillon, il devient actif pour la première. A partir de ce moment toutes les actions importantes sont tracées :

- activation/inactivation
- attribution/retrait de droits
- suppression/création (elle précède la première activation)

Lorsque l'utilisateur devient actif pour la première fois, les droits qui lui ont été attribués sont propagés sur les systèmes distants concernés, et l'attribution des droits est persistée dans la trace d'audit.

Il est important de noter qu'un utilisateur ne peut être activé avant la date de début renseignée dans sa fiche, et que si personne ne s'en charge, IdMe le fera automatiquement.

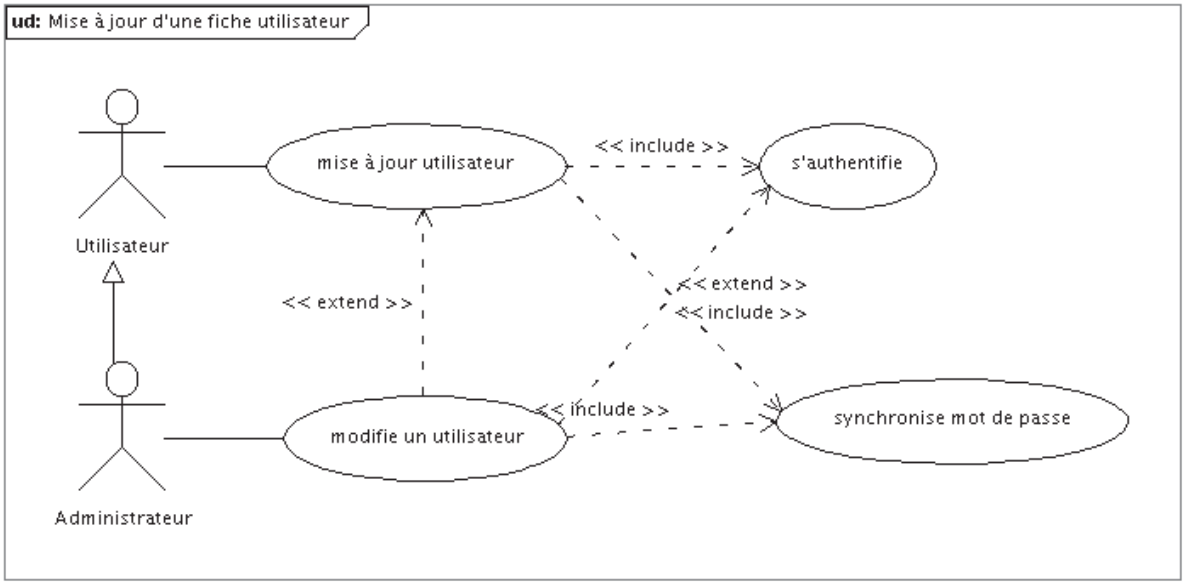

Figure 21 – Mise à jour d'un utilisateur

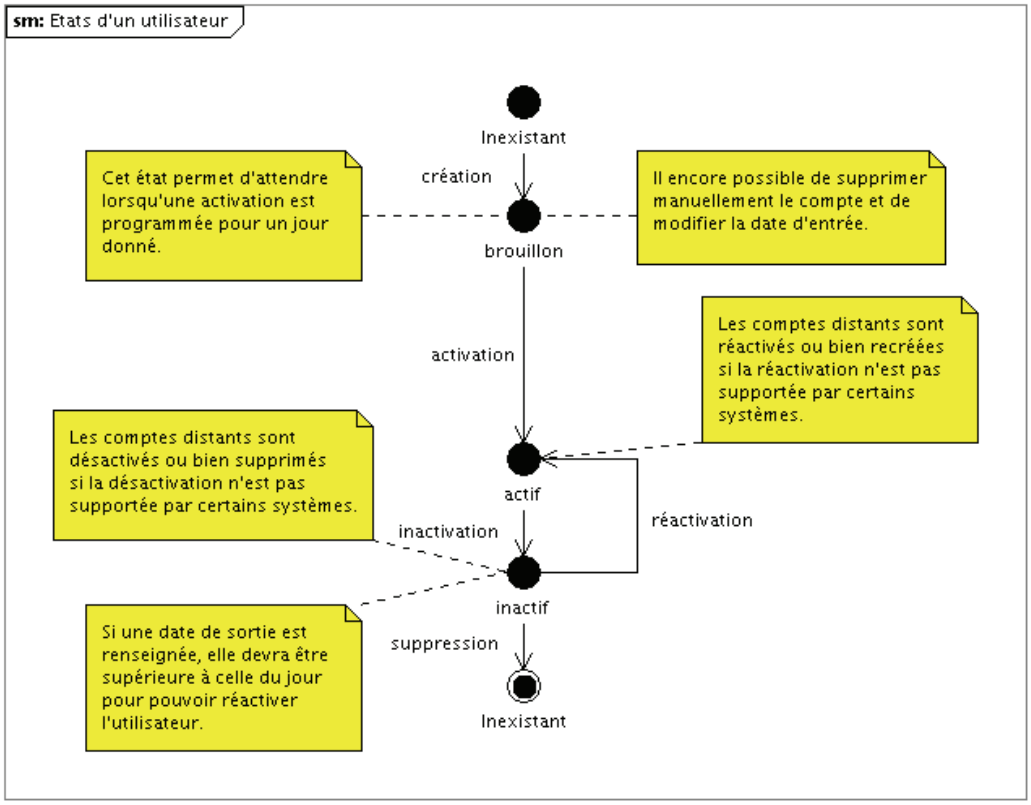

Figure 22 – Etats d'un utilisateur

4.4.1.5. Mise à jour d'une fiche utilisateur

Les cas d'utilisations relatifs à la mise à jour d'un utilisateur sont regroupés dans cette partie (cf. figure 21). Il est question de deux cas : la mise à jour utilisateur orienté utilisateur final, et la modification utilisateur orienté administrateur, tous les deux pouvant engendrer des changements d'états (cf. figure 22).

- **mise à jour utilisateur** : l'utilisateur une fois connecté arrive sur sa fiche personnelle. Il peut ensuite mettre à jour ses données personnelles y compris son mot de passe. Mais pas les autres informations de sécurité telles que les droits, ceci est réservé aux administrateurs (cf. cas Modifie un utilisateur). Il modifie la fiche contenant ses renseignements. Le login n'est pas modifiable. Si une des valeurs saisies n'est pas correcte, l'utilisateur est prévenu, sinon la mise à jour est effectuée et un message de confirmation est affiché. L'action est loguée si c'est indiqué dans la configuration des logs.

- **modifie un utilisateur** : l'administrateur sélectionne l'action de modification et l'utilisateur. Il modifie la fiche contenant les renseignements sur l'utilisateur. Le login n'est pas modifiable. Si une des valeurs saisies n'est pas correcte, l'administrateur est prévenu, sinon l'utilisateur est créé et un message de confirmation est affiché. L'action est loguée si c'est indiqué dans la configuration des logs. Ce cas étend le cas « mise à jour utilisateur » car il permet aussi bien de modifier les données personnelles de l'utilisateur que ses données de sécurité.

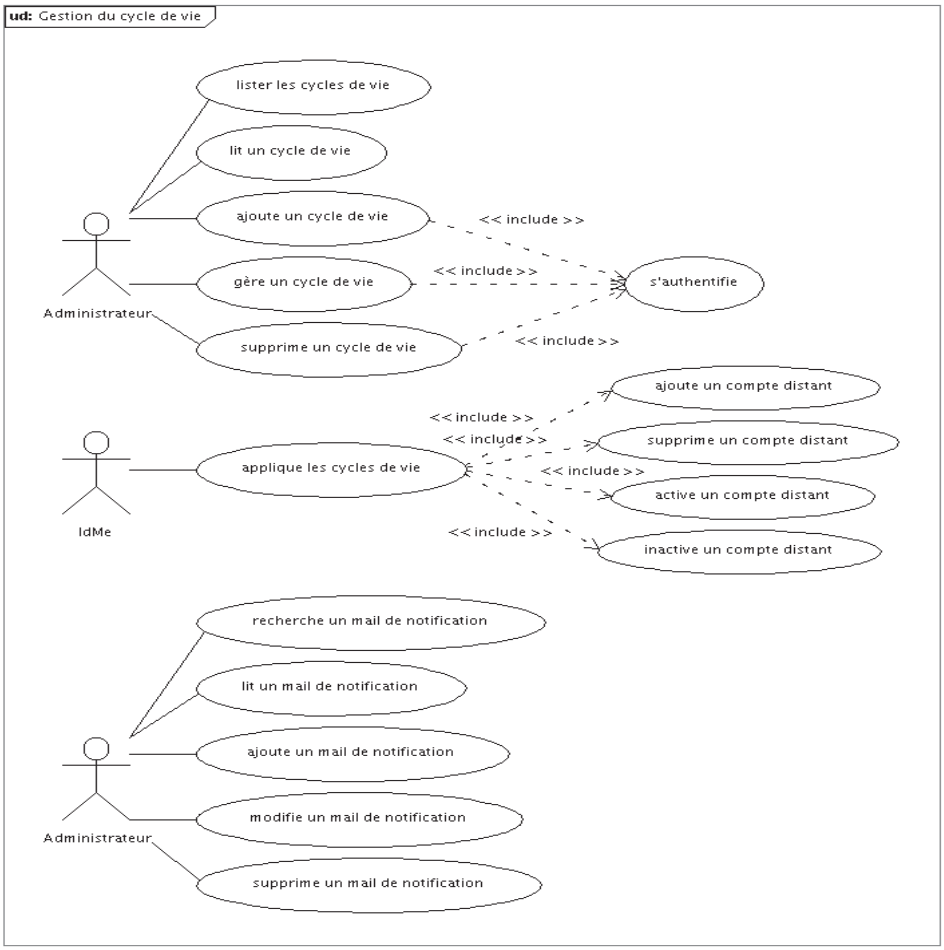

Figure 23 – Gestion du cycle de vie

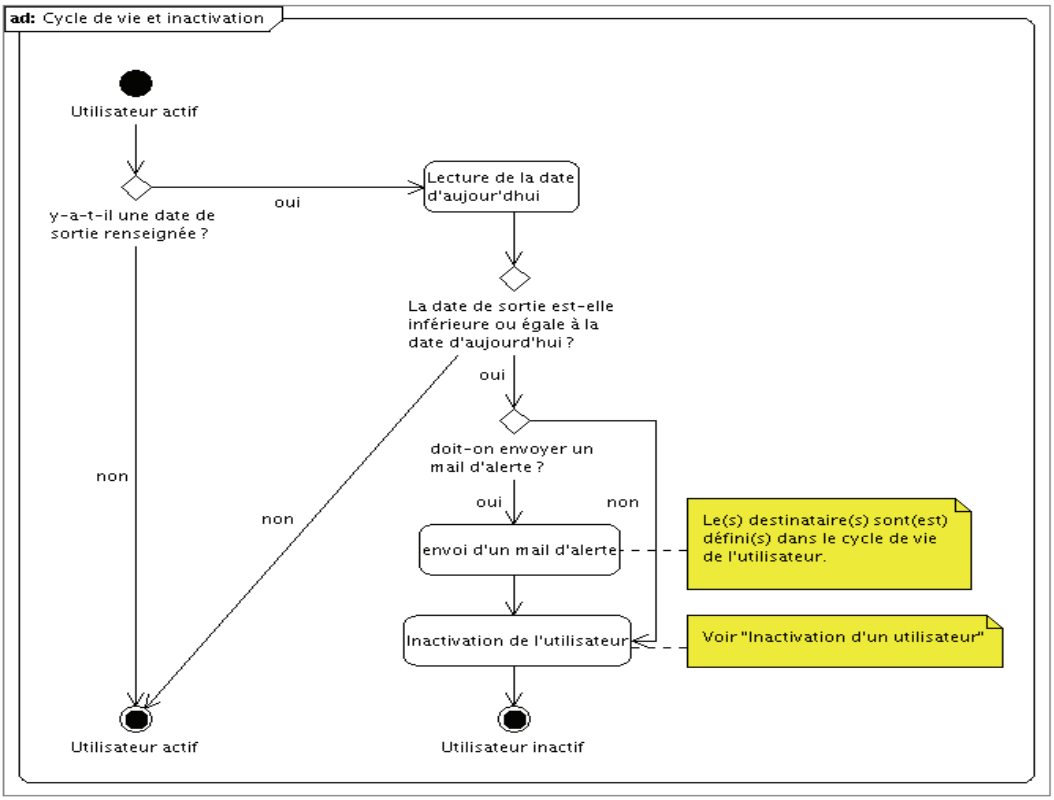

Figure 24 – Cycle de vie et inactivation

4.4.1.6. Gestion du cycle de vie

Un cycle de vie (cf. figure 23) est un ensemble d'options accompagnant les changements d'état des utilisateurs (cf. figure 22).

Un cycle de vie propose plusieurs fonctionnalités mais surtout des notifications par mail :

- rendre obligatoire la date de fin d'un utilisateur (cas utile pour les personnes temporaires)
- déterminer la nécessité de la suppression d'un compte utilisateur
- activation, inactivation et suppression automatique d'un utilisateur en fonction de sa date de sortie et par conséquent de son cycle de vie
- notification avant et/ou pendant l'inactivation d'un utilisateur (cf. figure 24)
- notification avant et/ou pendant la suppression d'un utilisateur

Les notifications sont très utiles en entreprise car elles permettent d'agir uniquement si cela est nécessaire, et non pas à devoir chaque jour vérifier la date de validité des dates de sortie des utilisateurs.

Les notifications sont envoyées sous forme d'e-mail et sont personnalisables. Certaines valeurs des utilisateurs concernés seront disponibles sous formes de variables à insérer dans le sujet et dans le corps du mail.

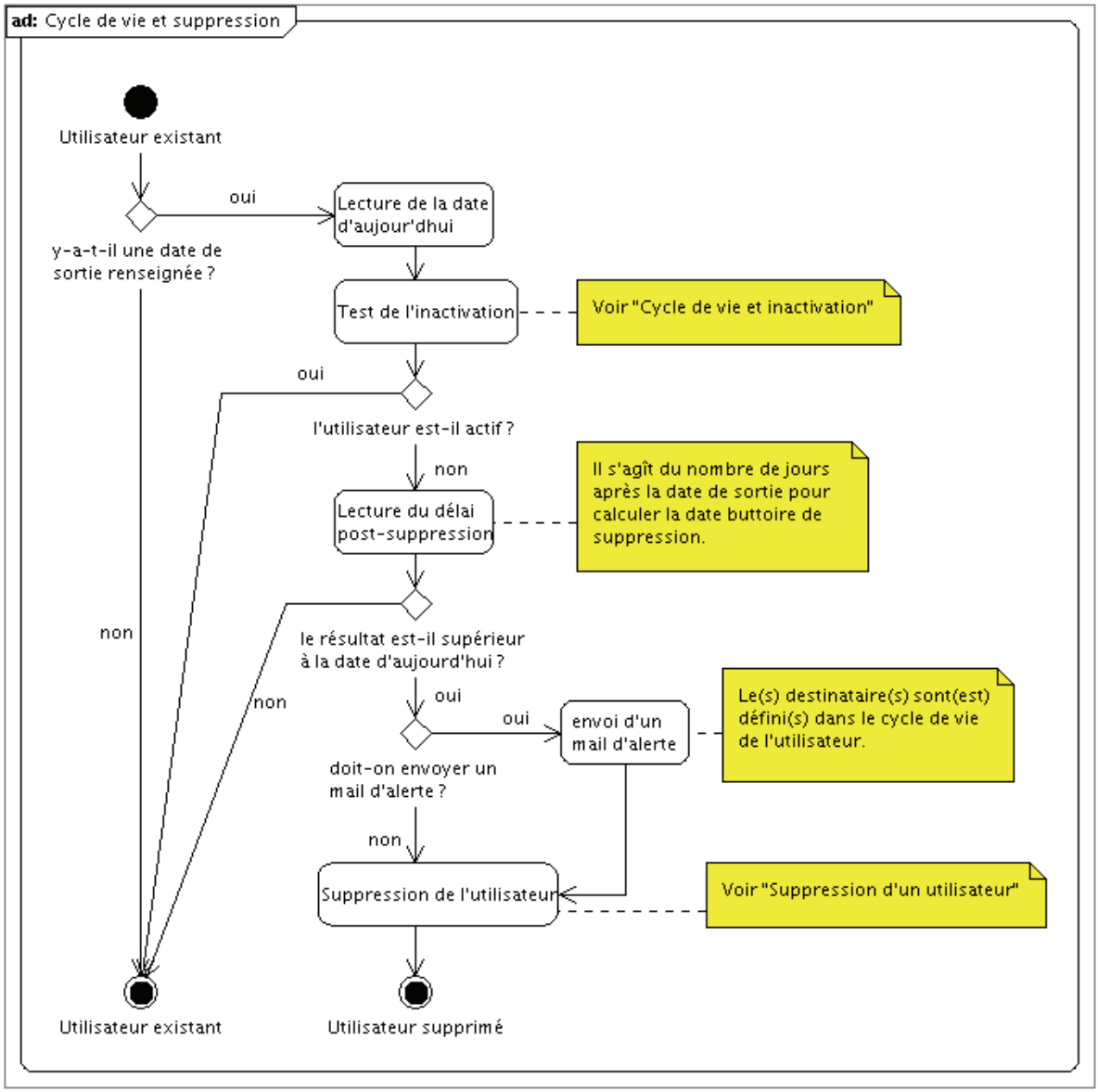

Figure 25 – Cycle de vie et suppression

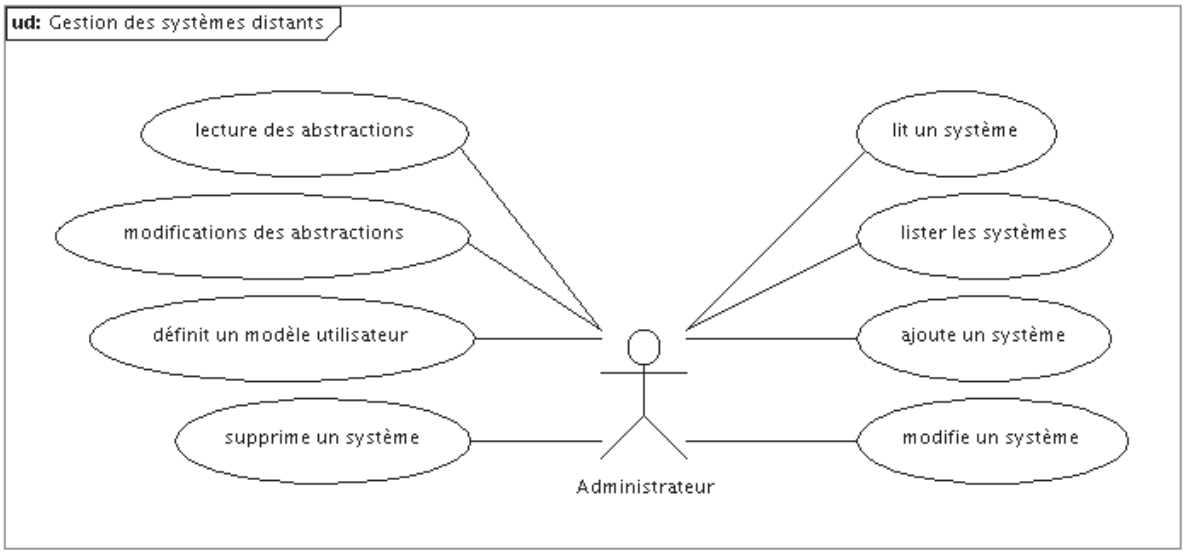

Figure 26 – Gestion des systèmes distants

Par exemple : Monsieur %NOMCOMPLET% sera inactif à la date du %DATEFIN% sera traité par le gestionnaire de cycle de vie et envoyé au destinataire sous la forme :

Monsieur Rodolf Dupont sera inactif à la date du 31/07/2011.

Les variables disponibles qui concerneront l'utilisateur ciblé seront les suivantes : l'identifiant, le prénom, le nom, le nom complet, l'adresse e-mail, le numéro de téléphone, le numéro de téléphone portable, la date de fin, la date de suppression.

La gestion des cycles de vie inclut donc les cas d'utilisation courants : ajouter, supprimer, modifier, lister, et lire un cycle de vie. Elle inclue aussi les cas des mails de notification.

Le cas d'application d'un cycle de vie correspond à une action en concordance avec les fonctionnalités précédemment citées. Elle peut être d'origine manuelle ou automatique et inclut lorsque c'est nécessaire (activation/inactivation, suppression) les cas d'ajout/suppression ou d'inactivation de compte distant. Par exemple, l'inactivation d'un utilisateur demande au système distant d'inactiver le compte, si ce dernier ne supporte pas l'inactivation il faudra retirer les droits distants. De plus, si l'option de gestion exclusive des droits est activée, le retrait de tous les droits distants engendrera la suppression du compte associé et une notification sera envoyée (cf. figure 25).

4.4.1.7. Gestion des systèmes distants

La gestion des systèmes distants (cf. figure 26) représente les outils qui permettent la communication entre IdMe et le système cible.

Un système cible est une base d'utilisateur sous la forme de base de donnée MySQL, d'un annuaire, ou d'un fichier plat. Il est définit par les informations permettant de communiquer avec lui :

- une adresse IP
- un port TCP
- un identifiant
- un mot de passe
- etc.

On associe alors à un système donné à des abstractions. Une abstraction définit une donnée du système distant. Par exemple, dans le cas d'un annuaire, il peut s'agir d'un attribut. L'abstraction fait alors le lien entre la donnée distante et une donnée

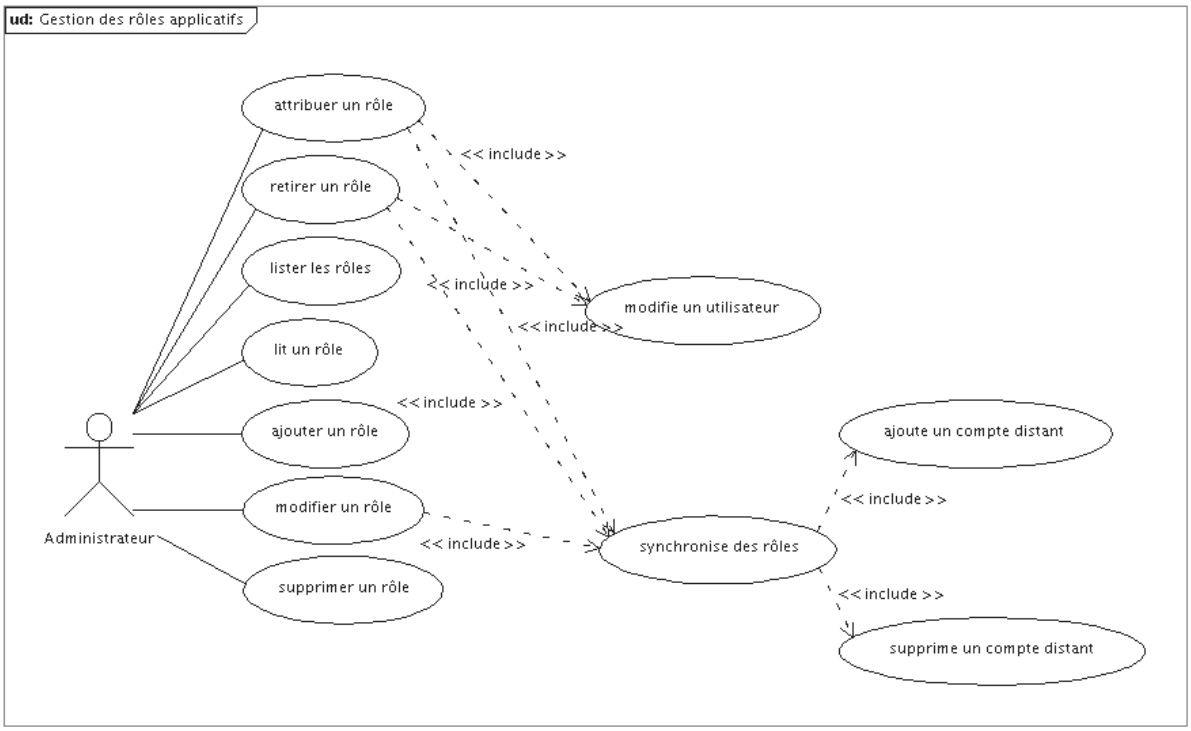

Figure 27 – Gestion des rôles applicatifs

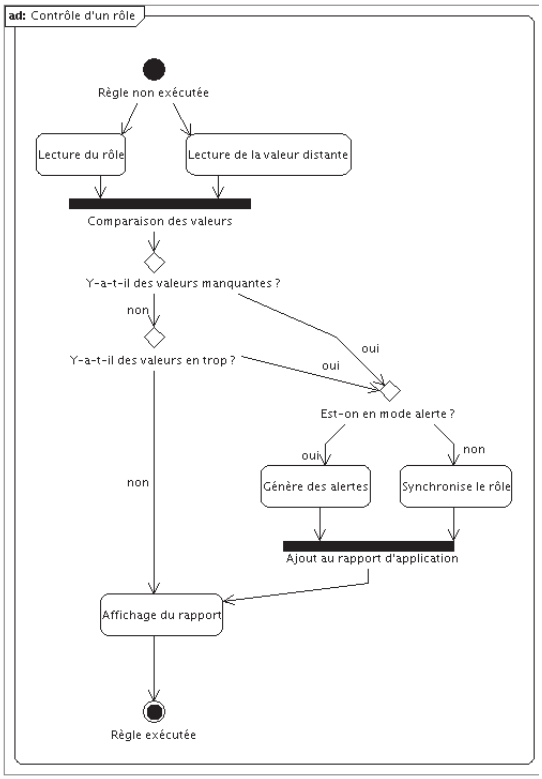

Figure 28 – Contrôle d'un rôle

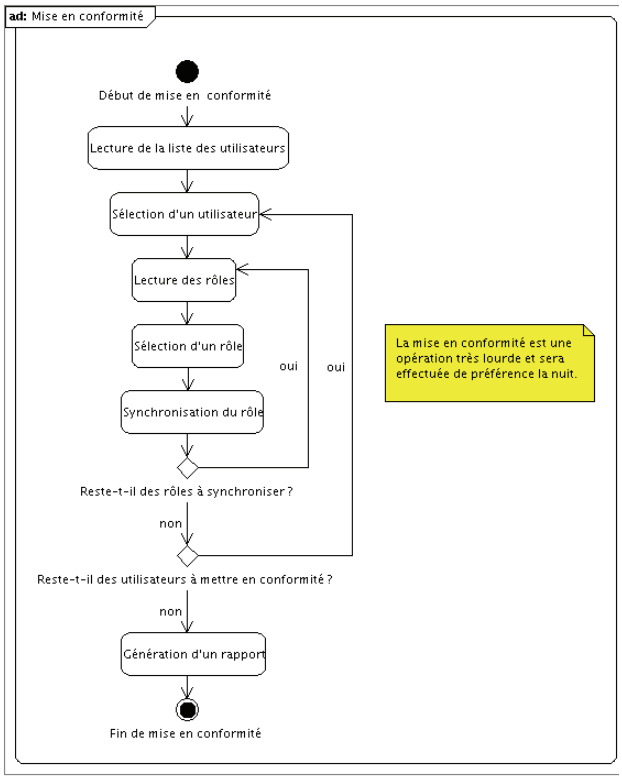

Figure 29 – Mise en conformité

intelligible par IdMe. On attribuera un nom logique qui simplifiera la manipulation des données et identifiera de manière unique l'abstraction, tout système distant confondu. Par exemple givenName sur un annuaire pourrait devenir firstName ou prénom.

Enfin pour la création d'une entrée utilisateur distante il est nécessaire d'en connaître le format. C'est le rôle du modèle utilisateur, qui définit la liste des abstractions nécessaires et la correspondance avec les attributs d'un utilisateur d'IdMe.

4.4.1.8. Gestion des rôles applicatifs

Les rôles applicatifs (cf. figure 27) sont rattachés à un utilisateur et leur attribution ou retrait est tracé dans la trace d'audit. Ils peuvent avoir plusieurs sens :

- **informatif** : on vérifie manuellement dans l'interface que la personne a bien ce rôle. Par exemple, la personne est responsable d'un groupe d'échange sur la gestion d'identité, on pourrait utiliser un rôle pour l'indiquer.
- **alimentation aval de comptes (provisioning)** : on associe une abstraction à un rôle ainsi qu'une ou plusieurs valeur(s). L'affectation du rôle à l'utilisateur aura pour effet de propager la valeur renseignée dans le rôle sur le système distant correspondant, pour l'attribut spécifié.
- **applicatif** : il est aussi affecté à une application. Il est utilisé par les applications clientes pour restreindre les droits des utilisateurs.

Il est possible de procéder au contrôle des rôles (cf. figure 28) affectés à un utilisateur sur les systèmes distants. Ceci permet de voir si les rôles distants sont en concordance avec ceux indiqués dans IdMe. Il est parfois possible qu'un administrateur, par exemple de base de donnée, ne soit pas au courant que l'administration d'une table est désormais centralisée dans IdMe. Il modifie les droits de l'utilisateur dans la base et les droits ne sont plus en accord avec ceux inscrits dans IdMe.

Le contrôle de conformité (cf. figure 29) est précisément utile dans ce cas. Il permet de recenser les droits utilisateurs distants et de vérifier leur cohérence. Un rapport de conformité est alors généré.

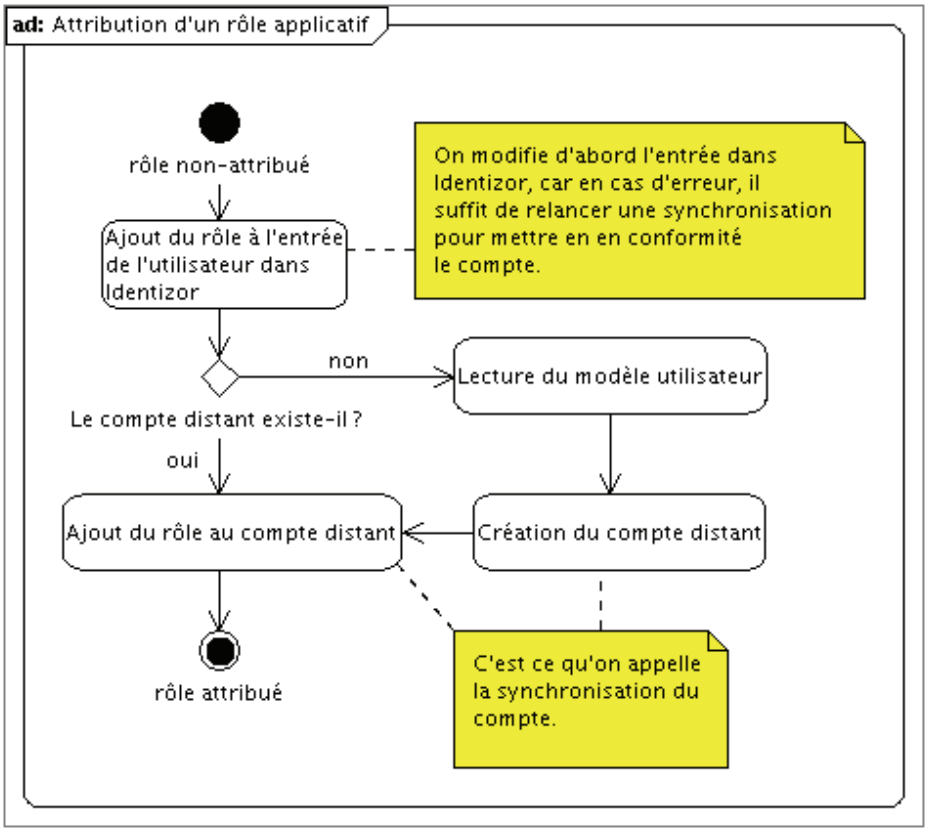

Figure 30 – Attribution d'un rôle applicatif

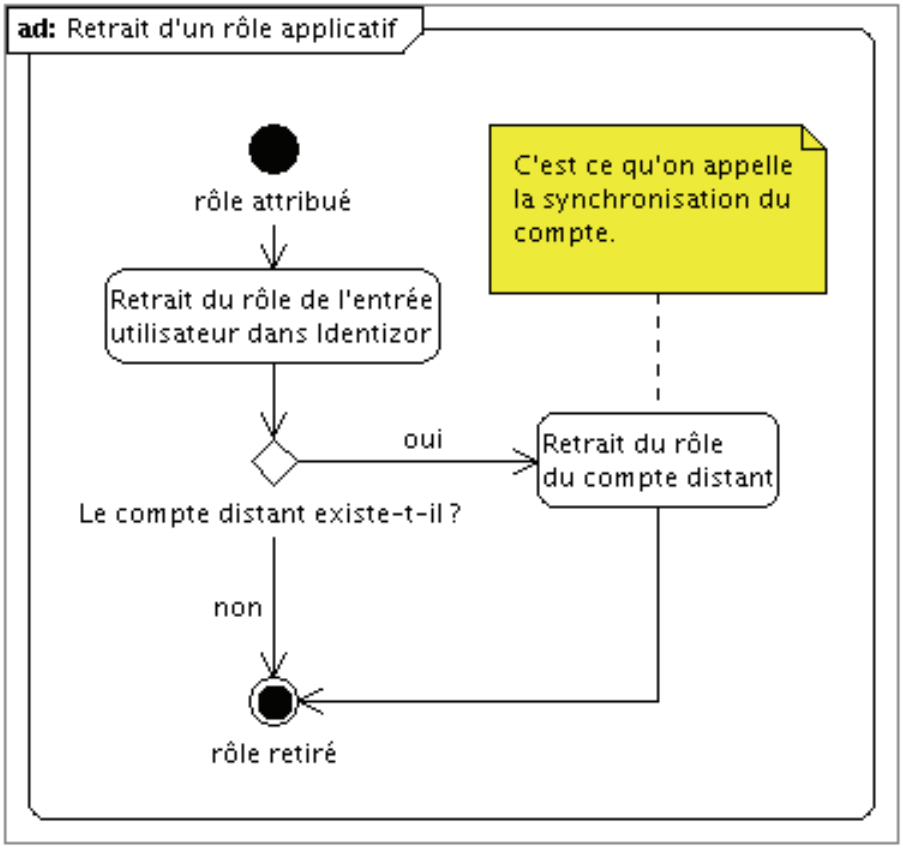

Figure 31 – Retrait d'un rôle applicatif

Les principaux cas d'utilisation sont détaillés ci-dessous :

- **attribuer un rôle** (cf. figure 30) : l'administrateur attribue le rôle à une fiche utilisateur et valide sa saisie. Une opération de synchronisation est automatiquement lancée (cf. UC « synchronise un rôle ») afin de propager l'information sur le système distant. Un message de confirmation est alors affiché.
- **retirer un rôle** (cf. figure 31) : l'administrateur enlève le rôle d'une fiche utilisateur et valide sa saisie. Une opération de synchronisation identique à celle de l'attribution d'un rôle est automatiquement lancée (cf. UC « synchronise un rôle »). Un message de confirmation est ensuite affiché.
- **contrôler un rôle** (cf. figure 28) : le contrôle peut s'effectuer (acteur Habilitateur) la nuit à une heure déterminée, de manière manuelle. L'administrateur sélectionne un rôle, l'action « contrôler » et le mode de contrôle (mise à jour automatique ou alerte). Il valide sa saisie. Une fois le contrôle terminé, un rapport est affiché. Ce contrôle permet de vérifier que les valeurs des systèmes distants sont en accord avec celles définies dans le rôle.
  Le cas de la mise à jour automatique est traité dans l'UC « synchronise un rôle ».
- **synchronise un rôle** : les systèmes distants sont mis à jour en fonction des informations du rôle affecté à l'utilisateur. S'il manque des valeurs elles sont ajoutées, s'il y en a en trop et qu'elles sont indésirables, elles sont supprimées.
- **ajoute un compte distant** : le gestionnaire sélectionne l'utilisateur et effectue l'action définie dans son cycle de vie.
- **supprime un compte distant** : le gestionnaire sélectionne l'utilisateur et effectue l'action définie dans son cycle de vie. Ce cas peut aussi être déclenché lorsqu'un utilisateur ayant un compte distant n'a plus de droits associés suite au retrait d'un rôle. C'est donc ici une action indirecte qui découle du retrait d'un rôle.
- **met en conformité** : on synchronise les rôles de tous les utilisateurs d'IdMe avec les comptes distants. Cette opération garantie la concordance entre les informations contenues dans IdMe et celles des systèmes distants. Elle sera réalisée de préférence la nuit.

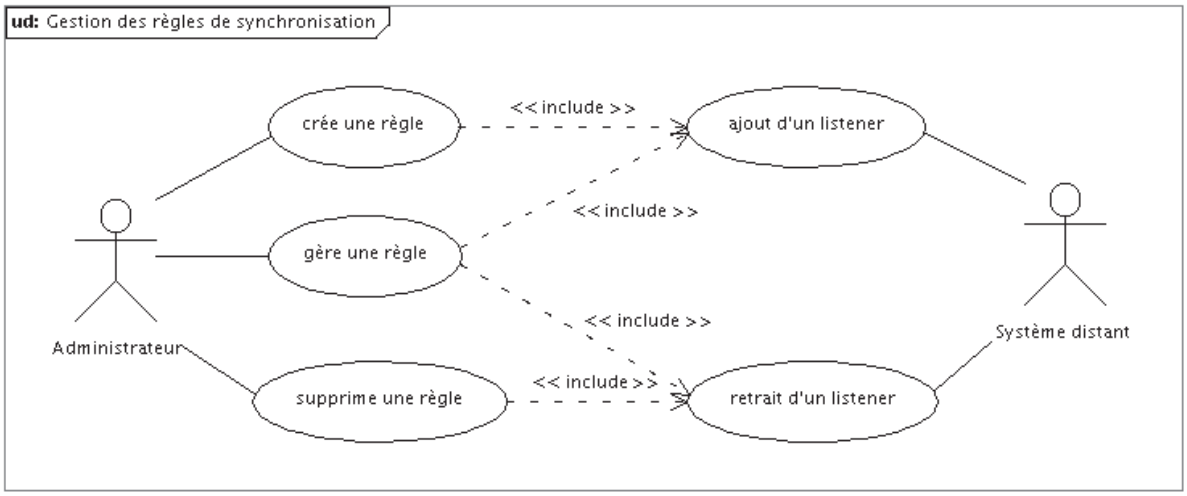

Figure 32 – Gestion des règles de synchronisation

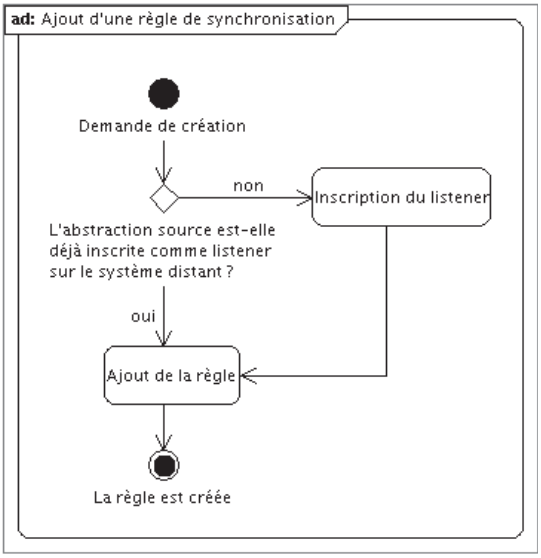

Figure 33 – Ajout d'une règle de synchronisation

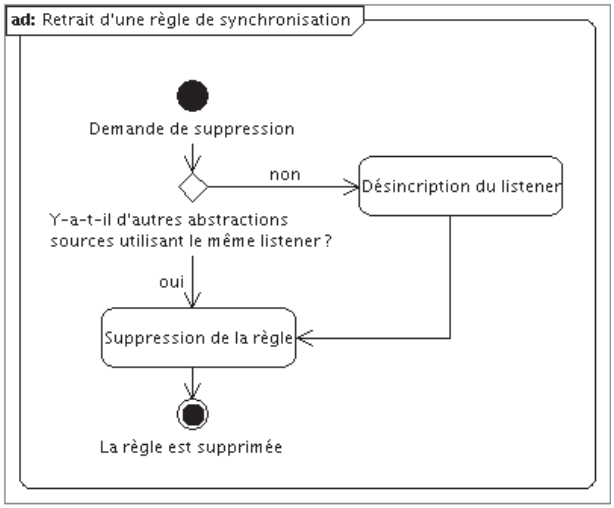

Figure 34 – Retrait d'une règle de synchronisation

4.4.2. Cas importants

Il s'agît des cas représentant des fonctionnalités majeures d'IdMe, mais non primordiales.

4.4.2.1. Gestion et application des règles de synchronisation

Les règles de synchronisation (cf. figure 32) mettent en œuvre la transmission de données entre 2 systèmes distants ou entre un système distant et le moteur. On distingue donc :

- la transmission des données dont le contenu est géré centralement (via IdMe) et poussé vers des systèmes satellites (principe du provisioning)
- la transmission systématique de données à chaque modification. Bien entendu, seules les données souscrivant à ce service seront surveillées et transmises.

Une règle de synchronisation contient donc :

- un système source
- une abstraction source
- un système destination
- une abstraction destination
- un libellé ou un code pour pouvoir l'identifier de manière unique
- un état (active ou non)

La notion de synchronisation et de surveillance des données implique la notion de « listener » (voir cas d'attribution et de retrait d'un rôle applicatif). Un listener est une abstraction d'un système source (système distant ou IdMe) pour laquelle l'outil va surveiller les modifications de tous les utilisateurs. Ainsi, pour une règle donnée surveillant les numéros de téléphone des utilisateurs d'un système, chaque modification sera évaluée et transmises aux systèmes destination (cf. figure 33 et 34) via le synchronisateur.

Les cas d'utilisations des gestions des règles de synchronisation ne seront pas détaillés étant donné qu'il sont similaires aux autres cas de manipulation : créer une règle, gère une règle, supprime une règle.

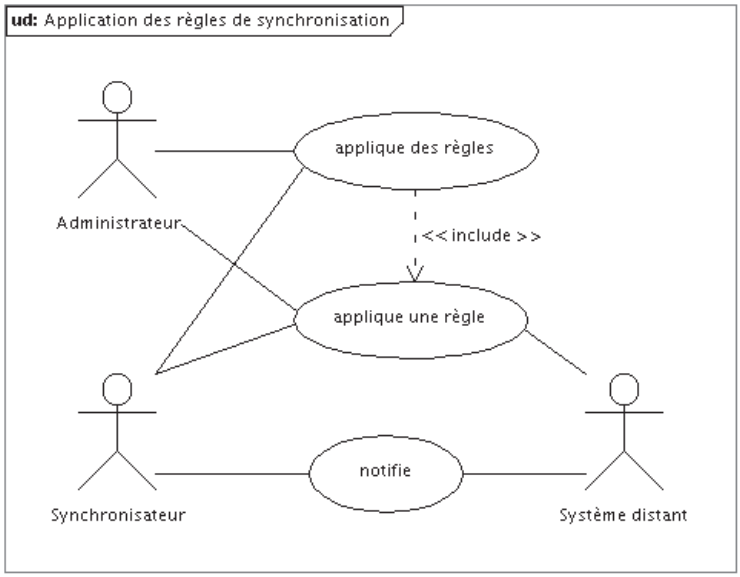

Figure 35 – Application des règles de synchronisation

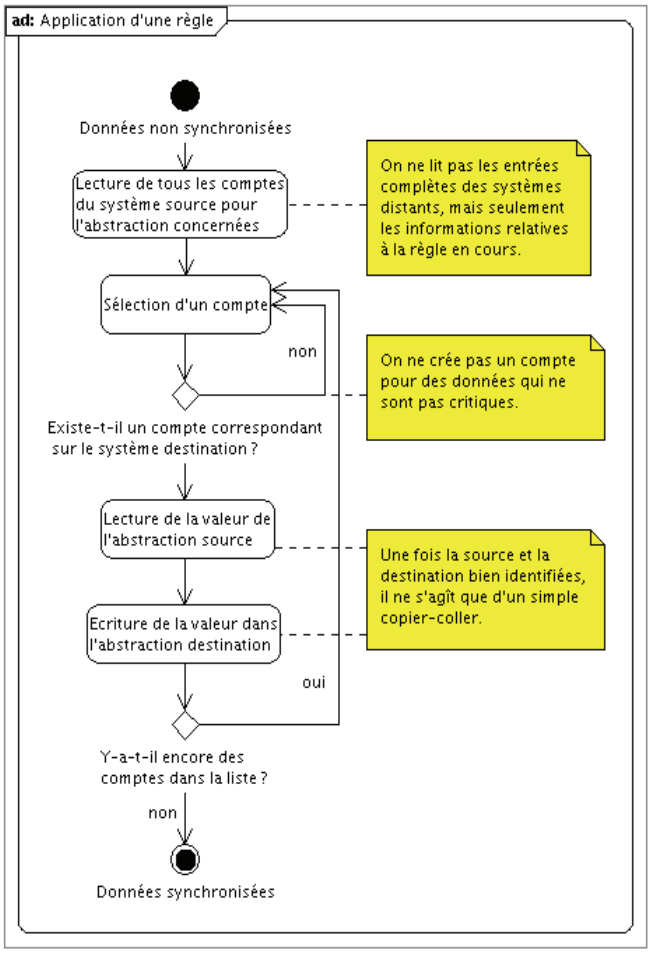

Figure 36 – Application d'une règle

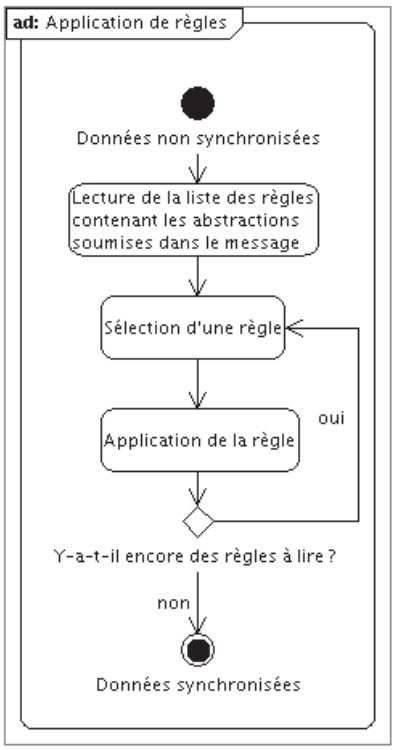

Figure 37 – Application des règles

- **ajout d'un listener** : l'abstraction source de la règle est inscrite côté système distant afin de préciser qu'il faudra la surveiller. Ceci afin de pouvoir notifier le synchronisateur lorsque les données auront été rafraîchies. La synchronisation pourra alors avoir lieu.
- **retrait d'un listener** : le listener est désinscrit du système distant. Les informations concernant cette abstraction ne seront plus rafraichies.

L'application des règles de synchronisation (cf. figure 35) se fait de manière manuelle ou automatique, voici les cas d'utilisation correspondants :

- **applique une règle** (cf. figure 36) : une règle est appliquée à intervalles réguliers (déclenchée par le synchronisateur) ou bien ponctuellement lorsqu'un besoin de rafraichissement est présent. L'application consiste à synchroniser les donnés des systèmes distants en fonction de la liste des abstractions qu'elle contient et du sens de synchronisation spécifié.
- **applique des règles** (cf. figure 37): cette fonctionnalité évite d'avoir à appliquer répétitivement toute une série de règles. Elle fait appel au cas « applique une règle » pour chacune des règles concernées.
- **notifie** : le système distant contacte le synchronisateur pour lui notifier les changements et la liste des modifications effectuées. Le synchronisateur n'est pas obligé de traiter l'information immédiatement.

4.4.2.2. Gestion des profils métiers

Le fait de centraliser l'administration permet de gagner du temps puisqu'il n'est pas nécessaire de se loguer sur de multiples plateformes pour affecter les droits nécessaires à un utilisateur. Or, lors de la création d'un utilisateur où l'on affecte une série de droits à ce dernier sur des systèmes distants, il est néanmoins nécessaire d'assigner manuellement chacun des rôles applicatifs, et s'il y en a par exemple une quinzaine, il faudra répéter l'opération d'assignation d'un rôle autant de fois.

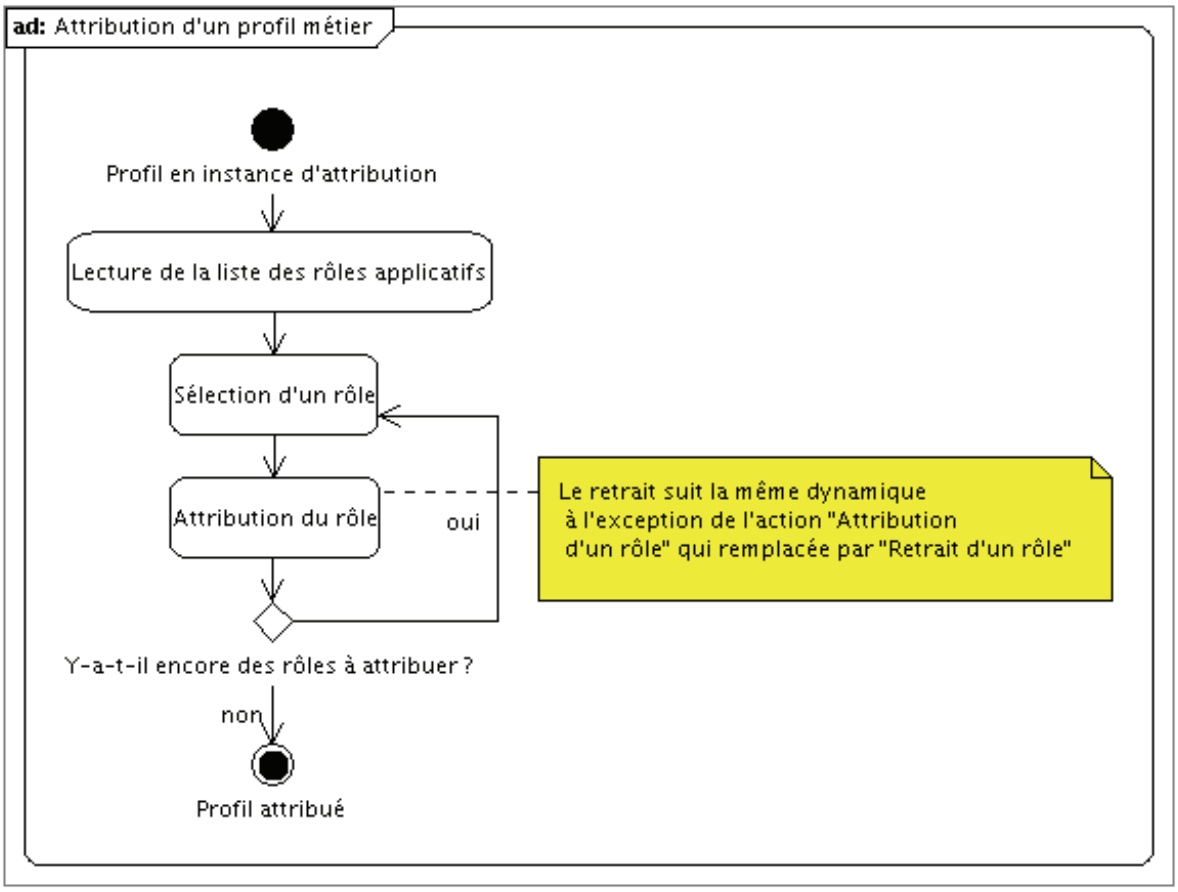

Figure 38 –Attribution d'un profil métier

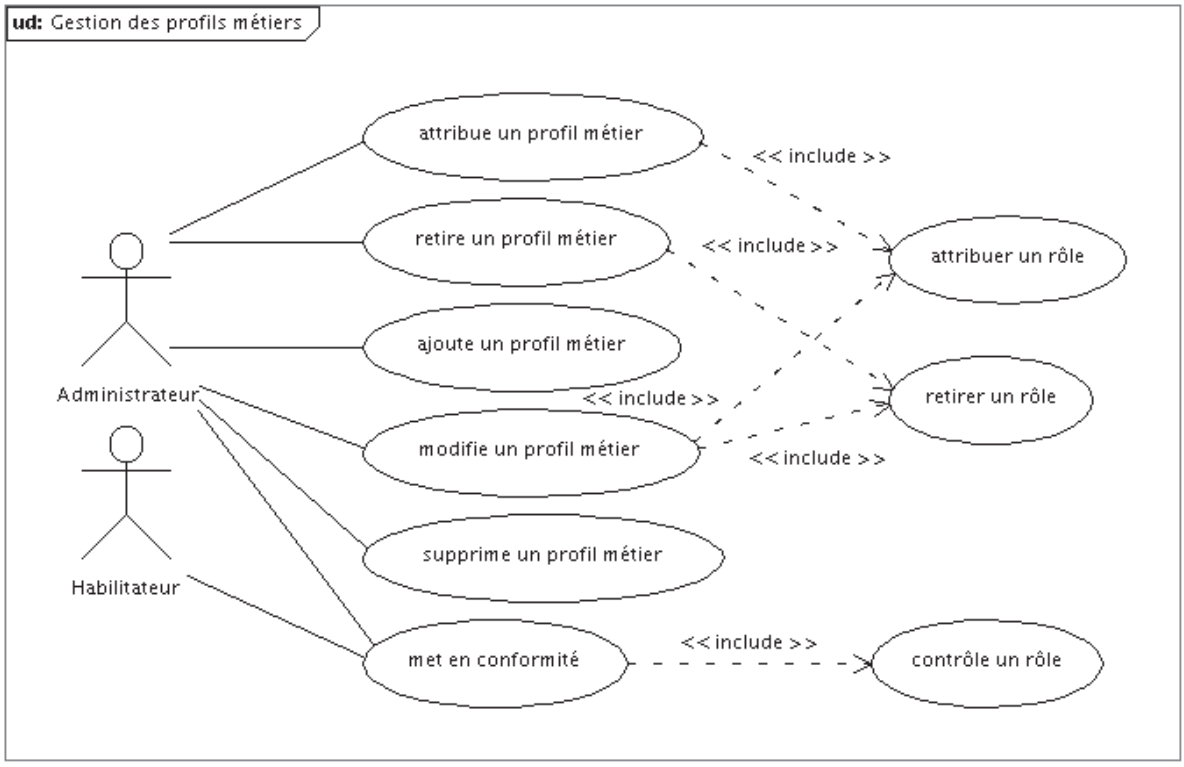

Figure 39 – Gestion des profils métier

La notion de profil métier pallie à ce problème en regroupant une série de rôles applicatifs en fonction d'un métier déterminé. Par exemple, si un trader a besoin d'accéder à 20 applications avec 25 rôles applicatifs, il suffira, après l'avoir configuré, d'attribuer ce profil métier (cf. figure 38) à cette personne. On divise donc par 25 le temps d'affection des rôles applicatifs dans ce cas précis.

La gestion des profils métiers (cf. figure 39) a donc 2 intérêts principaux :

- diminuer le temps d'administration des droits utilisateurs (précédemment décrit)
- augmenter la lisibilité des droits utilisateurs en offrant un panel applicatif à un ensemble d'utilisateurs regroupés par métier

Les cas d'utilisation des profils métier ne seront pas ici détaillés étant donné qu'ils sont surtout basés sur leur manipulation. Seuls l'attribution et le retrait présenteraient un intérêt mais il ne font qu'effectuer une boucle sur l'attribution ou le retrait de rôles.

4.4.3. Cas utiles

Il s'agît des cas représentant des fonctionnalités mineures d'IdMe, à l'exception de l'authentification et du SSO qui ne sont pas négligeables en temps de réalisation, mais qui toutefois, n'ont pas reçu une priorité élevée.

4.4.3.1. Synchronisation des mots de passe

La synchronisation des mots de passe s'établit en 2 parties :

- la configuration : le moteur permet de spécifier une expression régulière pour les mots de passe des utilisateurs IdMe. Chaque système distant contient aussi une expression régulière spécifique à son format et à un état signifiant la participation ou non du système à la synchronisation.

  Par conséquent, lorsqu'un utilisateur tente de modifier son mot de passe, IdMe vérifie d'abord si le format lui convient, puis il va confronter chaque expression régulière de chacun des systèmes distants au mot de passe soumis. Si le mot de passe satisfait toutes les règles, il est alors mis à jour dans IdMe puis dans tous les systèmes distants.
- la propagation : via la modification d'un utilisateur (cf. cas « Synchronise mot de passe »)

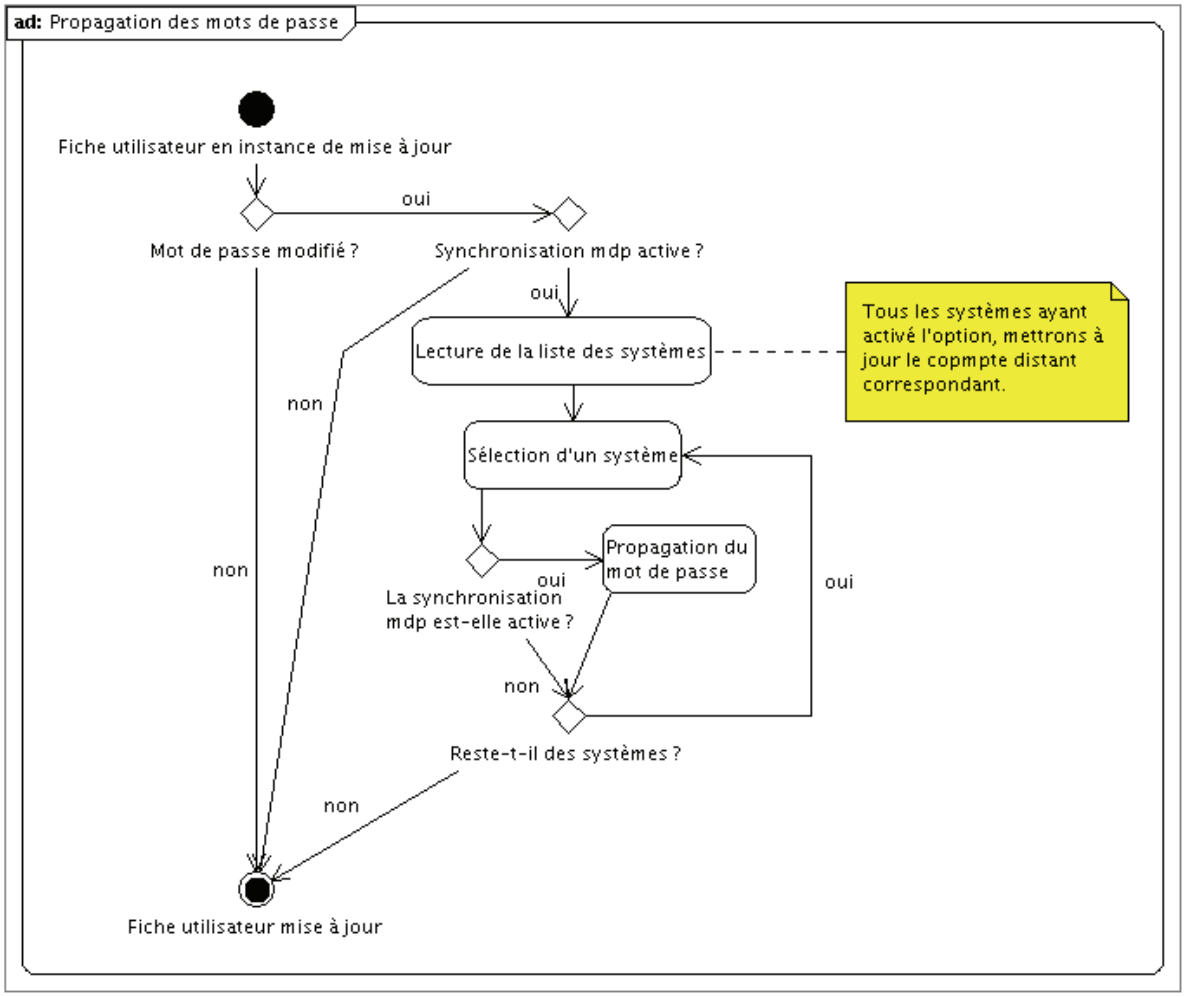

Figure 40 – Propagation des mots de passe

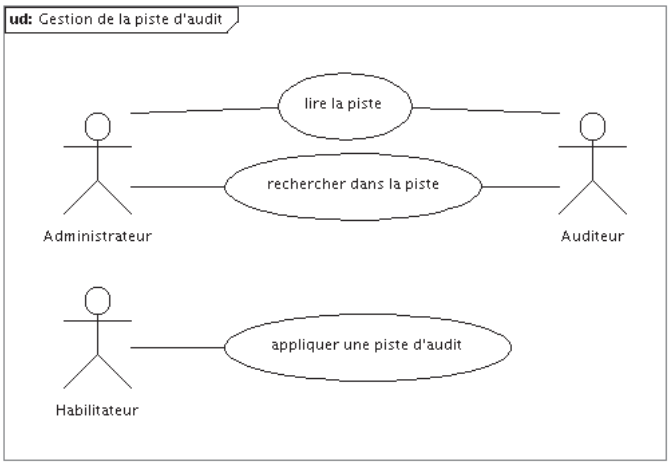

Figure 41 – Gestion de la piste d'audit

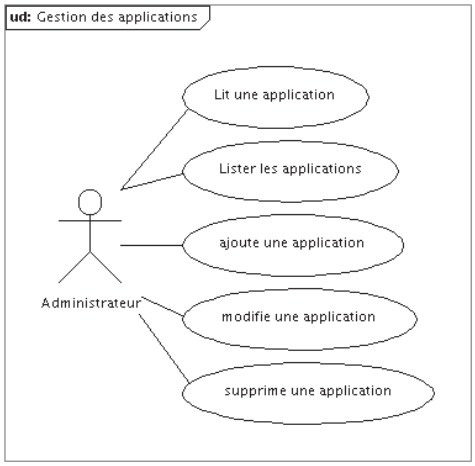

Figure 42 – Gestion des applications

Les cas d'utilisations suivant décrivent la synchronisation :

- **configure mdp synchro** : l'administrateur choisit un format de mot de passe accepté par IdMe, et par les systèmes distants. Ceci revient à choisir le plus petit dénominateur commun. Une expression régulière pourrait être utilisée pour spécifier ce format. Le format des mots de passe IdMe devient en fait celui de tous les systèmes concernés (option activée dans la fiche du système). La synchronisation pourra être ensuite activée en cochant une case. Une confirmation sera alors demandée.
- **synchronise mot de passe** (cf. figure 40) : l'utilisateur met à jour son mot de passe dans sa fiche. Si le mot de passe est valide, il est enregistré dans IdMe, puis propagé sur les systèmes distants, le mot de passe est alors le même, peu importe le système.

4.4.3.2. Gestion de la piste d'audit

La piste d'audit (cf. figure 41) recense toutes les opérations d'habilitation effectuées sur les utilisateurs, cela inclut aussi les changements d'état tels que la création, l'activation, l'inactivation, la suppression. Ces opérations sont regroupées dans le cas d'utilisation « Appliquer une piste d'audit »

La piste d'audit est consultable pour les super administrateurs, les administrateurs, et les auditeurs. Elle permet d'effectuer des recherches multicritères.

4.4.3.3. Gestion des applications

La gestion des applications (cf. figure 42) est similaire à celle des profils métier. La finalité n'est pas la même étant donné qu'une application regroupe des informations nécessaires au SSO (activation ou non) ainsi qu'à l'authentification de l'application elle-même (identifiant/mot de passe) pour souscrire de manière individuelle à ce service.

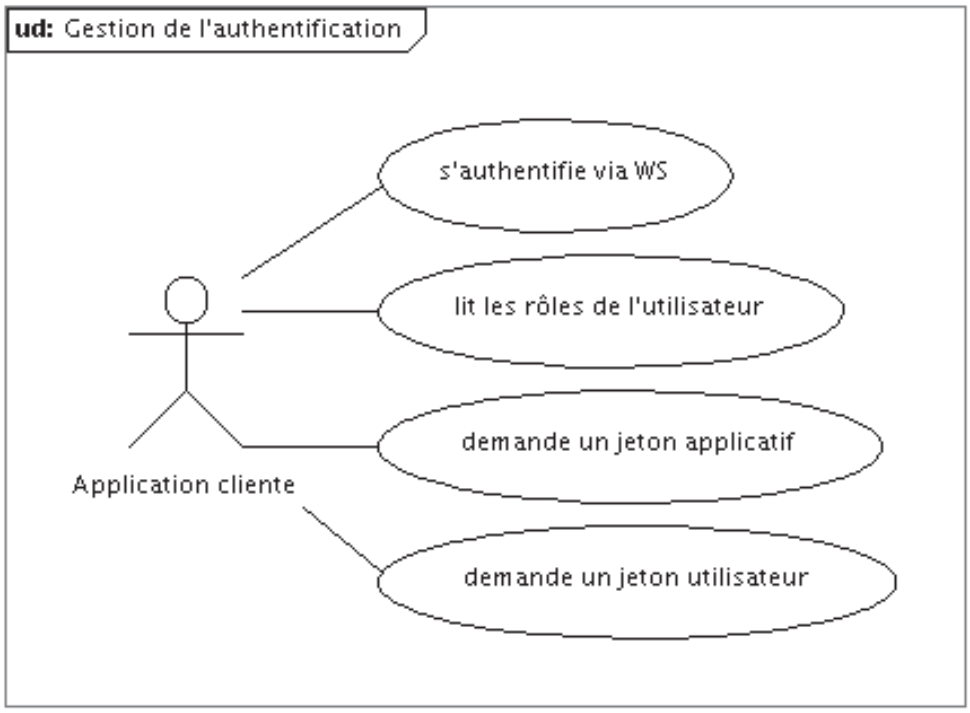

Figure 43 – Gestion de l'authentification

Tableau I - Codes erreurs d'authentification

Code	Erreur
-1	identifiant utilisateur ou mot de passe invalide
-2	jeton de l'application invalide
-3	authentification réussie mais l'application n'admet pas de SSO
-4	format d'adresse IP invalide

Elle se voit affecter l'ensemble des rôles applicatifs la concernant. Les rôles étant indépendants, il n'est pas exclus qu'ils soient utilisés par plusieurs applications.

Enfin, il est tout à fait possible d'utiliser les applications à des fins informatives pour des applications existantes non intégrées à IdMe par exemple.

4.4.3.4. Gestion de l'authentification et du SSO

La gestion de l'authentification et du SSO (cf. figure 43) se décompose en plusieurs parties :

- le recensement des applications (cf. 5.4.3.3)
- l'authentification des applications qui auront souscrit à ce service. Elles devront pour cela être enregistrées dans IdMe, et renseigner un identifiant et un mot de passe qui sera utilisé par l'application cliente, afin de prouver qu'elle a bien les droits d'accès à ce service. Il n'est pas rare de voir des services se réserver l'utilisation de certaines applications puisqu'elles en ont financé le développement, et ne veulent donc pas que d'autres services en bénéficient sans avoir dépensé un seul sou. L'authentification de l'application répond à ce besoin. Une fois la session applicative ouverte, les utilisateurs pourront s'authentifier.
- SSO : les applications, qui auront souscrit au service d'authentification, pourront bénéficier d'une session globale SSO, commune à toutes les applications gérées par IdMe.

Les cas d'utilisation sont les suivants :

- **s'authentifie via WS** : l'application soumet le jeton applicatif ainsi que l'identifiant et le mot de passe de l'utilisateur, si ce dernier est authentifié correctement un résultat l'indiquant est renvoyé (mode sans SSO), sinon un code erreur est retourné (cf. tableau I).
- **lit les rôles de l'utilisateur** : l'application soumet le jeton applicatif ainsi que le jeton utilisateur, les rôles de l'utilisateur pour cette application sont renvoyés.
- **demande un jeton applicatif** : ce cas permet d'authentifier une application auprès du moteur afin de vérifier qu'elle est bien inscrite au service d'authentification, et d'effectuer des requêtes utilisateurs par la suite. L'application soumet donc son code applicatif et son mot de passe. Si elle est authentifiée correctement, elle obtient un jeton applicatif.

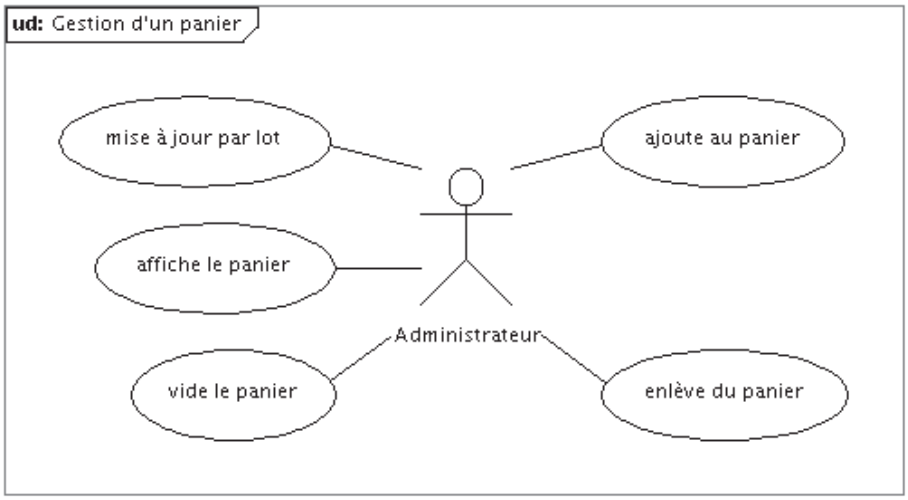

Figure 44 – Gestion d'un panier

- **demande un jeton utilisateur** : ce cas permet d'authentifier un utilisateur auprès du moteur et d'obtenir un jeton de session qui donnera accès aux fonctionnalités de SSO. L'application soumet le jeton applicatif ainsi que l'identifiant et le mot de passe de l'utilisateur, si ce dernier est authentifié correctement un jeton utilisateur est renvoyé.

4.4.3.5. Gestion d'un panier pour le traitement par lot

La gestion d'un panier (cf. figure 44) permet de sélectionner des utilisateurs via les listes de résultats issues de recherches utilisateurs. Il est possible d'affiner le contenu d'un panier (ou caddy) et d'effectuer des modifications groupées. C'est-à-dire impactant tous les utilisateurs concernés. Les cas d'utilisation ne seront pas ici détaillés, étant assez explicites.

5. Analyse et conception

5.1. Document d'analyse

Le document d'analyse permet d'appréhender la demande du client. Il transcrit les besoins en un langage intermédiaire entre le fonctionnel et le technique.

L'analyse, selon [5] et java.cnam.fr nous permet à partir des cas d'utilisation de :

- décrire les objets du système
- décrire le comportement des objets

Tout ceci doit être effectué sans se préoccuper de la technologie et du langage de développement cible, ainsi que des problèmes techniques (performance, distribution, interface utilisateur, etc.). Néanmoins l'analyse doit proposer une architecture logique qui sera adaptée techniquement lors des étapes suivantes de développement.

Le processus est donc le suivant pour chaque cas d'utilisation :

- découvrir les objets candidats
- découvrir les interactions entre ces objets
- décrire les classes

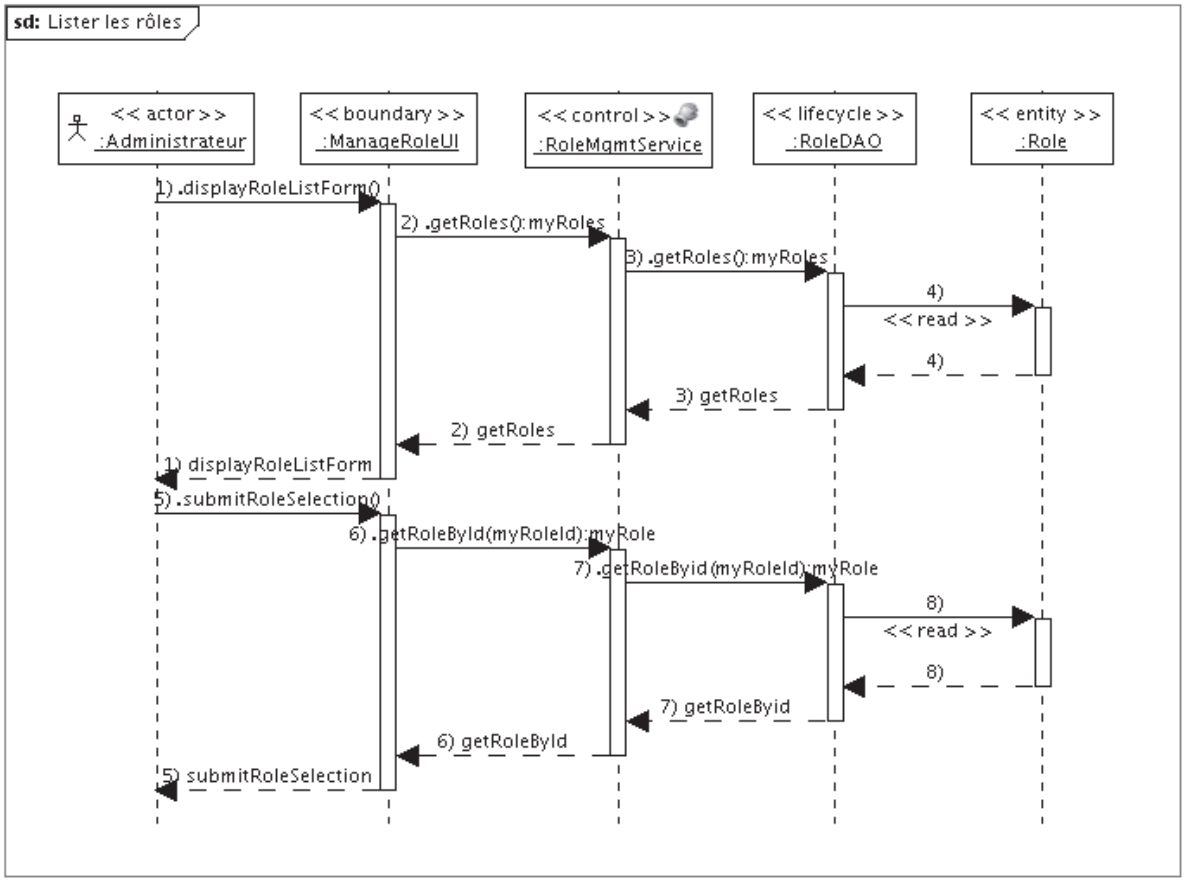

Figure 45 – Lister les rôles (analyse)

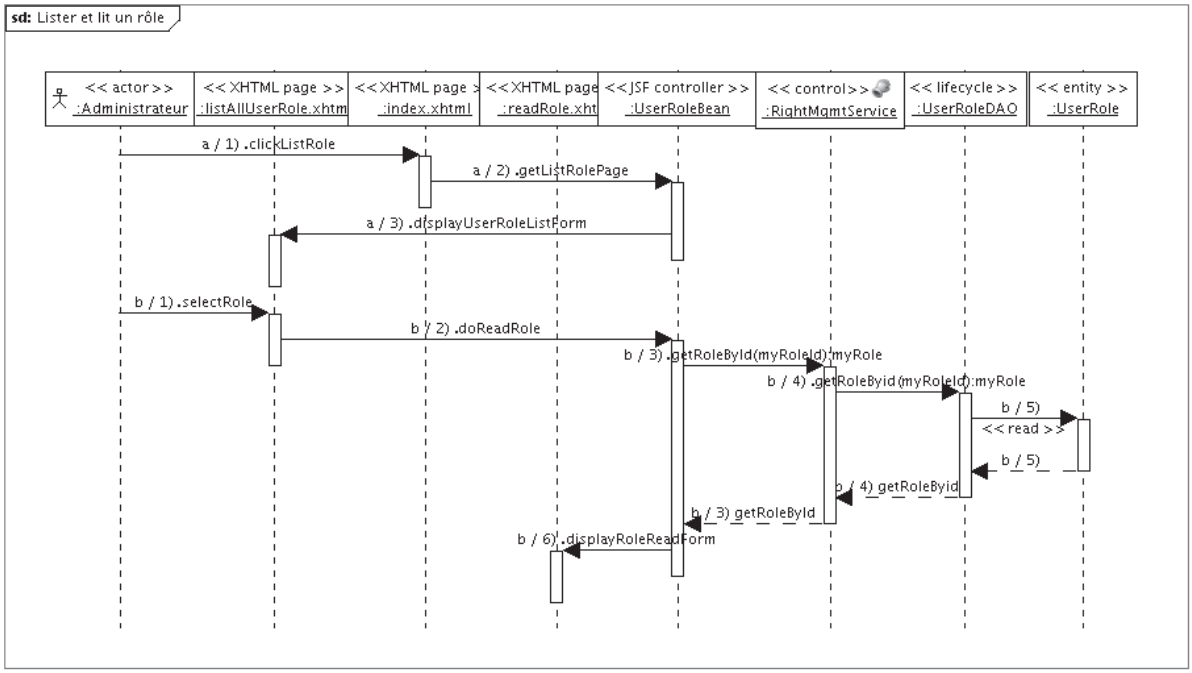

Figure 46 – Lister et lire un rôle (conception)

L'architecture logique est découpée en 4 couches selon [8] et utilise le modèle MVC :

- Données : objets métier notés « entity » représentant les données durables et persistantes.
- Accès aux données : objets responsables de trouver les objets « entity » et notés « lifecycle »
- Métier : objets assurant une coordination d'autres objets (entre la présentation et les données)
- Présentation : objets à la frontière entre un système et un acteur

Les types de diagrammes utilisés sont donc les suivants :

- Cas d'utilisation
- Diagrammes de séquence [7]
- Diagrammes de classe [7]

Ce document ne sera pas détaillé étant donné qu'il est similaire au document de conception, mais sans les aspects techniques.

5.2. Document de conception

Le document de conception est le résultat de la fusion entre le document d'analyse et l'apport technique des technologies choisies (voir chapitre suivant). La conception consiste, selon [5] et java.cnam.fr à réaliser le logiciel modélisé en analyse. Dans la pratique, les deux documents sont très proches dans la forme même si :

- l'analyse décrit ce que l'on va construire
- la conception décrit comment on va le construire

La conception étant l'affinement de l'analyse, les notions abordées de manière logique sont adaptées aux technologies choisies. Il n'y a donc pas de changements fondamentaux. Par exemple, la figure 45 illustre la lecture de rôle durant la phase d'analyse alors que la figure 46, lors de la phase de conception. On observe que la couche la plus affectée est celle de l'IHM.

5.3. Les acteurs

Les acteurs dans l'analyse ont été retravaillés. Ainsi, on voit la disparition des acteurs Synchronisateur, Habilitateur, Applicateur de cycle, au profit d'un nouvel acteur

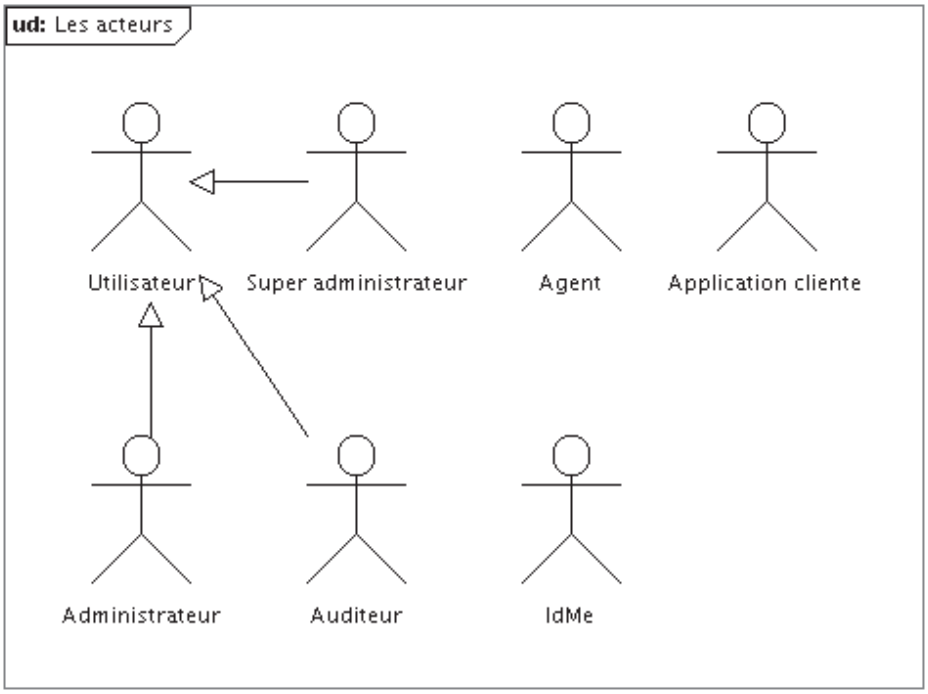

Figure 47 – Les acteurs

Figure 48 – Gestion de l'agent

IdMe (cf. figure 47). Il regroupe ainsi les rôles de synchronisateur puisque la synchronisation sera effectuée par l'outil de manière automatique, et applicateur de cycle, car ce dernier peut opéré de manière automatique ou manuelle, et souvent à des heures tardives. Il est donc plus facile de d'ordonner à une machine d'effectuer la nuit certaines opérations, plutôt que de demander à une personne d'effectuer une astreinte contraignante, et parfois coûteuse.

Le rôle d'habilitateur est assigné à l'administrateur car le seul cas valable où l'utilisateur final pourrait être l'habilitateur serait dans le cadre d'un workflow, ce qui n'est pas dans le périmètre du projet.

Un autre rôle fait son apparition, celui de l'agent. Il s'agît d'un acteur qui s'intercale entre les acteurs IdMe et Système distant.

5.4. L'agent

Etant donné les besoins exprimés précédemment, on se rend compte rapidement que l'accès au système distant est très spécifique d'un système à l'autre. En effet, l'accès à un annuaire LDAP n'est pas le même qu'à une base de donnée, et encore moins à un fichier plat. Ceci engendre donc du code spécifique, et l'on distingue la nécessité de créer un module séparé qui se chargera de dialoguer avec le système distant. Ce point de départ nous amène à scinder le projet en 2 sous-parties : le moteur (aussi appelé IdMe) et l'agent (cf. figure 48).

Ceci aura donc plusieurs avantages :

- l'agent permettra d'élaborer une langue unique intelligible par le moteur pour tous les systèmes distants
- la modification d'un agent ne nécessitera pas une maintenance du moteur
- la création d'un agent ne nécessitera pas la création ou la prise en compte de ce dernier par le moteur. Il suffira qu'il parle la même langue.

Par ailleurs, on y précise aussi que la synchronisation ne traite que les comptes dont l'identifiant est le même afin de limiter la durée de développement. C'est-à-dire qu'un utilisateur dont l'identifiant est rdupont, devra avoir des comptes nommés rdupont sur les systèmes distants pour que la synchronisation ou le provisioning s'opère. En effet, ce sujet pourrait faire l'objet d'un sous projet à lui-même. Il soulève des questions telles que : « comment puis-je rattacher un compte distant manuellement à mon compte

Tableau II – Type rôle et type synchronisation

	roleType	**synchType**
Rien	*false*	*false*
Rôle seul	*true*	*false*
Synchronisation seule	*false*	*true*
Rôle mais synchro source uniquement	*true*	*true*

IdMe ? » ou bien « comment puis-je spécifier une règle automatique de nommage des comptes sur un système distant ? ». Cette partie a donc été volontairement simplifiée.

L'agent, en tant que sous projet sera donc indépendant et aura sa propre IHM.

5.4.1. Gestion des abstractions

Les abstractions gérées par l'agent permettent d'effectuer la correspondance entre les valeurs issues des systèmes distants. L'entité gérée s'appelle donc Abstraction et contient non seulement la correspondance entre le nom physique d'un attribut et son nom logique, mais aussi des informations précisant si elle est obligatoire en création, si elle est publique ou privée, s'il s'agît d'un identifiant d'entrée utilisateur, etc.

Les abstractions peuvent être utilisées par le moteur de deux manières :

- en provisioning
- en synchronisation

Ceci peut engendrer certains problèmes lorsqu'une abstraction est utilisée des deux manières. Pour cela des attributs ont été créés afin d'éviter les conflits ou de les limiter.

Précisions sur la classe Abstraction :

- Les attributs roleType et synchType permettent de déterminer ce qu'on peut faire avec l'abstraction au niveau de la configuration des synchronisations, afin d'éviter les corruptions du type rôle et synchro destination (le rôle est écrasé par une synchro). Les possibilités sont décrites dans le tableau II.
- L'attribut listenerStatus indique si l'attribut est concerné en tant que source de synchronisation et ne pourra être effacé via l'UI de l'agent si c'est le cas.
- L'attribut publisherStatus indique si l'attribut est concerné en tant que destination de synchronisation et ne pourra être effacé via l'UI de l'agent si c'est le cas.
- L'attribut roleUsed indique si l'attribut est utilisé en tant que rôle et ne pourra être effacé via l'UI de l'agent si c'est le cas.
- L'attribut name reflète un nom logique d'attribut visible via RemoteConfigService et destiné à IdMe.
- L'attribut localName reflète un nom physique d'attribut visible via RemoteConfigService et est destiné à une utilisation interne à l'Agent. Il s'agit en fait du nom de l'attribut sur le système distant.

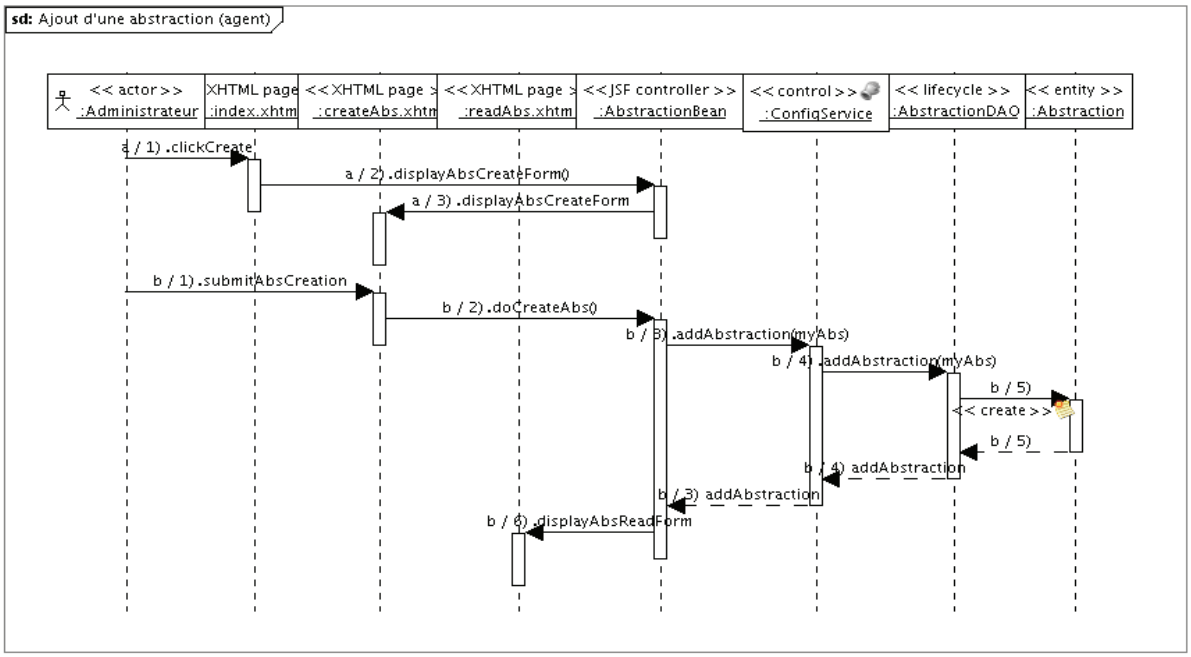

Figure 49 – Ajout d'une abstraction (agent)

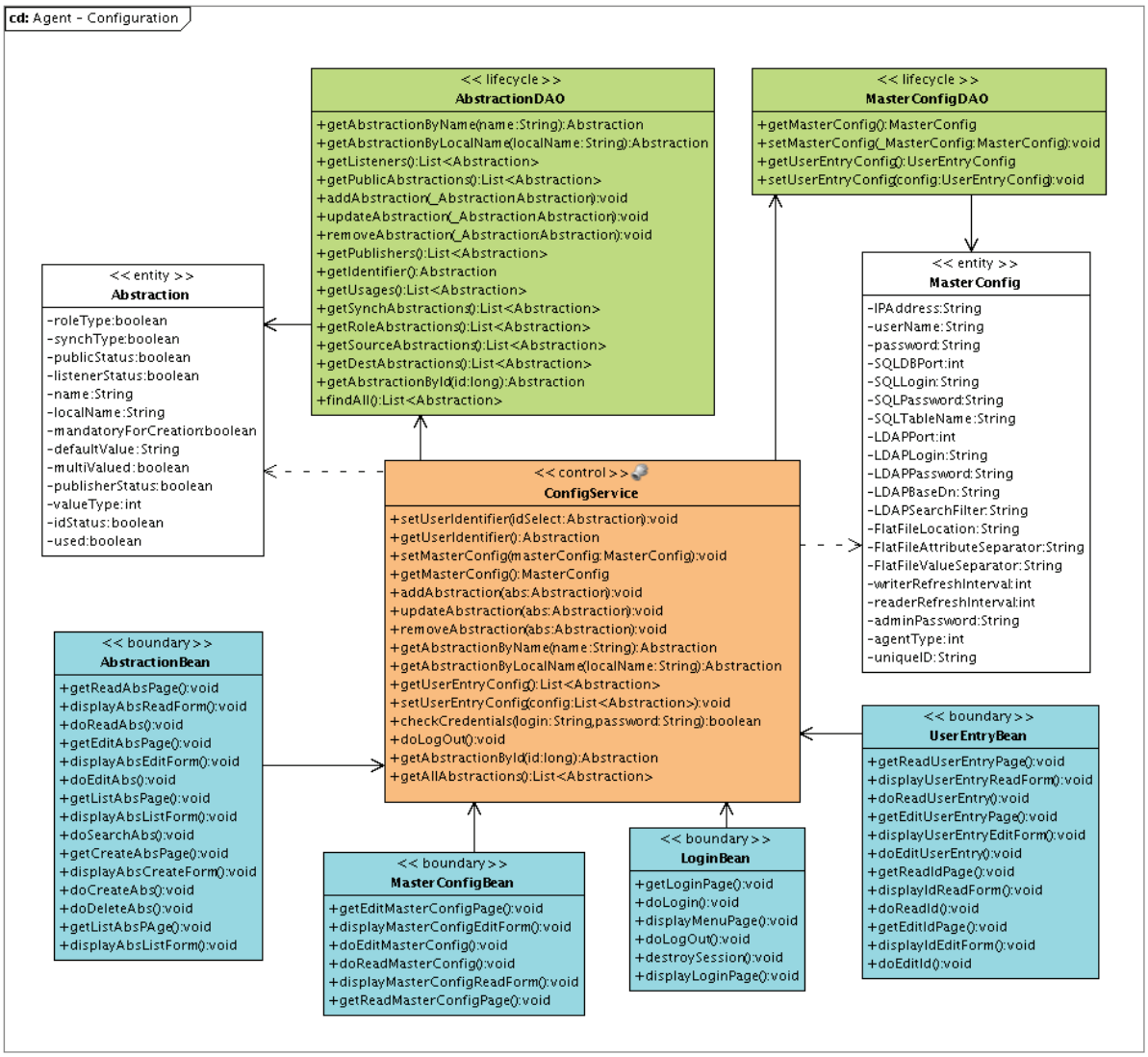

Figure 50 – Diagramme de classe de la configuration de l'agent

- L'attribut idStatus permet de préciser si l'abstraction est l'identifiant d'une entrée utilisateur.

L'utilisation de l'attribut publique conjointement à l'entrée utilisateur, permet de spécifier des attributs :

- que l'on peut renseigner
- que l'on doit renseigner
- qui ne sont pas visibles publiquement par le moteur, mais qui seront renseignés lors de la création d'une entrée. Par exemple, dans le cas d'une création d'utilisateur LDAP, l'attribut objectClass est généralement renseigné avec trois ou quatre valeurs. Ceci est propre au système distant et il préférable, dans ce cas, de diminuer la complexité au niveau du moteur. De plus, ceci permet aussi d'augmenter la lisibilité puisqu'on aura moins d'attributs côté moteur.

La manipulation des abstractions se fait au travers des couches suivantes (cf. figure 49):

- Abstraction : l'entité représentant une abstraction
- AbstractionDAO : le DAO gérant la persistance des abstractions
- ConfigService : le service de configuration
- AbstractionBean : le bean managé JSF assurant le lien entre ConfigService et les pages XHTML
- Pages XHTML

5.4.2. Gestion de la configuration

La configuration de l'agent (cf. figure 50) se découpe en trois parties :

- la configuration locale
- l'entrée utilisateur
- l'accès distant (depuis le moteur)

5.4.2.1. Configuration locale

Cette partie permet de configurer l'accès au système distant. On y entre les mots de passe, les adresses IP, les ports TCP, les délais de rafraîchissement, le type d'agent (LDAP, SQL, fichier plat), etc.

Toutes ces informations sont contrôlées et dépendantes du type d'agent, ainsi lorsqu'on active l'agent, les opérations suivantes sont effectuées :

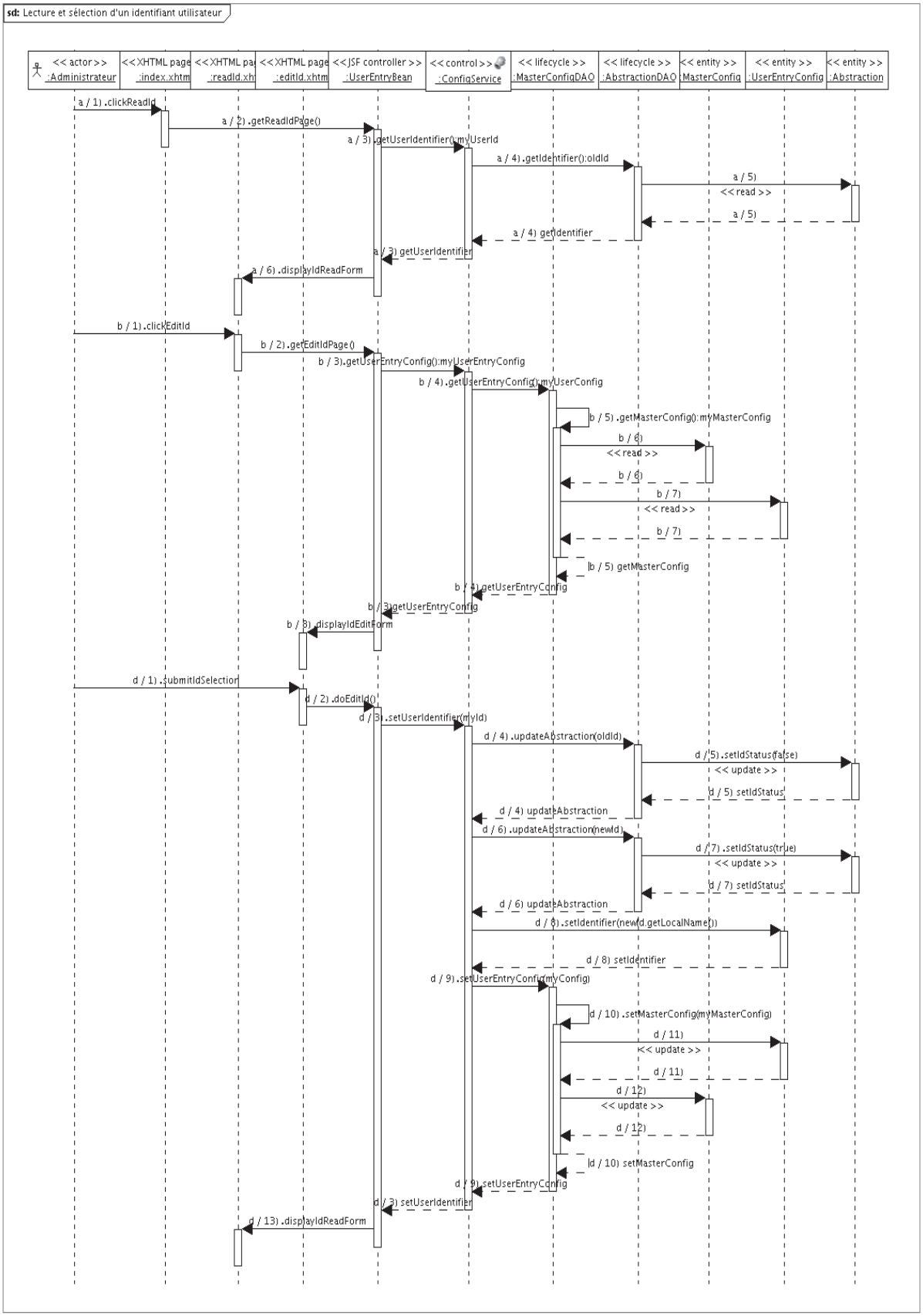

Figure 51 – Lecture et sélection d'un identifiant utilisateur

- la validité de l'accès au système distant est vérifiée
- la synchronisation est démarrée
- le provisioning est démarré et prêt à recevoir les requêtes du moteur

La manipulation de la configuration se fait au travers des couches suivantes :

- MasterConfig: l'entité représentant la configuration
- MasterConfigDAO : le DAO gérant la persistance de la configuration
- ConfigService : le service de configuration
- MasterConfigBean : le bean managé JSF assurant le lien entre ConfigService et les pages XHTML
- Pages XHTML

5.4.2.2. L'entrée utilisateur

L'entrée utilisateur représente l'ensemble des attributs nécessaires à la création d'un compte sur le système distant. En ajoutant, comme vu précédemment, un attribut userEntryStatus, on supprime le besoin de créer une nouvelle entité. Ainsi, la configuration de l'entrée utilisateur se présente dans un écran à part, permettant de sélectionner les abstractions disponibles. Mais l'information d'appartenance à l'entrée utilisateur est incluse dans les abstractions.

Par ailleurs, tout entrée nécessite un identifiant qui sera, lui aussi, déterminé dans un écran spécifique, mais dont l'information sera portée par une abstraction via un attribut idStatus (cf. figure 51).

La manipulation de l'entrée utilisateur se fait au travers des couches suivantes :

- Abstraction : l'entité représentant une abstraction
- AbstractionDAO : le DAO gérant la persistance des abstractions
- ConfigService : le service de configuration
- UserEntryBean : le bean managé JSF assurant le lien entre ConfigService et les pages XHTML
- Pages XHTML

5.4.2.3. Accès distant

L'accès distant se fait grâce à l'appel au web service ConfigWS. Il permet au moteur d'effectuer les opérations suivantes :

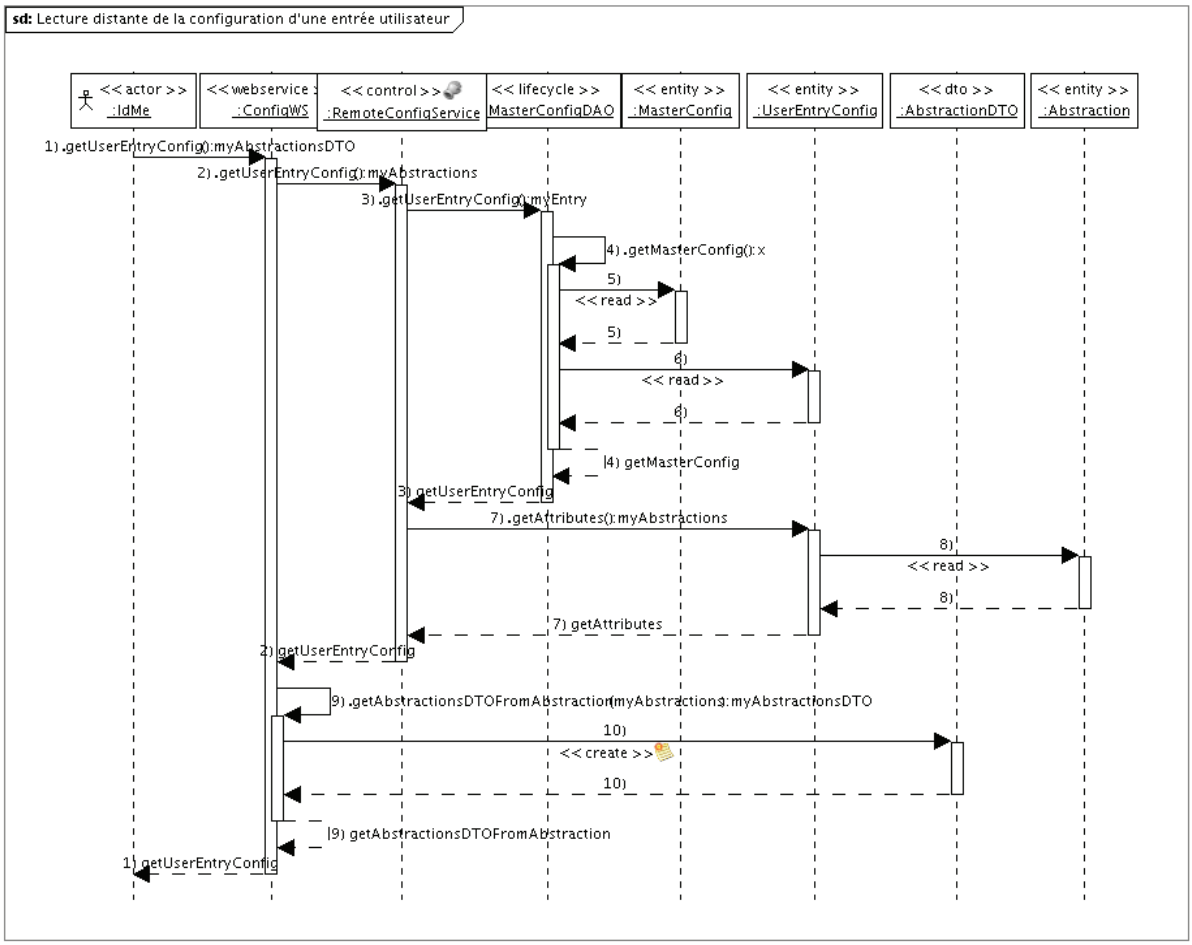

Figure 52 - Lecture distante de la configuration d'une entrée utilisateur

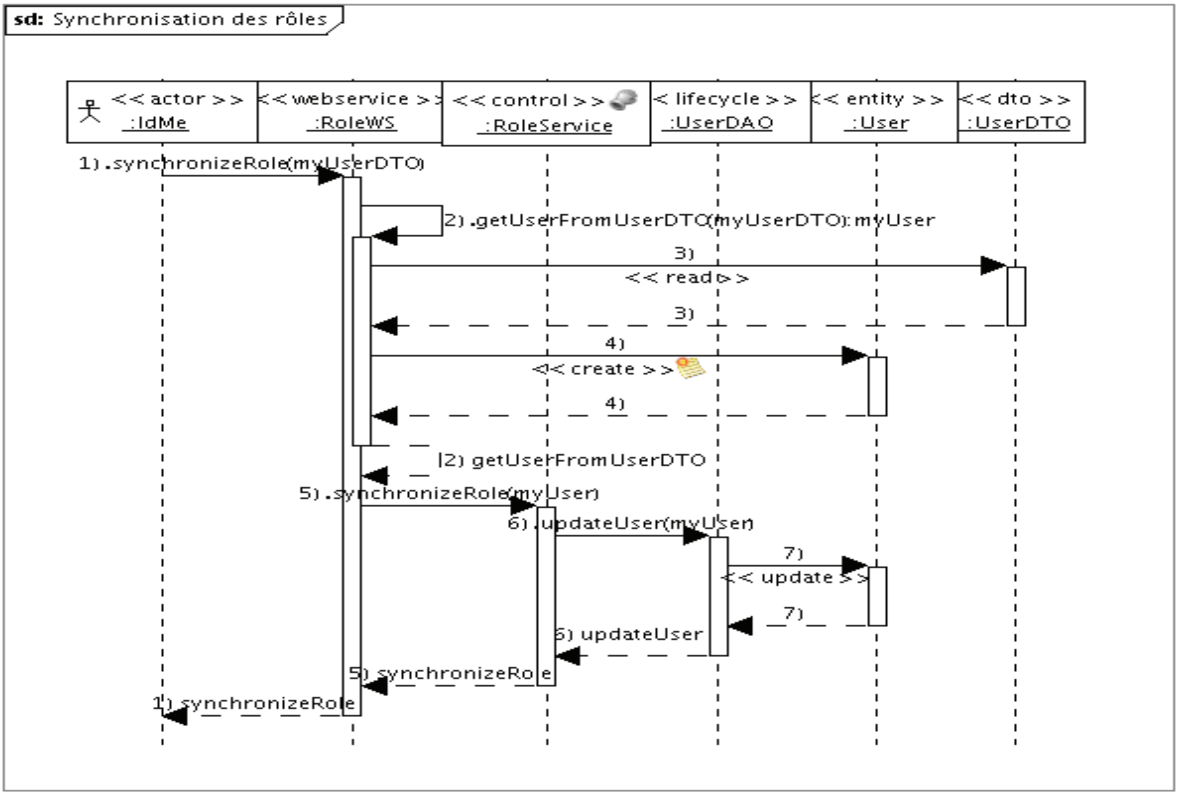

Figure 53 – Synchronisation des rôles

- lire les abstractions disponibles
- marquer les abstractions comme utilisées grâce à l'attribut used, ceci empêche toute modification de l'abstraction, et donc évite toute incohérence entre le moteur et l'agent
- lire le contenu de l'entrée utilisateur (cf. figure 52)
- ajout et retraits de listeners (voir la synchronisation)

La manipulation de la configuration distante se fait au travers des couches suivantes :

- Abstraction, MasterConfig : les entités concernées
- AbstractionDAO, MasterConfigDAO : les DAO concernés
- RemoteConfigService : le service de configuration distante
- ConfigWS : le web service permettant l'accès au service de configuration distante

5.4.3. Gestion du provisioning

Le provisioning est disponible via le web service de gestion des rôles et comptes distants : RoleWS. Il est possible d'effectuer les opérations suivantes :

- créer un compte : pour cela on fournit l'identifiant de l'utilisateur ainsi que les valeurs adéquates pour chacune des abstractions définies dans l'entrée utilisateur. Un contrôle des attributs et des valeurs effectuées est bien entendu effectué.
- modifier les rôles d'un compte utilisateur distant : on fournit l'identifiant de l'utilisateur, puis les rôles à modifier. Ceci se concrétise par l'affectation d'une valeur à l'abstraction correspondante sur le système distant.
- contrôler les rôles d'un compte utilisateur distant : on fournit l'identifiant de l'utilisateur, puis les rôles à contrôler. Pour cela on effectue une lecture de l'abstraction correspondante sur le système distant et on compare la valeur obtenue avec celle passée en paramètre.
- activer ou inactiver des comptes utilisateurs
- synchroniser les rôles : force un rafraîchissement des rôles distants (cf. figure 53)

L'accès au système distant doit pouvoir se faire sans tenir compte du système lui-même. Selon [9] on utilise la programmation par interface. Ainsi, l'agent est développé autour de l'interface DAO qui est implémentée par 3 classes correspondant aux 3 types de systèmes distants supportés par l'agent : UserLDAPDAO, UserSQLDAO, UserFlatFileDAO.

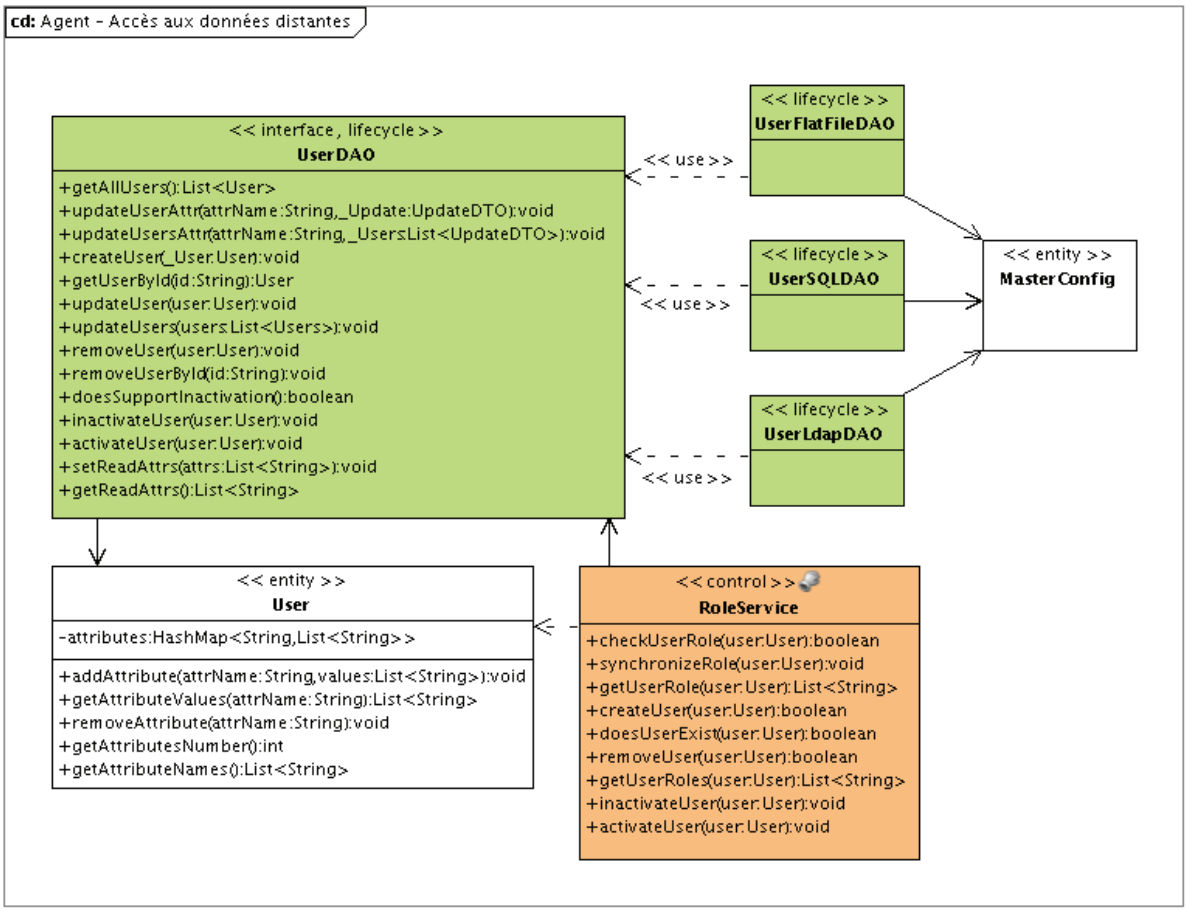

Figure 54 – Accès aux données distantes (agent)

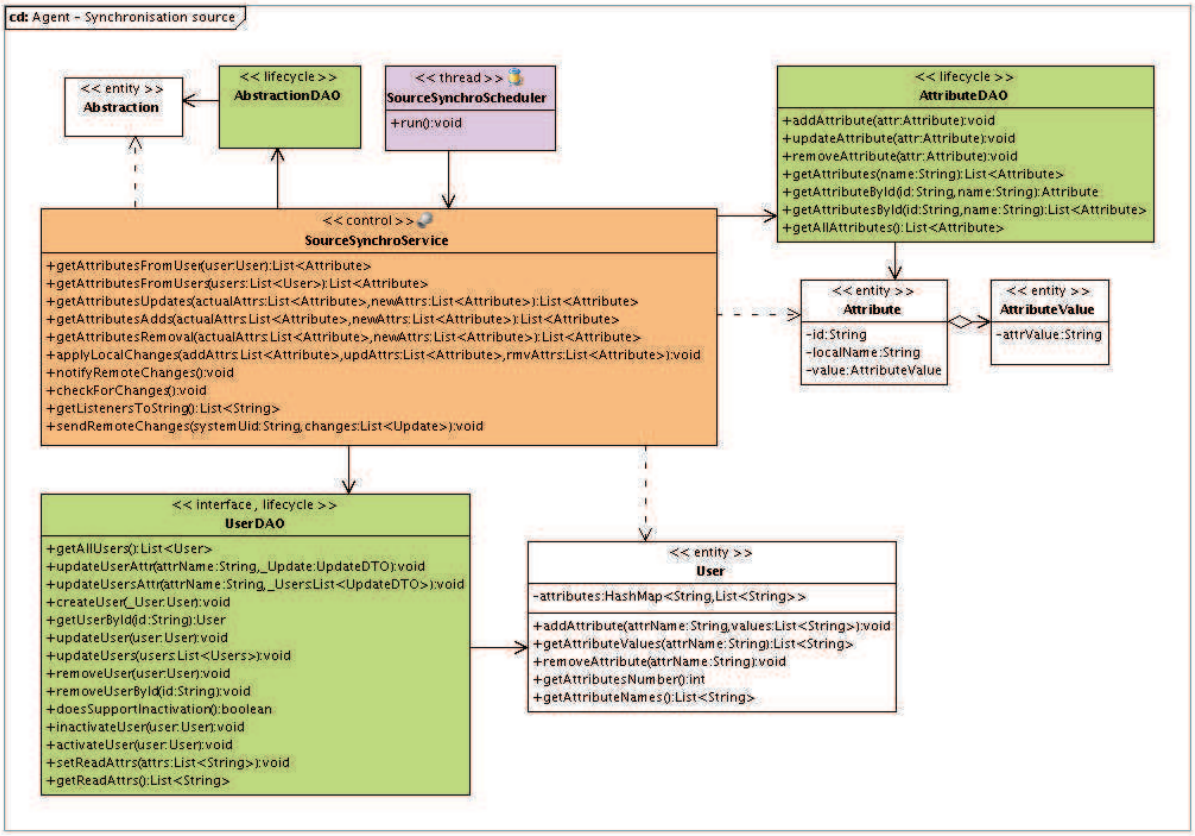

Figure 55 – Synchronisation source (agent)

L'instanciation du DAO utilisateur se fait grâce au pattern « Factory » (voir [11]). Ceci permet de déporter l'instanciation des classes UserDAO vers une classe spécialisée UserDAOFactory. Par ailleurs le pattern « Singleton » (voir [11]) est aussi utilisé afin de garantir que l'agent ne gère qu'un seul système distant. Ceci permet aussi de s'assurer des accès concurrent à ce dernier concernant la synchronisation et le provisioning.

Le support d'un nouveau type de système distant est donc simple à mettre en place puisqu'il s'agît d'écrire une classe respectant l'interface UserDAO et de mettre à jour UserDAOFactory pour qu'en fonction des nouvelles valeurs de configuration, elle puisse instancier la nouvelle classe.

La manipulation du provisioning (cf. figure 54) se fait au travers des couches suivantes :

- User, MasterConfig : les entités concernées
- UserDAO, MasterConfigDAO : les DAO concernés
- RoleService : le service de configuration distante
- RoleWS : le web service permettant l'accès au service de gestion des rôles

5.4.4. Gestion de la synchronisation

La synchronisation se découpe en deux parties :

- la détection et l'émission de modifications
- la réception et la propagation de modifications

5.4.4.1. Emission de modifications

L'émission de modifications (cf. figure 55) est liée à la notion de listener. Un listener est une abstraction qui, lorsqu'elle est marquée (attribut isListener), fait partie d'une liste d'attribut à vérifier.

Périodiquement, grâce à un thread réveillé selon l'intervalle défini dans la configuration de l'agent, une lecture des listeners est effectuée sur le système distant. S'il s'agît de la première détection, les valeurs sont stockées dans la table des attributs (les dernières valeurs) via le DAO AttributeDAO. Lors de la détection suivante, on compare les valeurs avec celles de la table des attributs, si certaines valeurs ont changé, elles sont envoyées au moteur, puis la table est mise à jour.

Il est important de noter qu'aucun routage n'est effectué au niveau de l'agent, on envoie donc systématiquement les modifications au moteur, qui jouera le rôle d'aiguilleur. Aucune communication d'un agent vers un autre n'aura donc lieu.

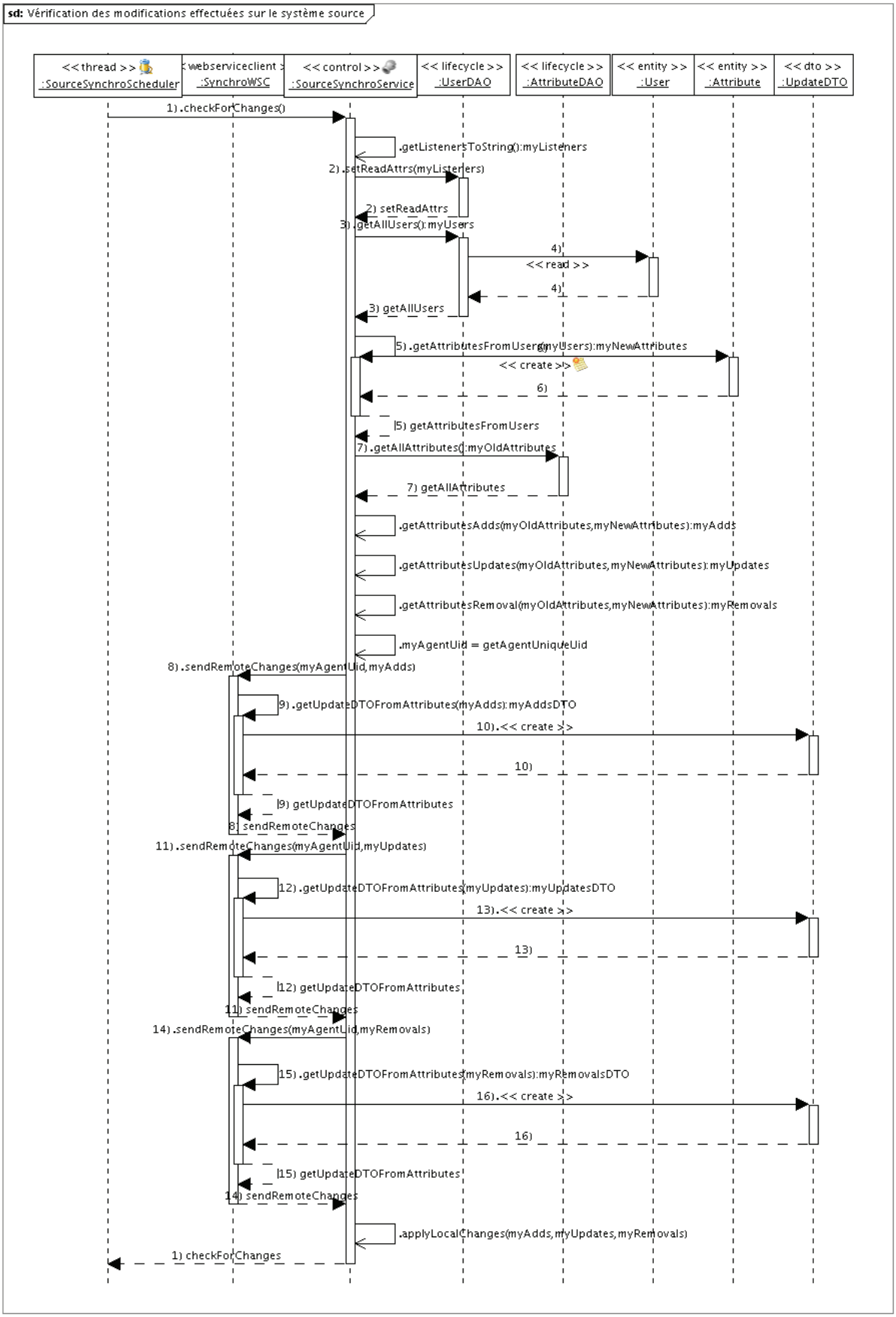

Figure 56 – Vérification des modifications effectuées sur le système source (agent)

La détection et l'émission des modifications (cf. figure 56) se fait au travers des couches suivantes :

- Abstraction, User, Attribute, AttributeValue : les entités concernées
- UserDAO, AttributeDAO, AbstractionDAO : les DAO concernés
- SourceSynchroService : le service de synchronisation en émission
- SourceSynchroScheduler: le thread gérant à intervalle régulier la détection
- SynchroWSC : le web service client permettant d'accéder au moteur

5.4.4.2. Réception de modifications

La réception de modifications est liée à la notion de publishers. Un publisher est une abstraction qui, lorsqu'elle est marquée (attribut isPublisher), fait partie d'une liste d'attributs autorisés à recevoir des modifications pour le système distant.

Lorsque le moteur notifie un agent, il associe un numéro de séquence afin de garantir l'ordre dans lequel le système doit être mis à jour.

Périodiquement, grâce à un thread réveillé selon l'intervalle défini dans la configuration de l'agent, une lecture des numéros de séquences non traités est effectuée, en commençant par le plus petit, donc le plus ancien. L'agent contacte ensuite le moteur en fournissant son identifiant unique et le numéro de séquence, il obtient alors les modifications à effectuer.

L'agent filtre ensuite les modifications à effectuer en fonction des publishers inscrits, puis procède à la mise à jour du système.

La réception des modifications se fait au travers des couches suivantes :

- Abstraction, User, Attribute, AttributeValue, UpdateSequence : les entités concernées
- UserDAO, AttributeDAO, UpdateSequenceDAO : les DAO concernés
- DestinationSynchroService : le service de synchronisation en réception
- DestinationSynchroScheduler: le thread gérant à intervalle régulier la réception
- SynchroWSC : le web service client permettant d'accéder au moteur

5.5. Le moteur

Le moteur est la colonne vertébrale du projet. Il contient la base des utilisateurs, dialogue avec les agents et fournis des services d'authentification et de SSO.

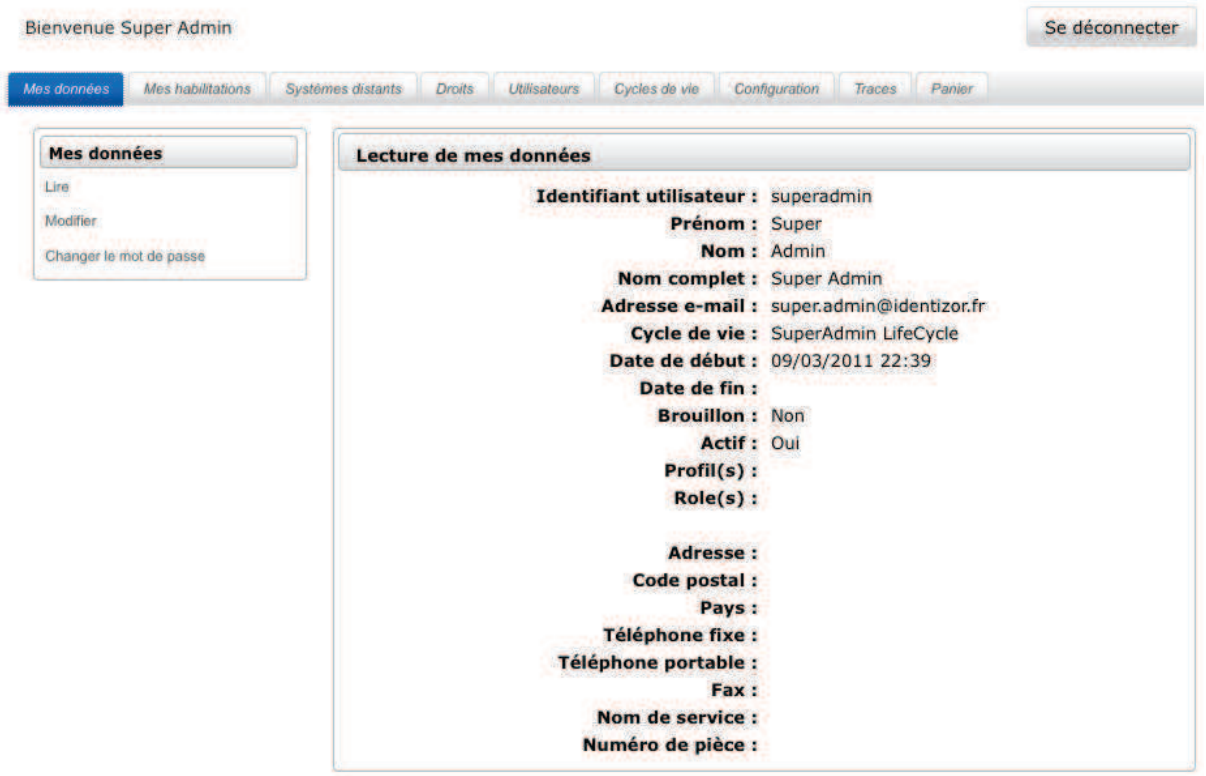

Figure 57 – Ecran d'accueil d'un utilisateur

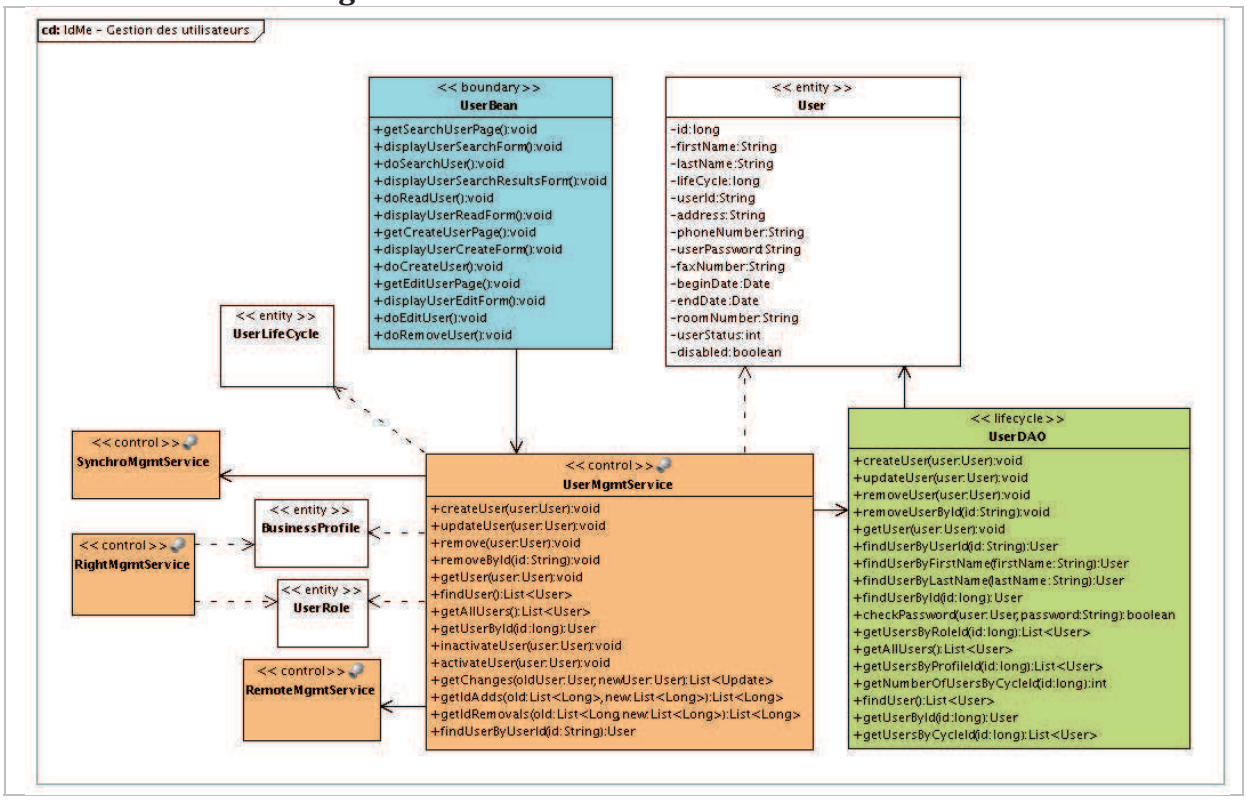

Figure 58 – Gestion des utilisateurs

5.5.1. Gestion de l'authentification moteur

L'authentification à l'IHM du moteur (cf. figure 57) se fait via un formulaire demandant identifiant et mot de passe (standard JEE). L'utilisateur ainsi authentifié obtient un ou plusieurs rôles. Les rôles disponibles sont directement issus des acteurs et sont les suivants :

- Utilisateur : tous les utilisateurs authentifiés ont ce rôle
- Super administrateur : il a tous les droits et est créé s'il n'existe pas lors de la première connexion d'un utilisateur.
- Administrateur : il a tous les droits
- Auditeur : droits sur la trace d'audit

5.5.2. Gestion des utilisateurs

5.5.2.1. Utilisateurs

La gestion des utilisateurs (cf. figure 58), accessible aux administrateurs via un onglet dédié, prend en compte la création, la suppression, la modification des utilisateurs. Une fiche utilisateur permet donc la modification des données personnelles ainsi que des droits dans un écran dédié (rôles, profils).

La gestion des utilisateurs se fait au travers des couches suivantes :

- User : l'entité des utilisateurs
- UserDAO : le DAO des utilisateurs
- UserMgmtService : le service de gestion des utilisateurs
- UserBean : le bean managé JSF pour les utilisateurs
- Pages XHTML

5.5.2.2. Données personnelles

La gestion des données personnelles est accessible à tous les utilisateurs et comprends la fiche des données, dont la plupart sont modifiables (les données critiques comme les habilitations ne le sont évidemment pas). Elle comprend aussi une interface de recherche dans les traces d'audit mais restreintes à l'utilisateur connecté.

La gestion des données personnelles se fait au travers des couches suivantes :

- User, AuditTrace : les entités concernées

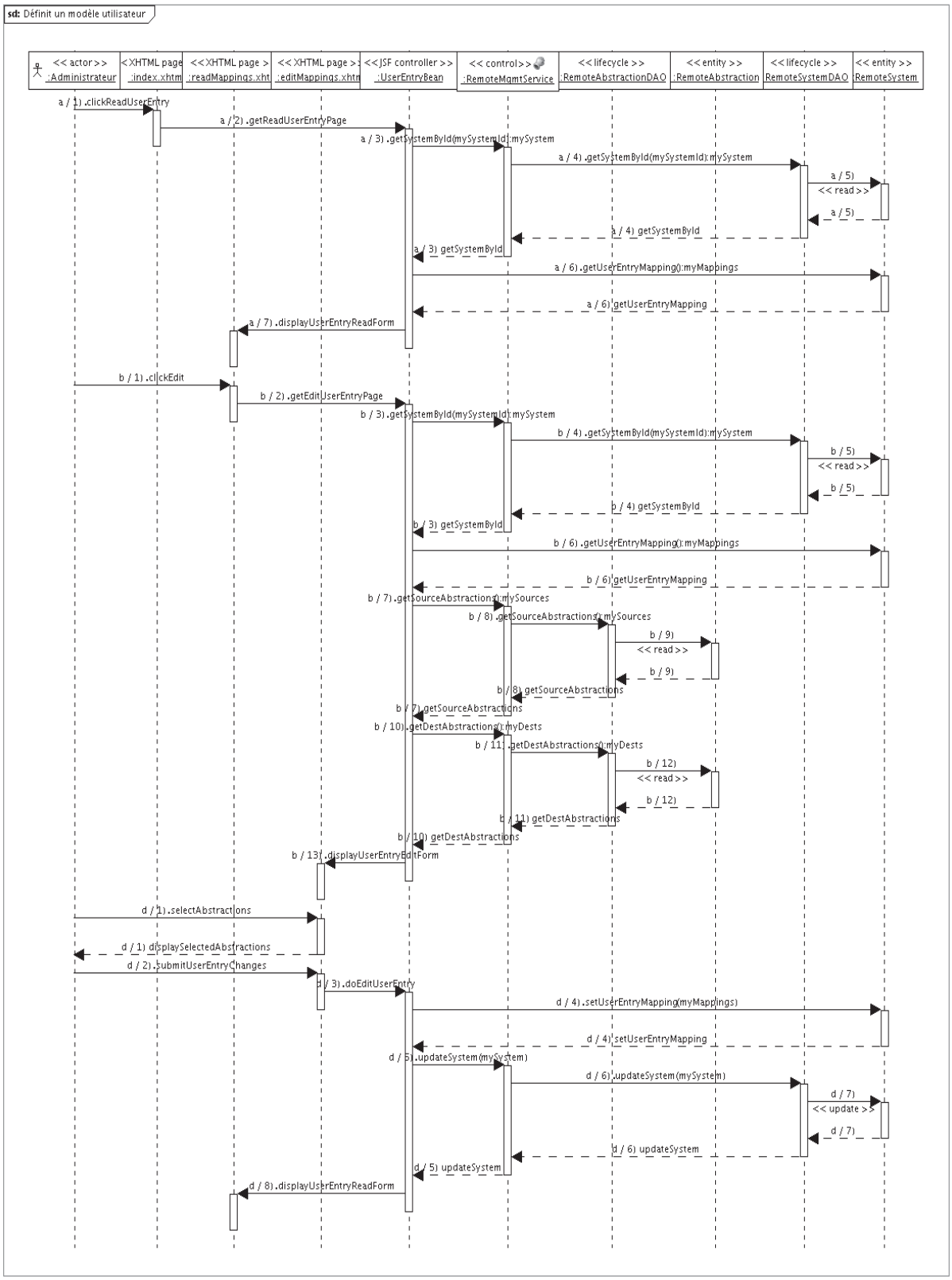

Figure 59 – Définit un modèle utilisateur

- UserDAO, AuditTraceDAO : le DAO des entités concernées
- UserMgmtService, ConfigService : les services concernés
- SelfUserBean, SelfAuditTraceBean : les bean managé JSF concernés
- Pages XHTML

5.5.2.3. Panier

La gestion du panier permet la modification par lot des utilisateurs. On sélectionne les utilisateurs dans l'onglet gestion des utilisateurs. La modification groupée des informations se fait alors via un formulaire dédié et sur des informations non critiques.

Le panier n'est pas matérialisé au travers d'une entité et n'a donc pas de DAO associé. En effet, il est relatif à une session utilisateur et donc n'a pas lieu d'être persisté. Aussi, le retrouvons nous au sein du service CaddyMgmtService

La gestion du panier se fait au travers des couches suivantes :

- User : l'entité utilisateur
- UserMgmtService, CaddyMgmtService: les services concernés
- CaddyBean : le bean managé JSF du panier
- Pages XHTML

5.5.3. Gestion des systèmes distants

La gestion des systèmes distants est découpée en 4 sections :

- l'accès aux systèmes distants qui se fait via les agents. Chaque système doit être inscrit auprès du moteur. Il suffit pour cela de rempli ses coordonnées (adresse IP, port TCP, mot de passe, etc.).
- la sélection des abstractions : une fois le système inscrit, les abstractions disponibles sur l'agent distant sont affichées. Il est alors possible de sélectionner ces abstractions, qui seront alors marquées comme utilisées côté agent, et seront utilisables dans le moteur. On pourra alors les associer à des rôles ou à des règles de synchronisation (cf. 6.2.6).
- la constitution de l'entrée utilisateur : on y sélectionne les abstractions de l'entrée utilisateur et on les associe aux attributs du moteur (cf. figure 59).
- les règles de synchronisation : elles contiennent les systèmes source et destination, les attributs source et destination, ainsi qu'un indicateur d'activité.

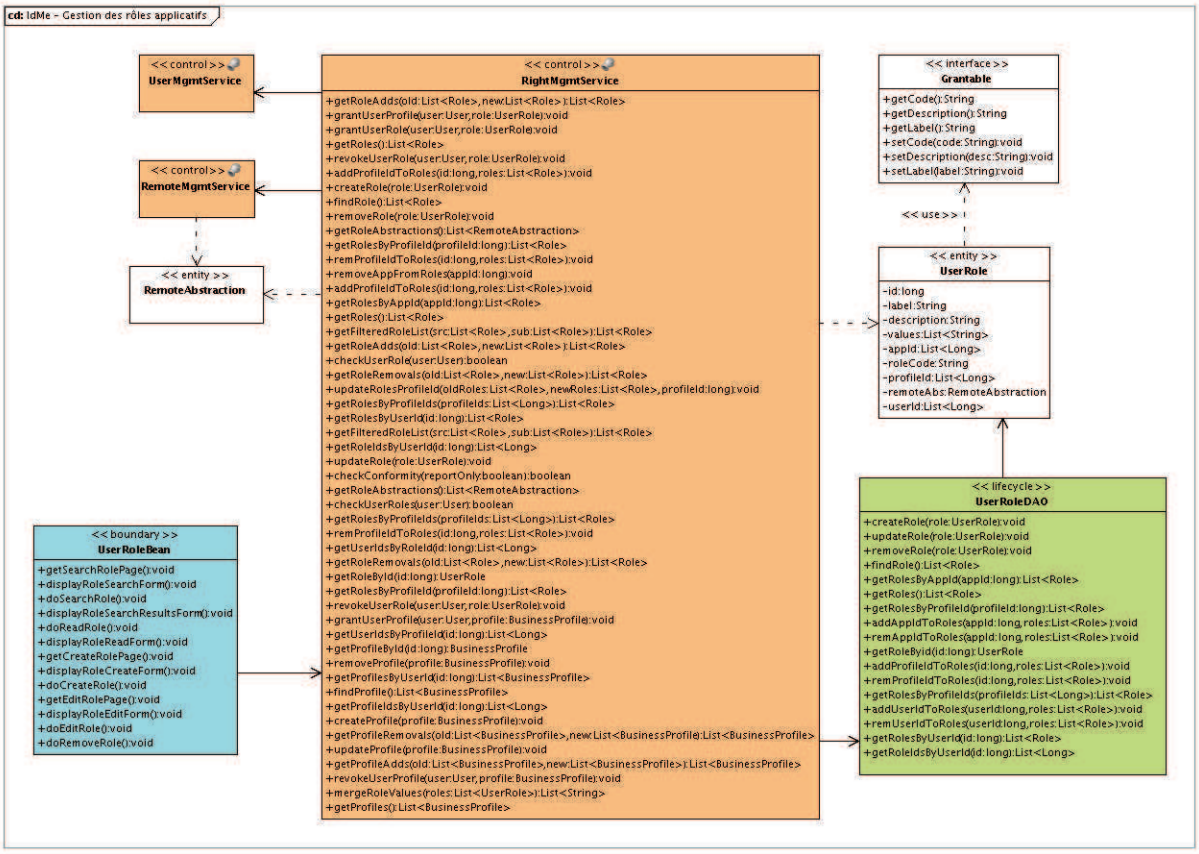

Figure 60 – Gestion des rôles applicatifs

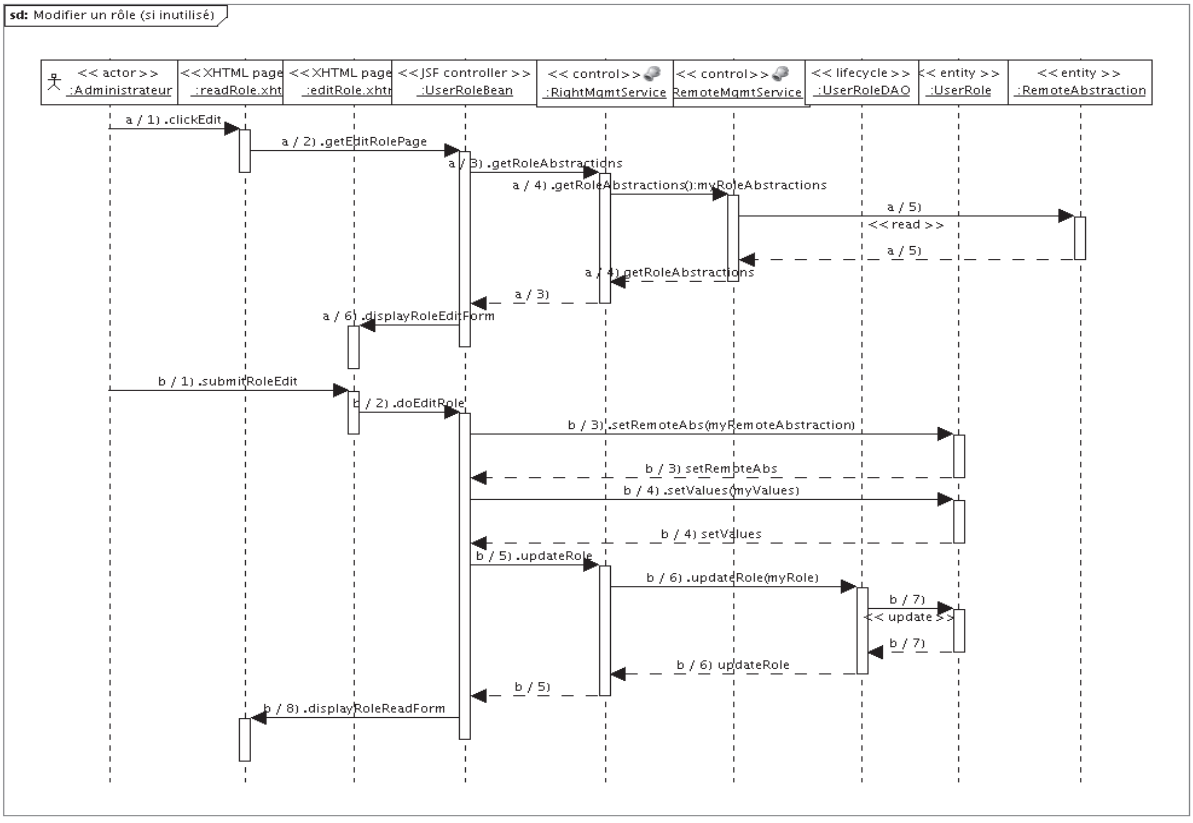

Figure 61 – Modifier un rôle (si inutilisé)

La manipulation des systèmes distants et des abstractions se fait au travers des couches suivantes :

- RemoteSystem, RemoteAbstraction : les entités concernées
- RemoteSystemDAO, RemoteAbstractionDAO : le DAO des entités concernées
- RemoteMgmtService : le service de gestion des objets distants
- RemoteSystemBean, RemoteSystemAbstractionsMappingsBean : les bean managé JSF concernés
- Pages XHTML

La manipulation des règles de synchronisation se fait au travers des couches suivantes :

- SynchroRule : la règle de synchronisation
- SynchroRule DAO : le DAO des règles de synchronisation
- RemoteMgmtService : le service de gestion des objets distants
- RemoteSystemBean, RemoteSystemAbstractionsMappingsBean : les bean managé JSF concernés
- Pages XHTML

5.5.4. Gestion des droits

La gestion des droits s'articule en 3 parties :

- la création, modification, suppression des rôles (cf. figure 60) et des profiles
- l'affectation des droits aux utilisateurs : si le rôle est associée à une abstraction alors, l'affectation du droits à l'utilisateur va engendrer la propagation d'un ou plusieurs valeurs sur le système distant. L'affectation d'un profil engendre l'affectation de plusieurs profils, et par conséquent l'affectation de droits à un ou plusieurs comptes.
- la traçabilité des droits est systématique et non optionnelle. Une affectation ou un retrait de droit engendre une ou plusieurs lignes de trace.

La manipulation des rôles (cf. figure 61) et des profils se fait via les couches suivantes :

- UserRole, BusinessProfile : les entités concernées
- UserRole DAO, BusinessProfileDAO : le DAO des entités concernées
- RightMgmtService : le service de gestion des objets distants

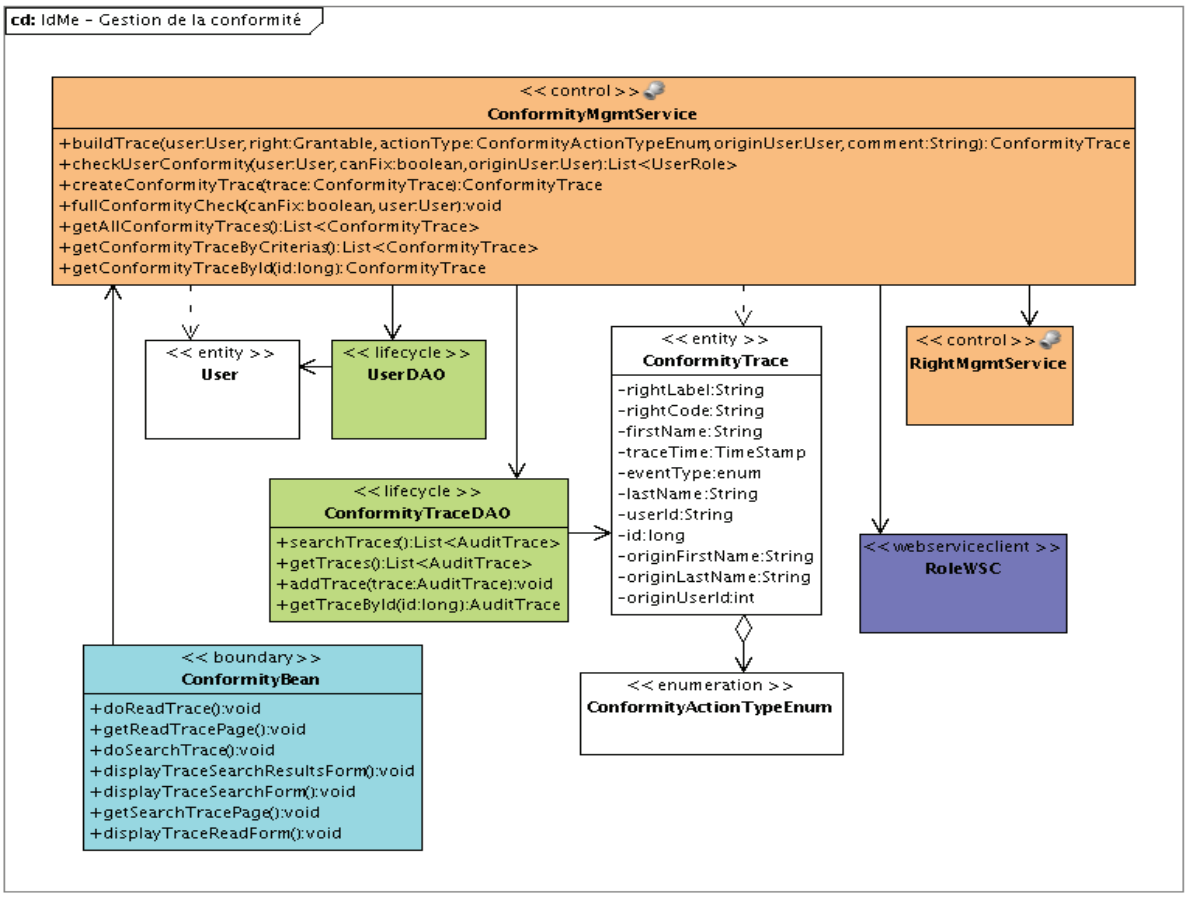

Figure 62 – Gestion de la conformité

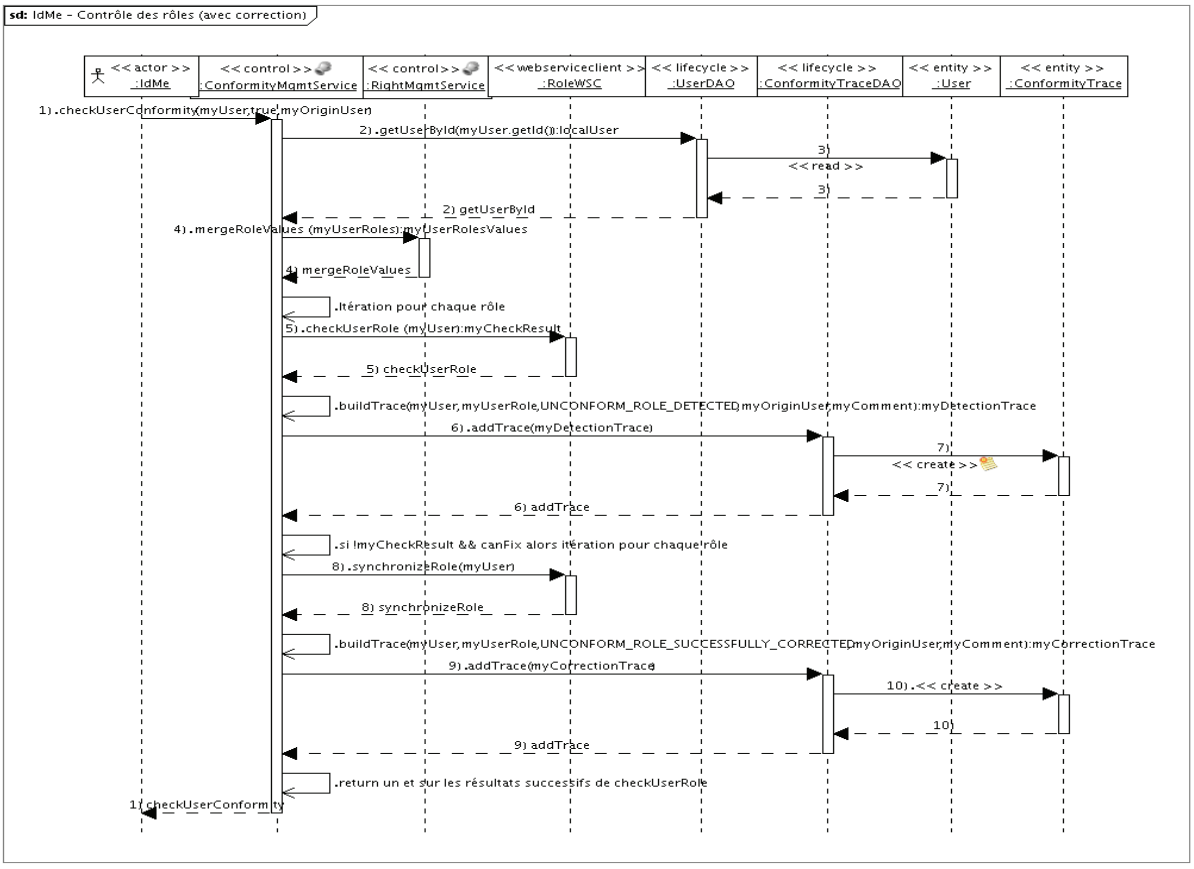

Figure 63 – Contrôle des rôles (avec correction)

- RemoteSystemBean, RemoteSystemAbstractionsMappingsBean : les bean managé JSF concernés
- Pages XHTML

Par ailleurs, les classes UserRole et BusinessProfile ont été regroupées via une interface Grantable toujours selon le principe de programmation par interface [9], qui permet entre autre de simplifier les actions de trace lorsqu'il s'agît d'une opération sur les droits.

5.5.5. Gestion de la conformité

La gestion de la conformité (cf. figure 62) est ici traitée comme un sujet à part entière et non comme une fonctionnalité partielle.

Il est donc désormais possible de lancer une vérification de la conformité pour un utilisateur manuellement au sein de sa fiche, ou bien pour tous les utilisateurs, et dans ce cas, soit immédiatement, soit de manière programmée.

La vérification de conformité (cf. figure 63) pour un utilisateur s'effectuera donc de la façon suivante :

1) évaluation des droits de l'utilisateur (transformation des profils en rôles, etc.)
2) contrôle unitaire des valeurs présentes pour chacune des abstractions côté moteur et côté agent. En effet, le contrôle n'est plus seulement au niveau des rôles mais aussi des abstractions car il faut prendre en compte le fait de pouvoir assigner une même abstraction à plusieurs rôles. L'évaluation des droits va donc jusqu'à fusionner les rôles et évaluer les valeurs à propager.
3) Ecriture d'une ligne de trace dans la trace de conformité pour chaque rôle non conforme.

Le formulaire permettant de lancer la vérification de conformité propose aussi une option de correction des droits. Si cette option est cochée, les droits seront alors mis en conformité et le résultat de la tentative sera aussi inscrit dans la trace.

La vérification pour tous les utilisateurs sera une simple boucle sur l'intégralité de la base.

La manipulation de la trace de conformité se fait au travers des couches suivantes :

- ConformityTrace, User : les entités concernées

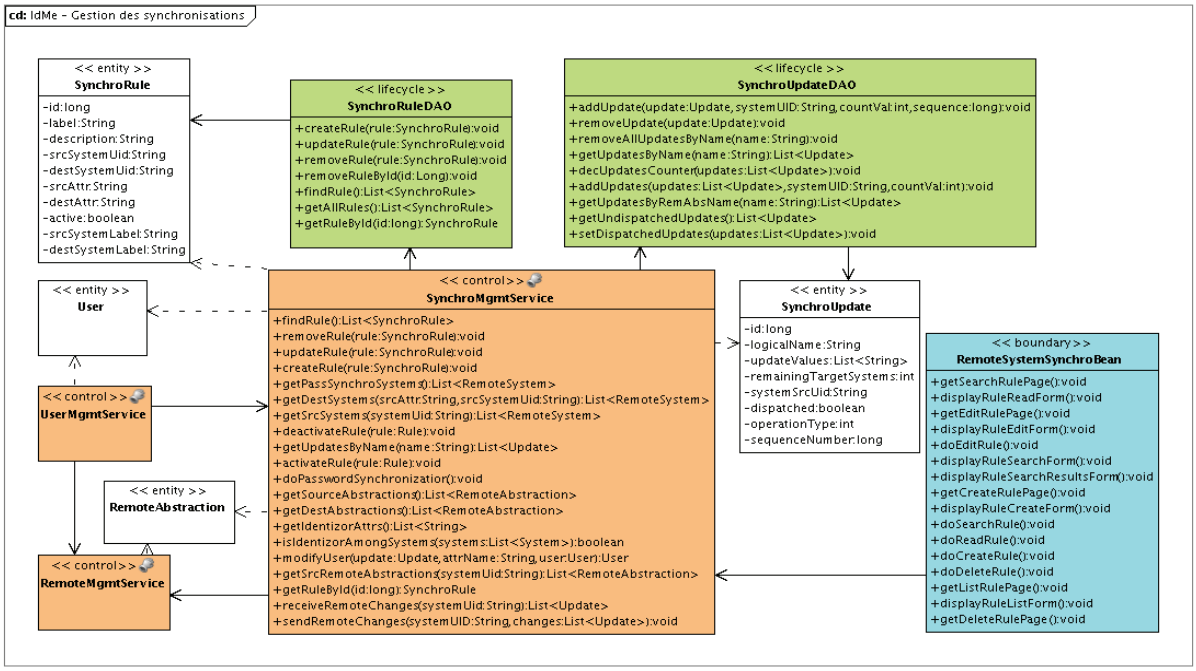

Figure 64 – Gestion des synchronisations

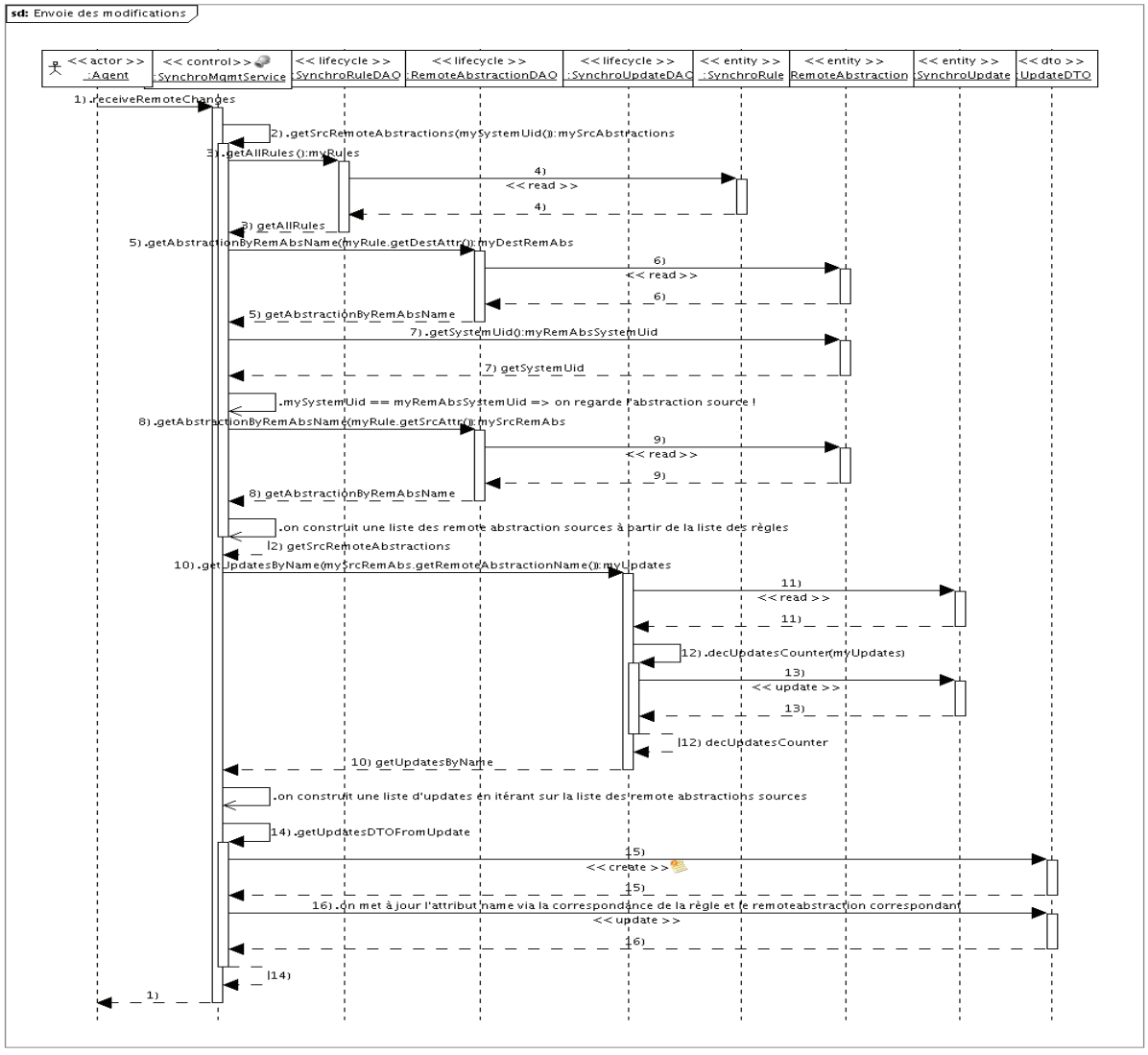

Figure 65 – Envoi des modifications

- ConformityTraceDAO, UserDAO : le DAO des entités concernées
- ConformityMgmtService, RightMgmtService : les services concernés
- ConformityBean : le bean managé JSF de la conformité
- Pages XHTML
- RoleWSC : le web service client permettant d'accéder à

La trace de conformité est consultable à la manière de la trace d'audit et sont toutes les deux regroupées dans un onglet dédié.

5.5.6. Gestion de la synchronisation

La synchronisation est effectuée de 2 façons différentes :

- entre 2 agents
- entre un agent et le moteur

La synchronisation (cf. figure 64) est basée sur les règles. Elles ont le rôle d'aiguilleur. Ainsi, lors de l'activation d'une règle, l'abstraction source est déclarée comme listener auprès du système source, et l'abstraction destination comme publisher auprès du système destination.

Voici les opérations qui sont effectuées lors d'une réception de modifications :

1) filtre sur les utilisateurs (seuls les modifications pour des utilisateurs existants dans le moteur sont conservées)
2) création d'un numéro de séquence
3) construction d'une liste de systèmes destinataires en fonction des règles ayant pour source le système dont les modifications sont issues
4) les modifications sont persistées dans la base du moteur
5) les agents distants sont notifiés. Si un agent distant n'est pas joignable, le moteur essaie de nouveau via le service de Timer JEE

Lorsqu'un agent distant demande aux modifications (cf. figure 65) en fonction d'un numéro de séquence, elles sont servies, et si le système est le dernier à les lire (compteur), les données sont supprimées du moteur car elles n'ont plus lieu d'être.

La synchronisation se fait au travers des couches suivantes :

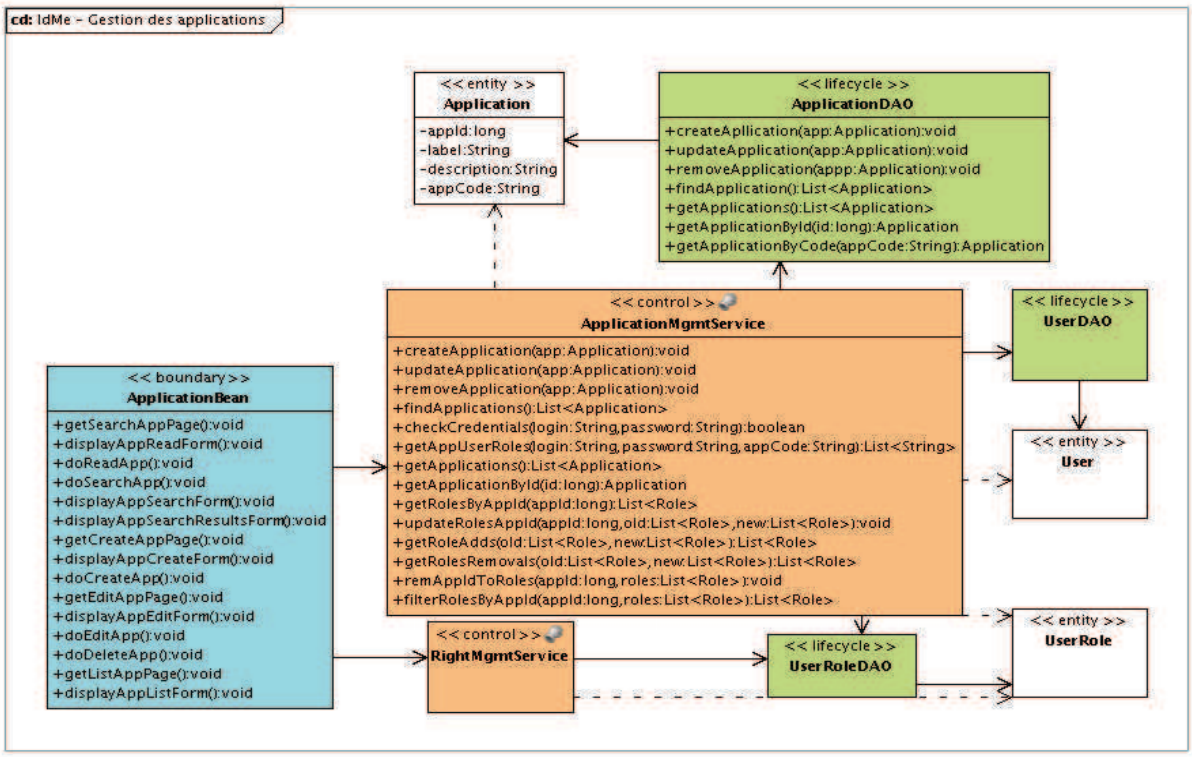

Figure 66 – Gestion des applications

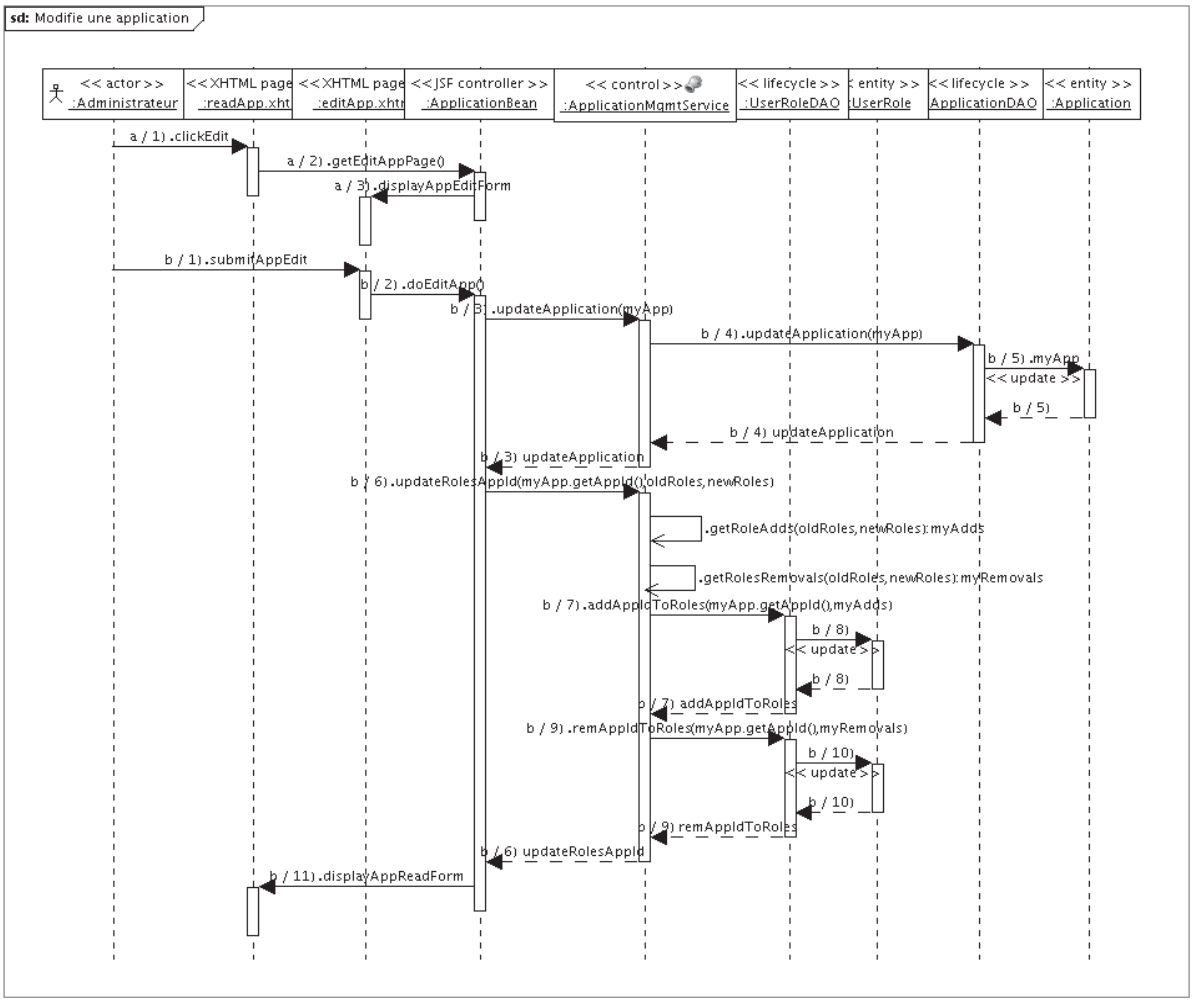

Figure 67 – Modifie une application

- ConformityTrace, User : les entités concernées
- ConformityTraceDAO, UserDAO : le DAO des entités concernées
- ConformityMgmtService, RightMgmtService : les services concernés
- ConformityBean : le bean managé JSF de la conformité
- Pages XHTML
- ConfigWSC : le web service client permettant d'accéder à la gestion des abstractions d'un agent

5.5.7. Gestion des applications

La gestion des applications (cf. figure 66) est découpée en 3 parties :

- la création, modification (cf. figure 67), suppression des applications : il s'agît de la gestion des entités et de leur persistance, ainsi que de l'association avec des rôles applicatifs
- l'authentification des utilisateurs d'applications clientes
- le SSO qui concerne les applications déjà inscrites et pour lesquelles l'option SSO est activée.

Le processus d'authentification se déroule de la manière suivante :

1) l'application s'authentifie auprès du moteur avec un login et un mot de passe propre à chaque application qui est renseigné dans le fichier web.xml de l'application web, elle obtient un jeton applicatif
2) l'application soumet l'identifiant et le mot de passe de l'utilisateur ainsi que son jeton applicatif au moteur, elle obtient un jeton utilisateur si elle est inscrite au SSO ou un code erreur certifiant que l'utilisateur est authentifié sinon.
3) l'application demande les rôles de l'utilisateur au moteur et les transmet au module d'authentification. L'application vit ensuite un cycle applicatif normal.

Les conditions pour que le SSO ou l'authentification IdMe fonctionne sont les suivantes :

1) le module SSO doit être installé sur le serveur applicatif
2) le code applicatif et le mot de passe associé doivent être renseignés dans le fichier web.xml de l'application dans les paramètres appCode et appPass
3) pour Tomcat, le type d'authentification devra être basculé de FORM à IDMEFORM
4) pour GlassFish, le paramètre httpservlet-security-provider du fichier sun-web.xml doit être renseigné à MySAM

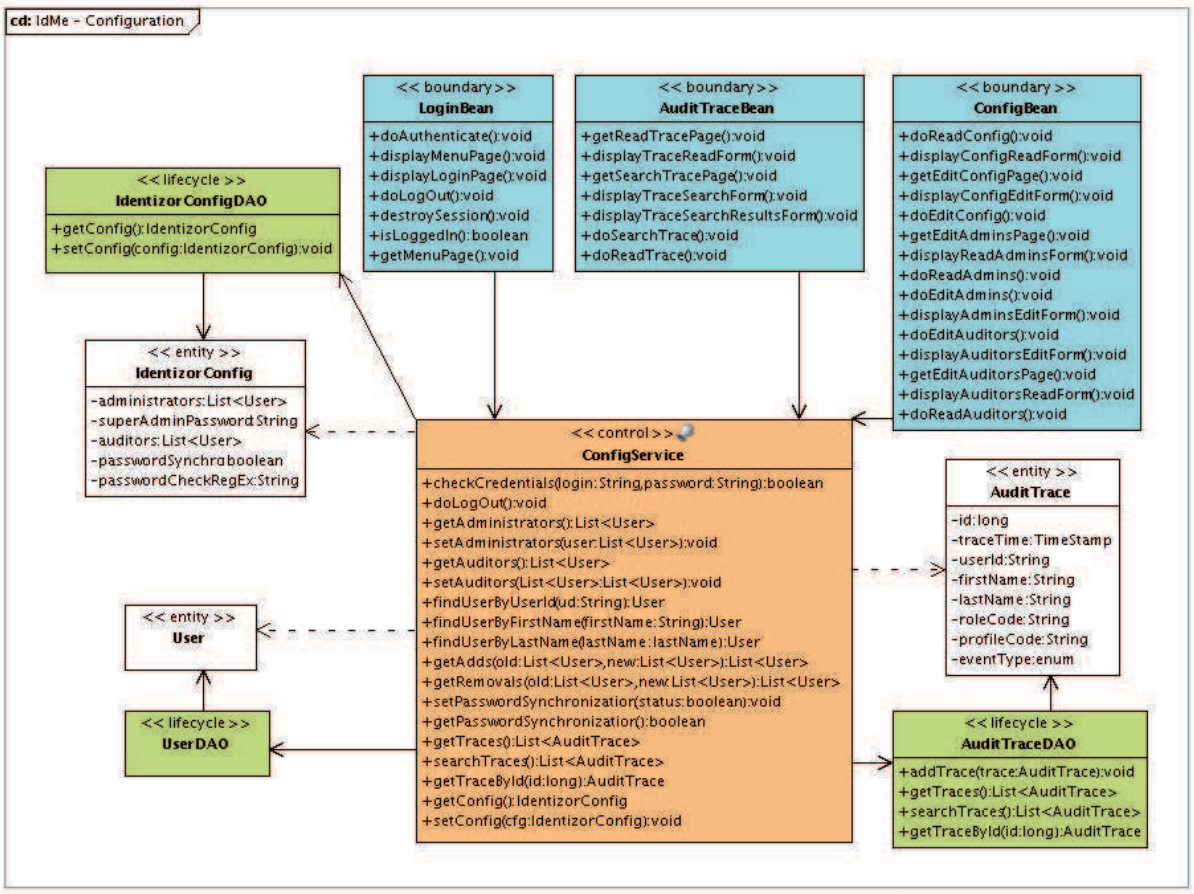

Figure 68 – Gestion de la configuration

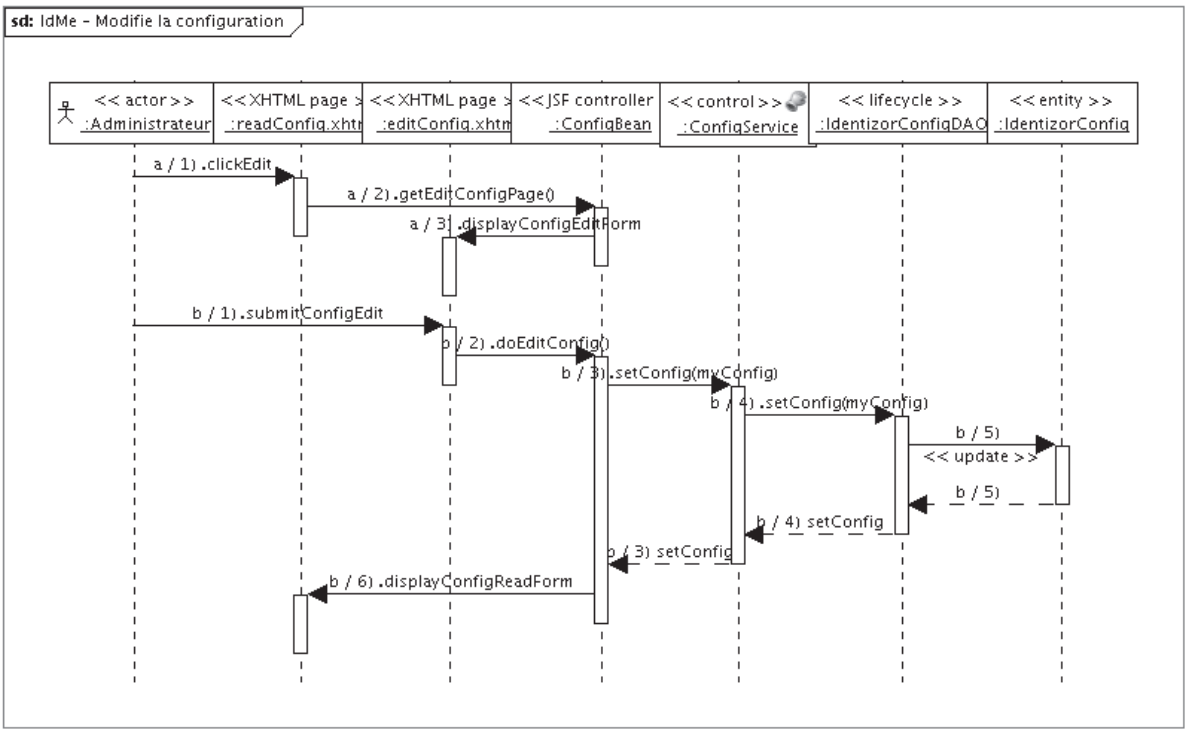

Figure 69 – Modifie la configuration

5) le moteur IdMe doit être démarré et joignable depuis la machine où l'application web est installée.

5.5.8. Gestion de la configuration

La gestion de la configuration (cf. figure 68) regroupe toutes les informations propres au moteur ou regroupant des options d'ordre plus général. On y retrouve les informations suivantes :

- identifiant de connexion pour les agents
- mot de passe de connexion pour les agents
- le super administrateur
- les administrateurs
- les auditeurs
- activation/désactivation de la synchronisation des mots de passe
- expression régulière de contrôle des mots de passe des utilisateurs (moteur)
- durée de validité d'un jeton applicatif
- durée de validité d'un jeton utilisateur

Ces informations sont lisibles et modifiables (cf. figure 69) par un super administrateur ou un administrateur.

6. Architecture et technologies utilisées

L'architecture choisie est web. Ceci pour plusieurs raisons :

- la plupart des logiciels actuels (Sun, Oracle, IBM, BMC, etc.) sont de même type (voire la quasi totalité à l'exception d'Evidian AccessMaster).
- une application peut être plus facilement déplacée d'une machine à l'autre
- il n'y a pas besoin de connaître l'environnement système mais seulement celui du conteneur applicatif
- une grande partie des optimisations peut être faite par des administrateurs, qu'il s'agisse de la mémoire, du processeur, de l'espace disque, etc.

L'application s'articule donc en 3 parties ou sous-systèmes :

- le moteur
- l'agent

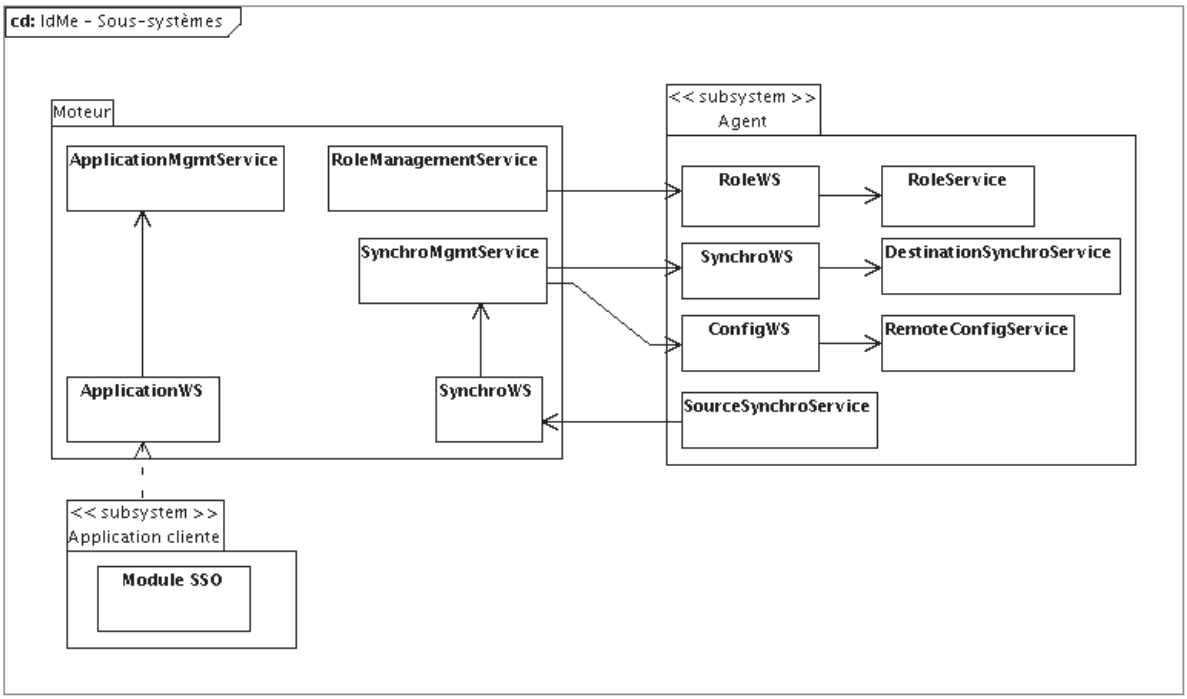

Figure 70 – Sous-systèmes d'IdMe

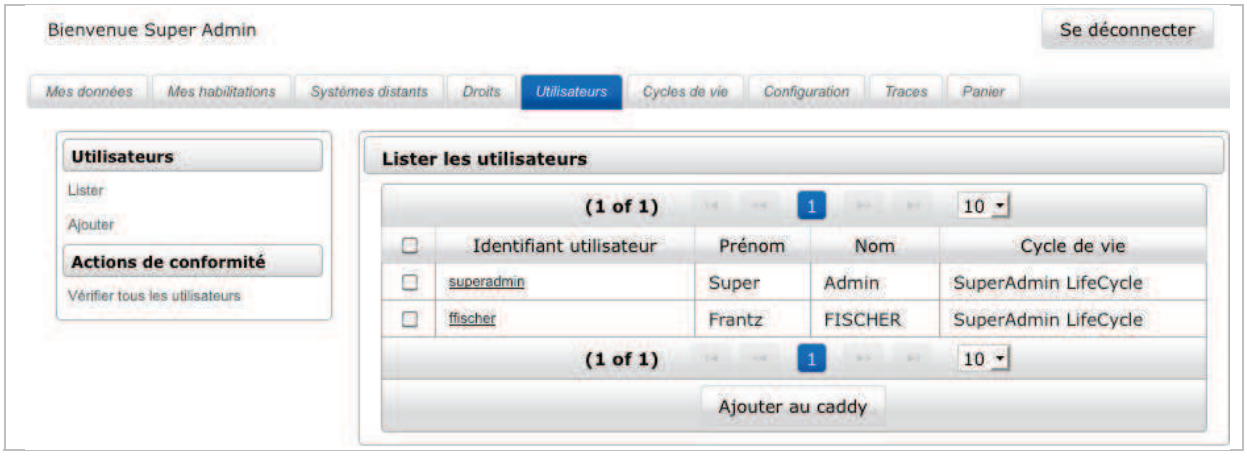

Figure 71 – Couche de présentation, liste des utilisateurs, IHM serveur

Figure 72 – Couche de présentation, vérification de conformité, IHM serveur

- les modules SSO

La liste des sous-systèmes est décrite dans la figure 70).

Le moteur comme l'agent utilisent le modèle MVC (Model View Controller, voir java.cnam.fr (MVC Web et JSF)) :

- Model : inclut la persistance, logique
- View : la couche de présentation (cf. figure 71)
- Controller : la liaison entre Model et View

6.1. Serveur d'application

6.1.1. Agent

L'agent est une application optimisée pour Tomcat 6.x. Il s'agissait à la base d'obtenir une application en une seule archive, qui peut être déployée facilement et rapidement, tout en consommant des ressources raisonnables. De plus Tomcat est un serveur largement utilisé, dont la version 6 (cf. tomcat.org), sortie depuis février 2007, a bénéficié de nombreux correctifs (cf. Changelog de Tomcat 6 - tomcat.org). Tomcat est donc un serveur stable. Pour la raison inverse Tomcat 7 a donc été écarté.

6.1.2. Moteur

Le moteur est découpé en deux parties :

- la partie regroupant la gestion des objets métier, de la persistance, des services, et des services web
- la partie gérant la présentation sous Tomcat 6 (cf. figure 72)

Ce projet étant développé dans l'optique d'un projet de plus grande envergure, il est nécessaire d'anticiper les technologies utilisées, et par conséquent, de choisir un serveur d'application le plus récent possible. Ceci toujours dans une optique de stabilité. Le choix était à l'époque assez restreint puisque GlassFish était le seul serveur d'application à supporter JEE6 (sorti en décembre 2009). Le seul autre serveur était jBoss, mais son stade d'évolution étant en version beta, la stabilité n'était donc pas satisfaisante.

Par ailleurs GlassFish apporte donc toute la pile de services disponibles avec JEE6, et notamment les EJB 3.1, JPA2, une architecture éprouvée (pools d'EJB, de ressources, etc.)

6.2. Persistance des données

La persistance des données est dans l'agent comme dans le moteur au standard JPA (cf. [14]) dans sa version 2 (cf. « What's New and Exciting in JPA 2.0 » sur jazoon.com).

JPA signifie Java Persistence API et possède les avantages suivants :

- permet de travailler avec des POJO (Plain Old Java Object) et peut donc les instancier et y accéder directement
- créer des requêtes
- rechercher des objets
- synchroniser et insérer des objets dans une base de données
- fournir un mécanisme de cache des données
- fournit une intégration avec les services de transaction (par exemple JTA)
- une modélisation du domaine métier via l'héritage et le polymorphisme
- une correspondance Objet/Relationnel
- JPQL, un langage objet proche du SQL permettant de manipuler les entités

JPA (cf. javapassion.com) propose des annotations qui permettent de déclarer les types des champs d'une entité qui seront traduits par des colonnes dans une table SQL. Ainsi, il n'est plus nécessaire d'avoir à maintenir un schéma car l'outil de correspondance Objet/Relationnel (ORM = Object Relational Mapping) qui implémente le standard JPA, se charge de maintenir les tables et leurs données automatiquement.

JPA2 via même jusqu'à proposer des requêtes JPQL typées utilisant la généricité, et permettant d'avoir un code propre. Par exemple voici une requête typée qui permet de d'évaluer les utilisateurs à activer pour la première fois :

TypedQuery<User> query = lem.createQuery("SELECT u FROM User u WHERE u.draft = FALSE AND u.firstActivation = FALSE AND u.beginDate <= :dateToCheck", getEntityClass());

Malgré tout, l'application de la généricité n'est pas complète et il reste certains morceaux d'API comme l'utilisation de requêtes natives qui ne permettent pas de respecter à 100% la généricité de Java. Par exemple, la requête suivante qui permet d'évaluer les utilisateurs pour lesquels la date de fin est renseignée :

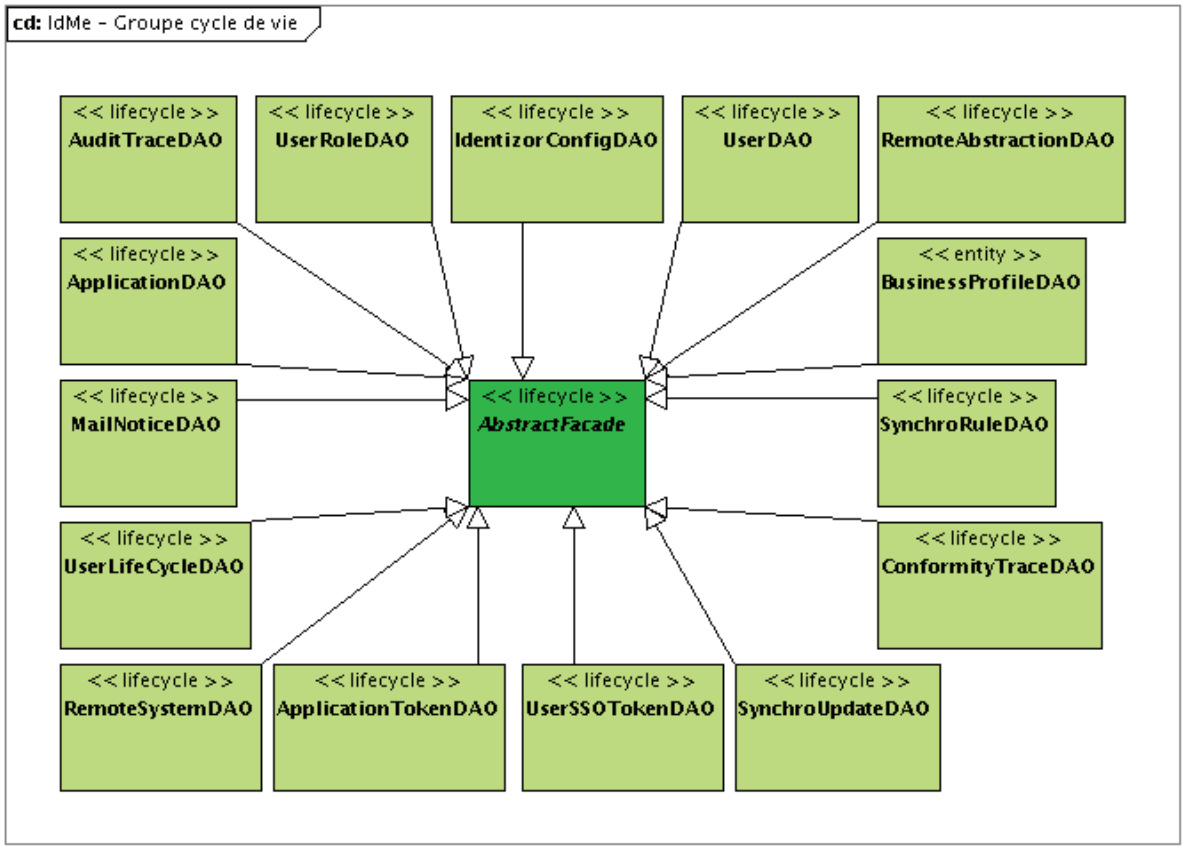

Figure 73 – Couche de persistance du moteur

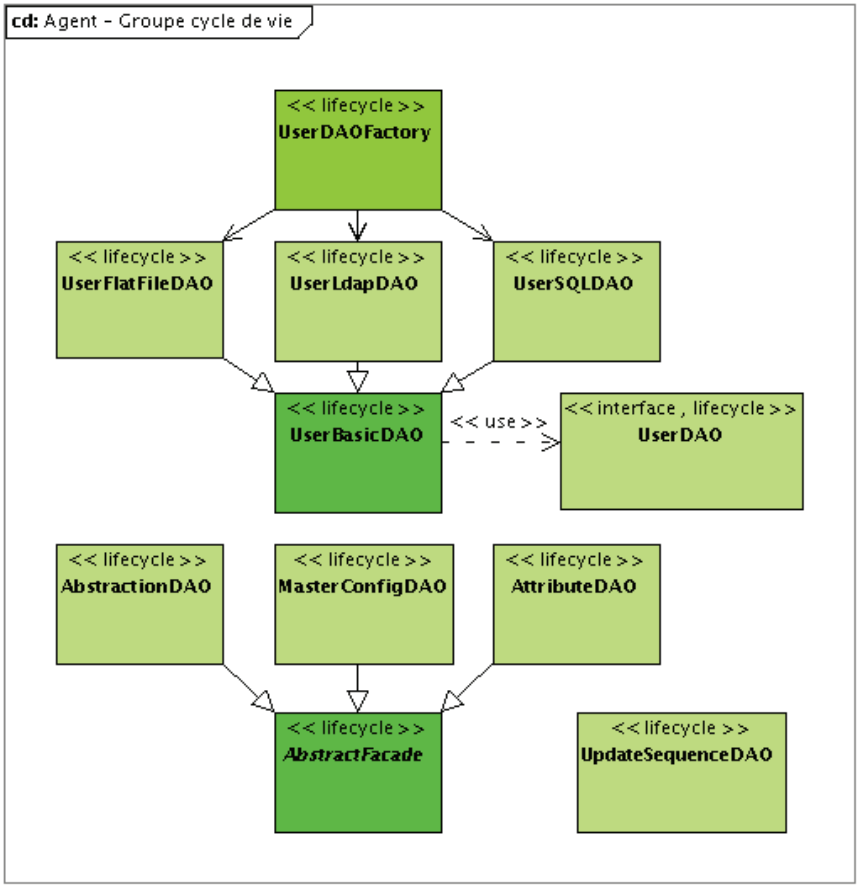

Figure 74 – Couche de persistance de l'agent

*Query query = lem.createNativeQuery("SELECT * FROM USERTABLE u WHERE u.endDate IS NOT NULL") <= ?", getEntityClass());*

Ceci engendre des avertissements que l'on peut supprimer avec l'annotation @SuppressWarnings("unchecked"). La persistance des données utilise donc JPA mais les services n'y accèdent pas directement comme dans certaines architectures. L'approche ici est différente, au sens où l'on conserve une couche DAO. Ceci permet de garantir une certaine abstraction et de pouvoir à tout moment changer de système de persistance, aussi bien d'un ORM à un autre, que d'utiliser un système de persistance complètement différent. J'ai d'ailleurs pu vérifier cet avantage lors de la refactorisation de la couche de persistance de l'agent (cf. section 7.8 sur la refactorisation).

Enfin, l'implémentation de la persistance de l'agent comme celle du moteur est basée sur le pattern « Abstract Facade » (cf. « Facade Pattern » [11]) qui permet de proposer une classe abstraite regroupant les méthodes communes à tous les DAO. Ceci permet d'augmenter la lisibilité et de faciliter la maintenance tout en réduisant la quantité de code à produire (cf. figures 73 et 74).

6.2.1. Agent

6.2.1.1. Stockage des données

Le stockage des données de l'agent devait permettre d'embarquer les données dans l'application elle-même pour respecter la contrainte de déploiement unique et avoir un produit à un seul endroit. De plus, la base devait être si possible open source. Par conséquent le choix des bases de données du marché devint restreint. Les trois principales étaient HyperSQL (cf. hsqldb.org) et SQLite (cf. SQLite.org) et Derby. Cette dernière basée sur Cloudscape a été intégrée au sein de la JDK suite aux demandes d'IBM. Néanmoins, selon l'étude de Pahr et Morken (cf. « APACHE DERBY SMP SCALABILITY » sur folk.ntnu.no) l'implémentation du multithreading ne serait pas performant. J'ai donc préféré écarter ce produit. HyperSQL comme SQLite sont utilisées dans des produits connus comme Firefox ou OpenOffice. SQLite n'étant pas codée en Java, elle ne pouvait être hébergée au sein de la même JVM que l'application web. HyperSQL fut donc choisie. La version actuelle est la 2.1.0 et apporte un nombre conséquent d'améliorations telles que le support multithread, les procédures stockées, l'optimisation des requêtes, etc.

6.2.1.2. Outil de persistance

L'outil de persistance est l'ORM Hibernate (cf. [10]) dans sa version 3.6. Avant la version 3.6 il était nécessaire d'utiliser le « Session Factory Pattern » (cf. lalilуna.de) qui permettait de garantir qu'une seule instance de session était présente par thread.

Cette nouvelle version apporte la gestion des entités par l'entity manager et garantit une gestion thread-safe des opérations. La notion d'entity manager est aussi présente dans le moteur au sein de JEE6. Ceci permet de limiter le nombre d'outils différents à connaître. La seule différence étant qu'il est instancié via injection de dépendance alors qu'avec l'agent il faut l'instancier manuellement. Pour s'assurer que l'entity manager est instancié une seule fois, on utilise le pattern « Singleton » (cf. [11]).

Par ailleurs, Hibernate fournit deux mécanismes de cache des données et de pool de connexion configurables. Pour chacun de ces mécanismes, plusieurs implémentations sont disponibles.

6.2.2. Moteur

6.2.2.1. Stockage des données

Le stockage des données du moteur devait être effectué par une base SQL open source, suffisamment répandue, fiable, performante et si possible multiplateforme. Les choix a été restreint à deux bases : MySQL et PostGreSQL. MySQL appartenant à Sun, et maintenant Oracle, le choix devint évident.

6.2.2.2. Outil de persistance

L'outil de persistance est l'ORM de GlassFish, fournissant l'accès à JPA2. L'implémentation embarquée est EclipseLink. L'entity manager est déclaré au sein des DAO via une injection de dépendance.

6.3. Couche métier

La couche métier est représentée par les entités de JPA2. Ces entités sont des POJO (Plain Old Java Object) et sont donc de simples objets. Il suffit d'ajouter l'annotation @Entity dans une classe ainsi que la déclarer dans le fichier persistence.xml pour qu'elle soit gérée par le gestionnaire JPA2. Les entités peuvent être sérialisées et détachées, et donc transmises d'une application à l'autre. Les DTO (pattern « Data Transfert Object » cf. [15]) ne sont alors plus nécessaires. J'en ai néanmoins conservé

l'utilité au sein d'IdMe pour certains échanges entre moteur et agent afin de limiter les données transmises. En effet, transmettre une entité amène parfois à envoyer trop de données. Ceci amène donc à une consommation excessive de ressources, ou tout du moins, inutile.

Les entités supportent la validation des beans. Ceci permet via la déclaration d'annotations de spécifier des contraintes sur les champs de l'entité. Par exemple, un champ firstName pourrait avoir les annotations suivantes :

```
@Pattern(regexp = "^[a-zA-Z'àâéèêôùûçÀÂÉÈÔÙÛÇ\\s-]+$"}
@Size(min = 1, max = 50, message = "{validator.firstname}")
@NotNull
@Column(length = 50)
```

Ceci spécifierait que la taille de la colonne dans la base est de 50 caractères, que la valeur ne doit pas être nulle, que sa taille doit être comprise entre 1 et 50 caractères et que son format en expression régulière est « ^[a-zA-Z'àâéèêôùûçÀÂÉÈÔÙÛÇ\\s-]+$ ».

L'avantage du bean validation n'est pas seulement dans sa capacité à facilement et explicitement exprimer des contraintes mais aussi dans les outils fournis pour les faire respecter. Ainsi, il y a deux principaux moyens de les appliquer :

- manuellement en se servant de l'API de validation des beans
- automatiquement avec JPA, lors de n'importe quelle opération.

On peut donc, non seulement garantir la saisie correcte d'une valeur via l'API, mais aussi garantir la cohérence de la base.

6.4. Couche Service

6.4.1. Agent

Etant donné que le serveur applicatif de l'agent est Tomcat, il ne propose aucun service d'EJB, ni d'équivalent. Par conséquent, les services sont des POJO, et sont instanciés au moment où l'on a besoin d'eux.

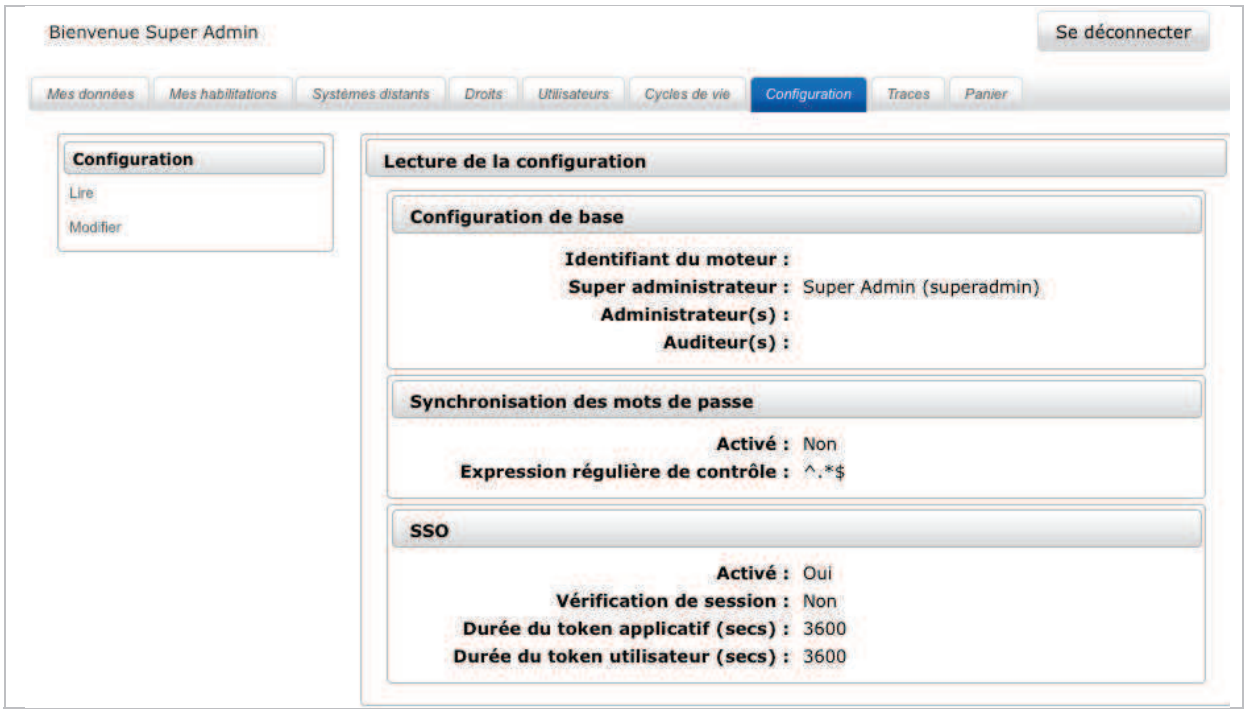

Figure 75 – IHM de configuration du moteur

6.4.2. Moteur

Le moteur est en environnement JEE6 (Java Enterprise Edition 6, cf. « The Java EE 6 Tutorial » sur oracle.com). Il bénéficie donc d'un pool d'EJB stateless et stateful. Les services sont tous des stateless session beans et retourne dans le pool d'EJB après l'exécution d'une méthode. Seule exception, le panier dont la durée de vie est liée à la durée d'une session utilisateur.

6.5. Couche Présentation

6.5.1. Java Server Faces (JSF)

Parmi les principales technologies disponibles pour la couche de présentation, on peut trouver Struts 2, Spring 3, JSF 2 (cf. [18]). Toutes sont disponibles sous la forme bibliothèques qui peuvent être utilisées dans des serveurs applicatifs légers de type Tomcat (conteneur web uniquement) ou bien dans un serveur plus lourd (conteur JEE).

Seul JSF est un standard, il est d'ailleurs intégré au sein de JEE6.

JSF est une évolution de JSP et des servlets. JSF1 introduit la notion de beans, des classes Java annotés à la façon de JPA dont la durée de vie peut être une session, une page, etc. Ces « JSF managed beans » permettent de prendre en charge une partie des fonctionnalités de contrôleur conjointement avec les services, dans un modèle MVC. Ainsi, on ne doit plus retrouver de code dans les pages de présentation.

JSF2 en plus de nombreuses améliorations et de l'intégration de technologies connexes, a remplacé l'utilisation des JSP par celle de pages XHTML. Le besoin de comportement spécifique dans une page devra alors être déporté soit dans un composant JSF personnalisé, soit dans un managed bean associé.

JSF permet aussi d'effectuer des contrôles grâce à des tags spécifiques. Ceci permet par exemple de limiter la taille d'un champ et d'afficher un message en conséquence. Depuis JSF2 (cf. figure 75), la prise en charge de la validation des beans. Ainsi, les contrôles des champs et la personnalisation des messages d'erreurs ne sont plus en double dans la couche métier et dans la couche présentation. Ceci apporte donc plusieurs avantages :

- un code plus lisible et plus maintenable, car on n'a pas a écrire 2 fois la même chose

Figure 76 – IHM de modification d'un cycle de vie

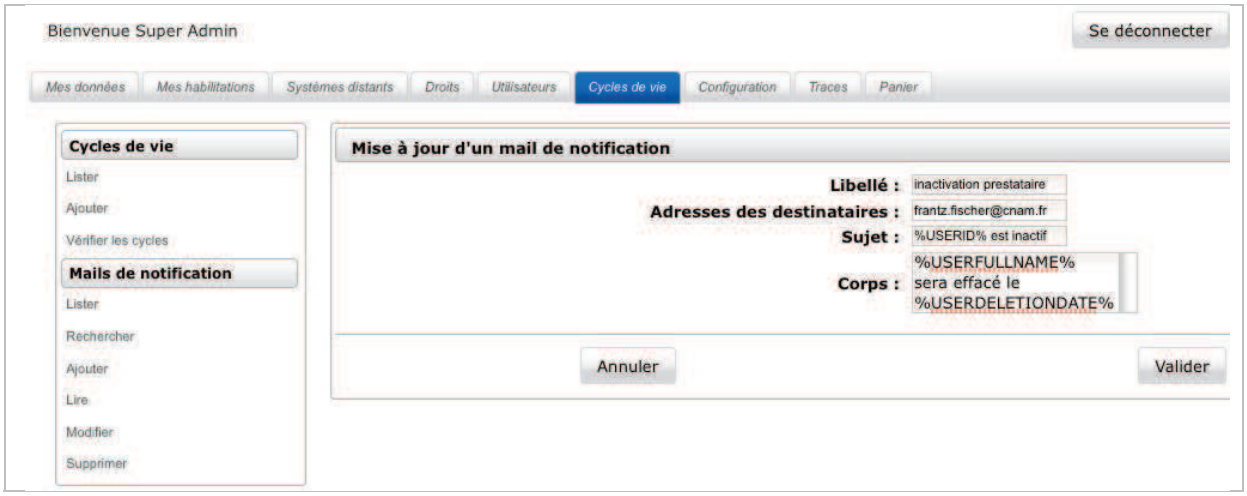

Figure 77 – IHM de modification d'un mail de notification

- un effort de test réduit car les tests unitaires ne sont nécessaires qu'au niveau métier

6.5.2. PrimeFaces

JSF apporte un socle technologique mais nécessite encore beaucoup de travail pour apporter à l'utilisateur final des fonctionnalités riches. Ainsi, l'ajout d'un calendrier, d'une double liste, d'un vérificateur de mot de passe, ne fait pas partie de JSF.

Pour cela on utilise une bibliothèque de composants. Les principales bibliothèques disponibles supportant JSF étaient : Icefaces 2, PrimeFaces 2, RichFaces 4.

Icefaces était en version beta, les performances pauvres, la lenteur de développement, les bugs présents, et le nombre faible de composants disponibles gratuitement ont fait que ce projet prometteur au départ à été écarté.

RichFaces (cf. richfaces.org) est probablement l'une des meilleures et des plus populaires bibliothèque de composants JSF, malheureusement son développement, bien que rapide, était trop tardif.

PrimeFaces (cf. primefaces.org), déjà en version stable durant l'été 2010, apportait l'ensemble des composants présents dans les autres bibliothèques, livré dans un fichier unique contrairement aux deux autres (une douzaine de fichiers). Cette bibliothèque est facile à utiliser, très bien documentée, performante et intuitive. De plus, son cycle de développement est très court et immédiatement testé chez les clients de l'éditeur.

C'est donc cette dernière bibliothèque qui fut choisie (cf. figures 76 et 77). Néanmoins, pour les besoins du composant calendrier, ne supportant pas encore le réglage de l'heure mais seulement de la date, le composant calendrier de RichFaces 4 (version finale disponible depuis avril 2011) est utilisé. Lors de la sortie de PrimeFaces 3.0 (actuellement en 2.2.1), prévue pour l'été 2011, il sera possible de supprimer toutes les dépendances avec RichFaces.

Enfin, il est important de noter qu'on n'utilise pas le mécanisme de navigation des pages JSF à son potentiel maximum. En effet, il n'existe qu'une page unique index.xhtml, étant donné que l'interface est basée sur une série d'onglets. Ainsi, on changera dynamiquement le contenu des onglets et de certaines parties de l'écran pour toujours revenir vers la page index.xhtml. Les états des différentes parties de l'écran sont donc gérés par des beans JSF de durée session.

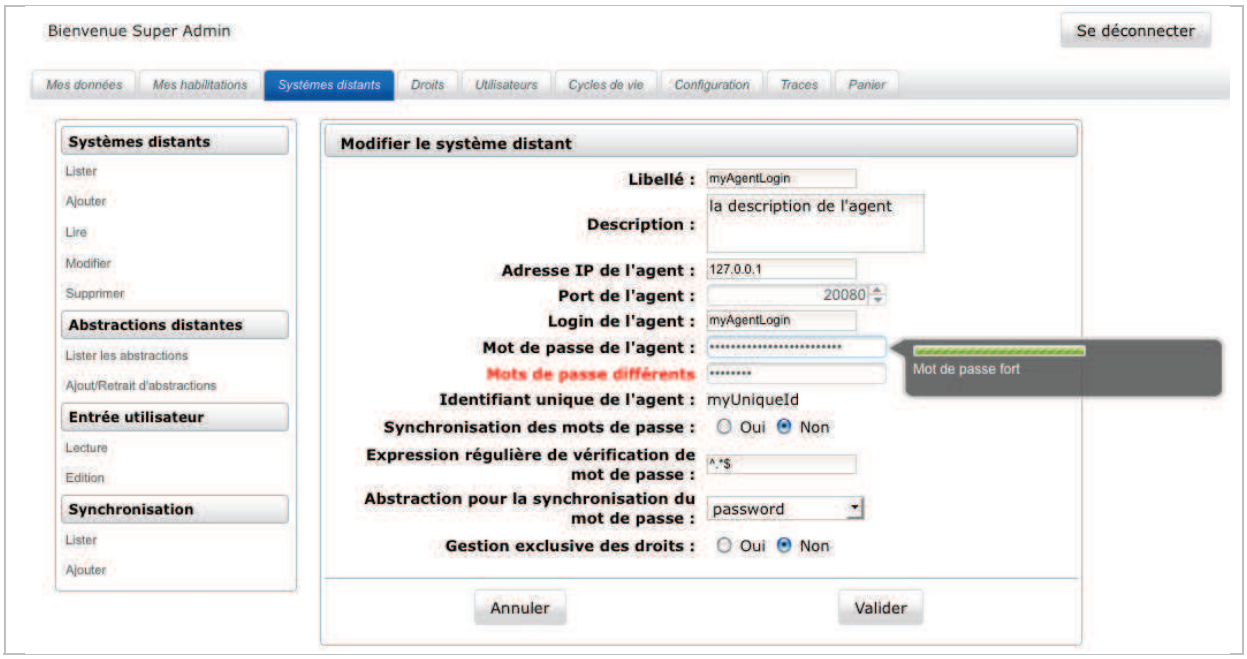

Figure 78 – Modification d'un système distant (mots de passe différents)

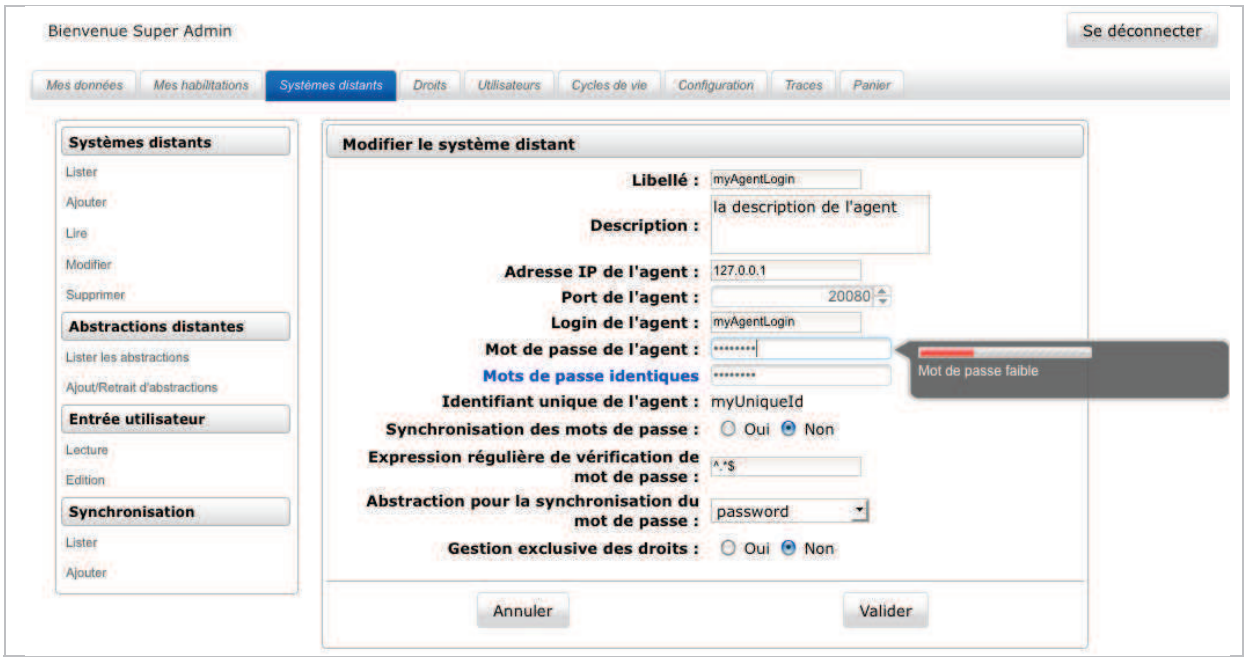

Figure 79 – Modification d'un système distant (mots de passe identiques)

6.5.3. Mise en forme

La plupart du temps, la mise en page n'utilise pas de tag HTML. En effet, JSF propose un mécanisme de grille. Le tag panelGrid permet d'effectuer la mise en page en spécifiant le nombre de colonnes. Il suffit ensuite de disposer les éléments dans l'ordre en respectant le nombre de colonnes. Le moteur JSF va ensuite transformer la page XHTML en page HTML. Le codé généré utilisera le tag table.

Le reste de la mise en page peut ensuite être effectuée à l'aide de CSS (Cascaded Style Sheet). Les CSS (cf. « Beginning CSS » [19]) permette la mise en forme de pages HTML et XHTML. Les tags JSF et implicitement de PrimeFaces supportent aussi l'utilisation de classe CSS (cf. [19]). Ainsi, au lieu de systématiquement et manuellement spécifier qu'un en-tête de formulaire doit utiliser telle taille de police, telle couleur, doit être en gras, etc. ; on spécifie simplement que tel composant utilise telle classe. Les paramètres sont alors centralisés dans un ou plusieurs fichiers CSS. Ceci permettra de faciliter grandement la maintenance puisqu'il suffira de modifier une classe dans un seul fichier pour mettre à jour l'ensemble des formulaires du produit.

6.5.4. JavaScript

Concernant le JavaScript, j'ai évité son utilisation au maximum, étant donné qu'il est plus propre d'avoir des composants JSF ou d'utiliser le tag Ajax de JSF2 et de PrimeFaces. Le temps manquant, il ne m'a pas été possible de creuser cette piste. L'utilisation est néanmoins réduite au minimum. Ainsi, chaque mot de passe est vérifié en format et longueur par le bean validation mais aussi par le composant *p:password*. En effet, on demande d'entrer le mot de passe 2 fois, étant donné qu'il n'est pas lisible. On effectue donc un contrôle de concordance en temps réel via JavaScript grâce à l'affichage d'un message (cf. figures 78 et 79), ce qui évite d'attendre la validation du formulaire mais ne compromet pas la sécurité pour autant (cf. « bean validation »).

6.6. Diverses technologies utilisées dans IdMe

6.6.1. Services web

Les services web utilisent la bibliothèque Metro 2.1 (cf. métro.java.net). Cette bibliothèque est fournie en standard avec GlassFish. De plus, les outils de génération automatique de code pour encapsuler les accès aux services web sont très faciles à utiliser et relativement fiables (aucun comportement inattendu relevé durant le projet).

6.6.2. Remote EJB

Les EJB distants ont été utilisés dans le moteur. En effet, les services implémentent une interface qui est disponible dans l'IHM (Interface Homme Machine) d'administration via un paquetage les contenant toutes.

Ceci est une application du modèle MVC et permet de découpler la couche présentation de la logique métier. Ainsi on pourrait, le cas échéant, développer une autre IHM sous la forme d'un client lourd. Ou bien encore développer des scripts d'administration pour les tâches répétitives.

Le choix de l'EJB et non du web service pour l'administration est dû à la performance. En effet, après plusieurs tests, les accès via EJB sont bien plus rapides qu'à travers des appels aux web services.

6.6.3. Application listeners

Hibernate lors de l'instanciation de l'entity manager a besoin de connaître le chemin d'accès à La base de données. Par défaut, il y accède de manière déclarative via le fichier persistence.xml. Malheureusement, la base est dans une application web, et par conséquent, son chemin est dépendant de l'installation du serveur d'application, ainsi que de l'emplacement de l'application à l'intérieur de ce dernier.

De plus, HyperSQL a besoin de recevoir une commande SHUTDOWN via une requête SQL afin de s'éteindre proprement et de purger ses caches en les persistant. Théoriquement il est possible d'effectuer cette opération en configurant le fichier persistence.xml mais cela n'a pas fonctionné.

Afin de pallier à ces problèmes, on utilise un listener d'application web (cf. [16]). Un listener est une interface java dont les méthodes correspondent à des évènements clés du cycle de vie des objets applicatifs. Ici, j'ai utilisé le ServletContextListener qui contient 2 méthodes :

- contextInitialized : méthode invoquée lorsque l'application est déployée ou démarrée
- contextDestroyed : invoqué lorsque l'application est arrêtée ou retirée du serveur

Ainsi, dans la méthode contextInitialized, on calcule le chemin de la base de donnée et on le transmet de manière programmatique à Hibernate.

La méthode contextDestroyed quant à elle, nous permet d'invoquer la commande SHUTDOWN manuellement afin d'arrêter la base avant l'application.

6.7. Gestion de la synchronisation

Lors de la synchronisation, la notification des agents destinataires de modifications se fait de manière programmée via le service de timer de JEE6 (cf. « Using the Timer Service » sur oracle.com). Ceci permet à intervalle régulier de tenter de contacter l'agent distant lorsqu'il n'est pas joignable.

La tentative de contact de l'agent distant est effectuée à l'aide d'une méthode asynchrone (cf. « Asynchronous Method Invocation » sur oracle.com), nouveauté de Java EE 6. Ceci permet de créer un thread dédié à l'attente d'une réponse de l'agent, et permet aussi au service de timer de nettoyer les données existantes. En cas d'échec, une nouvelle tâche planifiée est lancée. Lorsque celle-ci est échue on renouvelle la tentative.

La réception des modifications par les agents pose un problème. En effet, lorsqu'un agent sollicite le moteur avec un numéro de séquence, ce dernier doit non seulement renvoyer les modifications à l'agent, mais aussi décrémenter le compteur de lecture que chacune des modifications contient. Ce compteur, initialisé avec le nombre d'agents à servir, permet une fois à zéro, de savoir que les modifications n'ont plus lieu d'être et qu'elles peuvent être supprimées. Malheureusement, si deux agents demandent les modifications en même temps, le compteur risque d'être mis à jour de façon imprévisible et les modifications, soit effacées de manière prématurées, soit laissées à l'abandon dans la base, gaspillant ainsi des ressources. Pour palier à ce problème on utilise le pattern de concurrence « Coarsed-Grained Lock » (cf. [15]). Ceci permet de verrouiller une table contenant des verrous. Cette table est une classe encapsulant une table de hachage associant chaque numéro de séquence à un verrou, et proposant des méthodes de réservation d'un verrou et de sa libération. Ainsi, après chaque envoi des modifications, le compteur est décrémenté et le verrou relâché.

6.8. Envoi des notifications pour les cycles de vie

A intervalles réguliers, la vérification des utilisateurs à lieu. Cette opération est lancée grâce au timer de JEE6, utilisé de manière déclarative via l'annotation *@Schedule*.

La liste des utilisateurs étant construite, l'envoi des mails est effectué via l'API JavaMail (cf. « JavaMail API » sur oracle.com) pour laquelle GlassFish offre un service. Ainsi, l'application n'a pas besoin de connaître les détails d'accès au serveur de mail, tout

est au niveau de la configuration du serveur et peut être modifié sans avoir à modifier ou redéployer l'application.

6.9. Authentification SSO et architecture orienté service (SOA)

SOA vient de l'anglais Service Oriented Architecture [12]. Il ne faut donc pas confondre le « faux » SOA qui est communément associé aux web services avec le « vrai » SOA qui lui apporte réellement un service fonctionnel, où chaque couche technique est bien partitionnée. Ceci rejoint les concepts « loosely coupled » (faible dépendance entre les objets) et « high cohesion » (forte cohésion ou spécialisation) que l'on peut retrouver dans les ouvrages de conception objet [13]. Cette approche est d'ailleurs plutôt microscopique alors que SOA est plus macroscopique.

Définition simplifiée (cf. [17])

Un service est un composant logiciel :

- qui est défini par une interface et qui peut être indépendant de la plate forme
- qui est disponible via le réseau
- dont les opérations définies dans l'interface manipulent des objets métiers
- dont l'interface et son implémentation peuvent être décorées (cf. « Decorator Pattern » [11])

Un service peut-être réduit à l'utilisation d'un web service, ou peut être pris en compte globalement dans l'architecture d'une application.

IdMe est donc une architecture SOA où chaque service a un domaine fonctionnel bien défini. De plus, l'architecture va au-delà de son propre domaine puisqu'elle propose des services web d'authentification et de SSO aux applications clientes, en leur offrant une authentification et une gestion des utilisateurs déportées. Les services web utilisés entre le moteur et l'agent peuvent aussi être considérés comme des services externes mais sont plus des outils permettant de faciliter les échanges entre différentes parties d'un même produit.

L'authentification SSO côté moteur est donc basée sur des services web. Côté application cliente l'implémentation de modules pour les conteneurs web est nécessaire. L'utilisation de JAAS (cf. JAAS Reference Guide - oracle.com) viendrait à l'idée presque instantanément lorsqu'on parle d'authentification et de serveur web, seulement il n'est

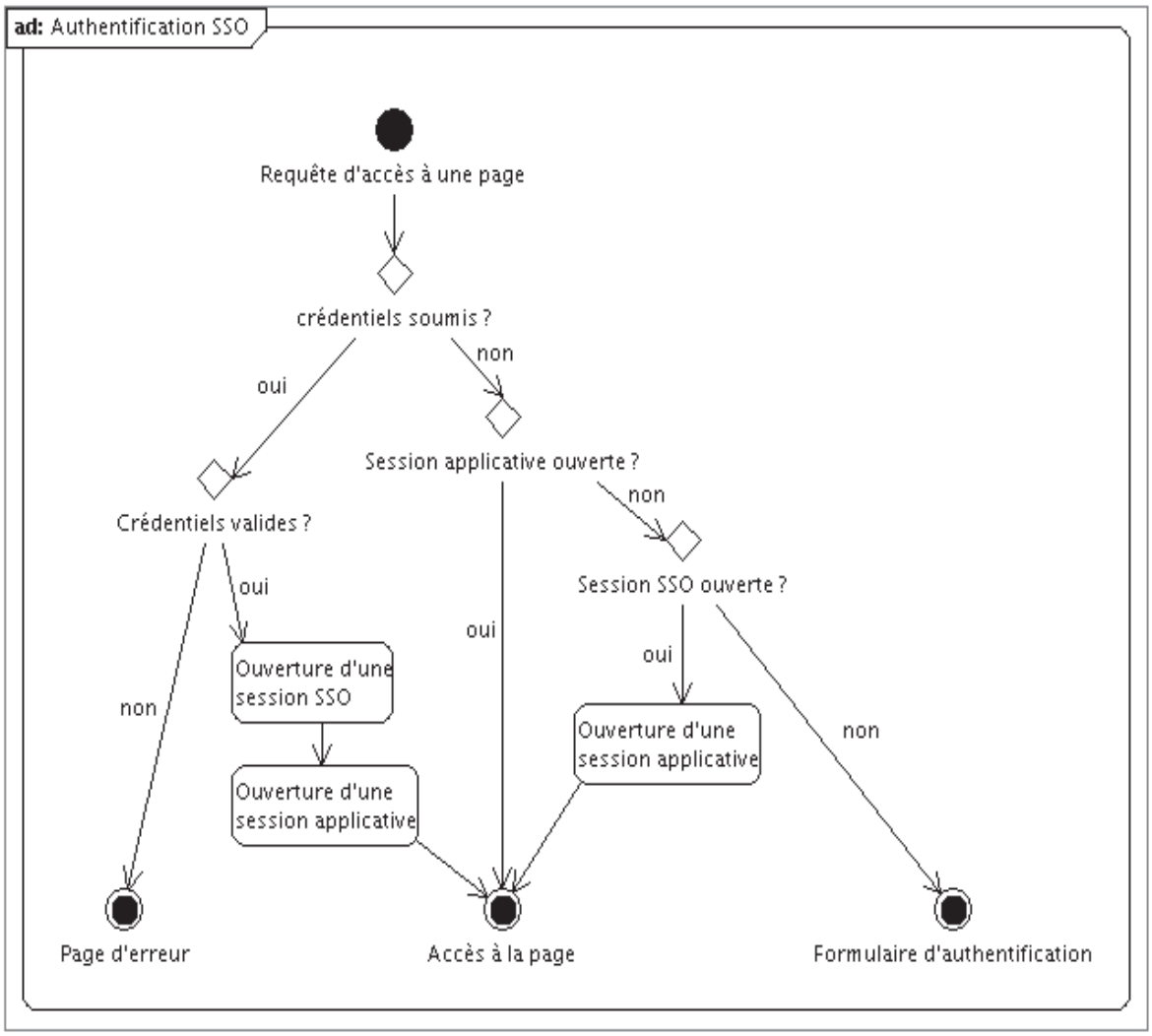

Figure 80 – Mécanisme d'authentification SSO

pas possible d'utiliser ce mécanisme car on va au-delà du comportement de base d'un serveur web. En effet, nous devons gérer la session SSO, et il n'est pas possible d'accéder aux cookies dans un module JAAS, ni d'ouvrir une session sans présenter de formulaire. Par conséquent, il faut implémenter des mécanismes spécifiques à chaque serveur web, des mécanismes au niveau de ceux qui instancient les modules JAAS.

La gestion de la session SSO est effectuée via un cookie dont le domaine est paramétrable, ainsi que le chemin racine (cf. figure 80).

Tout accès à une application est donc vérifié dans l'ordre suivant :

1) Y a-t-il une session applicative ouverte ?
2) Y a-t-il une session SSO ouverte ? (si oui, alors la session applicative est validée)
3) L'identifiant et le mot de passe utilisateurs sont ils correctes ? (si oui, une session SSO, puis une session applicative sont ouvertes)

6.9.1. Module Tomcat

Malheureusement, la documentation Tomcat n'est pas très détaillée concernant l'authentification autre que JAAS. On retrouve les notions de Valve et Authenticator. Après avoir trouvé un exemple d'utilisation d'Authenticator, il était beaucoup plus simple de lire le code source de Tomcat et de comprendre ce qu'il s'y passait. Au final l'authenticator est le chef d'orchestre de l'authentification. Il gère l'enchaînement des pages d'authentification. C'est lui qui est appelé à chaque lecture de page. Il vérifie si une session est ouverte pour l'utilisateur et redirige vers les pages d'erreurs so l'utilisateur n'est pas authentifié.

C'est lui qui gère les Realms (type de stockage des informations de sécurité comme une base de données, un annuaire, ou un fichier plat) et les modules d'authentification (JAAS, SecureId, biométrique, etc.). Par défaut, il existe 4 types d'authentification (classe Authenticator) supportés par Tomcat et détaillés sur le site oracle.com (cf. « Understanding Login Authentication » sur oracle.com) :

- FORM : authentification par formulaire
- BASIC : authentification par identifiant et mot de passe sans formulaire (généralement une boîte surgissant demande les crédentiels)
- DIGEST : authentification basique utilisant une transmission encryptée des informations vers le serveur
- CLIENT CERTIFICATE : authentification par certificat client

L'implémentation de ce module sera décrite dans la section sur la réalisation.

6.9.2. Module GlassFish

Nous retrouvons le même problème dans GlassFish. Il est nécessaire d'utiliser autre chose qu'un module d'authentification ou un realm, car ni l'un ni l'autre ne permet de valider une session sans s'être authentifié préalablement.

La documentation sur ce sujet est quasi inexistante, floue ou manque d'utilité et le code source, bien qu'open source, n'a pas été accessible. Par ailleurs l'authentification de GlassFish respecter la spécification « Servlet Container Profile » du JSR 196 (cf. jcp.org). Malgré cela, les informations glanées sur divers blogs et une page de documentation sur oracle.com constituèrent ma seule source exploitable d'information.

Finalement, l'interface ServerAuthModule est similaire à la classe Authenticator de Tomcat et permet en l'implémentant d'écrire la logique d'enchaînement des pages d'authentification, ainsi que les appels à des realms ou des modules d'authentification.

6.10. Trace applicative

L'outil de logging (trace applicative) choisi est SFL4J (cf. slf4j.org). Il est thread-safe et bien plus performant et moins prône aux deadlock que l'API de logging intégrée à Java. De plus, sa syntaxe est entièrement compatible avec LOG4J et il admet plusieurs backend (couches physiques) :

- LOG4J
- Java JDK logging
- NOP : No Operation (n'écrit pas de logs)
- Simple : dirige les logs vers System.err
- Jakarta Commons Logging
- Logback

La configuration est très simple, il suffit que le fichier jar de la couche physique soit dans le même classpath que SFL4J.

La couche physique choisie est logback. Elle est performante et thread-safe, et se configure très facilement via un fichier XML et toute une série de filtre et d'options sont disponibles. De plus il est possible de reconfigurer les logs à chaud ou même à distance via de MBeans (cf. [20]).

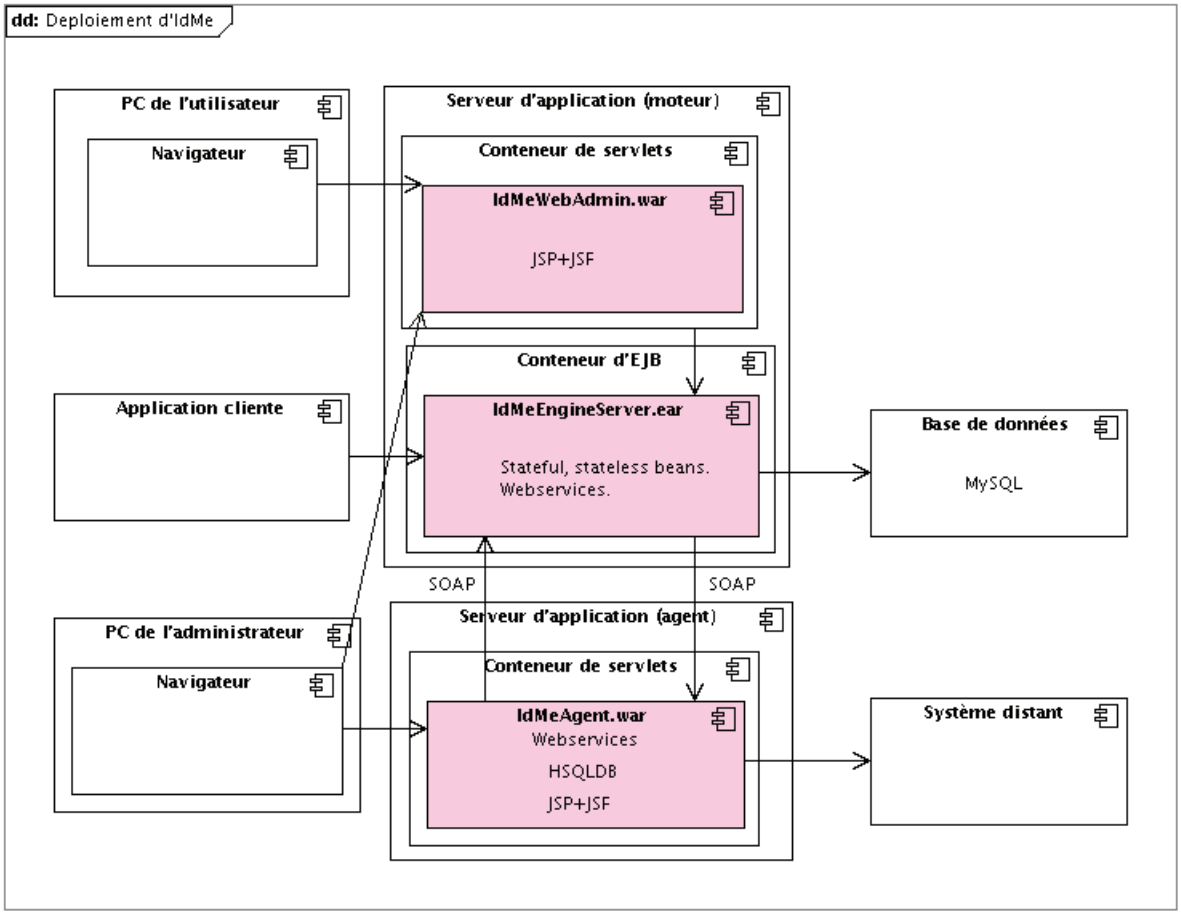

Figure 81 – Déploiement d'IdMe

6.11. Déploiement

Le déploiement du moteur est effectué en deux parties (cf. figure 81) :

- la partie métier contenant les couches métier, persistance, service et service web dans une archive jar : elle est accessible par RMI et par services web
- la partie administration contenant les interfaces des services et la couche de présentation (dans une archive war)

Le déploiement de l'agent est effectué via une archive unique war dans un serveur unique qui peut être soit sur la même machine que le système distant, soit sur une autre. A l'exception de l'agent en mode fichier plat, auquel cas il faut impérativement que le fichier soit accessible.

7. Réalisation

7.1. Environnement de développement

L'environnement de développement intégré est NetBeans (version 6.8 à 6.9.1) pour Mac OSX. Il permet sans configuration particulière :

- une vérification syntaxique de Java, des tags JSF, de JavaScript
- l'auto complétion des langages précités et des CSS
- la gestion des tests
- le débogage pas à pas d'application Java standard mais aussi web
- l'optimisation avec un « profiler »
- la génération automatique et la maintenance de services web
- l'administration simplifiée des serveurs choisis (GlassFish et Tomcat)
- une utilisation entièrement gratuite

Le suivi de version est assuré par subversion, dont le client est complètement intégré à NetBeans et permet l'accès à un repository distant stocké sur un SAN.

Le système de virtualisation VirtualBox a été utilisé pour les machines virtuelles et Ubuntu pour les systèmes distants.

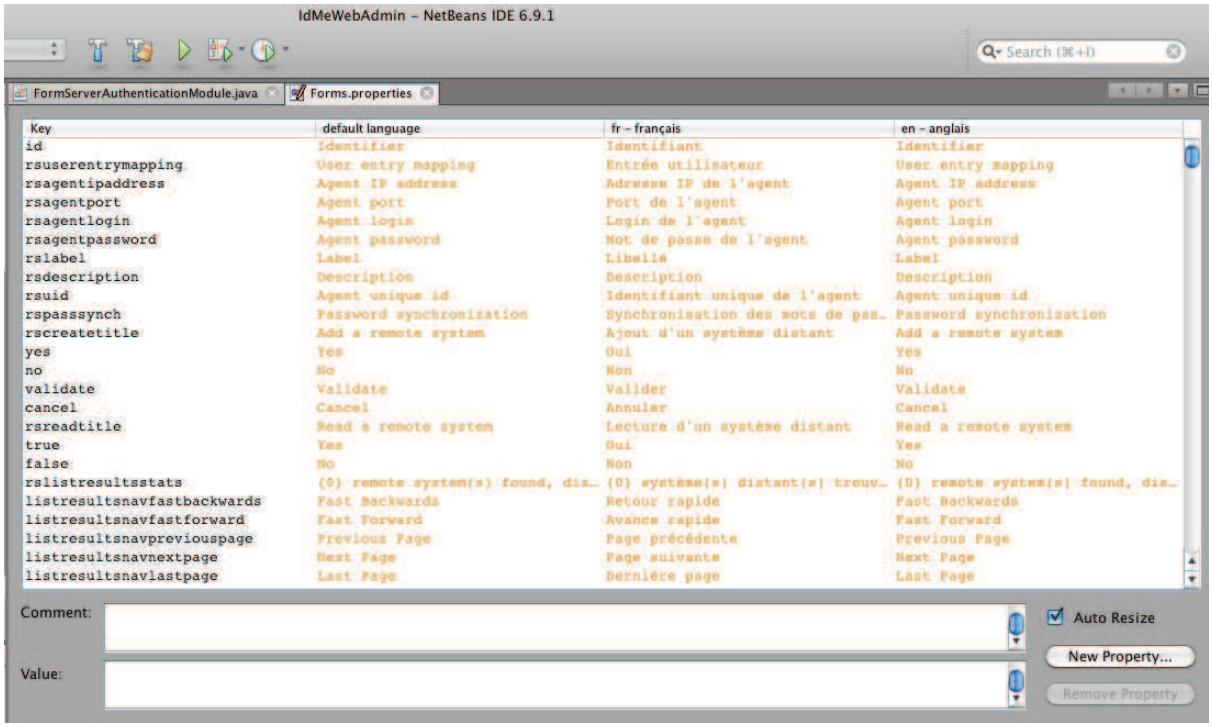

Figure 82 – Outil d'internationalisation dans NetBeans 6.9.1

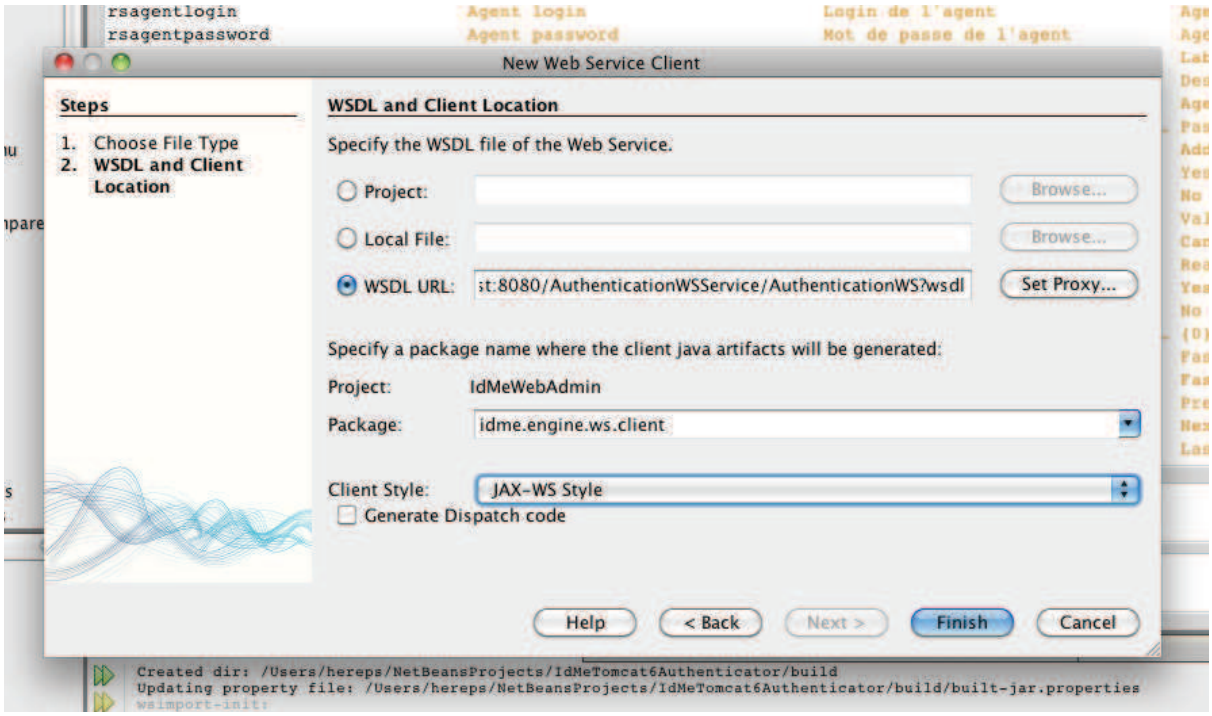

Figure 83 – Outil de création d'un web service client dans NetBeans 6.9.1

7.2. L'internationalisation

L'agent comme le moteur on été réalisés dès le début en prenant en compte l'internationalisation (cf. tutorial sur oracle.com). Ce mécanisme est intégré aux applications Java par la présence de fichiers propriétés facilement éditables via l'éditeur de NetBeans (cf. figure 82) à la manière d'une feuille de calcul tableur (une ligne par propriété, une colonne par langue). Ainsi, changer la langue de l'IHM pour l'utilisateur revient à changer les préférences de langue du navigateur. De plus, pour ajouter une langue au produit, il suffira d'ajouter un fichier de propriétés contenant les traductions nécessaires.

7.3. L'agent

L'agent fut le premier module développé. Il a été réalisé par couche, en commençant par les entités, puis les DAO, les services, les services web, et enfin l'IHM.

La synchronisation a été implémentée dans un second temps mais toujours en suivant le principe des couches, ainsi les entités et autres objets manipulés par la synchronisation n'ont pas été développés avant cette étape.

Enfin, chaque développement de classe a été suivi d'une classe de tests unitaires correspondante.

7.4. Le moteur

Le moteur a été développé après l'agent, en commençant par la manipulation des objets métiers, puis en continuant avec les interactions entre le moteur et l'agent. La mise en place du SSO dut la dernière étape, d'abord côté moteur puis côté module.

7.5. Les services web

Les services web ont été gérés de manière automatique, un outil intégré à NetBeans permettant de construire les interfaces des services web (cf. figure 83).

En effet, éditer les services web à la main est un travail fastidieux. Il est plus rapide de modifier le code d'un service web et de cliquer sur un bouton de régénération automatique, plutôt que d'avoir à éditer un fichier XML à la main.

De même côté client, il est plus rapide de cliquer sur un bouton de génération automatique de web service client, que d'avoir à écrire entre 10 et 20 classes pour prendre en charge l'encapsulation de ce dernier.

editUserRole.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-s
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.prime.com.tr/ui">

    <p:panel header="#{f_msg.ureditttitle}">
        <h:panelGrid width="100%" columns="2" columnClasses="formField,formValue" >
            <h:outputText value="#{f_msg.urlabel} : "/>
            <h:inputText id="userRoleLabel" value="#{userRoles.userRole.label}"/>

            <h:outputText value="#{f_msg.urcode} : "/>
            <h:inputText id="userRoleCode" value="#{userRoles.userRole.code}"/>

            <h:outputText value="#{f_msg.urdescription} : "/>
            <h:inputTextarea id="userRoleDescription" value="#{userRoles.userRole.description}"/>

            <h:outputText value="#{f_msg.ursystem} : "/>
            <h:selectOneMenu id="chosenSystem" value="#{userRoles.selectedSystem}">
                <p:ajax event="change" update="chosenAbstraction"/>
                <f:selectItem itemLabel="#{f_msg.urchoosesys}" itemValue=""/>
                <f:selectItems value="#{userRoles.remoteSystems}" var="rsItem" itemLabel="#{rsItem
            </h:selectOneMenu>

            <h:outputText value="#{f_msg.urabstraction} : "/>
            <h:selectOneMenu id="chosenAbstraction" value="#{userRoles.userRole.abstraction}" conv
                <p:ajax event="change" update="absMultiValued, userRoleMessages"/>
                <f:selectItem itemLabel="#{f_msg.urchooseabs}" itemValue=""/>
                <f:selectItems value="#{userRoles.remoteAbstractions}" var="raItem" itemLabel="#{r
            </h:selectOneMenu>

            <h:outputText value="#{f_msg.ramultivalued}"/>
            <h:outputText id="absMultiValued"
                          value="#{empty userRoles.userRole.abstraction.multivalued ? f_msg[false]
        </h:panelGrid>
```

Figure 84 – Exemple de page JSF (extrait de code)

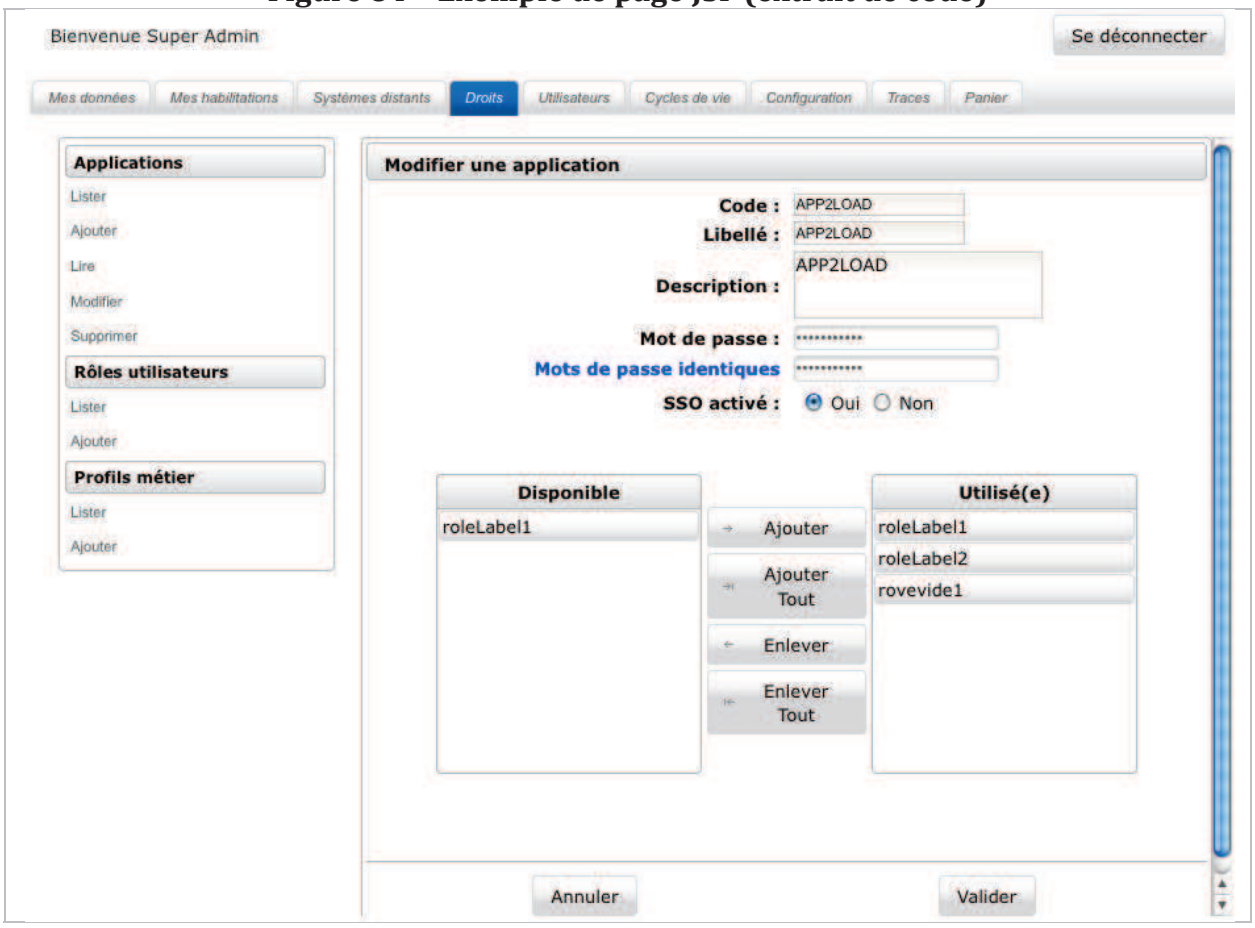

Figure 85 - Exemple de page JSF (rendu)

Il ne faut pas perdre de vue que cet outil fait non seulement gagner du temps de développement, mais aussi augmente la fiabilité car on élimine l'intrusion potentielles d'erreurs d'inattention, Ô combien nombreuses lorsqu'il s'agît d'une tâche rébarbative.

7.6. Les IHM

Les IHM ont été développées en évitant au maximum le code HTML et le JavaScript. Ainsi, j'ai maximisé l'utilisation des tags JSF, ceci augmente la lisibilité et la maintenance des pages, le code étant déporté dans les beans managés de JSF (cf. figures 84 et 85).

Les IHM sont basés sur des onglets, chaque onglet étant un groupe fonctionnel. Par exemple, l'onglet « Systèmes distants » regroupe la manipulation des agents, des abstractions, des règles de synchronisations. Ou bien encore celui des droits qui regroupe les rôles, les profiles, et les applications. Chaque onglet est chargé dynamiquement lorsque l'on clique dessus, ce qui accélère le chargement de la page d'accueil.

JSF avant la version 2.0 avait une réputation de lenteur et de consommation excessive de ressources. La version 2 est censée corriger en partie ces problèmes. Ajouté à cela l'utilisation de bibliothèques telles que PrimeFaces, les ressources consommées sont limitées au minimum. De plus, des produits issus d'Oracle, anciennement de Sun, et d'IBM, utilisaient déjà JSF avant la sortie de la version 2.0. Ces produits étant destinés à des centaines d'utilisateurs voire des milliers, il est raisonnable de penser que la technologie JSF est tout à fait à même de répondre au besoin actuels des entreprises. Néanmoins, il est toujours judicieux d'effectuer une étude de performance, mais il faut dans ce cas obtenir un cahier des charges bien précis émis par le clients chez qui le produit doit être intégré, afin d'adapter au mieux ce dernier.

Par ailleurs, la plus grande partie de l'outil IdMe étant destinée à des administrateurs, le nombre de ces derniers sera restreint et les ressources par conséquent aussi. Seuls persisteront deux onglets pour les utilisateurs afin qu'ils puissent effectuer les changements de données personnelles et de mot de passe.

La prise en main de JSF n'a pas été immédiate. Le temps d'adaptation fut conséquent étant donné que le fonctionnement diffère sensiblement d'une application en servlets et JSP. Le concept de durée de vie des bean managés n'étant pas forcément évident au départ, il est très tentant d'utiliser souvent des beans dont la durée est celle d'une session.

Notamment pour les formulaires où l'on ajoute des valeurs à certains composants. Si la durée de vie d'un bean est celle d'une requête, le composant ne sera jamais mis à jour puisque l'action de cliquer émet une nouvelle requête. Nouveauté de JSF 2, la durée « View » (notée ViewScoped) qui permet de conserver le bean tant que la page est utilisée. Ainsi, recharger une page lorsqu'on met à jour un composant ne détruit pas le bean et permet une mise à jour propre.

PrimeFaces fut assez simple à prendre en main, étant une bibliothèque intuitive et JSF étant un peu mieux maîtrisé. L'efficacité de cette bibliothèque réside dans le fait qu'elle met seulement à jour les parties d'une page qui le nécessite. Ainsi, le fait de cliquer sur un bouton n'engendre qu'un rafraîchissement partiel, sur les zones que l'on déclare à l'intérieur d'un tag. Ceci est très utile pour optimiser le chargement d'une page, et le nombre de bean instanciés.

Enfin, avec PrimeFaces, il est très facile de rendre des actions de composants dépendantes les unes des autres. Par exemple, dans une règle de synchronisation, lorsque l'on choisit un système source, la liste déroulante des abstractions source doit être mise à jour. Ceci est très simplement réalisé en spécifiant dans la première liste la mise à jour la seconde lors d'une modification, soit la ligne suivante (cf. figure 84) :

```
<p:ajax event= "change" update="chosenAbstraction"/>
```

Ceci ne représente que l'un des nombreux avantages qu'une bibliothèque de composants peut apporter en plus du gain sur le temps de développement.

7.7. SSO

L'implémentation des modules d'authentification SSO, côté Tomcat comme GlassFish a nécessité la réécriture de logique d'enchaînement des pages. La documentation sur ces deux serveurs étant quasiment inexistante concernant ce sujet, il a fallut se baser sur le code source pour en déduire le fonctionnement approprié. Par exemple, lorsqu'on essaie d'accéder à une page, le module de gestion de l'authentification vérifie si une session est ouverte, si ce n'est pas le cas, il redirige vers un formulaire d'authentification. Puis, lorsqu'on est authentifié correctement, on redirige vers la page originale, qui demande de nouveau au module d'authentification de vérifier si la session est ouverte. Le module

Figure 86 – Installation du module SSO pour Tomcat 6

ayant sauvegardé les informations d'authentification dans l'accès précédent, on ouvre ensuite une session puis on renvoie un code résultat autorisant l'accès.

Ces modules ont été développés après la mise en place des services d'authentification et de SSO côté moteur, étant tous les deux fortement dépendant de ce dernier.

7.7.1. Tomcat

Le seul reproche que l'on pourrait faire à Tomcat, est de devoir éditer le fichier Authenticators.properties de l'archive catalina.jar, afin d'inscrire le nouveau module d'authentification.

Ainsi, l'installation du module Tomcat pouvant être contraignante, un script Shell (cf. figure 86) a été écrit pour effectuer l'installation. L'administrateur n'a qu'à renseigner les champs demandés et redémarrer le serveur afin d'activer le module.

7.7.2. GlassFish

GlassFish permettant une configuration totale, il n'a pas été nécessaire, bien que possible, d'écrire un script installation.

7.8. Refactorisation (ou « code refactoring »)

La refactorisation (en anglais « refactoring ») permet de modifier le code, de le réévaluer, de l'alléger, de regrouper des classes fonctionnellement associées, de rendre plus spécifiques d'autres. Cette étape vise à améliorer le produit, le rendre plus maintenable, et si possible plus fiable par la même occasion. L'ouvrage « Refactoring des applications Java / J2EE » (cf. [22]) donne quelques lignes directrices dont on peut s'inspirer pour retoucher des projets.

7.8.1. Système de trace (logging)

Le système de logging initialement utilisé était celui de la machine virtuelle java. Il n'était pas très performant, et montrait des problèmes de concurrence et de deadlock. De plus la configuration était intégrée au code alors qu'avec logback elle est déportée dans un fichier XML ce qui est plus simple pour le produit. Le couple SFL4J/logback permet de configurer les logs en fonction des classes et par exemple de regrouper toutes les classes de synchronisation dans un fichier spécifique. Même raisonnement pour les autres parties du produit.

Cette partie était assez fastidieuse car il a fallut retoucher toutes les déclarations de classes. Etant donné qu'elles étaient identiques, un simple script a accéléré le processus.

Par la même occasion, SFL4J a été introduit dans tous les tests afin de pouvoir générer des logs facilement si cela devenait une nécessité.

7.8.2. Couche présentation

La bibliothèque de composants JSF utilisée initialement était IceFaces. Malheureusement elle s'est révélée être une impasse ou bien très lourde dans certains cas. Par exemple, dans le simple cas précité d'une mise à jour de règle. Le fait de demander la mise à jour d'un autre composant nécessitait un code spécifique, en ajoutant à cela un bug du moteur qui bouclait autant de fois que l'on cliquait, ralentissement l'interface au fur et à mesure de son utilisation.

Par ailleurs, la vitesse d'exécution, même en activant le paramètre Production dans javax.faces.PROJECT_STAGE, était décevante. Ceci sans compter les nombreux autres bugs, et la vitesse de progression du produit. Le nombre de composants étant limité, le besoin de changer d'éditeur devenait flagrant.

PrimeFaces étant choisi, il restait le problème du calendrier. Après quelques tests sur le composant calendrier de RichFaces, cette dernière, bien qu'en version beta, répondait tout à fait au besoin.

Il fallut ensuite reprendre chacune des pages XHTML pour utiliser la nouvelle bibliothèque. Par ailleurs, ma connaissance de JSF progressant, j'ai remplacé tout le formatage précédemment effectué à l'aide de tables, par des tags JSF, réduisant presque de moitié la taille des pages.

7.8.1. Couche de persistance de l'agent

La couche de persistance de l'agent était initialement de type JDBC. L'accès à la base de données était donc géré manuellement. L'emprunte mémoire était moins conséquente et l'agent fonctionnait correctement. Néanmoins, l'implémentation de la synchronisation restant à effectuer, c'était le moment de passer à une solution plus standardisée. De plus, ma maîtrise de JPA s'étant amélioré, migrer vers Hibernate n'était plus un obstacle insurmontable. Enfin, la facilité de maintenance des DAO ne s'en trouverait qu'augmentée. Les entités restant à implémenter pourrait alors en bénéficier.


```
23  @Entity
24  public class MailNotice implements Serializable {
25
26      /**
27       * Numéro de version pour la sérialisation
28       */
29      private static final long serialVersionUID = 1L;
30
31      /**
32       * Identifiant numérique
33       */
34      @Id
35      @GeneratedValue(strategy = GenerationType.AUTO)
36      private Long id;
37
38      /**
39       * Le libellé
40       */
41      @NotNull
42      @Size(min=1, max=20, message="{validator.label}")
43      @Column(length=20)
44      private String label;
45
46      /**
47       * Les adresses de destination séparées par des virgules et sans espace
48       */
49      @NotNull
50      @Size(min=1, max=512, message="{validator.mailaddressessize}")
51      @Pattern(regexp="^([a-zA-Z0-9_\\-\\.]+)@((\\[[0-9]{1,3}\\.[0-9]{1,3}\\.
52      @Column(length=512)
53      private String addresses;
54
55      /**
56       * Le sujet du mail
57       */
58      @NotNull
59      @Size(min=1, max=128, message="{validator.mailsubject}")
60      @Column(length=128)
61      private String subject;
62
63      /**
64       * Le corps du mail
65       */
66      @NotNull
67      @Size(min=1, max=4096, message="{validator.mailbody}")
68      @Column(length=4096)
69      private String body;
70
71      /**
72       * Getter de l'identifiant numérique
73       *
74       * @return l'identifiant
```

Figure 87 – Exemple d'annotations dans une entité JPA

Par ailleurs, les tests devaient être réécrits. Ils étaient beaucoup plus intensifs et plus poussés avec les DAO JDBC car la persistance devait être complètement vérifiée. Ce qui n'est plus le cas avec Hibernate et JPA puisque la persistance est gérée via une API qui a déjà été intensivement testée.

Le fait de passer sous Hibernate m'a donc permis de :

- bénéficier de la validation des beans, et donc des annotations (cf. figure 87). Avec la version JDBC des DAO, il fallait gérer toutes les valeurs et exceptions à la main, ceci engendrait des tests spécifiques aux DAO. Avec la validation des beans, on procède, comme dans le moteur avec un plan de tests (sur les entités), qui est plus ou moins similaire d'une entité à l'autre. Il est beaucoup plus rapide d'utiliser une annotation que d'avoir à écrire des conditions avec des levées d'exceptions.
- bénéficier d'une persistance simplifiée. On n'a plus à construire une requête, la concaténer, etc. On utilise directement les méthodes de JPA
- bénéficier du gestionnaire d'entités (« Entity Manager ») comme dans le moteur
- bénéficier d'un pool de connexion configurable, sans intervention dans le code, à la manière d'un serveur d'application
- bénéficier d'un cache des données pour augmenter les performances

7.9. Tests

Tous les tests sont automatisés à l'exception des tests d'intégration pour la synchronisation qui ont été effectués manuellement.

L'automatisation a été réalisée à l'aide de jUnit 4.8.2 (cf. [21]).

7.9.1. Tests unitaires

Ils concernent toutes les classes développées pour les entités. Pour chacune des entités les types de test effectués ont été les suivants :

- tests de valeurs valides
- tests de valeurs invalides
- tests de valeurs vides pour les chaines de caractères
- tests de valeurs nulles
- tests de valeurs valides aux limites inférieures et supérieures
- tests de valeurs invalides aux limites inférieures et supérieures

- tests des getters et des setters
- tests des annotations personnalisées

Les tests étant tous assez similaires, on voit rapidement l'avantage de JPA pour élaborer un gabarit de test qui permet d'être adapté à tous les tests, voire même de copier puis coller un test et l'adapter de la même manière.

La pertinence des tests unitaires a été évaluée à l'aide d'EMMA (cf. emma.sourceforge.net). EMMA est un outil de mesure du taux de couverture des tests. Il sert à vérifier le passage des tests dans chaque branche du code produit. Il se présente soit sous la forme d'un outil en ligne de commande, soit sous la forme d'un plug-in. Ce dernier a d'ailleurs été utilisé au sein de Netbeans et permet de générer automatiquement des tests instrumentés qui vont mesurer la couverture.

Le résultat est alors fourni de deux manières :

- sous forme de lignes colorées dans les fichiers source : lignes vertes pour le code parcouru, lignes jaunes pour le reste non visité.
- sous forme statistique en indiquant le pourcentage de code et de classes couverts

Les IHM n'ont pas été testées en couverture car l'outil ne permet pas de tester la partie XHTML. De plus, aucun test n'a été écrit pour les beans JSF étant donné qu'on teste directement les IHM (cf. Tests IHM). Enfin les classes minimales telles que des comparateurs ont été écartées des tests automatisés. En effet ces classes ont un résultat binaire : soit elles fonctionnent soit elles ne fonctionnement pas, et leur comportement est directement vérifiable dans une page XHTML (tri de résultats, etc.). L'effort de test a donc été concentré sur des classes jugées plus importantes.

Le résultat de couverture des classes est de 76%. Parmi ces classes, la couverture de code est aussi de 76%, ce qui est un résultat acceptable. En effet, aller plus loin dans la couverture prendrait de plus en plus de temps et le retour sur investissement serait moindre (cf. « Minimum Acceptable Code Coverage » sur www.bullseye.com).

7.9.2. Tests d'intégration

Les tests d'intégration concernent les DAO, les services et les services web. Il s'agît aussi bien de tests de persistance pour l'accès aux données, que de tests de communication entre le moteur et l'agent. Ces derniers tests ont par ailleurs demandé le développement de quelques outils pour pouvoir, par exemple, démarrer et arrêter un serveur Tomcat, et mettre à jour la base de l'agent.

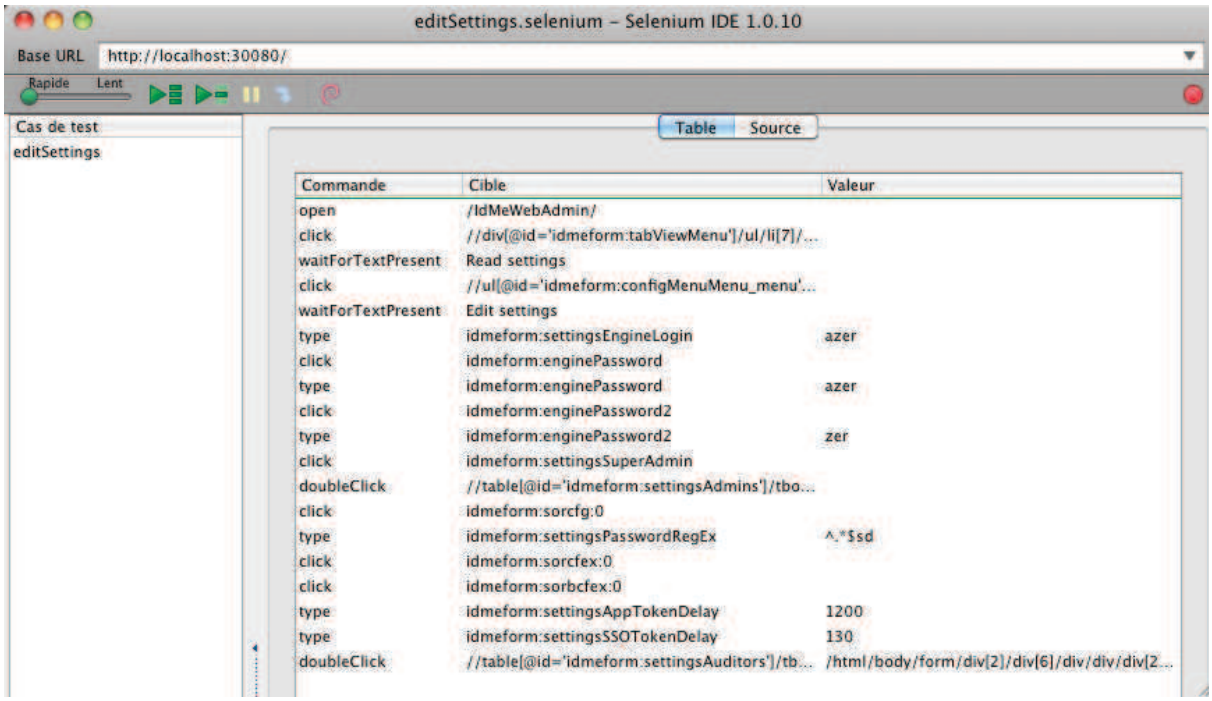

Figure 88 – Editeur Selenium intégré à Firefox

La nouveauté des EJB 3.1, à savoir le conteneur EJB embarqué (cf. « Using the Embedded EJB Container » sur netbeans.org), a été très utile pour les tests du moteur mettant en scène des beans de session. En effet, un test se déroule alors de la façon suivante :

1) instanciation du conteneur embarqué
2) recherche et instanciation des beans
3) tests
4) destruction du conteneur

Avant la version 3.1 il était plus difficile et plus coûteux en temps d'effectuer ces tests puisqu'un serveur d'application devait être démarré et les beans déployés. Les tests étaient donc dépendants d'un environnement préinstallé contrairement à la version 3.1 où les tests peuvent être exécutés d'une machine à l'autre sans retoucher le code ni préparer d'environnement.

Ceci permet donc de limiter la dépendance à l'environnement, seulement il n'est pas possible de l'éliminer totalement puisque IdMe est composé de deux parties : l'agent et le moteur. Ainsi, les tests de communication nécessitent forcément la présence des deux éléments et l'installation par exemple d'un agent dans un serveur d'application pour pouvoir lancer les tests d'intégration du moteur. Néanmoins, de par la légèreté de l'agent, le redéploiement et le temps perdu sont minimaux.

7.9.3. Tests IHM

Les tests des IHM pour le moteur comme pour l'agent ont été réalisés avec jUnit et Selenium (cf. seleniumhq.org). Selenium est un ensemble d'outils permettant de fabriquer des tests qui reproduisent le comportement d'un utilisateur en pilotant un navigateur web. Cet outil peut simuler les cliques de souris, la pression d'une touche de clavier, etc.

On distingue 2 parties :

- le moteur et l'API qui permettent d'exécuter des séquences de commandes :
  Selenium fournit une API java permettant de spécifier les différentes actions d'un utilisateur, et même d'attendre la présence de certains composants à l'écran (liste déroulante, texte, etc.). Cette API accède à un serveur qui pilote une instance de navigateur (dans le cas d'IdMe, il s'agît de Firefox 3.6).
- l'IDE (cf. figure 88) qui permet de prototyper des séquences de commandes : il s'agît d'un plug-in Firefox dans lequel on peut enregistrer une séquence d'actions

- à la manière d'une macro. L'outil convertit par la suite les actions en code Java pour jUnit. Ceci convient tout à fait pour prototyper ou découvrir le fonctionnement de l'API, mais n'est pas suffisant pour réaliser un jeu de test complet. Cet outil sert donc de base à un test qui sera développé directement dans le code Java.

Les tests IHM ont été développés en utilisant le pattern « Page Object » (cf. blog.josephwilk.net ou code.google.com). Ce pattern consiste à encapsuler les accès aux pages dans des objets pages. De plus, dans le cas d'IdMe, la page est toujours la même donc j'ai du adapter ce pattern aux composants utilisés dans la page. On obtient donc des objets onglets et formulaires qui contiennent du code relatif aux API de Selenium. Ainsi, lorsqu'on voudra obtenir un onglet, on appellera l'une de ses méthodes, même chose pour obtenir un formulaire, pour le valider, pour remplir des champs, etc.

Ceci permet alors d'écrire des tests plus simples axés sur la logique du test sans entrer dans le détail de l'implémentation, ce qui augmente la lisibilité et la maintenance, mais aussi accélère le développement du test puisqu'une fois les objets onglets et formulaires écrit, on ne s'occupe plus de Selenium et de sa syntaxe.

Deux difficultés se sont posées lors de la mise au point de ces tests :

- l'apprentissage d'XPath (cf. « XSLT » [23] et dpawson.co.uk) qui permet de désigner un composant dans une page XHTML via un chemin unique. On peut désigner une liste déroulante, un champ de saisie, un bouton, etc. Ceci peut paraître assez rébarbatif au sens où il faudrait rechercher le chemin de chaque composant manuellement afin de l'utiliser dans les APIs, mais il est possible soit d'utiliser l'IDE en mode macro mais la détection est souvent peu fiable, soit effectuer un clique droit sur les composant directement dans la page XHTML. On peut alors recopier le chemin et s'en servir dans les tests jUnit.
- la mise au point des tests étant donné que l'application utilise en permanence des fonctionnalités Ajax (cf. fr.wikipedia.org) et que le traitement de l'information est asynchrone. En effet, les méthodes du « Page Object Pattern » doivent attendre le chargement d'une page avant d'effectuer une vérification ou une action. Pour cela on effectue une boucle d'attente, via un thread mis en veille, qui recherche à intervalle régulier la présence d'un composant ou d'un chaine de caractère.

7.10. Analyse de la performance et examen de la mémoire utilisée

L'analyse de la performance (aussi appelé « profiling » cf. wikipedia.org) effectué dans IdMe est basique, mes connaissances en la matière étant limitées.

Ainsi j'ai procédé à des tests répétitifs tout en surveillant la consommation mémoire grâce à l'outil d'analyse intégré à NetBeans. Celui permet à tout moment de se connecter à une machine virtuelle Java et d'en obtenir des statistiques, ou déclencher un garbage collector à distance. Ainsi, lorsqu'on soupçonne une fuite mémoire, on peut déclencher ce nettoyage et vérifier si la mémoire utilisée est complètement libérée ou si une partie persiste.

7.10.1. Détections de fuite mémoire

Les fuites mémoires recherchées l'ont été via les opérations suivantes :

- lancement des tests IHM de manière successive pour le moteur
- lancement des tests IHM de manière successive pour l'agent
- lancement de la synchronisation de données en modifiant massivement (200 utilisateurs) sur un agent afin de propager les données d'un agent à un autre.

Une seul fuite a été détectée, et de taille puisque 200Mo de mémoire étaient perdus à chaque lancement des tests. Afin de la détecter, j'ai effectué une image de la JVM (« heap dump ») via l'outil de profiling. Cette image montrait les objets qui persistaient en mémoire après un nettoyage. Il s'avérait qu'aucun objet des IHM ne persistait, aucun DAO ni service, ni entités, mais néanmoins des objets Hibernate contenant des chaînes de caractères en rapport avec les entités. Après quelques recherches sur internet, ceci m'a permis de comprendre qu'il s'agissait du gestionnaire d'entités, dont plusieurs instances étaient présentes. La modification de son instanciation en utilisant le pattern « Singleton » a permis de résoudre ce problème.

7.10.2. Optimisation des beans de JSF

Toujours en utilisant la technique précédente, j'ai pu relevé les beans utilisés en navigant d'un formulaire à l'autre, ou bien en validant un formulaire. Ainsi, j'ai pu réduire au maximum les zones devant être mises à jour dans chaque composant, ce qui a permis de diminuer le nombre de beans instanciés par action. Ceci a eu pour effet

d'augmenter la vitesse de traitement, et cela s'est ressenti sur la vitesse de réponse de certaines pages.

7.10.3. Mémoire minimum nécessaire

Afin d'évaluer la mémoire minimum nécessaire, j'ai utilisé la même technique que celle de la recherche de fuite, tout en ayant au préalable augmenté considérablement la mémoire maximum de la machine virtuelle à 4Go. Après avoir effectué plusieurs cycle, sans lancer de « garbage collector », il suffisait de lire la mémoire Java utilisée. L'agent tourne donc avec 160Mo de mémoire et le serveur avec 768Mo.

7.11. Documentation

La documentation est une pièce maîtrise du développement, au sens où sans elle le projet n'existe pas vraiment, il n'a ni ligne de conduite, ni explication. Ceci peut alors être fâcheux, aussi bien au niveau des concepteurs, des programmeurs, que des utilisateurs.

7.11.1. Durant les phases de conception

Les phases de conception incluent donc l'expression des besoins, l'analyse et de document d'architecture et de conception. Plus ces documents en amont seront précis et moins de temps sera perdu dans les étapes en aval (réalisation, test, etc.).

Ces documents ont été produits à l'aide de Poseidon UML en version 6 qui permet de créer tous les diagrammes d'UML 2. Il va jusqu'à générer du code Java ou à destination d'autres langages. Cette fonctionnalité n'a pas été utilisée, j'ai préféré gérer le code directement dans NetBeans, Poseidon n'ayant pas connaissance des annotations JEE6, de JPA, de JSF, etc.

L'édition de documents s'est faite via Word 2008 pour Mac.

7.11.2. Durant la phase de réalisation

Durant la réalisation j'ai ajouté des commentaires aux endroits importants, le code évident se lisant comme du français.

De plus j'ai inséré des tags Javadoc (cf. « Javadoc tool » sur oracle.com) dans chacune des classes, aussi bien au niveau des variables, des méthodes, que des déclarations de classes elles-mêmes. Composant privé ou publique, tout a été documenté sous forme de Javadoc.

Javadoc est un système de documentation très avancé, basé sur des annotations, il permet, via un outil en ligne de commande nommé lui aussi Javadoc, de générer automatiquement une documentation sous forme de pages HTML. Ainsi, une documentation technique est disponible pour toute personne qui effectuera la maintenance.

7.11.3. A destination des utilisateurs

Trois documents ont été écrits à destination des utilisateurs (les administrateurs) :

- Installation et administration du moteur (déploiement, configuration de JavaMail pour l'envoi des notifications, configuration de la base SQL, utilisateur des différents onglets de l'IHM, etc.)
- installation et administration de l'agent (déploiement, utilisateur des différents onglets de l'IHM, signification des différents champs dans les formulaires, etc.)
- installation des modules d'authentification SSO (installation et configuration via script d'installation, principe de fonctionnement, configuration d'une application cliente, etc.)

7.12. Validation du projet

IdMe n'ayant aucun client final, l'évaluation et la validation était donc restreinte à mon chef de projet. Il procéda donc à la recette et à la vérification du comportement du produit par rapport aux spécifications initiales.

7.13. Pistes d'amélioration et d'évolution

Les pistes d'améliorations sont nombreuses. Il s'agît soit de corriger des fonctionnalités existantes, soit d'en augmenter le nombre.

On pourrait par exemple effectuer les opérations suivantes :

- Ecrire un script d'installation pour le déploiement du moteur
- Ecrire un script d'installation pour le module SSO GlassFish.
- Ajouter le support d'identifiant différents sur les systèmes distants
- Mise en place d'un système de workflows
- Respecter certains standards
- Augmenter le nombre de systèmes distants supportés
- Etc.

8. Conclusion

Le temps de réalisation du projet IdMe a finalement atteint 10 mois, dont 2 mois et demi de spécifications, et 3 semaines de refactorisation. Il est donc possible d'avoir un produit preuve de concept incluant des fonctionnalités importantes de la gestion d'identités, mais n'ayant pas pour autant la qualité commerciale d'un produit comme on peut en trouver sur le marché.

Améliorer le produit, le mettre aux normes, rajouter un système de règles pour s'affranchir de l'obligation des identifiants identiques d'un système à l'autre, ajouter un moteur de workflow, sont des étapes logiques et minimales pour obtenir un produit concurrentiel. On atteint facilement 4 à 5 ans de développement en ayant une dizaine de ressources. Outre le choix de technologies récentes, on risque d'avoir un produit, certes complet vis-à-vis des spécifications initiales, mais probablement peu en phase avec la situation du marché au moment de la livraison. En effet, l'écosystème de la gestion d'identité n'est pas encore très stable et ne permet pas facilement de tabler sur une durée aussi longue, à moins d'apporter de réelles innovations et de savoir les vendre.

Plutôt que de développer sa propre solution, il sera donc plus efficace et surtout moins risqué de racheter une société déjà bien implantée dans le marché, ayant une vision futuriste et affûtée de ce dernier à moyen voire long terme, proposant de préférence des fonctionnalités qui différencient le produit des concurrents.

Par ailleurs, ce projet m'a permis de faire évoluer mon profil de concepteur et développeur. En effet, depuis un certain temps je voulais dépasser le stade des JSP et des servlets, et m'orienter vers des architectures SOA, respectant mieux le modèle MVC. C'est désormais chose faite avec JEE6, JPA, Hibernate, JSF2, et les mécanismes intrinsèques à ces technologies dont je n'avais pas connaissance au début de ce projet. Loin d'avoir un niveau d'expert dans chacune des technologies précitées, j'ai pu acquérir un niveau suffisant pour approfondir mes connaissances, et les utiliser en entreprise.

9. Bibliographie

9.1. Ouvrages

[1] Elisa Bertino, Kenji Takahashi, 2010. In *Identity Management: Concepts, Technologies, and Systems.* Artech House Publishers, 196p.

[2] Olivier Iteanu, 2008. In *L'identité numérique en questions*. Eyrolles, 166p.

[3] Dobromir Todorov, 2007. In *Mechanics of User Identification and Authentication : Fundamentals of identity management*. Auerbach Publications, 728p.

[4] Nicolas Debaes, Pierre Pezziardi, Bruno Vincent, 2007. In *Gestion des identités : une politique pour le système d'information*. Octo Technology, 214p.

[5] C. T. Arrington, 2001. In *Enterprise Java with UML*. John Wiley & Sons, 451p.

[6] Peter Rittgen, 2007. In *Enterprise Modeling and Computing with UML*. IGI Global, 314p.

[7] Pierre-Alain Muller, Nathalie Gaertner, 2005. In *Modélisation objet avec UML*. Eyrolles, 514p.

[8] Antonio Goncalves, 2007. In *Les cahiers du programmeur : Java EE 5*. Eyrolles, 329p.

[9] Joshua Bloch, 2008. In *Effective Java (2nde édition)*. Prentice Hall, 384p.

[10] Anthony Patricio, 200. In *Java Persistence et Hibernate*. Eyrolles, 387p.

[11] Elisabeth Freeman, Eric Freeman, Bert Bates, Kathy Sierra, 2004. In *HeadFirst Design Patterns*. O'Reilly Media, 688p.

[12] Thomas Erl, 2005. In *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 792p.

[13] Brett D. McLaughlin, Gary Pollice, Dave West, 2006. In *Head First Object-Oriented Analysis and Design*. O'Reilly, 640p.

[14] Debu Panda, Reza Rahman, Derek Lane, 2007. *In EJB 3 in action*. Manning Publications, 712p.

[15] Martin Fowler, 2002. In *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 560p.

[16] Bert Bates, Kathy Sierra, Bryan Basham, 2008. In *Head First Servlets and JSP (2nde édition)*. O'Reilly Media, 912p.

[17] Eben Hewitt, 2009. In *SOA Cookbook*. O'Reilly Media, 740p.

[18] Ed Burns, Chris Schalk, Neil Griffin, 2009. In *Java Server Faces 2.0: The Complete Reference*. McGraw-Hill Osborne Media, 752p.

[19] Simon Collison, 2006. In *Beginning CSS Web Developpement: From Novice to Professional*. Apress, 448p.

[20] Heather Kreger, Ward Harold, Leigh Williamson, 2003. In *Java and JMX : Building Manageable Systems*. Addison-Wesley Professional, 592p.

[21] Petar Tahchiev, Felipe Leme, Vincent Massol, Gary Gregory, 2010. In *JUnit in Action : Second Edition*. Manning, 504p.

[22] Jean-Philippe Retaillé, Olivier Salvatori, Thierry Templier, Michel Hue, 2005. In *Refactoring des applications Java / J2EE*. Eyrolles, 396p.

[23] Michael Kay, 2008. In *XSLT 2.0 and XPath 2.0 Programmer's Reference (Programmer to Programmer).* Wrox, 1368p.

9.2. Liens Internet

Ajax (avril 2011)
http://fr.wikipedia.org/wiki/Ajax_%28informatique%29

Analyse Orientée Objet avec UML (mars 2011)
http://java.cnam.fr/iagl/glg204/cours/Arr_Analyse.pdf

APACHE DERBY SMP SCALABILITY (avril 2011)
http://folk.ntnu.no/andersmo/derby_project_report.pdf

Apache Tomcat (avril 2011)
http://tomcat.apache.org/

Apache Tomcat 6 changelog (avril 2011)
http://tomcat.apache.org/tomcat-6.0-doc/changelog.html

Conception avec UML (mars 2011)
http://java.cnam.fr/iagl/glg204/cours/Arr_Conc.pdf

Creating a Tomcat Valve to perform authentication (février 2011)
http://spnego.sourceforge.net/tomcat_valve.html

Dossier technique « Gestion des identités ». CLUSIF, 2007. (mars 2011)
http://www.clusif.asso.fr/fr/production/ouvrages/pdf/CLUSIF-Gestion-des-identites.pdf

EMMA : a free Java code coverage tool (mai 2011)
http://emma.sourceforge.net/

GlassFish (avril 2011)
http://glassfish.java.net/

HyperSQL Database (avril 2011)
http://www.hsqldb.org/

Introduction : Modélisation avec UML (mars 2011)
http://java.cnam.fr/iagl/glg204/cours/UMLvsProcessusDev.pdf

JAAS Reference Guide (avril 2011)
http://download.oracle.com/javase/1.4.2/docs/guide/security/jaas/JAASRefGuide.html

Javadoc tool Home Page
http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

JavaMail API (avril 2011)
http://www.oracle.com/technetwork/java/javamail/index.html

Java Persistence API: Simplifying Persistence (avril 2011)
http://www.javapassion.com/j2ee/javaee5persistence.pdf

JSR 196: JavaTM Authentication Service Provider Interface for Containers (avril 2011)
http://www.jcp.org/en/jsr/detail?id=196

LogBack backend pour SFL4J – The logback manual (avril 2011)
http://logback.qos.ch/manual/index.html

Minimum Acceptable Code Coverage (mai 2011)
http://www.bullseye.com/minimum.html

Modélisation des besoins avec UML (mars 2011)
http://java.cnam.fr/iagl/glg204/cours/Arr_ModelBesoins.pdf

MVC web et JSF, pages 93 et suivantes (avril 2011)
http://java.cnam.fr/iagl/glg203/cours/Swing_MVC_JSF_jmd.pdf

OpenID Authenticator for Tomcat (février 2011)
http://blog.facilelogin.com/2008/11/openid-authenticator-for-tomcat.html
Page Object Pattern (avril 2011)
http://blog.josephwilk.net/cucumber/page-object-pattern.html

Page Objects – Selenium (avril 2011)
http://code.google.com/p/selenium/wiki/PageObjects

PrimeFaces (avril 2011)
http://www.primefaces.org/

Profiling - Computer programming (avril 2011)
http://en.wikipedia.org/wiki/Profiling_%28computer_programming%29

RichFaces (avril 2011)
http://www.richfaces.org/

Sample Server Authentication Module (avril 2011)
http://download.oracle.com/docs/cd/E19798-01/821-1752/gizeb/index.html

Session Factory Pattern (avril 2011)
http://www.laliluna.de/articles/first-hibernate-example-tutorial.html

SFL4J système de logging (avril 2011)
http://www.slf4j.org/manual.html

Solvabilité 2 (avril 2011)
http://fr.wikipedia.org/wiki/Solvabilit%C3%A9_II

Solvabilité 2 : 10 questions pour comprendre la réforme et ses enjeux (avril 2011)
http://www.ffsa.fr/ffsa/jcms/p1_81578/solvabilite-2-10-questions-pour-comprendre-la-reforme-et-ses-enjeux?cc=fn_7371

Metro Web Services (avril 2011)
http://metro.java.net/discover/

Selenium web application testing system (avril 2011)
http://seleniumhq.org/

The Java EE 6 Tutorial (avril 2011)
http://download.oracle.com/javaee/6/tutorial/doc/

The Java EE 6 Tutorial - Asynchronous Method Invocation (avril 2011)
http://download.oracle.com/javaee/6/tutorial/doc/gkkqg.html

The Java EE 6 Tutorial - Using the Timer Service (avril 2011)
http://download.oracle.com/javaee/6/tutorial/doc/bnboy.html

The Java Tutorials : Internationalization (avril 2011)
http://download.oracle.com/javase/tutorial/i18n/

Understanding Login Authentication (avril 2011)
http://download.oracle.com/javaee/1.4/tutorial/doc/Security5.html

Using the Embedded EJB Container to Test Enterprise Applications (avril 2011)
http://netbeans.org/kb/docs/javaee/javaee-entapp-junit.html

What's New and Exciting in JPA 2.0. Mike Keith, 2009. Oracle. (avril 2011)
http://jazoon.com/portals/0/Content/ArchivWebsite/jazoon.com/jazoon09/download/presentations/8461.pdf

XSLT Version 2 (avril 2011)
http://www.dpawson.co.uk/xsl/rev2/rev2.html

10. Liste des figures

11. Liste des tableaux

IdMe - Un socle pour la gestion d'identités en entreprise.
Mémoire d'Ingénieur C.N.A.M., Paris 2011.

RESUMÉ

La gestion des identités est omniprésente de nos jours dans les entreprises. La question des habilitations est un enjeu majeur permettant notamment de surveiller les éventuelles inconsistances dans la sécurité des systèmes d'information, d'en garantir la traçabilité, et d'accélérer les processus d'habilitation et de synchronisation des données.

Les différentes étapes du développement informatique permettent de suivre pas à pas la construction d'un projet de gestion d'identités utilisant des technologies récentes ou très répandues telles que Java EE 6, JPA2, JSF2, Hibernate, MySQL, HyperSQL, etc.

Le suivi se déroule dans l'ordre chronologique d'un projet informatique : expression des besoins, analyse, conception, réalisation, test, optimisation, et documentation.

Le projet IdMe permet au final de comparer la persistance du développement d'un projet d'habilitation construit de toute pièce en un temps restreint, à un outil déjà disponible sur le marché.

Mots clés : identité, habilitation, synchronisation, java, cycle, projet.

SUMMARY

Nowadays identity management is at the heart of every company department. The problem of allowing rights is a main concern especially when tracing inconsistencies inside information system security. This is also useful when trying to guarantee the traceability of the system through audit logs, and speed up identity workflows or user data synchronization.

The various stages of project development allow us to follow step-by-step the progress of an identity management application building process, using up-to-date or wide-spread technologies such as Java EE 6, JPA2, JSF2, Hibernate, MySQL, HyperSQL, and so on.

The building report follows a typical and chronological order for software projects: concepts of operations, analysis, design, coding, test, profiling, and documenting.

Finally, the IdMe project allows us to compare the validity of an identity management application built from scratch within a short time, to an already available and commercial tool.

Keywords: identity, management, synchronization, java, cycle, project.