

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Saad Dahlab Blida

FACULTE DE SCIENCES EXACTES
DEPARTEMENT D'INFORMATIQUE

Mémoire de projet de fin d'études
Pour l'obtention du diplôme
D'ingénieur d'état en génie informatique

Thème :

**SYSTEME ANNUAIRE DISTRIBUE POUR
LA GESTION DES RESSOURCES
D'UN RESEAU DOCUMENTAIRE
BASE SUR LE MODELE LDAP**

Présenté par :

Mr Kedjour Hichem
Mlle Bounoua Fatma-Zohra



Soutenu le : 04 Décembre devant le jury :

Mlle Boustia	: présidente
Mme Nadder	: examinatrice
Mr Boukhlef	: examinateur
Mr Bennouar	: examinateur
Mme A. El-Maouhab	: promotrice

Promotion: 2002-2003

Remerciements

Nous tenons à remercier toutes les personnes qui ont rendu cette thèse possible pour leur aide et leurs contributions.

Nous remercions également Mme A. El-Maouhab, Mme F.Hessas, et Mr A.Bouali pour avoir suivi ce travail.

A Mr M.Mahieddine Professeur à l'Université de Blida

A toute l'équipe de Club Internet Fouka

A mon frère Amine

A Brahim Benseti

A mes chers parents

Je dédie aimablement ce travail.

Hichem

A ma sœur Nadia

A ma nièce Samia

A tous mes amis

A mes chers parents

Je dédie aimablement ce travail.

Fatma-Zohra

Résumé :

Le rôle des annuaires reste centré sur la gestion des utilisateurs. Leur champ d'application est pourtant plus vaste. Ils peuvent constituer un système référentiel pour toutes les ressources d'un système d'information (réseaux, machines, application, ...).

Parmi les normes de systèmes annuaires, la norme LDAP est une norme susceptible de créer, à l'échelle d'une institution ou d'un réseau d'institutions, un annuaire distribué perçu comme une seule entité.

L'objectif est d'étudier les spécifications de la norme LDAP ainsi que les environnements de développements pour les systèmes distribués (CORBA, J2EE).

Cette étude fera ensuite l'objet d'une application dans le domaine du réseau documentaire qui aura pour objet de gérer et de signaler les ressources du réseau documentaire nécessaires à l'ensemble des composants du système global.

Mots-clés : systèmes de catalogages, OPAC, systèmes distribués, CORBA, J2EE, systèmes annuaires, LDAP

Résumé :

Le rôle des annuaires reste centré sur la gestion des utilisateurs, leur champs d'application est pourtant plus vaste. Ils peuvent constituer un système référentiel pour toutes les ressources d'un système d'information (réseaux, machines, applications, ...).

Parmi les normes de systèmes annuaires, la norme LDAP est une norme susceptible de créer, à l'échelle d'une institution ou d'un réseau d'institutions, un annuaire distribué perçu comme une seule entité.

L'objectif est d'étudier les spécifications de la norme LDAP ainsi que les environnements de développements pour les systèmes distribués (CORBA, J2EE).

Cette étude fera ensuite l'objet d'une application dans le domaine du réseau documentaire qui aura pour objet de gérer et de signaler les ressources du réseau documentaire nécessaires à l'ensemble des composants du système global.

Mots-clés : systèmes de catalogages, OPAC, systèmes distribués, CORBA, J2EE, systèmes annuaires, LDAP

TABLE DES MATIERES

Introduction	7
<u>Chapitre 1: Les Systèmes Annuaire</u>	10
1. Introduction	10
2. Définition d'un système annuaire	11
3. Qu'est-ce qu'un annuaire?	11
4. Types d'annuaire	11
5. Caractéristiques des annuaire	13
6. Les différences entre un annuaire et une base de données.....	18
7. Qu'est-ce qu'un annuaire peut faire pour vous?	18
8. Les avantages d'un annuaire commun.....	18
9. Que doit-on utiliser un annuaire ou une base de données?	19
10. Les annuaire X.500	20
11. Conclusion	20
<u>Chapitre 2: Systèmes de catalogage et Open Public Access Catalog (OPAC)</u>	22
1. Introduction	22
2. Le catalogage	23
3. Normes de catalogage	24
4. Formats d'échanges et MARC.....	25
5. On-line Public Access Catalog (OPAC)	28
6. Conclusion	31
<u>Chapitre 3: Lightweight Directory Access Protocol</u>	31
1. Introduction	31
2. LDAP vs X.500.....	32
3. Architecture et concepts.....	32
4. Le protocole	33
5. Le modèle d'information	36
6. Le modèle de nommage.....	40
7. Le modèle fonctionnel	45
8. Le modèle de sécurité	46
9. Le modèle de réplication.....	46
10. Le format d'échange de données	47
11. Les URLs LDAP	47
12. Conclusion	49
<u>Chapitre 4: Les Systèmes Distribués et CORBA</u>	49
1. Introduction	49
2. Les systèmes distribués	49
3. CORBA	50
4. Langage de définition d'interface.....	51
5. L'architecture CORBA.....	53
6. Le courtier d'objets	53

7.	Références d'objets.....	54
8.	Service de nommage (Naming service)	54
9.	Les invocations	55
10.	GIOP/IIOP (General Inter-ORB Protocol/Internet Inter-ORB Protocol) 56	
11.	Conclusion	57
Chapitre 5: La plate-forme Java 2, Enterprise Edition		59
1.	Introduction	59
2.	L'architecture multi-étages (multi-tiers).....	59
3.	Les composants de l'application.....	61
4.	Les services standards	62
5.	Les containers J2EE	64
6.	Emballage et déploiement	64
7.	Les rôles de la plate-forme.....	66
8.	Entreprise JavaBeans	67
9.	JavaServer Pages	69
10.	La sécurité	70
11.	Conclusion	70
Chapitre 6: Conception et mise en œuvre.....		73
1.	Introduction	73
2.	Présentation du système	73
3.	Les Composants du système	75
4.	Conception du système	75
5.	Mise en œuvre	82
6.	Démarche de développement et évaluation	106
7.	Conclusion	107
Conclusion		109
Bibliographie :.....		112
Annexe A: Introduction à XML		114
1.	Introduction	114
2.	Structure d'un document XML	114
3.	La mise en page d'un document XML avec XSL.....	115
4.	Les analyseurs	115
5.	Conclusion	116

TABLE DES FIGURES

Figure 1-1 : Un service annuaire non répliqué avec des données stockées dans un seul serveur.	16
Figure 1-2 : Un service annuaire répliqué avec des données stockées dans trois serveurs.	16
Figure 2-1 : Exemple de schéma d'une notice de base.	22
Figure 2-2 : Partie de schéma de l'ISBD.	23
Figure 2-3: Une partie du format UNIMARC.	25
Figure 2-4 : Exemple d'une notice MARC.	25
Figure 2-5 : Un simple affichage bibliographique.	26
Figure 2-6 : Exemple d'un écran.	27
Figure 2-7 : Le catalogue en ligne de la bibliothèque anglaise.	28
Figure 3-1 : Le serveur LDAP comme un frontal à un serveur X.500.	31
Figure 3-2 : Serveur LDAP autonome.	32
Figure 3-3 : Le sous classement.	35
Figure 3-4 : Exemple de Directory Information Tree (DIT)	37
Figure 3-5 : Exemple de DIT montrant des Suffixes et des Références	40
Figure 3-6 : Exemple d'une recherche.	42
Figure 4-1 : L'architecture OMA.	50
Figure 4-2 : La coopération client/serveur.	51
Figure 4-3 : L'architecture CORBA.	52
Figure 4-4 : L'envoi de requêtes à un objet distant	54
Figure 4-5 : Invocation statique.	55
Figure 4-6 : Invocation dynamique.	56
Figure 5-1 : L'architecture Client/Serveur	59
Figure 5-2 : L'architecture multi-étages	60
Figure 5-3 : L'architecture de Java 2 Enterprise Edition.	61
Figure 5-4 : Descripteurs de déploiement J2EE	65
Figure 5-5 : Enterprise JavaBeans.	68
Figure 5-6 : Interaction entre un client et un EJB.	69
Figure 6-1 : Notre réseau documentaire.	73
Figure 6-2 : Les composants du système.	75
Figure 6-3 : Exemple de définition d'attribut.	76
Figure 6-4 : Exemple de définition d'une classe d'objet.	76
Figure 6-5 : le DIT.	77
Figure 6-6 : Les objets services.	78
Figure 6-7 : Les composants de l'application.	80
Figure 6-8 : Les différentes connexions.	82
Figure 6-9 : Description des attributs.	86
Figure 6-10 : L'interface Remote de l'EJB Catalogs.	92
Figure 6-11 : Exemple d'une page de recherche	96
Figure 6-12 : Exemple d'un résultat de recherche	97

Figure 6-13 : Le détail d'un document.....	98
Figure 6-14 : Connexion à un catalogue.....	99
Figure 6-15 : Résultat de la recherche.....	100
Figure 6-16 : Spécification du type de document à ajouter.....	101
Figure 6-17 : Insertion d'une nouvelle notice.....	102
Figure 6-18 : Fin de l'insertion.....	103
Figure 6-19 : Administration de l'application.....	104

INTRODUCTION

Introduction

Pour ne pas perdre ses connaissances l'homme inventa l'écriture. Elle lui permit de préserver tout ce qu'il a acquis comme information durant son existence. Seulement, au bout de quelques siècles, l'homme devant le grand nombre de documents qu'il amassa, perdait un temps considérable à la recherche de la moindre information. Il a donc inventé de petites techniques pratiques pour localiser les documents plus rapidement. C'est l'apparition des tables de matière, un livre de classement des documents. Ainsi, est né le premier CATALOGUE.

Aujourd'hui, un catalogue classique définit un ensemble de descriptions bibliographiques où chacune d'elles décrit un document (son titre, son auteur, son contenu,...). Le catalogue fournit ainsi le point d'accès aux documents, pour cela, il doit inclure suffisamment d'informations pour bien décrire un document et son contenu.

Jusque là, les catalogues étaient transcrits sur des fiches en papier au niveau des bibliothèques. Pour les consulter, les utilisateurs devaient s'y déplacer. La recherche, dans les centaines microfiches, était très difficile et prenait beaucoup de temps. C'est pour cela, que des bibliothèques ont exporté leurs catalogues sur des bases de données et ont installé des terminaux permettant de rechercher rapidement les notices. Avec les réseaux informatiques, le catalogue est devenu accessible en ligne et il est appelé OPAC (Open Public Access Catalog). Plusieurs OPACs fournissent l'accès à plusieurs catalogues; ce qui a donné naissance à des unions de catalogues.

D'autre part, l'homme n'avait pas que les catalogues à gérer, mais toutes les ressources dont il avait besoin telles que les ressources humaines, les annuaires téléphoniques, les pages jaunes, les groupes d'utilisateurs dans un réseau, etc. Pour cela, les systèmes de gestion de bases de données relationnelles furent très utilisés.

Les systèmes de gestion des bases de données relationnelles ont été mis en place pour exécuter des opérations complexes (des transactions, des requêtes SQL avancées...). Par contre pour gérer les ressources, les besoins changent. Les systèmes ont été optimisés pour mieux répondre aux exigences de ce genre. Ces nouveaux systèmes de gestion de données appelés Annuaires ont adopté leur propre technique d'organisation des données (plus de tables) et leurs propres méthodes d'accès (ce n'est plus du SQL).

Bien que leur champ d'application soit très vaste, les annuaires ont été généralement utilisés pour la gestion des utilisateurs. C'est pour cela, que nous allons aborder les systèmes annuaires, leur utilité, leur fonctionnement

fonctionnement et surtout quand est-ce il faut les utiliser ? Puisque par la suite nous allons les utiliser pour créer l'application qui gère l'ensemble des catalogues.

Ces différents points ont été étudiés aux chapitres 1 et 2 consacrés respectivement aux annuaires et aux OPACs.

La norme la plus répandue des systèmes annuaires est la norme LDAP (Lightweight Directory Access Protocol). Cette norme est décrite au chapitre 3.

Vu que les bibliothèques se regroupent de plus en plus sous des réseaux spécialisés, leurs catalogues se trouvent ainsi dispersés à travers le réseau et pour arriver à les gérer l'application se doit d'être **distribuée**. Nous avons donc opté pour deux technologies CORBA (Common Object Request Broker) et J2EE (Java 2, Enterprise Edition) que nous détaillons aux chapitres 4 et 5.

Enfin, le chapitre 6 est consacré à notre approche dans la mise en place du système global de gestion des catalogues sur un réseau documentaire. Nous exposons dans ce chapitre notre modèle conceptuel et également le modèle fonctionnel du système.

CHAPITRE I

1. Introduction

De nos jours, les ordinateurs et les applications deviennent de plus en plus indispensables dans notre vie. Pour améliorer leurs fonctionnalités et les rendre faciles à utiliser, les informations concernant les services, les ressources, les personnes, et les autres objets accessibles par ces applications doivent être organisés d'une manière claire, protégée, et partageable.

Les informations qui décrivent les personnes, les applications, le matériel, et les autres objets sont collectées dans une base de données spéciale appelée *annuaire*.

Ce chapitre décrit les systèmes annuaires, leurs caractéristiques, et leurs avantages. Le chapitre est très largement inspiré de [TIM 99] et [HEI 98] auxquels le lecteur pourra se reporter pour plus de précisions.

2. Définition d'un système annuaire

Un système annuaire ou service annuaire est une collection de logiciels, matériels, processus, politiques, et de procédures administratives nécessaires pour rendre les informations de l'annuaire disponibles aux clients de l'annuaire. Un système annuaire doit au moins inclure les composants suivants :

- Une liste des informations qu'il contient.
- Les logiciels serveurs qui maintiennent ces informations
- Les logiciels clients qui agissent de la part des utilisateurs ou autres entités accédant à ces informations.
- Le matériel sur lequel les applications s'exécutent.
- Des logiciels supplémentaires tels que les systèmes d'exploitation et les pilotes de périphériques.
- Un réseau qui permet de connecter les clients aux serveurs et les serveurs entre eux.
- Des politiques d'administration qui contrôlent les droits d'accès et les informations stockées.
- Des procédures de maintenance et de surveillance du système annuaire.

Généralement, on utilise le terme annuaire comme un synonyme de système annuaire. Mais il faut garder en tête qu'un annuaire est un ensemble de composants qui travaillent ensemble pour fournir le service.

3. Qu'est-ce qu'un annuaire?

Les annuaires font partie de notre vie quotidienne. Les annuaires téléphoniques, les pages jaunes, guides de la télé, les catalogues de bibliothèques et autres sont appelés des annuaires de chaque jour (everyday directories) ou parfois annuaires papiers (offline directories). Les annuaires dans le domaine informatique et les réseaux sont semblables, avec quelques différences. On les appelle annuaires électroniques (online directories). Les annuaires électroniques diffèrent des annuaires papiers dans les points suivants :

- les annuaires électroniques sont dynamiques
- les annuaires électroniques sont flexibles
- les annuaires électroniques peuvent être sécurisés

4. Types d'annuaires

Il y a plusieurs types d'annuaires. En bref, nous pouvons en distinguer les catégories suivantes :

- *Annuaire basé sur les systèmes d'exploitation de réseau* (NOS-based Directories). Tels que NDS de Novell, Active Directory de Microsoft, et StreetTalk de Banyan. Ils ont été développés spécialement pour servir au besoin de système d'exploitation de réseau (Network Operating System).
- *Annuaire spécifique aux applications* (Application-specific directories). Ce sont des annuaires liés ou encastres dans des applications. Par exemple Lotus Notes name et address book, Microsoft Exchange directory, et GroupWise directory de Novell.
- *Annuaire spécifique aux objectifs* (Purpose-specific directories). Ce sont des annuaires non extensibles qui ne sont pas spécifiques à une application mais conçus pour un but bien déterminé. Par exemple, le DNS (Domain Name System).
- *Annuaire à usage général basé sur un standard* (General-purpose, standards-based directories). Ces annuaires sont développés pour servir aux besoins d'une large gamme d'applications. Les annuaires X.500, par exemple.

5. Caractéristiques des annuaires

Les annuaires sont caractérisés par leur dynamique, leur flexibilité et leur sécurité.

5.1 Les annuaires sont dynamiques

La mise à jour d'un annuaire électronique est beaucoup plus simple (et nettement moins coûteuse) à réaliser que celle d'un annuaire papier. Ainsi, un annuaire en ligne (disponible sur le réseau) sera à jour beaucoup plus rapidement, d'autant plus que les personnes recensées dans l'annuaire peuvent elles-mêmes modifier les informations les concernant (si elles sont habilitées à le faire).

5.2 Flexibilité des annuaires

Les annuaires électroniques sont beaucoup plus flexibles dans leurs contenus et leurs organisations par rapport aux annuaires statiques.

5.2.1 Contenu flexible

Les annuaires papiers sont statiques en terme de leur contenu. C'est-à-dire qu'ils contiennent un ensemble très restreint et rarement étendu d'informations. Par exemple, vous n'avez aucune chance si vous cherchez des informations au delà du numéro de téléphone, l'adresse, et le nom dans votre annuaire téléphonique. Par contre, les annuaires électroniques peuvent être facilement étendus par des nouveaux types d'informations. Le coût d'une telle addition est très élevé dans un annuaire imprimé mais relativement bas dans le cas d'un annuaire électronique.

5.2.2 Organisation flexible

Les annuaires électroniques permettent d'organiser l'information selon des critères multiples contrairement aux annuaires papiers, imprimés une fois pour toute pour permettre de rechercher selon un critère figé (en général, l'ordre alphabétique selon le nom). De plus, les annuaires électroniques peuvent fournir des types de recherches plus avancés qui seraient difficiles ou impossibles de fournir dans la forme imprimée. Par exemple, si vous n'êtes pas certain de l'orthographe d'un nom, un annuaire électronique peut vous laisser chercher les noms semblants (sound like) à celui que vous fournissez. Ils peuvent aussi fournir des recherches basées sur les erreurs d'orthographe communes (common misspellings), sous chaînes des noms, et autres variations. Cette puissance de recherche est la clef qui donne aux utilisateurs le comportement de type « faites ce que je signifie » (« do what I mean »).

5.3 Sécurité des annuaire

Les informations d'un annuaire papier sont accessibles à n'importe quelle personne qui a l'accès à l'annuaire, ou ne deviennent pas accessibles du tout si elles ne figurent pas dans l'annuaire. Dans les annuaires électroniques, les clients accèdent à l'annuaire à travers un processus appelé authentification. L'annuaire utilise l'identité fournie par les clients en conjonction avec les listes

List) et autres informations pour décider quels sont les clients qui ont le droit d'accéder à telles informations.

Une ACL est une liste d'autorisations qui peut être attachée aux objets de l'annuaire. Une ACL liste les droits d'accès de chaque utilisateur ou groupe d'utilisateurs. Le Tableau 1 présente un exemple d'ACL pour l'entrée Employé dans l'annuaire de la compagnie avec personnel est le groupe de service personnel.

Tableau 1 : Exemple d'une ACL pour l'entrée Employé dans l'annuaire

Utilisateur ou Groupe	Droits d'accès
propriétaire	lecture, modifier
administrateurs	tous
personnel	lire tous les champs
les autres	lecture limitée

6. Les différences entre un annuaire et une base de données

Un annuaire peut être vu comme une base de données spécialisée. Il est intéressant de comparer les bases de données et les annuaires, car les différences sont liées à l'environnement et non pas avec les éléments fondamentaux. Les différences entre une base de données et un annuaire résident dans les points suivants :

- *Le rapport Lecture/Ecriture.* Ce rapport est très élevé dans les annuaires que dans les bases de données.
- *L'extensibilité.* Les annuaires sont extensibles plus facilement que les bases de données.
- *L'échelle de la distribution.* Les annuaires sont habituellement distribués plus largement que les bases de données.
- *L'échelle de la réplication.* Les annuaires sont souvent répliqués sur une très grande échelle que les bases de données.
- *La performance.* Les annuaires ont des caractéristiques de performance très différentes de celles des bases de données.
- *Les standards.* Le support des standards est important dans les annuaires, et moins dans les bases de données.

6.1 Le rapport Lecture/Ecriture

L'une des caractéristiques déterminantes d'un annuaire est qu'il est typiquement lu ou cherché plus souvent qu'il est écrit ou mis à jour. Ce n'est pas le cas avec les bases de données.

Les informations dans l'annuaire sont généralement lues plusieurs fois qu'elles sont écrites. Par exemple, des centaines de gens peuvent chercher le numéro de téléphone d'une personne, des milliers de clients peuvent lire les caractéristiques d'une imprimante particulière. Mais le numéro de téléphone et les caractéristiques d'imprimante changent rarement.

Pourquoi cette caractéristique est importante ? Elle oriente l'implémentation de l'annuaire. L'implémentation de l'annuaire peut être fortement optimisée pour les types d'opérations qui seront souvent exécutées. Si une opération est exécutée 10.000 fois plus qu'une autre, ça sera une bonne idée de dépenser plus de temps à concevoir cette opération afin qu'elle soit exécutée rapidement. Contrairement aux bases de données qui doivent être optimisées pour les opérations de lecture et d'écriture.

6.2 Extensibilité des informations

Une autre caractéristique déterminante d'un annuaire est que ses informations sont extensibles. Le terme **schéma d'annuaire** se réfère aux types d'information qui peuvent être stockés, les règles que ces informations doivent suivre, et la manière dont ces informations se comportent.

Les annuaires ne se limitent pas à un ensemble fixé de schémas qui peuvent être stockés ou retirés. Ces informations peuvent être étendues pour répondre aux nouveaux besoins et aux nouvelles applications. Un annuaire contient habituellement un ensemble utile prédéfini de types d'information qui peuvent être stockés, mais plusieurs installations ont des exigences spéciales qui nécessitent l'extension de cet ensemble prédéfini.

Bien que les bases de données soient utilisées pour stocker de nombreux types d'informations organisées en plusieurs manières, il sont limitées dans les types d'informations qui peuvent être stockées. Il est rare de trouver une base de données qui vous permet d'introduire de nouveaux types primitifs avec des sémantiques nouvelles.

6.3 Distribution des données

La distribution des données est un autre côté où les annuaires diffèrent des bases de données. La distribution des données se réfère à l'emplacement des informations dans les serveurs à travers le réseau. Les données peuvent être centralisées dans un seul serveur, ou peuvent être distribuées sur plusieurs serveurs.

La distribution des données est un facteur fondamental dans la conception des annuaires. L'un des objectifs d'annuaires est de permettre la distribution des données à travers les différentes parties du réseau. Cette capacité vise à adresser les environnements où l'autorité et l'administration doivent être distribués.

6.4 La réplication des données

La réplication est le processus de maintenir plusieurs copies des données de l'annuaire en différents emplacements. Il y a plusieurs raisons pour faire ceci :

- *La fiabilité.* En cas de panne d'une copie de l'annuaire, on peut avoir accès aux autres.
- *La disponibilité.* Les clients ont beaucoup de chance de trouver une copie disponible, si une partie du réseau est en panne.
- *La localité.* Les clients obtiennent une bonne performance dans les annuaires les plus proches.
- *La performance.* Plus de requêtes peuvent être servies, si des répliques additionnelles sont ajoutées.

Les bases de données supportent la réplication mais pratiquement avec un nombre restreint de répliques. Ceci est dû à la performance qui pose problème lors de l'implémentation de la réplication, car il faut s'assurer que les données dans toutes les répliques soient cohérentes à tous moments. Par contre la réplication des annuaires est presque toujours conséquente. Ceci signifie que l'incohérence temporaire entre les données des différentes répliques est acceptable. Cette caractéristique a des implications importantes sur le nombre de répliques que l'annuaire peut supporter et leurs distributions physiques à travers le réseau.

La disponibilité de l'annuaire est importante car il est utilisé par différentes applications dans leur fonctionnement fondamental comme l'authentification, le contrôle d'accès, et la gestion de la configuration. L'annuaire doit être disponible pour ces applications pour qu'elles fonctionnent correctement.

La réplication des données de l'annuaire rend celui-ci très proche des clients. Ceci aide à diminuer les problèmes du réseau qui peuvent empêcher les clients d'accéder à l'annuaire. Un exemple d'annuaire non répliqué est présenté dans la Figure 1-1, et un autre avec des données répliquées est présenté dans la Figure 1-2.

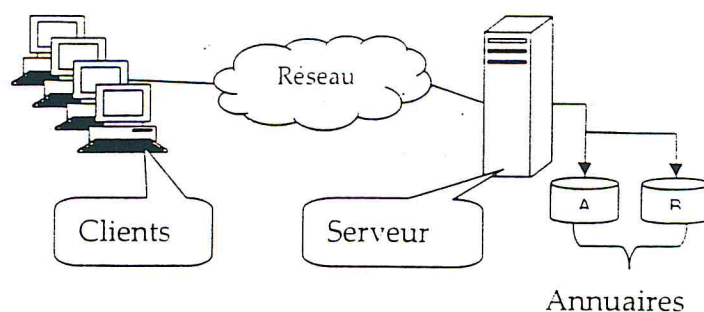


Figure 1-1 : Un service annuaire non répliqué avec des données stockées dans un seul serveur.

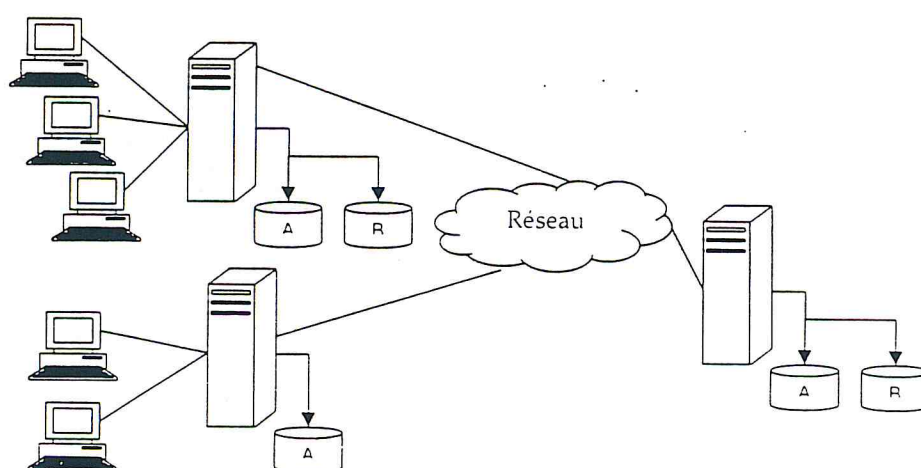


Figure 1-2 : Un service annuaire répliqué avec des données stockées dans trois serveurs.

Les annuaires sont répliqués sur une très grande échelle que les bases de données. Chaque partie (subnet) du réseau peut tenir sa propre réplique de l'annuaire pour minimiser le temps d'attente et augmenter la disponibilité.

6.5 La performance

La haute performance est une autre caractéristique qui différencie les annuaires des bases de données. La performance des bases de données est mesurée en terme de nombre de transactions exécutées par seconde. Celle-ci est aussi une mesure pour la performance d'annuaire, mais les besoins dans les annuaires sont plus rigoureux que ceux de la plupart des systèmes de bases de données.

Le besoin de la performance dans l'annuaire est dû aux variétés d'applications qui utilisent l'annuaire. Vu que les bases de données sont conçues et déployées pour être utilisées par un ensemble connu d'applications, les annuaires sont souvent déployés comme un composant de

l'infrastructure qui sera utilisé par un nombre inconnu, mais croissant d'applications développées à travers la compagnie, et même à travers Internet. Le développement de ces applications fait que l'accès à l'annuaire est distribué. Ce qui signifie que vous ne pouvez pas contrôler les types de requêtes auxquelles l'annuaire doit répondre. Donc, il est important que l'annuaire soit flexible et performant sans se soucier des types de requêtes qu'il doit satisfaire.

Un autre facteur qui exige la performance de l'annuaire est les types d'applications qui accèdent à l'annuaire. Les applications accèdent aux annuaires pour plusieurs buts. Si par exemple votre annuaire est utilisé par un logiciel de messagerie pour router les messages, une ou plusieurs consultations de l'annuaire sont nécessaires pour chaque partie du message. Ceci est une charge importante pour l'annuaire.

6.6 Les standards et l'interopérabilité

Le dernier facteur qui distingue les annuaires et les bases de données, ce sont les standards. Le monde des bases de données supporte plusieurs standards simplifiant la migration d'un système de base de données à un autre. Pourtant, ces standards ne fournissent pas une interopérabilité réelle. Dans le monde des annuaires, toutes les applications doivent être capables d'utiliser l'annuaire, la mise en oeuvre des standards d'interopérabilité est critique.

C'est là où LDAP (Lightweight Directory Access Protocol, traduisez Protocole léger d'accès aux annuaires et prononcez "èl-dap") intervient. LDAP fournit un modèle standard et un protocole utilisé pour accéder aux annuaires modernes tels que les annuaires X.500. LDAP donne la possibilité aux applications de travailler avec n'importe quel annuaire. Dans les bases de données, une application Oracle ne peut pas utiliser une base de données Informix. Ce type d'interopérabilité, absent dans les bases de données, est important aux annuaires pour deux raisons :

- Il permet de séparer les clients d'annuaire de serveurs d'annuaire.
- Il permet de séparer le processus de développement de la décision au sujet d'un annuaire d'un vendeur particulier.

Avant l'apparition de LDAP, chaque application qui utilise un annuaire vient avec son propre annuaire intégré. Ceci ne pose pas de problème au départ, mais considérez le cas où vous avez installé votre 19^{ème} application, donc votre 19^{ème} annuaire. Chaque utilisateur dans votre organisation qui utilise ces applications a besoin d'une entrée dans chaque annuaire; beaucoup d'exemplaires d'informations à maintenir. Ceci augmente les coûts et gêne les administrateurs du système.



Par l'existence d'un standard, les développeurs peuvent écrire des applications en utilisant des outils de leurs choix. Ces applications peuvent tourner avec n'importe quel annuaire qui supporte LDAP, ce qui va rendre l'annuaire comme une partie de l'infrastructure du réseau, et permettre d'augmenter le nombre d'applications qui profite de l'annuaire. De plus, il vous libère de dépendre d'un seul vendeur.

7. Qu'est-ce qu'un annuaire peut faire pour vous?

L'utilisation de l'annuaire ne se limite pas à la recherche de personnes ou de ressources. En effet, un annuaire peut servir à :

- *Trouver des objets.* C'est le but des annuaires habituels. La capacité de recherche est augmentée par les annuaires électroniques. Vous pouvez chercher par n'importe quel attribut et couvrir un champ très vaste. L'annuaire BigFoot (www.bigfoot.com) par exemple, contient tous les annuaires téléphoniques de l'Amérique du Nord.
- *Gestions des objets.* On peut utiliser un annuaire pour sauvegarder la configuration d'un réseau (ordinateurs, serveurs, leurs adresses IP et adresses MAC, ...).
- *Les applications de sécurité.* Par exemple, il permet de stocker les objets de sécurité utilisés par le PKI (Public Key Infrastructure).

8. Les avantages d'un annuaire commun

Les annuaires spécifiques aux applications stockent généralement les informations spécifiques à une application particulière et ils ne sont pas accessibles aux autres applications. Chaque application crée et gère son propre annuaire spécifique. La même information est stockée dans plusieurs annuaires différents, ce qui rend le travail des administrateurs difficile.

Par contre si un annuaire commun indépendant est disponible, les annuaires spécifiques ne sont plus nécessaires et les développeurs peuvent concentrer leurs efforts à résoudre les problèmes liés à l'application et non à développer un annuaire. Mais ce dernier doit être extensible, flexible, sécurisé, performant, et surtout standardisé (c'est-à-dire qu'il offre une interface standard indépendante de tout vendeur).

9. Doit-on utiliser un annuaire ou une base de données?

Voilà une brève liste de critères que vous devez prendre en considération afin que vous puissiez choisir ce qui est mieux pour votre application, un annuaire ou une autre technologie (base de données, système de fichier, ... etc.) :

- *Taille des informations.* Les annuaires sont bien pour stocker des informations de petite taille.
- *Nature des informations.* Les informations des annuaires ont un modèle basé sur les attributs. Si vous pouvez exprimer vos informations dans ce modèle, un annuaire est le bon choix.
- *Le rapport Lecture/Ecriture.* Les annuaires sont bons pour les informations qui sont lues souvent plus qu'écrites.
- *Capacité de recherche.* Si votre application nécessite une bonne capacité de recherche, un annuaire est un bon choix.
- *Accès basé sur des standards.*

10. Les annuaires X.500

Cette norme a été définie en 1988 par Le CCITT (Consultative Committee on International Telephony and Telegraphy) en collaboration avec l'ISO (International Standards Organization). La définition de cette norme d'annuaires a été élaborée, à l'origine, pour traiter la gestion des adresses électroniques.

10.1 Modèle de données

Un annuaire X500 est une collection d'informations de toutes catégories. Ces informations sont stockées dans la *Directory Information Base (DIB)*. La DIB est composée d'entrées constituées d'un ou plusieurs attributs qui peuvent prendre une ou plusieurs valeurs.

10.2 Structure

Les entrées de la DIB sont organisées hiérarchiquement, sous forme d'un arbre, le *Directory Information Tree (DIT)*. Ce DIT contient donc deux types d'entrées :

- Les nœuds qui sont des entrées à part entière, et sont la base d'un sous-arbre du DIT.
- Les feuilles qui terminent les branches des sous-arbres.

Chaque entrée de l'arbre est nommée par un nom, tenant compte du chemin parcouru dans l'arbre pour l'atteindre, depuis la racine. Il existe deux appellations pour ces entrées :

- Le nom distinctif relatif (RDN, Relative Distinguished Name).
- Le nom distinctif (dn, Distinguished Name), concaténation des différents RDNs du chemin utilisé depuis la racine, pour joindre l'entrée concernée.

10.3 Directory Access Protocol et Lightweight Directory Access Protocol

Le protocole d'accès et de requêtes à ces annuaires a été nommé DAP (Directory Access Protocole) et utilisait la pile du protocole OSI (Open Systems Interconnect).

L'inconvénient de cette norme est qu'elle est trop complexe et, par conséquent, trop lourde à mettre en œuvre.

Pour cette raison, une équipe de l'université du Michigan (Wengyik Yeong, Steve Kille, Colin Robbins, Tim Howes, Marc Wahl) a développé un protocole standard basé sur le modèle TCP/IP (Transmission Control Protocol/Internet Protocol) qui a pour nom *Lightweight Directory Access Protocol* (LDAP) en 1993 permettant d'accéder et de mettre à jour un annuaire X.500.

11. Conclusion

Les annuaires occupent une partie importante dans l'infrastructure d'une organisation. Ils permettent de centraliser les informations des différents objets et ressources de l'organisation. L'accès à ces dernières est standardisé, ce qui simplifie leurs mises en œuvre.

CHAPITRE II

1. Introduction

Les bibliothèques sont considérées comme un lien entre la source de connaissances et les utilisateurs. Leur fond documentaire est de plus en plus élargi par des nouveaux documents. Les utilisateurs ne peuvent plus se rappeler de l'emplacement des ouvrages sur les rayons. Le catalogue qui était une simple fiche est devenu un outil efficace et souple pour le repérage de documents.

Ce chapitre définit les normes et les formats de catalogage ainsi que les catalogues en ligne.

2. Le catalogage

Le catalogage est une tâche technique effectuée habituellement par du personnel qualifié du genre bibliothécaire adjoint spécialisé. Il consiste à décrire un ouvrage (titre, auteur, éditeur...) et à définir les points d'accès permettant de le retrouver (recherche par auteur, par sujet,...). Cette description physique et intellectuelle se matérialise par la création d'une *notice bibliographique* [BIB 01]. L'ensemble des notices d'une collection de documents forme un *catalogue*.

Titre : sous-titre ; autre titre / Auteur, second auteur...; auteurs de fonctions différentes. - Mention d'édition. - Lieu de publication : Editeur, date. - n p. : ill. ; n cm + matériel d'accompagnement. - (Collection. Sous-collection, ISSN; n°)
Notes. - ISBN
Différents accès : les vedettes

Figure 2-1 : Exemple de schéma d'une notice de base.

Le catalogue de bibliothèque fournit un accès direct aux publications en pointant directement sur leurs emplacements physiques dans la bibliothèque. Donc, le catalogue doit signaler toutes les ressources de la bibliothèque car un document mal catalogué ou non signalé est un document perdu.

Dans une bibliothèque traditionnelle, le catalogue est mis à la disposition des lecteurs sous forme de fiches de tailles identiques (12,5 x 7,5 cm), ou sous forme de listes (dites listings) éditées par l'ordinateur. Dans une bibliothèque automatisée, le catalogue est accessible en ligne et il est appelé *OPAC* (*On-line Public Access Catalog*).

3. Normes de catalogage

Chaque bibliothèque est libre de choisir les différents éléments (champs) qui décrivent un document. Par exemple, une bibliothèque peut utiliser le titre, le nom de l'auteur, et le nombre de pages pour décrire une monographie, et une autre bibliothèque utilise le titre et l'ISBN (International Standard Book Number). Ceci devient un problème lors de l'échange des notices. Des normes telles que la *Description Bibliographique Internationale Normalisée* (ISBD, International Standard Bibliographic Description) et les *Règles de catalogage anglo-américaines* (AACR, Anglo-American Cataloguing Rules) sont établies pour régulariser la forme et le contenu des descriptions bibliographiques.

Le but de ces normes est de faciliter les échanges des notices bibliographiques entre les agences bibliographiques et les bibliothèques internationales. En spécifiant les différents éléments qui composent la description bibliographique, l'ordre dans lequel les éléments doivent apparaître, et la ponctuation qu'il les sépare (Figure 2-2).

Note : Chaque zone, sauf la première, est précédée d'un point, espace, tiret, espace (. -).

Zone	Ponctuation prescrite précédant (ou encadrant) les éléments	Elément
1. Zone du titre et de la mention de responsabilité	1.1	Titre propre
	[]	1.2 Indication générale du type de document (facultatif)
	=	*1.3 Titre parallèle
	:	*1.4 Complément du titre
	1.5	Mentions de responsabilité
	/	Première mention
	;	*Mention suivante
2. Zone de l'édition		Mention d'édition
	=	*2.1 Mention parallèle d'édition (facultatif)
	2.3	Mentions de responsabilité relatives à l'édition
	/	Première mention
	;	*Mention suivante
	,	*2.4 Autre mention d'édition
	2.5	Mentions de responsabilité relatives à une autre mention d'édition
	/	Première mention
	;	*Mention suivante

Figure 2-2 : Partie de schéma de l'ISBD.

Ces règles permettent de décrire tous les types de document tels que monographie, publications en série, thèses et mémoires, documents cartographiques, ressources électroniques, musique imprimée, documents sonores, microfiches, etc.

3.1 Exemples de notices en ISBD

1. Anglo-American cataloguing rules / prepared by the American Library Association, the British Library, the Canadian Committee on Cataloguing, the Library Association, the Library of Congress ; edited by Michael Gorman and Paul W. Winkler. - 2nd ed. - Chicago : American Library Association, 1978. - 620 p. ; 26 cm.
ISBN 0-8389-3210-X. - ISBN 0-8389-3211-8 (paperback). - ISBN 0-88802-121-6 (Canadian Library Association). - ISBN 0-88802-122-4 (paperback). - ISBN 0-85365-681-9 (Library Association). - ISBN 0-85365-691-6 (paperback)
2. Album de photos [Document électronique] : 101 photos de qualité professionnelle. - Données graphiques (203 fichiers : 471 Mo). - Paris : Micro Application, 1994.

4. Formats d'échanges et MARC

La plupart des bibliothèques à travers le monde utilise le format MARC (*Machine Readable Cataloguing*) pour échanger leurs notices bibliographiques.

MARC (*Machine Readable Cataloguing*) est un format structuré permettant de répartir les données bibliographiques en zones et sous-zones suivant un ensemble de règles qui permettent à la machine de reconnaître les éléments constitutifs d'une notice bibliographique et de savoir comment les traiter (notices présentes dans une forme lisible en machine). Différents formats MARC sont utilisés selon les pays (USMARC, UKMARC, INTERMARC, IBERMARC, etc.). Le format UNIMARC est un format d'échange de données bibliographiques défini dans le cadre de l'IFLA (*International Federation of Library Association*). En 1977, la première version du manuel UNIMARC a été publiée [DIE 98]. La Library of Congress a publié récemment une version révisée sous le nom MARC 21.


```

0-4 Record length
5 Record status
    c = corrected record
    d = deleted record
    n = new record
    o = previously issued higher level record
    p = previously issued as an incomplete, pre-
        publication record
6 Type of record
    a = language materials, printed
    b = language materials, manuscript
    c = music scores, printed
    d = music scores, manuscript

```

Figure 2-3: Une partie du format UNIMARC.

```

Leader *****#a
001 2521854
005 19950215082838.3
008 950215s1994####enk####b####|||eng==
040 ##$aDLC $cDLC $dDLC
080 00$aHA29$b.A5828 1993
082 00$a300/.1/5195 $220
245 00 $aAnalyzing qualitative data /$cedited by Alan Bryman and
    Robert G. Burgess.
260 ##$aLondon ;$aNew York :$bRoutledge,$c1994.
300 ##$a11, 232 p. :$bill. ;$c24 cm.
504 ##$aIncludes bibliographical references and index.
020 ##$a0415060621
020 ##$a041506063X (pbk.)
650 #0$aSocial sciences $xStatistical methods.
650 #0$aSocial sciences $xResearch $xMethodology.
700 10$aBryman, Alan.
700 10$aBurgess, Robert G.
856 4#$3Table of contents $uhttp://www.loc.gov/catdir/toc/95-
    3471.html

```

Figure 2-4 : Exemple d'une notice MARC

5. On-line Public Access Catalog (OPAC)

OPAC (On-line Public Access Catalog) est un catalogue automatisé, il permet aux utilisateurs de consulter le catalogue en ligne (sur ordinateur). OPAC a étendu les capacités du catalogue notamment la capacité de recherche. Il permet de rechercher un élément suivant plusieurs critères. Les notices bibliographiques sont affichées sous forme d'un affichage bibliographique (Figure 2-5).

AUTHOR : Garland, Henry Burnand.
TITLE : The Oxford companion to German literature / by
Henry and Mary Garland. --
EDITION : 2nd ed. / by Mary Garland. --
PUBLISHER : Oxford [England] : Oxford University Press,
1986.
PHYSICAL DESCRIPTION : viii, 1020 p.
OTHER AUTHORS : Garland, Mary.
SUBJECTS : German literature -- Dictionaries. * German
literature -- Bio-bibliography.
CATALOGUING SOURCE NUMBER : 32755373

Figure 2-5 : Un simple affichage bibliographique.

Bien qu'il n'existe pas un format standard pour l'affichage bibliographique, Crawford, Stovel, et Bales (1986) ont décrit les caractéristiques suivantes pour un bon affichage bibliographique :

- 1- L'affichage doit fournir une quantité adéquate d'informations ;
- 2- Les utilisateurs doivent comprendre les informations affichées ;
- 3- L'affichage doit être lisible et attrayant ;
- 4- Les utilisateurs doivent être capables de localiser les informations nécessaires dans l'écran rapidement ; et
- 5- Toutes les informations doivent être affichées dans un seul écran.

OPAC permet généralement aux utilisateurs de choisir entre un bref et un long affichage. Crawford, Stovel, et Bales (1986) suggèrent qu'un affichage bref doit fournir aux utilisateurs les informations nécessaires pour identifier l'élément. Un affichage long ou complet doit donner aux utilisateurs autant d'informations qu'ils trouvent utiles [ANN 96].

5.1 OPAC et le Web

L'interface OPAC traditionnelle est basée sur des écrans. Un écran d'accueil, un pour choisir le catalogue, un pour saisir les données de recherche, et autres pour l'affichage de résultats. L'affichage était en mode texte et la navigation est basée sur des commandes et des raccourcis de clavier. Ces commandes et raccourcis doivent figurer sur chaque écran car l'utilisateur ne peut pas se souvenir d'eux (Figure 2-6).

Search Request: A=GO	ALL LISTINGS
Search Results: 192 Entries Found	Author Index

GOBAT SAMUEL 1799-1879	
2 JOURNAL DUN SEJOUR EN ABYSSINIE PENDANT LES ANNEES 1830 1831 ET 1832	
<1834> (NU)	
3 SAMUEL GOBAT EVANGELISCHER BISCHOF IN JERUSALEM SEIN LEBEN UND WIRKEN	
MEIST NACH SEI <1884> (NU)	
4 SAMUEL GOBAT MISSIONNAIRE EN ABYSSINIE ET EVEQUE A JERUSALEM SA VIE ET	
SON OEUVRE <1885> (NU)	
GOBBATO IMERO	
5 FOMA THE TERRIBLE A RUSSIAN FOLKTALE <1970> (XP)	
GOBINEAU ARTHUR COMTE DE 1816-1882	
6 ADELAIDE MADEMOISELLE IRNOIS PRECEDE DE SOUVENIRS DE VOYAGE <1982>	
(XP)	
7 NOUVELLES ASIATIQUES <1982> (XP)	

STArt over Type number to display record	CONTINUED on next page -
HELp GUIdE	<F8> FORward page
OTHer options	<F7> BACK page
NEXT COMMAND:	

Figure 2-6 : Exemple d'un écran.

Avec le Web, cette interface est devenue beaucoup plus ergonomique. L'affichage peut intégrer des composants non textuels (images, extraits sonores ou de bandes vidéo) (Figure 2-7). Les interfaces Web proposent des caractéristiques qui recouvrent une partie de celles que l'on trouve dans les OPACs traditionnels [THI 96] :

- Le choix du mode de recherche se décline généralement sous trois formes : simple, standard (plusieurs cases pour une recherche sur différents termes dans différents champs) et expert (une case pour l'entrée des différents termes avec un langage de commande spécifique) [THI 96].
- L'utilisateur peut sélectionner s'il le souhaite, à l'aide de la souris, le champ de recherche choisi, les opérateurs booléens utilisés, les opérateurs de proximité, le nombre maximum d'enregistrements ainsi que le mode de classement [THI 96].
- Les pages peuvent être déroulées pour les enregistrements particulièrement longs [THI 96].
- Des liens hypertextes peuvent être spécifiés à l'intérieur des enregistrements, permettant d'élargir la question à un champ particulier [THI 96].

British Library Public Catalogue

File Edit View Favorites Tools Help

Address Go

BLPC | **BASIC SEARCH**

Submit Search
Clear Form
Advanced
Help
Home

Author/Editor:
 Organisation:
 Title:
 Subject:
 Publisher Details:
 ISBN/ISSN:

Choose the type of material you require (All Material, Reference Material, or Document Supply Material), then enter keywords or phrases to identify the items you are looking for and select 'Submit Search'. Enter individual keywords separated by a space and phrases within double quotes. BLPC is

Internet

Figure 2-7 : Le catalogue en ligne de la bibliothèque anglaise.

6. Conclusion

« Le catalogue de bibliothèque est une aide à la découverte. Plus spécifiquement, c'est un outil conçu pour aider un ensemble défini de personnes à repérer de l'information dans une collection complète de données. » (Morgan, 1995).

Des normes et des formats de catalogage ont été établis afin de faciliter l'échange des descriptions bibliographiques.

XML (eXtended Mark-up Language) est devenu un standard d'échange de données. Va-t-il remplacer le format MARC ? Ce qui est sûr est que la structure des enregistrements XML est toujours donnée par les règles de catalogage telles que AACR (Anglo-American Cataloguing Rules) et ISBD (International Standard Bibliographic Description).

Le catalogue n'a pas pu échapper lui aussi à l'informatisation. Parti de simples fiches en carton, il est devenu un système assez complexe qu'on appelle les OPACs.

En améliorant ses capacités de recherche, ces OPACs ont permis de mieux répondre aux attentes des utilisateurs. De plus avec l'OPAC, le catalogue est désormais exporté en dehors des murs de la bibliothèque.

CHAPITRE III

1. Introduction

Lightweight Directory Access Protocol (LDAP) est un standard ouvert qui permet d'accéder et de mettre à jour les informations d'un annuaire.

Ce chapitre décrit l'architecture et les concepts de LDAP. La spécification de LDAP est disponible sur [RFC 2251, RFC 2252, RFC 2253, RFC 2254, et RFC 2255].

2. LDAP vs X.500

Initialement, le protocole LDAP a été conçu comme un moyen d'accès à un annuaire X.500. En 1995, LDAP est devenu un annuaire autonome (standalone LDAP) gérant sa propre base de données.

Un serveur LDAP peut être une *passerelle*. Dans ce cas, le client de LDAP communique avec une passerelle qui traduit et expédie des requêtes au serveur de l'annuaire X.500. Cette passerelle est connue comme un serveur LDAP. Le serveur LDAP doit communiquer en utilisant le protocole TCP/IP (Transmission Control Protocol/Internet Protocol) et la pile du protocole OSI (Open Systems Interconnect). De cette façon, les clients peuvent accéder à l'annuaire X.500 sans s'occuper de sa complexité (Figure 3-1).

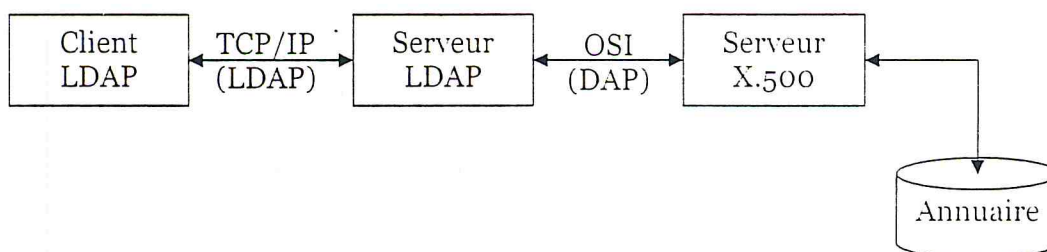


Figure 3-1 : Le serveur LDAP comme un frontal à un serveur X.500

De plus, un serveur LDAP peut s'exécuter d'une manière *autonome* (standalone). Dans ce cas, l'accès est direct à l'annuaire au lieu d'une passerelle aux serveurs X.500. Ceci élimine tout besoin de la pile du protocole OSI. Donc, le serveur LDAP n'utilise que le protocole TCP/IP pour la communication. Ces serveurs LDAP s'appellent souvent les serveurs LDAP autonomes, parce qu'ils ne dépendent pas d'un serveur X.500 d'annuaire (Figure 3-2).

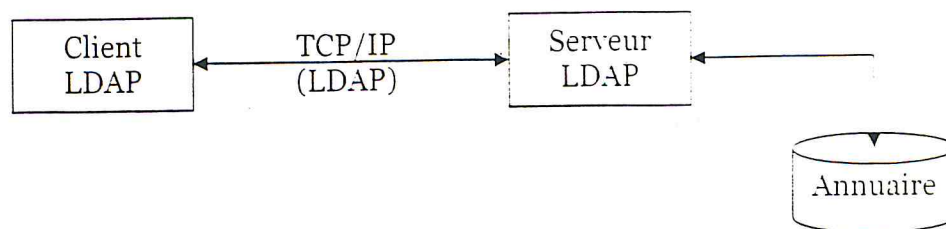


Figure 3-2 : Serveur LDAP autonome

Du point de vue du client, n'importe quel serveur mettant en application le protocole LDAP est un serveur d'annuaire LDAP, si le serveur met réellement en application l'annuaire ou une passerelle à un serveur X.500. L'annuaire consulté peut s'appeler un annuaire LDAP, si celui-ci est mis en application par un serveur autonome LDAP ou par un serveur X.500.

3. Architecture et concepts

LDAP définit :

- *Un protocole.* Il permet d'accéder à l'information stockée dans un annuaire LDAP.
- *Un modèle d'information.* Il décrit le type d'information stockée dans un annuaire LDAP.
- *Un modèle de nommage.* Il décrit comment l'information est organisée et référencée dans un annuaire LDAP.
- *Un modèle fonctionnel.* Il définit les opérations qu'on peut effectuer sur l'information stockée dans un annuaire LDAP.
- *Un modèle de sécurité.* Il définit comment l'information peut être protégée contre les accès non autorisés.
- *Un modèle de réplication.* Il définit comment la base est distribuée entre les serveurs.
- *Un format d'échange de données.*

Les sections suivantes les présentent en détail.

4. Le protocole

LDAP est basé sur le modèle client/serveur. La communication entre un client et un serveur s'établit par échange de messages. Ces messages spécifient les opérations demandées par le client (se connecter ou se

déconnecter, rechercher, comparer, créer, modifier ou effacer des entrées) et les réponses du serveur.

La communication entre serveurs s'établit par échange de leurs contenus (service de réplication) et par la création de liens entre les annuaires (on parle de referral service).

Le format de données transportées dans un message n'est pas l'ASCII (comme pour http, smtp...) mais le Basic Encoding Rules (BER), sous une forme allégée (appelée LBER, Lightweight BER).

Le protocole définit les mécanismes de sécurité qui sont les mécanismes d'authentification et de droits d'accès aux données.

Il définit aussi les opérations de base pour accéder et mettre à jour l'annuaire telles que la connexion, la déconnexion, la recherche, la comparaison, l'insertion, la suppression et la mise à jour.

5. Le modèle d'information

Le modèle d'information définit les types de données stockés dans l'annuaire. L'élément de base de l'annuaire s'appelle une entrée (Entry). Les entrées représentent des objets abstraits ou réels (tels que des personnes, des équipements informatiques, des paramètres de configuration, etc.).

5.1 La structure des entrées

Une entrée est une collection d'attributs qui contiennent des informations caractérisant l'objet que l'entrée définit. Chaque attribut a un type et une ou plusieurs valeurs.

Un type d'attribut est caractérisé par un nom qu'il identifie et un Object Identifier (OID), qui l'identifie également. Il peut être mono ou multi-valué. A chaque type d'attribut est lié une syntaxe.

La syntaxe définit le type de données qui est permis pour les valeurs d'un attribut et comment s'effectue la comparaison de ces valeurs lors d'une recherche (par exemple définir si la recherche sera sensible à la casse, c'est-à-dire si la recherche devra différencier minuscules et majuscules).

Le Tableau 2 montre les principales syntaxes d'attributs définies pour LDAP.

Tableau 2 : Exemple de syntaxes d'attributs de LDAP

syntaxe	Description
bin	Données binaires, comparaison byte/ byte.
ces	Case exact string, Attribut texte. La casse intervient pendant les comparaisons.
cis	Case Ignore string, Attribut texte. La casse n'est pas prise en compte pendant les comparaisons.
tel	Attribut texte représentant un numéro de téléphone (les virgules et les espaces sont ignorés).
int	nombre entier, comparaison numérique.
dn	Distinguished Name (nom distingué).

Certains attributs ont des noms alias qui peuvent être utilisés lorsque le nom complet de l'attribut est utilisé. Par exemple, `cn` peut être utilisé quand on se réfère à l'attribut `commonName`.

Une petite liste d'attributs communs est dans le Tableau 3.

Tableau 3 : Attributs communs

Attribut, alias	Syntaxe	Description	Exemple
<code>commonName</code> , <code>cn</code>	cis	Nom commun d'une entrée	John Smith
<code>surname</code> , <code>sn</code>	cis	nom d'une personne	Smith
<code>organizationalUnitName</code> , <code>ou</code>	cis	Nom d'unité organisationnelle	itso
<code>organization</code> , <code>o</code>	cis	Nom d'organisation	IBM

5.2 La notion de classe d'objet

Par analogie avec la notion d'objet, on parle ainsi de classe d'objet. Une classe d'objet est une description d'une famille d'objets en les caractérisant par une liste d'attributs obligatoires, et/ou des attributs optionnels possibles.

De cette façon, un objet est une « instanciation » de la classe d'objet, c'est-à-dire un ensemble d'attributs avec des valeurs particulières. Par exemple, la classe d'objet `Person` a un attribut `surname`, tandis que l'objet représentant John Smith a un attribut `surname` avec la valeur `Smith`.

Une classe d'objet est définie par un nom qui l'identifie, un OID qui l'identifie également, des attributs obligatoires et des attributs optionnels.

5.3 Le schéma

Le schéma définit les types des objets qui peuvent être stockés dans l'annuaire, les types des attributs de chaque type d'objet et si ces attributs sont obligatoires ou optionnels ainsi que leurs syntaxes.

Chaque entrée de l'annuaire doit obligatoirement appartenir à une classe d'objet du schéma et ne doit contenir que les attributs du type d'objet en question.

Le schéma définit également le sous classement et l'héritage. Des objets peuvent être dérivés d'autres objets. Ceci est appelé *sous classement* (*subclassing*). Par exemple, supposons qu'on a défini une classe d'objet appelée Person qui a l'attribut surname et ainsi de suite. Une classe d'objet organizationalPerson pourrait être définie comme sous-classe de la classe d'objet person. La classe d'objet organizationPerson aurait les mêmes attributs que la classe d'objet person et pourrait ajouter d'autres attributs tels que title et officenumber. Person s'appellerait La classe d'objet supérieure de la classe d'objet organizationPerson. Toutes les classes héritent de la classe Top, qui est la classe de la hiérarchie des classes. La classe Top inclut un seul attribut obligatoire objectClass.

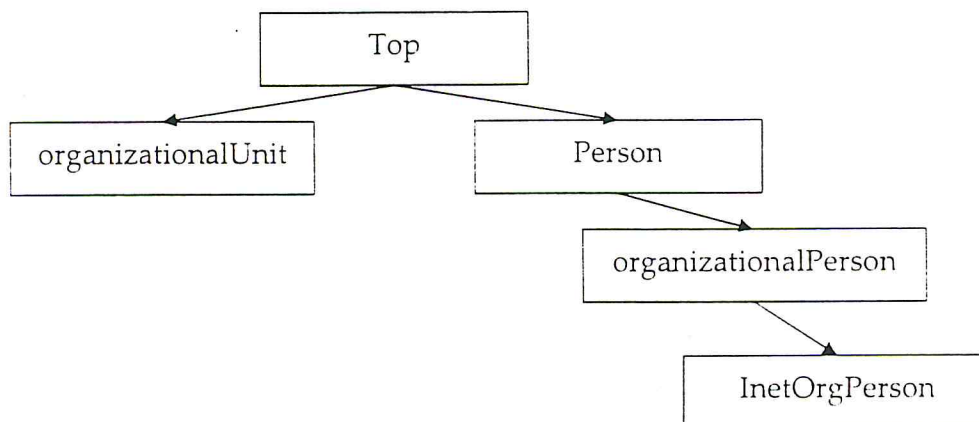


Figure 3-3 : Le sous classement.

Chaque entrée de l'annuaire a un attribut spécial appelé objectClass. La valeur de cet attribut spécifie la classe d'objet de l'entrée. Une des valeurs doit être top ou alias. Alias est utilisé si l'entrée est un alias pour une autre entrée, autrement top est utilisé. L'attribut objectClass détermine quels attributs l'entrée doit et peut avoir. Dans beaucoup de cas, une entrée peut appartenir à une ou plusieurs classes.

Il faut obligatoirement spécifier la parenté d'une classe d'objet à partir de la classe Top et passant par chaque ancêtre de la classe. Par exemple, pour la classe InetOrgPerson :

```
objectClass: Top
objectClass: Person
objectClass: organizationalPerson
objectClass: InetOrgPerson
```

Le tableau suivant présente des classes d'objet et leurs attributs obligatoires.

Tableau 4 : Classes d'objet et leurs attributs obligatoires

classes d'objet	Description	attributs obligatoires
InetOrgPerson	Définit des entrées pour une personne	commonName (cn) surname (sn) objectClass
organizationalUnit	Définit des entrées pour des unités organisationnelles	ou objectClass
organization	Définit des entrées pour des organisations	o objectClass

Quand une entrée est créée, le serveur vérifie si sa syntaxe est conforme à sa classe ou ses classes d'appartenance. C'est le processus de *Schema Checking*.

5.4 Les Identificateurs d'objets (OIDs)

Un *Object Identifier (OID)* est une suite unique de nombres entiers séparés par des points. Les OIDs sont utilisés pour coder l'identifiant des classes d'objets et des attributs. Ils sont formés de manière hiérarchique.

Par exemple :

```
2.5.4      - est la définition des types d'attributs
2.5.6      - est la définition des classes d'objets
1.3.6.1    - Internet OID
```

6. Le modèle de nommage

Le modèle de nommage définit comment sont nommées et organisées les entrées de l'annuaire. Les entrées sont organisées sous forme d'une structure hiérarchique appelée *arbre d'information de l'annuaire (DIT, Directory Information Tree)*. Chaque entrée est nommée par un *nom distinctif (DN, Distinguished Name)*.

6.1 L'arbre d'information de l'annuaire (Le DIT)

La classification des entrées se fait dans une arborescence hiérarchique. Chaque nœud de l'arbre représente une entrée de l'annuaire. L'entrée qui se trouve au sommet de l'arbre est nommée Root. Cette racine est conceptuelle mais elle n'existe pas réellement.

L'organisation des entrées dans le DIT est limitée par leurs définitions de classes d'objet correspondantes. Habituellement, un plan géographique ou organisationnel est suivi. Par exemple, les entrées qui représentent des pays seraient en haut du DIT. Au dessous, les pays seraient des organisations nationales, états, provinces, et ainsi de suite. Au-dessous de ce niveau, les entrées pourraient représenter des personnes dans ces organisations ou des subdivisions supplémentaires de l'organisation. Les feuilles du DIT pourraient représenter n'importe quel objet, tels que des personnes, matériel, processus d'application, et ainsi de suite. La profondeur ou la largeur du DIT n'est pas restreinte et peut être conçue pour satisfaire des exigences d'applications.

Un exemple d'un DIT est montré sur la figure suivante :

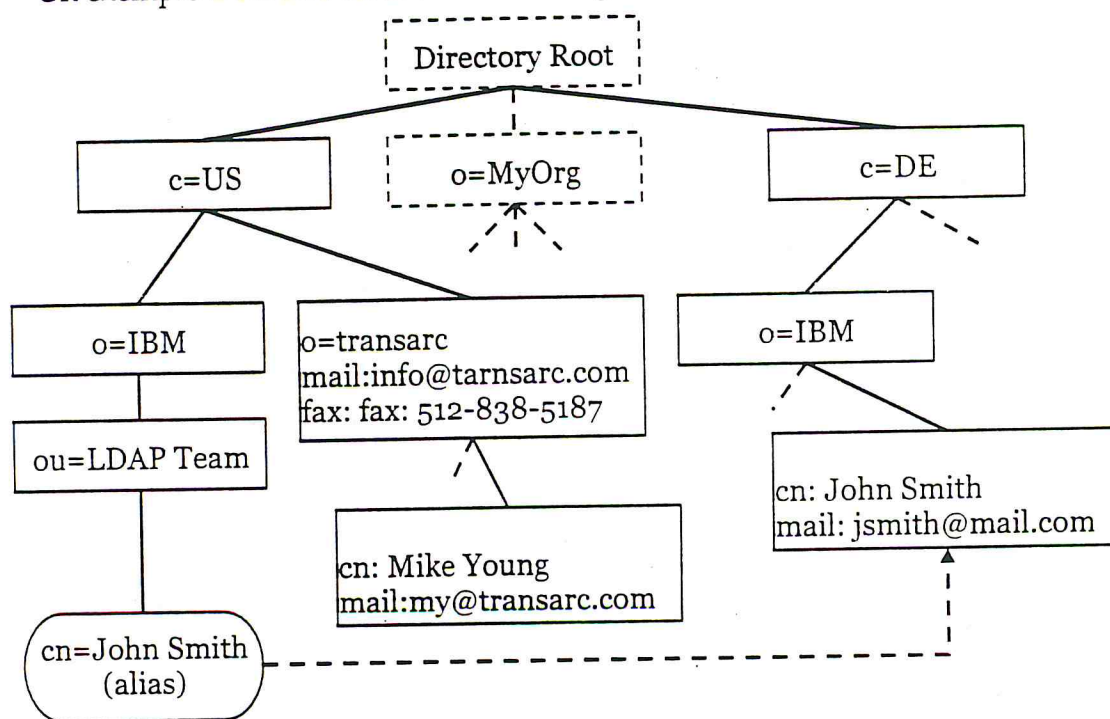


Figure 3-4 : Exemple de Directory Information Tree (DIT)

6.2 Le nom distinctif (Distinguished Name, DN)

Chaque entrée possède un nom distinctif (DN, Distinguished Name) qui l'identifie de façon claire et unique. Le nom distinctif est composé de la séquence des noms des entrées depuis l'entrée concernée et en remontant vers le suffixe. Chaque composant du DN est appelé un *nom distinctif relatif* (RDN,

Relative Distinguished Name) séparés par des virgules. Les entrées sont appelées selon leurs positions dans le DIT. Par exemple, le DN de l'entrée John Smith de la Figure 3-4 est `cn=John Smith, o=IBM, c=DE`.

Un nom distinctif relatif (RDN) est composé d'un attribut (ou des attributs) de l'entrée qu'il nomme utilisant toujours un attribut obligatoire. Dans le cas simple et commun, un RDN est de forme `<attribute name> = <value>`. Par exemple, le DN `cn=John Smith, o=IBM, c=DE` est construit en ajoutant le RDN `cn=John Smith` au DN de l'entrée ancêtre `o=IBM, c=DE`. `cn=John Smith` est un attribut dans l'entrée `cn=John Smith, o=IBM, c=DE`. Le DN d'une entrée est spécifié quand elle est créée.

6.3 Syntaxe du nom distinctif (DN)

Les DN sont utilisés en tant que clefs primaires aux entrées de l'annuaire. Les RDNs peuvent être plus compliqués que dans les exemples montrés ci-dessus. Un RDN peut se composer d'attributs multiples jointifs par "+" comme dans le DN `cn=John Smith+1=Stuttgart, o=IBM, c=DE`.

Si les valeurs d'attributs contiennent des caractères spéciaux, ces caractères doivent être précédés par un caractère antislash. Le DN suivant contient un caractère virgule `o=Transarc\, Inc., c=US`.

Les types d'attributs utilisés dans le RDN peuvent être représentés par un OID qui code l'identifiant de son objet. Par exemple, `cn=John` pourrait également être écrit comme `2.5.4.2=John`. Cependant, les noms d'attributs fréquemment utilisés peuvent être représentés par une chaîne de caractères qu'est évidemment plus facile à comprendre.

Le tableau suivant présente une partie des types d'attributs communs et leurs représentations par chaîne de caractères.

Tableau 5 : Représentations par chaîne de caractères du type d'attribut

Type d'attribut	Chaîne de caractères
CommonName	CN
OrganizationName	O
OrganizationalUnitName	OU
DomainComponent	DC

6.4 Alias, suffixes et referrals

Un serveur LDAP individuel ne pourrait pas stocker le DIT entier. Un serveur pourrait stocker les entrées pour un département particulier et non pas les entrées pour les ancêtres du département. Par exemple, un serveur pourrait stocker les entrées pour le département ITSO¹ de IBM. Le nœud le plus haut dans le DIT stocké par le serveur serait `ou=ITSO, o=IBM, c=US`. Le

¹ International Technical Support Organization

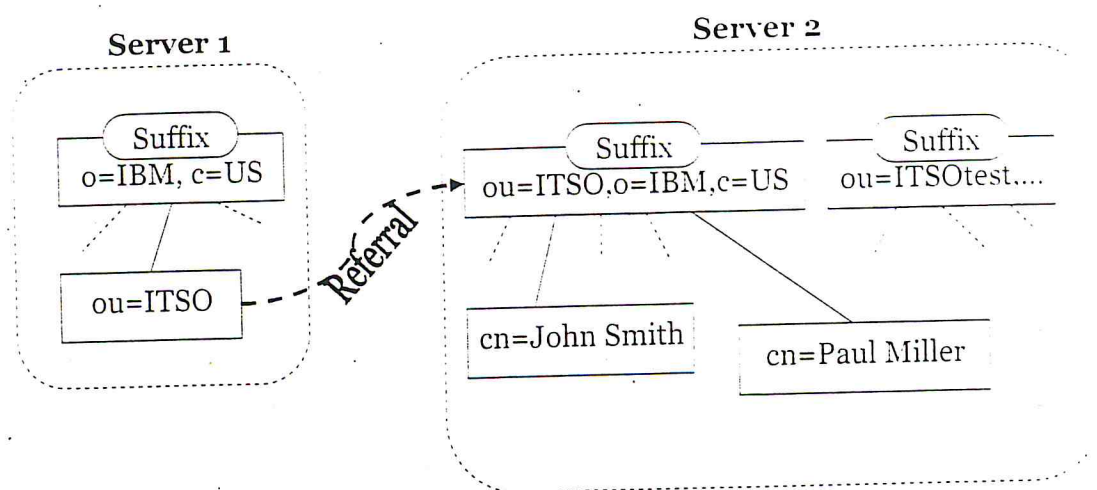


Figure 3-5 : Exemple de DIT montrant des Suffixes et des Références

Quand un client envoie une requête à un serveur LDAP, la réponse peut être une référence. Le client peut alors rentrer en contact avec l'autre serveur LDAP contenu dans la référence retournée par le serveur LDAP. Les références ne sont pas traitées par les serveurs. Ceci peut améliorer la performance du serveur.

7. Le modèle fonctionnel

Le modèle fonctionnel décrit le moyen d'accès aux données stockées dans un annuaire LDAP et les opérations qu'on peut leurs appliquer. Ces opérations peuvent être divisées comme suit :

- Les opérations d'interrogation.
- Les opérations de comparaison.
- Les opérations de mise à jour.
- Les opérations d'authentification.
- Les opérations étendues.

7.1 Interrogation

L'opération d'interrogation permet à un client de demander au serveur de rechercher le contenu d'une entrée. Voici quelques exemples de recherche :

- Trouver l'adresse postale pour `cn=John Smith, o=IBM, c=DE`.
- Trouver toutes les entrées qui sont des enfants de l'entrée `ou=ITSO, o=IBM, c=US`.

- Trouver l'adresse e-mail et le numéro de téléphone de n'importe qui à IBM dont le dernier nom contient les caractères « miller » et qui a également un numéro de fax.

La demande de recherche est composée de 8 paramètres (Tableau 6).

Tableau 6 : Paramètres de recherche.

Paramètre	Signification
Baseobject	un DN qui définit le point de départ de la recherche (l'endroit de l'arbre où doit commencer la recherche), appelé BaseObject (objet de base).
Scope	indique la profondeur de la recherche depuis le baseobject dans le DIT. Il y a trois choix : • BaseObject : la recherche n'est effectuée que sur le baseobject. • SingleLevel : la recherche est effectuée seulement sur les fils immédiats du baseobject; le baseobject lui-même n'est pas examiné. • WholeSubtree : la recherche est effectuée sur le baseobject et tous ses descendants.
Filtre	définit les conditions qui doivent être remplies pour que le résultat coïncide avec la demande. Le filtre de recherche est une combinaison booléenne de AVA (Attribut Assertion Value). Un AVA est une expression dont le résultat est Vrai, Faux ou Indéfini, exprimant un test entre un attribut et une de ses valeurs.
Attributes	les attributs que l'on souhaite connaître. Une liste vide signifie que tous les attributs de chaque entrée trouvée doivent être retournés.
DerefAliases	indique la façon suivant laquelle les alias sont manipulés dans la recherche. Il y a 4 valeurs possibles : • NeverDerefAliases : ne déférencie pas les alias dans la recherche en localisant le baseobject. Ceci permet aux entrées alias elles-mêmes d'être examinées. • DerefInsearching : différencie les alias des descendant du baseobject dans la recherche mais pas en localisant le baseobject de la recherche. • DerefFindingBaseObject : différencie les alias en localisant le baseobject de la recherche mais pas en recherchant des descendants du baseobject. • DerefAlways : différencie les alias dans la recherche en localisant le baseobject de la recherche.

Paramètre	Signification
Sizelimit	limite le nombre de réponses à retourner de la recherche. Une valeur de 0 indique qu'il n'y a pas de limite de taille.
Timelimit	limite le temps (en secondes) alloué à la recherche. Une valeur de 0 indique qu'il n'y a pas de restriction. Les serveurs sont libres d'imposer des limites plus strictes que demandées par le client.
AttrOnly	un flag indiquant si les résultats de la recherche doivent contenir les valeurs des attributs en plus de leurs types (AttrOnly = false), ou seulement les types (AttrOnly = true).

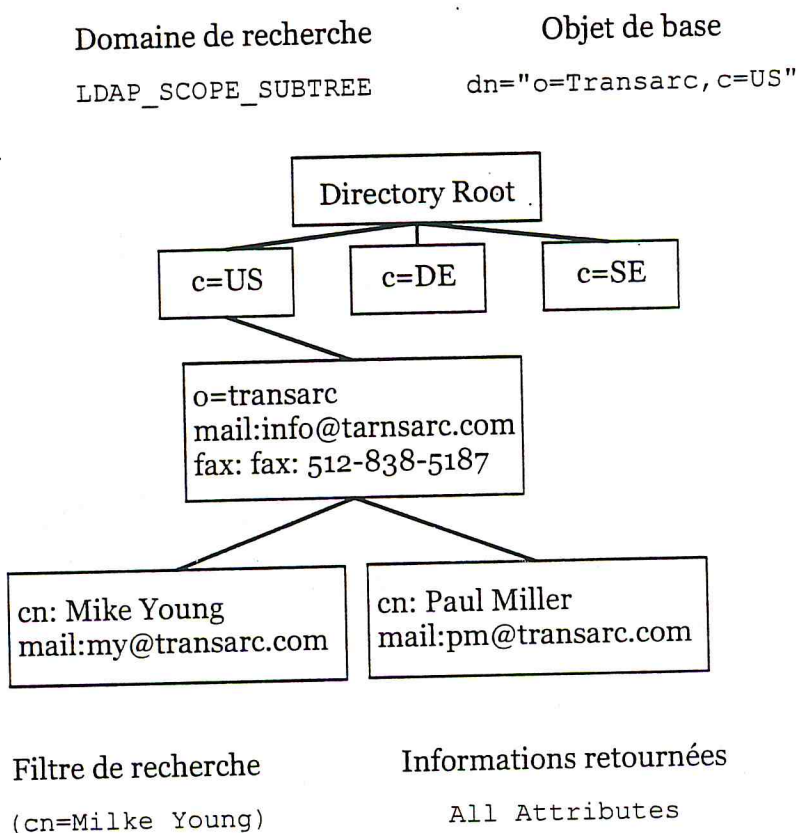


Figure 3-6 : Exemple d'une recherche.

7.2 La syntaxe du filtre de recherche

Le composant de base d'un filtre de recherche est un AVA (*Assertion Value* *Attribut*) de la forme:

attribut opérateur valeur

Par exemple, pour rechercher une personne appelée John Smith, le filtre de recherche serait cn=John Smith. Dans ce cas, le cn est l'attribut; = est

l'opérateur, et John Smith est la valeur. Les résultats retournés sont des entrées avec le nom commun John Smith.

Le Tableau 7 présente les opérateurs pour des filtres de recherche.

Tableau 7 : Opérateurs du Filtre de Recherche

Opérateur	Description	Exemple
=	Retourne les entrées dont l'attribut est égal à la valeur.	cn=John Smith trouve l'entrée avec le nom commun John Smith
>=	Retourne les entrées dont l'attribut est supérieur ou égal à la valeur.	sn>=smith trouve toutes les entrées de smith au z*
<=	Retourne les entrées dont l'attribut est inférieur ou égal à la valeur.	sn<=smith trouve toutes les entrées de a* au smith
=*	Retourne les entrées qui ont une valeur set pour cet attribut.	sn=* trouve toutes les entrées qui ont l'attribut sn
~=	Retourne les entrées dont la valeur d'attribut est égale approximativement à la valeur spécifiée.	sn~= smit pourrait trouver l'entrée "sn=smith"

Le caractère « * » égale n'importe quelle sous chaîne et peut être employé avec l'opérateur =. Par exemple, cn = J*Smi* égalerait John Smith et Jan Smitty. Des filtres de recherche peuvent être combinés avec les opérateurs booléens pour former des filtres de recherche plus complexes. La syntaxe pour des filtres de recherche combinés est:

```
("&" or "|" (filter1) (filter2) (filter3) ...)
("!" (filter))
```

Les opérateurs booléens sont présentés dans le Tableau 8.



Tableau 8 : Opérateurs Booléens

Opérateur Booléen	Description
&	Retourne des entrées assortissant tous les critères du filtre indiqué.
	Retourne des entrées assortissant un ou plusieurs des critères du filtre indiqué.
!	Retourne des entrées pour lesquelles le filtre n'est pas vrai. Cet opérateur peut seulement être appliqué à un filtre simple. <code>!(filter)</code> est valide, mais <code>!(filter1) (filter2)</code> n'est pas valide.

Par exemple, `((sn=Smith) (sn=Miller))` égale des entrées avec le nom de famille Smith ou le nom de famille Miller. Les opérateurs booléens peuvent également être nichés comme dans `((sn=Smith) (&(ou=Austin) (sn=Miller)))`, qui égale n'importe quelle entrée avec le nom de famille Smith ou avec le nom de famille Miller qui a également l'attribut unité organisationnelle Austin.

7.3 Comparaison

L'opération de comparaison permet de comparer une affirmation (`attribute=value`) avec une entrée. Si l'entrée a cette valeur, la comparaison retourne `true`. Autrement, la comparaison retourne `false`. Bien que la comparaison soit équivalente à une recherche, la différence est que si l'entrée n'a pas du tout l'attribut (l'attribut n'est pas présent) ou si l'attribut n'a pas cette valeur, la recherche ne retournera rien alors que la comparaison retournera un code d'erreur si l'attribut n'existe pas et `false` si l'entrée existe, mais elle n'a pas un attribut égale à la valeur indiquée.

7.4 Mise à jour

Les opérations de mise à jour modifient le contenu de l'annuaire. Le Tableau 9 présente les opérations de mise à jour.

Tableau 9 : Opérations de Mise à Jour

Opération	Description
add	Ajoute de nouvelles entrées à l'annuaire.
delete	Supprime des entrées existantes de l'annuaire. Seulement les nœuds feuilles qui peuvent être supprimés.
modify	Change les attributs et les valeurs contenus dans une entrée existante. Permet à de nouveaux attributs d'être ajoutés et aux attributs existants d'être supprimés ou modifiés.
modifyDN	Change le composant moins significatif (la plupart gauche) d'un DN ou déplace un sous-arbre des entrées à un nouvel endroit dans le DIT.

7.5 Authentification

Les opérations d'authentification sont employées pour établir et finir une session entre un client LDAP et un serveur LDAP. Le Tableau 10 récapitule les opérations d'authentification.

Tableau 10 : Opérations d'Authentification

Opération	Description
bind	Initie une session entre un client et un serveur LDAP et permet l'authentification du client au serveur.
unbind	Clôture une session client/serveur.
abandon	Permet d'abandonner l'opération précédemment envoyée au serveur.

7.6 Opérations Etendues LDAP (LDAP extended operations) et Contrôles LDAP (LDAP controls)

Contrôles LDAP et opérations étendues LDAP permettent au protocole LDAP d'être extensible sans avoir à changer le protocole lui-même.

- Contrôles LDAP : paramètres supplémentaires ajoutés à une opération qui en modifie le comportement.
- Opérations étendues : rajoutent de nouvelles opérations au protocole LDAP.

8. Le Modèle de Sécurité

Le modèle de sécurité définit comment les données de l'annuaire peuvent être protégées des accès non autorisés.

8.1 L'authentification

L'authentification permet la vérification de l'identité et éventuellement le mot de passe des utilisateurs lors de leurs connexions. Les mécanismes d'authentification lors de l'ouverture d'une session (bind) sont :

- *Anonymous authentication*. Accès sans authentification permettant d'atteindre les données sans restrictions d'accès.
- *Root DN authentication*. Accès administrateur (tous les droits).
- *Mot de passe en clair*. Un DN plus un password qui transite en clair sur le réseau.
- *Mot de passe + SSL (LDAPS) ou TLS*. La session est chiffrée et le mot de passe ne transite plus en clair.
- *Certificats sur SSL*. Echange de certificats SSL (clefs publiques/privées).

- *Simple Authentication and Security Layer (SASL)*. Mécanisme externe d'authentification.

8.2 Le contrôle d'accès

Une fois le client authentifié, le serveur lui attribue des droits d'accès (lecture, écriture, comparaison, etc.) aux données (entrées, attributs, valeurs) qui lui ont été décrits dans les entrées sous la forme d'un attribut Access Control List (ACL). La description des droits utilise l'identité des utilisateurs de l'annuaire, décrits dans des entrées de l'arbre. Ce mécanisme permet la protection d'une partie d'une entrée ou d'un sous-arbre entier.

9. Le modèle de réplication

Le modèle de réplication définit comment répliquer l'annuaire sur plusieurs serveurs. La réplication de l'annuaire peut pallier à une panne de l'un des serveurs ou une coupure du réseau et garantir la qualité du service (temps de réponse et sûreté de fonctionnement). Elle permet également de répartir le travail entre plusieurs serveurs et d'améliorer les performances en plaçant les serveurs près des clients. Le modèle de réplication n'est pas encore standard mais il est proposé par plusieurs serveurs. On peut répliquer un sous arbre, l'arbre entier ou seulement une partie des entrées et de leurs attributs qu'on doit spécifier. Par exemple, on ne réplique que les objets de type personne. Les serveurs doivent tous utiliser le même schéma de données et les règles d'accès aux données répliquées doivent l'être aussi. La mise en œuvre de la réplication nécessite de la prévoir à la conception du DIT (Directory Information Tree).

10. Le format d'échange de données

LDAP Data Interchange Format (LDIF) est le standard de représentation des entrées, sous forme texte, permettant ainsi une manipulation facile des données. Le format utilisé est l'ASCII. Toute donnée qui n'est pas ASCII, est codée en base 64.

LDIF est typiquement employé pour afficher ou modifier les informations de l'annuaire selon deux modes :

- Importer et exporter les données de l'annuaire entre les serveurs d'annuaire, par exemple quand un serveur LDAP doit être déplacé à une autre place (matériellement).
- Décrire un ensemble de modifications qui doivent être appliquées sur des entrées.

Par exemple, une entrée de type personne est définie comme suit:


```
dn: cn= June Rossi, ou= accounting, o= Ace Industry, c= US
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: June Rossi
sn: Rossi
givenName: June
mail: rossi@aceindustry.com
userPassword: {sha} KDIE3AL9DK
```

11. Les URLs LDAP

Les URLs LDAP permettent aux clients Internet d'avoir un accès direct au protocole LDAP.

syntaxe :

```
ldap[s]://<hostname>:<port>/<base_dn>?<attributes>?<scope>?<filter>
```

<base_dn> : DN de l'entrée qui est le point de départ de la recherche

<attributes> : les attributs que l'on veut consulter

<scope> : la profondeur de recherche dans le DIT à partir du

<base_dn>

- base : s'arrête au niveau courant (par défaut)

- one : descend d'un niveau

- sub : parcourt tous les sous-niveaux

<filter> : filtre de recherche, par défaut (objectClass=*)

Exemples:

```
ldap://ldap.netscape.com/ou=Sales,o=Netscape,c=US
```

```
ldap://ldap.ibm.com/cn=John%20Smith,ou=people,o=ibm
```

12. Conclusion

Les annuaires LDAP peuvent être utilisés comme une infrastructure pour l'entreprise puisque ils fournissent un annuaire standard qui peut être partagé par toutes les applications de l'entreprise.

LDAP est à sa version 3, et des membres de l'IETF (Internet Engineering Task Force) travaillent actuellement afin de l'enrichir.

CHAPITRE IV

1. Introduction

Par nature, les données d'une application et leurs utilisateurs ne se trouvent pas dans le même endroit. Il a fallu donc découper les applications pour suivre les données et les utilisateurs. Ceci va ajouter plusieurs difficultés aux développeurs car ils doivent s'occuper de la communication, de la sécurité, de la concurrence, etc. Pour soulager les développeurs, des environnements ont été développés afin qu'ils les déchargent des tâches répétitives. Ainsi, les développeurs peuvent concentrer leurs efforts à résoudre les problèmes liés à l'application.

Ce chapitre va décrire CORBA qui est l'un de ces nombreux environnements. La spécification de CORBA est disponible sur [www.omg.org/corba].

2. Les systèmes distribués

Un système distribué est organisé comme un ensemble de processus qui s'exécutent sur des sites reliés par un réseau de communication.

Les données sont décentralisées et réparties sur les différents sites. L'utilisateur final sur l'un des sites peut accéder aux données de n'importe quel autre site. Cette distribution est transparente aux clients.

2.1 Les systèmes à objets distribués

Un système à objets distribués est un système composé d'une collection d'objets distribués à travers un réseau. Ces objets peuvent être distribués sur un ensemble de machines qui interagissent par des échanges de messages et des invocations distantes des méthodes des objets. Ce système fournit l'interopérabilité entre systèmes hétérogènes.

Un client peut invoquer des services sur tous les objets du système (comme s'ils étaient locaux à une application) indépendamment de leurs localisations, des systèmes d'exploitation, des protocoles de communication et des langages. Parmi les systèmes à objets distribués les plus populaires, nous trouvons le *Common Object Request Broker Architecture* de l'OMG (**CORBA**), le *Distributed Component Object Model* de Microsoft (**DCOM**) et le *Java/Remote Method Invocation* de JavaSoft (**Java/RMI**).

3. CORBA

CORBA (*Common Object Request Broker Architecture*) est une architecture permettant l'intégration des applications orientées objets distribuées dans un environnement hétérogène en terme de systèmes et de langages de

programmation basée sur le modèle client/serveur. Ainsi, les concepts objets mis en avant sont la réutilisabilité, l'interopérabilité et la portabilité.

L'architecture CORBA a été définie dans les années 1990 par l'OMG (*Object Management Group*); un consortium international créé en 1989 et regroupant plus de 800 membres (vendeurs de logiciels, constructeurs et utilisateurs).

L'OMA (*Object Management Architecture*) est le modèle de référence de CORBA. Il est formé de quatre composants :

- L'ORB (*Object Request Broker*) qui se trouve au cœur de l'architecture, est un bus logiciel de communication. Il assure le transport des requêtes entre les clients et les serveurs.
- Les services objets (*Object Services*) fournissent des services de base tels que le service de persistance (*Persistence Service*), le service de nommage ou de sécurité des communications. Ces services sont implantés sous forme d'objets et interagissent via l'ORB.
- Les utilitaires communs (*Common Facilities*) qui sont des extensions de l'ensemble des services de base et qui sont spécifiques au domaine. Par exemple, un ensemble de services pour supporter des documents distribués ou des applications multimédia. Comme les services objets, ce sont des objets qui interagissent en utilisant l'ORB.
- Les objets d'applications (*Application Objects*) sont les composants des applications distribuées et utilisent les services objets et les fonctionnalités communes.

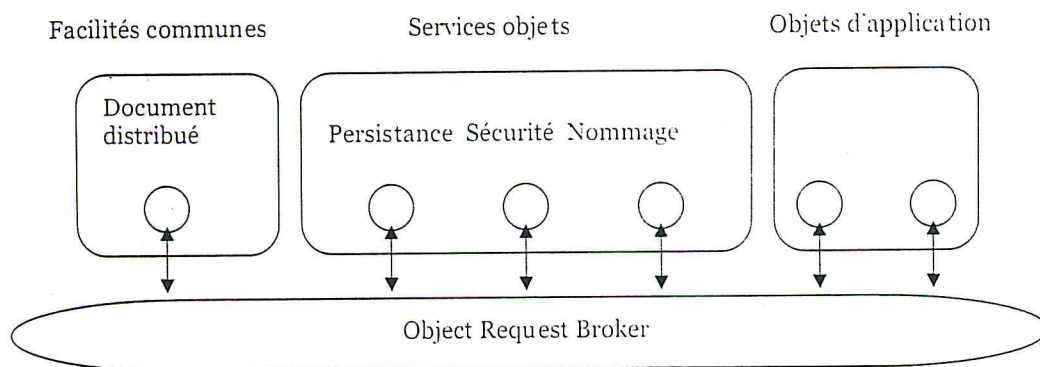


Figure 4-1 : L'architecture OMA

4. Langage de définition d'interface

IDL (*Interface Definition Language*) est un langage de définition des interfaces d'objets (attributs et méthodes). Il est purement déclaratif (il ne

sur l'implantation). La correspondance entre IDL et un langage de programmation est fournie pour les langages C, C++, Java, Ada, Cobol, Smalltalk dans lesquels les méthodes des objets spécifiées en IDL peuvent être écrites.

L'IDL fournit des interfaces indépendantes du système d'exploitation et du langage de programmation pour accéder à tous les services offerts par les serveurs. Cela permet l'interaction entre les objets clients et serveurs écrits dans des langages différents.

La coopération client/serveur est exprimée par le contrat IDL en masquant l'hétérogénéité et les problèmes d'interopérabilité et en séparant l'interface de l'implémentation. Un contrat IDL définit de manière précise les types d'objets (ou interface IDL) et les types de données échangés entre les objets lors des invocations.

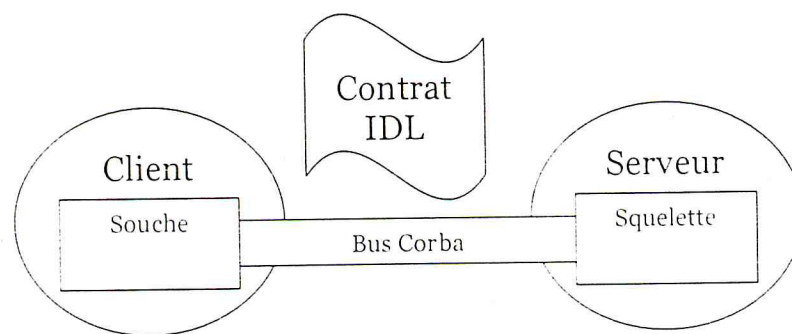


Figure 4-2 : La coopération client/serveur

Ces contrats IDL sont projetés en souches IDL (ou interface d'invocations statiques) dans l'environnement de programmation du client et en squelettes IDL (ou interface de squelettes statiques) dans l'environnement de programmation du serveur.

5. L'architecture CORBA

L'architecture CORBA est basée sur un modèle orienté objet client/serveur. Chaque application peut exporter certaines de ces fonctionnalités (services) sous la forme d'objets CORBA. L'interaction entre applications est matérialisée par les invocations à distance des méthodes des objets. La notion de Client/Serveur intervient seulement lors de l'utilisation d'un objet. L'application qui implante l'objet est le serveur, l'application qui utilise l'objet est le client. Une application peut-être à la fois cliente et serveuse.

La Figure 4-3 montre les composants de l'architecture CORBA permettant la communication entre les objets.

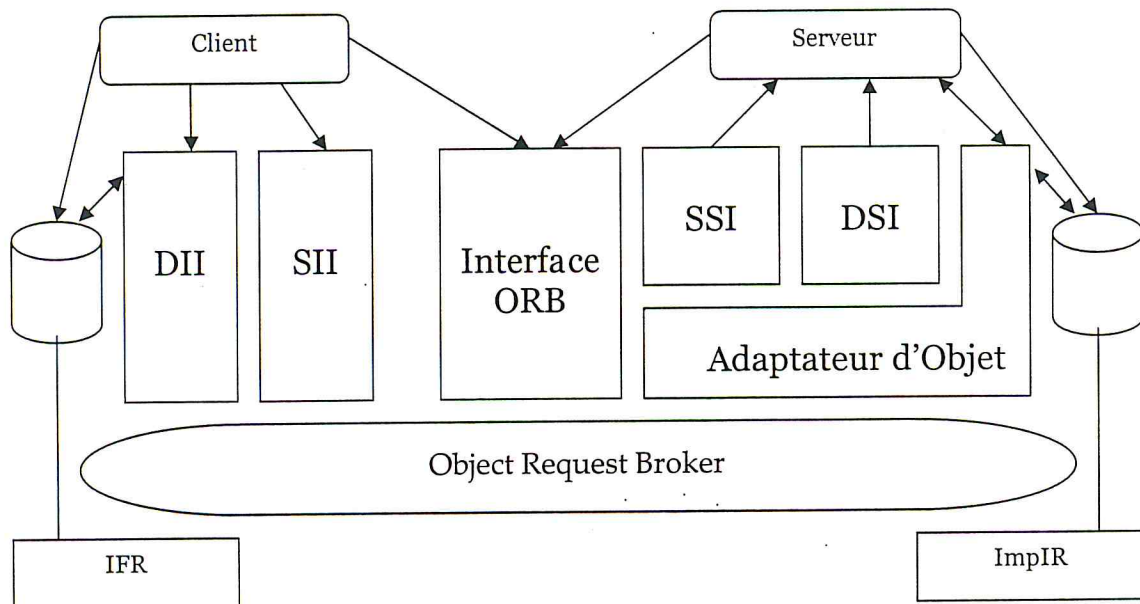


Figure 4-3 : L'architecture CORBA

- **ORB** (*Object Request Broker*) est le noyau de transport des requêtes aux objets.
- **Interface ORB** est utilisée par le client et le serveur pour communiquer de manière standard avec l'ORB indépendamment du fournisseur d'ORB. Il donne les primitives de base telle que l'initialisation pour accéder à l'ORB.
- **SII** (*Static Invocation Interface*) est l'interface d'invocation statique permettant la conversion des invocations et les résultats en messages transmissibles sur le réseau. Cette interface est générée automatiquement par un compilateur IDL (Interface Description Language) vers un langage de programmation cible.
- **SSI** (*Skeleton Static Interface*) est l'interface de squelette statique qui permet de convertir les messages transmis au serveur, d'invoquer le serveur et de traduire les résultats parvenant au client. Cette interface est générée automatiquement par un compilateur IDL (Interface Description Language) vers un langage de programmation cible.
- **DII** (*Dynamic Invocation Interface*) est l'interface d'invocation permettant à un client en cours d'exécution d'invoquer un serveur sans utiliser d'interface SII et donc sans connaître au préalable l'interface IDL de ce serveur.

- **DSI** (*Dynamic Skeleton Interface*) est l'interface du squelette permettant de traiter les requêtes dynamiquement, sans avoir de SSI.
- **OA** (*Object Adaptor*) fournit l'environnement d'exécution des instances d'objets serveurs. Il crée les références d'objets et aiguille les requêtes vers les objets serveurs. Les plus courants sont *BOA* (*Basic OA*) et *POA* (*Portable OA*).
- **IFR** (*InterFace Repository*) est une base de données pour stocker les interfaces des objets serveurs (leurs noms, leurs méthodes et les paramètres de ces méthodes et leurs attributs).
- **ImplR** (*Implementation Repository*) est une base de données pour stocker les implantations des objets serveurs.

6. Le courtier d'objets

Le cœur de l'architecture CORBA est constitué par l'ORB (Object Request Broker). Il joue le rôle de courtier de requêtes entre les objets clients et serveurs permettant ainsi aux objets d'invoquer d'autres objets locaux ou distants dans un environnement hétérogène et d'en recevoir les réponses. Un objet client peut alors invoquer une méthode sur un objet serveur localisé sur la même machine ou sur une autre machine du réseau. La distribution est transparente à l'utilisateur. Le client ne se préoccupe pas de la localisation de l'objet, ni de son langage de programmation, ni de son système d'exploitation. Ce modèle permet de coordonner l'interaction entre deux objets.

6.1 Les caractéristiques

Les caractéristiques fournies par le courtier d'objets sont :

- La liaison avec de nombreux langages de programmation.
- La transparence des invocations. Les requêtes aux objets apparaissent comme locales.
- L'invocation statique et dynamique. Les invocations statiques sont contrôlées à la compilation alors que les invocations dynamiques sont construites et contrôlées à l'exécution.

6.2 Appel de méthodes à distance

L'ORB permet aux objets de réaliser des appels de méthodes d'objets distants de manière transparente. L'appel de méthodes d'objets distribués se fait par introduction d'une *souche* (*stub*). Celle-ci offre les fonctionnalités de l'appelé à l'appelant en les transformant en envois de messages. L'appelant envoie une requête contenant le nom et les paramètres de l'appel codés à l'appelé. Ici, un *squelette* (*skeleton*) reçoit la requête, la décode et réalise l'appel effectif de l'appelé. Ainsi, les paramètres de retour sont transmis sur le réseau

(Figure 4-4). Donc, les communications entre les clients et les serveurs via le réseau se résument à des appels de souches.

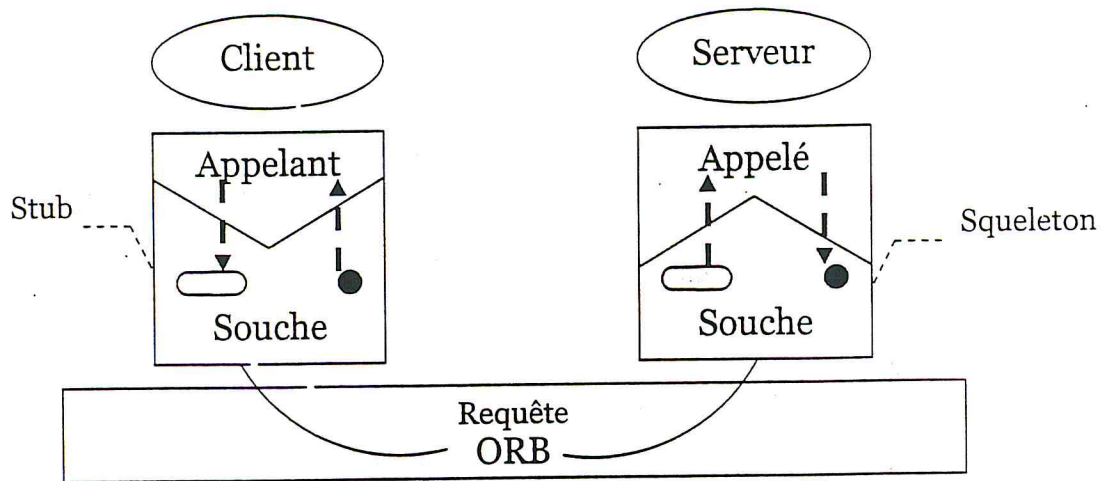


Figure 4-4 : L'envoi de requêtes à un objet distant

L'appel de méthodes se fait de manière statique si on connaît à la compilation les interfaces IDL ou de manière dynamique.

7. Références d'objets

Pour accéder à un serveur, le client utilise *une référence d'objet* (*reference object*) qui est un représentant de l'objet serveur. Une référence d'objet contient l'information permettant d'identifier un objet de manière unique. C'est l'adaptateur d'objet qui associe une référence d'objet à chacune des instances d'objets serveurs. Une référence d'objet contient :

- *Le type de l'objet* pour différencier des types d'objets différents.
- *L'adresse réseau* qui contient une adresse IP et un numéro de port acceptant des invocations de méthodes pour cet objet.
- *Une clé de l'objet* qui est l'identité de l'adaptateur et de l'objet sur cet adaptateur.

8. Service de nommage (Naming service)

Le service de nommage est un mécanisme qui permet d'associer des noms sous forme de chaînes de caractères aux objets car les références attribuées par l'ORB ne sont pas significatives pour les utilisateurs. Comme l'ORB ne traite que des références, le service de nommage associe un nom à une référence. Chaque objet possède un identificateur unique.

Les noms des objets sont organisés de façon hiérarchique. Chaque répertoire est appelé un *contexte de nommage*. Les clients peuvent localiser le

nom de l'objet demandé en parcourant la hiérarchie. Chaque objet nommé est organisé en une structure contenant :

- *l'identificateur* qui est le nom de l'objet,
- *le genre* qui est un attribut décrivant le nom ; par exemple, un objet, une fabrique, etc.
- *la référence de l'objet (IOR)*.

9. Les invocations

9.1 Les invocations statiques

Les invocations statiques nécessitent de connaître les interfaces IDL à la compilation. Par l'IDL, les objets indiquent à leurs clients les opérations possibles. Les souches sont générées automatiquement par un précompilateur conforme à CORBA à partir des fichiers IDL. Le compilateur de ce code et de celui fourni par le programmeur afin d'implanter les objets génère au moins quatre types de fichiers de sortie :

- 1- des fichiers d'import qui décrivent les objets pour un référentiel d'interfaces.
- 2- des souches clients pour les méthodes définies en IDL. Ces souches sont appelées par un programme client quand il a besoin d'accéder statiquement aux services définis en IDL via l'ORB.
- 3- des souches serveurs qui appellent les méthodes sur le serveur. Elles sont appelées parfois des interfaces entrantes.
- 4- le code qui implante les classes serveurs.

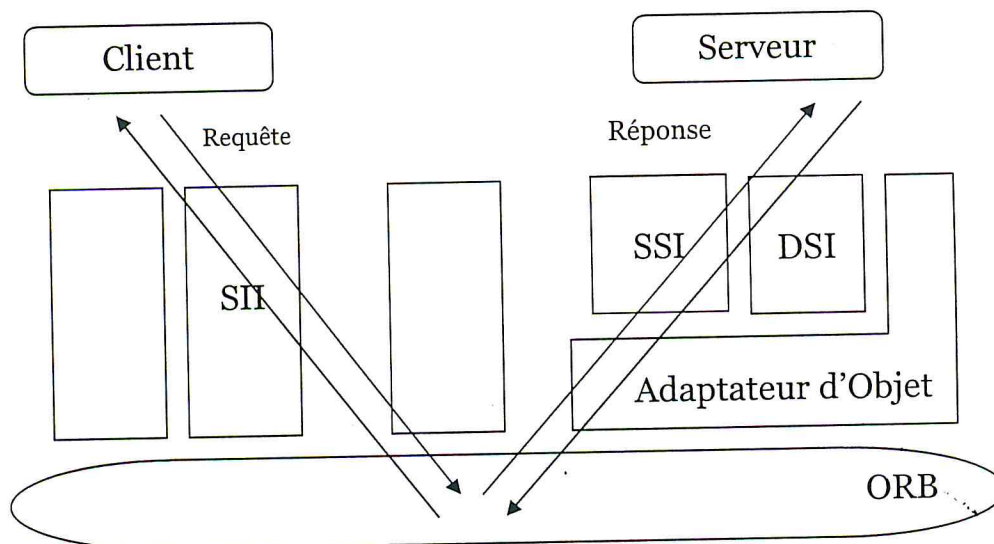


Figure 4-5 : Invocation statique

4.4 Remote Method Invocation - Internet Inter-ORB Protocol (RMI-IIOP)

RMI-IIOP permet aux applications J2EE d'accéder aux services CORBA définis par les composants qui existent à l'extérieur du produit J2EE. Cette API peut être utilisée aussi pour accéder aux composants Enterprise JavaBeans dans l'application.

4.5 Java IDL

Cette API permet aux composants d'une application J2EE d'invoquer des objets CORBA externes en utilisant le protocole IIOP. Ces objets CORBA sont complètement indépendants de l'application J2EE et peuvent être utilisés par d'autres applications.

4.6 JDBC API

JDBC est une API standard qui permet la connexion avec les systèmes de base de données. Les composants d'une application peuvent utiliser JDBC pour traiter les données à partir d'une base de données relationnelle ou autre.

4.7 Java Message Service (JMS)

Cette API définit un mécanisme standard pour les composants pour envoyer et recevoir des messages. Elle permet l'usage des systèmes de la messagerie de l'entreprise (IBM MQSeries par exemple).

4.8 Java Naming and Directory Interface (JNDI)

Cette API permet aux composants J2EE de rechercher d'autres objets distants. L'API JNDI a été conçue pour standardiser l'accès à une variété d'interfaces de nommage et d'annuaires.

4.9 JavaMail

JavaMail permet aux applications d'envoyer des notifications email (e-mail notifications).

4.10 JavaBeans Activation Framework (JAF)

JAF API fournit une framework pour traiter les différents types de MIME¹.

4.11 Java API for XML Parsing (JAXP)

JAXP fournit un support pour analyser les documents XML (eXtended Mark-up Language).

¹ Multi-purpose Internet Mail Extension : norme permettant le transfert de tous types de fichiers par courrier électronique sur le réseau internet.

5. Les containers J2EE

Les containers fournissent le support d'exécution pour les composants d'application J2EE. Les composants d'une application J2EE ne communiquent jamais entre eux directement. Ils utilisent les protocoles et les méthodes fournis par le container pour agir les uns avec les autres et avec les services de la plate-forme. Interposer un container entre les composants d'une application et les services J2EE, lui permet d'injecter d'une manière transparente les services définis dans le descripteur de déploiement du composant. Parmi ces services il y a :

- *La gestion des transactions.* Le composant doit seulement indiquer les méthodes qui doivent s'exécuter en une seule unité et le container prend en charge le reste.
- *Le contrôle de sécurité.* Le composant doit juste indiquer les droits d'accès.
- *Java Naming and Directory Interface (JNDI).* Pour localiser les objets distribués.
- *Gestion des connexions distantes.* Gestion de la communication entre le client et le serveur.

Il y a plusieurs types de containers :

5.1 Le container des applications client

Il gère l'exécution des composants de l'application client. Les composants de l'application client et leur container s'exécutent dans le client.

5.2 Le container des applets

Il gère l'exécution des applets. Il consiste en un browser Web et un plug-in Java. Ils s'exécutent dans le client.

5.3 Le container Web

Il gère l'exécution des composants Web (Pages JSP et Servlets) de l'application. Les composants Web et leur container s'exécutent dans le serveur J2EE.

5.4 Le container Enterprise JavaBeans

Il gère l'exécution des enterprise beans. Les EJBs et leur container s'exécutent dans le serveur J2EE.

6. Emballage et déploiement

La plate-forme J2EE permet aux développeurs de créer les applications sous forme de composants réutilisables. Le processus d'assemblage des composants en modules et les modules en application est appelé *emballage*

(Packaging). Le processus d'installation et d'adaptation d'une application dans un environnement opérationnel est appelé *déploiement* (Deployment).

Une application J2EE et chacun de ses modules a son propre descripteur de déploiement. Un descripteur de déploiement est un document XML qui décrit la configuration de déploiement d'un composant. Par exemple, un descripteur de déploiement d'un module enterprise bean déclare les attributs des transactions et les droits d'accès pour un enterprise bean (Figure 5-4).

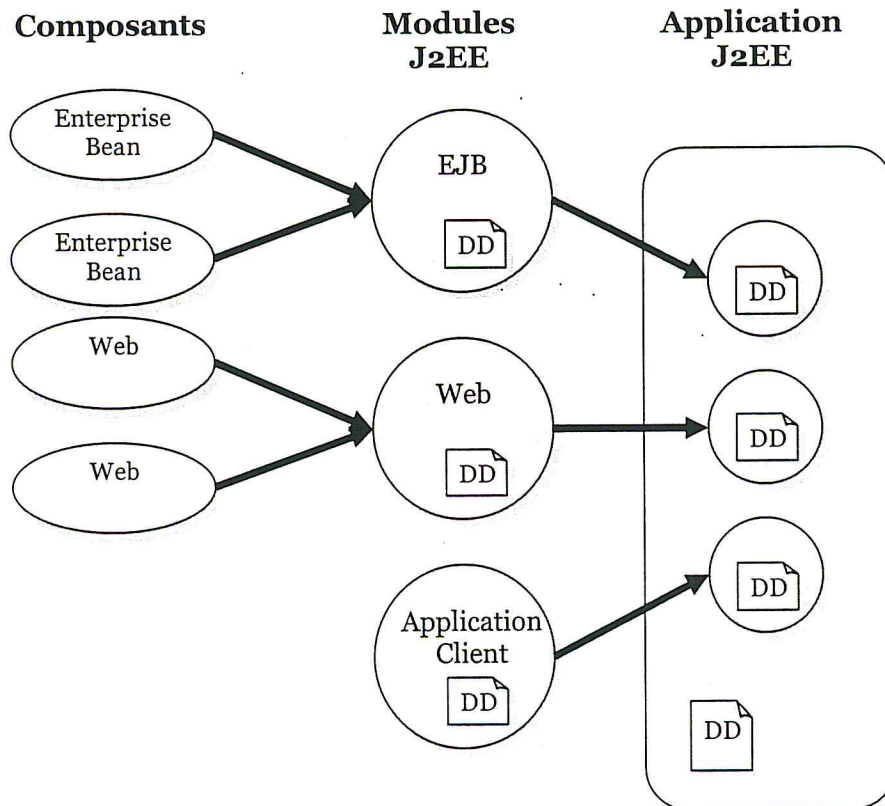


Figure 5-4 : Descripteurs de déploiement J2EE

L'application J2EE est délivrée dans un fichier Enterprise ARchive (EAR). Un fichier EAR est un fichier Java ARchive standard avec l'extension `.ear`. Les composants Web sont assemblés dans les fichiers Web ARchive (WAR), les Enterprise JavaBeans dans les fichiers EJB JAR, et les applications client dans les fichiers JAR.

Durant le déploiement, l'application est installée dans la plate-forme J2EE, configurée, et intégrée dans l'infrastructure existante.

7. Les rôles de la plate-forme

La plate-forme J2EE définit plusieurs rôles distincts dans le cycle de vie et de déploiement d'une application J2EE : fournisseur de produit J2EE, fournisseur des composants d'application, assembleur d'application, déployeur, administrateur de système, et le fournisseur d'outils. En général, ces rôles sont définis pour aider à partager les tâches entre les différentes parties pendant le développement, le déploiement, et l'exécution d'une application J2EE. Les tâches telles que l'administration de système sont communes et non pas spécifiques au J2EE.

7.1 Fournisseur de produit J2EE

Un fournisseur de produit J2EE, typiquement un vendeur de système d'exploitation, un vendeur de système de base de données, ou un vendeur de serveur Web, rend disponible la plate-forme J2EE, les APIs, et les autres caractéristiques définies dans la spécification J2EE.

7.2 Fournisseur des composants d'application

Les fournisseurs des composants sont les personnes qui développent les composants Web, entreprise beans, applets, ou les applications client pour les utiliser dans les applications J2EE.

Le développeur d'entreprise beans écrit et compile le code source et délivre un fichier EJB JAR qui contient les entreprise beans avec leurs descripteurs.

Le développeur Web écrit et compile le code source des servlets, écrit les pages JSP et HTML, et délivre un fichier WAR.

Le développeur d'application client écrit et compile le code source et délivre un fichier JAR qui contient l'application client.

7.3 Assembleur d'application

L'assembleur d'application prend l'ensemble des composants développés par le fournisseur des composants et les assemble dans une application J2EE et délivre un fichier EAR qui contient l'application.

7.4 Déployeur

Le déployeur est un expert dans un environnement opérationnel spécifique. Il est responsable de déployer les composants J2EE et les applications dans cet environnement.

7.5 Administrateur de système

L'administrateur de système est responsable de la configuration et l'administration de l'infrastructure et le réseau de l'entreprise.

7.6 Fournisseur d'outils

Le fournisseur d'outils fournit les outils utilisés pour le développement, l'emballage, et le déploiement des applications.

8. Entreprise JavaBeans

Les entreprise beans sont les composants côté serveur qui encapsulent la logique métier de l'application. La logique métier est le code qui accomplit l'objectif de l'application. Par exemple, l'entreprise bean commande a une méthode `ajouterArticle()` et une méthode `retirerArticle()`. En invoquant ces méthodes, les clients distants peuvent accéder au service gestion des commandes fourni par l'application.

La gestion de persistance, transactions, sécurité, connexion avec les clients, et de cycle de vie est assurée par le container de l'EJB. Donc les développeurs peuvent concentrer leurs efforts sur les méthodes métiers de l'EJB.

8.1 Types d'entreprise beans

Il existe trois sortes d'entreprise beans : les beans session, les beans entity, et les beans message-driven.

8.2 Les beans session

Les entreprise beans du type session correspondent à des objets applicatifs non persistants. Ils sont utilisés pour traiter des opérations ne nécessitant pas la conservation des données des objets utilisés. Ils peuvent être de deux sortes:

- *Sans état (stateless)*. Les valeurs des attributs sont perdues entre deux utilisations. Et, puisqu'ils ne maintiennent pas d'état, ils peuvent être utilisés par plusieurs clients.
- *Avec état (stateful)*. Les valeurs des attributs sont conservées mais pas au-delà de la durée de vie de l'EJB. Un bean stateful s'exécute au nom d'un seul client.

8.3 Les beans entity

Les EJBs du type entity sont des objets persistants. Les attributs d'un EJB entity sont mis en correspondance avec les champs d'une table d'une base de données. La connexion et les requêtes SQL à la base de données, sont automatiquement gérées par le container de l'EJB.

Le container peut gérer la persistance de l'EJB, dans ce cas l'EJB est dit *CMP (Container Managed Persistence)*. Ou bien l'EJB gère sa propre persistance, dans ce cas il est dit *BMP (Bean Managed Persistence)*. Si la persistance est gérée

par l'EJB, le développeur doit écrire tout le code pour accéder à la base de données.

Tableau 11 : Exemple de Session/Entity bean

Session Bean	Entity Bean
Gestion de compte	Compte bancaire
Vérificateur de Compte Bancaire	Carte de crédit
Système d'entrée gestion de commandes	Commande, ligne de commande
Gestion de catalogue	Produits
Gestion d'achats	Commande, ligne de commande

8.4 Les beans Message-driven

Ils se comportent comme des attentifs Java Message Service (JMS), ils traitent les messages d'une manière asynchrone.

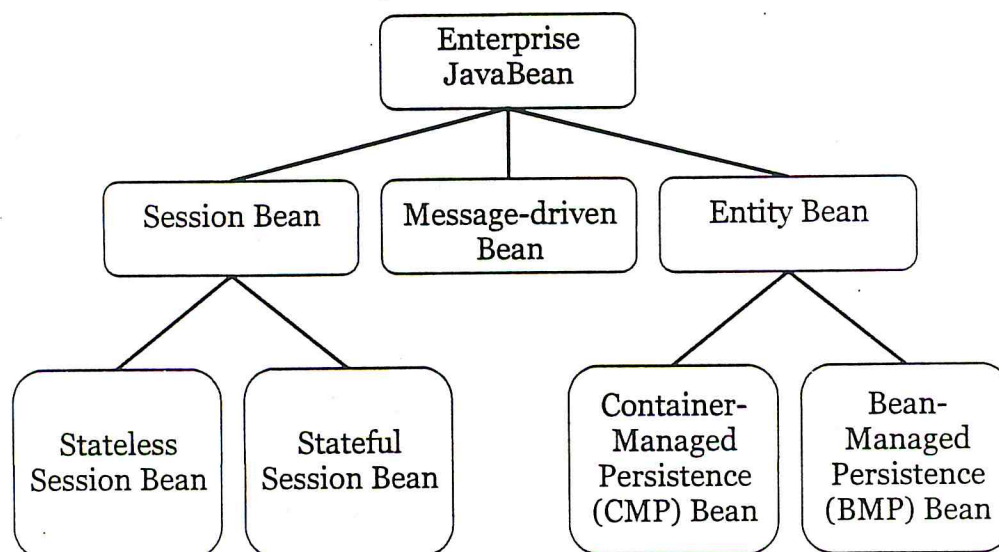


Figure 5-5 : Enterprise JavaBeans.

8.5 L'accès à un Enterprise Bean

Un EJB doit définir deux interfaces:

- Une interface distante fonctionnelle (*remote interface*) qui contient les méthodes métiers de l'objet.
- Une interface d'un objet de fabrique de référence (*home interface*) qui contient les méthodes pour rechercher et contrôler le cycle de vie de l'EJB.

Les clients utilisent ces deux interfaces pour accéder à l'EJB. Un client doit d'abord obtenir une référence sur l'interface Home en utilisant JNDI (Java

Naming and Directory Interface). Le client doit connaître à priori le nom symbolique de l'objet. Ensuite, le client peut utiliser la méthode `Create()`, `Find()`, ou n'importe quelle autre méthode qui permet d'obtenir une référence sur une interface `Remote`. Le client peut appeler maintenant les méthodes de l'EJB.

La Figure 5-6 illustre comment un client doit interagir avec un EJB.

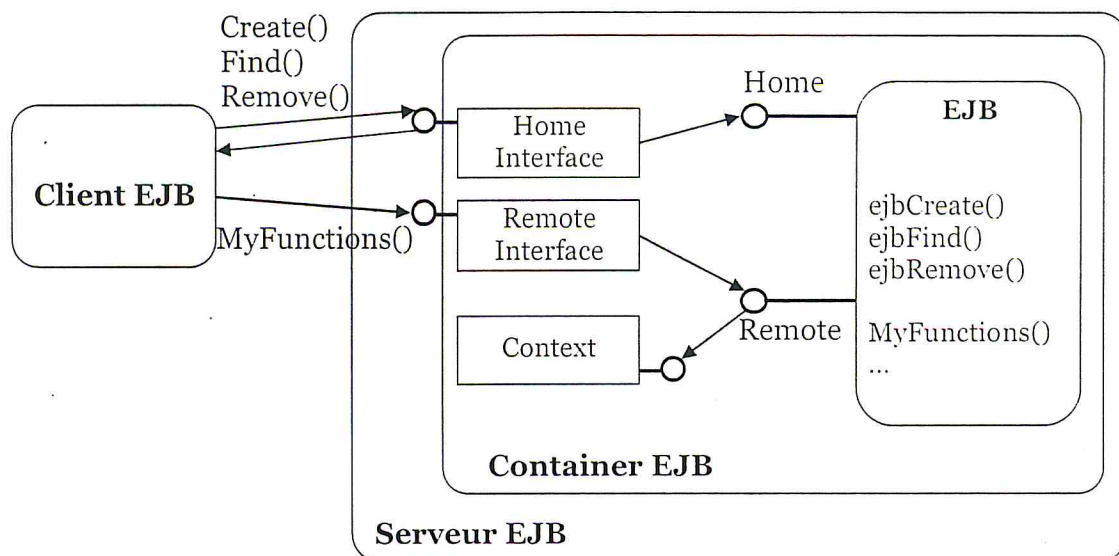


Figure 5-6 : Interaction entre un client et un EJB.

9. JavaServer Pages

JavaServer Pages est un standard Java défini au dessus de la technologie des servlets. Le but des JSPs est de simplifier la création et la gestion du contenu des pages Web dynamiques.

JSPs permettent de combiner l'HTML (XML) des pages Web avec du code Java dans le même document. Le code Java est entouré par des balises spéciales pour dire au container des JSPs que ce code doit être utilisé pour générer une servlet.

Exemple d'une page JSP:

```
<html>
<head>
  <title>Ma première page JSP</title>
</head>
<body>
  <%@ page language="java" %>
  <% System.out.println("Hello World"); %>
  <%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
  <p><c:out value="C'est encore plus sympa avec JSTL" /></p>
</body>
</html>
```

Il y a cinq principales balises JSP:

- *Balise de déclaration* (`<%! %>`). Permet la déclaration des variables ou des méthodes.
- *Balise des expressions* (`<%= %>`). Elle remplace `out.println()`.
- *Balise de directive* (`<%@ directive ... %>`). Donne des informations sur la page au serveur JSP.
- *Balise de Scriptlet* (`<% ... %>`). Pour écrire n'importe quel code Java valide.
- *Balise d'actions*. Permet d'utiliser les Javabeans et passer le contrôle entre les pages.

De plus, JavaServer Pages permet de définir des balises personnalisées.

10. La sécurité

La plate-forme J2EE fournit un service de sécurité qui assure que les ressources ne sont accédées que par les utilisateurs autorisés. Les droits d'accès sont spécifiés soit par déclaration ou soit par programmation. *La sécurité est déclarative* si les droits sont spécifiés dans les descripteurs de déploiement. Dans *la sécurité par programmation*, les droits sont contrôlés dans le programme en utilisant les APIs de sécurité.

Le contrôle d'accès implique deux étapes :

- 5- *L'authentification*. Une entité doit affirmer son identité à travers l'authentification. Elle fournit généralement les données de l'authentification (telles que le nom et le mot de passe).
- 6- *L'autorisation*. A chaque fois qu'une entité tente d'accéder à une ressource, le système doit vérifier si elle en a le droit.

11. Conclusion

La plate-forme Java 2, Enterprise Edition forme un standard pour le développement des applications distribuées. Elle décharge les développeurs des

tâches communes telles que la gestion des transaction et la sécurité, et leur permet de concentrer sur le problème métier de l'application. Ainsi, elle définit des rôles distincts dans le développement ce qui permettra de diviser le travail entre les différentes équipes de développement.

CHAPITRE VI

9.2 Les invocations dynamiques

L'appel de méthodes de manière dynamique permet à un client en cours d'exécution d'émettre des requêtes sur des objets. La requête contient l'objet à appeler, la méthode à exécuter et la liste d'arguments que le code client obtient à partir d'un référentiel d'interfaces ou d'une source exécutable similaire.

L'appel dynamique est plus souple que l'appel statique mais plus complexe.

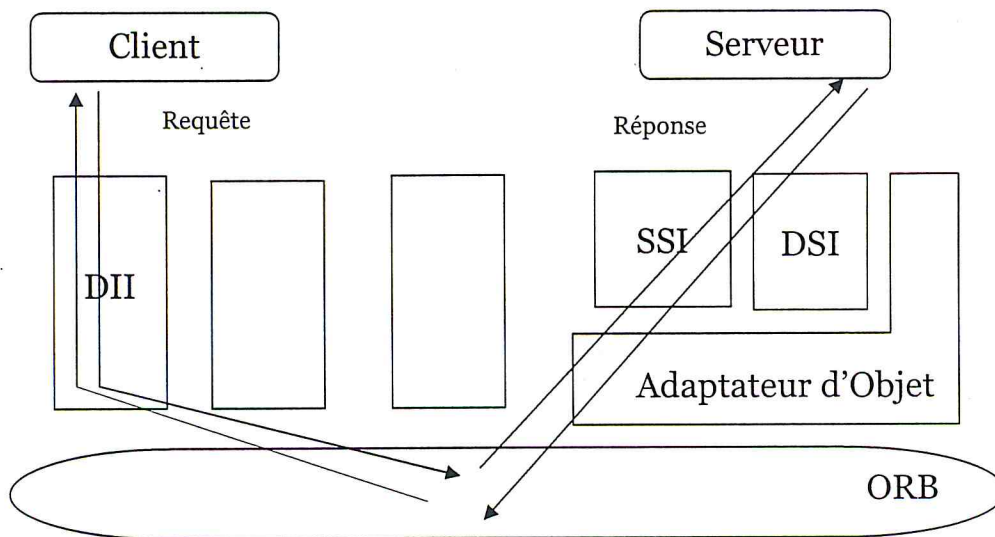


Figure 4-6 : Invocation dynamique

Les invocations faites sur un serveur sont réalisées via une interface écrite dans le langage standard IDL (Interface Definition Language) qui décrit les services offerts par le serveur indépendamment de son langage d'implémentation et du système sur lequel il s'exécute.

10. GIOP/IIOP (General Inter-ORB Protocol/Internet Inter-ORB Protocol)

GIOP (General Inter-ORB Protocol) spécifie un ensemble de formats pour les messages ainsi qu'une représentation commune des données échangées entre les ORBs.

La représentation des données communes est basée sur la spécification CDR (Common Data Representation).

IIOP (Internet Inter-ORB Protocol) est l'implémentation du protocole GIOP basé sur TCP/IP.

11. Conclusion

L'OMG a défini le standard CORBA qui permet de développer des applications orientées objets distribuées dans la perspective de créer une infrastructure qui permet à des objets de communiquer indépendamment des langages de programmation, des systèmes d'exploitation et des protocoles de communication.

CHAPITRE V

Chapitre 5: La plate-forme Java 2, Enterprise Edition

1. Introduction

Le développement d'un système distribué est devenu presque une nécessité pour l'entreprise. Ceci a rapidement rendu la vie des développeurs un vrai cauchemar. Les développeurs doivent assurer que l'application est performante, sécurisée, transactionnelle, et réutilisable. Ces considérations sont répétitives et communes à toutes les applications distribuées.

Pour réduire les coûts et les délais de développement des applications distribuées, la plate-forme Java 2, Enterprise Edition (J2EE) fournit une approche basée sur les composants pour développer, assembler, et déployer les applications entreprise.

Ce chapitre donne une vue générale sur la plate-forme J2EE mais ne couvre pas toute la spécification J2EE. La spécification de J2EE est disponible sur [\[java.sun.com/j2ee\]](http://java.sun.com/j2ee).

2. L'architecture multi-étages (multi-tiers)

Dans les années 90s, les systèmes informatiques utilisent l'architecture client/serveur. L'interface utilisateur s'exécutait généralement dans un ordinateur de bureau (desktop computer). Ceci était *l'étage client (client tier)*. Les données utilisées par le client résidaient dans une base de données et étaient fournies par un serveur (Figure 5-1).

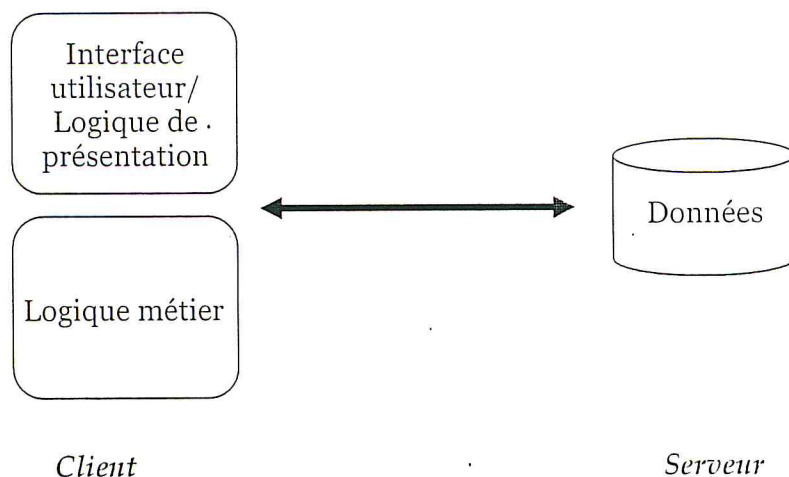


Figure 5-1 : L'architecture Client/Serveur

A travers une longue expérience, les développeurs se sont aperçus que maintenir un système distribué flexible (dynamique) dans le modèle Client/Serveur est une tâche très difficile. Par exemple, la logique métier était dans l'application client. A chaque fois que celle-ci a besoin d'une modification, l'application devrait être réinstallée dans toutes les machines client.

Une fois les limitations du modèle Client/Serveur sont devenues apparentes, les développeurs ont commencé à chercher une meilleure solution. Le résultat est le *modèle multi-étages* (Figure 5-2).

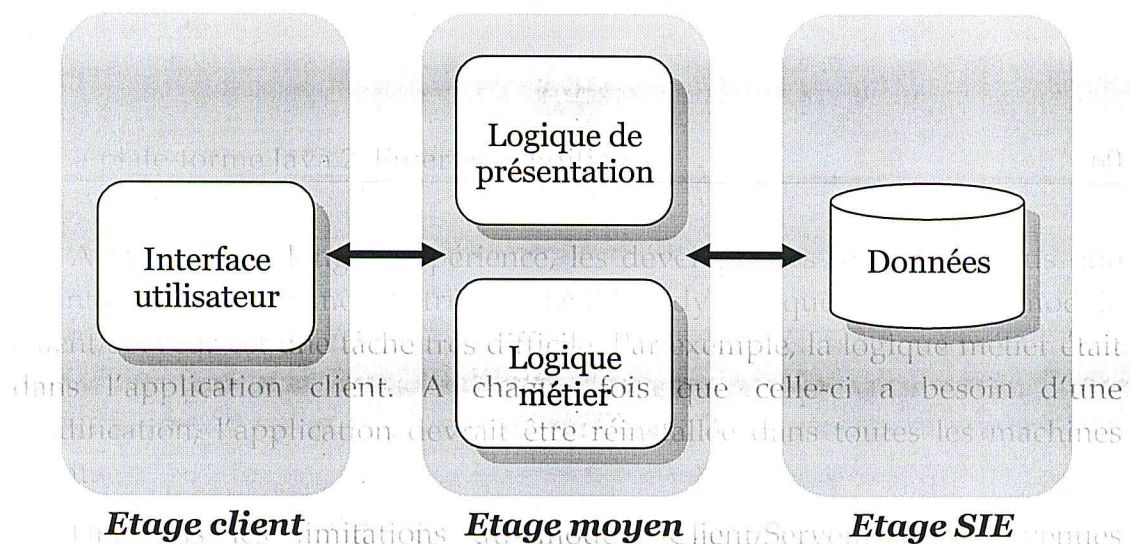


Figure 5-2 : L'architecture multi-étages

Dans le modèle multi-étages, la logique de présentation et la logique métier sont migrées vers l'étage moyen. Maintenant la mise à jour se fait dans une seule machine.

La plate forme J2EE fournit une architecture multi-étages. Elle définit l'*étage client*, l'*étage moyen* (constitué d'un ou plusieurs sous-étages), et un dernier étage qui fournit les services des systèmes d'informations existants. L'étage client supporte plusieurs types de client. L'étage moyen supporte les services client à travers les containers Web dans l'*étage Web* et supporte les services composant la logique métier à travers les containers Enterprise JavaBeans (EJB) dans l'*étage EJB*. L'*étage de système d'information de l'entreprise* (SIE) supporte l'accès aux systèmes d'informations existants.

Etage client

Etage moyen

Etage SIE

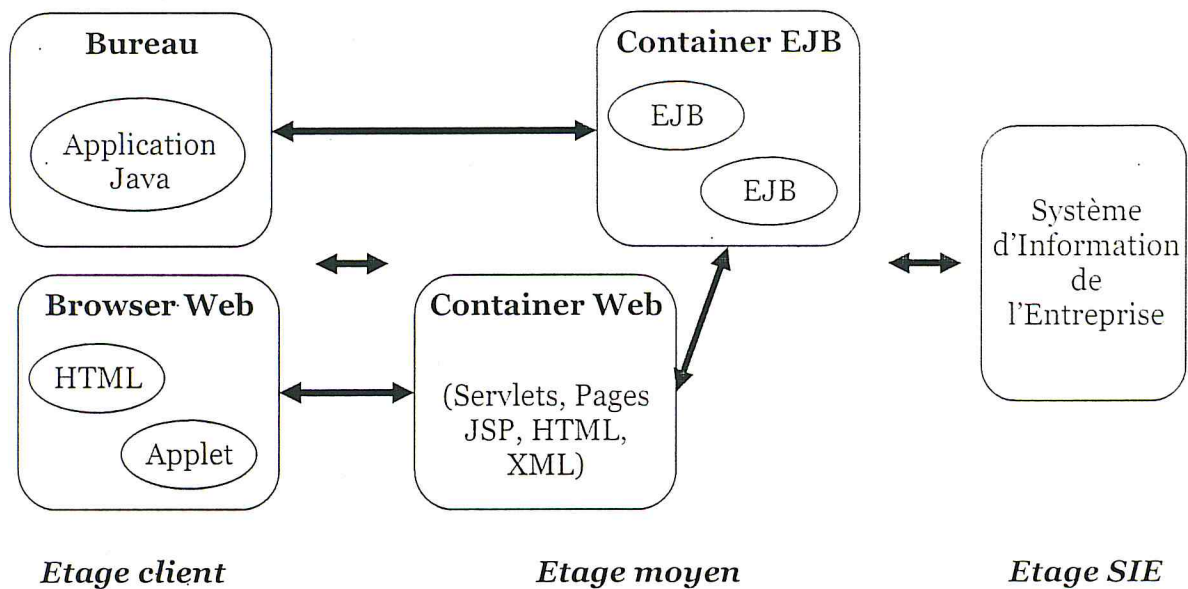


Figure 5-3 : L'architecture de Java 2 Enterprise Edition

3. Les composants de l'application

Une application J2EE est un ensemble de composants. Chaque type de composant doit s'exécuter dans un container (voir 5-Les containers J2EE). Ces composants sont écrits en langage Java et compilés de la même manière qu'un autre programme Java. Les composants d'une application peuvent être emballés dans un fichier JAR, incluent les descripteurs de déploiement, et fournissent des informations aux containers qui les gèrent (voir 6-Emballage et déploiement). La spécification J2EE définit les composants suivants :

- *Les applications client et les applets* sont les composants qui s'exécutent dans le client.
- *Les servlets et les JavaServer Pages (JSP)* sont des composants Web qui s'exécutent dans le serveur.
- *Enterprise JavaBeans (EJB) (Enterprise Beans)* sont des composants métiers qui s'exécutent dans le serveur.

3.1 Les applications client

Les applications client sont des programmes Java qui s'exécutent dans un ordinateur de bureau. Elles ont une interface graphique. Elles accèdent directement aux EJBs (Enterprise Beans) qui s'exécutent dans l'étage métier (étage EJB).

3.2 Les applets

Une applet est un composant qui s'exécute dans un browser Web, c'est-à-dire dans la machine virtuelle du client. Elles peuvent être utilisées pour traiter la logique de présentation et fournir une interface utilisateur pour l'application J2EE. De simples pages HTML peuvent aussi être utilisées.

3.3 Les Servlets et les JavaServer Pages

Les servlets sont des programmes Java intégrés avec le serveur Web pour traiter dynamiquement les requêtes des clients et fournir des réponses. Les servlets peuvent être utilisées pour remplacer les programmes CGI (*Common Getway Interface*). JavaServer Pages sont une extension de la technologie des servlets. C'est un moyen simplifié pour créer des servlets. Bien que les pages JSP apparaissent différentes pendant le développement, elles sont compilées en servlets par le serveur Web lors de leurs premières exécution.

Ces composants sont exécutés dans le serveur J2EE pour générer un contenu dynamique aux clients. Ils utilisent les éléments HTML ou XML plus les différentes scripts. Ils sont appelés *les composants Web*.

3.4 Enterprise JavaBeans

Enterprise JavaBeans (ou simplement *enterprise beans*) sont des composants coté serveur qui contiennent la logique métier de l'application. Ce sont des composants transactionnels, sécurisés et multi-utilisateurs.

Il existe trois sortes d'enterprise beans : les *beans session*, les *beans entity*, et les *beans message-driven*. (Voir 8-Enterprise JavaBeans).

4. Les services standards

Les composants doivent utiliser un ensemble de services standard pour interagir avec les autres. Le standard J2EE inclut les services suivants :

4.1 HTTP

Les APIs (Application Programming Interface) de HTTP coté client sont définies par le package `java.net`. Les APIs HTTP coté serveur sont définies par les interfaces des servlets et JSP.

4.2 HTTPS

Les mêmes APIs de HTTP sont utilisées au-dessus du protocole SSL.

4.3 Java Transaction API (JTA)

JTA fournit un moyen aux composants J2EE et aux clients pour gérer leurs propres transactions. JTA permet aussi la participation de plusieurs composants à une seule transaction.

1. Introduction

En bref, notre travail consiste à créer un catalogue distribué qui permet la gestion d'un réseau documentaire. Ce nouveau système va permettre aux utilisateurs d'interroger plusieurs catalogues à la fois.

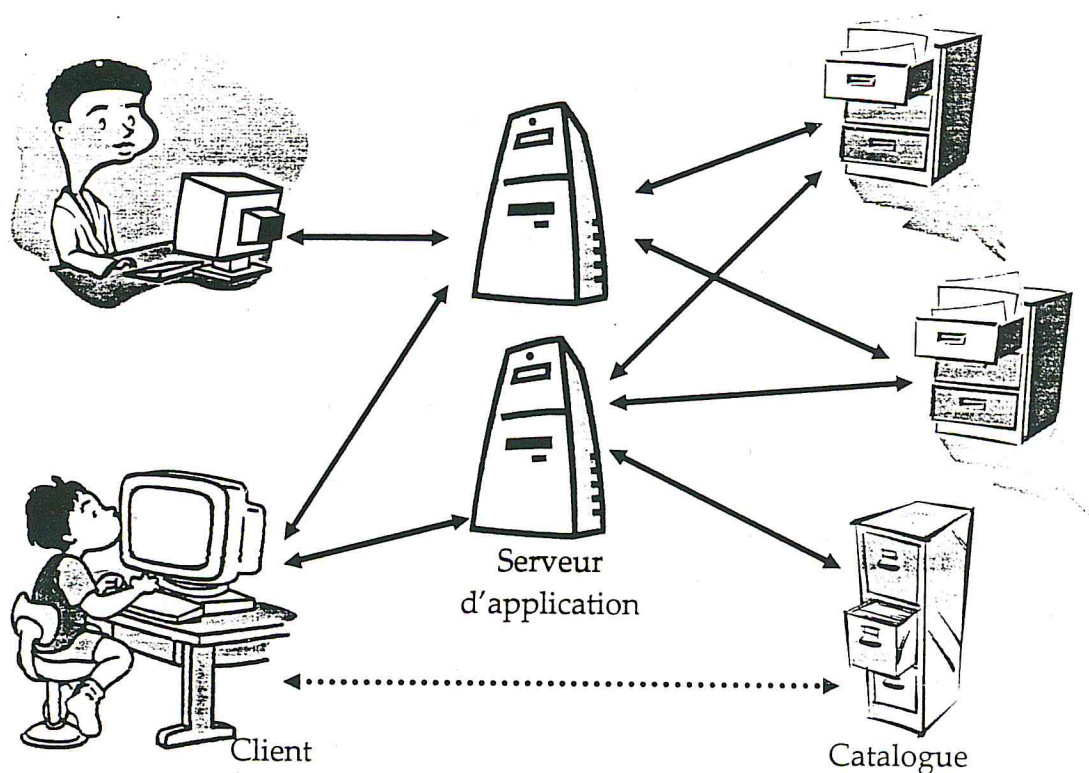


Figure 6-1 : Notre réseau documentaire.

Tous les catalogues qui composent le réseau documentaire sont connectés à des serveurs d'applications auxquels les clients peuvent trouver les services de tous les catalogues.

Le système joue le rôle d'un OPAC (il fournit les opérations de recherche) et permet aussi l'administration des différents catalogues.

2. Présentation du système

Le premier but du système est de créer un OPAC qui permet l'interrogation de tous les catalogues de l'union. Cet OPAC qui regroupera les catalogues de toute l'Algérie permettra aux algériens de gagner le temps perdu dans le feuilletage des catalogues papier.

Le deuxième but, c'est que le système va permettre d'échanger les notices bibliographiques entre les catalogues. Ainsi, peut être qu'une agence de catalogage verra le jour en Algérie.

2.1 Un simple scénario de recherche

- 1- Un utilisateur se connecte au système.
- 2- Il choisit parmi une liste les catalogues à interroger.
- 3- Il entre sa requête puis lance la recherche.
- 4- Il reçoit les résultats.

2.2 La relation entre les catalogues

Comme la Figure 6-1 (page 73) le montre, le système est composé d'un ensemble de catalogues reliés tous à un serveur.

Les catalogues sont complètement séparés les uns des autres. Le but du système n'est pas de fournir un moyen direct qui permet la communication entre les catalogues mais d'assurer les échanges des notices bibliographiques.

Les échanges bibliographiques se font par **import**. C'est-à-dire une bibliothèque recherche les notices qui l'intéressent dans un catalogue, et décide alors d'importer les résultats ou non dans son propre catalogue.

2.3 Administration et sécurité des catalogues

Le système fournit une interface qui permet l'administration des catalogues. Un administrateur d'un catalogue peut insérer, modifier, et supprimer des notices. Il peut aussi importer les notices d'un autre catalogue.

En plus des administrateurs, il y a les gestionnaires qui peuvent administrer une partie du catalogue ou seulement ont le droit de visualiser des notices que des utilisateurs anonymes n'ont pas l'autorisation de voir.

La sécurité est assurée par le catalogue lui-même. C'est-à-dire c'est lui qui contient et gère la liste des droits d'accès.

Ces mécanismes de sécurité permettent aux bibliothèques de contrôler les notices qui sont partagées avec l'ensemble du système.

2.4 Les catalogues et les annuaires

Techniquement, un catalogue contient des données de petite taille (de simples notices). Ces notices sont exprimées sous forme d'attributs (titre, auteur, ...). Le rapport Lecture/Ecriture est trop élevé. Il faut avoir une grande capacité de recherche. De plus, ces données sont partagées avec l'ensemble du système ce qui nécessite un accès normalisé.

Le choix le plus raisonnable pour implémenter un catalogue est donc d'utiliser un annuaire.

3. Les Composants du système

Le système est composé d'un ou plusieurs catalogues implémentés sous forme d'annuaires et d'une application J2EE (Figure 6-2).

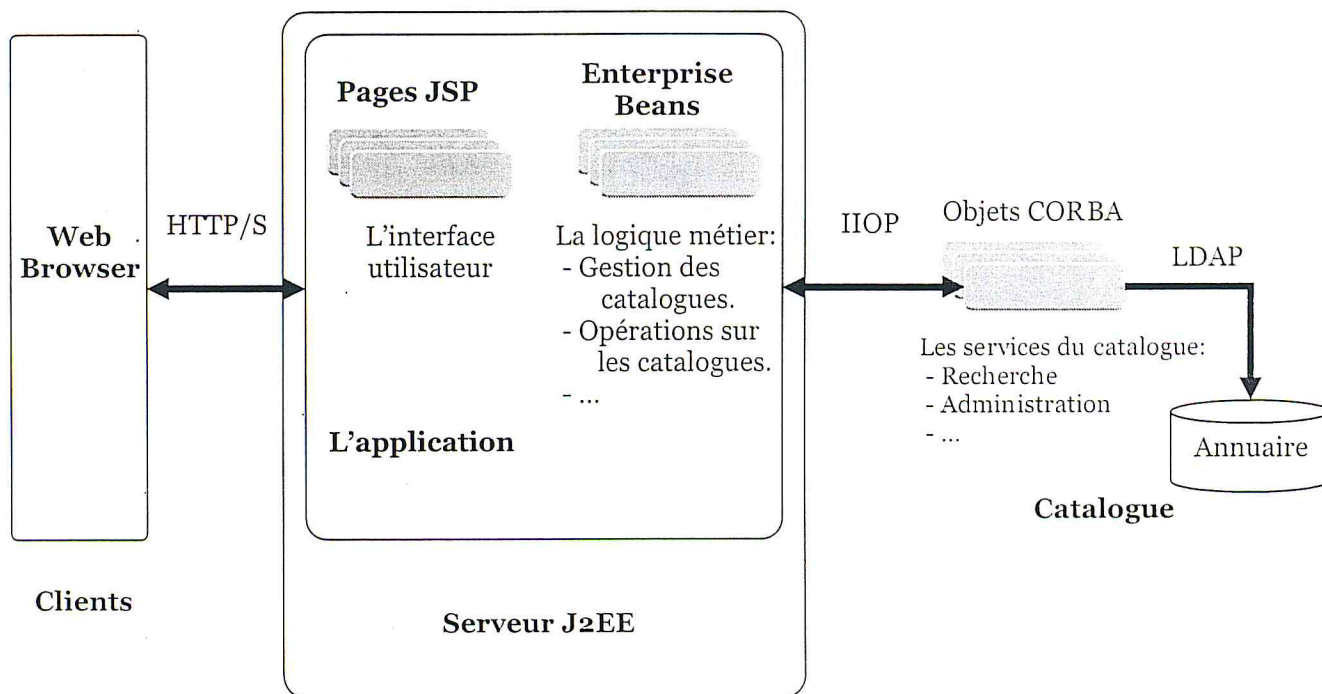


Figure 6-2 : Les composants du système.

Les services du catalogue sont présentés par des objets CORBA. La logique métier est implémentée sous forme d'Enterprise JavaBeans. Et la logique de présentation par des pages JSPs. Les clients sont des **clients légers**, ils ont seulement besoin d'un browser Web.

4. Conception du système

4.1 Conception de l'annuaire (catalogue)

La conception de l'annuaire consiste à concevoir le schéma LDAP, le DIT (Directory Information Tree), et les objets de services.

4.1.1 Le schéma LDAP

L'annuaire va stocker les notices bibliographiques qui deviennent des « **entrées** ». Et les éléments qui composent une notice deviennent des

« **attributs** ». Les éléments qui décrivent une notice sont définis dans les règles du catalogage adoptées par la bibliothèque (ISBD ou AACR).

Le schéma doit définir un attribut pour chaque élément. Le schéma de la Figure 6-3 définit un attribut nommé « name » qui stocke des valeurs de la syntaxe `directoryString` (UTF-8). La syntaxe est spécifiée par un OID, par exemple l’OID 1.3.6.1.4.1.1466.115.121.1.15 identifie la syntaxe `directoryString`. De plus, `EQUALITY` et `SUBSTR` signifient qu’il faut ignorer la casse lors de la comparaison.

Chaque attribut a sa propre OID ("2.5.4.41" pour l’attribut "name"). Les OIDs sont distribués gratuitement par l’IANA (Internet Assigned Numbers Authority¹).

```

attributetype ( 2.5.4.41 NAME 'name'
    EQUALITY caseIgnoreMatch
    SUBSTR caseIgnoreSubstringsMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15)
attributetype (1.3.6.1.4.1.200 NAME 'title' SUP name )
attributetype ( 2.5.4.4.2.1 NAME 'ISBN'
    EQUALITY numericStringMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.36)

```

Figure 6-3 : Exemple de définition d’attribut.

Le schéma standard définit les éléments communs tels que le nom, le titre, l’adresse, etc. Il ne faut pas les redéfinir.

Les règles de catalogage définissent plusieurs types de document (monographie, publications en série, thèses et mémoires, documents cartographiques, ressources électroniques, musique imprimée, documents sonores, microfiches, ...etc.). Le schéma doit définir une **classe d’objets** pour chaque type de document.

Le schéma de la Figure 6-4 définit une classe d’objet nommée « monograph » qui contient deux attributs obligatoires (Title et ISBN) et six attributs optionnels (Author, Published, Description, Keyword, Abstract, et Edition).

```

objectclass ( 1.1.2.2.1 NAME 'monograph' SUP top
    MUST ( Title $ ISBN)
    MAY (Author $ Published $ Description $
        Keyword $ Abstract $ Edition)
)

```

Figure 6-4 : Exemple de définition d’une classe d’objet.

¹ <http://www.iana.org/>

Les schémas LDAP sont extensibles, ce qui permet facilement d'ajouter les nouveaux éléments et les nouveaux types de documents. C'est-à-dire le système est extensible pour les nouveaux types de données.

4.1.2 Le DIT (Directory Information Tree)

Habituellement, un document se trouve dans la bibliothèque. Celle-ci forme une unité d'organisation au sein d'une organisation. Les feuilles du DIT représentent les notices. Donc le DIT va être plat (c'est-à-dire les notices sont directement collées à bibliothèque) (Figure 6-5).

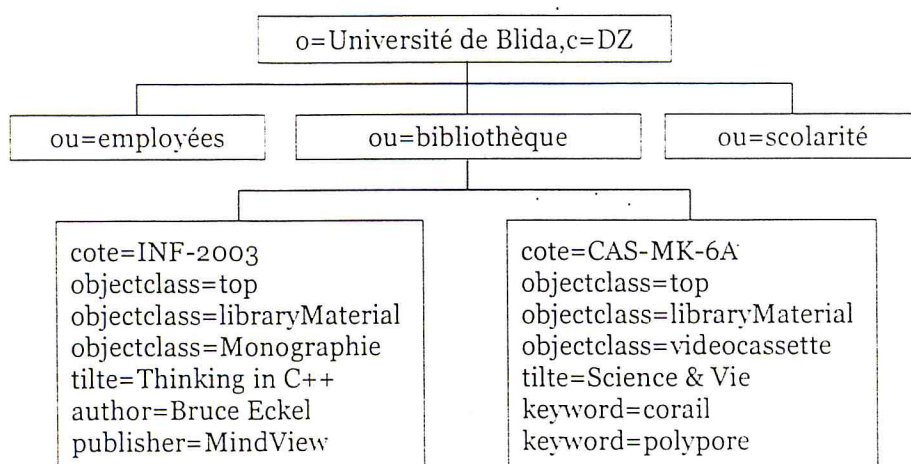


Figure 6-5 : le DIT.

4.1.3 Le nom distinctif

Les bibliothécaires attribuent généralement une cote unique pour chaque document. Cette cote permet d'identifier d'une manière unique chaque document. On va utiliser cette cote pour former le nom distinctif des entrées. Par exemple, les noms distinctifs des deux entrées de la Figure 6-5 sont :

```

cote=INF-2003,ou=bibliothèque,o=Université de Blida,c=DZ
cote=CAS-MK-6A,ou=bibliothèque,o=Université de Blida,c=DZ
  
```

Puisque l'accès à l'annuaire n'est pas direct chaque bibliothèque peut adapter sa propre méthode de nommage.

4.1.4 L'internationalisation

Dans le protocole LDAP V3, les chaînes de caractères sont codées au format UTF-8. Comme le nom l'indique, UTF-8 (UCS Transformation Format) est une transformation des caractères Unicode à une représentation octet qui

prend de un à huit octets par caractère. UTF-8 simplifie et normalise la manipulation des caractères des langues nationales [HEI 99].

Donc, il n'y a pas de problème avec la langue des notices.

4.1.5 L'accès à l'annuaire

Comme nous l'avons dit, l'accès à l'annuaire n'est pas direct, il est assuré par des objets services. C'est-à-dire le reste du système accède à l'annuaire par le biais d'un ensemble d'objets qui encapsulent les services du catalogue (Figure 6-6).

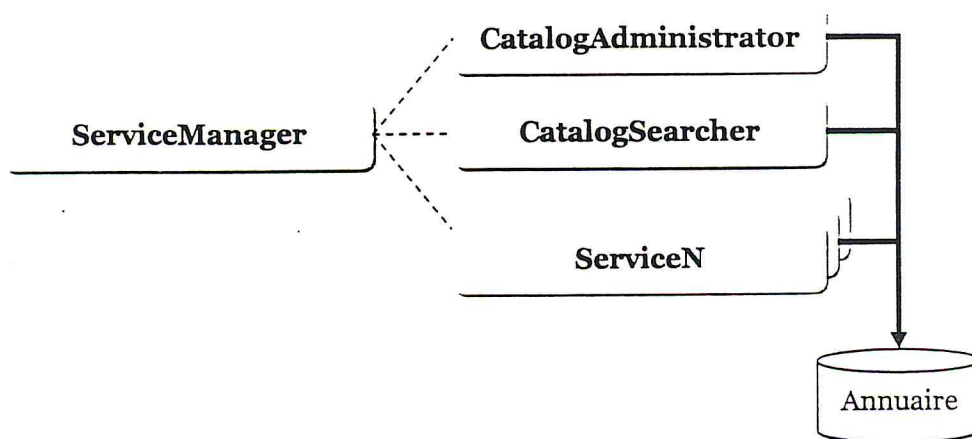


Figure 6-6 : Les objets services.

Ces objets services sont implémentés par des objets CORBA. Leur utilisation permet de rendre l'annuaire transparent aux applications et de fournir une interface de programmation orientée objet.

Aussi, l'utilisation de ces objets services facilite **l'intégration des catalogues existants** qui n'utilisent pas forcément LDAP. Il suffit d'implémenter ces services pour que ce soit transparent aux clients qui ne sauront pas s'il s'agit d'un annuaire ou d'une base de données ou simplement d'un fichier texte. De plus, si l'annuaire change et si de nouveaux services sont installés les anciennes applications fonctionneront toujours, tandis que les nouvelles applications pourront profiter de ces nouveaux services.

Par exemple, si nous avons un service de recherche qui contient une seule méthode `SearchByKeyword()` qui permet la recherche par mots clés. Nous pourrions rajouter de nouvelles méthodes telles que `SearchByTitle()` pour la recherche par titre sans avoir à modifier les anciennes applications sur le système. Les nouvelles applications pourront faire appel à ce nouveau service.

Chaque nouveau service doit supporter l'interface des versions précédentes. Nous pourrions également améliorer les fonctionnalités existantes.

En bref, cette approche permet de mieux gérer le catalogue. L'installation des nouvelles fonctionnalités, le changement de l'annuaire, ou autre mis à jour ne vont pas entraîner une modification dans le reste du système. Ainsi, elle permet d'intégrer facilement les catalogues existants.

Les services sont gérés par l'objet **ServiceManager** qui est responsable de créer les objets service. Techniquement parlant, c'est un fabriquant d'objets (object factory). Les clients doivent d'abord récupérer une référence sur le **ServiceManager** puis ils lui demandent une référence sur le service voulu. Si le service demandé existe, une référence est retournée au client, sinon le client est informé que cette bibliothèque ne fournit pas ce service pour le moment. Ceci permet aux bibliothèques d'implémenter que les services qui les intéressent. Le **ServiceManager** permet aussi d'énumérer les services fournis, ce qui va permettre aux clients de demander dynamiquement des services, de les explorer et de les utiliser en les invoquant dynamiquement.

Actuellement, le système fournit deux services, un service de recherche décrit par l'interface **CatalogSearcher** et un service d'administration décrit par l'interface **CatalogAdministrator**.

Un objet **CatalogSearcher** assure les opérations de recherche pour un **utilisateur anonyme**. Il contient des méthodes qui permettent la recherche par titre, nom, mot clé, etc. Une possibilité d'exécuter des requêtes LDAP directement est prévue pour des utilisateurs spécialisés tels que les bibliothécaires. Les résultats de recherche sont retournés aux clients sous format XML¹ (eXtended Mark-up Language).

Un objet **CatalogAdministrator** et un objet **CatalogSearcher** mais pour les utilisateurs authentifiés (ceux qui ont un nom et un mots de passe), notamment l'administrateur du catalogue. Un objet **CatalogAdministrator** assure les opérations de modifications (mis à jour, insertion, suppression, importation). Ces opérations sont exécutées suivant les droits accordés à l'utilisateur.

En plus des objets services, il y a les *analyseurs XML* (*parseurs XML*) qui permettent de convertir les résultats de recherche vers XML et de même pour les requêtes.

4.2 Conception de l'application (la logique métier)

La logique métier est implémentée par des **Enterprise JavaBeans**, elle contient un service OPAC (recherche), un service d'administration des

¹ Une introduction à XML est présenté dans l'annexe A (page 114).

catalogues, et un service qui permet de localiser l'ensemble des catalogues qui constituent le système.

Un seul EJB (Enterprise JavaBean) suffira pour exposer les opérations de recherche et d'administration (Figure 6-7).

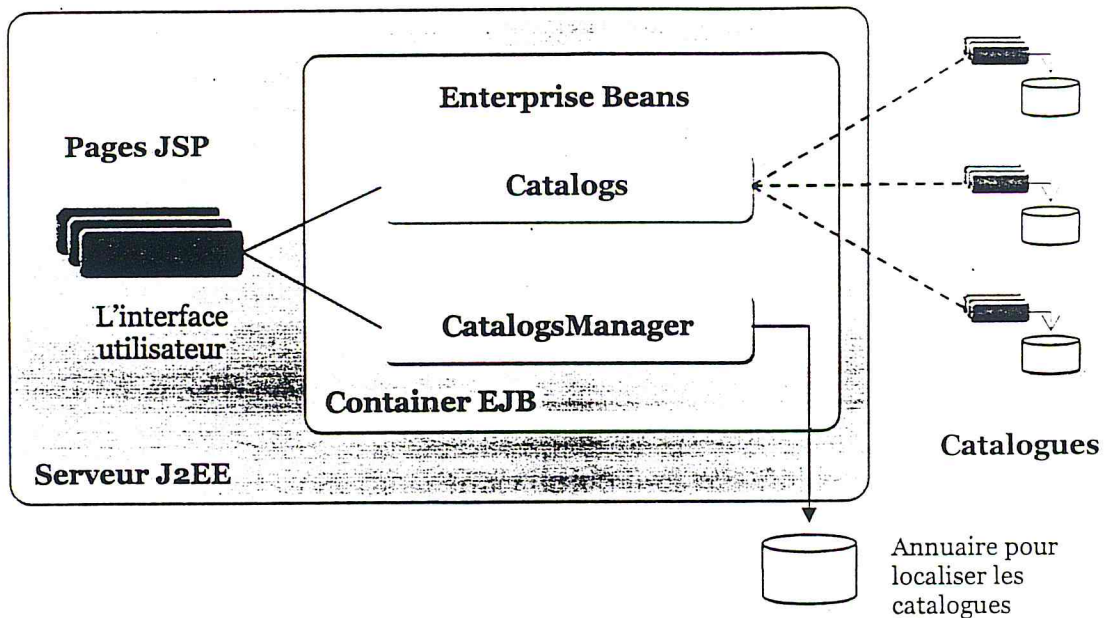


Figure 6-7 : Les composants de l'application.

L'EJB **Catalogs** expose des méthodes de recherche, de connexion à un catalogue, et d'administration d'un catalogue. L'utilisateur peut choisir les catalogues à interroger, ceci lui oblige à garder l'état du client pendant toute la session. De plus, l'EJB **Catalogs** ne représente pas un ensemble de données. L'EJB **Catalogs** sera donc un EJB *session avec état* (stateful session bean).

4.2.1 Comment l'EJB Catalogs fonctionne ?

Dès son création, il va récupérer une référence sur le **ServiceManager** de chaque catalogue. Puis il demande une référence sur un **CatalogSearcher** de chaque catalogue.

Une opération de recherche consiste à interroger chaque référence de **CatalogSearcher** puis les résultats sont regroupés dans un seul document XML.

Si l'utilisateur veut administrer un catalogue, la référence de **CatalogSearcher** est remplacée par une référence de **CatalogAdministrator** après l'authentification de l'utilisateur.

Il est clair que l'EJB **Catalogs** va seulement refléter les opérations de **CatalogSearcher** et celles de **CatalogAdministrator** mais il interroge plusieurs catalogues à la fois.

4.2.2 La gestion des catalogues

Informatiquement parlant, les catalogues sont des **ressources** et comme toute ressource nous avons besoin de quelques informations pour la localiser et l'utiliser. Le point d'accès d'un catalogue est un objet CORBA (le **ServiceManager**) et pour qu'on puisse s'y connecter on a besoin de l'adresse du serveur sur lequel il s'exécute plus d'autres informations telles que le numéro du port et son nom symbolique. Et sans oublier les informations concernant le catalogue lui-même telles que son nom et sa description.

Ces informations sont gérées par un EJB appelé **CatalogsManager**. Il est implémenté sous forme d'un EJB entity dont la persistance est gérée par le container (Container Managed Persistence). Et il est préférable d'utiliser un annuaire LDAP si le container le supporte.

La gestion des catalogues n'est pas publique, seul l'administrateur du système peut ajouter, modifier, ou supprimer des catalogues.

4.3 Conception de l'interface utilisateur

Nous constatons deux types de clients, des **clients lourds** qui sont ceux qui disposent des applications installées sur leurs postes pour utiliser le système, et des **clients légers** qui eux par contre ne disposent que d'un simple browser Web (Microsoft Internet Explorer ou Opera de Opera Software par exemple).

A chaque mise à jour, un client lourd doit installer la nouvelle version de l'application. Le client léger reste donc la solution idéale dès qu'elle est possible.

En ce qui concerne notre application, nous avons opté pour la solution **clients légers**. L'interface utilisateurs sera un site Web composé de plusieurs pages Jsp qui vont permettre de générer des pages HTML.

4.4 La sécurité des connexions

Pour les opérations de recherche, il est inutile de sécuriser les connexions car les informations qui circulent sont publiques. Par contre dans le cas des modifications, les communications doivent être sécurisées car nous ne voulons pas que quelqu'un puisse altérer les informations qui circulent sur le réseau.

La Figure 6-8 montre les différentes connexions qu'il faut sécuriser.

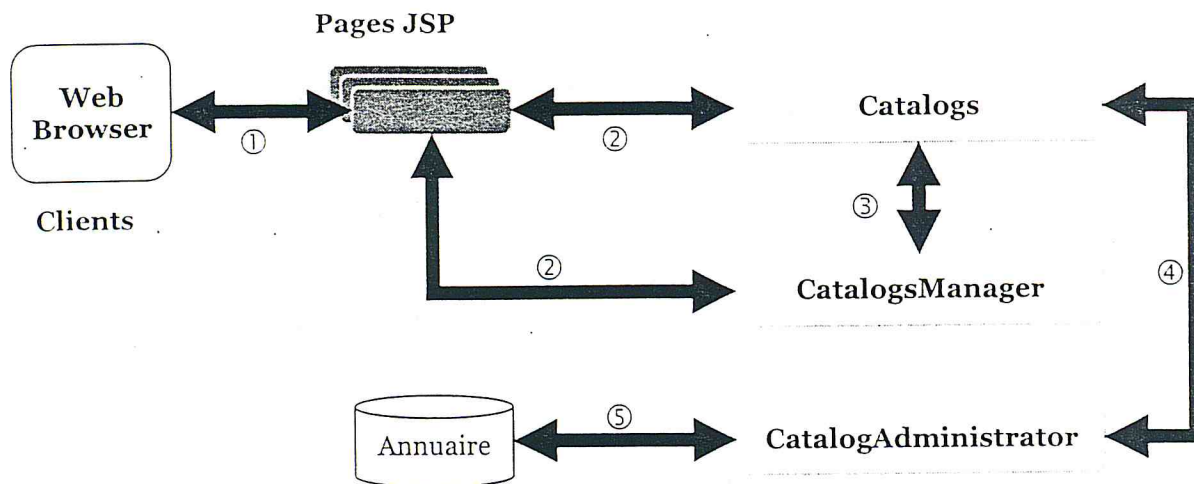


Figure 6-8 : Les différentes connexions.

① : Elle représente la connexion entre le client et le serveur Web. Pour les opérations de recherche, les pages sont transmises par le protocole HTTP et dès que le client se connecte à un catalogue tout le transfert se fera par le protocole HTTPS même ceux de recherche car il existe des notices non publiques.

② : Elle représente la communication entre le serveur Web et le serveur d'application. Dans le cas où l'utilisateur est connecté à un catalogue, la communication est assurée par la couche SSL¹.

③ : Ce cas ne se présente seulement quand les deux objets ne s'exécutent pas sur le même serveur. Nous utilisons alors le SSL.

④ : Elle représente la connexion entre le serveur d'application et un objet CORBA. La communication est assurée par l'ORB. Beaucoup d'ORBs utilisent SSL pour sécuriser la communication.

⑤ : Elle représente la connexion entre l'objet et le serveur LDAP. Plusieurs méthodes sont possibles mais nous avons choisi de continuer à utiliser SSL.

5. Mise en oeuvre

5.1 Environnement de développement

- OpenLDAP de OpenLDAP Foundation² : un serveur LDAP v3 doté de plusieurs outils.

¹ <http://www.netscape.com/eng/ssl3>

² <http://www.openldap.org>

- **LDAP Browser\Editor** version 2.8.2 de Jarek Gawor¹ : une application qui permet l'exploration d'un annuaire LDAP.
- **J2EE(TM) 1.4 Application Server Beta 2** de Sun Microsystems² : un serveur J2EE.
- **Java™ 2 SDK, Standard Edition, version 1.4.2** de Sun Microsystems³.
- **UltraEdit®-32 Professional Text/HEX Editor** version 9.20b de IDM Computer Solutions, Inc⁴ : un éditeur de texte très pratique.
- **PC avec Windows XP Professional Edition, AMD Duron à 600Mhz, et 256Mo de RAM.**
- **Microsoft Visual Studio .NET** : le meilleur outil de développement sur la planète. Nous l'avons utilisé pour apprendre, développer, et déboguer la partie XML et XSLT.

5.2 Mise en oeuvre du catalogue

5.2.1 L'annuaire

Nous avons dit que l'annuaire doit permettre de stocker tous les types de documents et qu'à chaque type de document est associé une classe d'objet. Il faut donc définir un attribut pour chaque élément de la description bibliographique.

La définition des attributs est simple. Par exemple, l'attribut *Cote* est de même type que l'attribut *name* standard, donc pour définir l'attribut *Cote* il suffit d'hériter de l'attribut *name* comme le montre le code qui suit écrit en syntaxe de OpenLDAP:

```
attributetype ( 1.1.2.1.7
    NAME 'Cote'
    DESC 'Cote'
    SUP name
    SINGLE-VALUE
)
```

"1.1.2.1.7" est l'identificateur de l'attribut (OID), "NAME 'Cote'" donne le nom "Cote" à l'attribut, "DESC 'Cote'" donne une description à l'attribut, "SUP name" permet à l'attribut *Cote* d'hériter toutes les caractéristiques de l'attribut *name*, et "SINGLE-VALUE" spécifie que chaque entrée ne peut posséder qu'un seul attribut *Cote*.

¹ <http://www.iit.edu/~gawojar/ldap>

² <http://java.sun.com/j2ee>

³ <http://java.sun.com/j2se>

⁴ <http://www.ultraedit.com>

Le Listing 6-1 montre la définition de quelques attributs dans notre application.

Listing 6-1 schema/catalog.schema

```
.....
attributetype ( 1.1.2.1.10
    NAME 'DocTitle'
    DESC 'Titre'
    SUP name
    SINGLE-VALUE
)
attributetype ( 1.1.2.1.11
    NAME 'OtherAuthors'
    DESC 'Autres Auteurs'
    SUP name
)
attributetype ( 1.1.2.1.12
    NAME 'OtherEditors'
    DESC 'Autres éditeurs'
    SUP name
)
.....
```



Nous avons dit également que les descriptions bibliographiques sont stockées d'une manière plate dans le sous arbre ou=library,... et pour distinguer entre les notices bibliographiques et les autres objets (comme les personnes) lors de la recherche, toute classe qui représente un type de document doit hériter de la classe abstraite libraryMaterial qui est définie comme suit :

```
objectclass (1.1.2.2.1 NAME 'libraryMaterial' SUP top ABSTRACT
    MUST ( Cote $ DocTitle )
)
```

Comme vous avez remarqué, chaque document doit obligatoirement comporter une cote et un titre.

Le Listing 6-2 montre la définition de quelques classes d'objet dans notre application.

Listing 6-2 schema/catalog.schema

```
.....
objectclass (1.1.2.2.2 NAME 'Monograph'
    SUP libraryMaterial STRUCTURAL
    DESC 'Monographie'
    MUST ( Author )
    MAY ( Editor $ Edition $ MainTitle $ OtherAuthors $
        OtherEditors $ Publisher $ Editor $
        EditionInformation $ Description $ ISBN $
        Keyword $ Abstract)
)
```

```
objectclass (1.1.2.2.3 NAME 'VideoCassette'  
    SUP libraryMaterial STRUCTURAL  
    DESC 'Cassette vidéo'  
    MAY ( Description S Keyword )  
    .....  
)
```

La classe Monograph permet de définir une monographie. Elle hérite comme toutes les autres de la classe abstraite libraryMaterial, donc elle doit contenir obligatoirement l'attribut Cote et l'attribut DocTitle. De plus, la définition spécifie que l'attribut Author est lui aussi obligatoire. Par contre, les attributs spécifiés par le mot clé MAY tels que Editor, Edition,... sont optionnels. En bref, une notice qui décrit une monographie doit contenir une cote, un titre, et la liste des auteurs, elle peut contenir d'autres informations comme le nom de l'éditeur, la liste des mots clés, un résumé, etc. mais ceux-ci ne sont pas obligatoires.

Ces classes sont écrites et conseillées par un spécialiste du catalogage. Mais le choix des formats revient à l'ensemble des bibliothèques qui participent à cette union des catalogues. Ce sont elles qui savent lesquels leurs conviennent le mieux. Malheureusement au cours de notre travail, nous n'avons pas eu l'occasion de rencontrer des bibliothécaires spécialistes, donc nous nous sommes contenté de quelques éléments communs.

Notre système explore le schéma d'une manière dynamique pour accéder à l'annuaire, ce qui permet une grande flexibilité et facilite le travail des experts qui seront amenés à utiliser ce système.

5.2.2 La description des attributs

Selon le schéma de l'annuaire, le titre est stocké dans un attribut appelé DocTitle, le résumé dans un attribut appelé Abstract, une monographie est stockée dans une classe appelée Monograph, etc. Lors de l'affichage d'une notice, ces noms d'attributs ne sont pas significatifs pour l'utilisateur final. De plus, le schéma est dynamique alors nous pouvons toujours insérer de nouveaux attributs et de nouvelles classes (d'autres types de documents seront inventés). Donc, il faut associer aux noms internes (utilisés dans le schéma) les noms à afficher aux utilisateurs. Par exemple, l'application affiche Titre au lieu de DocTitle. Ainsi, le catalogue sera auto-descriptif.

La solution que nous proposons est d'enregistrer également les descriptions des attributs dans l'annuaire. Ceci a été possible en définissant la classe elementDescription qui stocke les noms qui seront affichés à l'utilisateur. La classe elementDescription est définie comme suit :


```

objectclass (1.1.2.2.1.0 NAME 'elementDescription'
    SUP top STRUCTURAL
    MUST { cn $ ln $ displayName }
)

```

Elle contient trois attributs obligatoires, l'attribut `cn` (CommonName) qui contient l'attribut à décrire (Monograph, DocTitle, Abstract,...), l'attribut `ln` (langue) qui spécifie la langue utilisée pour décrire l'attribut (AR, FR, EN,...), et l'attribut `displayName` qui contient le nom qui s'affichera à l'utilisateur.

Les entrées appartenantes à cette classe d'objet sont organisées dans le DIT de la même façon que les notices (Figure 6-9).

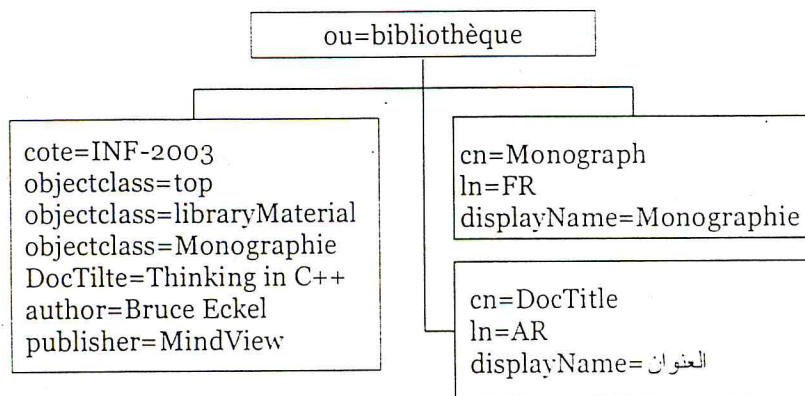


Figure 6-9 : Description des attributs.

Pour les noms distinctifs, nous aurons par exemple :

```

cn=Monograph+ln=FR,ou=bibliothèque,...
cn=DocTitle+ln=AR,ou=bibliothèque,...

```

5.2.3 Les objets services

Les objets services sont ceux qui fournissent ou encapsulent les services d'un catalogue. Dans notre application il y a deux objets, **CatalogSearcher** et **CatalogAdministrator** en plus de **ServiceManager** qui gère les différents objets.

Le Listing 6-3 représente un extrait de l'interface IDL de ces trois objets.

Listing 6-3 catalog/catalog.idl

```

.....
module catalog
{
    .....
    exception NotImplemented {};
}

```

```

.....
interface CatalogSearcher
{
    XmlDocument SearchByKeyword(in wstring keywords,
                                in string language)raises (InternalError);
    .....
    XmlDocument GetItemDetail(in ItemID itemID,
                               in string language)raises (InternalError);
    .....
    XmlDocument LdapSearch(in LdapQuery ldapQuery,
                           in string language)
        raises (NotImplemented, InternalError);
    .....
};
interface CatalogAdministrator : CatalogSearcher
{
    .....
    void ModifyItem(in ItemID itemID,
                    in XmlDocument item)
        raises (InternalError);
    void AddItem(in XmlDocument item)
        raises (InternalError);
    void RemoveItem(in ItemID itemID)
        raises (InternalError);
    .....
    XmlDocument GetDocumentTypes(in string language);
    XmlDocument GetDocumentAttributes(
        in wstring documentTypeID,
        in string language);
    void Close();
};
interface ServiceManager
{
    CatalogSearcher GetCatalogSearcher()
        raises (InternalError, ServiceNotProvided);
    CatalogAdministrator GetCatalogAdministrator(
        in wstring userDN,
        in wstring userPassword)
        raises (InternalError, ServiceNotProvided);
    void ModifyItem(in ItemID itemID,
                    in XmlDocument item)
        raises (InternalError);
    any GetCatalogService(in wstring ServiceName)
        raises (ServiceNotProvided);
    void AddItem(in XmlDocument item)
        raises (InternalError);
    readonly attribute wstring CatalogDescription;
};

```

L'objet **ServiceManager** contient la méthode **GetCatalogSearcher()** qui permet de retourner aux clients une référence sur un **CatalogSearcher** et la méthode **GetCatalogAdministrator()** qui permet de retourner une référence sur un **CatalogAdministrator**.

```

CatalogSearcher GetCatalogSearcher()
    raises (InternalError, ServiceNotProvided);
CatalogAdministrator GetCatalogAdministrator(
    in wstring userDN,
    in wstring userPassword)
    raises (InternalError, ServiceNotProvided);

```


Les clients anonymes partagent le même **CatalogSearcher** qui sera créé par le **ServiceManager** pendant l'initialisation du serveur.

Pourquoi tous les clients partagent-ils le même **CatalogSearcher** ? L'objet **CatalogSearcher** ne stocke pas les informations relatives à un client, le partager par tous les clients permet de minimiser le nombre de connexions à l'annuaire. Si chaque client possède son propre **CatalogSearcher**, nous pourrions éviter de spécifier la langue du résultat dans chaque méthode. Dans ce cas, il faut détruire l'objet à la fin du travail, mais il arrive souvent que la connexion entre le client et le serveur soit interrompue, et notre objet reste dans la mémoire.

Les objets **CatalogAdministrators** ne sont créés que pour les utilisateurs enregistrés dans l'annuaire. Bien que l'objet **CatalogAdministrator** présente les méthodes d'administration, cela ne signifie pas que l'utilisateur arrive toujours à modifier car les droits d'accès sont assurés par le serveur LDAP et non par le **CatalogAdministrator**.

5.2.4 Format de données

Le format que nous avons choisi pour transférer nos données est le XML. Avant de nous décider, nous avons cherché une structure de données qui permet de stocker des données que nous ne connaissions ni leur structure ni leur taille puisque le schéma est exploré d'une manière dynamique. Ces caractéristiques se trouvent bien dans XML. De plus, XML est lisible et ne nécessite aucune connaissance théorique.

Le Listing 6-4 présente le format général d'un résultat de recherche (utilisé aussi pour le détail d'un document).

Listing 6-4 : généré par la méthode `catalog.CatalogSearcher.LdapSearch()`

```
<?xml version="1.0" encoding="utf-8" ?>
<search-result>
  <Document document-type="Monograph" id="Cote=INF-2003">
    <DocTitle>Thinking In Java</DocTitle>
    <Cote>INF-2003</Cote>
    <Author>Bruce Eckel</Author>
    <Keyword-set>
      <Keyword>Java</Keyword>
      <Keyword>OOP</Keyword>
    </Keyword-set>
    <description>...</description>
    <Edition>...</Edition>
    <Abstract>...</Abstract>
  </Document>
  <Document document-type="Serial" id="Cote=INF-1999">
    <DocTitle>World Business Report</DocTitle>
    <Cote>...</Cote>
```



```

...
</Document>
...
<description ln="fr">
  <document-type>
    <Serial>Série</Serial>
    <Monograph>Monographie</Monograph>
  </document-type>
  <document-element>
    <Cote>Cote</Cote>
    <Keyword>Mot-clé</Keyword>
    <Abstract>Résumé</Abstract>
    <DocTitle>Titre</DocTitle>
  </document-element>
</description>
</search-result>

```

Comme vous avez remarqué le résultat est codé au format UTF-8. Chaque document est identifié par son nom distinctif relatif (RDN). Tous les attributs qui décrivent les documents ainsi que les types de documents utilisés sont décrits dans le résultat (par une langue choisie par le client) car le schéma est dynamique et nous ne pouvons pas connaître l'affichage final qu'après exécution. Les clients utilisent ces descriptions pour afficher les informations aux utilisateurs.

Chaque élément Document contient la description d'un document dont les valeurs des attributs sont stockées dans des éléments qui portent les mêmes noms que les attributs et si par exemple l'attribut *x* est multivalué alors ses valeurs sont regroupées dans un élément *x-set*.

Avant qu'un gestionnaire de catalogue puisse insérer des documents, il doit au préalable restituer la liste des types de documents détenus par le catalogue en appelant la méthode `CatalogAdministrator::GetDocumentTypes()`. Le Listing 6-5 présente le format du résultat d'un tel appel.

Listing 6-5 généré par `catalog.CatalogAdministrator.GetDocumentTypes()`

```

<?xml version="1.0" encoding="utf-8" ?>
<catalog-document ln="fr">
  <document-type id="Monograph">Monographie
</document-type>
  <document-type id="VideoCassette">Cassette vidéo
</document-type>
  <document-type id="Serial">Série</document-type>
  .....
</catalog-document>

```

Maintenant que le gestionnaire connaît les types de documents supportés, il sait qu'il peut insérer par exemple une monographie. Il lui faut maintenant les attributs qui composent une monographie, lesquels sont obligatoires et lesquels peuvent avoir plusieurs valeurs. Ceci est le rôle de la méthode `CatalogAdministrator::GetDocumentAttributes()`. Le Listing 6-6 présente le format du résultat d'un appel à cette méthode.

Listing 6-6 généré par `catalog.CatalogAdministrator.GetDocumentAttributes()`

```
<?xml version="1.0" encoding="utf-8" ?>
<document-element id="Monograph" name="دراسة" ln="ar">
  <element id="Cote" name="الرقم" single-value="true"
    mandatory="true" />
  <element id="DocTitle" name="العنوان" single-value="true"
    mandatory="true" />
  <element id="Author" name="المؤلف" mandatory="true" />
  <element id="Editor" name="الناشر" single-value="true" />
  <element id="Edition" name="الطبعة" />
  .....
</document-element>
```

Comme XML est le langage choisi comme format d'échange de données, le client est également tenu de l'utiliser. Il peut alors insérer un document en appelant la méthode `CatalogAdministrator::AddItem()` comme c'est précisé dans le Listing 6-7.

Listing 6-7 pour appeler `catalog.CatalogAdministrator.AddItem()`

```
<?xml version="1.0" encoding="utf-8" ?>
<add-item>
  <item>
    <Type>Monograph</Type>
    <Cote>INF-3600</Cote>
    <Author>Michel Lai</Author>
    <DocTitle>UML
      La notation unifiée de modelisation objet
      De Java aux EJB
    </DocTitle>
    <Edition>2e édition, 2000</Edition>
    <Editor>DUNOD</Editor>
    .....
  </item>
  <item>...</item>
  .....
</add-item>
```

Il serait également possible d'insérer plusieurs documents à la fois mais nous n'avons pas prévu cette fonctionnalité dans la version actuelle de notre système.

Le format de modification utilisé pour appeler la méthode `CatalogAdministrator::ModifyItem()` est presque identique à celui de l'insertion, sauf que l'élément `<Type>` ne doit pas figurer. Ce format est décrit dans le Listing 6-8.

Listing 6-8 pour appeler `catalog.CatalogAdministrator.ModifyItem()`

```
<?xml version="1.0" encoding="utf-8" ?>
<modify-item>
  <item>
    <Cote>INF-3600</Cote>
    <Author>Michel Lai</Author>
    <DocTitre>UML
      La notation unifiée de modelisation objet
      De Java aux EJB
    </DocTitle>
    <Edition>2eme Edition, 2000</Edition>
    <Editor>DUNOD</Editor>
    .....
  </item>
  <item>...</item>
  .....
</modify-item>
```

Lors d'une modification les attributs sont complètement remplacés par les nouvelles valeurs, c'est-à-dire les attributs non mentionnés ne seront pas supprimés. Il est fort possible que cette technique de modification soit améliorée dans le futur (l'utilisateur pourra spécifier les attributs à modifier, ceux à supprimer, et ceux à laisser).

5.2.5 Modèle fonctionnel

Le pseudo code du Listing 6-9 décrit l'utilisation de ces objets.

Listing 6-9 Exemple d'utilisation des objets services

```
ServiceManager srvMan;
...
CatalogAdministrator catAdm;
catAdm = srvMan.GetCatalogAdministrator("cn=Manager,...",
    "+++");

srchRes = catAdm.SearchByKeyword("C++", "fr");
srchRes = catAdm.SearchByAuthor("Bruce*", "ar");
srchRes = catAdm.SearchBySubject("Informatique", "en");
srchRes = catAdm.SearchByTitle("*java*", "fr");
srchRes = catAdm.SearchByNumber("INF*", "fr");

srchRes = catAdm.GetItemDetail("INF-3200", "fr");

String newItem;
newItem = "<?xml version=\"1.0\" encoding=\"utf-8\" ?> "
```



```

+ "<add-item>"
+ "  <item>"
+ "    <Type>Monograph</Type>"
+ "    <Cote>INF-3603</Cote>"
+ "    <Author>Michel Lai</Author>"
+ "    <DocTitle>UML"
+ "      La notation unifiée de modélisation objet"
+ "      De Java aux EJB"
+ "    </DocTitle>"
+ "  </item>"
+ "</add-item>"

try
{
  // Insertion
  catAdm.AddItem(newItem);
  // suppression
  catAdm.RemoveItem("INF-3600");
}
catch (InternalError e)
{
  // erreur
}

```

5.3 Mise en œuvre de l'application (les EJBs)

Les services métiers de l'application sont assurés par l'objet **Catalogs** qui encapsule toutes les opérations relatives aux catalogues, et par l'objet **CatalogsManager** qui gère l'ensemble des catalogues du système.

La Figure 6-10 présente l'interface **Remote** de l'EJB **Catalogs**.

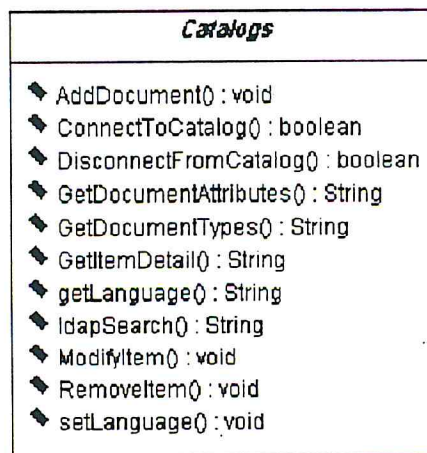


Figure 6-10 : L'interface Remote de l'EJB Catalogs.

Vous remarquerez que les mêmes méthodes définies dans le catalogue existent aussi dans l'EJB. Chacune de ces méthodes appelle une méthode d'un ou plusieurs catalogues.

5.3.1 Le cycle de vie de l'EJB Catalogs

Lors de sa naissance (sa création), un **EJB Catalogs** récupère la liste des catalogues disponibles puis pour chaque catalogue il va récupérer une référence sur son **CatalogSearcher**. L'objet est donc prêt à faire les opérations de recherche.

La recherche est assurée par la méthode `ldapSearch` qui va déclencher la recherche dans tous catalogues simultanément. Le résultat est envoyé au client une fois la recherche terminée dans tous les catalogues tout en respectant un délai maximal (time out).

Si l'objet était créé suite à l'appel d'un client privilégié c'est à dire d'administrer un catalogue, il peut alors récupérer une instance sur l'objet **CatalogAdministrator**. Il peut même en récupérer plusieurs.

Avec les **CatalogAdministrators**, notre EJB a atteint la maturité et il peut assurer l'échange des notices bibliographiques ainsi que l'insertion de nouvelles notices, leur suppression, et leur modification.

5.3.2 Le déroulement de la recherche

Lorsqu'un client appelle la méthode `ldapSearch` de l'objet **Catalogs**, celle-ci va lancer la recherche dans tous les catalogues simultanément (il va créer une thread pour chaque catalogue). Donc il va récupérer un résultat de chaque catalogue, ces résultats sont de la forme décrite dans le Listing 6-4 (page 88). L'objet regroupe tous les résultats reçus dans un même document XML qui sera retourné au client. Le format de ce document est décrit dans le Listing 6-10.

Listing 6-10 : généré par la méthode `catalogs.Catalogs.LdapSearch()`

```
<?xml version="1.0" encoding="UTF-8" ?>
<search-result>
  <catalog id="cat_001" name="Bibliothèque de Blida"
    can-show-detail="true" can-update="true">
    <Document document-type="Monograph" id="INF-2003">
      <DocTitle>La planète Mars</DocTitle>
      ...
    </Document>
    <Document document-type="Monograph" id="INF-1999">
      <DocTitle>Mars, un rêve de l'humanité
      </DocTitle>
      ...
    </Document>
    ...
    <description ln="fr">
      ....
    </description>
  </catalog>
  <catalog id="cat_002" name="مكتبة المستقبل">
```

```

        can-show-detail="true" can-import="true">
        <Document document-type="VideoCassette"
            id="CV/S23-197">
            <docTitle>جولة في المربع</DocTitle>
            ...
        </Document>
        <description ln="fr">
            ...
        </description>
    </catalog>
    <catalog id="cat_005" ... > ... </catalog>
</search-result>

```

5.3.3 Le catalogue d'Oran « le bruit »

Un jeune étudiant de l'université de Blida a lancé une recherche sur quelque chose qui l'intéresse. Il a obtenu un ensemble de notices bibliographiques de différents catalogues. Parmi ces résultats, il y a des documents localisés dans une bibliothèque d'un vieux quartier d'Oran. Pour notre étudiant qui ne peut pas se déplacer à Oran, il va automatiquement négliger les résultats de ce catalogue. Ceci n'est pas seulement le problème des gens qui habitent Blida, c'est ce qu'on appelle le **bruit**.

Donc l'idéal serait de fournir aux utilisateurs la possibilité de spécifier un rayon de recherche (50 Km, par exemple). Ou simplement de leur donner le droit de spécifier les catalogues qui participeront à la recherche et ceux qui n'y participeraient pas. (L'implémentation actuelle ne donne pas cette possibilité).

5.3.4 Le CatalogsManager

Le **CatalogsManager** représente les données nécessaires pour localiser un catalogue. Ces données sont le nom du catalogue, son adresse sur le réseau, et le numéro du port sur lequel l'ORB tourne. Plus d'autres informations utiles telles que la description du catalogue.

Nous avons implémenté cet objet sous forme d'EJB **entity** avec une persistance gérée par le container. Le Listing 6-11 décrit l'interface **CatalogsManager**.

 catalogs/CatalogsManager.java

```

package catalogs;

import javax.ejb.*;
import java.util.*;

public interface CatalogsManager extends EJBLocalObject
{

```



```
public String getCatalogID();  
public void setCatalogName(String catalogName);  
public String getCatalogName();  
public void setCatalogDescription(  
    String catalogDescription);  
public String getCatalogDescription();  
public void setHostAddr(String hostAddr);  
public String getHostAddr();  
public void setOrbPort(String orbPort);  
public String getOrbPort();  
}
```

A ce stade, nous venons de décrire le cœur de l'application. Elle réalise la recherche et l'administration des catalogues. Mais pour un utilisateur final, une interface claire et ergonomique est une valeur ajoutée pour notre travail. Nous décrivons dans ce qui suit les différentes interfaces conçues pour le système.

5.4 L'interface utilisateur

L'application est destinée principalement au grand public, ce qui ajoute d'autres challenges. Premièrement l'application doit être largement disponible, l'Internet fournira le support pour sa diffusion. Deuxièmement, l'interface doit être facile à utiliser (du genre Google). De plus, l'interface doit présenter toute la puissance de l'application.

L'interface sera présentée sous forme de pages HTML. Le fait d'avoir utiliser XML facilitera l'accès à l'application par un téléphone cellulaire ou un PDA (Personal Digital Assistant) et ceci à travers le langage adéquat c'est à dire WML (Wireless Markup Language).

L'interface se divise en trois parties. Une partie publique qui permet de rechercher les notices bibliographiques (la partie OPAC), une partie réservée aux administrateurs des catalogues qui permet d'ajouter, de modifier, de supprimer, et d'échanger des notices bibliographiques, et une partie qui permet d'administrer l'application elle-même.

5.4.1 L'interface OPAC

Cette interface est très simple car elle ressemble aux moteurs de recherche répandus sur le net. L'utilisateur entre sa requête puis lance la recherche. Il obtient un résultat composé d'un ensemble de notices bibliographiques dont il peut afficher le détail.

La Figure 6-11 présente un exemple d'une simple page de recherche qui permet de rechercher des documents soit par des mots-clés, soit par leurs cotes, ou soit en utilisant directement une requête LDAP.

Rechercher - Microsoft Internet Explorer

Fichier Edit View Favorites Tools Help

Back Search Favorites Media Go

http://localhost:1024/catalogs/search.jsp

Requête LDAP Submit

Mot-clé sémiologie Submit

Cote Submit

Accueil | Rechercher | Résultats précédents | Connecter à un catalogue | Gestion des catalogues | Fermer session

Done Local intranet

Figure 6-11 : Exemple d'une page de recherche

Puisque la recherche est basée sur des requêtes LDAP, on peut fournir plusieurs critères de recherches qui seront combinés pour obtenir des résultats très fins.

Le résultat de la recherche qui est un simple document XML sera transformé par un fichier XSL suivant le client (HTML, WML) avant d'être affiché au client. L'utilisation des transformations XSL simplifie la génération de l'interface, et surtout en cas de modification. De plus, ces transformations permettent de générer facilement des interfaces complexes.

La Figure 6-12 montre le résultat de notre recherche pour le mot clé « sémiologie » sous format HTML.

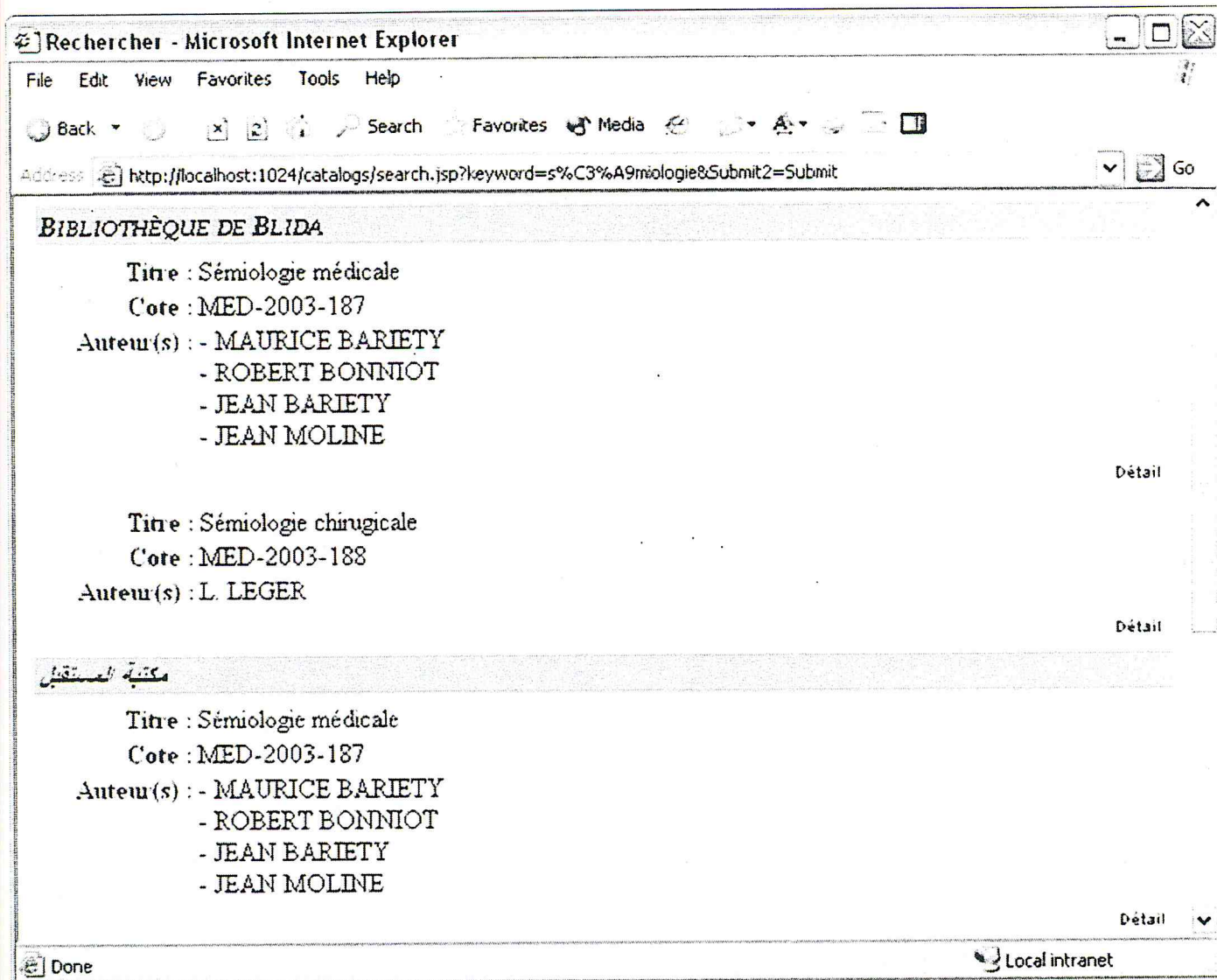


Figure 6-12 : Exemple d'un résultat de recherche

Les résultats sont regroupés par catalogue. Les notices sont affichées en mode réduit (le titre, la cote, et les auteurs). Au-dessous de chaque notice, nous trouvons une barre qui contient les opérations que l'utilisateur peut effectuer sur la notice.

Pour les utilisateurs anonymes, ils peuvent seulement afficher le détail de la notice. Par exemple la Figure 6-13 affiche le détail de la première notice.

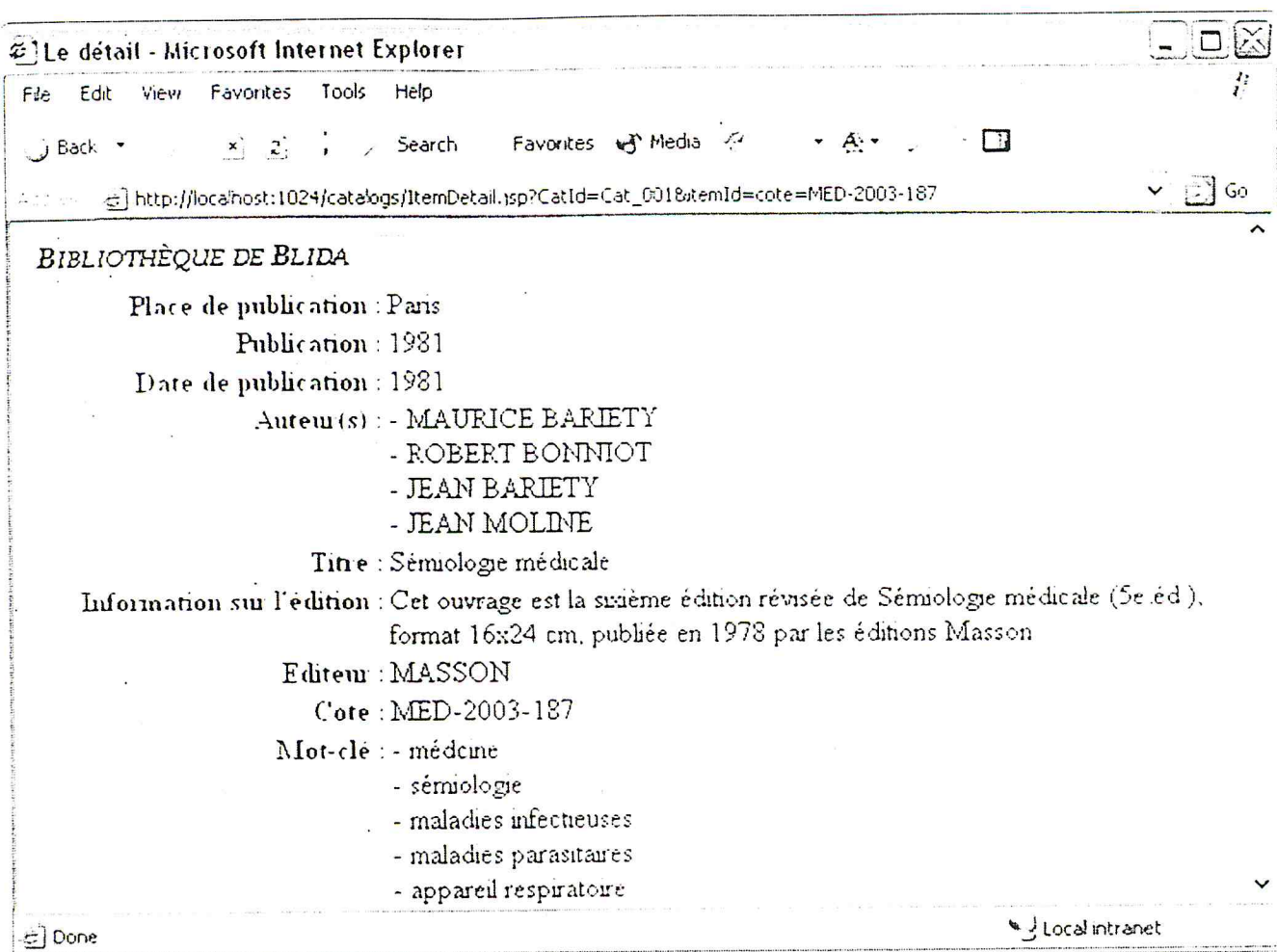


Figure 6-13 : Le détail d'un document.

En mode détail, tous les éléments qui décrivent la notice seront affichés.

Maintenant que notre utilisateur connaît bien l'emplacement du document plus quelques informations fournies dans la notice, il peut aller à la bibliothèque pour récupérer le document.

5.4.2 L'interface Administration des catalogues

Avec cette interface, les administrateurs peuvent administrer leurs catalogues même à distance. L'administration d'un catalogue inclut l'ajout de nouvelles notices et la modification ou la suppression des notices existantes. Mais le but principal c'est l'importation des notices à partir des autres catalogues.

Avant de commencer l'administration, l'utilisateur doit se connecter à un catalogue, c'est-à-dire ouvrir une session dans ce catalogue en passant par l'opération d'authentification (pour récupérer une instance sur un

CatalogAdministrator). La Figure 6-14 montre une opération d'authentification.

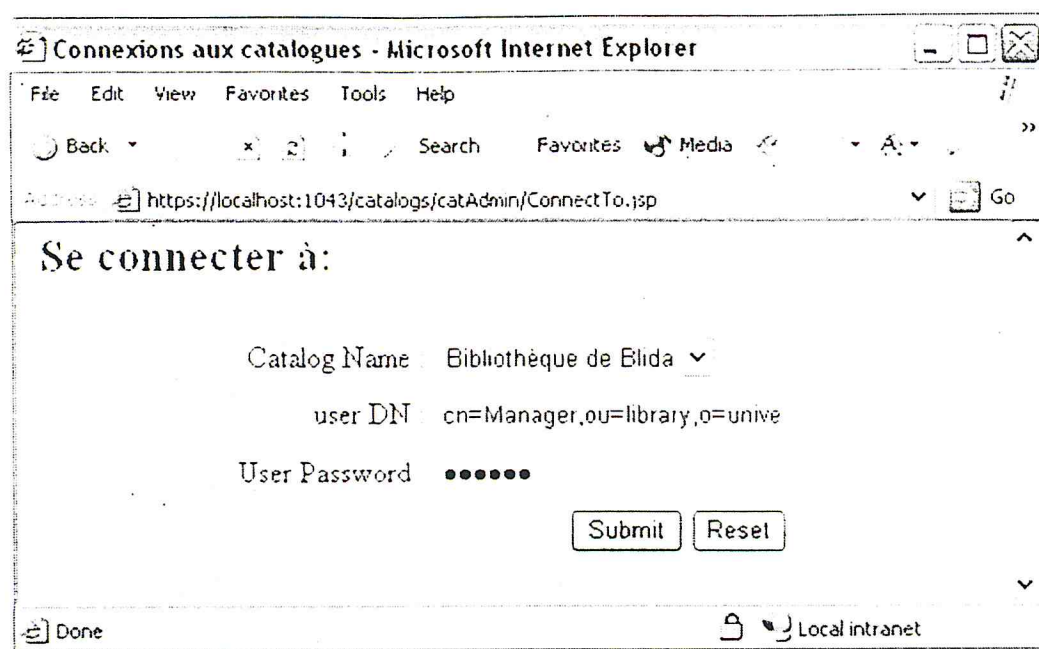


Figure 6-14 : Connexion à un catalogue.

L'utilisateur doit fournir son nom distinctif (DN) et son mot de passe afin qu'il puisse se connecter à un catalogue pour l'administrer. L'utilisateur peut bien entendu administrer plusieurs catalogues.

La communication ici est complètement sécurisée car le transfert de données passe en mode SSL en utilisant le protocole HTTPs. Ce mode est maintenu pendant toutes les opérations effectuées par un administrateur.

La Figure 6-15 montre le résultat obtenu après la recherche pour le mot clé « maladies infectieuses » par l'administrateur du catalogue de la bibliothèque de Blida.

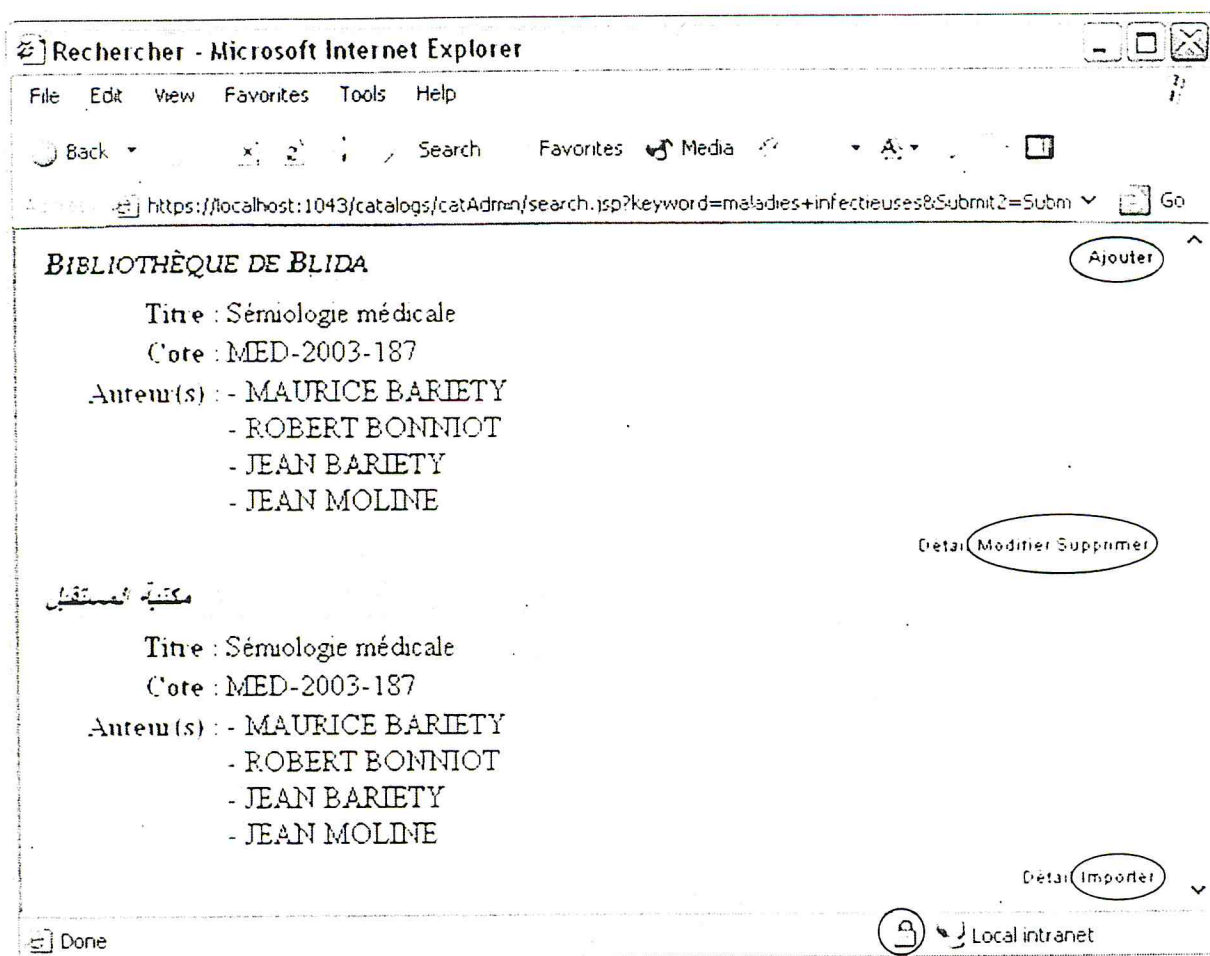


Figure 6-15 : Résultat de la recherche

Ce qui diffère d'un résultat de recherche ordinaire c'est que nous sommes en mode sécurisé et que nous pouvons effectuer plusieurs opérations sur les notices. Pour les catalogues administrés (« Bibliothèque de Blida » dans notre cas), nous pouvons modifier ou supprimer la notice et pour les catalogues non administrés nous pouvons importer leurs notices.

En plus, l'administration peut effectuer quelques opérations sur le catalogue. Si vous regardez à droite du premier catalogue, vous trouvez l'opération « Ajouter » qui permet d'ajouter une nouvelle notice dans le catalogue.

Avant d'ajouter une notice, l'utilisateur doit d'abord vérifier si elle existe déjà dans un autre catalogue (le catalogue de l'éditeur du document par exemple). Si la notice existe déjà, il aura qu'à l'importer (en cliquant simplement sur le lien « importer » relatif à cette notice). L'importation ne s'effectue pas d'un seul coup, la notice est affichée à l'utilisateur pour qu'il puisse apporter ses modifications (ne sera ce que pour la cote).

Si elle n'existe pas, l'utilisateur va ajouter la notice. Il doit d'abord spécifier le type du document qu'elle décrit (Figure 6-16). Il aura donc à choisir parmi les types disponibles dans son catalogue (pour les nouveaux types, il faut modifier le schéma de l'annuaire).

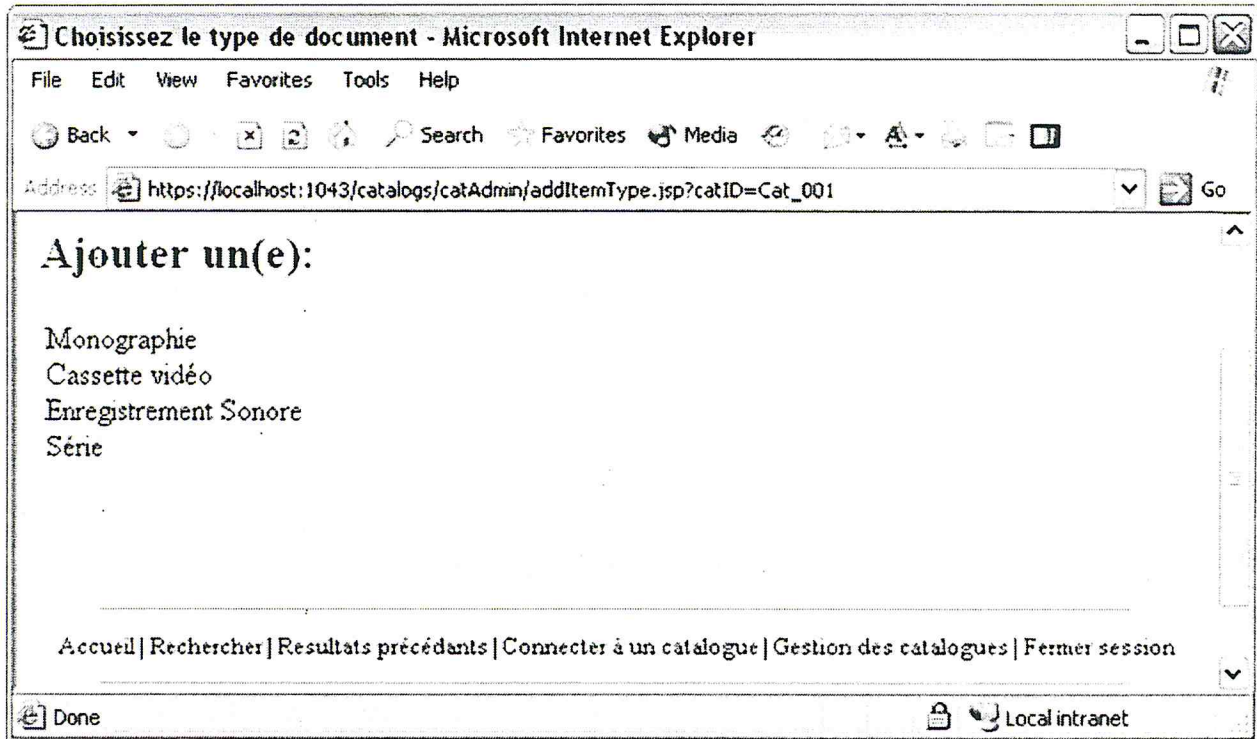


Figure 6-16 : Spécification du type de document à ajouter

Une fois le type est spécifié, l'utilisateur peut commencer à saisir sa notice dans un formulaire qui permet d'entrer tous les éléments de la notice (Figure 6-17).

Ajouter un document - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back * 2 Search Favorites Media Go

https://localhost:1043/catalogs/catAdmin/AddItem.jsp?catId=Cat_001&docTypeId=Mc

Insérer un nouveau document dans le catalogue Bibliothèque de Blida

Cote MED-2003-187

Titre Sémiologie médicale

Auteur(s) MAURICE BARIETY
ROBERT BONNIOT
JEAN BARIETY

Editeur MASSON

Edition 6e édition révisée

Local intranet

Figure 6-17 : Insertion d'une nouvelle notice.

Le formulaire est constitué d'un ensemble de zones de texte qui permettent de saisir les informations. Nous avons utilisé les zones de texte mono lignes pour entrer les éléments de la notice qui ne peuvent avoir qu'une seule valeur. Par exemple, la cote, le titre, etc. Par contre pour les éléments qui peuvent avoir plusieurs valeurs comme les auteurs par exemple, nous avons utilisé des zones de texte multi lignes, chaque valeur est entrée dans une ligne. Le formulaire contient aussi des petites étoiles devant les éléments obligatoires.

Avant d'envoyer le formulaire, un petit script s'assure que tous les éléments obligatoires ne sont pas vides. Si l'un d'entre eux est vide, un message de rappel ou d'erreur est affiché à l'utilisateur.

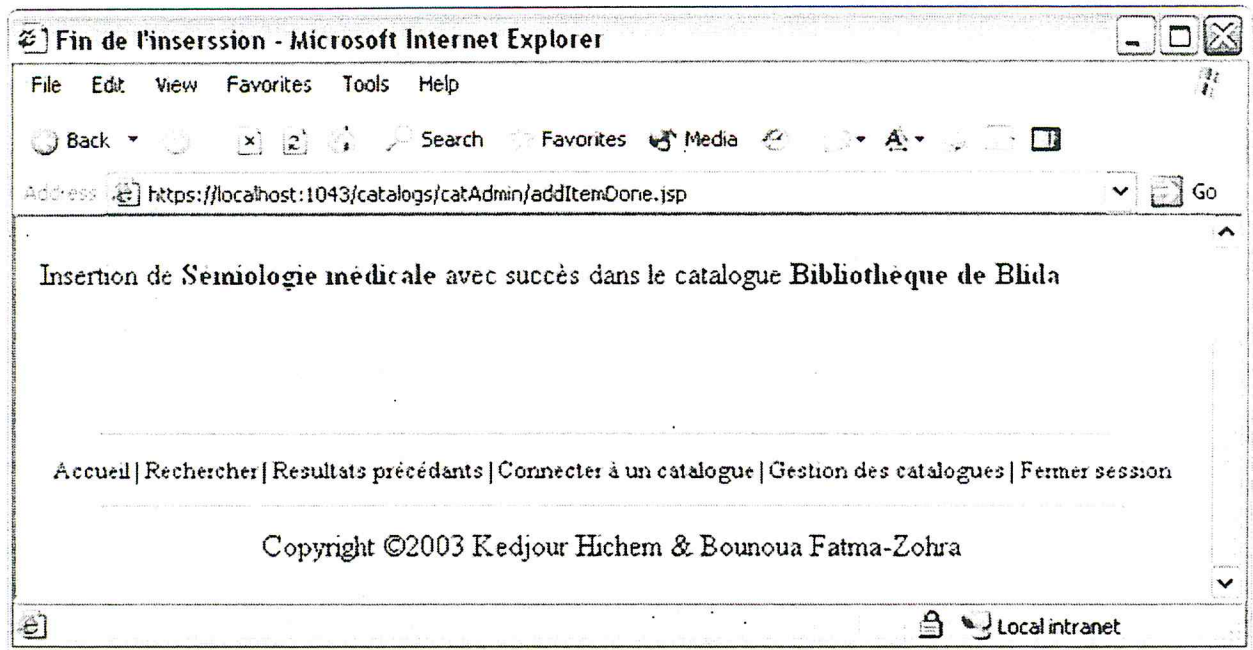


Figure 6-18 : Fin de l'insertion

5.4.3 L'interface Administrateur de l'application

Cette interface permet de gérer l'ensemble des catalogues participant à l'union. Elle permet d'ajouter de nouveaux catalogues (et de les supprimer). Ces opérations sont réservées aux administrateurs de l'application. Elles sont toutes regroupées dans une seule page web (Figure 6-19).

L'administrateur passe d'abord par une étape d'authentification. Nous pouvons utilisé n'importe quel mécanisme d'authentification J2EE. Pour notre application, nous avons utilisé l'authentification par formulaires.

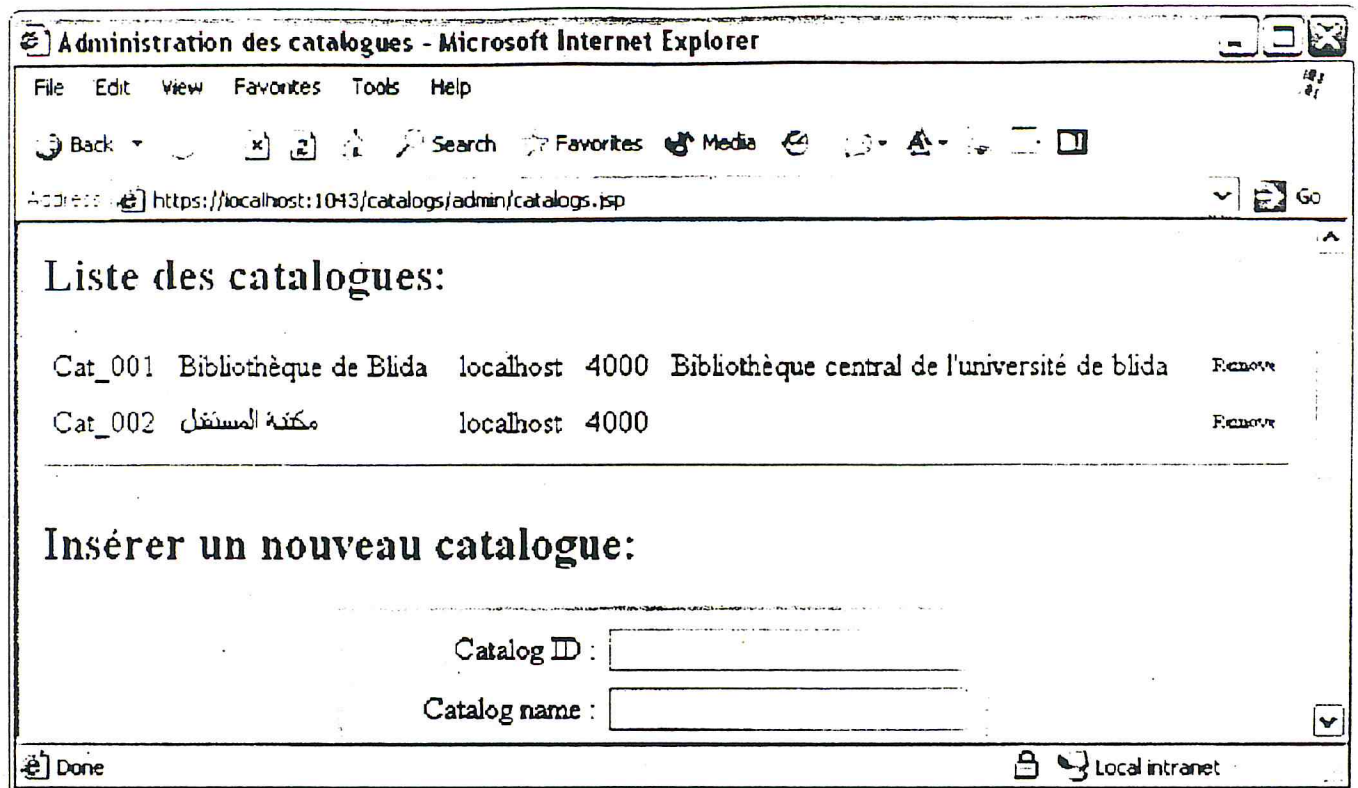


Figure 6-19 : Administration de l'application

La page affiche la liste des catalogues qui existent plus un petit formulaire qui permet d'insérer un nouveau catalogue. Pour cette dernière opération, il faut indiquer l'adresse du serveur où est localisé le catalogue et le numéro du port sur lequel il est disponible.

5.4.4 Fonctionnement de l'interface

Nous vous rappelons que les services de l'application sont présentés par deux objets seulement, **Catalogs** pour faire des opérations sur les catalogues et **CatalogsManager** pour gérer l'application. L'interface est composée d'un ensemble de pages Jsp faisant appel à ces deux objets.

Les pages de l'interface OPAC et de l'administration des catalogues ont toutes les deux besoin d'une instance de l'EJB **Catalogs**. Pour gérer cette instance, nous avons utilisé un JavaBean (appelé **CatalogsSessionBean**). Il contient une seule méthode `getCatalogs()`. Toutes les pages contiennent la ligne suivante :

```
<jsp:useBean id="cat" scope="session"
    class="webBeans.CatalogsSessionBean" />
```

Elle permet d'instancier l'objet `cat` du type **CatalogsSessionBean** qui va gérer une instance de **Catalogs**.

Le rôle des pages est de récupérer les données de l'utilisateur, les vérifier, les transformer si nécessaire, appeler l'EJB, récupérer le résultat, le transformer si nécessaire, et l'afficher à l'utilisateur. Le pseudo code du Listing 6-12 montre le traitement nécessaire pour une recherche.

Listing 6-12 web/search.jsp

```

1 <jsp:useBean id="cat" scope="session"
  class="webBeans.CatalogsSessionBean" />
  <%
2 String ldapQuery = request.getParameter("ldapQuery");
  %>
3 <c:set var="srchRes" >
  <%= cat.getCatalogs().ldapSearch(ldapQuery) %>
  </c:set>
4 <c:import var="theXslt" url="/xsl/SearchResult.xsl" />
5 <x:transform xslt="${theXslt}" xml="${srchRes}" />

```

- La 1^{ère} ligne crée une instance de **CatalogsSessionBean**.
- La 2^{ème} ligne récupère la requête LDAP entrée par l'utilisateur. Elle est affectée à la variable `ldapQuery`.
- La 3^{ème} ligne lance la recherche en appelant la méthode `ldapSearch()` de l'EJB **Catalogs** et affecte le résultat à la variable `srchRes`.
- La 4^{ème} ligne charge le fichier de la transformation XSL qui permet de transformer le résultat XML vers HTML et stocker le résultat de la transformation dans la variable `theXslt`.
- La 5^{ème} ligne transforme le résultat de la recherche en page HTML et l'affiche.

Dans cette opération de recherche, le résultat a été obtenu sous format XML, puis il a été transformé en HTML avant son affichage. Nous pouvons également trouver l'opération inverse comme le cas d'une d'insertion [Listing 6-13].

Listing 6-13 web\catAdmin\addItemDone.jsp

```

1 <jsp:useBean id="cat" scope="session"
  class="webBeans.CatalogsSessionBean" />
  <%
2 FormToXml trans = new FormToXml("add-item");
3 trans.transform(request);
4 String transRes = trans.getResult();
5 cat.getCatalogs().AddDocument(
  request.getParameter("_jspParam_CatalogId"),
  transRes);
  %>

```

- La 1^{ère} ligne crée une instance de **CatalogsSessionBean**.

- La 2^{ème} ligne crée une instance du transformateur XML qui permet de transformer les données d'un formulaire en XML.
- La 3^{ème} ligne transforme le formulaire.
- La 4^{ème} ligne récupère le résultat de la transformation.
- La 5^{ème} ajoute le nouveau document dans le catalogue identifié dans le paramètre `_jspParam_CatalogId` de la page.

A travers ces deux exemples, vous avez bien vu l'avantage de l'architecture multi-étages. Le code de l'interface ne contient pas la logique métier de l'application, il s'occupe seulement d'afficher et de récupérer les données.

Pour l'interface Administrateur de l'application c'est la même technique mais avec **CatalogsManager**.

Toutes les pages partagent la même entête et le même pied de page. Ces derniers sont conçus comme des pages indépendantes. Pour les inclure dans les autres pages, nous avons utilisé la directive `jsp:include` comme suit :

```
<jsp:include page="/common/banner.jsp" flush="true" />
```

En plus, les pages partagent aussi les feuilles de style pour garder la même apparence dans toute l'application.

6. Démarche de développement et évaluation

Dans le développement de l'application, nous n'avons pas suivi l'ordre décrit précédemment, c'est-à-dire développer les objets services puis les EJBs puis les interfaces, la programmation s'est déroulée autrement comme nous l'expliquons dans les paragraphes qui suivent.

6.1 L'apprentissage

Pendant la collecte des informations, nous avons commencé par apprendre un minimum de Java. Nous nous sommes appliqués par la suite à étudier LDAP à travers les programmes des exemples trouvés dans la documentation. Nous avons fini par écrire nos propres codes de test. Et nous avons adopté la même méthode d'apprentissage pour CORBA, J2EE, et XML.

Arrivés à l'étape de la conception, nous avons déjà des mini-codes pour faire pratiquement toutes les opérations (interroger les serveurs LDAP, créer des objets et des clients CORBA et de petites applications J2EE).

6.2 Création d'un prototype

Après quelques essais pour la mise en œuvre de la globalité du système, nous n'étions plus sûrs s'il sera réalisable. Pour s'assurer, nous avons développé un petit prototype. Il y avait un `CatalogSearcher` avec une

seule méthode `search()` sans paramètres (la requête était incluse dans le code source). Un petit EJB appelé aussi `CatalogSearcher` plus une page Jsp (il n'y avait pas XML, nous ne le connaissions pas encore).

Après des efforts, le prototype a fonctionné. Bien que le système était basique nous avons tout de même beaucoup d'indications par exemple, qu'il était nécessaire d'utiliser un langage tel que XML. Ce qui nous a amené à modifier les schémas de conception.

Cette étape nous a été plus que bénéfique puisque nous nous sommes familiarisés avec les technologies LDAP, CORBA et J2EE.

6.3 Développement et tests

L'application a été créée étape par étape. A chaque fois que nous modifions une fonction, nous compilons l'application pour tester l'effet de la nouvelle fonction.

Ce procédé nous a permis de détecter rapidement les erreurs et de les corriger. Il nous a également permis de mieux étudier les cas particuliers et de les traiter plus facilement.

6.4 L'évaluation

D'après les tests, l'application répond aux besoins fixés au préalable. La gestion de la concurrence, le cycle de vie des objets, la sécurité,... ont été garantis par CORBA et par J2EE. Reste que les prototypes ont besoin d'être perfectionnés ne serait ce que sur le plan de la sécurité.

La connexion avec l'annuaire n'est pas encore sécurisée, et l'objet `ServiceManager` n'est pas tout à fait implémenté.

7. Conclusion

L'objectif de départ était de créer une application qui gère l'ensemble des catalogues d'un réseau documentaire. Cette application a été conçue en utilisant le modèle J2EE. Elle contient trois parties, une partie pour gérer les catalogues, une pour regrouper l'ensemble des catalogues, et une qui représente les interfaces utilisateur.

Chaque catalogue est implémenté dans un annuaire LDAP et présente ces services à travers des objets CORBA.

Le middleware est implémenté par des Enterprise Java Beans, il permet de regrouper virtuellement tous les catalogues au sein d'un seul catalogue global.

Et finalement, les interfaces utilisateur implémentées comme une application Web permettent de viser un très grand nombre d'utilisateurs.

CONCLUSION

Conclusion

La gestion des documents est une tâche nécessaire pour préserver notre patrimoine de connaissance. Le catalogage permet de décrire précisément les documents et de garder leur trace. Ceci a rendu l'accès aux documents relié aux catalogues, et les documents mal catalogués sont des documents perdus. Avec les ordinateurs les catalogues ont quitté leur support papier, ce qui les a rendu accessibles à tout le monde à travers les interfaces OPACs.

Notre objectif était de créer une application qui permet de gérer un ensemble de catalogues.

Après l'étude des catalogues, nous avons vu que les annuaires forment le meilleur support pour les stocker. Les annuaires optimisent bien les opérations effectuées sur les catalogues (fréquemment la recherche et l'insertion) et surtout garantissent leur extensibilité pour supporter les nouveaux documents.

Les fonctionnalités des catalogues sont présentées par des objets services. Cette approche permet de faciliter leur gestion. L'installation des nouvelles fonctionnalités, le changement de l'annuaire, ou autre mise à jour ne va pas entraîner une modification dans le reste du système. Elle permet ainsi d'intégrer facilement les catalogues existants (qui n'utilisent pas LDAP).

Par exemple si nous voulons que les clients d'une bibliothèque puissent savoir si le document est disponible, nous aurons à gérer la file d'attente des emprunts (appelée aussi la gestion de la circulation des documents). Il suffit dans ce cas de développer et d'implémenter un nouveau service appelé par exemple GestionnaireDeCirculation (CirculationManager). L'ajout de ce service n'aura aucun effet sur les applications existantes.

Ces services sont implémentés sous forme d'objets CORBA, ce qui facilite leurs développements. CORBA est indépendante des systèmes et des langages de programmation, ce qui permet de développer des clients sur n'importe quel système.

L'application qui gère le système global (tous les catalogues) est une application J2EE, elle présente les services fournis par les catalogues à travers des EJBs. Cette application donne l'illusion aux clients qu'il existe un seul catalogue. L'intégration d'un nouveau catalogue est rendue simple.

CORBA, J2EE, et LDAP fournissent des fonctions de gestion de la concurrence, la sécurité, et tout ce qui est commun dans les environnements distribués.

L'application présente aussi une simple interface Web accessible au grand public d'Internet.

Il reste probablement des améliorations à faire au niveau de l'application mais nous tenons à énumérer quelques points importants qui éviteront de reprendre tout le code en cas de modification :

1. Pour l'ajout de nouvelles fonctionnalités, il suffit de rajouter des objets services et le ServiceManager s'occupe du reste.
2. Pour modifier les interfaces, les transformations ne porteront que sur les fichiers XML.
3. Pour la définition de nouveaux types de document, il faut simplement revoir le schéma de l'annuaire.

Nous ne considérons pas le travail présenté dans ce mémoire comme terminé.

Une éventuelle extension pour le projet serait de le doter des fonctionnalités suivantes :

- Implémentation de la méthode GetCatalogService de l'objet ServiceManager, et l'ajout de méthodes pour énumérer les services fournis.
- Amélioration de la technique de modification afin que l'utilisateur puisse spécifier les attributs à modifier, ceux à supprimer et ceux à garder.
- La possibilité de spécifier les catalogues à interroger.
- La possibilité d'accéder à l'application par un téléphone cellulaire ou un PDA (Personal Digital Assistants).
- La possibilité de spécifier un rayon de recherche pour les utilisateurs (50 Km, par exemple).
- L'ajout à l'EJB Catalogs d'autres opérations de recherche.

BIBLIOGRAPHIE

Bibliographie :

[ADA 01] Adam Tauno Williams, *LDAP and OpenLDAP (on the Linux Platform)*,
<ftp://kalamazoolinux.org/pub/pdf/ldapv3.pdf>, 2001

[ANN 96] Annie T. Luk, *Evaluating Bibliographic Displays from the Users' Point of View: A Focus Group Study*, LIS 2888L Directed Research, 1996

[BIB 01] Biblio2001fr, *Catataloguer avec le SU*,
http://membres.lycos.fr/biblio2001fr/travail_quotidien/catalogage_avec_WinIBW/WinIBW.htm, 2001

[DIE 98] Diego-Angel DIAZ, *Le repérage de l'information sur Internet : catalogage et indexation des ressources sur le World Wide Web*, <http://membres.lycos.fr/ddiaz/>, 1998

[GOP 98] Gopalan Suresh Raj, *A Detailed Comparison of CORBA, DCOM and Java/RMI*, <http://my.execpc.com/~gopalan/misc/compare.html>, Septembre 1998

[HEI 98] Heinz Johner, Larry Brown, Franz-Stefan Hinner, Wolfgang Reis, Johan Westman, *Understanding LDAP*, SG24-4986-00, International Technical Support Organization, June 1998

[HEI 99] Heinz Johner, Michel Melot, Harri Stranden, Permana Widhiasta, *LDAP Implementation Cookbook*, SG24-5110-00, International Technical Support Organization, June 1999

[JEA 02] Jean-François Pillou, *Introduction aux annuaires LDAP*,
<http://www.commentcamarche.com>, 2002

[MIC 00] Michel Lai, *UML La notation unifiée de modélisation objet De Java aux EJB*, 2e édition, DUNOD, 2000

[SYL 98] Sylvain BARTHELEMY, *CORBA 2.0, introduction par l'exemple*,
<http://www.barth.netliberte.org>, Avril 1998

[THI 96] Thierry Samain, *L'accès aux catalogues des bibliothèques par Internet*, Bibliothèque de l'Université Laval (Québec), 1996

[TIM 99] Tim Howes, Gordon Good, and Mark Smith, *Understanding and Deploying LDAP Directory Services, Chapter 1 - Directory Services Overview*,
<http://www.computerbooksonline.com/chapters/ldapchap.htm>, 1999

ANNEXES

1. Introduction

XML, *eXtensible Markup Language* (langage à balises extensibles), a été mis au point par le *XML Working Group* sous l'égide du *World Wide Web Consortium* (W3C) en 1996. Il s'agit d'un langage qui permet de structurer les données grâce à des balises (markup). Une balise est composée de mots encadrés par '`<`' et '`>`' du type: `<markup>`.

XML est à considéré comme un métalangage permettant de définir de nouvelles balises. Alors que HTML a un nombre de balises limité.

Contrairement à HTML, les balises XML sont destinées à décrire le contenu plutôt que la présentation. Ainsi, le contenu est séparé de la présentation. Un document XML est un format texte.

2. Structure d'un document XML

Un document XML est structuré en de 2 parties :

L'entête : appelée prologue contient le numéro de version de la norme XML utilisée pour créer le document (cette déclaration est obligatoire) ainsi que le jeu de caractères utilisé dans le document (ISO-8859-1, utf-8). La syntaxe du prologue est du type :

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

Le corps : Le contenu du document XML est structuré sous la forme d'une hiérarchie de balises. Chaque donnée est encadrée par une balise ouvrante `<balise>` et une balise fermante `</balise>`. Une donnée peut être composée d'un ensemble de données encapsulées par des balises.

Les balises peuvent comporter des attributs. Un attribut est de la forme `clé="valeur"`. La syntaxe d'une balise contenant un attribut est la suivante :

```
<balise cle="valeur">
Voici un exemple d'un document XML :
<?xml version="1.0" encoding="utf-8" ?>
<Document document-type="Monograph">
    <DocTitle>Thinking In Java</DocTitle>
    <Keyword-set>
        <Keyword>Java</Keyword>
        <Keyword>Programming</Keyword>
    </Keyword-set>
    <Cote>INF-2003</Cote>
    <Author>Bruce Eckel</Author>
</Document>
```



3. La mise en page d'un document XML avec XSL

Le standard XML définit la structuration des données et non leur représentation. Ainsi, la représentation de données de documents XML est assurée par XSL (eXtensible StyleSheet Language).

XSL est un langage permettant de définir des feuilles de style extensible pour les documents XML. Ce standard a deux composantes :

- **XSLT** (eXtensible Stylesheet Language Transformation) : le langage de transformation des données permettant de transformer un document XML à l'aide des feuilles de style contenant des règles en un autre document XML de structure différente ou en d'autres formats tels que HTML.
- **XSL/FO** (XSL Formatting Objects) : le langage de formatage des données. Il permet la mise en page de ce qui a été généré par XSLT.

Un document XSL est lui-même un document XML. Toute feuille de style XSL est comprise entre les balises `<xsl:stylesheet ...>` et `</xsl:stylesheet>`.

Les règles contenues dans cette feuille sont appelées *template rules*. Ce sont des balises définissant les transformations à effectuer sur les éléments de l'arbre produit par XSLT.

4. Les analyseurs

Il est essentiel de pouvoir récupérer les données contenues dans le document XML. L'*analyseur* ou *parseur* (en anglais *parser*) permet de parcourir le document XML et d'en extraire les informations. Ainsi, il crée une structure hiérarchique contenant ces informations.

Il existe deux APIs pour traiter les documents XML :

- **DOM** (Document Object Model) : Une API définissant la structure d'un document selon une représentation objet. Elle lit l'intégralité du document

XML afin de construire sa représentation en mémoire. Ensuite, le programme peut librement et rapidement parcourir cette représentation interne pour y garder les informations souhaitées.

- **SAX (Simple API for XML)**: Une API reposant sur un modèle événementiel. L'analyseur SAX s'occupe d'interpréter le document XML, identifiant les différentes balises et attributs. Dès qu'un élément est reconnu, SAX avertit le programme afin que celui-ci fasse un traitement approprié. Le programme reçoit donc des événements du type « ouverture de la balise X », « zone de texte », « fermeture de la balise ».

5. Conclusion

Xml se caractérise par les points suivants :

- **La lisibilité** : aucune connaissance ne doit théoriquement être nécessaire pour comprendre le contenu d'un document XML. En plus, le contenu est accessible par un éditeur de texte sans avoir besoin d'un logiciel spécifique.
- **Auto-descriptif**
- **Une structure hiérarchique** : permet de modéliser la majorité des problèmes informatiques.
- **Universalité et portabilité** : les différents jeux de caractères sont pris en compte.
- **Déployable** : il peut être facilement distribuer par n'importe quels protocoles à même de transporter du texte, comme HTTP.
- **Extensibilité** : XML permet de définir de nouvelles balises et par conséquent un document XML peut être utilisable dans tous les domaines d'applications.
- **Intégrabilité** : un document XML est utilisable par toute application qui possède un analyseur XML.

XML est devenu un standard adopté par des applications variées. Ainsi, il est particulièrement destiné à l'échange de données et de documents.