# EJB components Migration Service and Automatic Deployment

Stéphane Frénot, Sotres Miguel Avin, Nada Almasri

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

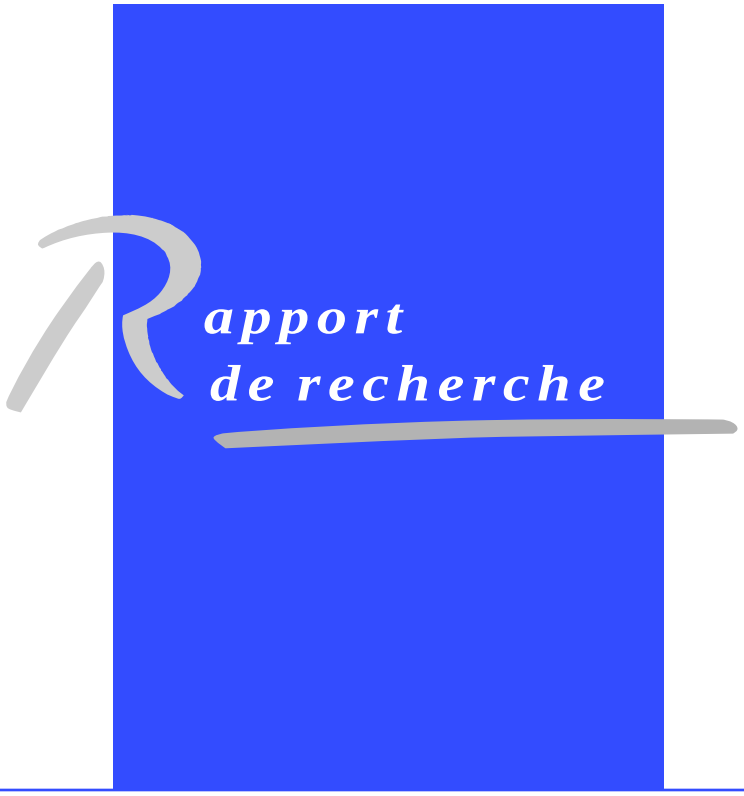
# *EJB components Migration Service and Automatic Deployment*

Stéphane Frénot , Miguel Sotres Avin , Nada Almasri

**No 4480**

Juin 2002

THÈME 1




ISSN 0249-6399 ISRN INRIA/RR--4480--FR+ENG

# EJB components Migration Service and Automatic Deployment

Stéphane Frénot* , Miguel Sotres Avin , Nada Almasri†

Thème 1 — R´eseaux et systèmes
Projet ARÈS

Rapport de recherche n ° 4480 — juin 2002 — 15 pages

**Abstract:** The explosive growth of the World Wide Web and the expected arrival of the fourth generation networks pose significant challenges to distributed system developers. Future distributed applications are expected to support the increasing mobility of users and to be dynamic enough to maintain the load-balancing between the different hosts constituting the distributed environment. With the evolution of the wireless technology users are increasingly mobile, and they expect to reach personalized information and to access specified applications regardless of their physical location. Indeed, the use of application servers allows defining efficient distributed applications, by providing tools facilitating their development, their deployment and their execution. However, even if the application servers present a powerful economic solution, they are not dynamic enough compared to the principles of the user's mobility. In our approach, we suppose that users change the hosting network regularly, and that this network has characteristics that do not enable users to reach the applications which they usually use in their original network. By authorizing the mobility of the applications between application servers, we enable the applications to follow the user when changing his/her original network. In this report, we propose integrating a standard service into the application servers to enable migrating hosted applications. The service makes it possible to look up, migrate and re-deploy an application on a federation of application servers while taking into account certain criteria of mobility. The service integrates two principal components: the first component allows locating the initial application and the second one performs an intelligent migration of this application.

**Key-words:** code migration, resources migration, EJB components, application servers

*(Résumé : tsvp)*

* stephane.frenot@insa-lyon.fr
† nalmasri@insa-lyon.fr

Unit´e de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
T´el´ephone : 04 76 61 52 00 - International: +33 4 76 61 52 00
T´el´ecopie : 04 76 61 52 52 - International: +33 4 76 61 52 52

# SERVICE DE MIGRATION ET DE DEPLOIEMENT AUTOMATIQUE D'EJB

**Résumé :** La croissance explosive du World Wide Web et l'arriv´ee pr´evue des r´eseaux de quatrième g´en´eration soulèvent de nouveau d´efis pour les d´eveloppeur de systèmes distribu´es. Les applications distribu´ees du futur devront prendre en compte la mobilit´e croissante des utilisateurs et devront être suffisamment dynamiques pour maintenir l'´equilibre de charge entre les diff´erents hôtes de l'environnement distribu´e. Avec l'´evolution des technologies sans-fils les utilisateurs sont plus mobiles, et ils s'attendent à pouvoir atteindre de l'information personnalis´ee et à pouvoir acc´eder à des applications sp´ecifiques ind´ependamment de leur localistaion physique. En fait, l'utilisation de serveurs d'applications permettent de d´efinir des applications distribu´ee efficaces en fournissant des outils qui facilite leurs d´eveloppement, leurs d´eploiement et leurs ex´ecutions. Cependant, même si les serveurs d'applications repr´esentent une solution ´economique et puissante, ils ne sont pas suffisamment dynamiques pour prendre en compte la mobilit´e des utilisateurs. Dans notre approche, nous supposons que l'utilisateur change de r´eseau d'accueil de manière r´egulière, et que le r´eseau d'accueil ne permet pas forc´ement aux utilisateurs d'acc´eder aux applications qu'il a l'habitude d'utiliser sur son r´eseau d'origine. En autorisant la mobilit´e des applications entre serveurs d'applications, nous permettons à l'application de suivre l'utilisateur lorsque celui-ci change de r´eseau.

Dans ce rapport, nous proposons d'int´egrer un service standard dans un serveur d'application afin de permettre la migration des applications h´eberg´ees. Le service permet de rechercher, migrer et red´eploier une application sur une f´ed´eration de serveurs d'applications en prenant en compte certains critères de mobilit´e. Le service intègre deux composants principaux : le premier composant permet la localisation d'une application initiale et le second r´ealise la migration intelligente de l'application.

# 1 Introduction

Users mobility became a reality that we need to consider while building information systems. Although the use of Web interfaces gives a remote access to the applications of the information systems, this solution does not work for a certain number of cases. For example, in the case of a secured Intranet, if the user is on an external network, he/she cannot reach the intended application. Another case is when dealing with applications of a high bandwidth consumption. In this case, the remote access is highly constrained by the means of communication used to inter-connect the two networks. One of the solutions under consideration for the first case is to use mechanisms like mobile IP [20] which allows saving the IP address of the users, even if they are not any more located in the original network. In this case a virtual tunnel is created between the user's host in the external network (foreign network) and the original network (home network) to simulate that the user's node belongs to the original network. This solution however reveal problems concerning queries redirection and packets encapsulation.

In our approach, we would like to allow applications to follow the user. In this case we prefer considering the term "naming" rather than that of the mobility. Indeed while in the mobility concept it is intended to maintain the same user environment wherever the user is, in the naming concept the user can be located in a different environment which offers some other specific fuctionalities. In this case the users must be able to use the local applications (and resources) of the information system as well as the applications offered by their home network. It is then necessary to migrate these applications to the users.

In this report we are more interested in the mechanisms related to the mobility of the code rather than the mechanism of caching or requests redirection. We consider that we have specific environments where the available applications are hosted by application servers. Thus each network visited by the user has one (or several) application server hosting a set of component-based applications. We present in this article an architecture that allows migrating the sources (application components and data) automatically between application servers according to the user's needs. We first present the principles of application servers and the principles related to the mobile code. Then we present the proposed architecture allowing application servers to exchange applications dynamically. Finally we discuss the implementation details concerning exchanging components between two application servers.

# 2 Application servers and mobility

## 2.1 Application servers

Application servers became largely exploited by the industry [22]. There are currently three industrial platforms of application servers : CORBA-CCM platform [10], Windows DNA platform from Microsoft [6] and J2EE platform from Sun [13]. These three platforms share a number of concepts facilitating the implementation of the component-based distributed applications.

Component : An application is described by a number of components which cooperate. These components are characterized by a manipulating interface. According to the platform, the interface

description language can be either a neutral language (IDL) or the same language used for implementing the component. The client and the server communicate without handling their respective locations ; they use objects of low level (stub and skeleton) which mask the network and the structure transfer problems.

Deployment : A component-based application hosted by the application server is activated only when a client tries to access it. Before being accessed, the application must be ready for use, that is, the application server should prepare a suitable execution environment for the application according to some predefined elements.

Application server's basic services : Application servers provide basic services to the hosted applications. These services release the component from "traditional" tasks like persistence, transaction management and inter-application message exchange.

These principles are the standard functionalities offered in one way or another by the three platforms component-based applications. The J2EE Platform has two advantages compared to CORBA-CCM and to Windows DNA [19]. The first advantage is that it is not related to a specific product as the Windows DNA platform, and the second one is that it already has many implementations of application servers, some of them are commercial [4], [5], [7] while others are free open-sources [15], [14].

## 2.2 J2EE Platform

We present in this section some principles that are specific to the J2EE platform [18] and that seem important to be highlighted.

Execution platform : The execution platform of the components implemented respecting to the J2EE specification rely on the Java language for the execution of the code on one hand, and on the principle of the EJB container for the execution of the components on the other hand.

- Java : The J2EE depends entirely on the Java virtual machine JVM. Thus, J2EE application servers may exchange components easily without the need to adapt the components for the underlying application server.

- EJB container : The components hosted by a J2EE platform are called EJB (Enterprise Java Beans) and are executed in an EJB container. All the activated components are executed in the container environment which manage the access to the components during their life cycle.

Components typology : J2EE platform offers a distributed components typology. The EJB are divided into four categories, corresponding to four application uses :

- Stateless Session EJB : It represents a service that is accessed in the same way by all clients, thus it doesn't need to maintain any conversational state.

- Statefull Session EJB : It represents a service that will be associated to a specific client. It allows implementing a service having different conversational states with the client.

- Entity EJB : It represents a line of a table in a database.

- Message Driven EJB : It is invoked automatically by a messaging system.

Deployment : In order to implement a component-based application in an application server, the application should define how it wants to use the standard services offered by the application server and how its different components interact with each others. This is done using XML descriptor

files structured according to some predefined DTD files (Data Type Definition) presented by Sun Microsystems and by EJB's application server vendors. Each application server that adds additional functionality can accept a corresponding new descriptive parameter in the descriptor file. This approach allows plotting the relation between the component and its resources in a later phase following the components implementation. It is, thus, possible to adapt the component according to the available resources (different databases, external electronic mail system, etc...). Using the descriptor files and the components implementation, the application server generates the necessary classes in order to support all the components of the applications with the required services during their life cycle. This preparation process is what is known as the deployment process. When an EJB is deployed on the server, its instantiation is then restricted to the EJB's container. Clients interested by the EJB would then get a remote reference to the container by accessing a naming service where all EJBs available on the application server are cited. All calls to the EJB would be then intercepted and managed by the container. bility Principles

## 2.3 Mobile code

### 2.3.1 Mobility Principles

Current component-based distributed systems provide an abstraction environment where the components can interact regardless to their location. This makes it possible for the application developer to deploy the components on various hosts according to a given criteria without modifying the code implementation. On the other hand once the component is associated to a specific host it can't migrate any more.

This represents a considerable limitation for long-duration applications since they are not able to adapt to the modifications of the execution conditions. The code mobility can be defined as the capacity to dynamically change the links between the code's fragments and the place where it is running. Several research works on the transfer of code between the nodes of a network were carried out [12], [23].

The advantages related to the use of mobile components are [8] :

- Load-balancing : By moving the components of an application server towards another one, the load of the host machines constituting the distributed architecture can be equally balanced to provide all hosts with the same level of performance.

- Communication performance : The components, which interact regularly, can be moved on the same node in order to reduce the costs of communication. This characteristic is very important within the users mobility context.

- Availability : Components of an application can be moved towards different nodes in order to insure the application's availability in case of an unexpected failure in the original node.

- Reconfiguration : Migrating components allow maintaining a service rendered by the component-based application to be continuously available even at the time of an update of the hosting application server.

### 2.3.2 Mobility Typology

Current mobile code systems offer two types of code mobility [9] : Strong mobility and Weak mobility. The Strong type represents the capacity of a system to migrate the code as well as its state of execution. Weak mobility is the capacity of a system to migrate only the code. It can be accompanied by initializing data, but not the running state. One serious problem concerning the code migration is the management of the resources used by the code.

### 2.3.3 Resource Migration

Once a component is moved towards another host, all links with the resources must be reorganized according to their types [9][3][2][16]. The link between a component and a resource has three types :

- By identifier : it is the strongest type of links. The component must be dependent constantly on a specific and single resource. This class of links is used when the resource cannot be substituted by another one.

- By value : after its transfer, the component must be able to use a resource with a value identical to that of the original host. It is the case when the component wants to reach the contents of the resource locally.

- By type : it is the weakest class of links. The component requires that the resource stay continually compatible with a certain type. This class of links is usually used for ”standard” resources like printers.

In addition a resource can be either mobile or not. For example a resource of type ”storage” can be transferable while a resource of type ”printer” can’t be. The resources can be marked as ”free” or ”fixed”. The first one can be transferred toward another node, while the second is always associated with the same node. This characteristic is established according to the constraints of the application. For example, even if it is possible to transfer the contents from a database across the network, this would highly affect the performance. On the other hand, it would be better to prevent this kind of transfer for security reasons.

After we presented the basics of application servers and component mobility, we present in the next section our adopted architecture to perform the migration service.

# 3 Architecture

The basic services offered by the application server allow implementing component-based applications in a simple and a fast way without the need to consider non-functional duties such as security, transaction, persistence, etc. Applications implemented in such a way are still centralized in the application server where the components are deployed. The migration solution would then allow migrating components between application servers located in different networks. We chose more particularly to handle the J2EE compliant application servers since the platform is on one hand available and on the other hand open enough to be able to test our migration architecture. We consider that the application servers are inter-connected and know each other mutually.

The architecture allows dynamic migration for EJB components toward an application server located on the LAN where the user is currently situated. In this context, we use the weak mobility mechanism since our objective is to enable migrating the deployment package (jar file) in order to re-deploy the components locally on the target machine. The mobility of the deployment file is thus a passive mobility, i.e. independent of the execution of the code.

The architecture proposes adding a new standard service into application servers beside its existing ones to allow the localization, the migration and, the local deployment of an EJB component. Thus, if a client sends a request to the application server asking for an EJB component that doesn't exist on the local application server, the request should not be rejected and the client should not get an exception (error message). Instead, the application server searches the required service on other connected application servers, transfer it, and then deploy it locally in order to be available for the client.

To implement this approach the naming and directory service (JNDI) which is one of the standard services of the application server is extended by adding a localization service and a component transfer service. We then modified the JNDI service so that : if the component does not exist locally, it looks for it in other application servers. If the component is found, several solutions for accessing the component can be selected, in particular two main solutions are proposed : either the remote access to the component (redirection), or the migration and the local deployment of the component. To implement this approach, we set up two services. A service allowing the localization of the components and a service allowing their transfer.

## 3.1 Localization Service

In a classical application server, when a client required an EJB component that does not exist locally, an exception is raised and the client must redo another request in order to re-execute a new search. In our approach, the client's code should not be modified. It is thus necessary to either redefine a new naming service or insert between the client and the existing naming service a new service that behave on behalf of the naming service. We chose to insert a "proxy JNDI" to handle the client calls.

- JNDI Proxy : "JNDI proxy" has the responsibility to search for the component on the local directory and if it is not found it looks for it on the other application servers. Its API is the same as the one of the java naming service JNDI (lookup method). However an important question is : how this JNDI proxy can find the component on a "non-local" application server ? To answer this question several techniques were studied in order to locate the component. These techniques are based on the protocols of service discovery.

- Service discovery : In order to know what application servers are to be contacted to search for a remote component, several solutions were presented. The most promising one is to use a service discovery protocol [17][1][11] in order to register the active application servers. We did not yet consider this solution. At the mean time our proxy JNDI uses a hard coded list of application servers to be contacted. The use of a localization service (discovery) allow searching only certain servers, according to their proximity or their availability. The UML sequence diagram (figure 1) presents the sequence of calls to move a component. The JNDI proxy masks the call to the local JNDI (sequences

1, 2, 3, 4). If the component is not found locally an exception is raised by the local JNDI. It is intercepted by the JNDI proxy which handles the search of another location of the component (sequences 5, 6, 7). Once the service is located on a remote application server, the JNDI proxy calls the transfer service in order to migrate the service locally (sequence 8).

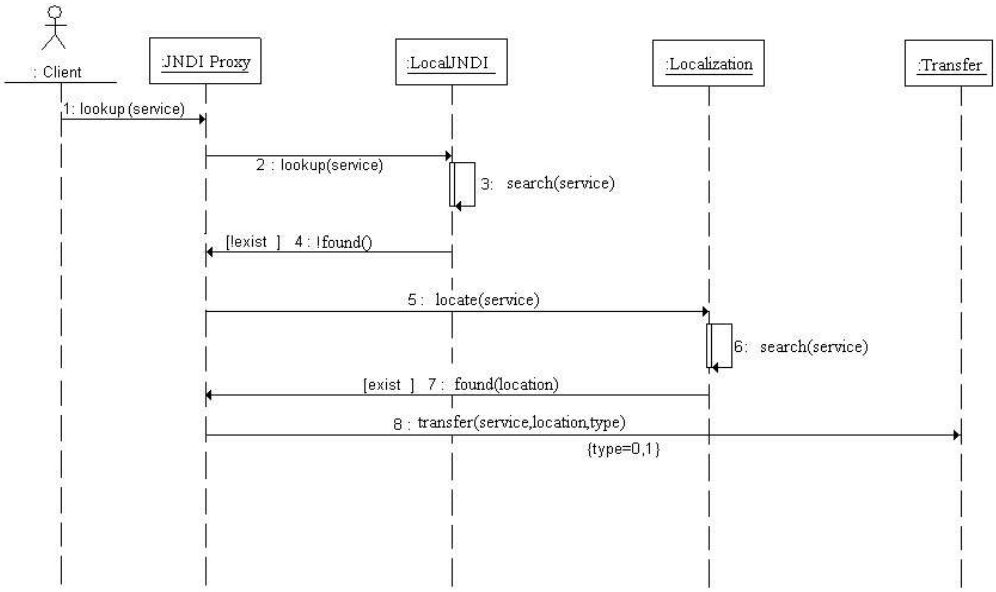



FIG. 1 – JNDI Call Sequence

## 3.2 Transfer Service

Once the localization is accessed by the JNDI proxy, a decision on the type of transfer should be taken. There can be several kinds of transfer :

- No transfer : In this case the component refuses to be moved. The JNDI proxy JNDI then returns to the client a reference on the remote container of the EJB component.

- Free transfer : The component is able to migrate between the application servers. For example a Stateless Session EJB that does not use further resources can be migrated between the servers without causing any problem.

- Multiple transfer : The component uses other components. A decision must be made to know if the other associated components must (or can) also be migrated.

- Transfer without the resources : The external resources of the component are not transferable (volume , confidentiality constraints...). In this case the local application server must obtain the references on the remote resources, so that the migrated component can remotely contact them.

- Transfer with resources : The component uses external resources (access to a small data base...) that can be easily moved across the network. Whether the resource can migrate or not is a decision that should be taken by the component itself. In this case it is necessary to recreate the suitable resource environment on the local server that would receive the migrated component with its resources.

The type of transfer is a part of the external characteristics of the component and is related to the initial deployment. We thus added a ¡Migration¿ tag in the deployment DTD to indicate the different cases of migration. The DTD is analyzed by an XML parser provided by the application server and the result of the analysis is used by the transfer service to make the adequate decision. The choice of transfer can be also made initially according to the type of the EJB component to migrate. Indeed a Stateless EJB is obviously easier to migrate than a Statefull one which is itself easier to migrate than an Entity which requires handling the contacts with the database.

# 4 Implementation

We implemented our approach on a platform based on several hosts of application servers (WebLogic6 application server) that are situated on the same LAN. An initial server hosts two linked components, when a user changes the application server, the second one has the responsibility to search the component on the original application server. Figure 2 illustrates our scenario.

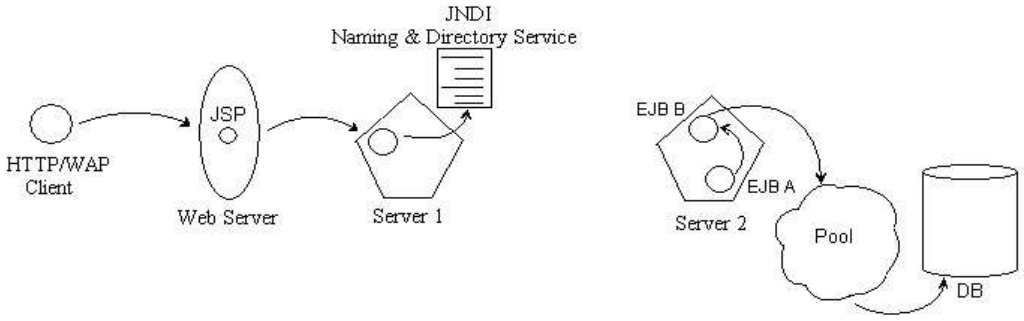



FIG. 2 – Test Architecture

In this scenario the client is a Web server which accesses the remote component by the intermediary of a servlet. To look for the component, the servlet initiate a reference to the JNDI proxy which masks the local naming directory JNDI. With this scenario we can validate the first four types of migration : request redirection (Without transfer), migration unit (free transfer), multiple transfer and transfer with redirection of resource. In all the types, the transfer service accesses the component deployment file to analyze the migration instructions.

## 4.1 Query Redirection

In this migration type, the component is not moved. However this type enables us to validate the localization service which is responsible of finding the container of the remote component and to return its reference to the client. The following diagram (figure 3) illustrates the sequence of calls for this solution.

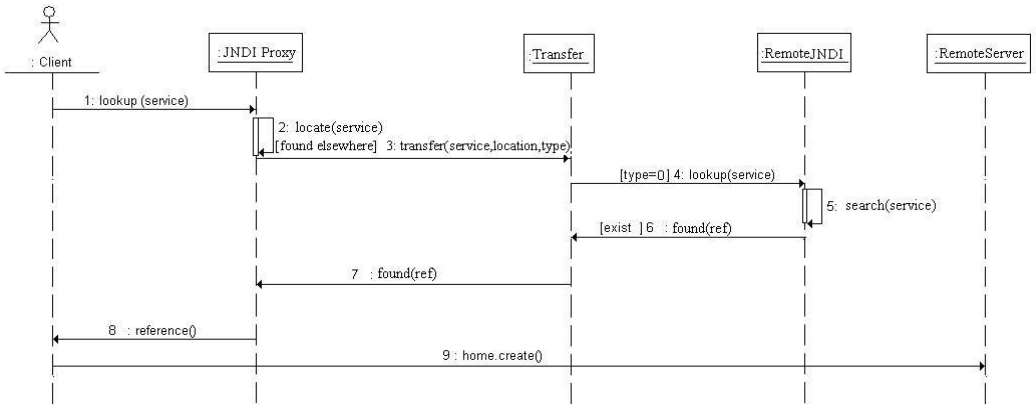



FIG. 3 – Component Localisation and Request Direction

When the transfer service receives a request for migration (sequences 1, 2, 3), it asks the remote JNDI the reference of the EJB container (sequences 4, 5, 6, 7, 8, 9) Obviously this mode is used to validate a possible protocol of a dynamic service discovery. We however did not go on in this direction.

## 4.2 Multiple Migration

Multiple migration covers the free migration. In this migration type the service consists of two components. The deployment descriptor of the first component indicates a reference on the second component. The file analysis is then recursive and the deployment can be done starting from the root component and going down in the tree of the components while choosing for each component its type of transfer.

The diagram in figure 4 represents the initial environment before the migration. In this configuration, the deployment file indicates that EJB A uses EJB B (ejb/Product). The transfer service thus will continue the analysis to figure out how to migrate EJB B. It can choose to either migrate the two components, or to migrate only the initial component.

- Migration of the two components : In this case (figure 5), during the use of the components the references will be resolved by the local naming directory service of the second application server (calls 2, 3).

- Migration of the initial component : In this case (figure 6), the JNDI proxy does not find the second component in the local naming directory (calls 1,2,3) and must thus make it accessible using

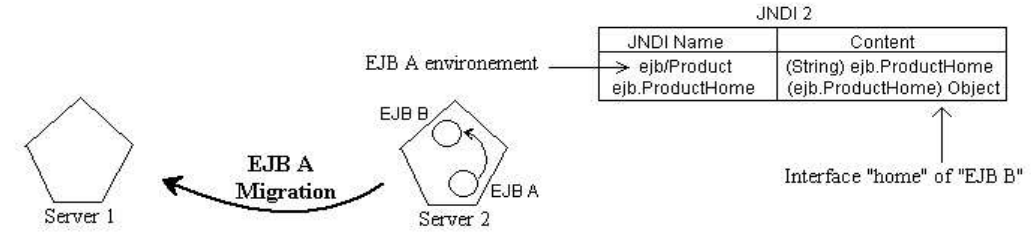



FIG. 4 – Initial Environment

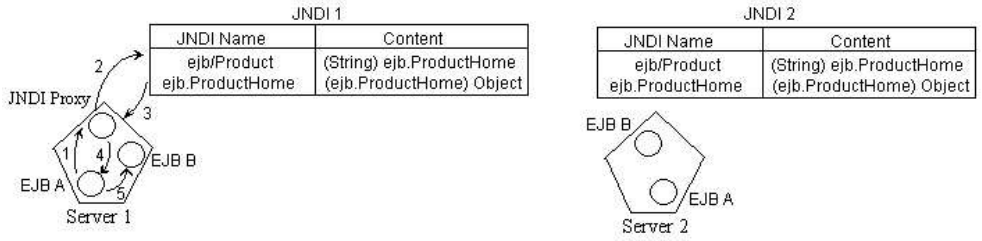



FIG. 5 – All EJB Components Migrate

the localization service and the redirection type (no transfer mode) for the second component (calls 4, 5, 6, 7). Obviously in this case, it is possible thereafter to migrate the component

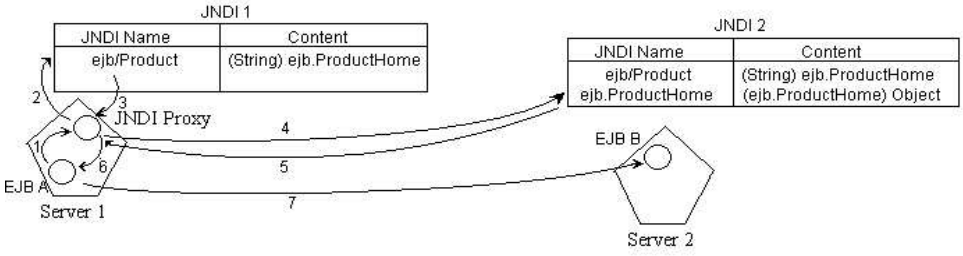



FIG. 6 – Partial EJB Migration

## 4.3 Resources Migration

In this mode of migration, we study the possibility of migrating the resources of an application, while taking as an example the connection to a data base. The concerned EJBs are mainly EJBs of the Entity type. It is however necessary to specify the behavior of an application server with this family of EJB.

When using an Entity EJB of the type Container Managed Persistence, it is the application server who handles the communication with the database. The communication is handled according to two elements declared on a configuration file specific to the application server : ”ConnexionPool” and ”DataSource” (figure 7).

The connection pool (JDBCConnectionPool tag) represents the connection to the data base. It is known only by the application server which is connected to the data base under a pseudo user (bob/secret). The connectionPool specifies certain parameters as the number of initial connections (3) and the number of maximum connections (5). The connection pool is accessible by the DataSource by the intermediary of a logical name (DIRECTORY). The DataSource (section JDBCDataSource) identifies a data source that is accessible to the EJB Entities. It is identified by a JNDI name which is accessible by the EJB (exemples.dataSource.Employee) and is associated with a connection pool (DIRECTORY).

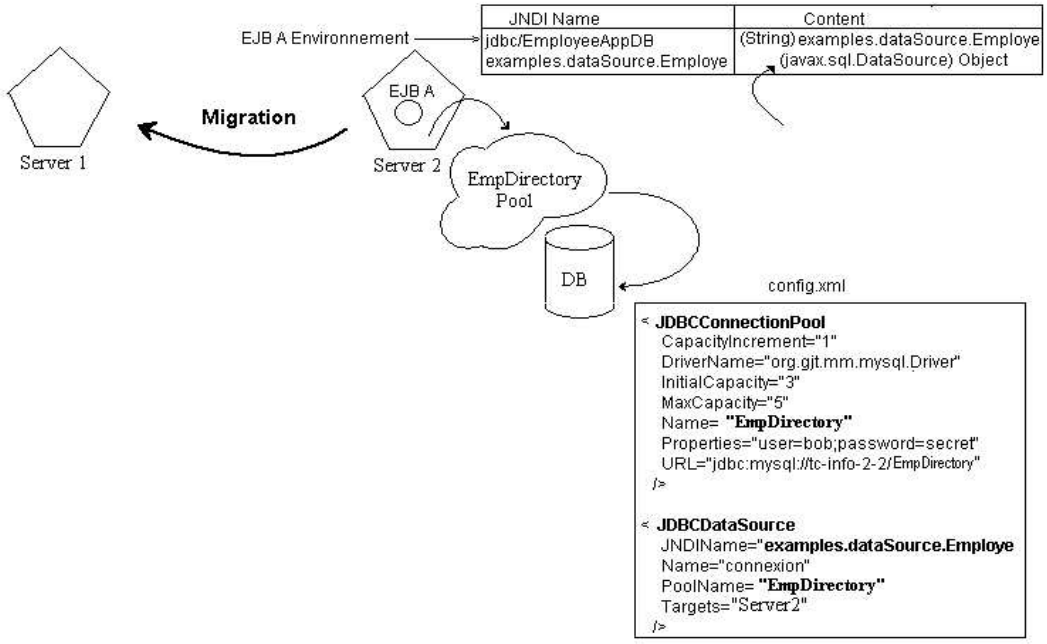



FIG. 7 – EJB/DB Relation Description

During the migration of an Entity EJB, it is necessary that the EJB continue to work with its data table. Here we also considered several modes for migrating the resources

- Dynamic creation of the base : It consists of recreating dynamically the concerned table on the application server where the EJB is going to migrate. In this case the EJB runs properly but does not have any initial data. This is possible for services that produce local data (a service of local telephone number for example). Some application servers as JBoss recreate dynamically at the time of the deployment the required tables

- Dynamic Creation of the connection pool : In this case the application server receiving the migrated EJB is responsible of the dynamic creation of a connection pool on the original database. This approach poses a number of technical problems on the Weblogic platform which does not allow the dynamic creation of connection pools. Figure 8 illustrates this mechanism.

- Reuse of the remote pool : This last solution consists of reusing the original remote pool provided by the application server. Connection to the remote pool is made at the migration time.

It is important to state that we did not study the possibilities of migrating data, because this is out of our initial study purposes and it raises many problems concerning some technologies of the distributed databases (synchronization, redundancy etc...)

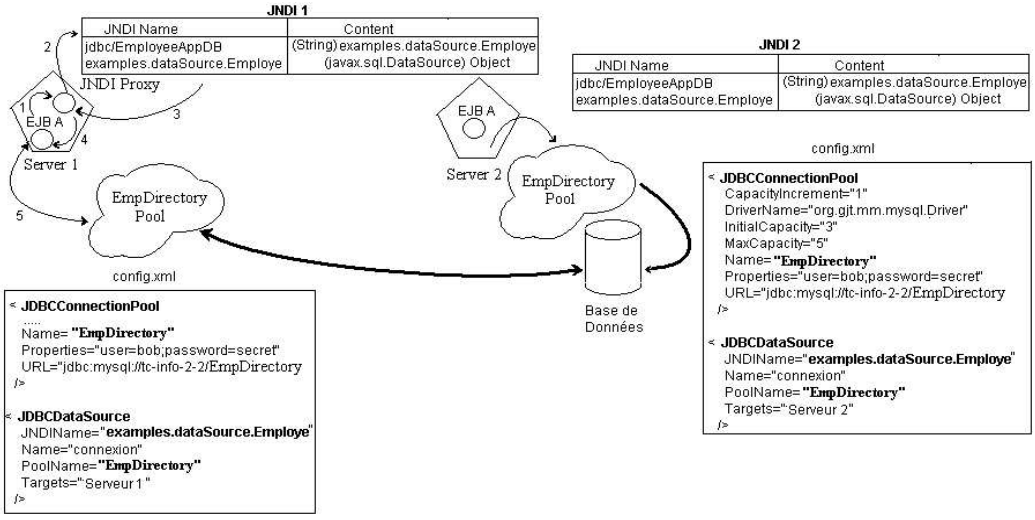



FIG. 8 – Dynamic Creation of a ConnexionPool

# 5 Conclusions

In this article we presented an architecture of migrating components between application servers compliant with the J2EE specification. Implementing our approach on a commercial platform enabled us to validate the concepts related to the mobile code.

We proposed an extension of the naming and directory service so that it can find components deployed on another remote server, migrate them, and deploy them locally. We considered that the migrated components are intensively used and that the cost of the transfer and the redeployment is very small compared to the number of following requests.

This architecture is however not limited to the naming problems. It is also adapted to the problems of the automatic deployment of component on huge sites. Indeed according to the user's needs, the components are automatically searched for and installed.

This architecture can also be a solution for the load balancing of the applications [21]. By installing a more powerful application server beside the original server, it would be possible to dynamically migrate the applications on the new server.

# Références

[1] C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol. In *EUNICE 2000 Proceedings, Sixth EUNICE Open European Summer School*, Twente, Netherlands, September 2000.

[2] S. Bouchenak. Making java applications mobile or persistent. In *6th USENIX Conference on Object-Oriented Technologies and Systems*, San Antonio (Texas - USA), 29th January - 2nd February 2001.

[3] E. Bruneton. Indirection free referencing for mobile components. In *Third European Research Seminar on Advances in Distributed Systems (ERSADS'99)*, Madeira Island, Portugal, 23-28 Avril 1999.

[4] Bea Corporation. http ://www.bea.com.

[5] IBM Corporation. http ://www.ibm.com.

[6] Microsoft Corporation. http ://www.microsoft.com.

[7] Oracle Corporation. http ://www.oracle.com.

[8] G.H. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer, vol 27, no 4, pp. 38-47*, 1994.

[9] A. Fuggeta, G.P. Picco, and G. Vigna. Understanding code mobility. *IEEE Trans. Of Software Ing., No 5, pp. 342-361*, May 1998.

[10] The Object Management Group. http ://www.omg.org.

[11] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource discovery in distributed networks. In *18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing, Atlanta*, May 1999.

[12] O. Holder, I. BenShaul, and H. Gazit. Dynamic layout of distributed application in fargo. In *Proc. ICSE'99, Los Angeles*, May 1999.

[13] Sun Microsystem J2EE. Java 2 platform enterprise edidtion specification, v1.3. http ://java.sun.com/j2ee, July 2001.

[14] JBoss. http ://www.jboss.org.

[15] JOnAS. http ://www.objectweb.org/jonas.

[16] R. Litiu and A. Prakash. Dacia : A mobile component framework for building adaptive distributed applications. In *Principles of Distributed Computing (PODC) 2000 Middleware symposium*, Portland, 2000.

[17] R.E. McGrath. Discovery and its discontents : Discovery protocols for ubiquitous computing. Technical report, Computer Science department, The university of Illinois, US, 2000.

[18] Sun Microsystems. Enterprise java beans specification, v2. http ://java.sun.com/products/ejb/, August 2001.

[19] E. Roman and R. Oberg. J2ee vs. com+ and windows dna - white paper. Technical report, http ://www.theserverside.com, 1999.

[20] J.D. Solomon. *Mobile IP the Internet Unplugged*. Prentice Hall, N.J., 1998.

[21] C. Steketee. Process migration and load balancing in amoeba. *Australian Computer Science Communications 21(1), 324-335*, 1999.

[22] M.J. Tucker. A snapshot of application servers in manufacturing. Technical report, ebizQ.net - http ://e-serv.ebizq.net/aps/tucker_2.html, 2000.

[23] T.Wals, P.A. Nixon, and S.A. Dobson. A managed architecture for mobile distributed applications. Technical report, Computer Science Department, The University of Dublin, Ireland, http ://www.cs.tcd.ie/publications/, 1999.

Unit´e de recherche INRIA Lorraine, Technop^ole de Nancy-Brabois, Campus scientifi que,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rh^one-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399