Modèle d'Administration des Systèmes Distribués à Base de Composants

Nada Al Masri

Institut National des Sciences Appliquées de Lyon

THESE

Pour obtenir le grade de
Docteur de l'Institut National des Sciences Appliquées de Lyon

Préparée au laboratoire : CITI
Dans le cadre du projet : DARTS
Et l'action ARES de l'INRIA

Ecole Doctorale : Informatique et Information pour la Société
Spécialité : Documents Multimédia, Images et Systèmes d'Informations Communicants

Présentée et soutenue par

Nada ALMASRI

Le 25 Mars 2005

Modèle d'Administration des Systèmes Distribués à Base de Composants.

Application à la Gestion de Déploiement d'EJBs dans une Fédération de Serveurs d'Applications

Directeurs de Thèse

M Stéphane Frenot
M Stéphane Ubeda

Jury

M Festor Olivier, Rapporteur
M Philippe Laurent
M Riveill Michel, Rapporteur
M Vandome Gérard

A la mémoire de mon père

Remerciement

Je tiens à remercier toutes les personnes qui ont rendu cette thèse possible par leurs aides et leurs contributions.

J'adresse mes sincères remerciements à Monsieur Stéphane FRENOT, Maître de conférence à l'INSA de Lyon et à Monsieur Stéphane UBEDA, professeur à l'INSA de Lyon pour leur encadrement et leur soutien tout au long des années de thèse.

Je remercie également tous les membres du jury :

Je remercie également tous les membres du jury :

- M Festor Olivier, Directeur de Recherche INRIA, LORIA
- M Philippe Laurent, Professeur à l'Université de Franche-Comté
- M Riveill Michel, Professeur à l'Ecole supérieur en Science Informatique (ESSI) / Université de Nice – Sophia Antipolis
- M Vandome Gérard, Ingénieur de recherche de la société Bull

Une pensée très spéciale à une personne généreuse qui m'a toujours aidée et encouragée, mon amie Hana BADAWI. Je remercie également mon amie de bureau Houda KHEDHER pour sa compagnie et pour son aide en Français.

Enfin, je garde une place toute particulière à ma mère qui a toujours été ma meilleure amie et m'a toujours soutenue et qui a su m'amener à ce niveau d'étude en me laissant toujours le choix de l'orientation. Je remercie également mon mari Mohammad et mes frères Anas, Amjad, et Aman qui m'ont beaucoup encouragée pendant la thèse.

Abstract

Component-Oriented Systems are increasingly evolving and being adopted by both industrial and academic worlds. They have resulted from the advances achieved in both software engineering and distributed systems.

Component models such as JavaBeans, COM, CORBA were presented as reusable components that can be composed together to get a complete software. At the same time, advances in distributed systems lead to another type of component models such as EJB, DCOM and CCM that treat the distribution as their main objective. Platforms conforming to these models support a number of middleware services that facilitates the distribution without explicit intervention from the system developers. These platforms can be seen as dedicated hosting servers of the applications' components.

In our works, we concentrate on the EJB component model; we present an administration platform for a federation of servers cooperating to host a number of applications. The federation includes different servers with different configurations, characteristics, services and resources. Our main objective is to manage the deployment of EJB components in the federation depending on their needs to the available resources and services offered by the servers.

The proposed administration platform is based on the JMX instrumentation platform. We propose a 3-level instrumentation model to collect management information from 3 environment levels: network and system low level, application server level, and components level. The administration platform gets all necessary information about the execution environment from this instrumentation platform modeled in 3 levels where all the elements are also modeled as managed objects.

Furthermore, the administration platform defines a number of services to control the deployment management in the federation such as a migration service, EJB evaluation service, and EJB instrumentation service. Beside these services, we define a number of rules to consider while taking decision concerning the deployment process and the communication between the application servers.

To apply a non-centralized management over the nodes of the federation, we propose deploying the administration services on all servers in the federation to get a distributed management.

Table des matières

Table de figures

Table de tableaux

1. Introduction

1.1. LES SYSTEMES DISTRIBUES ET LES MODELES A COMPOSANTS

1.2. MOTIVATIONS

1.3. OBJECTIFS

1.3.1. Instrumentation et Administration d'une infrastructure de serveurs

1.3.2. Déploiement d'applications sur un parc hétérogène de serveurs d'applications

1.4. PLAN

L'essor d'un parc considérable de micro-ordinateurs très puissants, du à la baisse des prix et à l'émergence de nouveaux processeurs plus performants au cours de la dernière décennie, ainsi que l'évolution croissante de l'Internet et des technologies informatiques a entraîné des modifications radicales dans la vision de conception des applications : Les nouveaux réseaux de télécommunication rapides et à large bande passante ont permis le regroupement des ordinateurs en réseaux, d'abord de taille réduite (entreprise), puis moyenne (nationale) et enfin mondiale avec Internet. Tout ceci a favorisé la démocratisation des communications informatiques et l'émergence des applications distribuées.

1.1. Les systèmes distribués et les modèles à composants

Les systèmes distribués sont une réalité depuis environ 30 ans. En offrant la possibilité à des machines réparties sur un réseau d'exploiter des ressources mises à disposition par d'autres machines, les barrières de la localisation et de la disponibilité ont été franchies. Il est maintenant devenu simple et évident d'interagir avec une application distante par l'intermédiaire d'une page web ou encore de synchroniser une base de données avec un site distant. Si le modèle d'interaction distant a été spécifié dès le début avec le principe des Appels de Procédures Distantes (RPC), il a largement évolué pour d'une part prendre en compte les avancées issues des objets et d'autres part les avancées issues des systèmes d'exploitation comme les machines virtuelles.

L'évolution des plates-formes à composants distribués a abouti dans les années 2000 à la définition d'infrastructures complètes d'hébergement de composants logiciels appelées serveurs d'applications (SA). Le rôle de ces serveurs d'applications est de standardiser le développement, le déploiement, les interactions et la gestion d'applications. Les applications développées conformément aux spécifications de ces infrastructures sont préparées (packagées) dans des structures de fichiers qui sont déposées sur les serveurs d'applications. Ceux-ci les rendent actives et sont alors utilisables par des clients localisés n'importe où, soit par des pages web, soit par des clients légers. Si les premiers jets conformes aux principes des serveurs d'applications ont été apportés par des plates-formes CORBA comme OrbixWeb d'Iona, l'adoption massive du modèle n'est apparue qu'avec la spécification J2EE de Sun (Le dernier acteur étant Microsoft avec son très récent .Net). L'infrastructure J2EE repose sur la notion de conteneur permettant l'exécution « isolée » de composants logiciels. Le conteneur de pages web permet d'héberger des applications orientées Web et le conteneur EJB permet d'héberger des applications orientées objet. Les applications ainsi hébergées peuvent :

- dialoguer avec d'autres applications similaires par l'intermédiaire d'appels de méthodes distantes (ou locales),
- exploiter des services offerts par le serveur d'applications,
- et enfin sont partiellement gérées en terme de cycle d'exécution par le conteneur qui les héberge.

1.2. Motivations

Les serveurs d'applications ont permis d'identifier très clairement deux nouvelles activités dans le domaine des applications distribuées à composants :

- Le packaging consiste à réunir dans une même unité un ensemble de fichiers composant l'application

- Le déploiement consiste à installer l'unité de packaging de manière atomique sur un serveur d'applications, en vérifiant les liaison avec les services externes (table de bases de données, moteur de page Web…) et en mettant l'application à disposition des conteneur d'exécution.

Un des succès des serveurs d'application provient du fait qu'ils reposent sur les machines virtuelles java. Les machines virtuelles permettent principalement d'obtenir une standardisation d'un système d'exploitation. Ainsi les serveurs d'applications ne dépendent plus du système d'exploitation hôte mais uniquement de la disponibilité d'une machine virtuelle java.

La machine virtuelle a apporté aux applications distribuées l'homogénéisation nécessaire à leur mise en œuvre et à leur exploitation simple et fiable. Mais en masquant les spécificité des systèmes d'exploitation elles ont contribuées à masquer les besoins réels des applications et les ressources offertes par le système hôte. Les systèmes reposant sur des machines virtuelles sont vus comme des systèmes « best-effort » où les applications sont hébergées « temps que ça tient » sans prendre en compte l'état du système. Si cette approche est initialement nécessaire pour permettre une cohabitation "la plus facile possible", elle présente trop problèmes sur le long terme. En effet elle ne permet pas de répondre à un certain nombre d'interrogations qu'on peut se poser concernant le déploiement d'applications :

- Si mon parc de serveurs d'applications est hétérogène, sur quel serveur dois-je déployer telle ou telle application (partiellement ou totalement) ?

- Quel est l'état à l'instant t de tel ou tel serveur d'applications ? A t'il suffisamment de ressources disponibles ?
- Les clients qui s'y connectent seront-ils satisfaits en terme de temps de réponse ?

Si le problème de la gestion de ressources en environnement temps-réel, embarqué ou propriétaire est une problématique largement étudiée, elle l'est nettement moins dans le domaine du « best-effort ». La complexité des couches intermédiaires entre le système d'exploitation et les applications (machines virtuelles, serveurs d'applications, composants logiciels), la diversité des systèmes et des réseaux sont autant de facteurs de freins à l'évaluation et au suivi de tels systèmes.

1.3. Objectifs

Dans ce travail nous cherchons à fournir un début de réponse concernant la gestion d'applications distribuées à base de composants et de serveurs d'applications. Nous nous sommes centré sur le modèle des EJB et des serveurs d'applications J2EE. Afin de pouvoir optimiser cette gestion, notre travail s'est centré sur deux axes :

- Comment instrumenter et administrer une infrastructure de serveurs d'applications ?
- Comment gérer le déploiement d'application sur un parc hétérogène de serveurs d'applications ainsi instrumentés ?

1.3.1. Instrumentation et Administration d'une infrastructure de serveurs

Cet partie pose le problème de la remonté et la standardisation des informations concernant les ressources disponibles sur les serveurs d'applications. En masquant le système les machines virtuelles masquent également les ressources mises à disposition des Serveurs d'applications. De plus les serveurs d'application présentent plusieurs « couches » qui masquent également l'accès à ces ressources (couche d'appel distant, couche d'interposition de services, API d'accès aux services, conteneur d'exécution…). Nous avons cherché dans ce travail à définir une infrastructure de supervision et d'instrumentation permettant d'identifier tous les éléments de toutes les couches du système. Cette infrastructure offre un modèle

standardisé de supervision directement exploitable par l'infrastructure de déploiement d'applications.

1.3.2. Déploiement d'applications sur un parc hétérogène de serveurs d'applications

Nous définissons dans ce travail la notion de fédération de serveurs d'applications. Une fédération est constituée par un ensemble hétérogène de machines hébergeant des serveurs d'applications instrumentés. Notre travail propose de définir une infrastructure de déploiement globale et adaptée d'applications sur cette fédération. Parmi les nombreux avantages offerts par une telle fédération nous avons étudié les possibilités suivantes :

- Evaluation quantitative des applications à installer
- Optimisation du déploiement en fonction des ressources disponibles sur la fédération
- Relocalisation d'application en cas de changement du contexte d'exécution.

1.4. Plan

La suite de ce travail est structurée ainsi :

- Chapitre **2** : présente le cadre de notre travail et l'état de l'art sur l'administration des systèmes à base de composants.
- Chapitre **3** : décrit la problématique et introduit notre proposition pour l'administration du déploiement sur une fédération de serveurs d'applications. Nous montrons ensuite deux scénarios d'utilisations.
- Chapitres **4 :** présente la première partie de notre proposition concernant les services de gestion que nous proposons.
- Chapitre **5**: explique comment nous établissons la communication entre les serveurs de la fédération, et présente un modèle de coopération afin d'établir une action de déploiement sur la fédération.
- Chapitre **6** : conclut la thèse avec un bilan de travaux réalisés et les perspectives.

2. Systèmes à composants et administration

Notre objectif est de proposer une architecture d'administration adaptée à la gestion d'une fédération de serveurs à composants. Dans ce contexte nous présentons d'une part les systèmes à base de composants, puis nous aborderons les systèmes d'administration classique. Nous fournissons ensuite un état de l'art des travaux portant sur l'administration de systèmes distribués. Dans une dernière partie nous apportons nos commentaires concernant l'adéquation entre les systèmes d'administration et le principe de fédération de serveurs.

2.1. Cadre du travail

2.1.1. Systèmes distribués à base de composants

Les systèmes distribués à base de composants sont les fruits de l'union de deux concepts : *la programmation par objet* et *les systèmes distribués*. La programmation objet permet la décomposition logique d'une application, ce qui facilite l'implantation des systèmes distribués.

Un objet distribué (ou réparti[*]) est un composant logiciel indépendant contenant sa propre intelligence de fonctionnement et capable d'interopérer avec d'autres objets distribués, indépendamment des types d'ordinateurs, des langages de programmation ayant servi à les développer, ou des systèmes d'exploitation. Les objets distribués ont radicalement modifié les phases de conception, développement et distribution des logiciels dans le sens où un objet est devenu à la fois une unité de composition, de distribution et de déploiement. Cet objet multi-tâches est appelé un *composant* [BAC00, NIE95, PLA98, SZY02].

L'OMG, Microsoft, Sun ont chacun présenté leur propre approche pour l'implantation des systèmes distribués à base de composants :

- **L'OMG (CORBA/CCM):**

Depuis 1989, une association internationale appelée l'OMG *(Object Management Group)* [OMG], définit la spécification de l'architecture d'un système à objets répartis, appelée CORBA object model *(Common Object Request Broker Architecture)*. Ce model a été étendu en 2002 pour présenter une modèle de composant plus souple appelé CCM (CORBA Component Model), CCM offre en plus du modèle de composant CORBA des services middleware, ainsi qu'une infrastructure d'exécution et de déploiement de services.

[*] Les mots « distribué » et « réparti » sont deux traductions possibles pour le mot «distributed» en anglais. Dans ce travail les deux termes sont des synonymes pour la même technologie.

- **Microsoft (DCOM/.Net) :**

DCOM (Distributed Component Object Model) [DCOM] est le concurrent direct de CORBA que cherche à imposer Microsoft pour les plates-formes de type Windows. Ce Modèle a été présenté en 1996 comme une extension du modèle COM (Componenet Object Model) qui n'étais pas destiné aux applications distribués. Finalement Microsoft a présenté .Net qui ne représente pas un modèle à composants, mais une plateforme d'implantation des systèmes distribués qui intègre l'ensemble des technologies Microsoft existants (COM/COM+, DCOM, Web services, …). .Net a introduit le support de web services avec l'adoption du protocole SOAP (Simple object Access Protocol) basé sur XML (Extensible Markup Language) en plus de DCOM.

- **Sun:**

EJB(Enterprise Java Beans) [EJBb03] est le modèle à composants qui a été proposé par Sun pour les systèmes d'objets distribués. Ce modèle est basé initialement sur la technologie RMI (Remote Method Invocation) permettant l'invocation de méthodes d'objets java à distance. Il adopte aujourd'hui le protocole RMI-IIOP pour permettre l'interopérabilité avec CORBA. De plus, Sun définit la plate-forme J2EE qui représente un ensemble de services middleware permettant l'exécution des composants EJB. Dernièrement, Sun a présenté SunOne (Sun Open network Environment) le concurrent de .Net, une extension de la plate-forme J2EE qui intègre les web services.

Le Tableau 2.1 présente les technologies équivalentes des trois environnements.

	Modèle à composants	Infrastructure	Invocation distante
OMG	CORBA	CCM	ORB
Sun	EJB	J2EE	RMI
Microsoft	COM/webservices	.Net	SOAP/DCOM

Tableau 2.1 Analyse des plates-formes de systèmes distribués à base de composants

Parmi ces trois modèles à composants, nous détaillons le modèle que l'on a adopté dans cette thèse : le modèle EJB, qui représente le cœur de la plate-forme J2EE.

2.1.1.1. Plate-forme J2EE et modèle de composants EJB

La plate-forme J2EE (Java 2 Enterprise Edition) [J2EE03] présentée par Sun définit une architecture multi-niveaux (ou n-tiers) pour les applications distribuées java. Elle spécifie à la fois l'infrastructure des applications distribuées multi-niveaux et les API (Application Programming Interface) des services utilisés pour concevoir ces applications.

L'infrastructure des applications se constitue de plusieurs conteneurs sur les différents niveaux. Un *conteneur* représente l'environnement d'exécution chargé de gérer des composants applicatifs et de leur donner accès aux API J2EE. Comme illustré dans Figure 2.1, les conteneurs définis par Sun sont les suivants :

- Conteneur d'application client : Gère l'exécution des applications clients. Ce conteneur tourne sur la machine cliente.
- Conteneur d'applet : Gère l'exécution des applets. Il se constitue d'un navigateur Web et un d'un JavaPlug-in qui s'exécutent ensemble sur un client.
- Conteneur web : Gère l'exécution des pages JSP (Java Server Pages) et des servlets pour les applications J2EE. Les composants Web et leur conteneur s'exécutent sur un serveur J2EE.
- Conteneur d'EJB : Gère l'exécution des composants métiers appelés EJB (Enterprise Java Beans). Les EJB et leur conteneur s'exécutent sur le serveur J2EE.

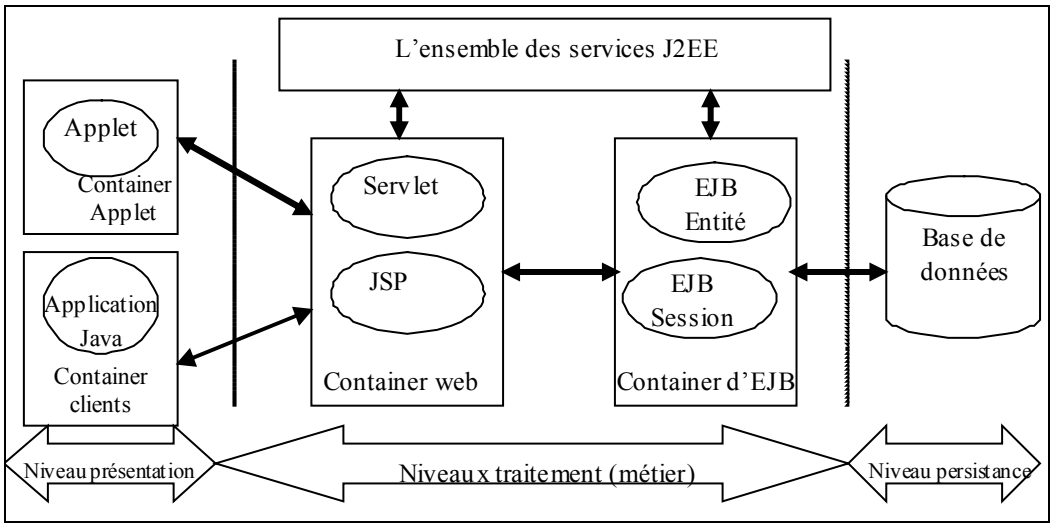

Figure 2.1 La Plateforme J2EE

Ces conteneurs fournissent aux applications un ensemble de services comme les connexions aux bases de données, la messagerie, les transactions, etc. L'architecture J2EE unifie l'accès à ces services par des API.

Le tiers de traitement propose deux conteneurs, le conteneur de présentation avec des servlets, et le conteneur d'application pour la logique métier avec les composants EJB.

Composants EJB

Un composant EJB (Enterprise JavaBeans) [EJBa03] est un composant Java portable, réutilisable et déployable qui peut être assemblé avec d'autre composants EJB pour créer des applications. Les EJBs suivent le modèle de composants d'EJB et s'exécutent dans un conteneur EJB qui leur fournit des services non fonctionnels. Notamment les services principaux fournis sont : la transaction, la persistance, la sécurité, et la localisation.

Le modèle de composants EJB définit 3 types de composants EJB et un descripteur de déploiement XML avec lequel le conteneur associe les services nécessaires aux EJBs. Ces 3 types sont les suivants :

- Les Enterprise JavaBeans de type ***session***

  Ce type de composant s'occupe de la partie traitement de l'application. Il est chargé d'effectuer une tâche pour un client. Dans cette catégorie d'Entreprise JavaBeans, il y a deux groupes de composants EJB :

  - **Sans état (stateless)** : Un composant de session sans état ne maintient pas d'état conversationnel avec le client. L'exemple typique est un convertisseur Euro/Franc qui aurait une seule méthode : `euroToFranc(double valeur)`. Les différentes invocations de méthodes sur un EJB sans état ne seront pas forcément traitées par la même instance d'EJB.
  - **Avec état (stateful)** : Un composant avec état est dédié à un certain client pendant toute la durée de son instanciation. Concrètement, si le client modifie une variable d'instance du composant, il retrouve cette valeur lors de ses prochains appels. Un exemple sur ce type d'EJB est un caddie virtuel ; quand un client ajoute ou supprime un objet l'EJB maintient ces modifications pour ce client.

- Les Enterprise JavaBeans de type ***Entité***

  Ce type d'EJB s'occupe de la partie "données" de l'application. Il peut désigner quelque chose de concret (un client ou une facture) ou d'abstrait (une enchère ou une réservation). Sa caractéristique la plus importante est la *persistance,* c'est à

dire que ce composant existe sur un support de stockage persistant comme une base de données.

- Les Enterprise JavaBeans de type ***Message-Driven***

  Un Message-Driven Bean est un EJB qui permet à l'application de traiter des messages de manière asynchrone. En fait, cet EJB réagit aux messages reçus au travers de JMS (Java Message Service).

Un EJB, quel que soit son type, est constitué des éléments suivants :

- **Classe du bean**, qui encapsule les données associées au bean et contient les méthodes métier implémentées par le développeur qui permettent d'accéder à ces données. Elle renferme aussi les différentes méthodes utilisées par le conteneur pour gérer le cycle de vie de ses instances.
- **Interface Home**, qui définit les méthodes utilisées par le client pour créer, rechercher, et supprimer des instances du bean. Elle est mise en oeuvre par le conteneur lors du déploiement dans une classe appelé EJB Home.
- **Interface Component,** qui est mise en oeuvre par le conteneur lors du déploiement dans une classe appelée EJBObject. Le client du bean après avoir utilisé l'interface EJB Home pour accéder au bean, utilise cette interface pour appeler indirectement les méthodes métier implémentées dans la classe bean.
- **Classe Primary Key**, qui est définie seulement pour un entity bean. Elle enferme une ou plusieurs variables qui identifient de manière unique une instance de bean d'entité spécifique. Cette classe contient en plus les méthodes qui permettent de gérer des objets Primary Key.

Dans la section suivante nous présentons plus en détails la fonctionnalité des interfaces home et component.

Interfaces *Home* et *Component* et objets EJBObject et EJBHome:

Un client peut accéder aux EJB beans à travers les interfaces *home* et *component*. Ces interfaces offrent une vue sur l'EJB au client. L'interface h*ome* d'un EJB définit les méthodes de cycle de vie d'un EJB : des méthodes pour la création de nouveaux objet de l'EJB, pour la suppression des objets EJB, et pour chercher des objets. L'interface *component* définit les méthodes métiers : les méthodes qu'un bean présente au monde extérieur.

Le conteneur génère d'une manière automatique les classes qui implémentent les interfaces home et component et qui introduisent les appels explicites des API des services non fonctionnels pour gérer les transactions, la sécurité, la persistance et d'autres aspects non - fonctionnels. Le conteneur instancie ensuite un seul objet de la classe EJBHome qui implémente l'interface home et zero ou plusieurs objets de la

classe EJBObject qui implémente l'interface component. Bien que la vue client de l'EJB déployé sur le serveur soit fournie par des classes implémentées par le conteneur, le conteneur lui-même reste transparent au client.

Pour qu'un client récupère une référence d'un EJB, il contacte d'abord l'objet EJBHome (issu de l'interface home) et appelle la méthode de création d'un nouvel objet de l'EJB. Cette méthode crée un objet EJBObject (issu de l'interface component) et le renvoie au client. Le client peut alors appeler les méthodes métier définies par l'EJB.

Tous les deux, l'interface home et l'interface component, peuvent être définies comme des interface remote ou bien des interface locales. De ce fait, un client d'un EJB peut être *remote* ou *local* selon si le bean définit des interfaces remotes ou locales.

Les interfaces *remote component* et *remote home* sont des interfaces RMI. Le conteneur crée des objets qui implémentent ces interfaces et qui représentent l'EJB. Ces objets sont des objets java de type « remote java objects » et ils sont accessibles depuis les APIs standard de JAVA RMI. Les arguments des méthodes des interfaces remote component et remote home sont passés par copie. Un client remote peut accéder à l'EJB depuis une machine virtuel java (JVM) différente de la JVM de l'EJB qui s'exécute.

Un client local doit être dans la même JVM que celle où l'EJB s'exécute. Ce client accède au bean à travers des interfaces *local component* et *local home.* Le conteneur crée des classes qui implémentent ces interfaces pour répresenter le bean. Les objets issus des ces classes sont des objets java locaux non accessibles à distance. Les arguments et les résultats des méthodes sont passés par référence. Un client local d'un EJB peut être un autre EJB.

Le choix des interfaces locales ou remote dépend de l'utilisation de l'EJB. Si l'EJB ne sera jamais appelé depuis l'extérieur, alors l'utilisation des interfaces locales est préférable car les interfaces remotes impliquent une surcharge du réseau et des piles du client et du serveur.

Descripteur de déploiement des EJB

Pour que le conteneur puisse créer les objets EJBObject et EJBHome, il a besoin de savoir comment l'EJB doit être géré au moment de l'exécution (at runtime). Ces informations concernent la description du type de transaction, de sécurité, de nommage et de persistance que le bean attend de recevoir de la part du serveur. Le comportement de l'EJB à l'exécution dépend de ces informations.

La spécification des EJB demande l'existence de ces informations à l'aide d'un fichier XML appelé *descripteur de déploiement.* Le fichier est construit selon

un fichier DTD standard. Le document définit plusieurs balises XML qui décrivent pour chaque EJB codé la description des services non fonctionnels, les ressources et les autres EJB référencés par l'EJB d'origine.

Fichier jar et déploiement

Un fichier jar est un fichier zip utilisé spécialement pour packager les classes java. Il peut être utilisé pour packager des applets, des applications java, des javaBeans, et des EJB.

Après avoir développé les EJB, c'est le rôle de déployeur de regrouper les EJB dans un package jar et de fournir le descripteur de déploiement. Notamment le jar doit contenir les fichiers suivants :

- La classe de l'implantation de l'EJB qui contient le code des méthodes métiers et les méthodes d'instanciation (l'équivalent des constructeurs).
- L'interface component qui décrit les méthodes métiers de l'EJB. C'est l'interface qui est utilisée pour donner la vue du bean au client. Il est également utilisé par le conteneur pour créer l'objet EJBObject.
- L'interface remote qui décrit les méthodes de cycle de vie de l'EJB. L'interface est utilisée par le client. Le conteneur utilise également cette interface pour créer l'objet EJBHome.
- La classe « primary key » est une classe simple qui fournit un pointeur sur une base de données. Cette classe est nécessaire seulement pour les EJB de type entity.
- Le descripteur de déploiement qui indique au conteneur comment créer les classes d'implantation de l'EJB et comment l'EJB interagit avec les services non fonctionnels du serveur.

Déploiement et accès aux EJBs

Le déploiement d'un composant EJB est un processus de préparation pour rendre l'EJB utilisable. Le déployeur s'occupe de packager les EJB, de fournir le descripteur de déploiement et ensuite d'activer le processus de déploiement sur un serveur. Le serveur, selon les descripteurs de déploiement du composant, lie le composant aux services non fonctionnels et l'enregistre dans le répertoire de nommage afin qu'il soit lié et accessible aux autres composants et aux clients. Précisément, le conteneurs crée un objet EJBHome pour chaque EJB et il le lie avec un nom unique (ce nom est défini par le développeur dans le descripteur de déploiement) dans le répertoire de nommage JNDI de la plate-forme J2EE.

Quand un client veut accéder à un EJB, il contacte le service de nommage JNDI de la plate-forme J2EE et il demande l'objet lié avec le nom de l'EJB. Il

obtient alors l'objet EJBHome généré automatiquement par le conteneur. Le client appelle la méthode de création d'un nouvel objet EJB. La méthode renvoie un objet EJBObject que le client peut finalement utiliser pour appeler les méthodes métiers de l'EJB.

Les serveurs d'applications EJB (Jonas, JBoss)

Un serveur d'applications EJB est l'environnement d'exécution des applications côté serveur. Il prend en charge l'ensemble des fonctionnalités qui permettent à N clients d'utiliser une même application à base d' EJB :

- Gestion de la session utilisateur (invocation à distance RMI-IIOP)
- Gestion de la persistance des composants EJB
- Gestion des montées en charge
- Gestion de la sécurité
- Gestion d'accès au répertoire de nommage

L'ensemble de ces fonctionnalités est fourni à l'aide des APIs standardisées par Sun. La Figure 2.2 montre le scénario d'un appel d'une méthode métier d'un bean déployé sur un serveur d'applications.

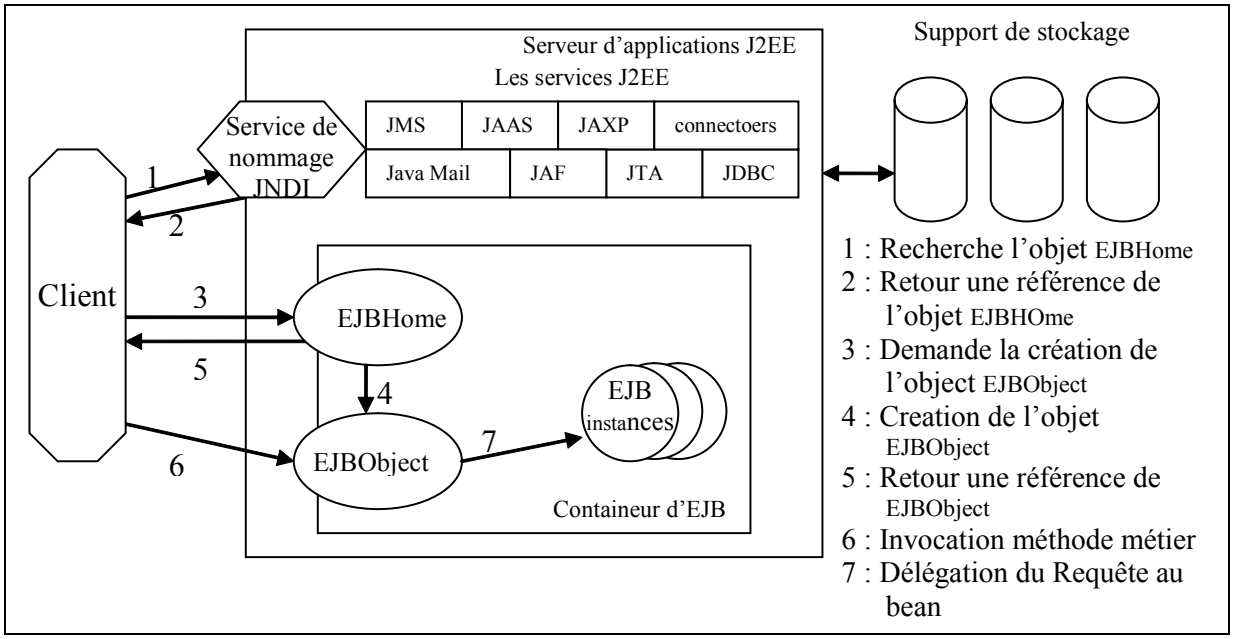

Figure 2.2 Scénario d'un appel d'une méthode métier d'un EJB

Sur le marché on trouve plusieurs implantations des serveurs d'applications EJB. Dans cette section nous présentons deux implantations open source : JBoss et JONAS.

- **Serveur d'applications JBoss :**

JBoss [JBOSS, FLEa02] est un serveur d'applications EJB développé par le groupe JBoss, un groupe de programmeurs volontaires. Le groupe a commencé par développer un conteneur EJB de base, ensuite il a continué à développer tous les autres services de J2EE. Le but final annoncé par le groupe est de développer une implantation complète des services J2EE.

L'architecture de JBoss se base principalement sur JMX (Java Management Extension) [JMX02, JMX03] une plate-forme de gestion d'applications java qui permet l'intégration de modules logiciels. Les services principaux du serveur JBoss sont développés comme des unités d'intégration JMX appelé MBean (Managed Bean). Tous ces services sont inscrits dans la même instance MBeanServeur du JMX. Le MBeanServeur est donc le cœur de JBoss, il permet l'ajout de nouveaux services ainsi que l'accès à tous les services enregistrés.

Quand le serveur JBoss démarre, une seule instance de javax.management.MBeanServer est créée. L'instance joue le rôle d'un composant micronoyau d'agrégation. Le noyau est donc une simple unité d'agrégation mais pas la source des fonctionnalités. Les fonctionnalités sont rendues par les MBeans qui sont interconnectés par le MBeanServer. Le serveur démarre comme un simple conteneur de MBeans, il charge ensuite sa propre personnalité en se basant sur les services définis dans un fichier de configuration jboss-service.xml. Le fichier contient les informations nécessaires pour le démarrage de chaque MBean : son nom absolu, ses attributs, et les autres MBeans dont il dépend.

- **Serveur d'applications JOnAS :**

JonAS [JONAS] est un serveur d'applications français développé dans le cadre du consortium Objectweb.

JOnAS est conçu comme un ensemble de services. La plupart des composants du serveur JOnAS sont pré-définis mais il est possible d'intégrer un nouveau service en suivant les standardisations de JOnAS.

Contrairement à JBoss, JOnAS ne se base pas sur JMX, mais il l'utilise pour définir un service de management, ou il instrumente tous les services JOnAS avec des MBeans.

2.1.1.2. Clusters dans la Plate-Forme J2EE

Les applications construites sur la plate-forme J2EE sont destinées à une utilisation sur le WEB et sont supposées fonctionner 24 heures sur 24 et servir un grand nombre de clients simultanément. La disponibilité et la capacité de croissance de l'application sont alors fondamentales pour répondre à tous moments aux requêtes des clients web dans un délai acceptable.

La capacité de croissance consiste à pouvoir répondre au nombre croissant des clients. Les clusters fournissent cette fonctionnalité en ajoutant des serveurs supplémentaires et en équilibrant la charge des requêtes sur l'ensemble des serveurs.

La disponibilité est le fait que l'application soit toujours accessible au client, même en cas de panne de certaines machines. Cette fonctionnalité est assurée dans les clusters en créant des répliques d'un EJB sur plusieurs serveurs dans le cluster.

Les clusters sont présentés par les fournisseurs de plates-formes J2EE comme une solution aux problèmes de capacité de croissance et de disponibilité. Un cluster est un groupe de machines qui travaillent ensemble afin d'assurer la capacité de croissance et la disponibilité des applications J2EE. Cette définition est un peu vague car chaque vendeur présente une définition propre à lui selon la manière dont il implante sa technique de clustering [ABR02, TYL02]. Jgroup [MEL01, MEL02] est un projet de recherche qui présente un modèle de clustering qui vise les objets distribués basés sur RMI. Ils proposent une architecture qui permet la réplication des objets distribués sur plusieurs serveurs toute en garantissant la consistance entre les objets répliqués.

Quelle que soit l'implantation, les deux problèmes sont traités d'une manière centralisée ; un seul serveur d'administration gère l'ensemble des serveurs. Au moment du déploiement plusieurs répliques de l'EJB sont déployées sur plusieurs serveurs d'applications. Si un serveur d'applications tombe en panne, les serveurs d'applications qui possèdent les répliques de ces EJB prennent le relais.

2.1.2. Les plates-formes d'administration

Après avoir présenté les systèmes à base de composants distribués et la plate-forme J2EE qui constitue l'environnement que nous cherchons à administrer dans cette thèse, nous introduisons dans cette section les plates-formes d'administration. Nous présentons l'évolution des standards d'administration. Nous détaillons ensuite JMX, le standard d'administration des applications java que nous adoptons dans cette thèse.

2.1.2.1. Evolution des standards d'administration

Les premiers périphériques réseaux à administrer (« port », « convertiseur de protocoles », et « routeur ») se résumaient à de simples machines administrées directement par des liaisons séries. Avec l'explosion des réseaux en taille et en technologie, le nombre d'éléments du réseau est devenu plus important et il est devenu nécessaire de trouver une solution pour contrôler et superviser ces machines. Certains vendeurs fournissent un système d'administration hors bande pour administrer toutes les machines du même réseau local. D'autres utilisent une approche sur le réseau (in-band technologie) où des informations de contrôle et d'adressage sont envoyées dans le réseau local selon un protocole spécifique de gestion. Les logiciels clients de gestion ont évolués du terminal en mode texte, à des logiciels graphiques. Les premiers systèmes d'administration sont appelés des NMS (Network Management Systems) [SIN02]. Les protocoles de gestion comme SNMP (Simple Network Management Protocol) [CAS90] et CMIP (Common Management Information Protocol) [ISO91] sont apparus pour standardiser l'échange des informations de gestion.

Avec l'évolution de l'Internet et de l'E-commerce des solutions pour une gestion globale et standardisée, qui permet le contrôle de la partie logicielle ainsi que de la partie matérielle, sont devenues une nécessité. Le protocole WBEM (Web Based Enterprise Management) [WBEM] et le modèle d'information CIM (Common Information Model) [CIM03, CIM04] pour la modélisation des informations de gestion sont les deux innovations dans le monde de gestion à grande échelle. Ils ont été standardisés par le DMTF (Distributed Management Task Force).

D'autres standards de gestion ont été présentés par d'autres organisations pour la gestion des applications. Par exemple, Le « Open Group Enterprise Management Program » a récemment publié deux API de gestion AIC (Application Instrumentation and Control) [OPEb04] et ARM (Application Response Measurement) [OPEa04]. AIC est une API développée en C et en java pour exposer les métriques et les seuils d'applications. ARM est une méthode pour capturer le temps nécessaire pour accomplir une unité de travail à l'intérieur d'une application. AIC et ARM sont tous les deux des outils d'instrumentation mais ils ne présentent pas une plate-forme d'instrumentation.

Dans le monde Java, on trouve la spécification de JMX (Java Management eXtensions) [JMX02, JMX03] ; un ensemble de services et d'API (Application Programming Interface) qui forment une base d'administration pour les applications java. JMX a été adopté par la plupart des serveurs d'applications pour l'administration des services J2EE.

La majorité des systèmes d'administration [MAR98, TAY96] suit le modèle Manager/Agent. Ce modèle définit trois éléments de base :

- ***Objet administrable*** : est une abstraction d'une ressource du système comme un périphérique, un service, ou une application. Cette abstraction expose les informations de gestion dans la forme d'interfaces accessibles aux ***Agents***.
- ***Agent*** : est l'élément responsable de l'accès aux ressources à gérer. Il enregistre une ou plusieurs ressources à gérer, et il expose leurs attributs et leurs méthodes aux applications de gestion**.** L'agent offre également des services de gestion aux managers comme le service de « planification de tâches », « événement », etc.
- ***Manager*** : est l'élément responsable de l'application des tâches de gestion. Il communique avec l'***Agent*** pour demander des informations de gestion ou l'exécution l'une des méthodes de **l'objet administrable**.

A la suite de cette section nous présentons les détails de JMX, la plate-forme que nous adoptons pour l'administration.

2.1.2.2. JMX : plate-forme d'administration d'applications Java

JMX (Java Management eXtension) définit une architecture et les services permettant d'administrer des réseaux et des applications JAVA [KRE01, HAN04]. Comme la majorité des systèmes d'administration, JMX suit le modèle Manager/Agent.

En JMX, l'objet administrable est appelé le MBean (Managed Bean). Pour être administrable, les ressources logicielles ou matérielles doivent être instrumentées sous la forme de composants MBean. Ces composants fournissent une interface d'administration et l'instrumentation nécessaire permettant de contrôler les ressources associées. L'interface d'administration définie par JMX doit permettre aux vendeurs de composants de produire des composants administrables et aux vendeurs d'outils d'administration d'utiliser cette interface pour administrer ces composants.

L'architecture JMX est divisée en trois couches (Figure 2.3):

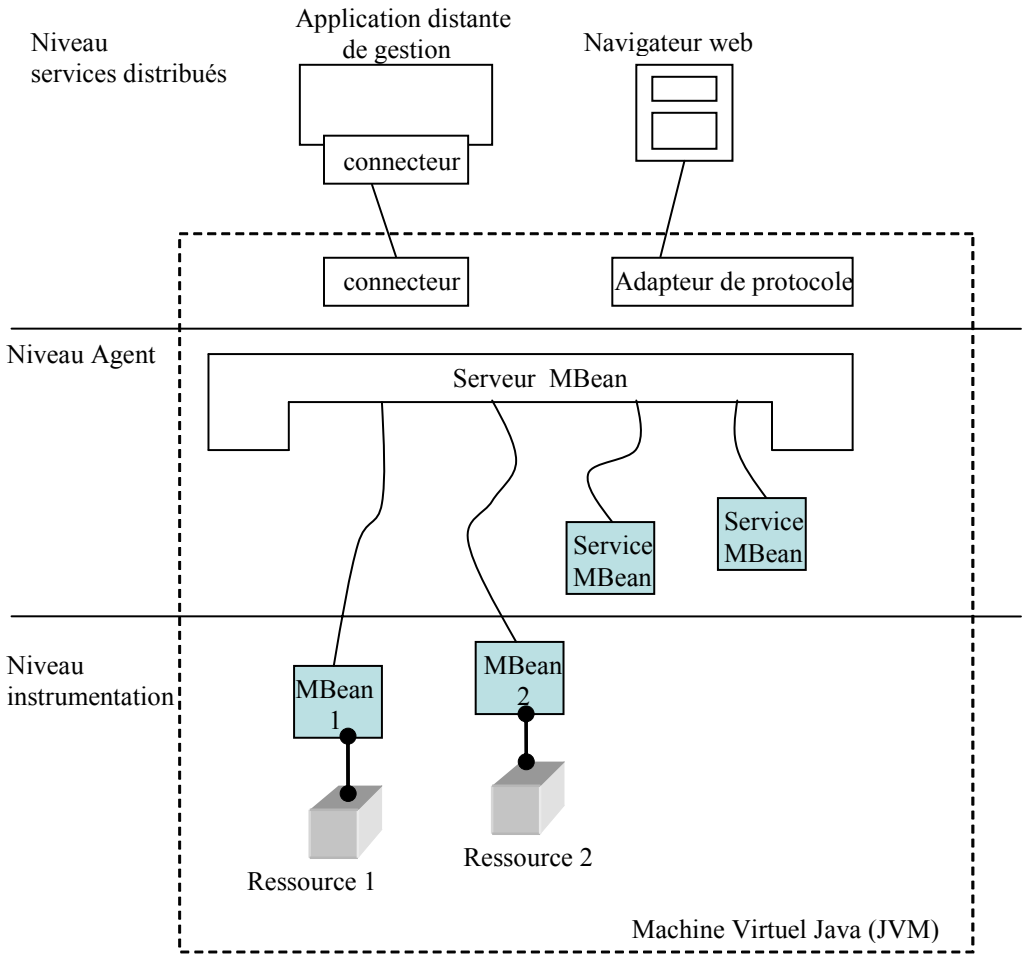

Figure 2.3 Architecture de JMX

- Le niveau **instrumentation** : Ce niveau correspond à l'instrumentation des ressources sous la forme de MBeans. Un MBean contient la logique d'administration permettant de contrôler la ressource associée au travers de l'interface d'administration.
- Le niveau **agent** : Les agents d'administration contrôlent les MBeans et les rendent accessibles aux applications d'administration distantes. Chaque agent a la responsabilité d'un ensemble de MBeans. Toutes les opérations d'administration requises sur un MBean doivent passer par son agent.
- Le niveau **service distribué** : Ce niveau spécifie des composants particuliers, appelés connecteurs ou adaptateurs de protocole, permettant aux applications d'administration d'accéder à distance aux agents JMX et de prendre en compte divers protocoles de communication.

Niveau instrumentation et les MBeans

Les ressources logicielles ou matérielles sont instrumentées au travers des MBeans. Cette instrumentation a pour rôle :

- D'abstraire l'état d'une ressource et de permettre de manipuler cet état,
- D'émettre des notifications lorsque certains événements associés à la ressource se produisent.

La granularité des ressources instrumentées sous la forme d'un MBean est établie par son programmeur. Un MBean peut représenter un objet appartenant à une application ou bien l'application dans sa globalité. Un MBean est un objet JAVA qui met en oeuvre une interface d'administration et qui se conforme à certaines règles de conception *(Design Pattern)*. Les attributs et méthodes d'un MBean représentent l'état de la ressource associée et les différentes opérations de contrôle permettant de l'administrer. L'interface d'administration d'un MBean représente :

- L'ensemble des attributs qui peuvent être lus ou modifiés,
- L'ensemble des méthodes permettant de déclencher des opérations d'administration sur la ressource associée (par exemple une opération de réinitialisation),
- L'ensemble des notifications qui peuvent être émises par la ressource. Le modèle de notification utilisé par JMX est proche du modèle de notification des Java Beans.

Les MBeans peuvent être construits sous la forme de MBeans standards (*Standard MBean*) , MBeans Dynamiques (*Dynamic MBean*), MBeans Ouverts (*Open MBean*), ou MBeans modèles (*Model MBean*). Les MBeans standards se basent sur une interface d'administration fixée au moment de la compilation. Les attributs et les méthodes d'administration associés à une ressource sont déterminés de manière statique. Inversement, les MBeans dynamiques permettent de déterminer en cours d'exécution les attributs et les opérations d'administration associés à une ressource. Les MBeans ouverts et les MBeans modèles sont tous les deux des MBeans dynamiques.

Dans la suite de cette section nous détaillerons les types des MBean.

- **MBean standard :**

Un MBean standard est un objet Java qui doit mettre en oeuvre une interface d'administration spécifique. La définition de cette interface d'administration s'effectue en suivant des règles de conception issues du modèle Java Bean :

- Le nom de l'interface correspond au nom de la classe du MBean suffixé par la chaîne "MBean". Ainsi le nom de l'interface d'administration correspondant au MBean MyClass serait MyClassMBean.
- Pour chaque attribut public représentant la ressource administrée, il doit exister une méthode de type *get* et/ou *set* permettant respectivement de lire ou de modifier cet attribut. Le nom de ces méthodes dépend du nom de l'attribut associé. La règle de déclaration de ces méthodes est la suivante :

```
public AttributeType getAttributeName();
public void setAttributeName(AttributeType value);
```

- Pour toute opération d'administration de la ressource, il doit exister une méthode correspondante dans l'interface d'administration du MBean.

L'exemple suivant définit un MBean de classe "MyClass". Ce MBean possède un attribut d'administration de type entier appelé *state*. Cet attribut peut être lu et modifié. De plus, notre MBean met en oeuvre une méthode (*reset)* permettant de réinitialiser le MBean. L'interface d'administration et le code de la classe du MBean sont les suivants :

```
public interface ResourceClassMBean{
     public Integer getState();
     public void setState(Integer s);
     public void reset();
}
public class ResourceClass implements MyClassMBean{
     private Integer state = null;
     public Integer getState() { return(this.state); }
     public void setState(Integer s) { this.state = s; }
     public void reset() { this.state = null; }
     …
}
```

- **MBean dynamique :**

Un MBean dynamique est un objet JAVA classique qui doit mettre en oeuvre une interface prédéfinie appelée *DynamicMBean.* Cette interface permet de découvrir les attributs et les opérations d'administration associés au MBean en cours d'exécution. Au lieu d'exposer directement ses opérations au travers d'une interface d'administration, le MBean dynamique fournit une méthode permettant d'obtenir la

liste de tous les attributs et la signature des méthodes associées. Figure 2.4 montre le diagramme UML de l'interface *DynamicMBean.*

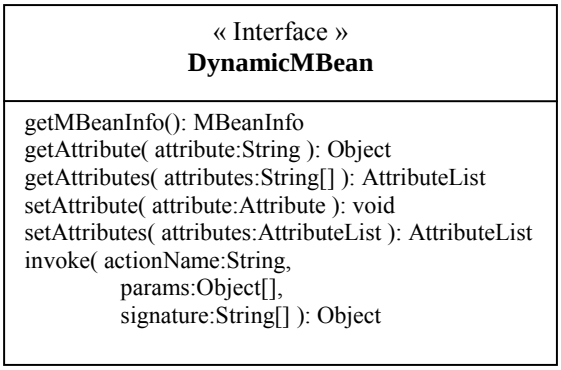

Figure 2.4 Diagramme UML de l'interface DynamicMBean

- La méthode ***getMBeanInfo*** retourne un objet de type MBeanInfo contenant la définition des attributs et des méthodes d'administration du MBean.
- Les méthodes de type ***getAttribute*** permettent de lire la valeur des attributs dont le nom est passé en paramètre.
- Les méthodes de type ***setAttribute*** permettent d'écrire la valeur des attributs dont le nom et la valeur sont passés en paramètre.
- La méthode ***Invoke*** permet d'exécuter la méthode dont le nom est passé en paramètre avec les paramètres correspondants. Le paramètre signature est un tableau contenant le type des objets passés en paramètre lors de l'appel de la méthode. Le paramètre *params* est un tableau contenant la valeur des objets passés en paramètre.

- **MBean Open :**

Un MBean Open est un MBean dynamique appelé Open car il dépend d'un petit ensemble prédéfini des types de données universels (int, String,..). Les applications de gestion et les MBeans Open sont alors capables de partager et d'utiliser les données de gestion et les opérations à l'exécution sans échange de types structurés complexes.

Ce type de MBean est utile quand les applications de gestion n'ont pas accès aux classes Java spécifiques de l'agent responsable de la ressource. L'agent et l'application de gestion peuvent alors intéroperer sans partager des classes spécifiques.

- **MBean Model :**

Un MBean model est un MBean générique et configurable qui peut être utilisé pour instrumenter un type de ressource spécifique rapidement (une unité de stockage, un clavier, ..). Le MBean model est aussi un MBean dynamique. Le but de ce type de MBean est de soulager les développeurs d'avoir à écrire un MBean spécifique pour chaque catégorie ressource. Pour gérer une ressource, le développeur peut simplement créer une instance MBean model et la configurer.

Niveau Agent

Dans l'architecture JMX, toutes les opérations d'administration passent par le biais des agents. Ces agents permettent aux applications d'administration de contrôler à distance des MBeans en :

- Lisant ou en modifiant les attributs du MBean
- Appelant des opérations définies par l'interface du MBean.
- Recevant des notifications émises par le MBean
- Utilisant certains services fournis par l'agent.

Un agent est composé d'un serveur de MBean et d'un ensemble des services de gestion comme le service de téléchargement dynamique "m-let" (dynamique loading) et le service de surveillance.

- **Le serveur de MBean :**

Le serveur de MBeans (MBean Server) est l'élément central d'un agent. C'est un référentiel qui maintient une liste de tous les MBeans contrôlés par l'agent. Lorsqu'une requête arrive à destination de l'agent, le serveur de MBeans recherche le MBean destinataire de la requête et exécute les opérations d'administration correspondantes. Toutes les opérations d'administration appliquées sur un MBean sont effectuées à partir de l'interface MBeanServer du serveur de MBean. Cette interface définit des méthodes permettant :

- D'instancier et d'enregistrer un nouveau MBean dans l'agent. On peut noter que le concepteur d'un MBean peut contrôler la manière dont un MBean est enregistré dans l'agent (ou retiré de l'agent). Un MBean peut définir des méthodes qui seront appelées avant et après son enregistrement (ou son retrait).
- D'accéder aux attributs du MBean et d'invoquer des opérations dessus.

Le serveur de MBeans centralise l'accès à tous les MBeans : il décharge les objets de devoir maintenir plusieurs références sur les MBeans. Afin d'assurer cette fonctionnalité, le serveur de MBeans utilise des objets de nommage pour identifier

les instances des MBeans d'une manière unique. La même classe MBean peut avoir plusieurs instances, mais chaque instance a un nom unique. La classe **ObjectName** encapsule un objet nom, composé d'un nom de domaine et d'un ensemble des propriétés. L'objet nom est representé comme un String avec le format suivant :

DomainName : property=value [, property2=value2] *

Le DomainName et les paires (propriété=clé) peuvent être n'importe qu'elle chaîne de caractères alphanumérique.

Un domaine est une catégorie abstraite utilisée pour grouper les MBeans d'une manière arbitraire. Par exemple, tous les MBeans de connectivité peuvent avoir le domain *Communications*. Puisque tous les objets doivent avoir un nom de domaine, le MBean serveur fournit un domaine par défaut appelé *DefaultDomain*.

Une paire propriété=valeur peut prendre n'importe quelle signification choisie par le développeur. Un objet ObjectName doit avoir au moins une paire clé. Les clés et leurs valeurs sont indépendants des attributs du MBean : le nom de l'objet est un identificateur statique qui identifie le MBean, tandis que les attributs sont les valeurs exposées au moment de l'exécution (runtime) d'une ressource. L'ordre des clés n'a pas d'importance. Un exemple d'un objet ObjectName qui identifie un adapteur du protocole HTML :

Communications :protocol=html,port=8082 (en considérant que le numéro du port ne sera pas changé).

Les autres fonctionnalités des agents sont fournies sous la forme de services (mis en oeuvre comme des MBean).

Parmi les services standards fournis par un agent, on peut citer les services de téléchargement et de surveillance. JMX permet aussi l'intégration d'autres services.

- **Le service de téléchargement (m-let service)** :

Ce service permet d'instancier et de télécharger de nouvelles classes de MBeans à partir du réseau. Ce service utilise un fichier XML contenant les informations pour le téléchargement des MBeans. La localisation de ce fichier est spécifiée à l'aide d'une URL. Lorsqu'un tel fichier est chargé, toutes les classes spécifiées sont téléchargées et une instance de chaque MBean déclaré dans le fichier est créée et enregistrée. Comme ce service est mis en oeuvre sous la forme d'un MBean, il est accessible aux autres MBeans, aux agents ou bien encore aux applications d'administration.

- **Le service de surveillance** :

Ce service permet d'observer les changements de valeurs des attributs d'un MBean. A intervalle régulier, ce service effectue des échantillonnages de certains attributs d'un MBean. Cet échantillonnage entraîne l'évaluation d'une condition associée aux attributs. Lorsqu'une condition est vraie, une notification est émise à tous les abonnés. Ces conditions sont liées à des classes de moniteurs particulières. Par exemple, le moniteur de type compteur défini par JMX émet une notification lorsque l'attribut associé atteint ou dépasse un seuil.

Niveau services distribués

Les connecteurs et les adaptateurs de protocole rendent les agents accessibles à distance. Les applications d'administration les utilisent pour accéder aux serveurs de MBeans. Malheureusement, ils ne sont pas décrits dans l'état actuel des spécifications de JMX. Un connecteur est utilisé pour connecter un agent à une application d'administration via le protocole d'administration de JMX. De manière analogue, un adaptateur de protocole est utilisé pour connecter un agent et une application d'administration via un protocole d'administration différent de JMX comme, par exemple, SNMP. Un adaptateur de protocole SNMP fournit une vision SNMP d'un agent.

Modèle de notification de JMX

L'interface d'administration des MBeans permet aux agents d'effectuer des opérations de configuration et de contrôle sur les ressources administrées. Cependant les applications d'administration ont souvent besoin de réagir à des changements d'état ou à d'autres événements se produisant sur les ressources. Le modèle de notification JMX définit un mécanisme permettant aux MBeans de diffuser ces événements (appelés notifications). Ce modèle est de type publish/subscribe (i.e. les objets désirant recevoir un type de notification particulier doivent s'y abonner). Lorsqu'un événement se produit, le producteur de l'événement prévient tous les objets abonnés à ce type d'événement. Le modèle de notification JMX définit principalement les éléments suivants :

- Une classe de base d'événement (Notification) permettant de signaler l'occurrence d'un événement. Cette classe peut être sous-classée pour identifier un événement particulier.
- Une interface (NotificationBroadcaster) devant être mise en oeuvre par les MBeans désirant émettre des événements. Cette interface définit principalement deux méthodes (addNotificationListener et removeNotificationListener) permettant respectivement à un objet de s'abonner ou de se désabonner à un événement particulier sur le broadcast. Lorsqu'un événement se produit, le

MBean appelle la méthode handleNotification sur tous les objets abonnés à cet événement.

- Une interface (NotificationListener) devant être mise en oeuvre par les objets désireux de recevoir des notifications issues des MBean. Elle définit la méthode (HandleNotification) appelée par un MBean lorsqu'il émet une notification.

2.1.3. Les plates-formes d'instrumentation

Dans le domaine de l'administration on trouve souvent le terme *instrumentation* utilisé sans avoir une définition claire de sa signification.

L'instrumentation est un outil permettant de faire de l'administration. Dans [ALMa02] nous avons défini l'instrumentation comme étant le processus fournissant une interface décrivant les informations nécessaires pour la gestion d'un élément. Contrairement à une plate-forme d'administration, une plate-forme d'instrumentation ne fournit pas de services de gestion. Son but est plutôt d'extraire les informations, qui seront utiles pour la gestion.

Dans la suite de cette section nous présentons une plate-forme d'instrumentation (NWS) qui vise à instrumenter un système Unix.

2.1.3.1. NWS

NWS (Network Weather Service) [RIC99] est un projet d'un group de travail de l'université de Californie et l'université de Tennessee aux USA. C'est un système distribué d'instrumentation et de prédiction de performances basé sur l'historique des mesures de performance de l'ensemble des ressources d'un réseau. Les mesures effectuées par NWS sont principalement des mesures de bas niveau concernant la CPU et le réseau. NWS a été implanté en C sous Unix.

Les composants de NWS sont des processus. Il y a 4 types de processus :

- Le processus de stockage (Persistent State process) : Il stock les mesures sur un support de stockage persistent.
- Processus de nommage (Name Server Process) : Il implante un répertoire à utiliser pour relier les noms de processus avec les informations de contact de bas niveau (ex : adresse TCP/IP et numéro de port)
- Sensor : C'est le processus qui collectionne les mesures de performance des ressources. Il emploie un ou plusieurs sondes pour faire des mesures spécifiques.
- Prévisionniste : C'est le processus qui produit des valeurs de prédiction de la performance durant un intervalle de temps précis pour une ressource précise.

Tous les composants communiquent avec des messages fortement typés. Les composants, sauf le processus de nommage, peuvent être distribués à travers plusieurs stations de travail. Plusieurs sensors distribués collectent les mesures de performance, ils les traitent et envoient les résultats aux processus de stockage. Chaque sensor est inscrit auprès d'un seul processus de stockage. Un seul processus de stockage contient les mesures d'un ou plusieurs sensors. Le processus de nommage spécifie l'emplacement de tous les processus du système NWS. Le prévisionniste, qui répond aux requêtes clients pour prédire la performance d'une ressource, il consulte le processus de stockage concernant la ressource demandée par la requête et il produit les valeurs de prédictions demandées.

NWS mesure principalement les activités de la CPU de chaque station de travail du réseau et la performance réseau. Les mesures principales sont les suivantes :

- La fraction du temps disponible pour un nouveau processus : le sensor CPU utilise les outils système de l'Unix *uptime* et *vmstat* afin de mesurer la disponibilité de la CPU.
- La latence du réseau : calculé par le transfert (aller-retour /round-trip) de 4 octets via une socket TCP d'un sensor source à un sensor destination.
- La bande passante du réseau : calculé par l'envoi et la réception d'un accusé de réception TCP après le transfert d'un message de 64K-octet en utilisant 32 octets.
- Le temps de connexion TCP

Chacune de ces mesures est associée à un intervalle de temps. Les sensors envoient les mesures aux processus de stockage sous forme de paire d'informations (intervalle de temps – mesure de performance).

2.1.4. Conclusion

Dans la section 2.1 nous avons présenté le contexte de notre travail. En premier place, nous avons montré l'environnement d'exécution que nous considérons pour notre recherche, notamment la plate-forme J2EE et les composants EJB. Ensuite, nous avons montré l'architecture de JMX, la plate-forme d'administration que nous adaptons pour développer des fonctionnalités d'administration pour les applications EJB. Finalement, nous avons précisé la notion de plate-forme d'instrumentation en présentant NWS.

2.2. Administration des systèmes à composants distribués

Dans cette section nous présentons un état de l'art des systèmes d'administration autour des systèmes distribués. Nous présentons en premier lieu les systèmes d'administration permettant d'administrer des systèmes distribués pris dans leur plus grande généralité. Dans la seconde partie, nous présentons les systèmes d'administrations orientés autour des systèmes distribués à base de composants.

2.2.1. Administration des systèmes distribués

Dans cette section nous présentons deux travaux de recherche qui visent l'administration des systèmes distribués spécifiques : MANDAS et ORYA.

2.2.1.1. MANDAS

MANDAS [BAU95, [KAT99] est un projet de l'université du Western Ontario au Canada. Il définit une architecture pour l'administration d'applications réparties. Pour MANDAS une application répartie consiste en un ensemble de processus coopérant sur des réseaux. Dans ce contexte la notion de composant correspond aux divers processus de l'application. Cette architecture est dérivée de l'architecture proposée par l'OSI pour l'administration de réseaux (CMIP). Le coeur de MANDAS est basé sur un service de surveillance, de contrôle et sur un référentiel conservant le modèle de l'application. Les opérations d'administration élémentaires sont fournies au travers de l'instrumentation des processus applicatifs.

L'architecture de MANDAS est organisée en quatre couches : les applications d'administration, les services d'administration, les agents d'administration et les objets administrés.

- Les applications d'administration sont utilisées par les administrateurs pour effectuer les taches d'administration telles que la configuration du système, l'analyse de performance etc...
- Les services d'administration fournissent les services permettant de contrôler une application.
- Les agents d'administration mettent en oeuvre la logique d'administration. Ils peuvent surveiller des objets administrés et les contrôler.
- Les objets administrés représentent des abstractions sur des ressources logicielles ou matérielles (i.e. processus ou machine). Ils fournissent une interface d'administration permettant aux agents de les manipuler.

Pour gérer des applications avec MANDAS, il faut que la gestion soit considérée dès le développement de l'application afin de pouvoir l'instrumenter. MANDAS permet de décrire une application distribuée sous la forme de l'ensemble des processus qui participent à l'application et de leurs dépendances. La sémantique d'une dépendance entre deux processus exprime simplement que le premier processus a besoin du deuxième pour s'exécuter. MANDAS nous présente une architecture d'administration classique qui est constituée des briques de bases permettant de contrôler l'exécution d'une application répartie. Ces briques sont les services de configuration, de surveillance et de reconfiguration ainsi qu'un service contenant une représentation persistante de l'application en cours d'exécution appelé "référentiel". Ce référentiel maintient à jour le modèle de l'architecture de l'application en fonction des créations et des destructions de processus et stocke les mesures de performance issues des différents senseurs de l'application. De plus MANDAS définit le langage de script QDL (Query and Directive Language) permettant d'interagir facilement avec les services d'administration. QDL est un langage d'interrogation assez complet permettant de construire des requêtes d'interrogation très complexes. QDL permet aux administrateurs d'obtenir dynamiquement des informations sur l'application en cours d'exécution.

2.2.1.2. Environnement de déploiement : ORYA

ORYA est un prototype d'un environnement de déploiement proposé dans une thèse de doctorat préparée au sein du LISTIC et du LSR/ADELE [MER04, LES02]. L'objectif de la thèse est de proposer une plate-forme permettant d'offrir un support automatisé aux activités de gestion du cycle de vie du déploiement. Le déploiement d'un logiciel consiste dans les processus suivantes : la sélection des composants, l'installation, le mise à jour et la désinstallation.

La plate-forme proposée est utilisée comme un moyen de gérer l'installation des logiciels sur un parc informatique de milliers de machines appartenants à une entreprise de taille moyenne. Les logiciels à installer sont recherchés sur les sites des producteurs, puis ils sont transférés sur un serveur de déploiement central dans l'entreprise. Ce serveur s'occupe de l'installation à distance des logiciels sur les différentes machines de l'entreprise.

Dans le cadre de cette thèse, [LES03] propose un modèle de procédés qui décrit les processus et les informations de déploiement comme les noms des logiciels à installer, les sites des producteurs, les outils à utiliser pour le transfert et pour l'installation, et la politique de déploiement de l'entreprise (les logiciels autorisés, les utilisateurs autorisés, etc.). Pour chaque installation, le serveur de déploiement applique un procédé de déploiement précis.

2.2.2. Administration des systèmes distribués à base de composants

Dans cette section nous présentons plusieurs projets qui visent l'administration des systèmes distribués orientés composants. Le premier projet vise l'administration des systèmes CORBA et DCOM, les deux derniers visent l'administration des systèmes EJB.

2.2.2.1. MIMO (CORBA et DCOM)

MIMO (MIddleware MOnitoring) [RAC00] est un système de monitoring et d'administration pour les environnements distribués hétérogènes. Le système a été développé par un groupe de travail universitaire en Allemagne (Technische Universität München). MIMO vise principalement les deux plates-formes de composants CORBA et DCOM.

MIMO se base sur une architecture à plusieurs niveaux de surveillance. MIMO considère six niveaux d'abstraction représentant le système sous-jacent à surveiller :

- Niveau application : le plus haut niveau dans le système de surveillance.
- Niveau interface : c'est le niveau qui s'intéresse aux interfaces IDL qui décrivent les fonctionnalités des composants d'une application.
- Niveau composants : ce niveau concerne les objets CORBA ou DCOM qui implantent les fonctionnalités décrites par les interfaces.
- Niveau middleware : ce niveau concerne les objets, les services et les APIs que fournissent les composants avec les tâches non fonctionnelles.
- Niveau environnement d'exécution (run-time environment) : ce niveau s'adresse au système d'exploitation ou à une machine virtuelle au-dessus du système d'exploitation.
- Niveau hardware : c'est le niveau qui décrit les nœuds sur lesquels l'ensemble des applications tournent.

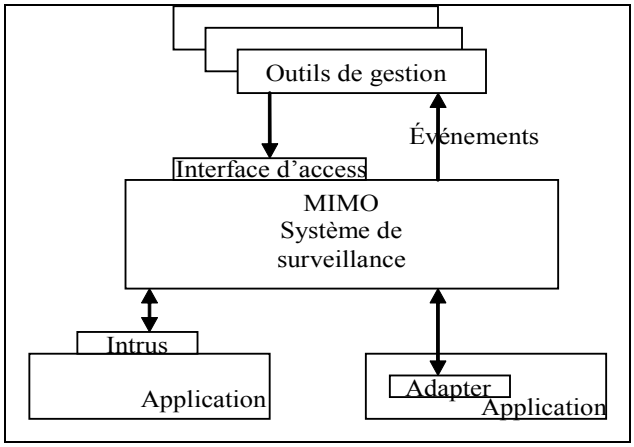

Figure 2.5 L'architecture du système de supervision à plusieurs niveaux

Ces niveaux représentent une abstraction de l'ensemble du système à surveiller. Les outils de gestion contactent le système de surveillance (MIMO) à travers une interface d'accès. Le système de surveillance de son côté collectionne les informations des applications à surveiller à l'aide des *intrus* et des *adapters*. La difference entre les intrus et les adapters est que les intrus sont intégrés de façon transparente dans les applications, tandis que les adapters sont intégrés en ajoutant du code aux applications. A l'aide des intrus et des adapters le superviseur MIMO rassemble les informations nécessaires pour la gestion. Les applications de gestion, appelées *outils de gestion*, contactent le superviseur à l'aide de l'interface *tool-monitor*.

Le système est implanté sur CORBA. Les points principaux du système sont:

- Les outils de gestion récupèrent l'IOR (Inter Operable Reference) de l'interface MIMO requise.
- L'interface *tool-monitor* fournit des méthodes pour s'attacher ou détacher de MIMO, et pour démarrer ou arrêter les requêtes.
- Les *intrus* et les *adapters* communiquent avec MIMO via deux canaux d'événements CORBA.
- Les *adapters* sont codés à la main par le développeur à condition qu'il puisse modifier l'application à surveiller. Les intrus par contre sont générés automatiquement. Ils emballent les méthodes de la librairie CORBA utilisées par l'application. Les méthodes emballées sont : la méthode d'initialisation, les méthodes de création et de suppression et la méthode d'invocation.

2.2.2.2. MKBEEM (EJB)

MKBEEM (Multilingual Knowledge-based European Electronic Marketplace) [MKBEEM] est un projet européen qui vise la gestion d'un système E-commerce basé sur les EJBs [LOP01, ASE01]. L'architecture de gestion proposée par ce projet se base sur JMX et l'encapsulation des EJBs en composants MBeans de type model.

L'architecture se base sur l'emballage des EJB dans des classes spécifiques aux MBeans de type model. Cet emballage est fait avant le déploiement de l'EJB. L'emballage consiste à refaire un EJB de gestion qui hérite de l'EJB d'origine. Le déploiement se fait ensuite pour l'EJB de gestion.

Depuis une autre machine virtuelle, un serveur JMX s'exécute pour instrumenter les EJB de gestion. Un MBean modèle est créé pour chaque EJB de gestion côté serveur d'applications. La communication entre un EJB de gestion et son MBean se fait avec des notifications RMI-IIOP.

Le mécanisme de notification consiste à créer un serveur de notification qui reçoit de la part des EJBs de gestion des notifications de type remote en utilisant RMI-IIOP.

2.2.2.3. EJBMonitor (EJB)

Un groupe de recherche en Irlande (Dublin City University) a développé un superviseur pour les composants EJB [ADR01].

L'architecture se base sur l'intégration d'une application espionne "spy-application" dans le serveur d'application EJB. Cette application a pour but de déployer un proxy EJB pour chaque EJB d'une application que l'on veut superviser. Le proxy EJB *vole* l'identité de l'EJB d'origine et se charge de recevoir les requêtes client à sa place. L'application espionne procède comme suit :

- Créer un proxy EJB avec le même nom et identité de l'EJB d'origine
- Donner un nom différent pour l'EJB d'origine afin qu'il ne soit reconnu que par le proxy EJB

A partir du moment où le proxy EJB a pris le nom de l'EJB d'origine il commence à recevoir les appels client à la place de l'EJB d'origine. Le proxy envoie une notification d'invocation à chaque fois qu'il reçoit une invocation d'une méthode. Ensuite, il dirige l'appel vers l'EJB d'origine. La Figure 2.6 montre la séquence d'opérations suite à un appel client.

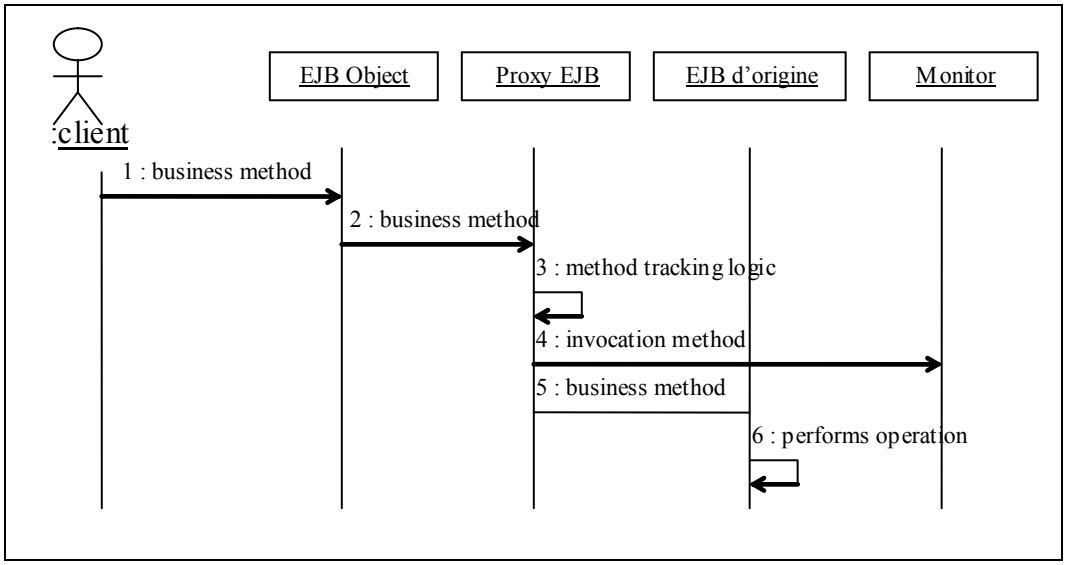

Figure 2.6 La séquence d'opérations d'un appel supervise

2.3. Commentaires

Dans ce chapitre nous avons présenté l'état de l'art des deux axes qui constituent le cadre de la thèse : les systèmes distribués et les systèmes d'administrations.

Nous avons ensuite présenté quelques travaux qui visaient l'administration des systèmes distribués à base de composants. Dans un premier temps, nous avons présenté les travaux sur l'administration des systèmes distribués comme MANDAS et ORYA. Dans un deuxième temps, nous avons présenté les travaux qui visaient l'administration des systèmes distribués orientés composants comme MIMO, MKBEEM et EJBMonitor.

MANDAS présente une architecture qui vise l'administration et l'instrumentation des systèmes distribués. L'architecture suit le modèle de gestion du réseau pour la gestion d'applications. Il n'adresse donc pas les problèmes liés à la gestion des applications. Il ne permet pas de modéliser l'architecture de l'application à différents niveaux de granularité, il n'y a pas de mécanisme de composition. Ces différents niveaux sont pourtant fondamentaux pour gérer la complexité d'une application.

ORYA ne représente pas une plate-forme d'administration des applications distribuées mais il vise l'administration du déploiement d'applications dans les parcs informatiques d'entreprises. Les phases de déploiement considérées dans ORYA ne

prennent pas en compte l'administration des applications en cours d'exécution. Ceci limite l'utilisation d'ORAY à des contextes précis.

L'architecture présentée par MIMO est très bien adaptée pour les systèmes à composants, mais il ne fait que l'instrumentation des plates-formes DCOM et CORBA et il n'offre pas de services d'administration.

Concernant la plate-forme J2EE, MKBEEM ne traite pas l'administration globale du système, mais il propose simplement un moyen d'instrumenter les composants EJB. Les autres niveaux de l'environnement (middleware, hardware, réseau) ne sont pas considérés. Tout comme MKBEEM, EJBMonitor permet de surveiller seulement les composants EJB en les espionnant par un proxy EJB. C'est à l'administrateur de réagir manuellement en fonction des événements générés par l'EJBMonitor.

Chacune de ces études a ses propres stratégies et protocoles d'administration. Malgré l'existence de plusieurs standards et protocoles d'administrations qui ont été présentés dans la section 2.1.2, réutilisent ces standards d'administration pour gérer un système distribué. L'utilisation de ces standards est encore limitée à l'administration des réseaux.

Nous pensons qu'il est impératif de passer par ces standards d'administration afin que l'administration soit unifiée et que les systèmes administrés soient intégrables dans d'autres systèmes d'administration.

Dans notre approche nous cherchons à administrer une fédération de serveurs d'applications hétérogènes. La standardisation est donc un élément vital pour l'administration de la fédération. De plus, dans le contexte d'une fédération il n'est pas suffisant de se contenter des services d'administration isolés. L'administration elle-même doit être distribuée. D'autre part, dans le cadre d'une fédération, il faut chercher à automatiser les décisions de gestion et ne pas laisser toute la responsabilité à un administrateur.

Dans le chapitre suivant nous introduisons notre approche d'administration pour un système distribué à base des composants EJB. Nous basons notre proposition sur les standards d'administration présentés dans ce chapitre, notamment JMX.

3. Problématique et proposition

Dans ce chapitre nous présentons la problématique traitée dans cette thèse. Nous mettons l'accent sur les points faibles de la distribution et du déploiement dans la plate-forme J2EE, et nous en déduisons la nécessité d'une architecture plus coopérative entre les serveurs d'applications. Nous présentons ensuite la notion de fédération de serveurs d'applications que nous avons définie pour créer un environnement plus adaptable et plus robuste pour l'exécution d'applications EJB. Finalement nous discutons notre proposition dans le cadre de l'administration du déploiement dans la fédération.

3.1. Limitation de la plate-forme J2EE

3.1.1. Vision de la distribution actuelle dans la plate-forme J2EE

La distribution des applications peut être réalisée d'une manière verticale ou horizontale (Figure 3.1). Une application distribuée horizontalement est une application décomposée sur plusieurs machines représentant les différentes couches d'un système multi-tiers. L'application est décomposée selon les tâches à réaliser : une partie pour la présentation graphique, une partie pour le code fonctionnel, et une partie pour la gestion des données. La distribution verticale par contre est la vision traditionnelle des systèmes distribués. Dans cette vision la distribution est effectuée dans une seule couche où plusieurs tâches du même ordre peuvent être distribuées entre plusieurs machines.

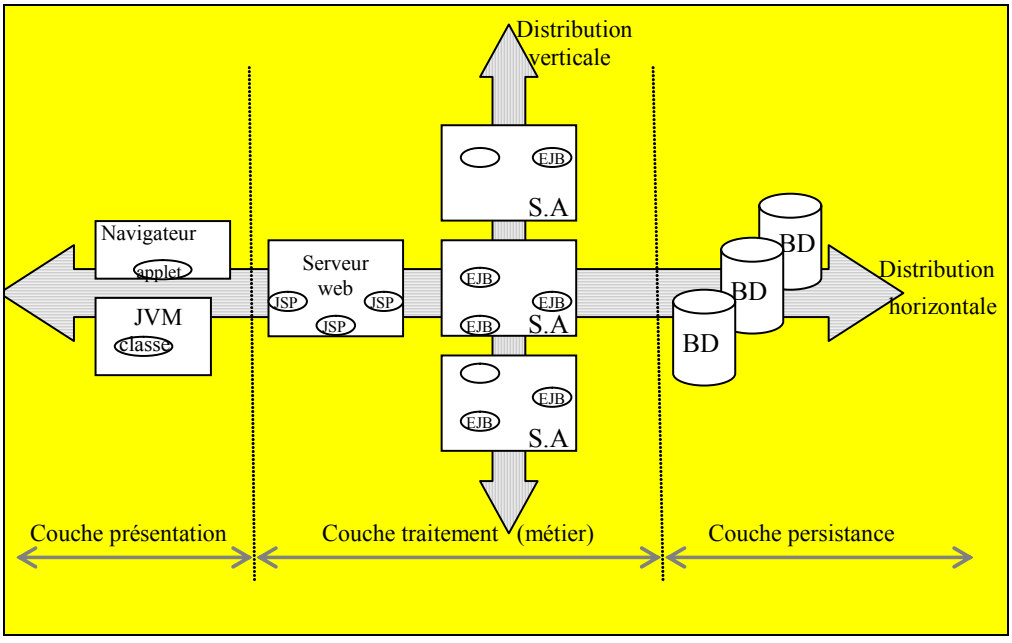

Figure 3.1 La distribution dans la plate-forme J2EE

Dans l'approche J2EE l'application est répartie sur plusieurs couches d'une manière horizontale. La partie présentation de l'application peut être une application java qui tourne sur une machine virtuelle, elle peut être également une page dans un navigateur web qui contacte des servlets et des pages JSP (Java Server Page) sur un serveur Web. La partie métier de l'application est constituée de plusieurs composants EJBs de types variés qui peuvent être sur un même serveur d'applications ou sur plusieurs serveurs. La partie données est mise en œuvre dans une base données qui peut être répartie. Cette vision de distribution dans la plate-forme J2EE permet la décomposition logique d'une application sur plusieurs couches (présentation, métier, persistance).

La distribution verticale dans la plate-forme J2EE consiste à répartir les composants EJBs dans la couche métier sur plusieurs serveurs d'applications. Ce type de distribution souffre de deux limitations majeures :

- **Pas de protocole de communication inter-SA :** La distribution des composants EJB dans la couche de traitement est la responsabilité du déployeur qui place chaque EJB sur un serveur d'applications. Pour que deux composants EJBs communiquent entre eux, un EJB agit comme un client auprès du deuxième [EJBb03], mais il n'existe aucune communication directe entre les serveurs d'applications hébergeant les EJBs (Figure 3.2).

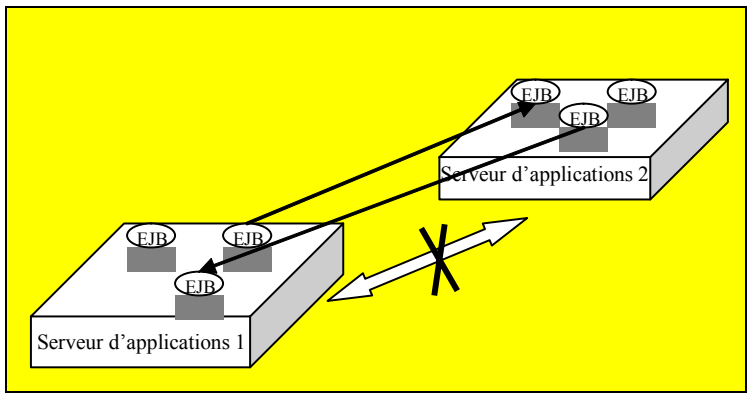

Figure 3.2 Communications entre les composants EJB

- **Pas de prise en compte de l'hétérogénéité des serveurs d'applications :** Les serveurs d'applications intègrent de plus en plus des services additionnels différents de ceux standardisés par la plate-forme J2EE. Les serveurs deviennent donc hétérogènes au niveau des services middleware offerts. Cette hétérogénéité

empêche parfois la coopération entre les composants EJB hébergés par différents serveurs d'applications. De plus, un EJB devient dépendant du serveur d'applications ce qui en empêche l'échange avec d'autres serveurs.

Ces deux limitations entraînent plusieurs problèmes concernant le traitement de la distribution dans la plate-forme J2EE.

3.1.2. Problèmes liés au manque de communication entre les SA

Les différents élément d'une application distribué sur la plate-forme J2EE communiquent directement entre eux afin d'organiser le fonctionnement de l'applications. Les serveurs d'applications situés au niveau middleware ne communiquent pas entre eux à cause de l'absence d'un protocole de communications inter-serveurs.

Cette absence de protocole limite la plate-forme selon deux axes :

- Le déploiement d'un composant EJB est effectué d'une manière isolée sur un serveur EJB. Si un serveur J2EE échoue à déployer un EJB à cause d'un manque de ressource ou de service le processus de déploiement échoue avec une erreur sans envisager la possibilité de chercher l'élément manquant sur un autre serveur d'applications.

- A l'exécution, deux EJBs installés sur deux serveurs d'applications différents communiquent sans que les serveurs ne se connaissent. Si les deux serveurs d'applications pouvaient négocier avant tout dialogue, ils pourraient établir des communications plus optimisées. Par exemple, ils pourraient s'échanger les instances d'EJB avant l'utilisation, ou les rapprocher sur des serveurs de cache.

Cette limitation empêche tout coopération directe entre les serveurs d'applications. Toute coopération entre les serveurs est effectué dans le cadre d'un regroupement des serveurs (comme les clusters) où un serveur d'administration synchronise cette tâche de coopération.

3.1.3. Problèmes liés à la non prise en compte de l'hétérogénéité

Comme la plate-forme J2EE se base initialement sur des services standardisés, la question de l'hétérogénéité n'était pas évoquée. La standardisation des services permet qu'un EJB s'exécute sur n'importe qu'elle serveur d'applications EJB. Néanmoins, il est possible pour un serveur d'applications d'avoir plusieurs services additionnels. L'utilisation des services additionnels par un EJB implique une sorte de dépendance entre l'EJB et le serveur d'applications. De plus, l'hétérogénéité n'est pas limitée au niveau des services middleware, elle existe aussi au niveau système d'exploitation et hardware. Le passage par la machine virtuel java ne fait que masquer les différences entre les systèmes d'exploitation mais la performance d'un EJB est sûrement affecté par la machine où le serveur d'applications s'exécute.

Ce manque de la notion d'hétérogénéité entraîne principalement deux problèmes dans la plate-forme J2EE :

- **Clusters nécessairement homogènes :**

Le seul regroupement entre plusieurs serveurs d'applications EJB se fait par la notion de grappe (*cluster*): une notion définie de manière propriétaire par les fournisseurs de serveurs d'applications [BEA, JONAS, JBOSS] mais qui ne traite pas réellement la distribution des EJB (cf 2.1.1.2).

Dans un cluster les EJB sont traités de manière homogène. Même si un type d'EJB présente un besoin particulier (quantité de mémoire, CPU, service additionnel), il est automatiquement répliqué sur tous les membres du cluster même ceux qui ne sont pas adaptés à son hébergement.

Pour prendre en considération les besoins particuliers des EJB, c'est au déployeur de spécifier l'emplacement pour chaque EJB. Afin de pouvoir prendre cette décision, le déployeur doit connaître l'état de tous les serveurs du cluster.

D'autre part, les serveurs d'un cluster ne communiquent pas directement pour réaliser les tâches de gestion. Un seul serveur d'administration est le responsable de la gestion. S'il échoue, les autres serveurs continuent à servir mais aucune tâche d'administration ne pourra plus être effectuée.

- **Manque d'adaptation :**

La plate-forme J2EE possède un ensemble de services standards offerts par le conteneur. Ces services sont gérés par la plate-forme est sont mis à disposition des EJB au moment du déploiement. Une des limitations de la plate-forme est de pouvoir gérer des services additionnels. Si la majorité des serveurs d'applications J2EE open-source permettent d'ajouter de nouveaux services (JBoss, Jonas), ils ne peuvent pas s'adapter dynamiquement aux besoins spécifiques des EJB.

Par exemple, si un EJB nécessite un service de reconnaissance vocale et que celui-ci n'est pas installé sur le serveur d'applications concerné l'EJB ne peut pas être déployé.

3.1.4. Conclusion : Vers la notion de fédération

L'absence de la notion d'hétérogénéité et le manque de communication entre les serveurs d'applications dans la plate-forme J2EE impliquent que toute la coopération doit être effectuée au niveau des composants EJB. En effet pour gérer une application distribuée à base d'EJB le développeur prévoit la coopération entre les composants EJB.

Nous pensons que cette tâche de coopération ne doit pas être limitée au niveau des composants ; le développeur n'a pas à intervenir pour assurer la coopération au niveau middleware. Une plate-forme qui prend en compte l'hétérogénéité entre les différents éléments participants, et qui permet la communication entre ces éléments offre un environnement dynamique qui soulage le développeur des tâches qui ne sont pas forcement d'ordre métier.

Cette raison nous a poussé à définir un environnement de serveurs d'applications hétérogènes qui permet la distribution verticale et l'adaptation automatique en permettant une coopération directe entre les serveurs d'applications. Nous appelons cette environnement une ***fédération*** de serveurs d'applications.

Dans la section suivante nous présentons une définition de la fédération, le but de cette fédération, et les éléments nécessaires pour sa constitution.

3.2. Fédération de serveurs d'applications J2EE

3.2.1. Définition

Une fédération de serveurs d'applications est un ensemble de serveurs hétérogènes hébergeant une plate-forme J2EE distribués sur plusieurs nœuds d'exécution, qui communiquent entre eux et qui échangent des composants EJB.

Nous avons choisi le terme ***fédération*** pour refléter le fait que les serveurs d'applications ne sont pas forcément identiques ; chaque serveur a ses propres configurations, caractéristiques, et ressources et il peut se joindre ou se retirer de la fédération à tout moment. Cette diversité permet d'héberger différents types d'applications qui ont éventuellement des besoins variés au niveau des services requis par leurs composants logiciels.

3.2.2. Objectifs

Le but d'une fédération est d'offrir un environnent coopératif de serveurs où les applications sont distribuées de manière automatique selon leurs besoins particuliers et où les composants des applications sont dynamiquement adaptés suite aux changements des applications ou de l'environnement d'exécution.

Les différents types de changement au niveau de l'application ont été identifiés en [HOF93] en quatre catégories : les changement de structures (ajout ou retrait de composant), les changement de géométrie (site d'exécution), les changements d'implantations (mise en œuvre), et les changement d'interface.

En ce qui concerne l'environnement d'exécution, les changements peuvent être au niveau des services middleware, ou au niveau du système d'exploitation et des matériels.

Notre fédération vise principalement deux objectifs :

- **Déploiement global adapté des EJB :** Le déploiement visé dans la fédération est un déploiement global adapté. Il est global dans le sens où il considère le déploiement de l'EJB sur le meilleur serveur parmi un ensemble de serveurs d'applications. Une action de déploiement peut être déclanché par un serveur d'applications mais elle est traitée par l'ensemble de serveurs. Il est adapté dans le sens où il est effectué en fonction des besoins des EJB et des caractéristiques des serveurs d'applications disponibles pour les héberger.

- **Adaptation dynamique :** Suite aux différents types de changements au niveau de l'application ou au niveau de l'environnement d'exécution, les besoins des EJBs changent aussi. Le deuxième objectif de la fédération, est de détecter automatiquement les nouveaux besoins des composants afin de fournir une adaptation dynamique. Cette adaptation consiste à relocaliser les composants sur des serveurs mieux adaptés à leurs nouveaux besoins.

3.2.3. Architecture de la fédération

Dans l'architecture muti-tiers, la fédération est située dans la couche de traitement (Figure 3.3). La couche de présentation et la couche de persistance ne sont pas prises en compte dans un premier tems afin de simplifier l'approche.

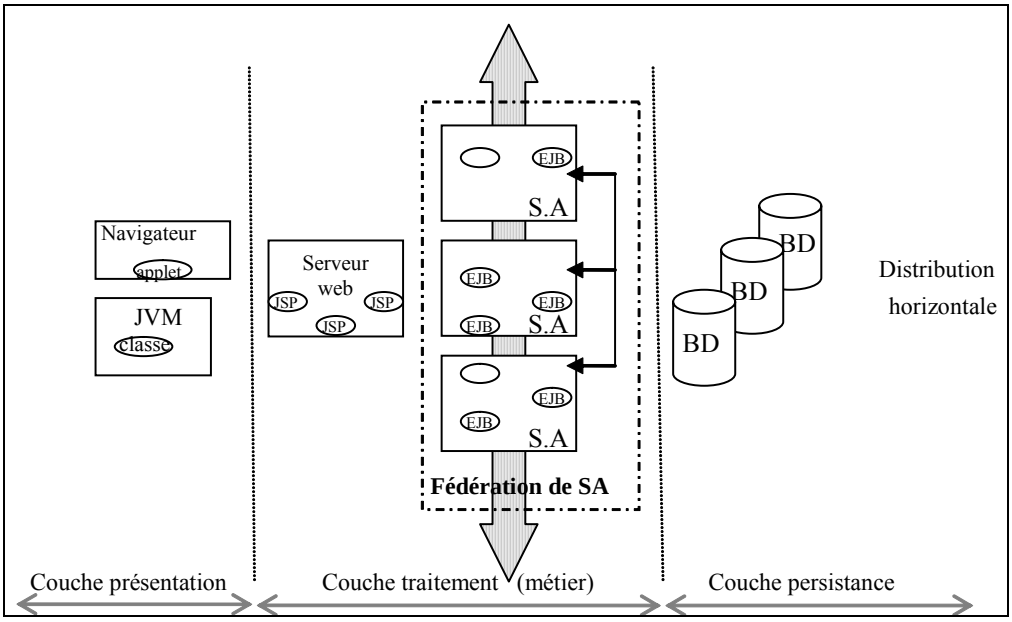

Figure 3.3 Emplacement de la fédération

Comme illustré dans la Figure 3.4, la fédération constitue un environnement d'exécution d'EJB composée de plusieurs niveaux : le bas niveau (système d'exploitation), le niveau middleware (les serveurs d'applications et ses services, qui représentent la plateforme d'exécution des composants EJB) et le niveau application (les composants EJB qui représentent la logique métier). Les trois niveaux coopèrent pour gérer les EJB déployés sur les serveurs de la fédération.

Chaque nœud de la fédération est une machine physique, un serveur d'applications, et un ensemble de services d'administration. Les nœuds peuvent être

hétérogènes au niveau système, et au niveau middleware. Néanmoins, ce qui permet la coopération entre les nœuds est d'avoir une interface unie de gestion[*].

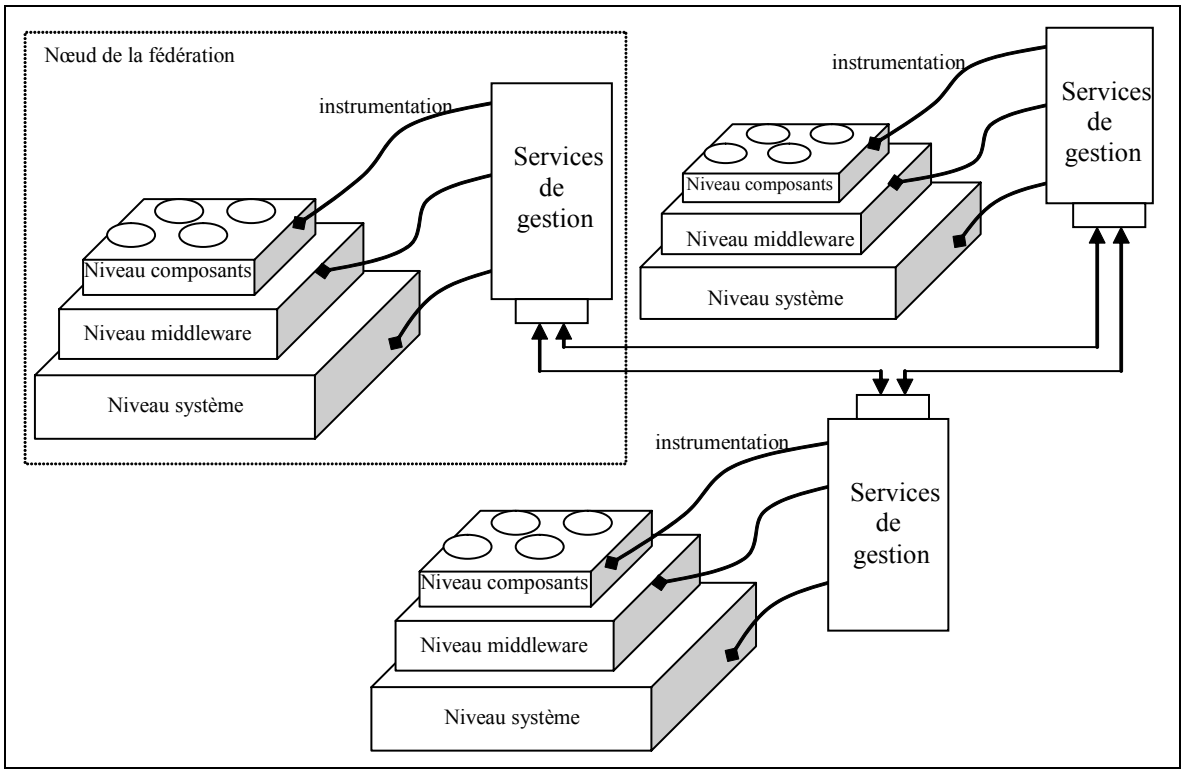

Figure 3.4 Architecture de la fédération

La constitution de la fédération dépend principalement de trois éléments qui se résument à :

- Permettre la coopération entre les serveurs d'applications afin d'effectuer des tâches d'ordre administratif.
- Identifier les serveurs d'applications participants à la fédération par un protocole qui permet à un nouveau serveur d'applications de se joindre à (ou se retirer de) la fédération en informant les autres serveurs de son existence (ou de son absence).
- Assurer la sécurité dans la fédération et contrôler l'accès aux informations de gestion d'une manière qui permette la coopération entre les serveurs tout en préservant la sécurité de chacun des serveurs.

[*] Dans cette thèse nous employons les mots *gestion* et *administration* d'une manière interchangeable.

Les deux derniers points ne sont pas étudiés dans le cadre de cette thèse. Nous focalisons sur l'analyse du déploiement de composants sur la fédération en permettant la coopération entre les serveurs.

Dans la fédération, la coopération s'effectue au niveau de la gestion du déploiement permettant la communication entre les différents niveaux sur un même nœud (système, middleware et composants), et entre les différents nœuds de la fédération. La coopération consiste à échanger les informations de gestion entre les nœuds et à effectuer des opérations de gestion afin de réaliser une action de déploiement.

Chaque nœud de la fédération défini un ensemble de services de gestion permettant l'instrumentation et la gestion des différentes couches afin d'améliorer le déploiement des composants et leurs adaptation au changement.

Avant d'identifier les services de gestion nécessaire dans chaque nœud afin de gérer de déploiement sur la fédération nous détaillons dans la section suivante le processus de déploiement dans la fédération.

3.2.4. Vers un déploiement global adapté

Le *déploiement* est un terme utilisé pour définir tous les processus impliqués pour mettre en service un nouveau logiciel correctement dans son environnement, y compris l'installation, la configuration et le fonctionnement. En général, cette définition du déploiement est associée aux applications utilisées par un seul utilisateur et qui s'exécute sur une seule machine. Dans ce contexte, plusieurs travaux de recherches ont permis la présentation des outils performants de déploiement comme SoftwareDock [HALa99, HALb99, DOCK], Tivoli [TIVOLI], InstallShield [SHIELD], JavaWebStart [JWS]. Chacun de ces outils permet d'effectuer une ou plusieurs phases de déploiement pour un logiciel sur une machine précise au niveau client d'une architecture 3-tier.

Le déploiement dans la plateforme J2EE est par contre effectué au niveau intermédiaire (traitement) et il consiste à fournir aux EJBs les services non fonctionnels et à mettre ces EJB à disposition des autres composants en les enregistrant dans le répertoire de nommage JNDI. Ce processus est effectué par le serveur d'applications.

Dans le contexte de notre fédération nous définissons le déploiement comme étant *le processus permettant d'installer un composant EJB sur le serveur qui répond au mieux à ses besoins et de l'adapter dynamiquement aux changements éventuels survenus au niveau de l'application ou au niveau de l'environnement d'exécution.*

Nous distinguons donc trois phases dans le processus du déploiement dans la fédération (Figure 3.5):

- La phase d'évaluation : C'est une phase qui doit être effectuée avant l'installation. Elle consiste à identifier les besoins de l'EJB.
- La phase décisionnelle : cette phase consiste à détecter les caractéristiques des serveurs d'applications disponibles dans la fédération afin de prendre une décision concernant l'emplacement de l'EJB.
- La phase d'adaptation : Après l'installation de l'EJB, le serveur continue à observer le fonctionnement de l'EJB afin de détecter un changement de besoins et adapter l'EJB à ces nouveaux besoins.

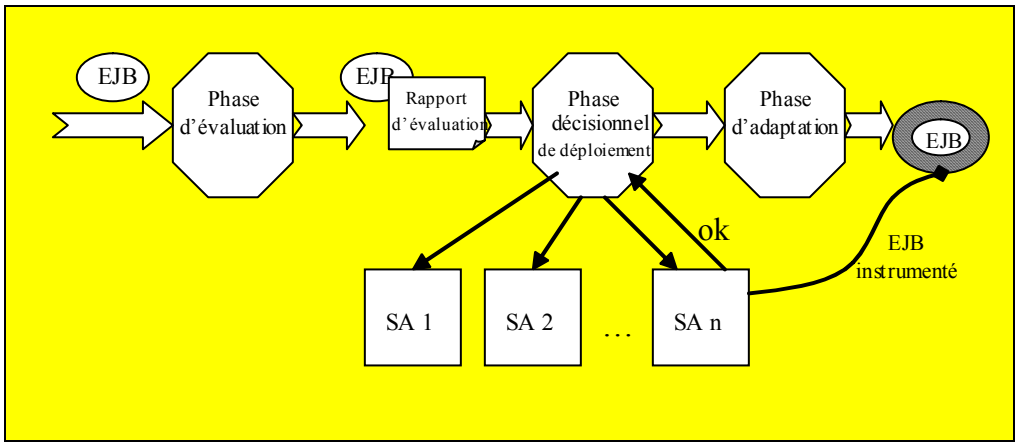

Figure 3.5 Les phases de déploiement dans la fédération

Après avoir établi une définition détaillée du déploiement global adapté, nous présentons dans la section suivante notre approche de gestion du déploiement dans la fédération.

3.3. Proposition pour la gestion du déploiement dans la fédération

Dans cette section nous présentons dans un premier temps notre modèle d'administration de la fédération. Ce modèle n'est pas limité à une fédération de serveurs d'applications EJB car il peut être vu comme un modèle d'administration généralisé pour la gestion des serveurs de différents modèles de composants comme CCM ou DCOM. Dans un deuxième temps nous détaillons l'architecture adoptée pour le mdoèle dans le cadre de la plate-forme J2EE.

3.3.1. Le Modèle

Dans notre modèle chaque serveur gère son propre environnement. Afin de réaliser le déploiement adapté sur la fédération, les serveurs coopèrent au niveau de l'administration en présentant la même interface de gestion fournissant les mêmes services de gestion.

3.3.1.1. Eléments du modèle

Le modèle proposé est organisé en deux parties (Figure 3.6):

- Un ensemble de service de gestion pour chaque serveur de la fédération,
- Un service de coordination pour l'organisation de la gestion du déploiement global.

Le but des services de gestion est de gérer les trois phases de déploiement global adapté. Le service de coordination définit la manière d'échanger des informations de gestion entre les services de gestion d'un même serveur et entre les services des différents serveurs, il définit aussi l'ensemble des règles à respecter lors de la prise d'une décision de gestion globale dans la fédération.

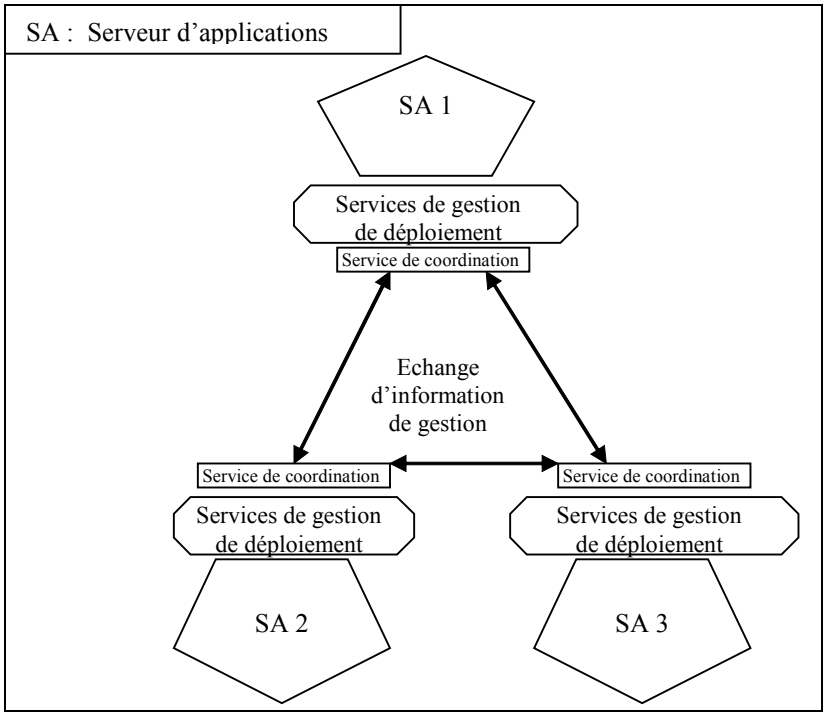

Figure 3.6 Modèle de gestion du déploiement dans une fédération de serveurs d'applications

3.3.1.2. Stratégie d'administration

Notre modèle d'administration propose une gestion **décentralisée** où il n'y a pas un seul serveur pour l'administration globale de la fédération. Afin d'établir la gestion décentralisée nous donnons le droit à chaque serveur de :

- contrôler son propre environnement (système, middleware, composants),
- connaître les serveurs participant à la fédération,
- consulter l'état de chaque serveur, et
- prendre une décision concernant le choix du serveur pour déployer un composant.

Avec cette stratégie de gestion, l'administration n'est pas sous la responsabilité d'un seul serveur d'administration. Elle n'est pas distribuée non plus car on ne distribue pas les services de gestion sur quelques serveurs mais on permet à chaque serveur de prendre le rôle d'un serveur d'administration pour effectuer des décisions de déploiement. Avec cette stratégie de décentralisation, si un serveur tombe en panne il ne risque pas d'influencer le fonctionnement des autres serveurs dans la fédération.

Tous les serveurs d'applications ont le droit d'initialiser une demande de déploiement d'un EJB dans la fédération. Le serveur essaie de traiter le déploiement localement dans le serveur. En cas d'impossibilité d'effectuer le déploiement, le

serveur ne rejette pas l'EJB mais il joue le rôle d'administrateur en essayant de contacter les autres serveurs disponibles dans la fédération. Le serveur prend ensuite une décision de déploiement en se basant sur la correspondance entre les besoins de l'EJB et les réponses obtenues des autres serveurs.

3.3.1.3. Services nécessaires

Afin de permettre à chaque serveur d'applications de contrôler son propre environnement, chaque serveur possède un ensemble de services d'administration. Ces services doivent répondre aux objectifs de la fédération :

Le premier objectif concerne le déploiement global adapté des EJB dans la fédération. Pour administrer ce processus il nous faut :

- Connaître l'état de chaque serveur dans la fédération afin de prédire le pourcentage de disponibilité des ressources demandées par l'EJB. La décision finale de déploiement se base sur la comparaison de disponibilité de ressources fournies par les différents serveurs.
- Connaître les besoins des EJB. Ces besoins sont principalement : les ressources systèmes (CPU, mémoire), réseaux (bande passante) et middleware (services J2EE et services additionnels).
- Permettre la localisation d'un EJB qui se trouve sur la fédération.

Le second objectif concerne l'adaptation automatique. Afin d'atteindre cet objectif nous avons fixé deux points principaux à réaliser :

- Superviser le comportement des EJB déployés sur les serveurs de la fédération afin de détecter une anomalie de fonctionnement.
- Prendre une décision de migration de l'EJB vers un autre serveur lorsqu'une anomalie est causée par un manque ou une limitation de ressource.

En décomposant les deux objectifs en sous-tâches nous arrivons à identifier les services nécessaires pour le déploiement :

- Service d'instrumentation de l'environnement d'exécution pour extraire les informations de gestion nécessaire pour chaque serveur
- Service d'évaluation de besoins qui permet d'évaluer les besoins d'un EJB avant son déploiement sur la fédération
- Service de localisation permettant la recherche d'un EJB sur la fédération
- Service de suivi des composants EJB afin de détecter un changement de besoins
- Service de migration des composants EJB permettant l'échange d'EJB entre les serveurs afin de les adapter à leurs nouveaux besoins.

3.3.1.4. Gestion de coopération

L'ensemble de services présentés ci-dessus participe au processus du déploiement adapté. Mais il reste à établir la coopération d'une part entre les services du même serveur et d'autre part entre les services des différents serveurs de la fédération. Cette coopération doit être basée sur un ensemble de règles spécifiant les décisions à prendre pour effectuer une action de déploiement sur la fédération.

En effet, les décisions à prendre affectent le type de coopération à établir entre les services. Ces décisions peuvent être différentes d'un serveur à un autre. Par exemple, un serveur d'applications peut accepter de déployer un EJB localement même avant de faire l'évaluation de besoins, ensuite en fonction des mesures effectués par le service de suivi il peut décider de garder l'EJB ou de faire l'évaluation et de l'envoyer vers un autre serveur de la fédération. Dans ce cas la coopération entre les services consiste à appeler le service d'évaluation après avoir installé l'EJB sur le serveur. Sur un serveur d'applications qui n'accepte pas d'effectuer l'installation avant l'évaluation, le service d'évaluation doit être appelé avant l'installation et le service de suivi est appelé ensuite pour assurer que l'EJB ne dépasse pas ce qui a été déclaré dans la phase d'évaluation.

Afin de ne pas lier les services de déploiement fortement entre eux, nous proposons l'ajout d'un service de coordination dans chaque serveur, ce service effectue la coordination entre les services en fonction des règles du serveur d'application. En plus de la coordination entre les services d'un même serveur, le service assure aussi la communication entre les services des différents serveurs de la fédération. Par exemple, quand un serveur d'application ne peut pas assurer les besoins d'un EJB, le service de coordination contacte les autres serveurs dans la fédération (notamment le service d'instrumentation) afin de déterminer le serveur le mieux adapté pour accueillir l'EJB.

3.3.1.5. Conclusion

Le modèle que l'on propose pour la gestion de déploiement dans la fédération adopte une stratégie d'administration décentralisée afin d'éviter la dépendance sur un seul serveur d'administration. Le modèle se base principalement sur un ensemble de services de gestion de déploiement et un service de coordination, l'ensemble de ces services doit être implanté par chaque serveur de la fédération.

Afin d'assurer la coopération entre les services des différents serveurs de la fédération, il faut que l'accès aux services soit standardisé en passant par un des standards d'administration présenté en section 2.1.2. Dans la section suivante nous adoptons JMX pour définir l'architecture de notre modèle.

3.4. Architecture basée sur JMX

Dans la fédération, les nœuds ne sont pas homogènes, ils sont différents selon les vendeurs des périphériques, des systèmes d'exploitations, des serveurs d'applications, et des composants. Pour pallier à cette diversité, nous avons choisi JMX afin d'unifier l'interface d'administration dans la fédération (Figure 3.7).

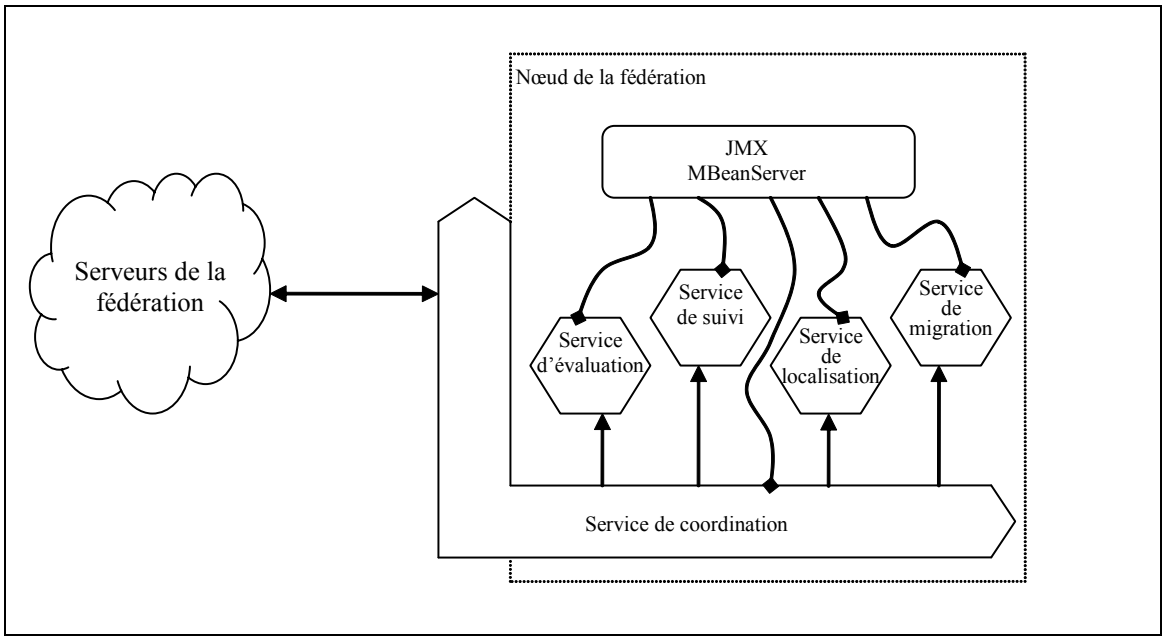

Figure 3.7 Architecture de gestion de déploiement basé sur JMX

Sur chaque noeud de la fédération, un MBeanServeur regroupe l'ensemble de service de gestion en les enregistrant en tant que MBeans. Le service de coordination organise la coopération entre les services d'un même nœud et entre les serveurs de la fédération.

Dans le reste de cette section nous justifions le choix de JMX, puis nous présentons l'architecture de chacun des éléments de notre modèle : les services de gestion et le service de coordination. Une mise en œuvre de cette architecture est décrite dans chapitre 4.

3.4.1. Choix de JMX

Plusieurs facteurs nous ont poussé à choisir JMX comme base pour notre proposition d'administration de la fédération. Nous étudions ces facteurs en détails :

- Le premier facteur est l'utilisation du langage de programmation java. En effet, comme la plate-forme J2EE est dépendante du langage de programmation Java, il

est donc plus logique d'utiliser JMX qui offre les API en Java pour la gestion des applications java. Une autre solution aurait été d'utiliser un autre standard d'administration en l'interfaçant avec java. Les efforts d'interfaçage auraient été des efforts perdus comme il existe déjà des services d'interfaçage entre JMX et les standards d'administration les plus connus (SNMP et WBEM/CIM).

- JMX offre un moyen d'intégration. En effet, chaque serveur d'applications fournit sa propre interface pour l'ajout des classes personnalisées. Ces classes sont parfois appelés "startup", elles peuvent être utilisées pour l'ajout des services additionnels qui ne sont pas spécifiés par le modèle J2EE : service de maintenance, service de planification de tâches, service de reconnaissance vocale. Le problème avec ces classes est qu'elles utilisent des interfaces différentes qui dépendent directement de l'implantation du vendeur du serveur. Cette dépendance fait perdre la notion de portabilité de J2EE et empêche la coopération entre les serveurs. Un service additionnel qui a été implanté pour un serveur ne sera pas compatible avec un autre serveur issu d'un autre vendeur. L'architecture de JMX offre une solution propre à ce problème. Donc en adoptant JMX dans la fédération nous offrons une interface standard d'extension permettant la coopération entre les serveurs.
- Un autre avantage de JMX est qu'elle permet l'ajout des nouveaux services (MBean) à l'exécution. Dans notre fédération cette caractéristique est primordiale. En effet, un élément essentiel de notre fédération est la coopération entre les serveurs, cette coopération peut impliquer un échange des services additionnels. Si chaque échange de service demande l'arrêt et le redémarrage du serveur, la coopération serait très coûteuse dans la fédération. Le service de téléchargement dynamique appelé m-let (cf. p.41) de JMX permet le téléchargement d'un MBean via le réseau.

Finalement, JMX a été utilisé pour la gestion des applications centralisées ainsi que pour des applications distribuées [KRE01, KRE02]. Il a également été approuvé pour la gestion des serveurs d'applications J2EE [ETHO02, FLEb02]. Dernièrement, JMX a été intégré dans le cœur de java [J2SE04] afin de permettre la gestion de la machine virtuelle.

3.4.2. Services de gestion

Dans section 3.3.1.3 nous avons identifié les services nécessaires pour réaliser le déploiement adapté sur la fédération. Dans cette section nous détaillons l'architecture de chacun des services proposés. La mise en œuvre de ces services est décrite dans chapitre 4.

3.4.2.1. Service d'instrumentation

Ce service présente la base de connaissance de la plate-forme d'administration. Il offre des outils d'instrumentation qui permettent la modélisation de l'ensemble des éléments de l'environnement d'exécution, en objets instrumentés.

Pour procéder au déploiement d'un EJB après en avoir fait l'évaluation, il faut choisir le serveur d'applications qui répond le mieux aux besoins de l'EJB. Ce choix se base principalement sur la connaissance de l'état de chaque serveur dans la fédération. Dans notre modèle nous proposons d'instrumenter chacun des serveurs dans la fédération afin de pouvoir évaluer le degré de correspondance de chaque serveur avec les besoins de l'EJB.

Le service d'instrumentation permet d'avoir une vue unifiée de tous les nœuds dans la fédération. Avec cette vue unifiée les informations de gestion de tous les nœuds dans la fédération peuvent être récupérées et analysées d'une manière homogène. L'ensemble de services de gestion utilise ce service pour récupérer des informations.

L'instrumentation de chaque nœud de la fédération est modélisée en trois couches (Figure 3.8) :

- Couches réseau et système d'exploitation : L'instrumentation de cette couche permet l'évaluation de la disponibilité des ressources pour un nouvel EJB.
- Couche middleware : L'instrumentation de cette couche sert à connaître les services additionnels qui existent dans chaque serveur d'applications. (les services de gestion que l'on propose se situe parmi les services additionnels).
- Couche application (EJB) : L'instrumentation au niveau de l'application n'est pas nécessaire pour la prise de décision de déploiement, mais elle est par contre nécessaire pour le suivi du comportement de l'EJB. Ce suivi permet l'adaptation automatique de l'EJB en cas de changement de besoins.

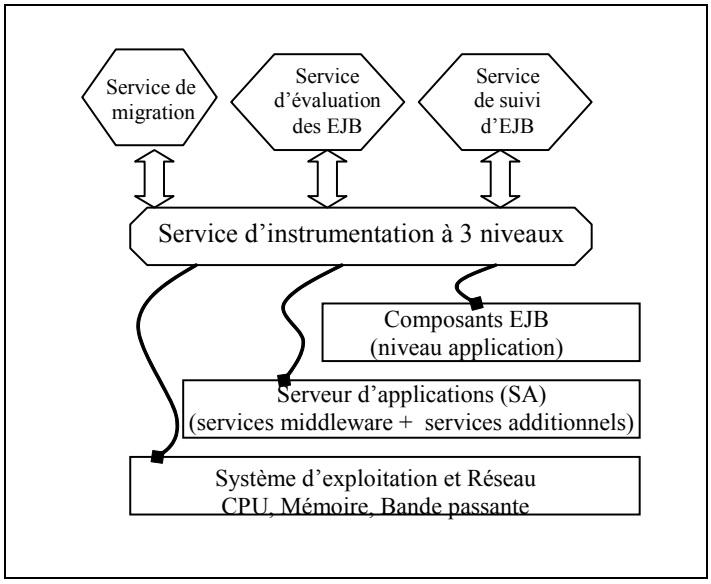

Figure 3.8 Instrumentation à 3 niveaux

3.4.2.2. Service d'évaluation de besoins

Avant le déploiement d'un EJB sur la fédération, notre modèle propose d'évaluer ses besoins. Cette phase d'évaluation permet au système de gestion d'établir un choix de déploiement. Elle permet également la détection de dégradation de performance et l'adaptation automatique après déploiement.

Les besoins que nous considérons pour évaluer un EJB sont organisés en deux catégories, les besoins au niveau système d'exploitation (mesures bas-niveau) et les besoins au niveau middleware.

- Mesures de bas niveau :
  - CPU : Le taux de consommation de la CPU par un EJB permet d'évaluer la surcharge de l'EJB sur le nœud de la fédération.
  - Mémoire : Les EJB qui consomment le plus de CPU ne consomment pas forcément le plus de mémoire et vice versa. D'autre part, la quantité de mémoire requise par l'EJB diffère selon son type. L'EJB stateless par exemple consomme moins de mémoire que l'EJB statefull car ce dernier instancie un objet par client afin de garder l'état de la session cliente. De ce fait, dans la fédération il est possible de trouver des nœuds qui consomment plus de CPU que de mémoire ou l'inverse.
  - Bande passante : la fréquence d'octets envoyés ou reçus par un EJB est une première mesure pour caractériser le trafic du réseau.

- Mesures au niveau Middleware :
  - Services additionnels et ressources : à ce niveau, nous avons jugé que les mesures sur les services standards non fonctionnels de la plate-forme J2EE n'ont pas un effet important sur la prise de décision de déploiement parce que ces services ont été standardisés et leur implantation est obligatoire dans tous les serveurs d'applications. Un EJB qui demande un service standard précis va le trouver sur n'importe quel serveur dans la fédération. Par contre, les ressources et les services additionnels non standardisés n'existent pas nécessairement sur tous les serveurs. Il nous faut alors connaître les services additionnels et les ressources demandés par l'EJB afin de choisir un serveur qui les offre.

L'ensemble de ces mesures participe à la prise de décision de déploiement pour chaque nouvel EJB.

Le service d'évaluation détecte les ressources nécessaires pour un EJB et fait les tests pour déterminer leurs taux de consommation. Ce service est composé de deux phases (Figure 3.9). Une phase pour déterminer les ressources utilisées par l'EJB dans la couche middleware, et une phase pour évaluer le taux de consommation des ressources dans la couche système.

La complexité possible du fonctionnement interne des EJB rend quasiment impossible une évaluation automatique des besoins de composants. Nous proposons de réaliser une évaluation semi-automatique, où le développeur de l'EJB donne un certain nombre de caractéristiques qui vont permettre une évaluation plus fine du comportement du composant sur un serveur d'applications. Parmi les informations que le développeur fournit, nous voyons les éléments suivants :

- Le nombre maximum de clients concurrents sur l'EJB
- La méthode qui consomme le plus de CPU
- La méthode qui consomme le plus de mémoire
- La méthode qui utilise le plus de bande passante

Ces éléments sont bien évidemment fonction des paramètres d'appel des méthodes.

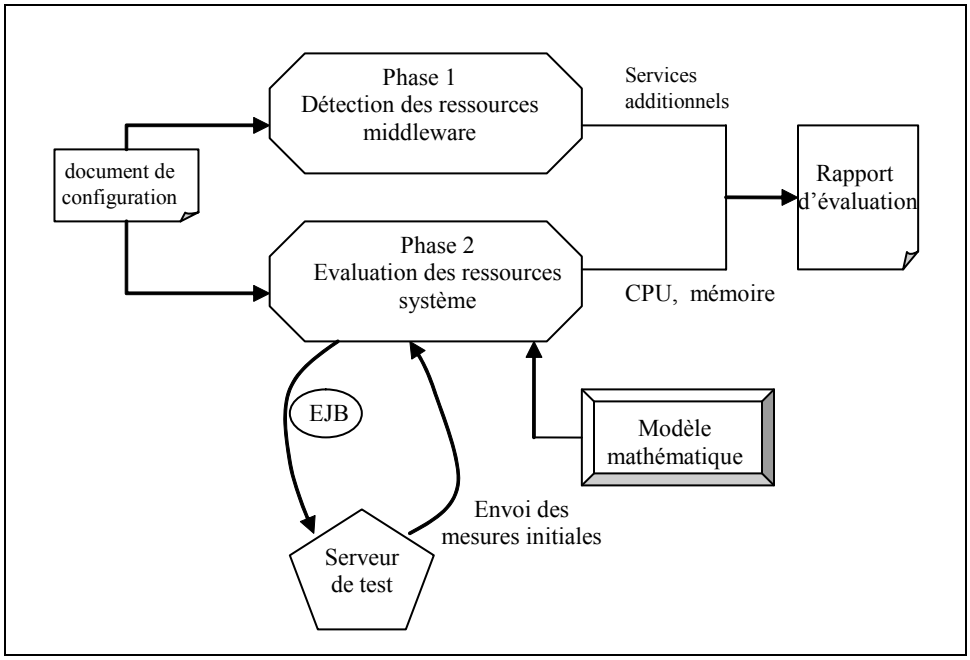

Figure 3.9 Architecture du service d'évaluation

Dans la première phase, nous ne considérons pas les services standards utilisés par un EJB comme des ressources car ceux-ci forment le cœur de la plateforme J2EE et ils sont obligatoires pour chaque serveur d'applications. Néanmoins, nous prenons en compte les services additionnels intégrés aux serveurs d'applications. Comme ces services ne sont pas standardisés, chaque serveur d'applications peut avoir des services additionnels qui n'existent pas forcement dans les autres serveurs de la fédération. La détection des services additionnels demandés par un EJB nous permet de le déployer sur un serveur d'applications qui les offre.

La deuxième phase consiste à détecter automatiquement le taux de consommation des ressources système, notamment la CPU, mémoire et bande passante. Dans cette phase, le service d'évaluation envoie l'EJB vers un serveur de test afin de faire des mesures isolées. Ces mesures sont utilisées avec un modèle mathématique afin de déterminer le taux de consommation qu'un EJB utilise (cf 4.2).

A la fin des deux phases, le service génère un rapport d'évaluation qui indique les valeurs représentant la consommation des ressources pour ce type d'EJB. Ce rapport est utilisé afin de choisir le serveur le plus adéquat.

3.4.2.3. Service de localisation

La fédération doit être vue de l'extérieur comme un seul serveur d'applications. Un client ne se charge pas de trouver ou est ce que un EJB a été

déployé sur la fédération. Il lui suffit de contacter n'importe quel serveur d'applications dans la fédération et c'est la responsabilité du serveur de trouver l'EJB et de le mettre à disposition du client. Il est donc indispensable de définir un service de localisation dans notre fédération.

La nécessité de ce service est motivée par deux points :

- Le déploiement sur la fédération se fait d'une manière globale et décentralisé, ce qui signifie que le client ne sait pas forcement où un EJB a été déployé et il ne peut pas le demander à un serveur d'administration (il n'y a pas un serveur administrateur qui connaît l'emplacement de tous les EJB).
- Il est possible qu'un EJB change de serveur pour l'adapter à de nouvelles conditions d'exécution ou suite au changement du comportement d'un serveur d'applications.

Parmi les services standards définis dans la plate-forme J2EE, chaque serveur d'applications contient un annuaire de nommage appelé JNDI (Java Naming and Directory Interface) [JNDI]. Ce service référence les EJB déployés sur le serveur. Si l'EJB n'existe pas sur le serveur, il ne se charge pas de localiser l'EJB sur les autres serveurs. Le but de notre service de localisation est de chercher un EJB qui n'a pas été trouvé localement sur l'ensemble des serveurs de la fédération.

Deux approches sont possibles pour réaliser un service de nommage qui prend en compte la recherche d'un composant sur toute la fédération, une approche centralisée et une approche répartie.

Annuaire de nommage centralisé

Un annuaire centralisé enregistre les composants déployés sur tous les serveurs dans la fédération. Les serveurs d'applications qui n'ont pas le composant demandé, accèdent à l'annuaire centralisé afin de retrouver l'adresse du serveur qui dispose du composant (Figure 3.10).

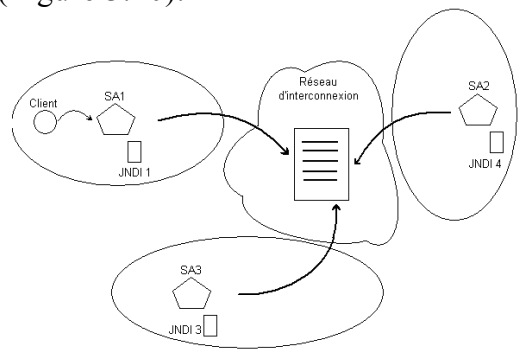

Figure 3.10 Approche centralisé d'annuaire de nommage

Les inconvénients de cette solution sont les suivantes :

- Faible disponibilité : si l'annuaire central n'est pas disponible toute l'architecture est bloquée.
- Actualisation des données de l'annuaire : il faut mettre en œuvre un mécanisme d'actualisation du contenu de l'annuaire afin qu'il contienne les composants effectivement déployés sur chaque serveur.

Nous évitons cette approche centralisée afin de conserver l'aspect de notre fédération où il n'y a pas de dépendance entre les serveurs.

Annuaire de nommage réparti

Une architecture d'annuaires répartis peut être utilisée dans la fédération. Dans cette approche, chaque serveur d'application garde son annuaire local en ajoutant la fonctionnalité de pouvoir communiquer avec les autres serveurs dans la fédération. Ainsi on empêche les problèmes de disponibilité que pose un annuaire centralisé. Chaque annuaire enregistre les composants déployés localement sur le serveur. Lorsqu'un serveur d'applications n'implante pas localement un EJB demandé par un client, il fait une recherche de l'EJB dans la fédération (Figure 3.11).

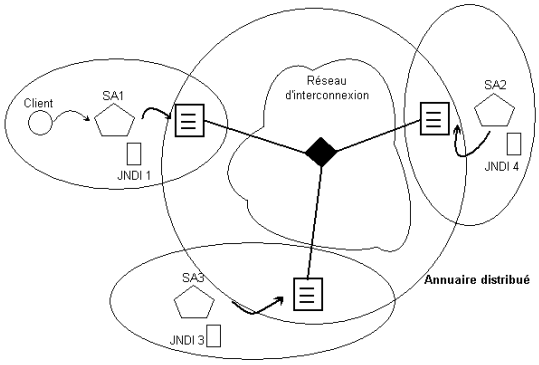

Figure 3.11 Approche répartie d'annuaires de nommage

Cette approche nous offre une meilleure disponibilité car si un annuaire tombe en panne seulement les composants du serveur d'applications associé ne seront pas accessibles depuis les autres serveurs.

Nous adaptons cette approche en réalisant un service de nommage étendu que l'on appelle : Proxy JNDI

La Figure 3.12 montre l'utilité de ce service. En effet quand l'annuaire JNDI du serveur d'applications échoue à trouver un EJB, le service de localisation cherche l'EJB sur l'ensemble des serveurs de la fédération. Quand l'EJB est localisé, le service signale avoir trouvé l'EJB.

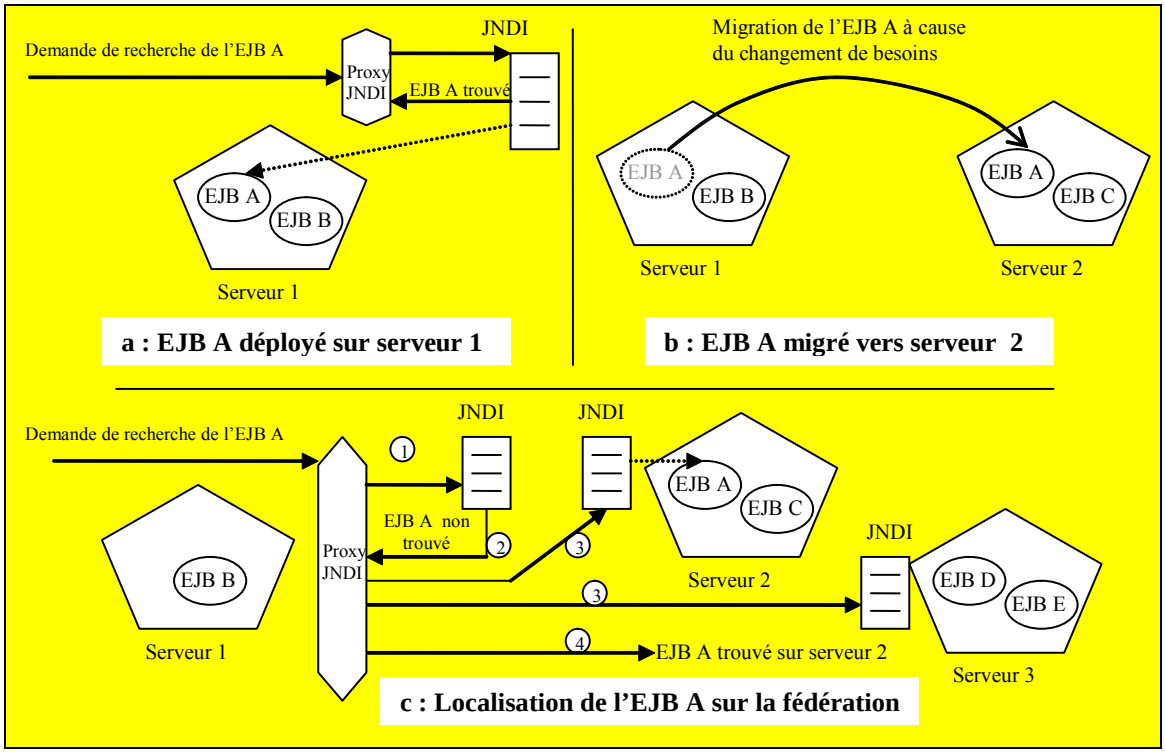

Figure 3.12 Rôle du service de localisation

3.4.2.4. Service de suivi des EJB déployés

L'évaluation d'un EJB permet de décrire les besoins initiaux d'un EJB. Suite à cette évaluation l'EJB est placé sur le serveur d'applications qui correspond le mieux au comportement de l'EJB. Néanmoins, les besoins d'un composant ne sont pas toujours statiques, et ils peuvent évoluer avec le temps. C'est pourquoi nous proposons dans notre modèle d'ajouter un service de suivi des EJB. Le but de ce service est de permettre à chaque serveur dans la fédération de suivre l'évolution des besoins de ces EJB.

La Figure 3.13 montre le fonctionnement du service de suivi. Dès le déploiement d'un nouveau EJB sur un serveur, le service commence à suivre son fonctionnement. Le service reçoit en entrée le rapport d'évaluation de l'EJB (cf 3.4.2.2). Le service repose sur l'instrumentation afin de suivre le fonctionnement de l'EJB. Après le déploiement d'un EJB, le service commence à envoyer périodiquement des appels de suivi. Le but de ces appels est de vérifier que l'EJB ne dépasse pas le seuil de consommation indiqué par le rapport d'évaluation (cf 4.2).

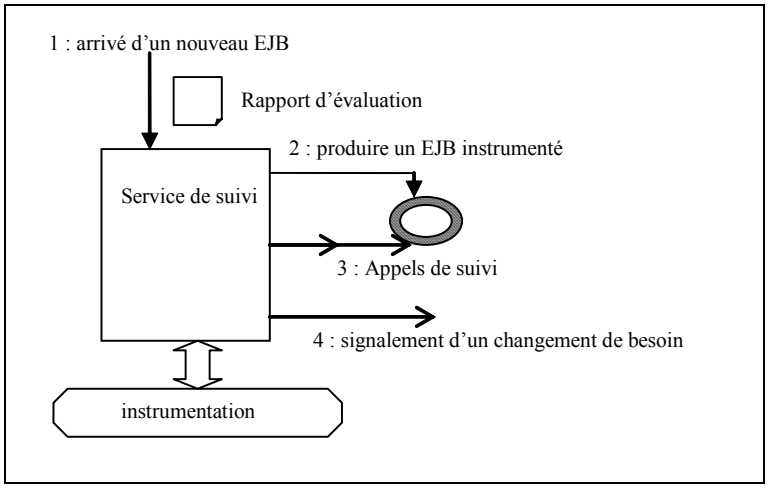

Figure 3.13 Fonctionnement du service de suivi

Nous avons deux stratégies de suivi des composants EJB. La première stratégie consiste à envelopper le composant et intercepter tous les appels vers l'EJB. Avec cette approche on obtient une référence sur chaque instance de l'EJB et en conséquence on récupère des statistiques et des mesures détaillées sur le fonctionnement de toutes les instances (ex : la méthodes appelée plus souvent, les instances les plus actives,..). Par contre, cette approche est très coûteuse comme elle implique une intervention sur le code de l'EJB à chaque appel d'une méthode.

La deuxième stratégie consiste à suivre une seule instance d'évaluation de chaque type d'EJB. Avec cette approche de suivi on ne peut pas obtenir les détails de fonctionnement de toutes les instances de l'EJB. Par contre, on peut estimer la performance de toutes les instances à partir de l'instance d'évaluation.

Dans notre service de suivi, nous adoptons la seconde approche. Elle nous paraît suffisamment satisfaisante au niveau de la détection du dysfonctionnement ou de la dégradation de performance, tout en restant dans des limites raisonnables au niveau des coûts.

Le service de suivi envoie régulièrement des appels à l'instance d'évaluation représentant un type d'EJB. Principalement, les appels testent le temps d'exécution et la quantité de mémoire pour chaque méthode invoquée. La fréquence d'appels ne doit pas être trop élevée pour ne pas influencer la performance du serveur. Elle ne doit pas non plus être trop espacée, afin d'éviter les coûts de passivation et de re-activation de l'instance d'évaluation.

3.4.2.5. Service de migration des composants

L'échange des EJB entre les serveurs de la fédération est essentiel pour permettre la coopération entre les serveurs afin d'assurer pour les composants une adaptation automatique suite aux changements de besoins.

Le service de migration permet à chaque serveur dans la fédération d'envoyer ou recevoir des EJB des autres serveurs.

Principalement, la décision de faire migrer un EJB se fait pour deux raisons :

- Permettre le déplacement d'un composant EJB d'un serveur qui ne répond plus aux besoins de l'EJB sur un serveur mieux adapté pour accueillir le composant.
- Permettre aux clients d'avoir une référence sur un composant EJB qui n'est pas déployé sur le serveur d'applications contacté par le client mais retrouvé ailleurs dans la fédération.

Nous avons décomposé le service de migration en plusieurs phases (Figure 3.14).

- **Dans la première phase :** le service reçoit une notification de migration qui indique le nom de l'EJB à faire migrer. Cette notification peut provenir de deux sources différentes :
  - suite à un appel de suivi qui détermine la nécessité de migrer un EJB afin de l'adapter aux nouveaux changements. Dans ce cas, le service de migration demande de migrer un EJB qui existe dans le serveur local vers un serveur d'applications mieux adapté pour l'EJB.
  - suite à une recherche d'EJB qui n'a pas été trouvé sur le serveur local. Dans ce cas, c'est le service de localisation qui contacte le service de migration pour migrer un EJB du serveur d'applications distant vers le serveur d'applications local.

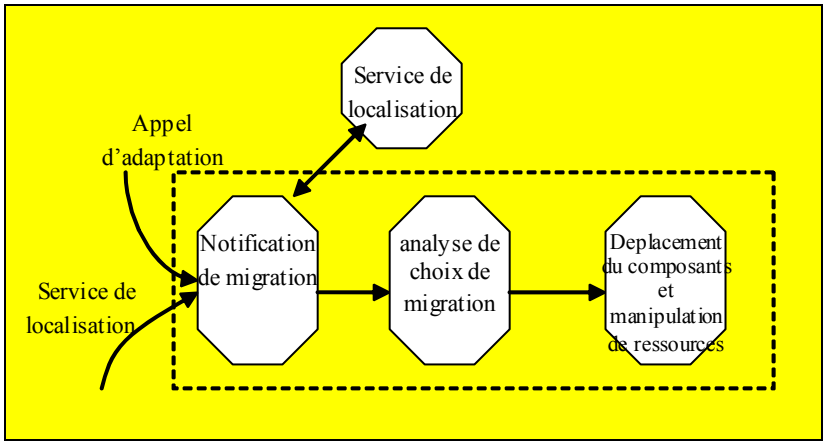

Figure 3.14 Architecture de service de migration

- **Dans la deuxième phase** : le service détermine le type de migration à appliquer selon les caractéristiques de l'EJB et les ressources et services qu'il demande. Le service distingue quatre types de migration possibles :

- Redirection de requête : Ce type consiste à satisfaire les requêtes en les redirigant vers l'EJB déployé sur un autre serveur de la fédération sans faire de migration. La redirection est appliquée quand l'EJB est fortement lié avec une ressource non transférable.
- Transfert libre : Ce type permet la migration de l'EJB sur un autre serveur d'applications. Ce type de migration est effectué quand l'EJB n'est pas lié avec d'autres ressources.
- Transfert multiple : Quand un EJB référence un autre EJB la migration peut être effectué d'une manière récursive. Ce type de transfert implique la migration de plusieurs composants EJB, pour chaque EJB un type de migration est choisi selon ses liens avec les ressources.
- Transfert avec ressource : Ce type de transfert est appliqué quand les ressources demandées par l'EJB sont transférable ou quand il est possible de gérer des liens à distant vers ces ressources.

La réalisation de ces types de transfert est détaillée en 4.5.

- **Dans la troisième phase** : on traite le processus de migration selon la source qui demande la migration. Dans le premier cas (appel d'adaptation), l'EJB est déplacé du serveur local à un serveur distant, et l'EJB est retiré du serveur local. Dans le deuxième cas (service de localisation), l'EJB est déplacé du serveur distant au serveur local, mais il n'est pas nécessairement retiré du serveur distant.

3.4.2.6. Conclusion

Dans la section 3.2.4 nous avons spécifié les trois phases principales du déploiement global que l'on applique dans la fédération, notamment la phase d'évaluation, la phase décisionnelle, et la phase d'adaptation.

Pour déployer un EJB sur la fédération, le déployeur peut déclencher le processus de déploiement depuis n'importe quel serveur d'applications dans la fédération car chaque serveur définit tous les services nécessaires pour effectuer un déploiement global.

La première phase de déploiement consiste à évaluer les besoins de l'EJB avec le service d'évaluation. Ce dernier renvoie en retour un rapport d'évaluation de l'EJB.

La deuxième phase concerne la prise de décision concernant le choix de placement de l'EJB. Cette décision est prise suite à une comparaison entre les besoins de l'EJB (indiqués dans le rapport d'évaluation) et l'état de chaque serveur dans la fédération. L'état d'un serveur est obtenu par le service d'instrumentation. Après avoir choisi le serveur le mieux placé pour déployer l'EJB, le service de

déploiement local propre au serveur choisi est contacté pour effectuer l'installation réelle.

La phase d'adaptation commence après l'installation des composants. Le service de suivi supervise les composants EJB déployés afin de détecter un changement de besoin. Le changement de besoin est signalé si le résultat obtenu suite à un appel de suivi ne correspond pas au résultat donné dans le rapport d'évaluation. Suite à un changement de besoin un acte d'adaptation est déclenché.

L'ensemble des services participe à la réalisation des trois phases de déploiement, mais le déploiement global et l'adaptation ne peuvent se faire sans organiser l'interaction entre les services. Il nous faut donc coordonner d'une part la communication entre les différents services et d'autre part la communication entre les serveurs d'applications. Cette fonctionnalité est assurée par le service de coordination détaillé dans la section suivante.

3.4.3. Service de coordination

Partant de notre principe de décentralisation, nous proposons d'ajouter le service de coordination du déploiement adapté dans tous les serveurs de la fédération. Ce service a deux rôles principaux :

- organiser la coopération entre les services de gestion et
- prendre les décisions collectives de déploiement.

3.4.3.1. Organisation de la coopération

Afin de déployer un EJB sur la fédération, l'ensemble des services de gestion doivent réaliser leur part de travail sans être dépendants les uns des autres. En d'autres termes, les services ne doivent pas communiquer directement entre eux ; la communication doit être transparente et réalisée par une tierce partie : le service de coordination. Ce type de coopération offre une plus grande flexibilité du déploiement permettant ainsi de réaliser des différents scénarios de déploiement, où par exemple l'ordre d'appels des services n'est pas forcément le même.

Notre service de coordination est donc capable de lire et d'appliquer différents scénarios de déploiement. Nous appelons un scénario de déploiement le *procédé de déploiement.*

Le but de l'utilisation des procédés est de permettre un déploiement dynamique. La technique du procédé a été présentée dans le domaine de l'ingénierie de logiciels [DER94, DER99], et il a été utilisé par [LES03] pour exprimer un processus de déploiement. Cette approche permet, à la place d'appliquer toujours la

même solution, d'avoir un moteur de lecture de solutions capable de réaliser les scripts des solutions fournies.

Le premier rôle du service de coordination du déploiement global est donc de fournir un moteur de déploiement capable d'exécuter des procédés de déploiement.

3.4.3.2. Prise de décisions

Au cours d'un déploiement, plusieurs décisions doivent être prises. Ces décisions concernent principalement : le choix du serveur d'hébergement et le choix d'adaptation.

Choix du serveur d'hébergement

En se basant sur le rapport d'évaluation d'un EJB et sur la connaissance de l'état de chaque serveur de la fédération, un choix de serveur doit être fait. Afin de faire ce choix, plusieurs acteurs doivent être considérés :

- contacter les serveurs de la fédération : est ce qu'on envoie le rapport d'évaluation à tous les serveurs de la fédération et on attend en retour un rapport de correspondance par serveur ? ou bien, on interroge directement le service d'instrumentation des autres serveurs pour obtenir leurs états ? et d'autre part, est ce qu'on interroge tous les serveurs dans la fédération ? ou bien, on interroge les serveurs un par un jusqu'à ce qu'on trouve le serveur qui peut héberger l'EJB ? en d'autres termes, est ce qu'on cherche un choix optimum ou un choix acceptable ?
- quand on obtient des réponses sur l'état des serveurs dans la fédération, sur quels critères faut-il se baser pour prendre la décision ? le serveur qui offre une meilleure performance ? le serveur où l'hébergement sera le moins coûteux ?

Choix d'actions d'adaptation

Après le déploiement des composants, le service de suivi continue à vérifier le comportement des EJB afin de détecter un dysfonctionnement. Suite au signalement d'un dysfonctionnement, le service de coordination du déploiement doit prendre une décision d'adaptation qui peut être :

- Redéployer l'EJB sur un autre serveur qui convient mieux à ses nouveaux besoins,
- Limiter les fonctionnalités de l'EJB (ex : limiter le nombre des clients, ou interdire l'accès à une méthode),
- Rejeter l'EJB (le retirer).

Dans la fédération, nous définissons les deux choix d'hébergement et d'adaptation avec *des règles de gestion*. Le rôle de ces règles est de spécifier les

décisions à prendre en respectant les politiques de la fédération. Par exemple, dans une fédération où tous les serveurs sont des partis de confiance, il est possible d'autoriser les serveurs à voir l'état des autres serveurs dans la fédération. Chaque serveur de la fédération peut donc définir ses propres règles.

3.5. Scénarios d'utilisation

Dans cette section nous montrons deux exemples d'utilisation de la fédération. Nous montrons les différences que l'on peut avoir dans l'utilisation de la fédération afin de montrer la nécessité d'une fédération suffisamment flexible pour permettre son utilisation dans différents contextes d'utilisation.

3.5.1. Scénario 1 : coopération d'entreprises

Avec l'évolutions de l'Internet et les technologies de développement d'applications web et E-Business, il est de plus en plus courant d'étendre et d'élargir le business sur le web. Il n'est plus suffisant pour une entreprise d'avoir son propre système d'informations et son interface web.

La coopération entre les systèmes informatiques des entreprises est connue sur le nom Business to Business ou (B2B). Ce type de coopération permet l'échange de services entre les entreprises. Une entreprise de voyage par exemple peut contacter plusieurs compagnies aériennes pour négocier le prix d'un billet d'avion. Ce type de coopération se limite aux consultations des services des entreprises en échangeant des informations. La coopération n'inclut pas l'utilisation des ressources matérielles ou logicielles.

La fédération que l'on propose peut être utilisée comme la base de coopération entre plusieurs entreprises.

La Figure 3.15 montre une fédération d'entreprises. On considère que chaque entreprise a son propre cluster de serveurs d'applications. La fédération se fait donc entre les clusters des entreprises. Dans cette fédération il suffit que le serveur administrateur du cluster implémente les services d'administration. La coopération se fait donc entre les serveur SA1, SA2, SA3 qui représentent les administrateurs des clusters.

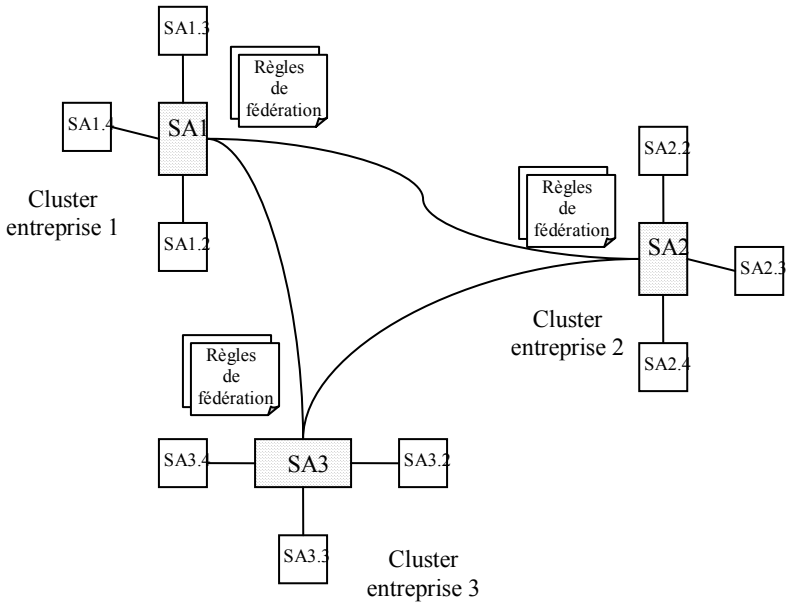

Figure 3.15 Fédération d'entreprises

3.5.2. Scénario 2 : entreprise d'hébergement d'applications

Dans le deuxième scénario, l'ensemble des serveurs d'applications appartenants à une entreprise d'hébergement d'applications forment une fédération. Le but de cette fédération est de faire coopérer les serveurs afin d'économiser les ressources de l'entreprise et de contrôler globalement toutes les applications.

Ce type d'entreprise offre un service d'hébergement d'applications en offrant l'espace disque, la mémoire, la CPU et d'autres ressources demandées. Selon les besoins des applications un contrat d'hébergement est rédigé entre le fournisseur de l'application et l'entreprise d'hébergement. L'entreprise est responsable de la maintenance des applications hébergées 24/24 tant que les applications respectent le contrat initial d'hébergement.

3.5.3. Points de différence entre les scénarios

On peut remarquer principalement deux différences entre les deux scénarios d'utilisation :

- La stratégie de déploiement : Dans le premier scénario chaque entreprise essaye de déployer ses applications localement avant de faire appel aux autres serveurs d'applications. Dans le deuxième scénario, il est plus performant de choisir le meilleur emplacement de l'EJB avant de le déployer.

- La prise de décisions : Dans le premier scénario, après la détection d'un dysfonctionnement, l'entreprise essaye de migrer l'EJB vers un autre serveur de façon à ce qu'il garde la même performance. Dans le deuxième scénario, un dysfonctionnement d'un composant peut signifier que l'EJB n'a pas respecté les termes qui ont été fixés au départ dans le contrat d'hébergement. Dans ce cas, l'entreprise peut décider de rejeter l'EJB tout simplement.

3.6. En résumé

Dans ce chapitre nous avons mis l'accent sur des points faibles de la plate-forme J2EE. Notamment nous avons précisé la limitation du déploiement qui ne traite que la distribution horizontale. Nous avons aussi montré le manque dans la notion des clusters qui, bien qu'elle traite le déploiement vertical, est loin d'être satisfaisante.

Nous avons en conséquence défini la notion de ***fédération*** qui représente un environnement de coopération entre serveurs d'applications. Son but est de surmonter les limitations de la plate-forme J2EE et de la notion de cluster en offrant aux composants EJB un environnement adaptable et en permettant un déploiement global.

Nous avons défini les caractéristiques de la fédération et analysé les parties nécessaires afin de la réaliser. Nous avons proposé une gestion non centralisée pour gérer le déploiement global. Pour appliquer la gestion non centralisée, nous avons défini une plate-forme de gestion à intégrer pour chaque serveur d'applications participant à la fédération. La plate-forme définit un ensemble de services qui participent à la gestion globale du déploiement. Nous avons ensuite défini des règles de gestion globale. Le but de ces règles est d'unifier une prise de décision globale par la majorité des serveurs de la fédération.

Dans le chapitre suivant nous allons détailler la première partie de notre modèle qui représente l'ensemble de services de la plate-forme de gestion proposée. Ensuite dans le chapitre 5 nous détaillerons la deuxième partie du modèle, le service de coordination, qui montre comment les serveurs de la fédération réalisent le déploiement global en se basant sur la plate-forme de gestion et en respectant les règles de gestion globale.

4. Mise en œuvre d'un système d'administration basé sur JMX

Dans ce chapitre nous détaillons notre proposition pour un système d'administration pour la gestion du déploiement dans une fédération de serveurs d'applications EJB. Nous présentons l'ensemble des services qui constituent cette plate-forme d'administration.

4.1. Service d'instrumentation à 3 niveaux

Le service d'instrumentation représente la source des informations pour la gestion des serveurs d'applications. Ce service se charge d'explorer l'environnement du serveur d'applications afin de fournir la base d'information sur laquelle tous les autres services de gestion dépendent.

Dans un environnement complexe comme les serveurs d'applications on distingue 3 couches qui nécessitent une instrumentation parce qu'elles participent toutes au fonctionnement des composants EJB. Ces couches sont : la couche système, la couche middleware et la couche d'applications.

4.1.1. Réalisation

Le service d'instrumentation permet d'unifier la vue des nœuds appartenant à la fédération. Une vue d'instrumentation d'un nœud expose les informations de gestion des trois niveaux d'environnement. On obtient donc trois couches d'instrumentation dans notre architecture.

Dans chaque couche on produit un ensemble de MBeans qui décrivent les ressources existantes dans cette couche et qui peuvent éventuellement influencer le déploiement des composants. Au niveau de la couche d'instrumentation de bas niveau on récupère des mesures concernant la CPU, le disque, et la bande passante ; au niveau de la couche middleware on considère les services standards de J2EE et les services additionnels ; et finalement au niveau de la couche d'applications on s'intéresse au cycle de vie des EJBs et au comportement des méthodes.

Pour chaque nœud dans la fédération une seule instance de MBeanServer est créée et tout les MBeans des trois couches sont enregistrés auprès de cette instance. Le MBeanServer est l'agent qui permet l'accès aux attributs et aux opérations de tous les MBeans enregistrés.

La Figure 4.1 montre le modèle de l'instrumentation basée sur JMX. Sur la couche la plus basse (système d'exploitation) on utilise NWS (cf 2.1.3.1) pour la collecte des informations des ressources. Sur la couche middleware, la spécification J2EE spécifie un modèle de gestion à respecter par les serveurs d'applications [JSR], ce modèle est implanté par JMX pour la plupart des fournisseurs de serveurs d'applications et même par les serveurs d'applications open source. Sur cette

couche, on étend le modèle afin d'inclure les services additionnels du middleware. Finalement, sur la couche d'applications on instrumente les EJBs déployés sur le serveur d'applications, notamment, le cycle de vie et les fonctionnalités de chaque EJB. Les trois couches de notre service d'instrumentation sont détaillées dans les 3 sections suivantes.

Pour résumer, les MBeans des trois niveaux sont enregistrés dans le même serveur JMX. On nomme les MBeans avec les critères de l'objet ObjectName. On définit le nom de domaine "*instrumentation"* pour regrouper tous les MBeans d'instrumentation des trois couches d'instrumentation. Chaque MBean est ensuite défini par deux propriétés. La première propriété est la *couche (layer),* elle peut avoir trois valeurs (application, middleware, system). La deuxième propriété est *ressource* et elle définit un nom simple de la ressource instrumentée présentée par le MBean.

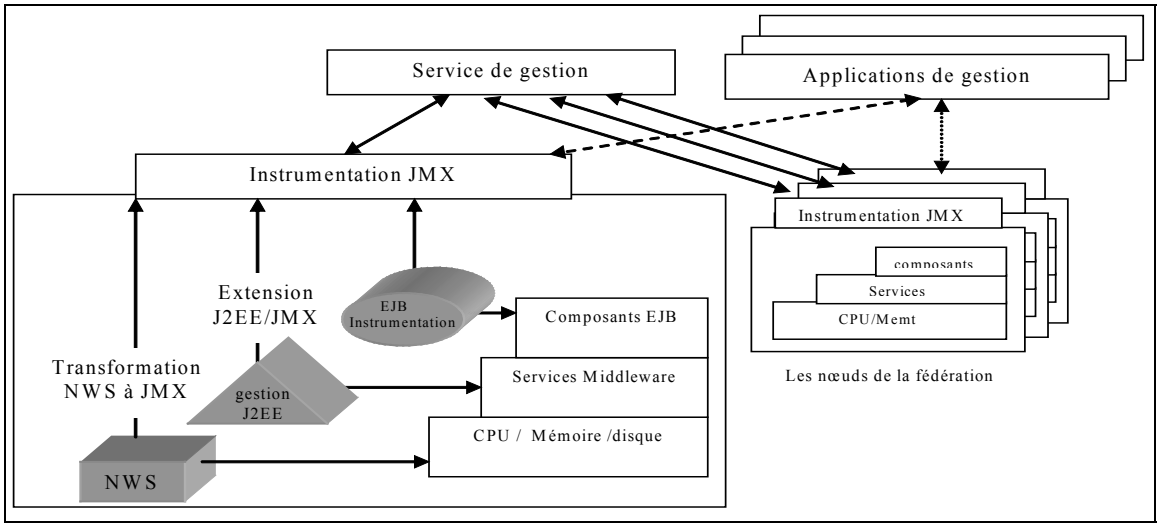

Figure 4.1 Modèle d'instrumentation pour une fédération de serveurs d'applications

4.1.1.1. Couche réseau et système

Avoir une bonne connaissance sur le réseau, le système et les capacités de chaque nœud de la fédération joue un rôle important pour déterminer le meilleur emplacement d'un EJB dans la fédération. Le choix de l'emplacement d'un EJB dépend de ses besoins comme par exemple l'usage intensif de bande passante sur le réseau, ou l'utilisation de la CPU.

De même, l'instrumentation de l'activité de la CPU, du disque, et de toutes les autres ressources système permet au système d'administration d'évaluer le taux de consommation des ressources accordées à chaque composant après le déploiement. Le système d'administration se base sur ces informations afin de

prendre la décision de garder le composant dans le nœud actuel ou de le redéployer sur un autre nœud du la fédération.

Pour l'instrumentation de cette couche on collecte les informations des ressources à l'aide de NWS (cf 2.1.3.1), qui mesure la fraction de disponibilité de la CPU pour un nouveau processus, la latence et la bande passante du réseau. Pour unifier l'interface d'instrumentation des trois couches, on fait remonter les mesures obtenues par NWS dans une interface JMX. On a donc mis en correspondance NWS sur JMX (Figure 4.2).

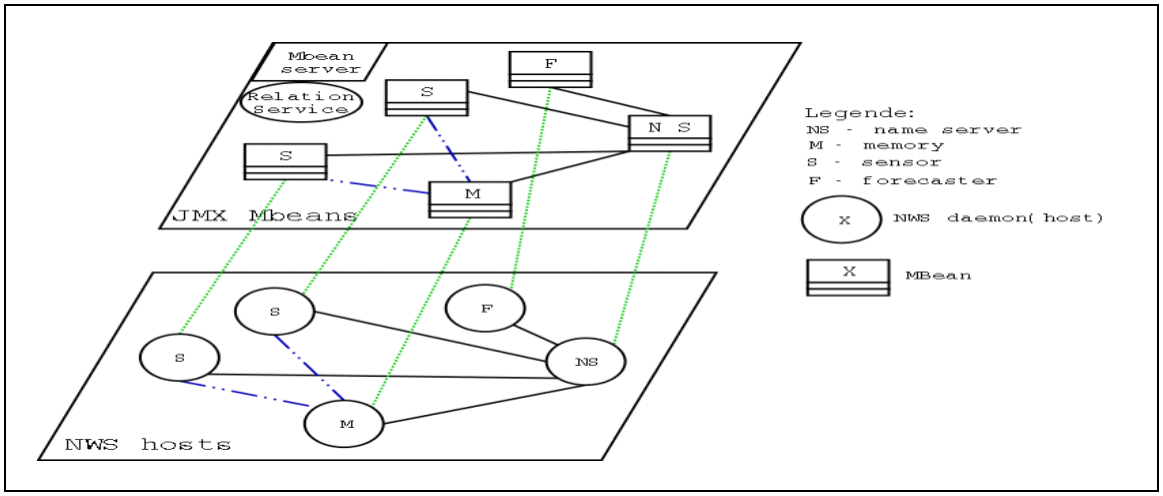

Figure 4.2 La correspondance NWS à JMX

Pour faire la correspondance NWS – JMX, chacun des 4 éléments NWS a été représenté par un MBean :

- Un MBean JmxNWSNameServer: il instrumente le processus de nommage. Ce MBean dispose d'une méthode permettant d'interroger librement le service de nommage et d'une méthode reconstruisant l'architecture de MBean correspondant aux éléments NWS participant à la fédération.
- Un MBean JmxNWSMemory : il instrumente le processus de stockage. Ce MBean permet l'accès à la mémoire de NWS.
- Un MBean JmxNWSSensor : il représente les sensors de NWS. Il fournit des méthodes d'arrêt et de démarrage d'une activité de collecte d'information. Les informations collectées sont rapatriées à l'instanciation du MBean JmxNWSMemory.
- Un MBean JmxForecaster : il instrumente la partie qui fait des prédictions dans NWS.

Ces quatre MBean héritent d'un MBean abstrait JmxNWSHost qui permet de remonter les informations (adresse, port), ainsi que les opérations communes à tous les hôtes (démarrer, arrêter).

Chaque nœud de la fédération peut instancier plusieurs MBeans. Un nœud doit avoir au moins un MBean JmxNWSSensor afin d'instrumenter la couche système d'exploitation. La Figure 4.3 montre le diagramme UML qui représente les MBeans de correspondance de NWS.

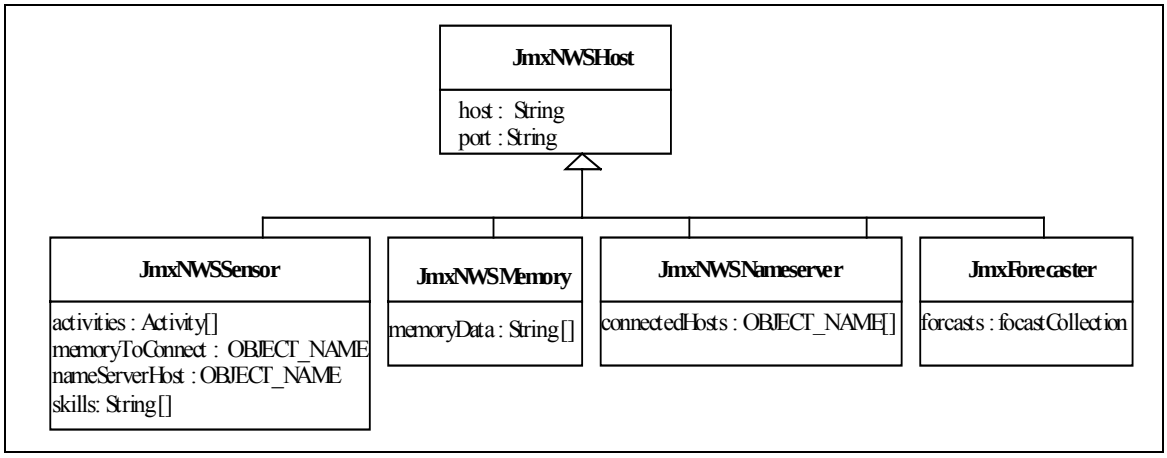

Figure 4.3 Répresentation UML des MBeans dans la couche système d'exploitation

Tout les MBeans créés dans cette couche sont enregistrés dans le MBeanServer afin de permettre aux autres services de gestion d'accéder aux informations exposées par les MBeans.

4.1.1.2. Couche middleware :

Chaque nœud dans la fédération contient un serveur d'applications qui héberge des composants EJB. Le serveur d'applications fournit aux EJB les services standards de la plate-forme J2EE et des services additionnels intégrés. Ces derniers peuvent varier d'un serveur à l'autre dans la fédération. L'instrumentation à ce niveau consiste à instrumenter les deux types de services (standards et additionnels).

Dans [JSR77] Sun définit la spécification d'un modèle d'information de gestion pour la plate-forme J2EE. Ce modèle spécifie tous les éléments de la plate-forme J2EE à instrumenter. La spécification repose sur l'utilisation de JMX pour la gestion des éléments du modèle. Le modèle n'est pas encore complet mais les serveurs d'applications ont intégrés JMX afin de respecter les spécifications de gestion.

Nous étendons le modèle présenté par Sun afin de prendre en considération les services additionnels. Nous considérons principalement deux services additionnels qui permettent l'interfaçage entre cette couche d'instrumentation et les

deux autres couches : niveau réseau/système et niveau composants. La Figure 4.4 montre le diagramme UML de la partie étendue du modèle J2EE de gestion.

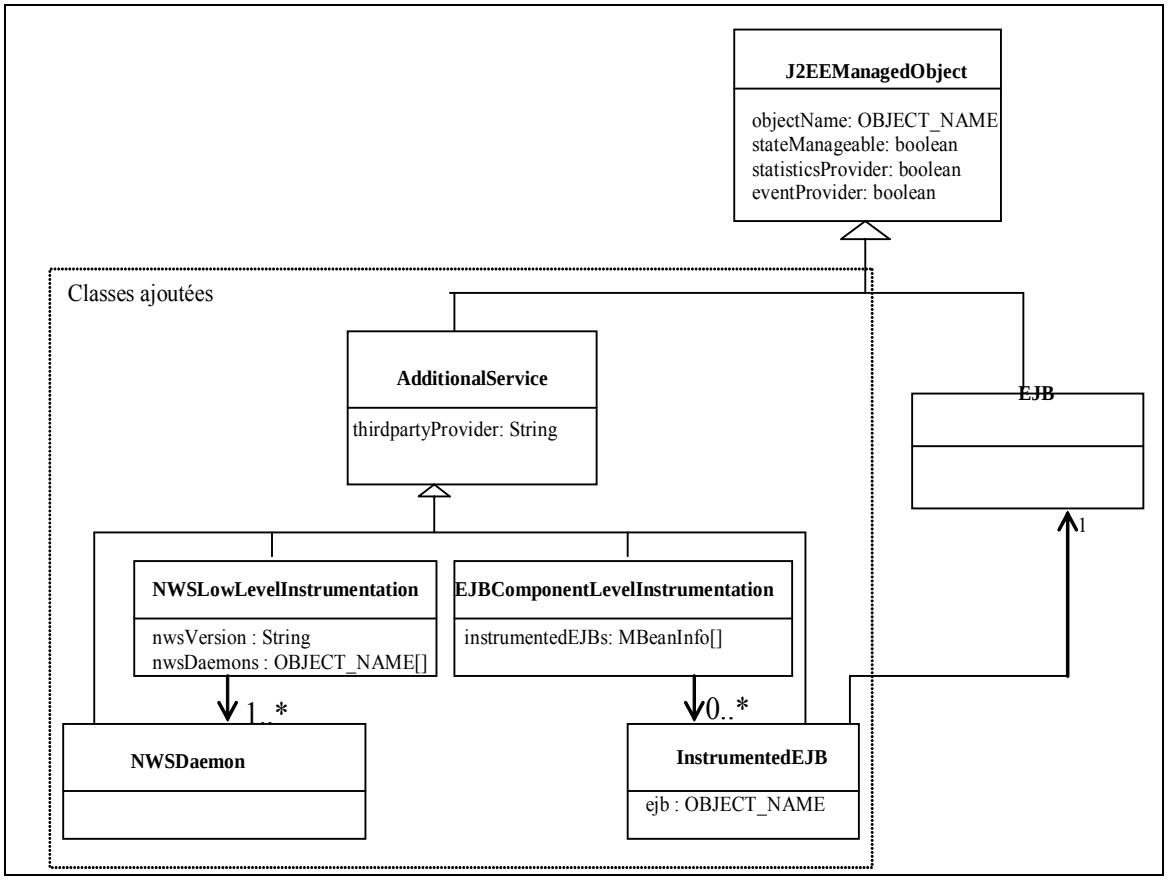

Figure 4.4 Représentation UML de l'extension du modèle de gestion J2EE

L'intégration des services additionnels au niveau du serveur d'applications n'est pas toujours évidente. Les serveurs d'applications commerciaux ne le permettent pas, tandis que les serveurs open source ne suivent pas la même approche d'intégration de services [NAN03]. Pour intégrer un service dans Jonas, il faut l'implémenter avec une interface spécifique à Jonas (org.objectweb.jonas.service.Service). Il faut ensuite modifier le fichier de configuration de jonas *jonas.properties* afin d'informer jonas du nouveau service, ce qui implique l'arrêt et la relance du serveur. JBoss par contre permet une intégration dynamique de nouveaux services. Il permet de déployer un nouveau service avec un descripteur de déploiement et une archive *.sar*. L'archive contient un MBean qui représente le nouveau service.

Quelle que soit l'implantation du service, chaque service additionnel doit être représenté par un MBean qui le représente dans l'interface d'instrumentation au niveau serveur d'applications. Une référence vers le service additionnel doit aussi

être lié dans l'arbre JNDI afin de permettre l'utilisation du service par les EJB d'une manière explicite.

4.1.1.3. Couche applications

A la différence des applications isolées, les applications distribuées à base de composants sont utilisées par plusieurs sessions clients en même temps, et elles sont supposés être disponibles en continu (24h/24). Ces exigences impliquent un contrôle total sur tous les composants, leurs comportements, leurs fonctionnalités, et leurs cycles de vie afin de gérer leurs création, suspension et réactivation.

La disponibilité d'un composant dépend de la disponibilité des ressources qu'il utilise. Pour un EJB qui déclare au moment du déploiement qu'il a besoin d'un certain taux de disponibilité d'une ressource pour atteindre la performance requise, le rôle de la fédération est d'assurer le taux de disponibilité de la ressource pour l'EJB. La fédération instrumente alors chaque EJB durant son cycle de vie afin de vérifier sa bonne performance et le cas échéant d'en déterminer la source.

Nous accomplissons ce contrôle sur les EJB, en instrumentant tous les EJB déployés sur un nœud. Au moment du déploiement, nous créons une interface d'instrumentation pour chaque EJB. Cette interface permet de récupérer des informations concernant le cycle de vie de l'EJB et ses fonctionnalités.

Le problème principal de l'instrumentation d'une application est la détermination de l'emplacement du code à instrumenter dans le code d'origine [KAT99]. Ce processus demande souvent une intervention humaine afin de choisir les parties cruciales du code. Dans les EJB, on pourrait définir ces parties depuis les deux interfaces de l'EJB : ***home*** et ***component***. Depuis l'interface home on peut identifier les méthodes de cycle de vie, et les méthodes fonctionnelles de l'EJB sont identifiées par l'interface component.

On crée pour chaque EJB un MBean qui le représente. Ce MBean est créé à l'aide du service de suivi des EJB (cf 4.3). Tout les MBeans de cette couche sont enregistrés dans le MBeanServeur pour exposer leurs attributs et opérations aux autres services de gestion.

4.1.2. Conclusion

Le service d'instrumentation est le cœur de notre plate-forme d'administration car il fourni, en plus d'une interface d'instrumentation, les outils de standardisation de la gestion. En effet, dans notre fédération nous implantons tous les services de gestion en tant que MBean.

Nous considérons que ces services de gestion se situent parmi les services additionnels au niveau middleware. Ils sont donc instrumentés, et ils peuvent communiquer localement en passant par le MBeanServeur.

4.2. Service d'évaluation des EJB

Le service d'évaluation permet de mesurer les besoins d'un EJB et de produire un rapport qui est utilisé pour prendre un choix de déploiement adapté sur la fédération.

Nous détaillons dans cette section la réalisation des deux phases du service : la phase de détection des ressources middleware et la phase d'évaluation des ressources système.

4.2.1. Détection des ressources au niveau middleware

Un EJB peut utiliser plusieurs ressources qui offrent des services qui ne sont pas spécifiés d'une manière standard. Ces ressources existent sur certains serveurs de la fédération mais pas forcement sur tous.

Afin de déployer l'EJB sur un serveur qui offre le service, nous procédons à une détection des ressources demandées par l'EJB. Dans le fichier de déploiement ejb-jar.xml, chaque EJB précise les ressources qu'il référence dans la balise <resource-ref>. La spécification des EJB précise que les EJB peuvent utiliser les ressources suivantes : DataSource, JMS (Java Message Service) [JMS], URL, JavaMail. L'exemple suivant montre comment un EJB déclare dans le fichier ejb-jar.xml une référence vers un pool de connexions à une base de données (objet DataSource).

```
<resource-ref>
     <description>
          Un objet DataSource pour la connexion à la base de
          données.
     </description>
     <res-ref-name>jdbc/EmployeeAppDB</res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
</resource-ref>
```

La balise <res-ref-name> indique le nom JNDI de l'objet de connexion à la base de données. La balise <res-type> indique la classe de l'objet lié au nom JNDI. La balise <res-auth> indique que le serveur d'application est autorisé à accéder à la ressource (la base de données).

Dans notre fédération, il est possible qu'un serveur déclare d'autres ressources additionnelles, les EJB peuvent donc utiliser cette même balise pour déclarer la ressource additionnelle. L'exemple suivant montre la déclaration d'une ressource additionnelle, un moteur de reconnaissance vocale.

```
<resource-ref>
     <description>
          Un objet représentatif d'un moteur de reconnaissance
          vocale.
     </description>
     <res-ref-name>servicesAdd/InterfaceVocale</res-ref-name>
     <res-type>dart.reconnaissanceV.ConnexionV</res-type>
     <res-auth>Container</res-auth>
</resource-ref>
```

Une analyse du descripteur de déploiement nous permet de connaître les ressources appelées par l'EJB et en conséquence de choisir le serveur qui supporte le service additionnel requis.

4.2.2. Evaluation des ressources système

Principalement nous évaluons un EJB selon la consommation de mémoire et de temps CPU. Nous avons établi un modèle d'estimation théorique, et nous avons réalisé un modèle de validation. Avant de présenter chacun des modèles nous présentons le fonctionnement d'un serveur d'applications au dessus de RMI (cf 4.2.2.1). Cette vision nous sert pour la présentation du modèle théorique (cf 4.2.2.2) et du modèle de validation (cf 4.2.2.3).

4.2.2.1. Serveur EJB au dessus de RMI

RMI [RMI, PIT01] est la base de communications entre un serveur d'applications EJB et un client. Le rôle de RMI est de fournir le mécanisme d'appel de procédure distante (Remote Procedure Call.) Il garantit que les appels de

méthodes des clients sont envoyées au serveur d'EJB et que les valeurs de retour du serveur atteignent le client.

Du côté serveur d'applications, il existe un pool fini de threads applicatifs. Le serveur ouvre un thread de connexion "RMI-connection" pour chaque client. Pour chaque connexion, le serveur relie un thread de connexion client à un thread applicatif. Si le nombre de threads de connexions des clients excède le nombre de threads d'applications disponibles, les threads de connexions RMI seront permutés afin d'obtenir l'accès aux thread d'applications. Tous les threads d'applications ont la même priorité et il n'y a aucun traitement préférentiel pour aucun d'entre eux. Cependant, les threads d'applications sont des threads Java et ils correspondent à des processus légers de la JVM. Ces processus sont mis en correspondance avec des threads système et sont exécutés en « round robin »[WOL96]. Tous les clients simultanés vont alors pouvoir profiter du temps CPU. En conséquence, nous pouvons modéliser un serveur d'EJB et ses clients concurrents dans une file d'attente de N threads simultanées (Figure 4.5).

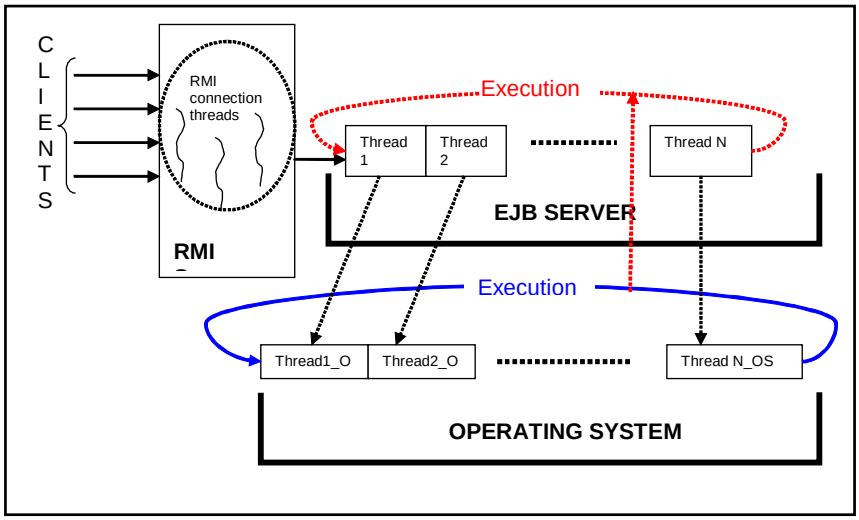

Figure 4.5 Modeler le serveur d'applications et les clients à un queuing système

Ces threads représentent l'exécution des méthodes d'un des composants EJB déployés sur le serveur. Nous nous basons sur ce mécanisme de threads pour réaliser un modèle de prédiction de la consommation CPU et mémoire par les EJB.

4.2.2.2. Modèle théorique

Nous représentons dans cette section un modèle pour calculer une estimation pour la consommation de mémoire et de CPU. Afin d'établir notre modèle nous avons supposé qu'un EJB déclare :

- le nombre maximum de clients simultanés appelant l'EJB,

- la méthode qui consomme le plus de mémoire (on l'appelle la méthode Memoire()),
- la méthode qui consomme le plus de CPU (on l'appelle la méthode CPU()).
- les valeurs des paramètres à passer aux méthode Memoire() et CPU()

Les méthodes Memoire() et CPU() doivent être des méthodes parmi les méthodes exposées dans les interfaces component ou home.

Dans nos calculs, nous considérons les EJB de type stateful et de type entité. Car ces deux types sont les types qui associent une instance par client. Tandis que les EJB stateless sont des EJB qui peuvent être partagés par plusieurs clients.

Estimation Mémoire

En considérant que ***Mem*** est la quantité de mémoire consommée par la méthode Memoire() d'un composant ***ejb***, et que ***Mi*** est la mémoire occupée par les attributs de l'instance de l'ejb, et que ***Mc*** est la mémoire occupée par la classe, et que N_{ejb} est le nombre des clients simultanés de l'ejb, alors la consommation totale de mémoire suite à l'instanciation d'un ejb par N clients est :

$$\text{Total\_ejb\_mem} = (N_{ejb} \times Mem) + (N_{ejb} \times Mi) + Mc \qquad (1)$$

Pour K composants déployés sur un serveur d'applications, la quantité maximum que le serveur d'applications alloue est :

$$\text{Max\_serv\_mem} = \sum_{j=1}^{k} \text{Total\_ejb\_mem} \qquad (2)$$

L'équation (1) montre la dépendance linéaire entre le nombre des clients simultanés d'un EJB et la quantité totale de la mémoire consommée par l'EJB.

Mem, Mi et Mc peuvent être calculé en déployant l'EJB sur un serveur de test (cf 4.2.2.3) afin de calculer total_ejb_mem.

Estimation CPU

Le but de l'analyse de la CPU est d'identifier l'influence de cette ressource sur l'exécution du serveur et des composants EJB afin de choisir le bon emplacement pour deployer l'EJB. On mesure la consommation CPU par le temps d'exécution.

Dans la suite de cette section nous montrons que notre modèle d'estimation du temps d'exécution CPU est le même que n'importe quel système de type d'attente de type round-robin car il se base sur RMI. Nous prouvons que le temps de réponse

d'un EJB sur un serveur d'applications dépend seulement de ses propres caractéristiques et du nombre total de clients connectés. Si nous connaissons le temps d'exécution d'un seul client exécutant l'EJB, nous pouvons déduire le temps d'exécution de plusieurs clients simultanément connectés.

Nous considérons que nous avons N clients simultanés qui accèdent au serveur, ceci signifie que dès qu'un thread finira son exécution, un nouveau prendra son relais. Cette supposition est faite afin d'assurer que nous analysons toujours le plus mauvais temps d'exécution avec un nombre constant de clients (N). Avec cette supposition on est certain que l'exécution d'un thread est faite du début à la fin en présence de N-1 autres threads.

Soit les notations suivantes :

k → indice variable des threads clients : 1<=k<=N

S_j → la quantum CPU pour un serveur j ($S_{isolated}$ représente cette valeur pour un serveur spécifique isolé de la fédération)

i_{kj} → le nombre de cycles que le $k^{ème}$ thread utilise pour accomplir sa tâche sur le serveur j. ($i_{kisolated}$ représente cette valeur pour un serveur isolé)

Dès que la queue est en état stable, le temps t_k nécessaire pour le $k^{ème}$ thread pour terminer l'exécution (dans la présence de N-1 threads) est :

$$t_k = i_{kj} \times S_j \times N$$

Même si on ne connait pas $i_{kj} \times S_j$, ce produit représente le temps d'exécution du $k^{ème}$ thread dans une condition d'isolation (exécuté seul sans aucun autre thread).

Ce produit ($t_{isolated} = i_{kisolated} \times S_{isolated}$) est calculé sur un serveur de test isolé de la fédération et il est utilisé comme valeur de base pour l'estimation de la consommation CPU. Ces deux formules sont des résultats habituels des systèmes à file d'attente. Ils signifient que :

1. Si on connaît la méthode qui consomme le plus de temps CPU pour un EJB, on peut trouver une mesure standard qui représente d'une manière intrinsèque l'EJB. Cette mesure peut être évaluée sur un serveur isolé. Si on connaît aussi le facteur de correspondance entre S_j et $S_{isolated}$ on peut déduire le temps d'exécution de cet EJB sur n'importe quel serveur dans la fédération.
2. Le temps d'exécution que l'on calcule dépend du nombre total de clients connectés au serveur. Ce résultat est utilisé pour évaluer le temps d'exécution au pire cas. Par exemple, si ejb_1 déclare un nombre

maximum de 100 clients, et que ejb_2 déclare un nombre maximum de 150 clients, et que ejb_3 déclare 150, alors le pire temps d'exécution est $t_{kj} = i_{kj} \times s_j \times 400$ ($i_{kj} \times s_j$ est un produit connu en serveur k).

Si on considère qu'une transaction est une requête complet d'un client, et que le $k^{ème}$ client émet la même requête d'une manière permanente au serveur, alors le nombre de transactions Tx qui peuvent être accomplies pour k clients en intervalle T est :

$$Tx_k = T/(i_{kj} \times s_j \times N) = T/(t_{k\_isolated} \times N)$$

Si on remplace $T/ t_{k\_isolated}$ par C_k, on obtient: $Tx_k = \frac{C_k}{N}$

La Figure 4.6 montre que le nombre de transactions pour un client pour un EJB évolue avec le nombre total de clients simultanés. On définit V_{klimit} comme la valeur minimum de Tx pour laquelle un client est satisfait du temps de réponse. N_{klimit} représente le nombre total de clients pour lesquels les clients peuvent avoir un temps de réponse égal à V_{klimit}.

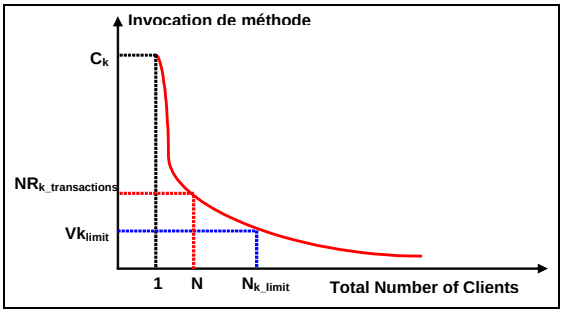

Figure 4.6 Evaluation de la CPU en terme de transaction

4.2.2.3. Modèle de validation

Nous avons défini une architecture de benchmarking afin de compléter et valider le modèle mathématique. L'architecture est composé principalement de 2 éléments : un élément de stress qui représente le côté client, et un système de mesure qui collecte les mesures du côté serveur.

Nous nous sommes basé sur quelques hypothèses afin d'établir le modèle de validation :

- Nous considérons qu'un EJB fournit au moment du déploiement la méthode qui consomme le plus de mémoire (Memory()) et qui utilise le plus de temps CPU (CPU()).
- Nous considérons également qu'à chaque moment de la durée de l'évaluation il y a N clients qui font des requêtes simultanées au serveur.

Selon les mesures que l'on veut obtenir, on configure l'élément de stress pour appeler la méthode Memory() ou la méthode CPU() de l'EJB. Du côté serveur, l'élément de collecte des mesures est implanté comme un EJB stateless appelé InstrumentEJB. Il y a seulement une instance de cet EJB afin de réduire la surcharge de cet EJB. Cet EJB a le rôle de calculer le temps d'exécution pour l'appel de la méthode CPU() et de calculer la quantité de mémoire pour la méthode Memory().

Mesures de mémoire

Afin de simuler N clients simultanés, nous modifions la méthode Memory() pour marquer à la fin une pause qui nous permet d'assurer la simultanéité des clients. L'élément de stress du côté client peut alors lancer les N clients d'une manière successive. Après le lancement de chaque client l'élément InstrumentedEJB collecte les mesures de mémoire.

Nous avons effectué une expérimentation afin de valider notre modèle mathématique. Dans cette expérimentation nous avons considéré deux composants EJB1 et EJB2 avec les valeurs suivants en Kilo octets :

	Mc	Mi	Mem
EJB1	0,5	0,5	1
EJB2	1	1	2

Tableau 4.1 Mesures de mémoire pour EJB1 et EJB2

On considère $C_{k,i}$ le $k^{ème}$ client qui stresse les deux EJB. L'indice i indique l'EJB appelé (1 ou 2). On lance 5 clients pour les deux EJB avec l'ordre suivant $\{C_{1,1}, C_{2,1}, C_{3,2}, C_{4,1}, C_{5,2}\}$. Suite au lancement de ces clients on obtient le résultat en Figure 4.7 qui montre l'évolution de la mémoire du côté serveur après le lancement de chacun des cinq clients.

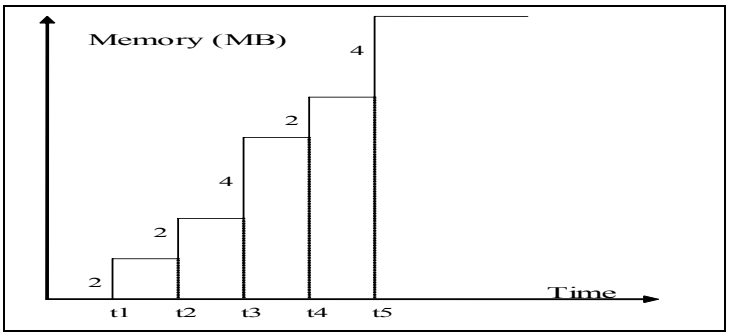

Figure 4.7 Evolution de la mémoire

Ce résultat montre la correspondance entre le modèle théorique et le modèle de validation et confirme donc les formules établies pour l'estimation de la mémoire.

Mesures de la CPU

Nous avons effectué 3 expérimentations afin de valider les formules établies pour l'estimation de la consommation de la CPU.

Dans la première expérimentation, nous considérons un seul client qui effectue 30 appels vers la méthode CPU() d'un EJB. A chaque appel, le temps d'exécution est calculé par l'élément InstrumentedEJB. En calculant la moyenne des valeurs obtenues, on obtient la valeur de 352ms. La Figure 4.8 représente tous les résultat obtenus et la moyenne calculée.

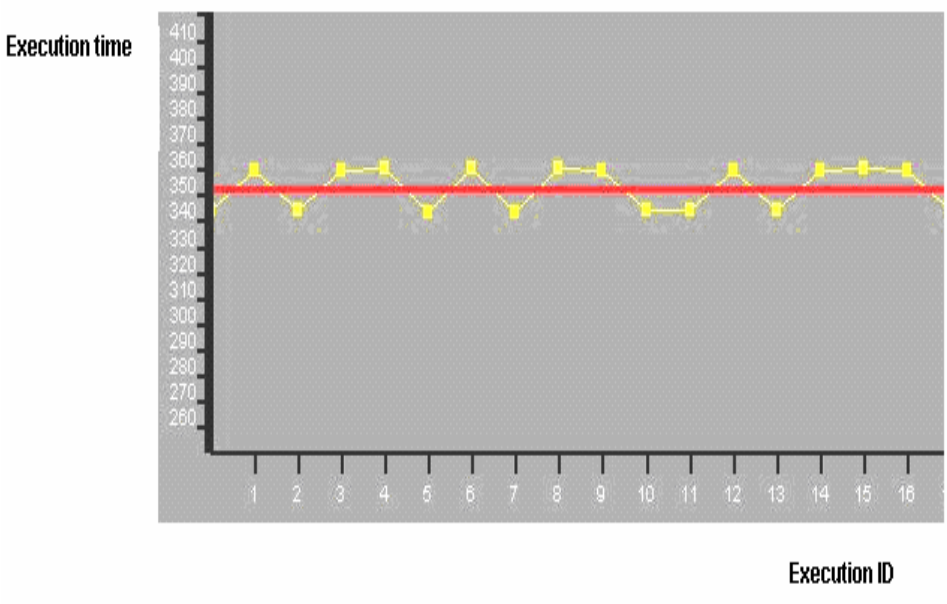

Figure 4.8 Un seul client exécutant 30 fois la méthode CPU()

Dans la deuxième expérimentation on considère 20 clients simultanés appellant chacun 30 fois la méthode CPU() d'un EJB. Pour chaque client i on obtient un ensemble de valeurs Vi représentant les 30 valeurs calculées pour le temps d'exécution. La Figure 4.9 représente les trois courbes obtenues.

La courbe en bas représente le temps d'exécution nominal de t0=352ms pour un seul client. Les deux courbes en haut représentent respectivement les valeurs réeles du temps d'exécution obtenues du modèle de validation et les valeurs obtenues par le modèle théorique pour les 20 clients (t=20×t0).

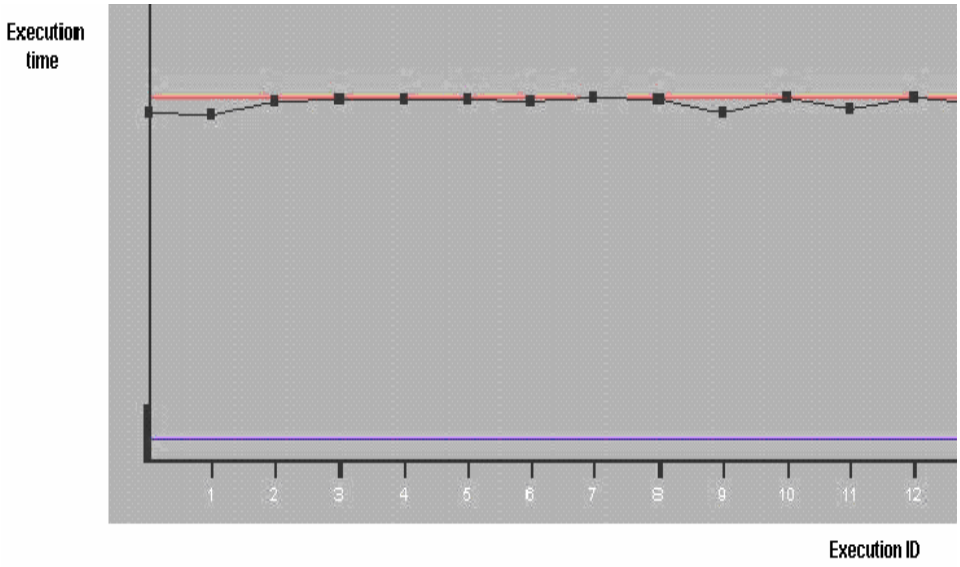

Figure 4.9 20 clients exécutant chacun la méthode CPU() 30 fois

La dernière expérimentation montre que le temps total d'exécution est influencé par le nombre total de clients connectés au serveur indépendamment de type d'EJB appelé.

Dans cette expérimentation nous avons deux composants déployés ejb_1 et ejb_2. On considère que les n=20 clients vont solliciter ejb_1 et ejb_2 dans une distribution équivalente (10 par 10) en appelant la méthode CPU() 30 fois. Pour chaque client on obtient Vi représentant les 30 calculs du temps d'exécution.

Les courbes en Figure 4.10 montrent que le temps de réponse est seulement influencé par son temps d'exécution et le nombre total des clients simultanés. Les 10 premiers valeurs représentes les résultats obtenus pour les clients du composant ejb_1 et les 10 autres valeurs représentent les résultats obtenus pour les clients du composant ejb_2.

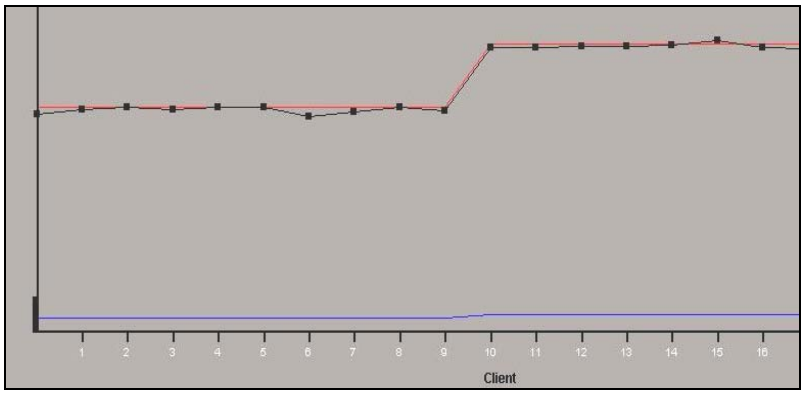

Figure 4.10 20 clients pour deux composants EJB1 et EJB2

Après avoir validé notre modèle théorique d'estimation de la consommation de la CPU, nous avons adopté le modèle théorique pour mesurer la consommation de CPU et de mémoire par chaque EJB. Les valeurs de base d'un EJB isolé doivent néanmoins être calculées sur un serveur de test isolé.

4.2.3. Génération d'un rapport d'évaluation

A la fin de l'évaluation d'un EJB, le service d'évaluation génère un rapport XML d'évaluation. Ce rapport comporte essentiellement deux parties. La première partie décrit les ressources additionnelles nécessaires demandées par l'EJB au niveau middleware. La deuxième partie décrit les ressources au niveau système, notamment la CPU et la mémoire. Pour chacune des ressources on décrit les valeurs significatives qui indiquent le taux de consommation de la ressource. La Figure 4.11 montre un exemple de rapport d'évaluation.

Le rapport d'évaluation est écrit en XML. Dans les parties décrivant les méthodes java nous avons suivi la présentation JavaML [BAR00]. Le rapport montre l'évaluation effectuée pour chaque EJB. Ces évaluations ont été générées après l'instanciation d'un EJB avec les valeurs des paramètres de création. Les valeurs d'évaluations de la CPU ont été générées après l'appel à une méthode qui fait des calculs appelée *calculate* qui prend en paramètre un *int*. La méthode compareRes représente la méthode qui consomme le plus de mémoire. Elle est appelée avec les paramètres "file0012.res" et "file0013.res"

```
<?xml version="1.0" ?>
<!DOCTYPE Evaluation SYSTEM "Evaluation.dtd">
<Evaluation>
  <EJB name="Calculation">
    <create-arguments>
      <create-argument name="base">
        <type name="int">
        <value name="10">
      </create-argument>
    </create-arguments>
    <System>
      <CPU>
        <method name="calculate" visibility= "public">
          <type name ="void">
          <formal-arguments>
            <formal-argument name="i">
              <type name="int"/>
              <value name="3945">
            </formal-argument>
          </formal-arguments>
          <evaluations>
            <evaluation name="T-isolated" value="234">
            <evaluation name="Vk-limit" value="45">
            <evaluation name="Nk-limit" value="15">
          <evaluations>
        </method>
      </CPU>
      <Memory>
          <method name="compareRes" visibility= "public">
          <type name ="void">
          <formal-arguments>
            <formal-argument name="first">
              <type name="String"/>
              <value name="file0012.res"/>
            </formal-argument>
            <formal-argument name="second">
              <type name="String"/>
              <value name="file0011.res"/>
            </formal-argument>
          </formal-arguments>
          <evaluations>
            <evaluation name="mem" value="512">
            <evaluation name="mi" value="120">
            <evaluation name="mc" value="200">
          <evaluations>
        </method>
      <Memory>
    </System>
    <Middleware>
      <res-ref-name>servicesAdd/InterfaceVocale</res-ref-name>
      <res-type>dart.reconnaissanceV.ConnexionV</res-type>
      <res-auth>Container</res-auth>
    </Middleware>
  </EJB>
</Evaluation>
```

Figure 4.11 Exemple de rapport d'évaluation

4.3. Service de suivi des besoins des EJB

Afin de superviser le comportement des composants EJB, le serveur d'applications instrumente tous les EJBs déployés. Le but du service de suivi est d'instrumenter les EJB dès leur déploiement sur le serveur et de réaliser des appels de suivi. Ces appels consistent à vérifier le fonctionnement des EJBs et à détecter une dégradation de performances.

Afin d'instrumenter les EJBs, le service de suivi crée pour chaque EJB un MBean qui le représente dans l'interface d'instrumentation du serveur. Cette instrumentation se fait d'une manière automatique et transparente.

4.3.1. MBean représentant et descripteur d'instrumentation

Le rôle du représentant MBean d'un EJB est d'exposer les opérations de gestion pour l'EJB. Ces opérations sont définies dans un descripteur d'instrumentation permettant de créer le MBean.

Nous choisissons de créer le descripteur d'instrumentation en XML car il permet la présentation des informations structurées d'une manière neutre indépendante du langage de programmation et facilement lisible par les applications. Il permet aussi la standardisation de la présentation des informations dans les differents domaines [ANA01].

Dans le descripteur d'instrumentation nous définissons les parties de code à instrumenter pour un EJB. Le descripteur est structuré principalement au trois parties : la description du code du cycle de vie, la description du code métier, et la description des opérations de déploiement de l'EJB.

La description du code de cycle de vie se situe dans l'interface home de l'EJB qui définit les méthodes de création, de suppression et de localisation des instances de l'EJB. L'instrumentation de cette partie de code permet de tracer les instances de l'EJB durant leur cycle de vie, de connaître le nombre d'instances créées et leurs états.

Le code métier de l'EJB est retrouvé dans l'interface remote de l'EJB qui décrit les méthodes fonctionnelles qui peuvent être appelées par un client. L'instrumentation de ces parties de code permet de détecter le comportement métier de l'EJB comme par exemple de trouver le nombre de fois qu'une méthode est appelée afin de trouver la méthode la plus utilisée par les clients, ou de tracer le temps de réponse d'une méthode, ou de trouver l'instance la plus active.

La troisième partie du descripteur concernant le déploiement de l'EJB est extraite du fichier de déploiement ejb-jar.xml de l'EJB. Cette partie ne représente pas un morceau de code de l'EJB à instrumenter, mais elle définit des informations nécessaires à la création du MBean représentant l'EJB comme le nom JNDI et les autre EJB référencés.

La Figure 4.12 montre un exemple de descripteur d'instrumentation d'un EJB "HelloWorld".

```
<?xml version="1.0" ?>
<!DOCTYPE MBean SYSTEM "MBean.dtd">
<MBean name="HelloWorld">
  <BusinessMethods>
      <method name="displayMessage"
              id="0"
              abstract="false"
              static="false"
              synchronized="false"
              volatile="false"
              transient="false">
            <type>
                  String
            </type>
            <throws/>
      </method>
  </BusinessMethods>
  <LifeCycleMethods>
  …
  </LifeCycleMethods>
  …
</MBean>
```

Figure 4.12 Exemple de descripteur d'instrumentation XML

Le MBean représentatif de l'EJB est alors créé en se basant sur les informations fournies par ce descripteur. Comme tout composant MBean, le MBean que l'on crée est composé d'une classe java et de son interface de gestion. La classe agit comme un client de l'EJB ; elle récupère l'objet "home" et crée une instance de l'EJB. Cette instance sert à donner des évaluations sur la consommation des ressources par l'EJB, on appelle cette instance l'instance d'évaluation. La classe du MBean contient aussi des méthodes qui ont les mêmes signatures que les méthodes de l'EJB (les méthodes métiers et les méthodes de cycle de vie). Ces méthodes d'emballage font appel aux méthodes de l'EJB à l'aide de l'instance d'évaluation. Le retour de ces méthodes est le même que celui des méthodes d'origines. Toutes les

méthodes d'emballage effectuent des calculs de mesures d'évaluation qui permettent de détecter une dégradation de performance de l'EJB. Ces mesures sont principalement le temps d'exécution de la méthode appelée et la quantité de mémoire utilisée par cette méthode.

Le mécanisme de génération du descripteur d'instrumentation et du MBean est détaillé dans la section suivante.

4.3.2. Réalisation

Le service de suivi se charge de la création des MBean représentants les EJB, de leurs instanciations, de leurs enregistrements dans le serveur JMX, et de leurs suivis après déploiement. Le MBean est créé au moment du déploiement de l'EJB. Le service est composé principalement de trois éléments d'instrumentation (un récepteur, un introspecteur et un parseur) et un élément de suivi (le superviseur) (cf. Figure 4.13).

Le récepteur est la partie chargée de détecter l'arrivé des nouveaux EJB pour le déploiement. Il est toujours à l'écoute du service de déploiement du serveur d'applications. Ce dernier envoie une notification au récepteur quand un nouveau EJB arrive. Le récepteur reçoit avec cette notification une référence vers l'EJB. Le récepteur extrait la description des interfaces home et remote de l'EJB et il les transmet au second élément du service, l'introspecteur.

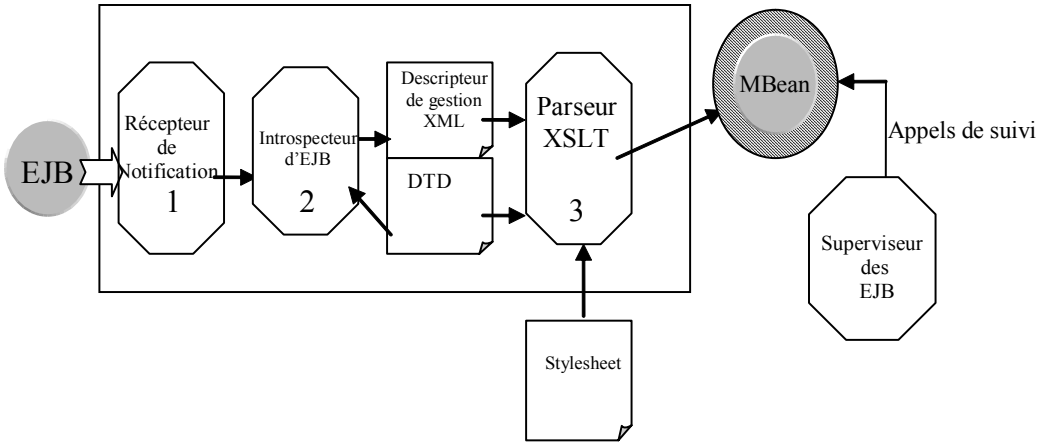

Figure 4.13 Service de suivi

Ce dernier procède à une introspection des interfaces reçues par le récepteur. Le résultat de cette introspection est un objet de type MBeanInfo décrivant les attributs, les méthodes de cycle de vie, et les méthodes métiers de l'EJB. L'introspecteur génère ensuite le descripteur XML d'instrumentation en respectant la DTD de définition. Cette phase de génération du descripteur XML est utile pour

l'ouverture sur d'autres systèmes d'administration comme SNMP ou WBEM où chaque système transforme le document dans sa propre présentation.

Le parseur XSLT [XSL99] transforme le descripteur XML selon une feuille de style XSL dans un MBean, notamment une interface et une classe java représentant le MBean. Pour chaque EJB, la classe java du MBean généré prend le nom de l'interface component de l'EJB précédé par le mot "Instrumented". L'interface a le même nom que la classe suivi par le mot "MBean". Par exemple, pour un EJB qui a l'interface component "HelloWorld", la classe du MBean a le nom "InstrumentedHelloWorld" et l'interface a le nom "InstrumentedHelloWorldMBean".

Le superviseur instancie et enregistre dans le serveur JMX un objet MBean pour chaque EJB. Ensuite il commence la supervision du fonctionnement des EJB en lançant des appels aux MBeans. Les méthodes à tester et la fréquence d'appels sont indiqués dans un fichier de configuration et peuvent être changés à tout moment.

L'exemple suivant montre un EJB HelloWorld (figure 4.14 4.15, 4.16) et la classe et l'interface du MBean générés (Figures 4.17, 4.18).

```
public class HelloWorldBean implements SessionBean{
      String message;
      public void ejbCreate(String message){ this.message=message;  }
      public String getHelloMessage(){ return this.message;  }
      public void ejbPostCreate(){}
      public void ejbRemove(){}
      public void ejbActivate(){}
      public void ejbPassivate(){}
      public void setSessionContext(SessionContext sc){}
}
```

Figure 4.14 Implantation de l'EJB HelloWorldBean

```
public interface HelloWorldHome extends EJBHome{
      HelloWorld create(String message)throws RemoteException,
                                                  CreateException;
}
```

Figure 4.15 Interface home de l'EJB

```
public interface HelloWorld extends EJBObject{
      public String getHelloMessage() throws RemoteException;
}
```

Figure 4.16 Interface component de l'EJB

```
public class InstrumentedHelloWorld implements
                                          InstrumentedHelloWorldMBean{
      HelloWorldHome home = null;
      HelloWorld instanceEvaluation = null;
      String mesures;

      public InstrumentedHelloWorld(){
            Context ctx = new InitialContext();
            home = (HelloWorldHome) ctx.lookup("HelloWorldBean");
            instanceEvaluation = home.create("");
      }
      public void create(String msg){
            initMesures();
            HelloWorld temp = home.create(msg);
            mesures = calculerMesures() ;
      }
      public String getHelloMessage(){
            String msg = new String();
            initMesures();
            msg = instanceEvaluation.getHelloMessage();
            mesures = calculerMesures();
            return msg;
      }
      public String getMesures(){
            return mesures;
      }
      public String calculerMesures(){
      …
}
```

Figure 4.17 La classe java du MBean

```
public interface InstrumentedHelloWorldMBean{
      public String create(String msg);
      public String getHelloMessage();
      public String getMesures();
      …
}
```

Figure 4.18 Interface java du MBean

L'EJB HelloWorld est un simple EJB de type session. Il a un seul attribut *message* et une seule méthode métier getHelloMessage() qui renvoie la valeur de *message*. L'attribut message est initialisé à la création de l'instance de l'EJB avec la méthode ejbCreate(String msg). En instrumentant cet EJB on obtient le MBean *InstrumentedHelloWorld* (figures 4.17, 4.18).

Ce MBean contient principalement deux attributs, un attribut qui garde une instance home de l'EJB, et un attribut qui garde une instance de l'EJB que l'on appelle l'instance d'évaluation. Ces deux attributs sont instanciés dans le constructeur au moment de la création de l'objet MBean. Le MBean contient des méthodes d'emballage pour les méthodes de l'EJB.

Dans chacune des méthodes on appelle la méthode correspondante de l'instance de l'évaluation. Les calculs des mesures sont initialisés avant l'appel et calculés après l'appel. Les mesures que l'on considère sont le temps d'exécution et la quantité de mémoire occupée. L'attribut *mesures* contient les mesures calculées après le dernier appel d'une des méthodes de l'EJB. La valeur de l'attribut mesure est la concaténation du temps d'exécution en millisecondes suivi par la quantité de mémoire occupée en octets.

Finalement, l'interface de gestion *InstrumentedHelloWorldMBean* indique les méthodes qui sont accessibles aux autres applications et services de gestion.

4.3.3. Appels de suivi

Après avoir créé un MBean pour chaque EJB, le service de suivi envoie régulièrement des appels aux MBeans dans la couche d'applications afin de tester le fonctionnement des EJB déployés, et pour s'assurer que l'EJB n'utilise que ce qui a été déclaré à la phase d'évaluation.

Principalement, on teste le temps d'exécution et la quantité de mémoire pour chaque méthode déclarée dans l'interface home et chaque méthode déclarée dans l'interface component en envoyant des appels de suivi aux MBean représentatifs.

Afin d'éviter que les appels de suivi soient trop élevés (ce qui influence la performance du serveur) ou trop espacés (ce qui ajoute les coûts de passivation et de re-activation de l'instance d'évaluation) nous choisissons de lancer les appels de suivi en fonction du temps de passivation.

L'algorithme appliqué par le service de suivi est le suivant :

```
Tant que (vrai)
   Si (tempsSystème – TempsDernierAppel) ≥ ( 3/4 temps de passivation)
      Pour chaque EJB déployé sur le serveur faire
         Pour chaque méthode dans l'interface home et l'interface component
            Invoquer la méthode correspondante sur le MBean représentative
            Mémoriser les mesures
         Fin pour
      Fin pour
      Analyser les mesures
   Fin Si
Fin Tant que
```

Dans la phase d'analyse des mesures, on vérifie si la méthode CPU qui a été fournie par le déployeur est bien la méthode qui consomme le plus de CPU et si le temps d'exécution ne dépasse pas le seuil fixé à la phase d'évaluation pour N clients. On fait les mêmes vérifications pour la méthode Memory. Si les mesures calculées à la phase d'évaluation ne correspondent pas aux mesures obtenues suites aux appels de suivi, le service signale un dysfonctionnement.

On considère deux causes possibles pour un dysfonctionnement : la première est qu'un des EJBs a changé de comportement (ex. le nombre max client a été dépassé ou les besoins ont été modifiés), la deuxième est que les caractéristiques du serveur ont été changés. Le service de suivi signale un dysfonctionnement en précisant la source de ce problème.

4.4. Service de localisation d'EJB

L'accès à la fédération ne ressemble pas à l'accès à un cluster car les serveurs de la fédération sont appelés individuellement par les clients et ne sont pas masqués par un concentrateur unique. Nous voulons permettre aux clients de chercher un composant EJB où qu'il soit sur la fédération en appelant n'importe quel serveur d'applications de la fédération. Cette démarche est nécessaire du fait que la

fédération est un environnement dynamique et adaptable où les composants changent de place en fonction des changements de leur comportement ou du comportement du serveur qui les héberge ou encore afin d'obtenir une meilleur performance pour le client.

Nous proposons d'ajouter un service de localisation pour chaque serveur afin de localiser les serveurs qui disposent d'un composant EJB déterminé.

Dans la suite de cette section nous présentons l'approche que l'on a adopté afin de réaliser le service de localisation.

4.4.1. Réalisation

JNDI (Java Naming and Directory Interface) [JNDI] est l'API utilisée par les serveurs d'applications afin de permettre l'accès au service d'annuaire du serveur. Si une recherche sur l'annuaire ne trouve pas le composant cherché, une exception est levée. Notre objectif est d'étendre les fonctionnalités de l'annuaire de sorte qu'une exception n'est renvoyée au client que si le composant demandé ne peut être retrouvé sur aucun serveur dans la fédération.

Au lieu de modifier l'API de JNDI pour ajouter la fonctionnalité de recherche de composants sur tous les serveurs dans la fédération nous adoptons une démarche intermédiaire. Nous ajoutons un proxy JNDI qui fait passerelle entre le client qui accède au service d'annuaire JNDI et le service d'annuaire JNDI standard (Figure 4.19).

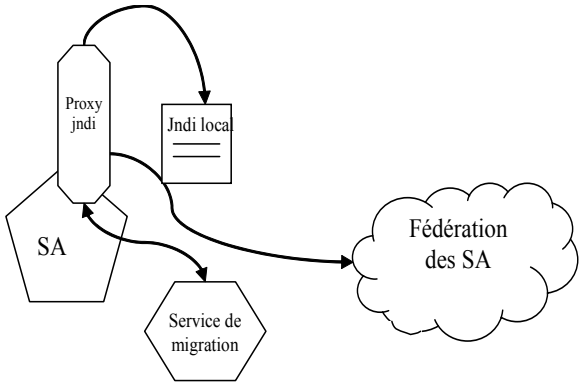

Figure 4.19 Architecture du proxy JNDI

En premier lieu, la passerelle JNDI cherche sur le service d'annuaire local le composant demandé. S'il est trouvé, elle rend au client qui l'a demandé le stub de l'objet home. Par contre, s'il n'est pas trouvé, le composant demandé est recherché sur les autres serveurs d'applications qui forment la fédération. Quand le composant est localisé, le proxy JNDI peut rediriger le client vers le serveur d'applications du composant, ou demander la migration du composant au serveur d'origine. Le choix

entre ces deux options peut optimiser la performance rendue au client de la fédération.

4.5. Service de migration

Suite à la détection d'un dysfonctionnement ou d'une dégradation de performance, il est nécessaire de re-adapter l'EJB automatiquement en permettant la migration d'un composant EJB vers un autre serveur d'applications mieux adapté pour accueillir l'EJB [FRE02].

La migration que l'on réalise dans notre fédération est une migration à froid. Plus précisément, on ne s'intéresse pas à migrer les instances actives d'un EJB mais la totalité du jar contenant le code et la description du composant. Les clusters proposent plusieurs mécanismes pour effectuer la migration à chaud.

La migration que l'on applique dans la fédération ne se limite pas à déplacer le code du composant mais aussi à gérer les liens avec les ressources référencées par l'EJB. Les ressources auxquelles le composant peut accéder sont les services de base de la plateforme J2EE, les références vers d'autres composants EJB et les références des ressources externes. Il faut donc déterminer quelles ressources sont déplaçables et lesquelles ne le sont pas. Les liens avec les ressources sont reconfigurés en fonction du type de ressource. Des références distantes aux ressources qui se trouvent à l'ancien endroit peuvent être créées. Ceci est une solution dans le cas où la ressource ne se trouve pas sur la nouvelle machine et il ne serait pas possible de déplacer la ressource à travers le réseau. Dans d'autres situations le déplacement de la ressource peut être envisageable.

4.5.1. Réalisation

Le service de migration répond à deux types de demandes de migration :

- Demande à migrer un EJB qui se trouve sur un serveur distant du serveur local.
- Demande à migrer un EJB qui se trouve sur le serveur local vers un autre serveur de la fédération.

Le service de migration reçoit la demande de migration sous forme d'une notification. Suite à la réception de cette notification le service analyse un choix de migration. Le service considère principalement quatre types de migration (cf 3.4.2.5). Le type de migration est choisi en fonction de type de l'EJB, du type des interfaces home et component que l'EJB utilise (remote ou locale) et des ressources que l'EJB référence. Nous ajoutons donc dans le fichier de déploiement une nouvelle

balise <migration> qui décrit le type de migration à appliquer. Le service procède au traitement de la demande de migration après avoir analysé la balise de migration.

Dans la suite nous détaillons chacun des types considérés dans la fédération.

4.5.1.1. Redirection de la requête

Le client, récupère une référence vers un autre serveur où se trouve le service. Cette stratégie est utilisée quand l'EJB n'est pas transférable et qu'un client tente d'avoir une référence d'un EJB qui n'est pas déployé sur le serveur d'origine mais ailleurs dans la fédération. Dans ce cas, le service de migration récupère du serveur distant, l'usine de fabrication de composants du type demandé (le stub de l'objet home). Désormais, le client accède au serveur d'applications distant. Dans cette stratégie il n'y pas eu réellement de déplacement mais plutôt une redirection du client vers le serveur distant. La Figure 4.20 montre le diagramme de séquences UML présentant cette stratégie.

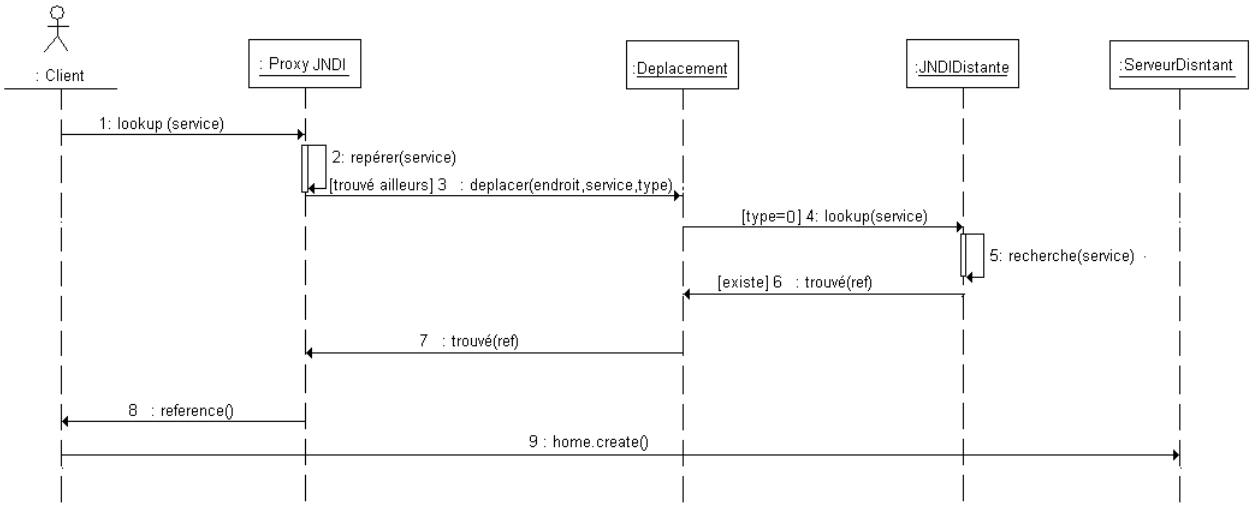

Figure 4.20 Digramme UML de séquences pour un redressement de requête

4.5.1.2. Transfert libre

Dans cette stratégie de transfert, le service de migration déplace le jar qui contient l'EJB et demande son déploiement sur un autre serveur. Dans ce cas, l'EJB doit être transférable et n'a pas de ressources ni d'autre EJB référencés. Le diagramme UML de séquence de cette stratégie est montré en Figure 4.21.

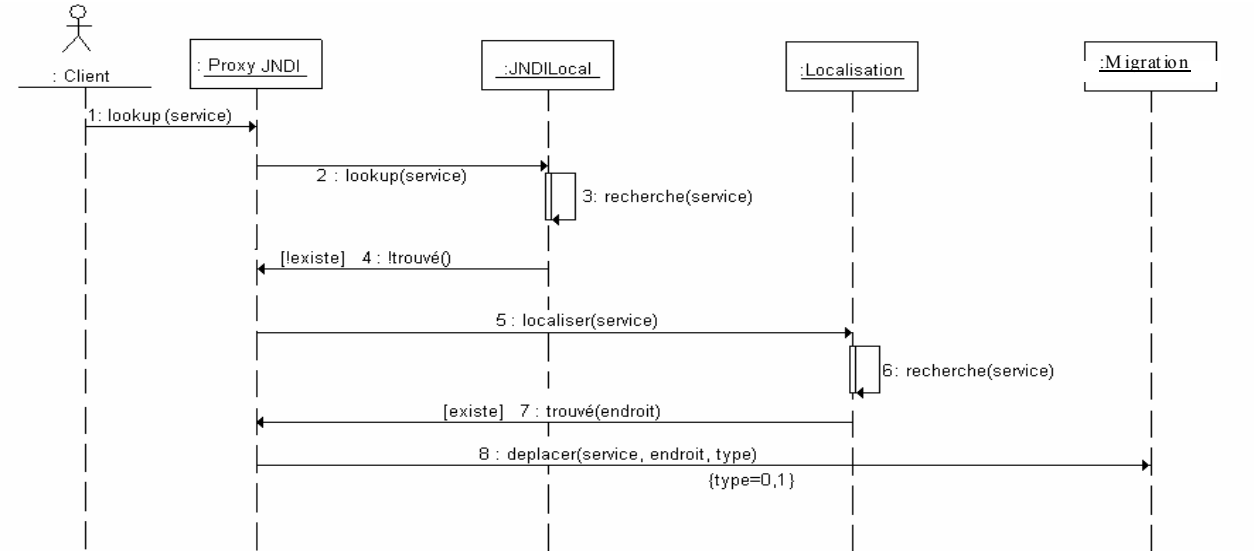

Figure 4.21 Diagramme UML pour le transfert libre

4.5.1.3. Transfert multiple

Ce type de transfert concerne les EJB qui utilisent d'autres EJB. Le transfert libre est donc un cas spécial du transfert multiple. On peut détecter qu'un EJB référence un autre par le descripteur XML de déploiement. Une analyse récursive est donc effectuée afin de choisir un type de transfert pour chaque EJB référencé.

La Figure 4.22 montre l'environnement initial d'un EJB A qui référence un EJB B avant d'effectuer une opération de migration du serveur 1 vers serveur 2.

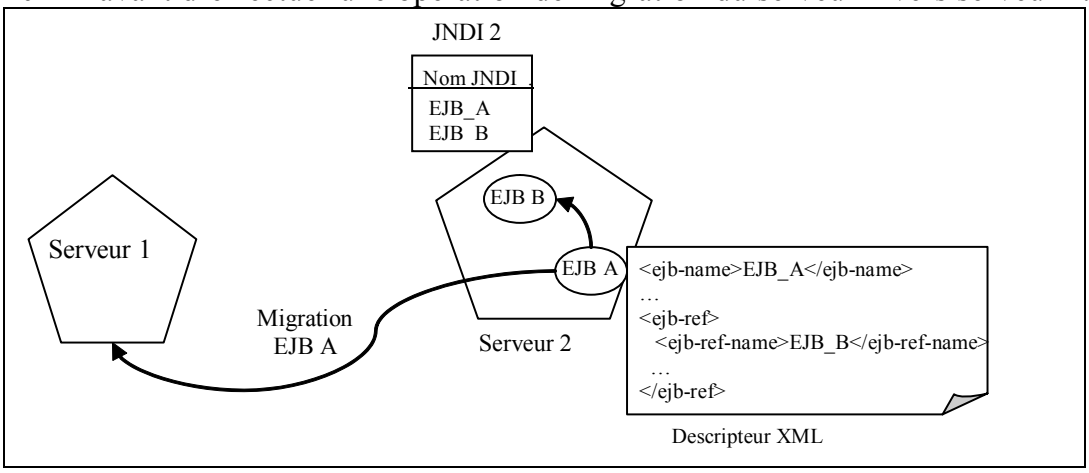

Figure 4.22 Environnement initial pour un EJB A avant la migration

Le service de migration va choisir un mode de déplacement pour l'EJB A et il va continuer l'analyse pour déterminer un mode de déplacement pour l'EJB B. En fonction des caractéristiques de chaque EJB, le service peut choisir de migrer les deux EJB ou seulement l'EJB initial (EJB A).

La Figure 4.23 montre le cas de migration des deux composants A et B. Après la migration les EJB sont disponibles sur le serveur 1 sans aucun lien avec le serveur 2. La Figure 4.24 montre le cas de migration d'un seul composant. Dans ce cas, quand l'EJB B est demandé par l'EJB A, le service de migration le rend accessible par redirection de requêtes (mode de transfert libre).

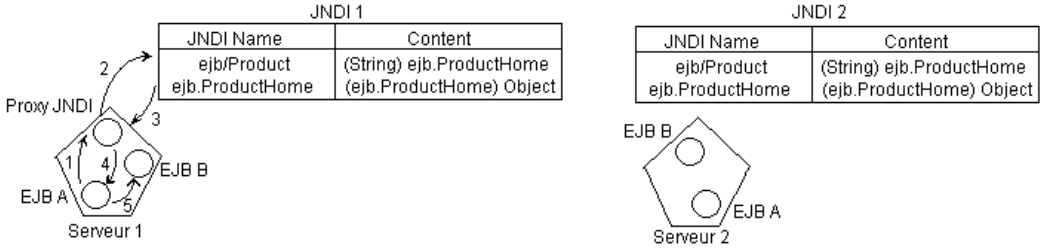

Figure 4.23 Migration des deux composants

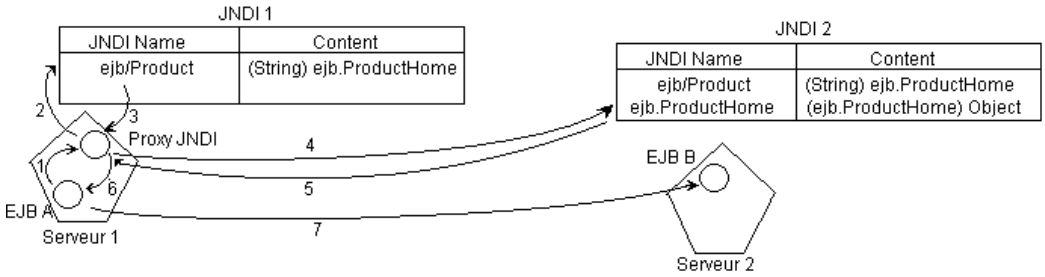

Figure 4.24 Migration du composant initial

4.5.1.4. Transfert avec ressource

Ce mode de migration est utilisé quand l'EJB appelle des ressources externes. Dans ce type de migration on gère les liens avec les ressources. Principalement nous considérons les connexions à une base de données pour un EJB de type entité CMP (Conteneur Managed Persistence). Ce type d'EJB confie la gestion de la persistance au conteneur en lui fournissant les indications pour établir la connexion avec une base de données.

Un serveur d'application offre des pools de connexions aux bases de données. Un pool de connexions est un ensemble de connexions qui sont créées au démarrage du serveur d'applications. Ainsi, lorsqu'un client a besoin de se connecter à une base de données, il demande au serveur une des connexions du pool. Le temps d'attente du client est réduit car la connexion avec la base de données est déjà établie.

Un composant peut récupérer une connexion vers une base de données d'un pool directement si le composant est hébergé dans le même serveur qui implante le pool. Par contre, si la connexion du pool est récupérée depuis un objet qui ne se trouve pas sur la même machine virtuelle que le serveur d'applications, il faut

utiliser un objet DataSource. Un DataSource est un objet de définition d'un pool de connexions à une base de données. Les DataSources ont un nom JNDI et peuvent donc être récupérés par des composants EJB depuis le JNDI.

Avant le déploiement d'un EJB CMP, l'EJB spécifie les caractères du pool et du DataSource qu'il utilise dans un fichier de configuration. L'objet datasource permet au serveur d'applications de récupérer des connexions du pool associé afin de gérer la persistance des EJB.

Après la migration des EJB de type CMP, il est important de permettre à l'EJB d'accéder à la base de données. Nous considérons plusieurs types de connexions avec la base.

- Création dynamique de la base : ce type consiste à recréer un nouveau pool et une nouvelle base de données dans le nouveau serveur d'applications. Dans ce cas l'EJB tourne proprement mais sans les données initiales de la base. Ce type est possible pour les services qui produisent des données locales.
- Création dynamique du pool de connexions : dans ce cas, le nouveau serveur d'applications est le responsable de la création du pool de connexions avec la base de données d'origine.
- Réutilisation du même pool à distance : cette solution consiste à utiliser le même pool d'origine sur la même base de données d'origine.

La Figure 4.25 montre l'environnement d'un EJB A que l'on veut migrer. Dans l'annuaire de JNDI on trouve le nom JNDI de l'EJB et le nom JNDI de l'objet DataSource qui définit le pool de connexions à la base de données. Le fichier de configuration config.xml montre la description du pool et de la DataSource.

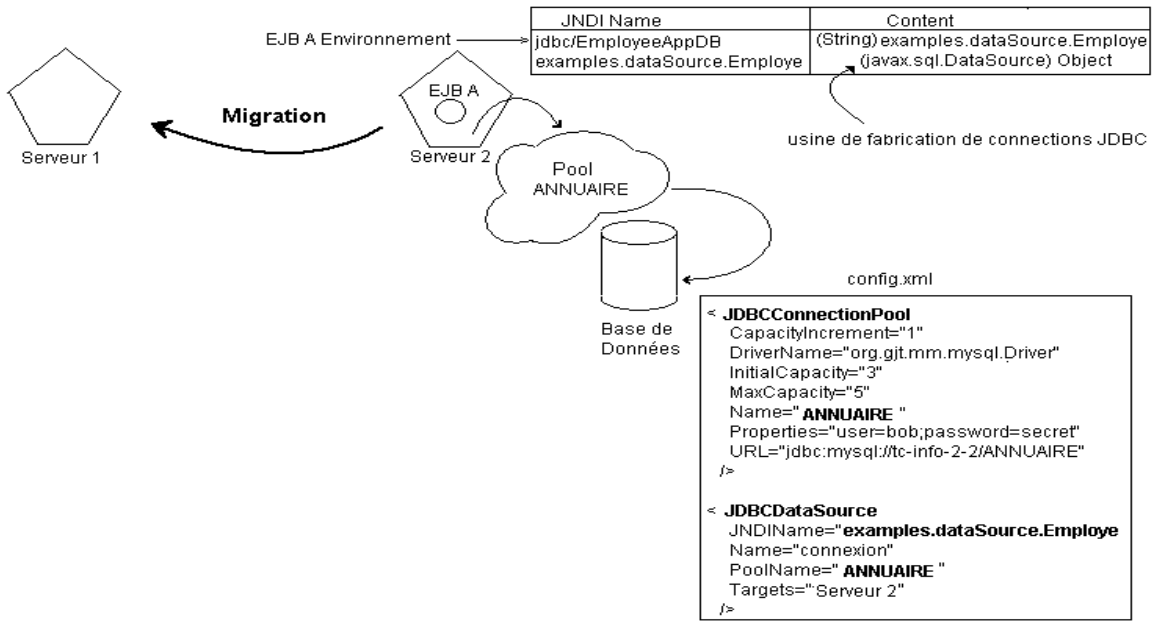

Figure 4.25 Environnement de l' EJB A avant la migration

La Figure 4.26 montre l'EJB A après avoir appliqué la migration de type "création dynamique du pool". La Figure 4.27 montre le type "réutilisation"

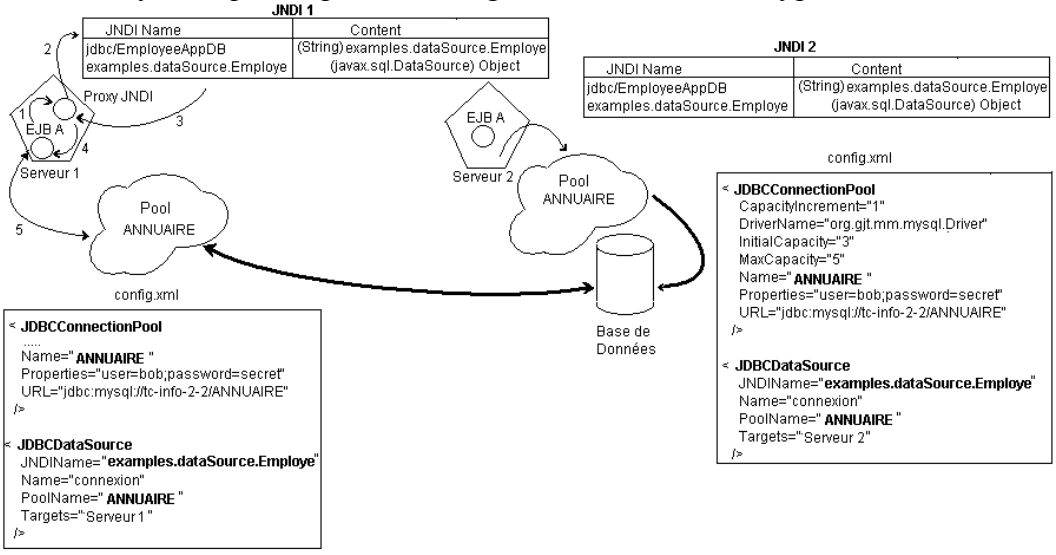

Figure 4.26 Migration de l'EJB A avec la création d'un nouveau pool

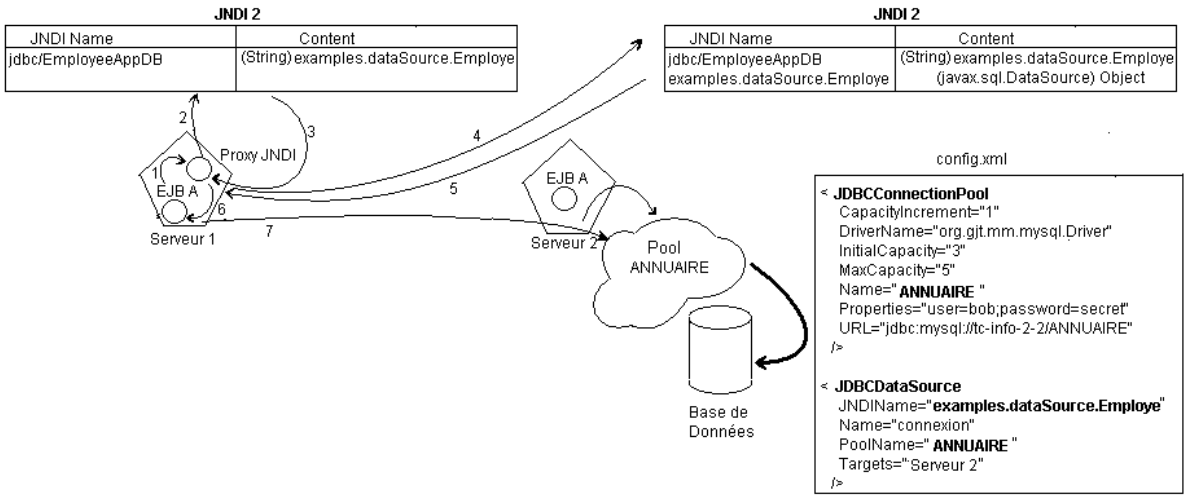

Figure 4.27 Migration de l'EJB avec une référence vers les ressources

4.6. En résumé

Dans ce chapitre nous avons présenté les services de la plate-forme de gestion du déploiement. Ces services communiquent entre eux afin de réaliser le déploiement adapté. Chaque service participe à une des phases de déploiement décrites en 3.2.4.

Le service d'évaluation réalise la première phase du déploiement. Le service d'instrumentation participe à la réalisation de la phase décisionnelle du déploiement. Le service de suivi participe à la phase d'adaptation.

Les services de localisation et de migration participent à la gestion de la globalité du déploiement et à la coopération entre les serveurs.

Dans le chapitre suivant nous allons montrer comment nous organisons la coopération entre les services et les serveurs d'applications et comment la communication est réalisée dans la fédération.

5. Communications et coordination du déploiement dans la fédération

Chacun des services présentés ci-dessus réalise une des tâches du déploiement adapté spécifié en 3.2.4. Afin d'accomplir une action de déploiement adapté sur la fédération, il faut d'une part assurer la communication entre les serveurs de la fédération et d'autre part organiser la coopération entre les différents services du déploiement. Dans ce chapitre, nous présentons l'approche que l'on utilise pour assurer la communication dans la fédération, ensuite nous détaillons le service de coordination qui réalise la tâche d'organisation.

5.1. Communications dans la fédération

Chaque serveur d'applications doit permettre aux services de gestion et aux autres serveurs d'applications de la fédération d'accéder à sa plate-forme d'administration. Tous les services sont implantés sous la forme de MBean. L'ensemble de ces MBeans est enregistré dans le MBeanServer qui permet l'accès aux attributs et aux opérations des MBeans.

Afin de permettre l'accès à distance nous déployons dans chaque serveur d'applications un EJB d'accès à l'interface d'instrumentation appelé le MEJB. Cet EJB contacte le MBeanServer localement pour récupérer les informations des MBeans et les envoie aux clients distants. L'idée d'un EJB d'accès a été présentée par sun [JSR77] et il fournit également les interfaces *home* et *component* d'implantation de l'EJB (javax.management.j2ee.ManagementHome, javax.management.j2ee.Management).

L'interface *home* contient une seule méthode de création d'un objet session de du MEJB. L'interface *remote* définit toutes les méthodes qui permettent l'accès aux MBeans. Connaissant le "nom" d'un MBean, on peut récupérer la liste de ses attributs et de ses opérations avec la méthode getMBeanInfo("nom"). Cet objet regroupe l'ensemble des attributs et des opérations du MBean invoqué. Ensuite, on peut récupérer ou changer la valeur d'un attribut avec les méthodes getAttribute(nomMBean, nomAttribut), setAttribute(nomMBean, attribut). On peut également invoquer une de ses opérations avec la méthodes invoke(nomMBean, nomOperation, listeDeParamètres, signature). La liste des méthodes principales du MEJB est présentée dans le Tableau 5.1.

Description des méthodes	
Integer	**getMBeanCount()** Retourne le nombre de MBeans enregistrés dans MEJB
Set	**queryNames(**ObjectName *name***,** QueryExp *query***)** Récupère l'ensemble des MBeans qui satisfont la requête
Object	**getAttribute(**ObjectName *name***,** String *attribute***)** Récupère la valeur d'un attribut d'un MBean qui a le nom "name"
AttributeList	**getAttributes(**ObjectName *name***,** String[] *attributes***)** Retourne les valeurs de plusieurs attributs d'un MBean
MBeanInfo	**getMBeanInfo(**ObjectName *name***)** Retourne un objet qui permet la récupération des attributs et des opérations qu'un MBean expose pour la gestion
Object	**invoke(**ObjectName *name***,** String *operationName***,** Object[] *params*, String[] *signature***)** Invoque une opération d'un MBean
Void	**setAttribute(**ObjectName *name***,** Attribute *attribute***)** Instancier la valeur d'un attribut d'un MBean

Tableau 5.1 Description des méthodes de l'interface remote de MEJB

L'utilisation du composant MEJB se fait de la même façon qu'un composant EJB standard. La Figure 5.1 montre le code qui permet à une application de gestion de récupérer les noms des attributs du MBean JmxNWSSensor dans la couche système d'exploitation de notre interface d'instrumentation.

```
// getting a reference to the MEJB
Context c=new InitialContext();
ManagementHome mejbHome = (ManagementHome) c.lookup("MEJBName");
Management mejb=home.create();
// Creating ObjectName to define the name of the MBean
ObjectName name="instrumentation: layer=system,
                                  resource=JmxNWSSensor"
// get all operations and attributes of the MBean de NWS
MBeanInfo mbi=mejb.getMBeanInfo(name);
MBeanAttributeInfo[] atts=mbi.getAttributes;
for(int i=0;i<atts.length;i++){
  System.out.println("Attribute:" + atts[i].getName() + " is
                    of the type:" + atts[i].getType() );
}
```

Figure 5.1 Code d'accès à la plate-forme d'instrumentation

La première étape est de récupérer une référence du composant MEJB en cherchant dans l'annuaire JNDI du serveur d'applications *c.lookup(nom).* On créer ensuite l'objet mejb qui permet l'accès à distance au composant MEJB *mejb=home.create().* On peut ensuite demander les opérations et les attributs du MBean JmxNWSSensor avec la méthode *mejb.getMBeanInfor(nom).* Finalement l'application de gestion peut traiter les attributs ou invoquer les opérations du MBean. Figure 5.2 montre l'accès d'une à la plate-forme d'administration d'un nœud de la fédération.

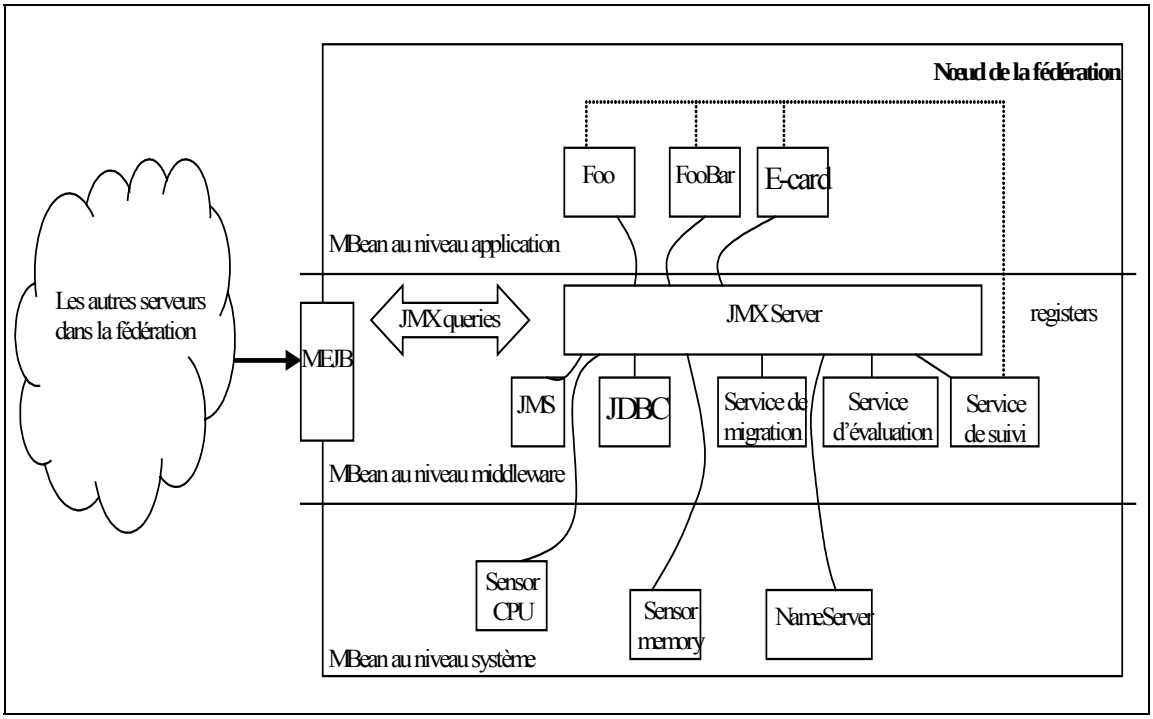

Figure 5.2 Accès à la plate-forme d'administration

Cet interfaçage entre JMX et EJB permet à notre fédération de profiter d'une part des services d'authentification et de sécurité fournis par la plate-forme J2EE et d'autre part d'appliquer le même protocole de communication utilisé par le serveur d'applications sans définir un nouveau connecteur ou adapter un protocole au niveau de la plate-forme JMX.

5.2. Service de coordination du déploiement adapté

Une partie essentielle de notre modèle d'administration est l'organisation de la coopération entre les services en précisant l'ordre d'appels entre les services, et les choix à prendre concernant le déploiement et l'adaptation. Cette tâche est réalisée par le service de coordination détaillé dans la section suivante.

5.2.1. Réalisation

Le but de ce service est d'organiser les déploiements des EJB à l'aide des différents services d'administration sans que les services communiquent entre eux directement. L'ensemble des services vont alors participer à la réalisation d'un scénario de déploiement sans qu'ils soient liés entre eux d'une manière forte.

Dans la section 3.4.3 nous avons identifié les deux rôles du service de coordination du déploiement :

- Exécuter des procédés de déploiement adapté,
- Prendre les décisions nécessaires pour le déploiement.

Nous identifions donc deux parties pour la réalisation du service (Figure 5.3):

- *Un moteur de procédés* qui analyse et applique des procédés de déploiement en signalant à chaque service d'administration le début de son activité,
- *Les règles de déploiement* qui indiquent au moteur de procédés comment prendre les décisions de déploiement, notamment le choix du serveur d'hébergement et le choix d'actions d'adaptations.

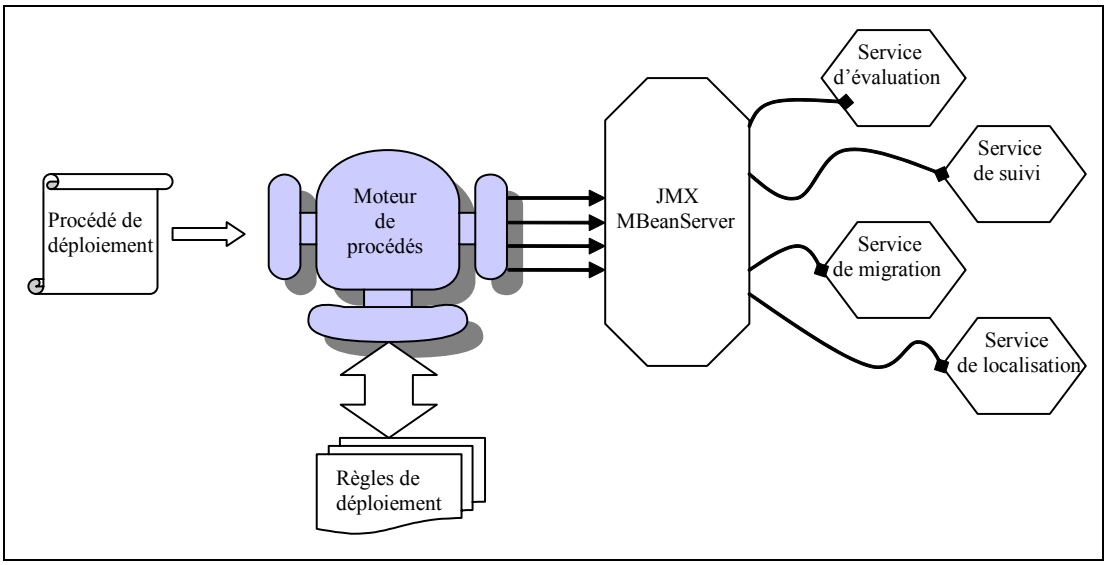

Figure 5.3 service de coordination

Le moteur de procédé est responsable de la globalité du déploiement. Il réalise le déploiement en se basant sur un procédé de déploiement qui indique, entre autre, l'ordre d'appels des services de gestion. Une seule fédération peut avoir un ou plusieurs procédés de déploiement. Il est possible d'appliquer le même procédé pour tous les serveurs d'applications ou bien de permettre à chaque serveur d'appliquer son propre procédé. Le moteur de procédé donc doit être capable de lire différents procédés, de configurer les connexions entre les services selon le procédé, et de lancer des actions de déploiement global.

En plus du procédé de déploiement le moteur de procédé dépend des règles de déploiement. Ces règles aussi peuvent être différentes d'un serveur à l'autre. Dans la fédération nous définissons ces règles en tant qu'attributs dans fichiers de configuration. Des règles différentes peuvent être appliquées au moment du déploiement de deux différents EJB. Par exemple, l'action d'adaptation pour un EJB

qui consomme plus de ce qui a été déclaré dans le rapport d'évaluation peut être sévère en rejetant l'EJB de la fédération si l'EJB ne provient pas d'un serveur de confiance.

D'autre part, un serveur d'application peut définir plus de règles qu'un autre serveur dans la fédération. Dans la fédération nous estimons que l'existence des règles de déploiement est importante, mais nous ne les avons pas établies.

Comme notre système d'administration est basé sur JMX, la réalisation du déploiement se base sur les appels JMX. Tous les services de la plate-forme d'administration sont enregistrés dans le MBeanServer. Leurs interfaces de gestion exposent les opérations que l'on peut invoquer sur les services.

Le moteur de procédé analyse un procédé et constitue les invocations des services selon l'ordre déterminé par le procédé de déploiement.

Dans la section suivante nous présentons comment définir un procédé, sachant qu'il est possible de créer plusieurs procédés décrivant différents scénarii de déploiement.

5.3. Le Procédé de déploiement

Un procédé peut être vu comme un ensemble d'activités ou d'opération liées les unes aux autres afin de réaliser un objectif. Le but de l'utilisation d'un procédé est de décrire l'interaction entre les différents éléments participant à la réalisation de l'objectif [DER94], [DER99], [LES02].

Dans la suite de cette section nous présentons le modèle de procédé sur lequel nous nous sommes basé, un exemple de procédé et le moteur de procédés.

5.3.1. Modèle de procédé

Dans le cadre de notre fédération nous définissons un modèle de procédé (Figure 5.4) qui comprend les éléments suivants :

- Les Activités : une activité est un pas dans un procédé qui contribue à la réalisation d'un objectif. Une activité est réalisée par un des services de notre système d'administration.
- Les Produits : les produits sont les objets que les activités doivent manipuler. Une activité manipule un ou plusieurs produits en entrée et génère un ou plusieurs produits en sortie.

- Les Ports : les ports représentent les points d'entrée d'une activité. Dans le cadre de notre fédération, les ports sont les méthodes d'entrée aux services.
- Les Flots de Données : les flots de données montrent comment les produits circulent entre les ports des activités. Dans le cadre de notre fédération, les flots de données spécifient l'ordre d'appel aux différents services.

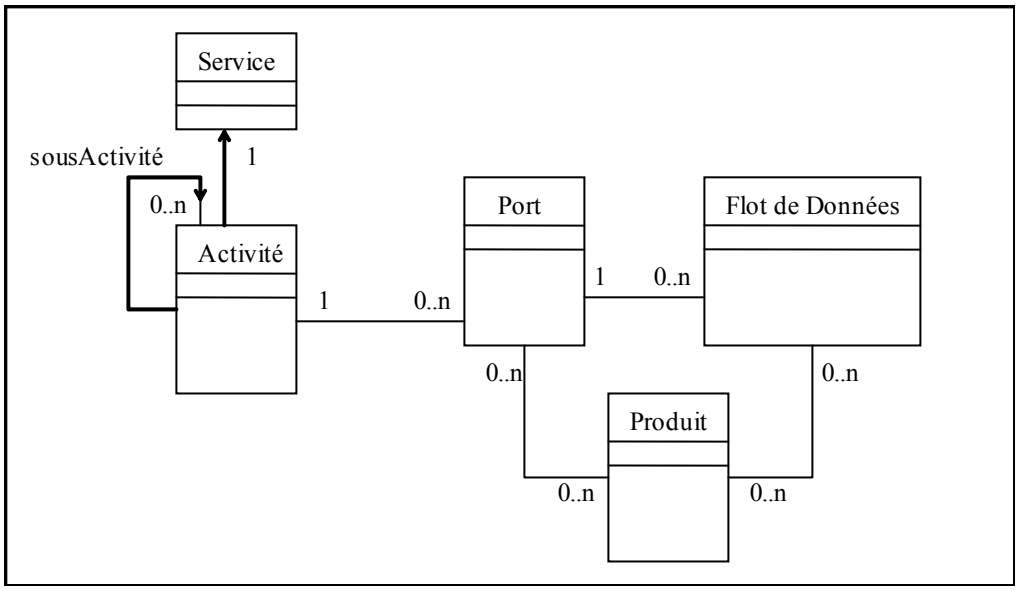

Figure 5.4 Modèle de procédés

Dans le contexte de notre fédération, les activités se limitent à : l'évaluation d'un EJB, la localisation d'un service, la migration des composants EJB, la prise de décision, le déploiement local. Quant aux produits, on peux mentionner les produits principaux de notre fédération : les composants EJB, les descripteurs de déploiement, les rapports d'évaluation, les rapports de correspondance et les MBeans représentatifs des EJBs.

5.3.2. Description d'un procédé

Dans cette section nous présentons un scénario de déploiement et nous implémentons le procédé qui le décrit. Dans le scénario que nous avons choisi, nous voulons qu'un serveur de la fédération puisse tout d'abord essayer de déployer un EJB localement, et il ne contacte les autres serveurs de la fédération que si le déploiement échoue localement. Dans ce cas la logique de notre déploiement global est la suivante :

1. Evaluer les besoins des EJB
2. Essayer de déployer les EJB localement

- Si ok :
  1. Instrumenter les EJB et suivre leur comportement
  2. En cas de dysfonctionnement, faire un choix d'adaptation selon les règles d'adaptations fournies
- Si échec :
  1. Contacter les autres serveurs pour prendre une décision de déploiement.
  2. Contacter le service de déploiement du serveur choisi

La Figure 5.5 montre la représentation graphique du procédé. Les rectangles représentent les activités, les liens entre les rectangles représentent les flots de données, les points noirs représentent les ports, et finalement les produits ont été notés sur les liens.

Nous avons décomposé ce procédé de déploiement en plusieurs activités :

- Evaluation : cette activité est réalisée par le service d'évaluation (cf 4.2). Le port d'entrée reçoit le package des EJB à évaluer, et il produit un rapport d'évaluation qui sera un des produits d'entrée sur le port de l'activité *Suivi des EJB.*
- Déploiement Local : cette activité est réalisée par le service local de déploiement propre au serveur. Le flot de données dirige le package à déployer vers le port d'entrée de cette activité. Si le déploiement est réussi, le flot de données se dirige vers l'activité *Suivi des EJB.* Si par contre le déploiement échoue, le flot de données se dirige vers l'activité *Choix du Serveur.*
- Suivi des EJB : cette activité est réalisée par le service de suivi (cf 4.3). En entrée, le service reçoit les EJB qui ont été récemment déployés localement sur le serveur. Il génère les MBeans représentatifs et il les enregistre auprès du service d'instrumentation. Le flot de données dirige les MBeans vers *l'Instrumentation*. Le service de suivi effectue l'activité de suivi et en cas de dysfonctionnement il dirige le flot de données vers l'activité d'*Adaptation.*

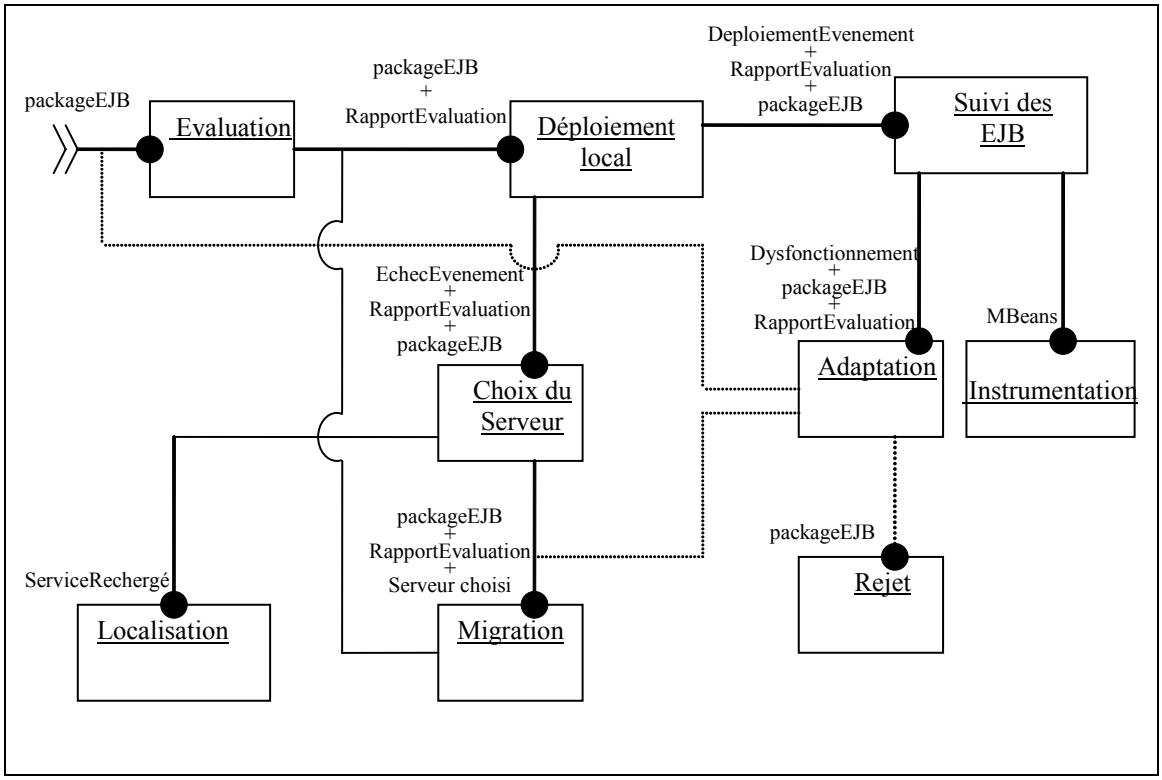

Figure 5.5 Exemple d'un procédé de déploiement

- Instrumentation : cette activité est réalisée par le service d'Instrumentation (cf 4.1). Au cours d'un procédé de déploiement, le rôle principal de cette activité est d'enregistrer les nouveaux MBeans dans la plate-forme d'instrumentation JMX.
- Adaptation : cette activité est réalisée par le service de déploiement global. Elle consiste à prendre une décision d'adaptation. Les critères selon lesquels la décision est prise sont fournis par les règles de déploiement global de la fédération. Le flot de données peut être dirigé vers l'activité de *Rejet*, de *Migration*, ou d'*Evaluation.*
- Rejet : cette activité consiste à retirer l'EJB du serveur après avoir constaté que l'EJB ne respecte pas les termes d'évaluations qui ont été fixés au moment du déploiement.
- Choix du Serveur : cette activité est réalisée par le service de déploiement global. Elle consiste à faire une décision concernant le choix du serveur qui répond le mieux aux besoins de l'EJB. L'activité reçoit le flot de données sortant de l'activité de déploiement local en cas d'échec du déploiement sur serveur local. Les produit d'entrées sont : l'événement d'échec de déploiement, le package d'EJB et le rapport d'évaluation. En se basant sur ces données et sur les règles de déploiement global de la fédération, l'activité choisit un autre serveur pour déployer l'EJB. Le flot de données est dirigé ensuite vers le service de

Migration. L'activité du choix du serveur peut aussi lancer l'activité de *localisation* pour chercher des services additionnels sur les serveurs de la fédération.

- Migration : cette activité est réalisée par le service de migration (cf 4.5). L'activité reçoit en entrée le package EJB à migrer, le rapport d'évaluation, et le serveur de destination. Elle dirige ensuite le flot de données vers l'activité de déploiement local du serveur destinataire.

Afin de décrire un procédé nous utilisaons XML. La Figure 5.6 montre une description simplifiée du procédé en XML.

Cette description est fournie au moteur de procédés afin de réaliser le scénario. Dans la section suivante nous présentons le fonctionnement du moteur de procédés.

Le moteur de procédé analyse un procédé et constitue les invocations des services selon l'ordre déterminé par le flot de données. L'analyse commence par l'activité *root* qui représente la première activité à invoquer. Dans l'exemple ci-dessous cette activité est *l'évaluation.* La balise <port> indique l'opération à invoquer sur le service qui réalise l'activité (le service d'évaluation). La balise <dataFlow> de chaque activité indique l'activité suivante à réaliser. Quand une activité définit plusieurs éléments dataFlow (ex : l'activité localDeployment), l'activité suivante à démarrer est déterminée par l'ensemble des produits disponibles après la terminaison de l'activité courante. Par exemple, pour déterminer quelle est l'activité suivante à l'activité *localDeployment,* le seul acteur est la disponibilité de tous les produits de l'élément dataFlow. Si les produits (événement de déploiement, rapport d'évaluation et package EJB) sont disponibles, l'activité suivante sera le *suivi.* Si par contre, les produits disponibles sont (l'événement d'échec de déploiement, le rapport d'évaluation et le package EJB), l'activité suivante sera la *prise de décision.*

La réalisation d'une activité correspond à l'invocation de la méthode d'accès au service de réalisation (le port). La Figure 5.7 montre l'appel JMX pour la réalisation de l'activité de l'évaluation.

```
< ?xml version= "1.0" encoding="ISO-8859-1" ?>
<process name="globalDeployment">
        <description>
                Exemple
        </description>
        <activity name="evaluation" type="root">
                <port name="evaluateEJB"/>
                        <product name="PackageEJB"/>
                </port>
                <dataFlow to-activity="localDeployment" to-port="deployEJB">
                        <product name="EvaluationReport"/>
                        <product name="PackageEJB" />
                </dataFlow>
        </activity>
        <activity name="localDeployment">
                <port name="deployEJB"/>
                        <product name="EvaluationReport"/>
                        <product name="PackageEJB"/>
                </port>
                <dataFlow to-activity="supervision" to-port="superviseEJB">
                        <product name="deploymentEvent"/>
                        <product name="EvaluationReport"/>
                        <product name="PackageEJB"/>
                </dataFlow>
                <dataFlow to-activity="serverChoice" to-port="chooseServer">
                        <product name="deploymentFailureEvent"/>
                        <product name="EvaluationReport"/>
                        <product name="PackageEJB"/>
                </dataFlow>
        </activity>
        <activity name="supervision">
                <port name="superviseEJB"/>
                        <product name="deploymentEvent"/>
                        <product name="EvaluationReport"/>
                        <product name="PackageEJB"/>
                </port>
                <dataFlow to-activity="instrumentation" to-port="registerMBean">
                        <product name="deploymentEvent"/>
                        <product name="EvaluationReport"/>
                        <product name="PackageEJB"/>
                </dataFlow>
                <dataFlow to-activity="adaptation" to-port="adaptEJB">
                        <product name="deploymentFailureEvent"/>
                        <product name="EvaluationReport"/>
                        <product name="PackageEJB"/>
                </dataFlow>
        </activity>
        <activity name="serverChoice">
                +  <port name="chooseServer"/>
                   </port>
                 + <dataFlow to-activity="localization" to-port="research">
                    </dataFlow>
                 + <dataFlow to-activity="migration" to-port="migrate">
                   </dataFlow>
        </activity>
        …
</process>
```

Figure 5.6 Exemple de description d'un procédé de déploiement avec XML

```
mbeanServer.invoke(
        "instrumentation: layer = Middleware, resource = EJBevaluation",
        "evaluateEJB",
        availableProducts,
        signature
        );
```

Figure 5.7 Invocation JMX pour la réalisation d'une activité

Le premier paramètre correspond au nom du MBean qui représente le service d'évaluation de l'EJB (cf 4.2). Comme tous les autres services additionnels au niveau middleware, le service est enregistré sous le nom du domaine *instrumentation* et la propriété "layer" a la valeur *middleware.* La propriété ressource représente le nom simple du service. Le deuxième paramètre représente le nom de la méthode à appeler sur le service (le port). Le troisième paramètre est un vecteur de produits attendus, sous forme de paramètres à passer à la méthode appelée. Finalement le dernier paramètre est un vecteur qui donne la signature de la méthode.

A l'intérieur d'un même serveur d'applications, le service de coordination contacte les autres services en passant par l'objet MBeanServer de la plateforme JMX. Par contre, pour contacter les autres serveurs de la fédération, le service passe par l'objet MEJB qui permet un accès distant à tous les services enregistrés en tant que MBean.

5.4. En résumé

Dans ce chapitre nous avons présenté le service de coordination du déploiement global afin de montrer comment les services de gestion présentés en chapitre 4 communiquent entre eux pour la réalisation d'une action de déploiement global. Ce service joue un rôle primordial dans la fédération en ajoutant l'aspect de flexibilité et de dynamique du déploiement.

Le service a principalement deux rôles effectués par deux éléments :

- le moteur de procédés, et
- les règles de déploiement.

Le moteur de procédés sait lire, analyser, et appliquer des procédés décrivant différents scénarii de déploiement. Afin de décrire un procédé de déploiement nous

avons défini un modèle de procédé qui se base sur [LES03]. Avec cette technique de procédé nous permettons que chaque serveur dans la fédération puisse appliquer son propre procédé de déploiement dans la fédération.

Les règles de déploiement indiquent au service de coordination comment prendre les décisions de déploiement, notamment le choix du serveur d'hébergement et le choix d'actions d'adaptations.

6. Bilan et Perspectives

6.1 RAPPEL DES OBJECTIFS

6.2. BILAN ET EVALUATION DU SYSTEME D'ADMINISTRATION

6.3. PERSPECTIVES

6.1. Rappel des objectifs

Le but que nous nous sommes fixé dans cette thèse est d'administrer des applications distribuées à base de composants. Nous nous sommes focalisé sur le modèle des EJB et les serveurs d'applications J2EE. L'infrastructure d'administration que nous avons présentée vise la gestion de déploiement des composants EJB sur une fédération de serveurs d'applications hétérogènes.

Nous nous sommes centré sur deux axes :

- Instrumenter et administrer l'ensemble de serveurs d'applications constituants la fédération,
- Réaliser un déploiement global adapté d'applications EJB sur la fédération ainsi instrumentée.

Le premier point sert d'une part à remonter les informations concernant les caractéristiques de l'environnement et les ressources disponibles sur les serveurs d'applications et d'autre part à fournir les services nécessaires pour réaliser l'administration sur la fédération.

Le second point permet de suivre le processus de déploiement sur la fédération. Le déploiement que nous visons est un déploiement à la fois *global* et *adapté*. Il est *global* dans le sens où il n'est pas réalisé sur un serveur d'applications isolé ; l'ensemble des serveurs de la fédération communiquent entre eux pour prendre une décision de déploiement. Le déploiement est *adapté* dans le sens où le choix de déploiement d'un EJB sur un serveur précis est pris en étudiant la correspondance entre les besoins déclarés par l'EJB et les ressources disponibles sur le serveur d'applications.

Ce travail est également motivé par le fait que, bien que la machine virtuelle ai standardisé la mise en œuvre des serveurs d'applications, un certains nombre de problèmes sont apparus. Le fait de ne pas prendre en compte l'hétérogénéité des serveurs d'applications et le manque de communication inter-serveurs, est pour nous un frein majeur à l'évaluation de la plate-forme J2EE.

6.2. Bilan et évaluation du système d'administration

Les travaux menés durant cette thèse concernent l'étude, la proposition et la réalisation d'un modèle de gestion décentralisé pour les applications distribuées à base de composants EJB. Le modèle de gestion a été proposé dans le cadre d'une fédération de serveurs d'applications. Dans notre modèle de gestion nous avons distingué deux parties : un ensemble de services de gestion et un coordinateur de déploiement global.

Pour la première partie, nous avons choisi JMX comme plate-forme de base pour la conception d'un service d'instrumentation et nous avons implanté plusieurs services de gestion permettant de contrôler plusieurs tâches de déploiement global. Celui-ci consiste à évaluer les besoins des composants EJB, choisir un serveur qui convient au mieux aux besoins des EJB, et suivre le comportement des composants durant leur exécution afin de les adapter.

Pour la deuxième partie, nous avons défini un modèle de procédés de déploiement qui permet de décrire le processus de déploiement à réaliser par le coordinateur du déploiement. Le procédé décrit les modalités de communications entre les services de gestions.

Notre travail de réalisation au niveau de la thèse est limité à la première partie. Nous avons intégré l'ensemble des services dans le serveur d'applications JBoss. Nous nous sommes servi du service interne JMX du serveur pour enregistrer les nouveaux services de gestion.

Pour la deuxième partie, nous avons défini le modèle de procédés mais nous n'avons pas réalisé le coordinateur du déploiement qui lit les procédés et qui les applique.

Les services de gestion que nous avons réalisés sont :

- **Le service d'instrumentation** qui supervise l'état des trois couches de l'environnement d'exécution : la couche système (CPU et mémoire), la couche middleware (services non - fonctionnels et services additionnels), et la couche composants (les EJB). Ce service est très important dans notre architecture car il représente le facteur de base de n'importe quel système de gestion : la connaissance de l'environnement. Il repose sur l'outil NWS pour la couche basse, le moteur JMX pour les deux autres.

- **Le service d'évaluation** qui détermine les besoins d'un composant EJB au niveau système et au niveau middleware. Le service génère à la fin un rapport d'évaluation. Au niveau système nous considérons principalement le temps d'exécution et la mémoire. Bien que nous croyions que les mesures réseau sont aussi importantes, dans ce service nous ne les avons pas considérées. Quant au niveau middleware, nous avons considéré les services additionnels.

- **Le service de migration** qui applique des actes de migration à froid pour déplacer un EJB d'un serveur à un autre dans la fédération. Le service permet un serveur d'application de faire migrer un EJB local sur un serveur distant dans la fédération ou faire migrer un EJB distant sur le serveur local. Le but de ce service est d'ajouter le dynamique dans la fédération en permettant de déplacer les EJB en fonction du changement de contexte d'exécution.

- **Le service de localisation** qui permet de localiser un EJB dans la fédération. La réalisation de ce service était nécessaire pour compléter la dynamique de la fédération. Après avoir migré un EJB il ne serait plus possible de retrouver l'EJB sans le service de localisation.

- **Le service de suivi** permet de superviser le fonctionnement des EJB après leur déploiement. Ce service ajoute l'aspect d'adaptation à la fédération. Dès qu'un dysfonctionnement est détecté, l'EJB source du dysfonctionnement est traité selon le choix d'adaptation adopté.

Comme tout autre système d'administration, notre système ajoute un coût d'administration au système administré. Le coût se résume principalement dans l'instrumentation du système, et l'interruption des appels de suivi.

Le système NWS est la source principale de surcharge car il intervient périodiquement afin de calculer les mesures sur la CPU, la mémoire et la bande passante.

L'utilisation de JMX pour la représentation des ressources dans les trois couches n'ajoute une surcharge qu'au moment de la génération des MBeans représentatifs. Cette génération ne se fait qu'une seule fois dans le cycle de vie de l'EJB déployé.

Après avoir généré les MBeans, ils seront appelés par les autres services de gestion dans notre système d'administration. Le service d'instrumentation lui-même n'ajoute pas de surcharge, mais c'est la fréquence d'appels des MBeans qui indique l'importance de la surcharge :

- **Le service de suivi** des EJB appelle les MBeans dans la couche d'applications afin de superviser le fonctionnement des EJB.

- **Le service de coordination** contacte les MBeans dans la couche système et la couche middleware afin de déterminer la correspondance entre les besoins d'un EJB et l'état d'un serveur d'applications.

Le coût d'appel diffère selon la complexité des opérations portées par le MBean contacté. La surcharge du service d'instrumentation est composée de deux parties, le coût des appels et la fréquence d'appels. La relation entre la surcharge et la fréquence d'appels est donc une relation proportionnelle : une fréquence important cause une surcharge plus importante.

La surcharge du service de suivi est affectée par la surcharge d'instrumentation des composants EJB et par la surcharge apportée par les appels de suivi. L'instrumentation des composants consiste à les représenter par des MBeans, ce processus est réalisé une seule fois au moment du déploiement de l'EJB. On peut donc négliger la surcharge de l'instrumentation. Quant à la surcharge d'appels de suivi, elle est proportionnelle à la fréquence d'appels.

Finalement, le coût de l'instrumentation du système devient plus important quand l'instrumentation est plus détaillée. Une instrumentation modérée des principales activités des services du système donne un coût acceptable et une connaissance satisfaisante de l'environnement. Quant au coût des appels de suivi, il devient moins important quand la fréquence d'appels est assez espacée. L'équilibre entre surcharge et précision pourrait être l'objet d'une prochaine étude.

6.3. Perspectives

Dans le cadre de notre modèle d'administration, nous définissons deux parties : une plate-forme d'administration et un service de coordination. Actuellement nous avons mis en œuvre les services qui constituent la plate-forme d'administration. Un travail important reste à effectuer pour fournir le service de

coordination. Le modèle de procédé que nous avons présenté en 5.3.1 est le premier pas pour la mise en œuvre du moteur de procédés. Quant au centre de prise de décision, des études d'expérimentations et des études théoriques sont nécessaires pour établir des règles standards de gestion de déploiement adapté dans la fédération.

Nous pensons que la gestion des ressources informatiques en générale, et la gestion d'applications en particulier devient de plus en plus une nécessité, et qu'il faut penser à l'administration comme un service non fonctionnel standard.

L'approche d'une fédération de serveur d'applications qui contrôle le déploiement d'une manière flexible est un pas pour l'ouverture vers une notion de coopération flexible et libre entre plusieurs parties. Ces parties peuvent être des entreprises de petite taille ou de taille moyenne, des universités, des centres de recherche, ou même des hôpitaux.

L'Internet a déjà offert le premier pas de coopération. Un niveau plus fort de coopération sera de permettre aux parties de partager leurs ressources matérielles et logicielles d'une manière libre et flexible. Une entreprise peut, par exemple, se joindre à une fédération en respectant certaines règles d'usages, et elle sera libre de quitter la fédération à tout moment. Chaque entreprise peut, par exemple, définir le quota de ressources matérielles à partager, et le type de logiciels à échanger.

La constitution d'un modèle de sécurité est essentielle pour une coopération sécurisée dans la fédération. Pour une fédération élargie où les serveurs d'applications ne sont pas tous des parties de confiance, la coopération doit être faite en différents degrés de sécurité. On peut par exemple créer différentes chartes à signer avant de joindre à la fédération. Selon le type de charte signer par le serveur, il sera accordé un niveau de permissions.

D'autre part, dans une fédération dynamique où les serveurs peuvent entrer ou sortir d'une manière automatique, il faut employer une technique de découverte des serveurs. Les serveurs de la fédération seront donc informés d'une manière dynamique de l'arrivé ou du retrait des autres serveurs. Les approches liées aux réseaux P2P (peer to peer) semblent un premier pas dans cette direction.

Finalement, notre modèle pour la constitution et la gestion de la fédération peut être élargi pour inclure non seulement les serveurs d'applications EJB mais aussi les autres plates-formes de composants tel que CCM ou DCOM.

Bibliographie

[ABR02] Abraham Kang, *“J2EE clustering, part 1”*, javaWold, February 2002, http://www.javaworld.com/jw-02-2001/jw-0223-extremescale.html.

[ADR01] Adrian Mos, John Murphy "*Performance Monitoring Of Java Component-Oriented Distributed Applications*", IEEE 9th International Conference on Software, Telecommunications and Computer Networks - SoftCOM 2001, Croatia-Italy, October 9-12, 2001.

[ALM00] N. Almasri, S. Frenot, *"Speech Integration in Medical Information Systems",* MedInfo 2000, Londres.

[ALMa02] N. Almasri, S. Frenot, "*Dynamic Instrumentation for the management of EJB-based applications"*, Journée Composants JC'2002, Grenoble.

[ALMb02] N. Almasi, S. Frenot, *"Dynamic Instrumentation for the management of EJB-based applications"*, DOA 2002, Poster, Californie, Octobre 2002.

[ALMc02] N. Almasi, S. Frenot, *"Instrumentation Dynamique pour l'administration d'applications à base d'EJB"*, Rapport de recherche de l’INRIA, rr-4481, INRIA : Le Chesnay Cedex – France, Juin 2002.

[ALU03] D. Alur, D. Malks, J. Crupi, *"Core J2EE Patterns: Best Practices and Design Strategies"* Second Edition, Upper Saddle River, NJ : Prentice Hall, 2003.

[ANA01] A. Anagnostaki, S. Pavlopoulos, D. Koutsouris, *"XML and the Vital Standard: the document-oriented approach for open telemédicine applications"* in Proc. MedInfo2001, V.Patel & al. (ed.), Amsterdam : IOS press, 2001, pp. 77-81.

[ASE01] J.I. Asensio, V.A. Villagr, J.E. Lopez-de-Vergara , R. Pignaton, J.J. Berrocal, "*Experiences in the management of an EJB-based e-commerce application*", in proceedings of the 8th HP OpenView University Association Plenary workshop (HPOVUA'01), Berlin, Germany, 2001

[ASH04] D. C. Ashmore , *“The J2EE Architect's Handbook: how to be a successful technical architect for J2EE applications”*, Lombard, Ill. : DVT press publisher, 2004.

[BAC00] F. Bachmann, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, R. Seacord, K. Wallnau, *“Technical Concepts of Component-Based Software Engineering”* [en ligne], Volume II, Pittsburgh : Carnegie Mellon University, May 2000, Technical Report CMU/SEI-2000-TR-008, http://www.sei.cmu.edu/staff/kcw/ 00tr008.pdf (consulté le 07/02/2005).

[BAR00] G. J. Badros, *“JavaML: A markup language for java source code”* [en ligne], In Proceedings of the Ninth International Conference on the World Wide Web, Amsterdam, The Netherlands : Elsevier Science B. V., May 2000, http://www9.org/w9cdrom/342/342.html (consulté le 07/02/05)

[BAU95] M. A. Bauer, H. L. Lutfiyya, J. W. Hong, J. P. Black, T. Kunz, D. J. Taylor, T. P. Martin, R. B. Bunt, D. L. Eager, P. J. Finnigan, J. A. Rolia, C. M. Woodside, and T. J. Teorey, *"MANDAS: Management of Distributed Applications and Systems"* Proceedings, International Workshop on Future Trends in Distributed Computing Systems (FTDCS95), Cheju, Korea (August 1995), pp. 200-206

[BEA] BEA Systems, Inc ., *"Achieving Scalability and High Availability for E-Business"* [en ligne], BEA white paper, USA: BEA Systems, 2003 http://dev2dev.bea.com/products/wlserver81/whitepapers/WLS_81_Clustering .jsp (consulté le 7/02/2005).

[BOL98] J J. Bolliger, and T. Gross, *"A Framework-Based Approach to the Development of Network-Aware Applications"*, IEEE Transactions on Software Engineering, May 1998, vol. 25, n°5, pp. 376-390

[BRU00] E. Bruneton, M. Riveill, *"JavaPod : une plate-forme à composants adaptable et extensible"* [en ligne], Rapport de recherche de l'INRIA, rr-3850. INRIA : Le Chesnay Cedex – France, Janvier 2000 http://www.inria.fr/rrrt/rr-3850.html (consulté le 7/02/2005).

[CAP01] J. Caple, M. Haim, *"The art of EJB deployment"* [en ligne], JavaWorld article, San Francisco, USA : JavaWorld, http://www.javaworld.com/javaworld/jw-08-2001/jw-0803-ejb.html (consulté le 8/02/2005).

[CAS90] J.D. Case, M. Fedor, M.L. Schoffstall, and C. Devin. *"Simple Network Management Protocol (SNMP)"*. The Internet Engineering Task Force, Network Working Group, Request for Comments 1157, USA : Computer and Information Science, May 1990.

[CIM03] Distributed Management Task Force (DMTF), *"CIM Technical Notes"*, January, 2003. http://www.dmtf.org/education/technote_CIM.pdf (consulté le 8/02/2005)

[CIM04] Distributed Management Task Force (DMTF). "Common Information Model", V2.9, 2004. http://www.dmtf.org/standards/cim (consulté le 8/02/2005)

[DAV02] P.C. David, T. *"Ledoux: An Infrastructure for Adaptable Middleware"*, LNCS, 2002, N°2519, pp. 773-790.

[DCOM] Microsoft Corporation, *"Distributed Component Object Model"* [en ligne], http://www.microsoft.com/com/tech/DCOM.asp (consulté le 8/02/2005).

[DEB02] M. Debusmann, M. Schmid, R. Kroeger, *"Generic Performance Instrumentation of EJB applications for service-level Management"*, NOMS 2002, Florenz, 15-19, April 2002

[DER94] J.C. Derniame and all, *"Life Cycle Process Support in PCIS"*, Proceedings of PCTE'94, San Francisco, November 1994.

[DER99] J.C. Derniame, B. Ali Kaba and D. Wastell., *"Software process: Principles, Methodology, Technology"*, Lecture Notes in Computer Science, 1999, N°1500.

[DIN01] P. Dinda, T. Gross, R. Karrer, B. Lowekamp, N. Miller, P. Steenkiste, and D. Sutherland. *"The Architecture of the Remos System"*. In IEEE Symposium on High Performance Distributed Computing (HPDC10), San Francisco, CA, 2001.

[DOCK] R. Hall, D. Heimbigner, and A.L.Wolf, *"Software Dock"* [en ligne], Colorado, USA : Software Engineering Research Laboratory (SERL), 1999, http://www.cs.colorado.edu/serl/cm/dock.html (consulté le 8/02/2005)

[EJBa03] Sun Microsystems, *"Enterprise Java Beans Specification"* [en ligne], Version 2.1, Novembre 2003, http://java.sun.com/products/ejb/ (consulté le 8/02/2005)

[EJBb03] Sun Microsystems, *"Enterprise Java Beans Specification"*, Chapitre 19, In Support for Distribution and Interoperability, Version 2.1, Novembre 2003.

[EMA02] Emmanuel Cecchet, Julie Marguerite and Willy Zwaenepoel, *"Performance and Scalability of EJB applications"*, 17th ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (Oopsla 2002), Seattle, WA, 4-8 november 2002.

[FLEa02] M. Fleury, S. Stark, *"JBoss Administration and Development"*, Indianapolis, Ind. : Sams, 2002, 512 p.

[FLEb02] M. Fleury, J. Lindfors, *"JMX: Managing J2EE with Java Management Extensions"*, Sams publisher, first edition, Indianapolis, Ind. : Sams, 2002, 394p.

[FRE02] S. Frenot M. Avin-Sotres, N. Almasri, *"EJB components Migration Service and Automatic Deployment"*, Rapport de recherche de l'INRIA, rr4480, INRIA : Le Chesnay Cedex – France, Juin 2002.

[FRE03] S. Frénot, T. Balan, *"A CPU Resource Consumption Prediction Mechanism for EJB deployment on a Federation of Servers"*, 2003. Workshop on Benchmarking, OOPSLA'2003, Anaheim, California, USA.

[GOK02] A. Gokhale, D. C. Schmidt, B. Natarajan, N. Wang: "*Applying Model-Integrated Computing to Component Middleware and Enterprise Applications*". Communications of the ACM, October 2002, Vol. 45, No. 10.

[GRO01] W. Grosso, *"Java RMI"*, 1st edition, Sebastopol, CA : O'Reilly publisher, 2001.

[HALa99] R. S. Hall, *"Agent-based Software Configuration and Deployment"*, Ph.D thesis of University of Colorado, USA : University of Colorado, 1999.

[HALb99] R. S. Hall, D. Heimbigner, and A. L. Wolf. *"A Cooperative Approach to Support Software Deployment Using the Software Dock"*. Proceedings of the 21st International Conference on Software Engineering, Los Angeles, CA, USA, May 1999, pp. 174-183.

[HAN04] J. Hanson, *"Pro JMX: Java Management Extensions"*, Berkeley, USA : Apress publisher, 2004.

HOF93] C. Hofmeister, Purtilo J., *"Dynamic Reconfiguration in Distributed Systems: Adapting Software Modules for Replacement"*, Proc. of the 13th International Conference in Distributed Computing Systems (ICDCS'93), IEEE Computer Society Press, Pittsburgh, Pennsylvania, May 25-28, 1993, pp. 101-110.

[ISO91] ISO. Information Processing Systems- Open Systems Interconnection- Common Management Information Protocol Specification(CMIP). International Organization for Standardization, International Standard 9596-1, 1991.

[J2EE03] Sun Microsystems, *"Java 2 Platform Enterprise Edition Specification"* [en ligne], V1.4, November 2003 http://java.sun.com/j2ee/ (consulté le 8/02/2005)

[J2SE04] Sun Microsystems, *"Java 2 Standard Edition"* [en ligne], V5.0 beta, May 2004, http://java.sun.com/j2se/1.5.0/docs/guide/jmx/index.html (consulté le 8/02/2005)

[JBOSS] JBoss group, *"JBoss Application server "* [en ligne], http ://www.jboss.org (consulté le 8/02/2005)

[JMS] Sun Microsystems, *"Java Message Service"* [en ligne], http://java.sun.com/ products/jms (consulté le 8/02/2005)

[JMX02] Sun Microsystems, *"Java Management eXtensions : Instrumentation and Agent specification"* [en ligne],, 2000, http://java.sun.com /products /JavaManagement/ (consulté le 8/02/2005)

[JMX03] Sun Microsystems, *"Java Management eXtensions : Remote API Specification"* [en ligne], V1.0 2003, http://java.sun.com /products /JavaManagement/ (consulté le 8/02/2005)

[JNDI] Sun Microsystems, *"JNDI: Java Naming and Directory interface"* [en ligne], http://java.sun.com/products/jndi/ (consulté le 8/02/2005)

[JONAS] ObjectWeb Consortium, *"JOnAS: Java Open Application Server"* [en ligne], http://jonas.objectweb.org (consulté le 8/02/2005)

[JSR77] Sun Microsystems, *"JSR 77: J2EE management"* [en ligne], 2001, http://www.jcp.org/en/jsr/detail?id=77 (consulté le 8/02/2005)

[JWS] Sun Microsystems, *"Java Web Start"* [en ligne], http://java.sun.com/products/ javawebstart (consulté le 8/02/2005)

[KAT99] M. J. Katchabaw, S. L. Howard, H. L. Lutfiyya, A. D. Marshall, M. A. Bauer. *"Making Distributed Applications Manageable Through Instrumentation"*, Journal of Systems and Software , 1999, Vol. 45, pp. 81-97

[KLE88] Klerer S. M, *"The OSI Management Architecture: an Overview"*, IEEE Network, mars 1988, vol. 2, N°2, pp20-29.

[KOB03] E. Koblentz, *"CIM: A Work in Progress"* [en ligne], eWeek Enterprise News & Reviews, 2003, July 28, http://www.eweek.com/article2/ 0,1759,1496119,00.asp (consulté le 8/02/2005).

[KRE01] H. Kreger, *"Java Management Extensions for application management"*, IBM Systems Journal, 2001, Vol 40, No 1.

[KRE02] H. Kreger, W. Harold, L. Williamson, *"Java and JMX: Building Manageable Systems"*, Harlow : Addison Wesley, 2002, 400 p.

[LES02] V. Lestideau, *"Un environnement de déploiement automatique pour les applications à base de composants"*, ICSSEA02,Paris, France, December 2002

[LES03] V. Lestideau, *"Modèle et environnement pour configurer et déployer des systèmes logiciels"*, thèse de doctorat en Informatique, Savoie : Université de Savoie, Ecole Supérieure d'Ingénieurs d'Annecy, 19 décembre 2003.

[LOP01] J.E. Lopez-de-Vergara, V.A. Villagra, J.I. Asensio, Jose I. Moreno, J.J. Berrocal, *"Management of E-Commerce Brokerage Services"* in Proceedings of the 5th World Multiconference of Systemic, Cybernetics and Informatics (SCI2001). Orlando, Florida, USA. July 22-25, 2001.

[LOW98] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. *"A Resource Query Interface for Network-Aware Applications"*. Seventh IEEE Symposium on High-Performance Distributed Computing, Chicago, Illinois, July 28-31, 1998.

[MAR98] J-P Martin-Flatin, S. Znaty and J-P Hubaux. *"A Survey of Distributed Network and Systems Management Paradigms"*. Technical Repport SSC, Switzerland : Ecole Polytechnique Federale de Laussanne EPFL, Aug 1998.

[MEL01] H. Meling and B. E. Helvik, *"ARM: Autonomous Replication Management in Jgroup"*, in Proceedings of the 4th European Research Seminar on Advances in Distributed Systems (ERSADS), Bertinoro, Italy, May 2001.

[MEL02] H. Meling, A. Montresor, O. Baboglu and B. E. Helvik, *"Jgroup/ARM: A Disrtibuted Object Group Platform with Autonomous Replication Mangement for Dependable Computin"*, Technical Report UBLCS 2002-12, October 2002.

[MER04] N. Merle and N. Belkhatir, *"Open Architecture for Building Large Scale Deployment Systems"*,The 2004 International Conference on Software Engineering Research and Practice (SERP'04),Las Vegas, Nevada, USA, June 2004.

[MIL00] N. Miller, P. Steenkiste, *"Collecting Network Status Information for Network-Aware Applications"*, Proceedings of the conference of Computer Communications, IEEE INFOCOM'2000, Israel, March 26-30, 2000.

[MKBEEM] MKBEEM project, Multilingual Knowledge Based European Electronic Market place [en ligne], http://mkbeem.elibel.tm.fr/ (consulté le 8/02/2005)

[NAN03] O. Nano, M. Blay, *"Une approche MDA pour l'intégration de services dans les plates-formes à composants"*, In Journées Objets, Composants et Modèles, Vannes, 5 février 2003. GDR ARP

[NIE95] O. Nierstrasz, L. Dami, *“Component-Oriented Software Technology”* In O. Nierstrasz and D.Tsichritzis (Eds.), *Object-Oriented Software Composition*, London : Prentice Hall,1995, pp.3-28.

[OMG] OMG Object Management Group, *“Common Object Request Broker Architecture”* [en ligne], http://www.corba.org/ (consulté le 8/02/2005)

[OPEa04] The Open Group, *"Application Response Measurement- ARM"* [en ligne], V4.0, April 2004, http://www.opengroup.org/management/arm.htm/ (consulté le 8/02/2005)

[OPEb04] The Open Group, *“Application Instrumentation and Control - AIC"* [en ligne], V1.4, November 1999, http://www.opengroup.org/management/aic.htm/ (consulté le 8/02/2005)

[PLA98] Plasil, F., and Stal, M. “*An architectural view of distributed objects and components in corba*, *java rmi and com/dcom*”. Software--Concepts and Tools, 1998, Vol. 19, N°1, pp. 14-28.

[PAL01] N. De Palma, *"Services d’Administration d’Application Reparties"*, thèse de doctorat, Université Joseph Fourier de Grenoble, 2001.

[PIT01] E. Pitt, K. McNiff, and K. McNiff, *"Java RMI: The Remote Method Invocation Guide"*, Harlow : Addison-Wesley publisher, 2001, 304 p.

[RAC00] G. Rackl, M. Lindermeier, M. Rudorfer, B Suss. *"MIMO: An Infrastructure for Monitoring and Managing Distributed Middleware Environments”*. Middleware 2000 IFIP/ACM International Conference on Distributed Systems Platforms, April 2000, New york.

[RIC99] Rich Wolski, Neil Spring, and Jim Hayes, *“The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing”*, Journal of Future Generation Computing Systems, October, 1999, Vol. 15, N° 5-6,pp. 757-768.

[RMI] Sun Microsystems, *“The Remote Method Invocation”* [en ligne], http://java.sun.com/rmi/ (consulté le 8/02/2005)

[ROM02] Ed Roman, S. Ambler, T. Jewell, *“Mastering Enterprise JavaBeans”*, Second Edition, New York : Wiley Computer Publishing, 2002, 639 p.

[SHI02] Shiping Chen, Ian Gorton,Anna Liu, and Yan Liu, *"Performance Prediction of COTS Component-based Enterprise Applications"*, CBSE5, Orlando, Florida, USA, May 2002

[SHIELD] InstallShield, *“InstallShield - Software Installation and Enterprise Application Management Solutions”* [en ligne], USA : InstallShield, http://www.installshield.com (consulté le 8/02/2005)

[SIN02] L. Sing, *“From Black Boxes to Enterprises"* [en ligne], IBM article, New York : IBM, 2002, http://www-106.ibm.com/developerworks/java/library/j-jmx1/ (consulté le 8/02/2005)

[TYL02] Tyler Jewell, *"EJB 2 Clustering with Application Servers"* [en ligne], San Fransisco, USA : O'Reilly & Associates, Inc., 12/15/2000, http://www.onjava.com/pub/a/onjava/2000/12/15/ejb_clustering.html (consulté le 8/02/2005)

[SUB01] N. Subramanian and L. Chung, *"Metrics for Software Adaptability"*, Proc. Software Quality Management (SQM 2001), April 18--20, Loughborough, UK.

[SZY02] C. Szyperski, *"Component Software : Beyond Object-Oriented Programming"*, second edition, Paris : Addison-Wesley, 2002, 589 p.

[TAY96] S.D. Taylor. *"Distributed Systems Management Architectures"*. Master Thesis in Computer Science, Canada : University of Waterloo, 1996.

[THO02] T.G. Thomas, *"J2EE Application Management, The Power of JMX"* [en ligne], TheServerSite.com, 2002, http://www.theserverside.com/articles/article.tss?l=AdventnetJMX (consulté le 8/02/2005)

[TIE01] B. L. Tierny, D. Gunter, J. Lee, M. Stoufer, *"Enabling Network-Aware Applications"*, in proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), August, 2001, San Francisco, CA.

[TIVOLI] Tivoli, *"IBM Tivoli Configuration Manager"* [en ligne]. http://www306.ibm.com/software/tivoli/products/config-mgr (consulté le 09/02/2005)

[WBEM] Distributed Management Task Force (DMTF). *"Web-Based Enterprise Management"*. Portland : DMTF, 2001.

[WEA04] J. L. Weaver, K. Mukhar, J. P. Crume, *"Beginning J2EE 1.4: From Novice to Professional"*, 1st edition, Berkeley, USA : Apress publisher, 2004, 624 p.

[WOL96] A. Wollrath, R. Riggs and J. Waldo, *"A distributed object model for the java System"*, 2nd conference on Object-Oriented Technologies and Systems COOTS, 1996, Toronto, Ontario, Canada

[XSL99] W3C*, "XSLT transformations"* [en ligne],version 1.0, W3C Recommendation, Novembre 1999, http://www.w3.org/TR/xslt (consulté le 8/02/2005).

[YAN02] L. Yan, *"Performance and Scalability Measurement of COTS EJB Technology"*, 14th Symposium on Computer Architecture and High Performance Computing (SCAB-PAD'02), October 28 - 30, 2002 , Brazil.