Université d'Évry-Val d'Essonne
Institut National des Télécommunications

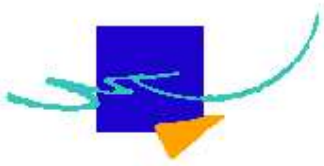
Rapport de Stage
DEA d'Informatique

SERVICES DE RECHERCHE INTELLIGENTS ET DÉPLOIEMENT DYNAMIQUE D'APPLICATIONS MULTI-COMPOSANTS

Dhouha AYED

Responsable de DEA : Jean-Marc DELOSME
Responsable de stage : Chantal TACONET

Juillet 2001



Ce stage de DEA a été réalisé au sein du laboratoire **Systèmes Répartis** du département **Informatique** de l'**Institut National des Télécommunications**

Remerciements

Ce stage a été réalisé au sein de l'équipe "systèmes répartis" de l'Institut National des Télécommunications d'Evry.
Je remercie, le chef du département informatique, Monsieur Guy Bernard, pour m'avoir accueilli dans son équipe.
Je remercie vivement Madame Chantal Taconet pour m'avoir proposé ce sujet et m'avoir encadré pendant toute la période de stage.
Je tiens à remercier toute léquipe de systèmes répartis notamment Monsieur Denis Conan, Monsieur Christian Bac, Monsieur Daniel Millot et Madame Sophie Chabridon pour leurs conseils et leurs aides.
Je tiens à exprimer ma profonde gratitude au thésards : Erik Putrycz, Olivier Villin, Ronaldo Ramos et Victor Budau pour leur aide précieuse et leur soutien.
Je remercie aussi ma collègue, stagaire de DEA, Lydialle Debassen.
Je finis en remerciant mes parents et ma famille pour leur soutien et tous mes amis pour leurs encouragements.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Les systèmes d'information modernes sont de plus en plus installés sur des réseaux à grande échelle. La mise en place de ces systèmes sur un réseau étendu se base sur l'utilisation d'applications distribuées complexes construites à partir de composants logiciels autonomes et coopératifs pouvant être placées sur de multiples serveurs distans.
Par ailleurs, le progrès de l'informatique mobile et des services réseau (réseaux sans fil, internet, ...) ainsi que l'évolution de la technologie des terminaux (PC portables, PDAs, téléphones mobiles, ...) permettent aux utilisateurs d'accéder aux différents services qu'ils demandent à partir d'endroits différents et à des moments variables : un même utilisateur, peut se connecter à une même application, le même jour, de son travail en utilisant un PC, de sa voiture à partir d'un PDA ou de chez lui à travers un ordinateur portable.

1.1 Problématique

La mobilité et la variabilité d'environnements de connexions des utilisateurs, implique une variabilité de l'environnement d'exécution des applications selon la localisation des utilisateurs, des débits des moyens de communication, de capacité de terminaux utilisés notamment de processeur, mémoire, système et langages supportés, ...
Cette variabilité de l'environnement d'exécution nécessite l'adaptation de chaque application à son contexte d'utilisation (utiliser une interface graphique s'adaptant à la taille de l'écran, ne pas placer tous les composants de l'application sur le terminal utilisateur s'il n'est pas assez puissant, ...).
D'autre part, l'installation manuelle des différents services sur chaque terminal à partir duquel l'utilisateur veut se connecter est une tâche rude. Il faut donc que ce genre d'installation soit automatique. L'installation automatique des différents composants d'une application s'appelle un déploiement.

1.2 Présentation du sujet

Notre objectif est de proposer une plate forme de déploiement pour que les applications distribuées, construites à partir de composants logiciels autonomes et coopératifs puissent etre utilisées dans un contexte mobile de manière plus simple, plus extensible et plus transparente.
Une ébauche de solution aux problèmes citées au dessus a été proposée, dans le projet Cesure en l'an 2000 fait par Gemplus en association avec l'INT, l'INRIA et le LIFL. Seulement, cette solution se limitait à une application bancaire et ne présentait pas un vrai système de déploiement.
Il s'agit donc de proposer une infrastructure de déploiement pour les applications multi-composants qui soit générale et qui permet de chercher d'une manière **dynamique les différents composants** d'une application ainsi que les différents serveurs sur lesquels s'exécuteront ces composants tout en assurant **une adaptation au contexte utilisateur**.

Dans le premier chapitre, nous ferons une étude comparative des services de recherche intelligents. Ces outils, permettent une recherche de services sur le réseau à partir de ses différentes propriétés. Cette spécificité leur permet d'être un élément esssentiel dans notre infrastructure, qui permettra de déterminer les différents composants d'une application en s'adaptant au contexte d'utilisation. Le deuxième chapitre présente une étude des applications multi-composants et du déploiement à travers le modèle CCM (Corba Coponent Model). Dans le troisième chapitre, nous décrivons l'infrastructure de déploiement que nous proposons tout en définissant une sémantique du déploiement en cas d'échec. Enfin, le quatrième chapitre présente un modèle UML pour le déploiement et une description IDL (Interface Definition Language) de l'interface du serveur de déploiement.

Chapitre 2

Etude comparative des services de recherche sur propriétés

Il existe plusieurs manières de rechercher un objet. La recherche d'un objet à partir d'un nom symbolique (exemple : URL) peut être suffisante lorsque le client connaît exactement ce qu'il cherche. Ce genre de recherche peut être comparé à la recherche dans des pages blanches où il faut connaître le nom de la personne recherchée pour pouvoir trouver son numéro de téléphone.

Souvent, les clients ont besoin d'un mécanisme plus dynamique pour localiser des objets. Par exemple, un client peut avoir juste une idée du type d'objet dont il a besoin mais pas toutes les informations qu'il faut pour faire un choix précis. Les services de recherche sur propriétés offrent des fonctionnalités permettant à des clients de localiser des objets à partir de leurs propriétés. Ce genre de service ne stocke pas les noms des objets mais une description détaillée des objets. Les clients peuvent alors bénéficier d'une recherche dynamique des services basée sur des requêtes à travers les descriptions du service. Ce type de recherche peut être comparé à des pages jaunes. Au lieu de lister les services par leur nom, les pages jaunes catégorisent les entrées par sujet et décrivent chaque entrée avec détails.
La recherche sur propriétés joue un rôle essentiel dans le déploiment des applications multi-composants puisqu'elle permet la recherche des différents composants d'une application de manière qu'elle s'adapte le mieux à l'environnement d'exécution.
Dans ce chapitre, nous allons étudier quatre services de recherche sur propriétés : le trader de Corba, le service de recherche de jini, le service de recherche de Salutation et le service de recherche de UDDI. Nous essayerons de donner assez de détails sur chacun d'eux afin que le lecteur puisse comprendre leur principe de fonctionnement ainsi que leurs points forts et leurs points faibles. Le chapitre se terminera par une comparaison des quatres services de recherche étudiés.

2.1 Concepts de courtage et terminologie

Le service de recherche sur propriétés appelé aussi service de courtage, service de “trading”, service de “vente” ou service de “médiation” est un service permettant de découvrir dynamiquement les services offerts. Il facilite donc l’annonce et la découverte des services. Quand un serveur désire annoncer (ou publier) son service, il enregistre son offre de service auprès du service de courtage : c’est l’opération d’exportation. Le programme qui publie un service s’appelle exportateur. Lorsqu’un client requiert un service, il émet une requête de recherche au service de courtage en spécifiant quelques caractéristiques de ce service : c’est l’opération d’importation et le programme qui l’effectue s’appelle importateur.
Le service de recherche enregistre les publications de services. Ces publications s’appellent des offres de service.

2.2 Trader CORBA

Le besoin d’utiliser des pages jaunes dans l’environnement Corba a donné naissance à un service de courtage propre à CORBA. Le principe de fonctionnement de ce service peut se résumer comme suit : Une application serveur va exporter vers le trader la description et la référence d’un objet qui lui est propre. Les applications clientes désirant utiliser cet objet interrogeront le trader en fournissant des critères de sélection. Une fois que l’application aura la référence sur l’objet demandé comme résultat, elle pourra l’utiliser à travers le bus CORBA [GGM97] (voir figure 2.1).

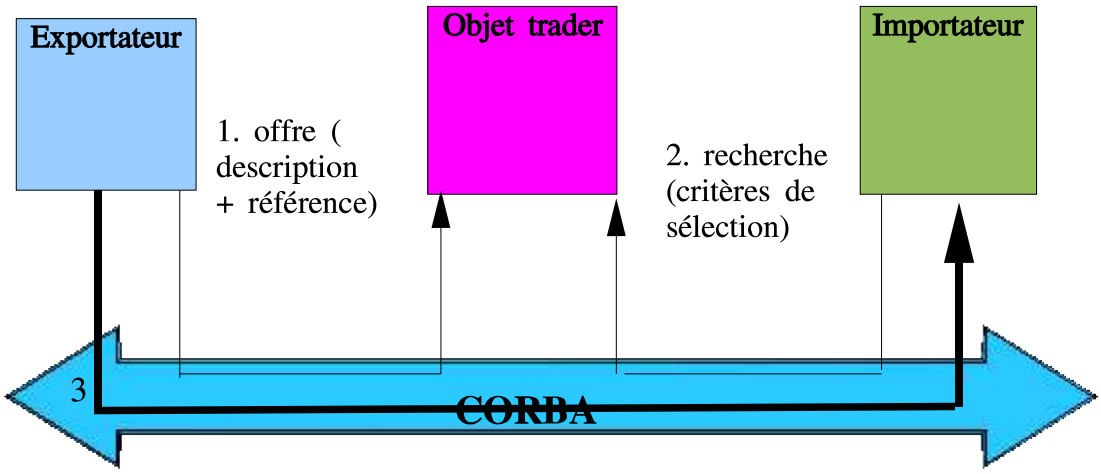


FIG. 2.1 – Fonctionnement du trader Corba

2.2.1 Types de services

Les services offerts sont décrits sur un réseau d'une manière abstraite au niveau du trader à l'aide d'une structure appelée type de service. Les types de services peuvent être comparés aux catégories des pages jaunes. La recherche d'un hôtel dans les pages jaunes, par exemple, suppose qu'il y ait un certain nombre d'informations communes à tous les hôtels comme un nom, un numéro de téléphone, une adresse, un prix pour une nuit dans une chambre, le type de service d'une offre représente les types d'informations qu'un importateur peut utiliser pour rechercher un service au niveau du trader ou d'une autre manière le type de service détermine les propriétés utilisées pour décrire une offre de service. Les types de service sont organisés d'une manière hiérarchique selon un arbre d'héritage.

Le ServiceTypeRepository joue le rôle de dépositaire de stockage des types d'offres de services et assure la vérification du typage lors de l'exécution des requêtes d'exportation et de recherche. Dans ce dépositaire, un type d'offres de services est caractérisé par quatre éléments[OMG00].

- Un nom unique identifiant le type de service.
- Des super types hérités permettant de définir une classification des types par héritage multiple.
- Une interface OMG IDL à laquelle les services exportés doivent se conformer.
- 0 ou plusieurs types de propriétés définissant l'aspect comportemental du service.

Le type de propriété est un triplet <nom, type, mode>. Le mode indique si la propriété est optionnelle ou obligatoire lors de l'enregistrement d'une offre de service de ce type et si elle peut changer de valeur après la publication de l'offre. Les modes de propriétés sont les suivants :

- normal : la valeur de la propriété est optionnelle, elle peut être modifiée ou supprimée.
- readonly : idem à normal mais si la valeur de la propriété est fournie à l'exportation alors elle ne peut plus être modifiée.
- mandatory : une valeur doit obligatoirement être affectée à la propriété lors de l'exportation.
- mandatory readonly : idem à mandatory mais non modifiable.

L'interface ServiceTypeRepository [OMG00] offre plusieurs opération pour gérer les différents types de service : l'opération add_type permet d'ajouter un type de service. La modification et la suppression d'un type se font respectivement par les opérations modify_type et remove_type (voir détails dans la section 2.2.6).

2.2.2 Exportation de services auprès du trader

Tout objet CORBA peut être exporté au niveau du service de recherche sur propriétés sous forme d'une offre de service. Une offre de service permet d'associer, pour un type de service

donné, la référence de l'objet CORBA avec les valeurs de propriétés qui le caractérisent (toutes les propriétés obligatoires et certaines optionnelles)[HV99].
Chaque offre de service a un identifiant unique renvoyé à l'exportateur par le trader lors d'une demande de publication. Il permet à l'exportateur de modifier ou retirer l'offre.
L'exportation de service se fait grâce à l'interface Register (voir détails dans la section 2.2.6).

2.2.3 Importation de services et langage de contraintes

Pour importer un service quelconque sur le réseau, un importateur envoie une requête vers le trader contenant :

1. Le nom du type de service. Exemple : hôtel.
2. Des contraintes sur le service recherché.Dans notre exemple, ces contraintes détermineront quels hôtels particuliers l'utilisateur cherche.Il peut par exemple chercher un hôtel dont le nombre d'étoiles est supérieur à deux et ayant une piscine.
3. Des préférences définissant un ordre dans lequel seront retournés les résultats.
4. Des politiques contrôlant des aspects non fonctionnels de recherche tels que le nombre d'offres retournées et s'il s'agit de retourner la description complète d'un service ou seulement la référence sur l'objet.

L'importation de service se fait grâce à l'interface Lookup détaillée dans la section 2.2.6.
La sélection d'offres de service se fait par une réduction de l'ensemble d'offres offerts par les différents traders par des politiques de recherches définies au niveau de chaque trader. Cet espace d'offres est alors filtré par le type de service demandé et une expression de contraintes. Les offres sélectionnées par le filtre sont ensuite ordonnées par une expression de préférences (voir figure 2.2).

Les contraintes et les préférences sont exprimées à l'aide d'un langage de contraintes appelé OCL (OMG Constraint Language). Elles sont construites à partir de noms de propriétés d'un type de service, des valeurs prises par ces propriétés et des opérateurs logiques, comparatifs, mathématiques et d'existence.

2.2.4 Fédération du service de recherche

La fédération des traders permet l'accès à un grand nombre d'offres de services sans avoir besoin d'enregistrer toutes les offres sur un seul trader. La fédération de recherche d'offres est transparente par rapport aux utilisateurs. Un trader fédéré est vu par un utilisateur comme un seul trader logique.

Les traders fédérés sont reliés entre eux à travers un graphe orienté. Chaque trader a des informations sur ce graphe. Les liens représentent des chemins de propagation des requêtes d'un trader source vers un trader destination. Chaque lien représente un arc dans le graphe de trading dont les sommets représentent des traders. Un lien décrit les connaissances qu'un trader a d'un autre service de courtage qu'il utilise et des informations sur le moment où il faut envoyer une

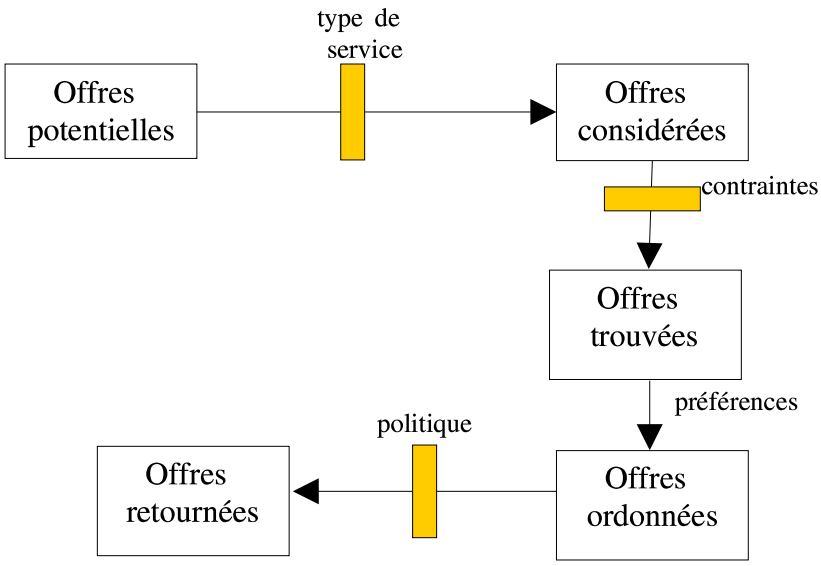


FIG. 2.2 – Filtrage des offres par le trader Corba

opération vers le trader cible[OMG00].

Les traders sont fédérés de manière qu'un trader agissant comme un client pour un autre trader. Supposons par exemple qu'un client envoie une requête vers un trader A. Le trader A fait des recherches au niveau de sa base de données mais fait suivre aussi la requête vers son trader fédéré B. Le trader B retournera ses résultats vers A qui les émergera à ses propres résultats et retournera les résultats finaux vers l'utilisateur.

La topologie de la fédération des traders peut être complexe et peut parfois contenir des boucles. La spécification de l'OMG [OMG00] demande aux traders fédérés d'implémenter la détection de boucles de manière à ce que les requêtes partant d'un client ne soient pas transmises d'un trader vers un autre indéfiniment.

2.2.5 Propriétés dynamiques

Généralement, une propriété d'une offre de service a une valeur fixe enregistrée par l'exportateur. Cette valeur ne change pas sauf si quelqu'un la modifie explicitement. Ce genre de propriétés statiques n'est pas adéquat pour certains types de services. Par exemple pour la recherche d'actions dans la bourse, le prix d'une action subit des fluctuations rapides. L'utilisation de propriétés statiques demande la mise à jour fréquente de la propriété prix de l'action.

Pour accommoder ce genre de situations, les traders offrent la possibilité d'utiliser des propriétés dynamiques. Les valeurs de ces propriétés ne résident pas dans le trader, elles seront évaluées à la demande lors d'une recherche à travers un objet d'évaluation de propriétés dynamiques désigné par l'exportateur.

La structure d'une propriété dynamique contient l'interface de l'évaluateur dynamique de propriétés, le type de données des propriétés dynamiques retournées et des informations supplémentaires d'implémentation[HV99]. Quand la valeur d'une propriété est demandée, le trader invoque la méthode evalDP à partir de l'interface Dynamic_PropEval (voir section 2.2.6).

2.2.6 API trader

Le trader CORBA offre onze interfaces faisant partie du module *Costrading*. Ces interfaces permettent l'importation et l'exportation de services, l'exportation de proxies d'offres de services, la modification de la structure de la fédération et la configuration du trader.

2.2.6.1 Les interfaces d'utilisation

La structure IDL du module *Costrading* est donnée dans la spécification de l'OMG [OMG00]. Cette structure définit quatre interfaces abstraites de base ayant comme tâche de regrouper les fonctionnalités reliées utilisées par d'autres interfaces.
Ces interfaces abstraites sont : *ImportAttributes*, *TraderComponents*, *SupportAttributes* et *LinkAttributes*. Le graphe d'héritage entre interfaces IDL est représenté sur la figure 2.3 [HV99].

Les interfaces les plus importantes sont les suivantes :

L'interface lookup : cette interface permet l'importation de services. Elle n'offre que l'opération *query* permettant aux importateurs de rechercher des objets. L'utilisation d'une telle opération nécessite six paramètres.

1. Le nom du type de service.
2. Des contraintes qui doivent être satisfaites par les propriétés des offres de services recherchées.
3. Des préférences définissant un ordre pour les services retournés.
4. Les stratégies à appliquer le long de la recherche.
5. La liste des noms de propriétés (exemple : aucune, toutes, certaines) dont les valeurs devront être retournées à l'application cliente.
6. Le nombre maximal de réponses désiré.

Si le nombre de réponses dépasse le nombre maximal spécifié, un itérateur sera retourné afin que le client puisse parcourir le reste de la liste.

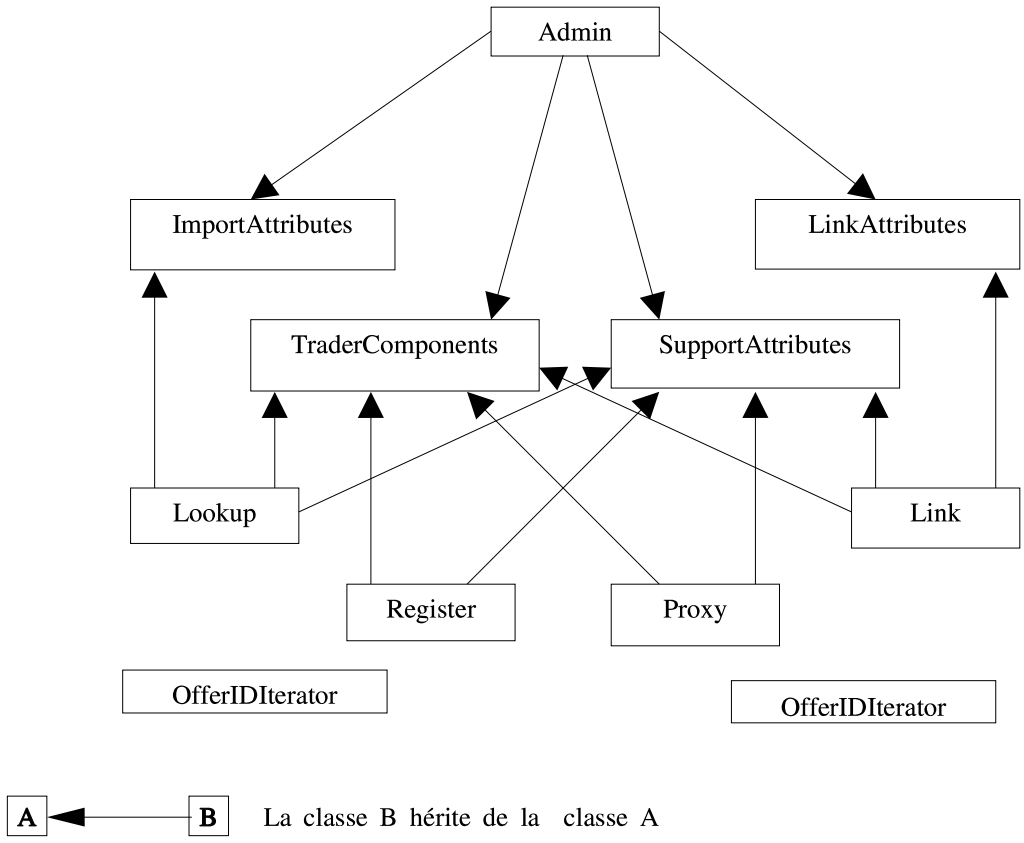


FIG. 2.3 – Graphe d'héritage entre interfaces IDL du module Costrading

L'interface register : cette interface permet aux différentes applications distribuées de publier leurs services et de mettre à jour le contenu du dépositaire du service de courtage. La publication d'un service (enregistrement d'un service au niveau du service de courtage) se fait à l'aide de l'opération *export* nécessitant les paramètres suivants :

1. Le nom du type de service,
2. Une référence à l'interface qui offre le service,
3. Des valeurs pour les propriétés du service.

Il est possible de modifier une description d'un service avec l'opération *modify* ou supprimer des services du dépositaire avec l'opération *withdraw*.

L'interface OfferIterator : cette interface permet le parcours des résultats de recherche qui n'ont pas pu être stockés dans la séquence IDL.

L'interface Link : cette interface permet de constituer une fédération de services de recherche. La connexion entre les traders se fait grâce à l'opération *add_link*. La modification et le retrait de liens se font respectivement à l'aide des opérations *modify_link* et *remove_link*.

L'interface DynamicPropEval : cette interface permet de calculer, à l'exécution, la valeur courante d'une propriété dynamique au travers de l'opération EvalDP.

L'interface proxy : cette interface permet de déterminer la référence réelle d'un objet offrant un service. Cette interface est utilisée, par exemple, pour trouver un objet (répondant aux critères demandés) dans un ensemble d'objets potentiels (par exemple s'il y a équilibrage de charge)[GGM97].

2.2.6.2 Les interfaces d'administration

Les interfaces d'administration sont au nombre de deux et permettent de contrôler le fonctionnement du service de courtage et manipuler le référentiel des types de service.

L'interface ServiceTypeRepository : permet de gérer le dépositaire des types de services. L'opération *add_type* permet d'ajouter un type.
La modification et la suppression d'un type se fait respectivement par les opérations *modify_type* et *remove_type*.

L'interface Admin [OMG00] : cette interface permet de faire l'administration proprement dite : elle permet de définir le nombre maximum de réponses retournées lors d'une recherche, des politiques de recherche tel que le nombre de traders pouvant être parcourus à partir de celui-ci, etc. Lorsqu'une requête est lancée vers un trader, ce sont les paramètres par défaut qui seront utilisés. Tous ces paramètres peuvent être surchargés dans une requête.

2.2.7 Conclusion

Enfin, comme conclusion pour cette section, Le service de recherche de Corba, présente une multitude de points forts. Il se distigue particulièrement par sa fédération de recherche, son utiliation d'un language de contraintes et de la notion de type de service. Il permet aussi aux

utilisateurs de spécifiers des politiques de recherche et de définir des propriétés dynamiques. Seulement, ce service n'est pas simple à utiliser et l'utilisation de son API nécessite l'écriture de beaucoup de lignes de code.

2.3 Service de recherche de Jini

Jini est un système distribué basé sur l'idée de la fédération de groupes et des ressources. Le but d'un tel système est d'offrir un outil flexible et facilement administré qui permet aux applications clientes de retrouver les différentes ressources du réseau d'une manière dynamique. Ces ressources peuvent être matérielles, logicielles ou une combinaison des deux.
Jini étend l'environnement d'une application java d'une seule machine virtuelle vers un réseau de machines locales. Ce réseau peut même être un réseau domestique connectant des téléviseurs, des réfrigérateurs, des chaines stéréo, ...

Jini offre des mécanismes pour la construction de services, leur recherche, leur communication et leur utilisation dans un système distribué.

Dans le cadre de ce stage, nous ne nous sommes intéressés qu'à la recherche des services.
Le service de recherche Jini est basé sur un trio de protocoles appelés : découverte, jointe et recherche. La paire de protocoles découverte et jointe se produit au moment où un service vient d'être implanté dans le réseau.

- La découverte a lieu lorsqu'un service est en quête d'un service de recherche auprès duquel s'enregistrer.
- La jointe a lieu au moment où le service de recherche est localisé et que le service veut s'y enregistrer.
- La recherche a lieu lorsqu'une application cliente a besoin de localiser et invoquer un service décrit par son type d'interface écrite en Java et un ensemble d'attributs.

La paire découverte/jointe est le processus permettant l'ajout d'un service au système Jini.

2.3.1 Vue générale sur le modèle du service de recherche de Jini

Le service de recherche maintient une collection plate d'articles décrivant des services. Chaque article représente une instance d'un service disponible au niveau de Jini [JIN01a]. Un article contient un stub RMI (si le service est implémenté comme un service distant) ou un autre objet que les programmes utilisent pour accéder au service (si le service utilise un proxy local) ainsi qu'une collection extensible d'attributs décrivant le service.
Dès qu'un nouveau service est créé, le service s'enregistre au niveau du service de recherche de Jini avec un ensemble initial d'attributs.

Les programmes ayant besoin d'un type de service particulier peuvent utiliser le service de recherche pour trouver une instance de ce service. une correspondance peut être faite en se basant sur des types de données spécifiques implémentés par le service ainsi que les attributs

spécifiques attachés au service. Par exemple, un programme qui a besoin d'utiliser des transactions peut chercher un service supportant le type net.jini.transaction.server.TransactionManager. Une grande variété de vues hiérarchiques peut être imposée à la collection des articles de services plate en faisant des agrégats d'attributs selon les types de services.
Le service de recherche offre un ensemble de méthodes permettant une exploration de la collection. Une variété d'interfaces utilisateur peut être construite en utilisant ces méthodes permettant aux utilisateurs et administrateurs la navigation. Une fois qu'un service est trouvé, l'utilisateur peut interagir avec le service en téléchargeant une applet " interface utilisateur " attachée comme un autre attribut à l'article de service.

La figure 2.4 illustre le fonctionnement de la recherche de services dans le système jini. Les différentes étapes de connexion au service de recherche, d'importation et d'exportation de services présentes sur la figure, vont être expliquées en détail dans ce qui suit.

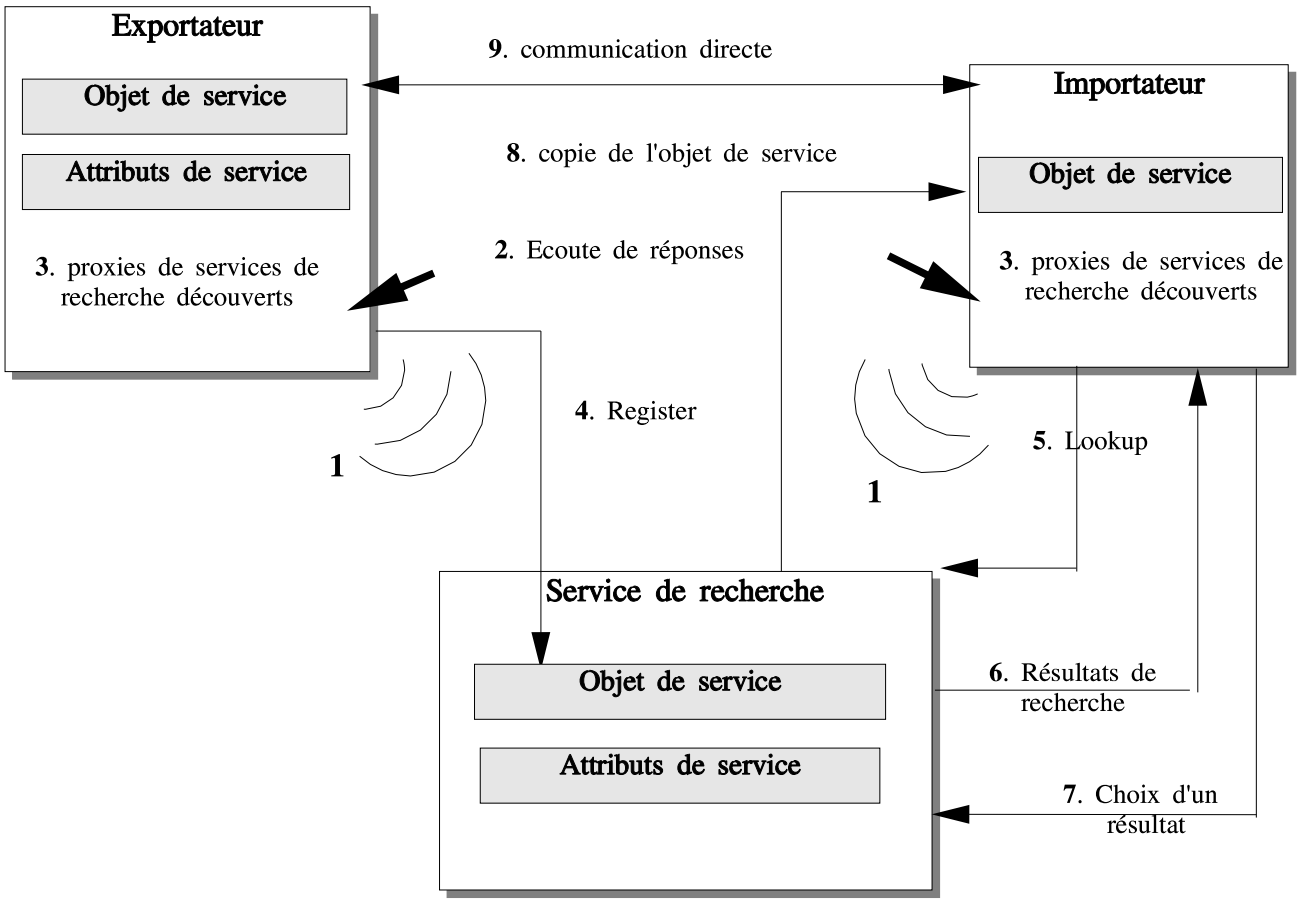


FIG. 2.4 – Différentes étapes pour la connexion, l'exportation et l'importation de services au niveau du service de recherche Jini.

2.3.2 Connexion au service de recherche

Un exportateur ou un importateur dans jini commence par chercher un service de recherche auprès duquel il exportera ou cherchera à importer des services. Ceci est réalisé à travers un multicast de requêtes (étape 1 de la figure 2.4) et une écoute de réponses à travers le réseau local (étape 2 de la figure 2.4) pour s'identifier auprès d'un service de recherche quelconque. Tous les services de recherche de proximité fournissent une réponse à ces entités appelantes sous forme d'objet de recherche (étape 3 de la figure 2.4). Dans le cas où il y a plusieurs services de recherche dans le réseau, ces objets seront mis dans un tableau et seront placés au niveau de l'appelant (exportateur ou un importateur) pour jouer le rôle de proxy pour le service de recherche. Si on veut que l'invocation des services de recherche se fasse à distance, au lieu de l'objet proxy, un stub RMI sera placé au niveau de l'exportateur ou l'importateur.

Les objets proxy de recherche représentent une instance de l'interface *ServiceRegistrar* suivante :

```
public interface ServiceRegistrar {
    ServiceRegistration register(ServiceItem item,
                                 long leaseDuration)
        throws RemoteException;

    Object lookup(ServiceTemplate tmpl)
        throws RemoteException;

    ServiceMatches
        lookup(ServiceTemplate tmpl, int maxMatches)
        throws RemoteException;

    EventRegistration notify(ServiceTemplate tmpl,
                             int transitions,
                             RemoteEventListener listener,
                             MarshalledObject handback,
                             long leaseDuration)
        throws RemoteException;

    Class[] getEntryClasses(ServiceTemplate tmpl)
        throws RemoteException;

    Object[] getFieldValues(ServiceTemplate tmpl,
                            int setIndex,
                            String field)
        throws NoSuchFieldException, RemoteException;

    Class[] getServiceTypes(ServiceTemplate tmpl,
                            String prefix)
        throws RemoteException;

    ServiceID getServiceID();
    LookupLocator getLocator() throws RemoteException;

    String[] getGroups() throws RemoteException;
}
```

ServiceRegistrar définit l'interface du service de recherche. Cette interface n'est pas distante ; chaque implémentation du service de recherche, exporte des objets proxy, locaux au niveau client, qui implémentent l'interface ServiceRegistrar utilisant un protocole spécifique pour communiquer avec le serveur. Les méthodes offertes par cette interface permettent essentiellement d'enregistrer les services (*register*), les rechercher (*lookup*) et recevoir des événements de

notification (*notify*) lorsqu'un service est modifié.
ServiceRegistrar offre aussi des méthodes pour explorer les différents services selon trois axes majeurs : par type de service, par valeurs d'attributs et par classes d'attributs.

La méthode *register* est utilisée pour enregistrer un nouveau service. Un service enregistré, est manipulé à l'aide d'une instance de l'interface *ServiceRegistration* suivante :

```
public interface ServiceRegistration {
    ServiceID getServiceID();
    Lease getLease();
    void addAttributes(Entry[] attrSets)
        throws UnknownLeaseException, RemoteException;
    void modifyAttributes(Entry[] attrSetTemplates,
                          Entry[] attrSets)
        throws UnknownLeaseException, RemoteException;
    void setAttributes(Entry[] attrSets)
        throws UnknownLeaseException, RemoteException;
}
```

La méthode *getServiceID* retourne l'identifiant du service.
La méthode *addAttributes* rajoute l'ensemble des attributs spécifiés en paramètre au service enregistré.
La méthode *modifyAttributes* sert à modifier l'ensemble des attributs existants ; Chaque valeur d'attribut se trouvant dans le tableau *attrSetTemplates* sera remplacée par la valeur d'attribut ayant le même indice dans le tableau *attrSets*.
La méthode *setAttributes* efface tous les attributs d'un service et les remplace par l'ensemble d'attributs spécifiés.

2.3.3 Exportation de services auprès d'un service de recherche

Une fois qu'un exportateur a identifié le service de recherche de proximité, il peut s'y enregistrer en invoquant la méthode register du proxy de recherche en lui passant comme paramètres une copie de l'interface du service qu'il offre et un tableau d'attributs décrivant le service (étape 4 de la figure 2.4). Cette étape copie l'objet de service et les attributs au niveau du service de recherche.

2.3.4 Importation de services auprès d'un service de recherche

Une fois qu'un importateur a identifié et s'est connecté au service de recherche de proximité, il peut faire des recherche en invoquant la méthode lookup du proxy de recherche en lui passant comme paramètres le type (l'interface) du service recherché et un tableau de valeurs d'attributs (étape 5 de la figure 2.4).
Le service de recherche enverra un ensemble de résultats parmi lesquels l'importateur fait un choix (étapes 6 et 7 de la figure 2.4). Une fois le choix de l'utilisateur s'est fixé sur un service, le service de recherche copiera l'objet de service au niveau de l'importateur (étape 9 de la figure 2.4) ce qui lui permettera une communication directe avec le service demandé (étape 10 de la figure 2.4).

2.3.5 Structure des données enregistrés au niveau du service de recherche

Les services sont enregistrés au niveau du service de recherche sous forme d'une instance de la classe ServiceItem (article de service) suivante :

```
public class ServiceItem implements Serializable {
    public ServiceItem(ServiceID serviceID,
                       Object service,
                       Entry[] attributeSets) {...}
    public ServiceID serviceID;
    public Object service;
    public Entry[] attributeSets;
}
```

Chaque service est assigné à un identifiant universel unique (UUID).
En plus de son identifiant, un service est caractérisé par une interface et un ensemble d 'attributs. Les attributs d'un service sont représentés comme un tableau d'ensemble d'attributs. Un ensemble d'attribut est représenté comme une instance d'une classe Java où chaque attribut est un champ public de cette classe. La classe offre un typage fort des attributs.
Un article de service peut contenir plusieurs instances de classes différentes d'attributs ou de la même classe avec des valeurs d'attributs différentes. Par exemple, un article peut avoir plusieurs instances de la classe " Nom ", chacune donnant un nom au service dans un langage différent, une instance de la classe " localisation " et d'une classe " Propriétaire ". D'une manière plus concrète un ensemble d'attributs est implémenté avec une classe implémentant l'interface net.jini.core.entry.Entry[JIN01b]

2.3.6 Méthodes de recherche d'offres de service

Les articles de services dans le service de recherche sont appariés en utilisant des instances de la classe ServiceTemplate.

```
public class ServiceTemplate implements Serializable {
    public ServiceTemplate(ServiceID serviceID,
                           Class[] serviceTypes,
                           Entry[] attributeSetTemplates) {...}
    public ServiceID serviceID;
    public Class[] serviceTypes;
    public Entry[] attributeSetTemplates;
}
```

Un article de service (item) correspond à un patron de service (tmpl) si :
- item.serviceID est égal à tmpl.serviceID (ou si tmpl.serviceID est null), et
- item.service est une instance de chaque type dans tmpl.serviceTypes, et
- item.attributeSets contient au moins une entrée égale à son entrée homologue dans tmpl. AttributeSetTemplates.

Une entrée correspond à un calque d'entrée si la classe du calque est la même ou une super classe de la classe de l'entrée et chaque champ non vide dans le calque est égal au champ correspondant dans l'entrée.

2.3.7 Conclusion

Le plus grand avantage du service de recherche de Jini se situe dans le fait que la connexion à ce service se fait d'une manière dynamique : aucune application n'est reliée par défaut à un service de recherche. Toute application voulant enregistrer ou rechercher des services doit commencer par rechercher un service de recherche auquel elle peut se connecter. Seulement, le service de recherche Jini se limite à la recherche de services de proximité.

2.4 Service de recherche de Salutation

" Salutation " a été conçu pour résoudre les problèmes de découverte de services et leurs utilisation à travers un large ensemble d'applications et équipements dans un environnement étendu de connectivité et mobilité. L'architecture de " Salutation " est conçue de manière à être indépendante du processeur, du Système d'exploitation et des protocoles de communication. " Salutation" offre des méthodes standards pour les applications afin d'exporter les services qu'ils offrent, importer les services dont ils ont besoin et établir une session interopérable avec les applications offrant ces services [Sal01].

2.4.1 Architecture de Salutation

2.4.1.1 Salutation Manager

L'architecture de " Salutation " est construite autour d'une entité midleware appelée " Salutation Manager " (SLM) qui joue le rôle d'un courtier pour les différentes applications. Salutation Manager permet aux applications de découvrir et utiliser les services offerts par d'autres applications.

Comme le montre la figure 2.5, chaque entité sur le réseau doit être reliée au maximum à un Salutation Manager qui peut être placé localement sur l'entité elle même ou sur une entité distante (dans ce cas Salutation Manager est invoqué à distance à travers du RPC).

Salutation Manager communique avec d'autres Salutation Managers pour assurer son rôle de courtier en utilisant un protocole de communication qui lui est propre. Ce protocole utilise le RPC version 2 de Sun Microsystems.

Salutation Manager offre deux interfaces indépendantes des couches basses du réseau (*Cf.* figure 2.6) :

1. Une API appelée " Salutation Manager API " (SLM-API) permettant aux importateurs et exportateurs la publication et la recherche de services.
2. Une interface maintenant une séparation avec les couches basses du réseau appelée " Salutation Manager Transport Interface " (SLM-TI) utilisée par les entités dépendantes des couches réseau appelés Transport Managers (TM).

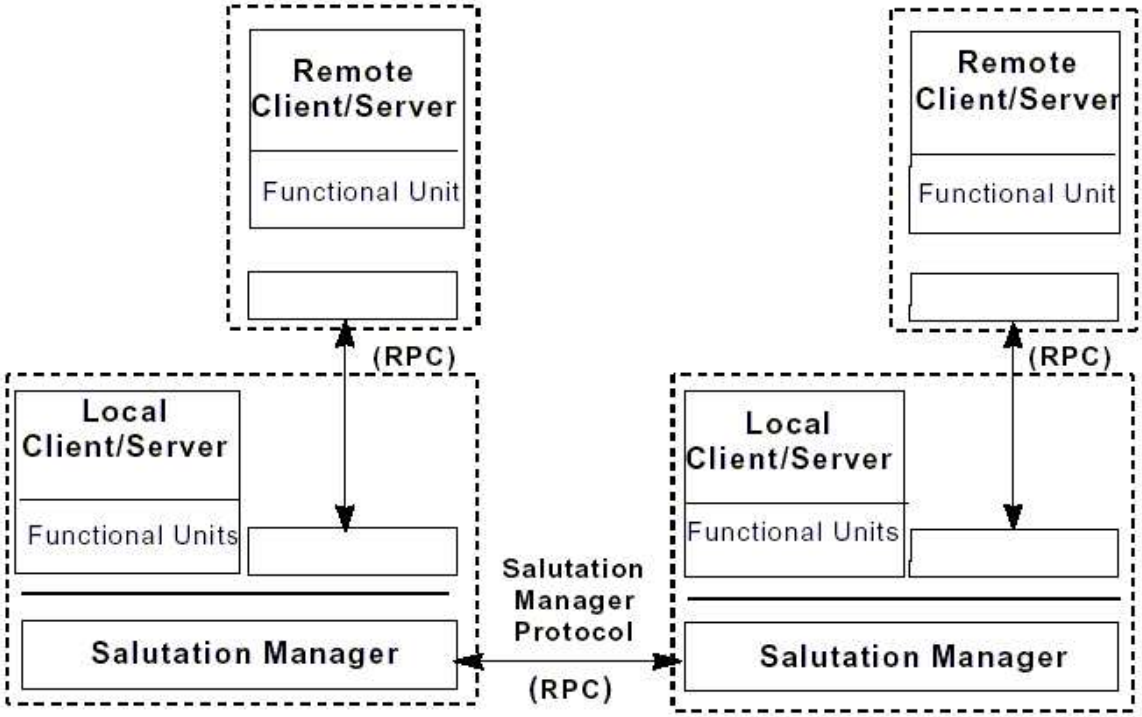


FIG. 2.5 – Architecture générale de salutation.

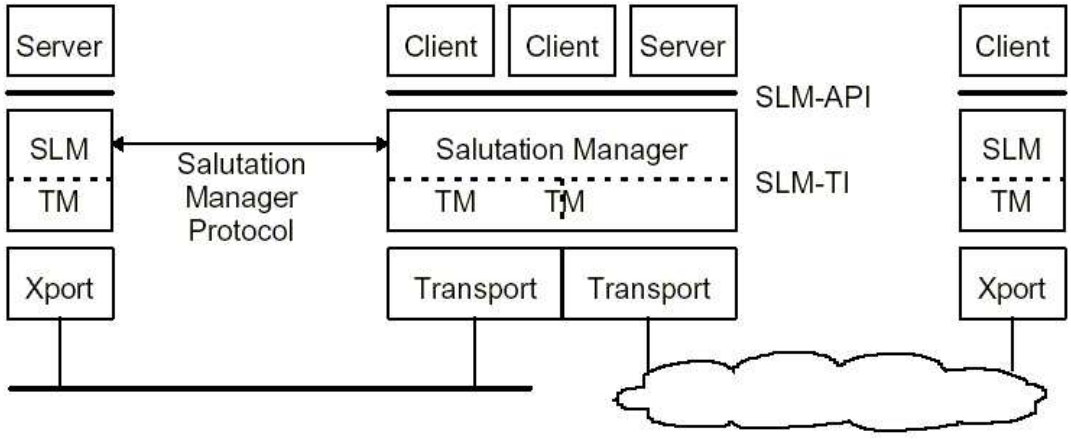


FIG. 2.6 – Architecture détaillée de salutation.

SLM-API et SLM-TI assurent tous les deux le rôle de courtage.

Chaque Salutation Manager a un identificateur universel unique (SLM-ID) qui l'identifie auprès des importateurs, exportateurs et Salutation Managers.

Un Salutation Manager travaille avec un ou plusieurs Transport Managers. Chaque Transport Manager découvre d'autres Salutation Managers distants connectés aux réseaux qu'il supporte.

Le Transport Manager récupère l'identificateur unique de chaque Salutation Manager distant découvert et l'enregistre avec l'identificateur du Salutation Manager local et maintient l'association entre ces identificateurs et leurs adresses réseaux.

Les Transport Managers peuvent découvrir les Salutation Manager de différentes manières :

- Les transport Managers peuvent avoir des tables statiques contenant les adresses réseau des Salutation Manager distants.
- Le Transport Manager fait un broadcast de requêtes pour trouver d'autres Salutation Manager distants.

2.4.1.2 Services de courtage

Pour assurer sa fonction de courtier, Salutation Manager offre trois services de courtage :

Base de registres : Le " Salutation Manager " comporte une base de registres pour enregistrer des informations sur les services connectés au Salutation Manager localement ou à distance. Ce référentiel peut aussi contenir des informations sur les services enregistrés dans d'autres Salutation Managers.

La Découverte de services : La découverte de services par un Salutation Manager se fait en découvrant d'autres Salutation Managers distants et en déterminant les services qui y sont enregistrés. Une comparaison est faite entre les types de services requis par le Salutation Manager local et les types de services disponibles au niveau du Salutation Manager distant. La transmission des types de services requis à partir du Salutation Manager local vers le Salutation Manager distant et la transmission de réponses du Salutation Manager distant vers le Salutation Manager local se fait à travers du RPC.

Selon les demandes d'un importateur, Salutation Manager peut déterminer :

- Les caractéristiques de tous les services enregistrés dans les Salutation Managers distants.
- Les caractéristiques d'un service spécifique enregistré dans les Salutation Manager distants.
- La présence d'un service vérifiant un ensemble de caractéristiques spécifiques dans les Salutation Managers distants.

La Disponibilité de service : Salutation Manager peut vérifier périodiquement la disponibilité d'un service avec le Salutation Manager auquel il est lié en échangeant des messages RPC.

2.4.2 Unités fonctionnelles

Une unité fonctionnelle est une représentation abstraite des fonctions offertes par une application . Par exemple, une fonction qui imprime des documents peut définir une unité fonctionnelle qu'on note [print]. A chaque unité fonctionnelle est associé un ensemble d'attributs. Par exemple, l'unité fonctionnelle [print] a comme attributs la taille du papier pouvant être utilisé, densité des pixels, couleurs,...
L'architecture de " Salutation " définit une syntaxe et une sémantique pour décrire les attributs de chaque unité fonctionnelle appelée " enregistrement de description d'unité fonctionnelle " décrits dans la section suivante.

2.4.3 Structure des données au niveau de salutation

Une entité qui a plusieurs fonctions sur le réseau, contient une multitude d'unités fonctionnelles. L'ensemble de ses unités fonctionnelles forme un service qui est décrit par un enregistrement de description de services. Un enregistrement de description de services est donc une collection d'enregistrements de description d'unités fonctionnelles. Les enregistrements de description d'unités fonctionnelles sont eux même composés d'enregistrements contenant les valeurs d'attributs décrivant les unités fonctionnelles.
La figure 2.7 décrit la structure des différents enregistrements.

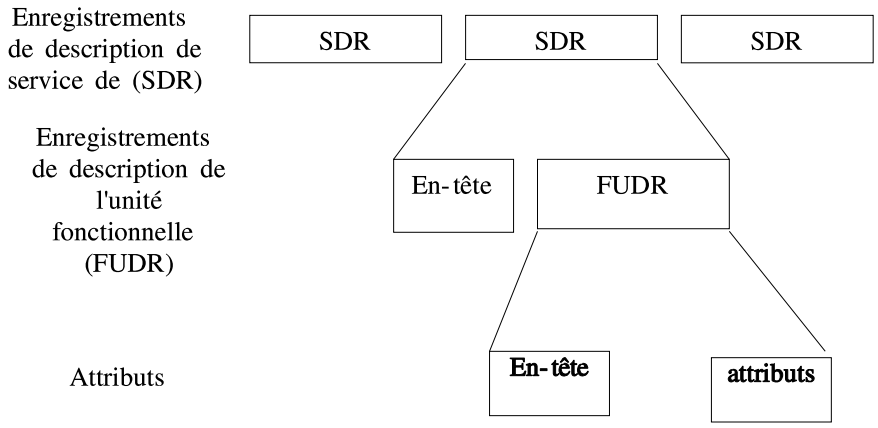


FIG. 2.7 – Structure des données au niveau de salutation.

Pour pouvoir décrire les différents services exportés et importés sur le réseau, trois types d'enregistrements sont définis :

- Enregistrement de description de services (SDR : Service Description Record)
- Enregistrement de description de l'unité fonctionnelle (FUDR : Functional Unit Description Record) : consiste en un champ entête contenant l'identificateur de l'unité fonctionnelle, exemple [Scan] et une clef suivi par zéro ou plusieurs enregistrements

d'attributs.
Les requetes et les réponses pour la recherche de services sont échangées sous forme de SDRs et FUDRs.

– Enregistrement d'attribut : cet enregistrement comporte un identificateur d'attribut identifiant le type d'attribut.
Dans le cas où l'attribut se trouve dans un FUDR enregistré sur la base des registres d'un Salutation Manager ou un FUDR représentant une requête, le champ suivant l'identificateur du type d'attribut sera un identificateur d'une fonction de comparaison d'attributs. Mais dans le cas où l'attribut se trouve dans un FUDR réponse (voir section 2.4.4), le champ suivant sera le résultat de la comparaison entre les deux attributs enregistrés et celui de la requête.

Les formats de l'enregistrement de description de service, de l'enregistrement de description de l'unité fonctionnelle et des enregistrements d'attributs sont spécifiés à l'aide d'une syntaxe abstraite définie par l'ISO 8824.
Il existe trois classes d'enregistrements de description de services :
– Les enregistrements de description de services enregistrés sur un Salutation Manager donné.
– Les enregistrements de description de services demandés représentant les requêtes utilisateurs.
– Les enregistrements de description de services Réponses représentant des réponses à des requêtes de recherche.
La fonction de ces trois classes sera mieux expliquée dans la section 2.4.4

2.4.4 Processus d'importation de services

L'application cliente construit un ou plusieurs enregistrements de description d'unités fonctionnelles englobants les différentes fonctions dont il a besoin et l'envoie au Salutation Manager à qui elle est reliée. Par exemple, si l'application cherche un simple service de FAX sans aucune particularité, elle enverra un simple enregistrement de description d'unité fonctionnelle qui est [FAX] sans aucune propriété. Par contre, si l'application a besoin d'un appareil ayant plusieurs fonctionnalités tels que l impression, le fax et le scannage, plusieurs enregistrements de description d'unités fonctionnelles doivent être rassemblés en un enregistrement de description de service.

Le salutation Manager de l'application cliente fait des recherches locales et distantes en envoyant l'enregistrement de description de service reçu vers d 'autres Salutations Managers. Cette recherche consiste en une comparaison entre le FUDR envoyé et celui enregistré dans la base de registres du Salutation Manager. S'il s'agit du même type d'unité fonctionnelle, il comparera les types et les valeurs des attributs.
Si la comparaison aboutit à un résultat positif, le Salutation Manager construit un nouveau FUDR représentant l'union entre le FUDR envoyé et celui enregistré. Ce nouveau FUDR est appelé

FUDR de comparaison et est renvoyé à l'application cliente comme résultat.
Si la comparaison échoue, un FUDR vide est retourné à l'application cliente.

2.4.5 API de Salutation Manager

Grâce à l'API de Salutation Manager [Sal01], les applications peuvent s'inscrire pour communiquer entre elles à travers le protocole de Salutation Manager indépendamment de la couche réseau. Cette API offre les fonctions suivantes (*Cf.* figures 2.8 et 2.9) :

Enregistrement de service : l'entité qui offre le service appelle slmRegisterCapability() ou slmUnregisterCapability() pour s'enregistrer ou s'effacer au niveau de Salutation Manager local comme une unité fonctionnelle.

Découverte de service : un client appelle slmSearchCapability() pour demander au Salutation Manager local de chercher des Salutations Managers contenant des unités fonctionnelles offrants certaines capacités. Le Salutation Manager local retourne au client, la liste des identificateurs de Salutation Managers contenant une unité fonctionnelle offrant le service demandé par le client. Si le client ne spécifie aucune capacité particulière en appelant slmSearchCapability(), la liste retournée, contiendra tous les identificateurs des Salutation Managers connus. Le client peut appeler aussi slmQueryCapability() pour découvrir quelles unités fonctionnelles sont enregistrées ainsi que leurs capacités sur le Salutation Manager spécifié par le client.

Utilisation de service : un client peut appeler slmOpenService pour demander au Salutation Manager local d'initier une session de service avec une unité fonctionnelle spécifique enregistrée au niveau du Salutation Manager Local ou d'un autre distant.
Le salutation Manager au niveau duquel l'unité fonctionnelle spécifiée est enregistrée, appelle la fonction fnOpenService() pour notifier l'unité fonctionnelle de cette demande d'ouverture de service. L'unité fonctionnelle peut accepter ou rejeter la demande.
Le résultat est renvoyé au client à travers le paramètre de retour de slmOpenService().
Une fois la session de service établie, le client appelle slmTransferData() pour que le Salutation Manager local puisse envoyer des données à l'autre bout de la session ouverte.
Le Salutation Manager se trouvant à l'autre bout, appelle la fonction fnReceiveData(), pour indiquer que les données sont bien reçues.
Le client et l'unité fonctionnelle refont le processus de transfert de données autant de fois qu'il faut.
Une fois le transfert terminé, le client appelle slmCloseService() pour demander au Salutation Manager local de fermer la session de service. Dans ce cas, le Salutation Manager distant appelle la fonction fnCloseService() pour notifier pour la fermeture de session.

Vérification de disponibilité : l'unité fonctionnelle peut appeler slmStartAvailabilityCheck pour demander au Salutation Manager local de vérifier périodiquement si une unité fonctionnelle spécifique enregistrée au niveau du Salutation Manager local ou distant est encore enregistrée et tourne.
Si Salutation Manager trouve que l'unité fonctionnelle spécifiée n'est plus disponible, il appelle la fonction fnNotifyFUUnvailability(). L'unité fonctionnelle appelle slmCancelAvailabilityCheck() lorsque la vérification de disponibilité n'est plus nécessaire.

Autres : Le client peut appeler slmGetVersion() pour avoir le numéro de version de l'API du Salutation Manager. Il peut aussi appeler slmGetLocalSLMID() pour avoir l'identifiant du Salutation Manager local ou appeler slmGetSLMIDbyAddr() pour avoir l'identifiant de Salutation Manager local sous forme de son adresse réseau.

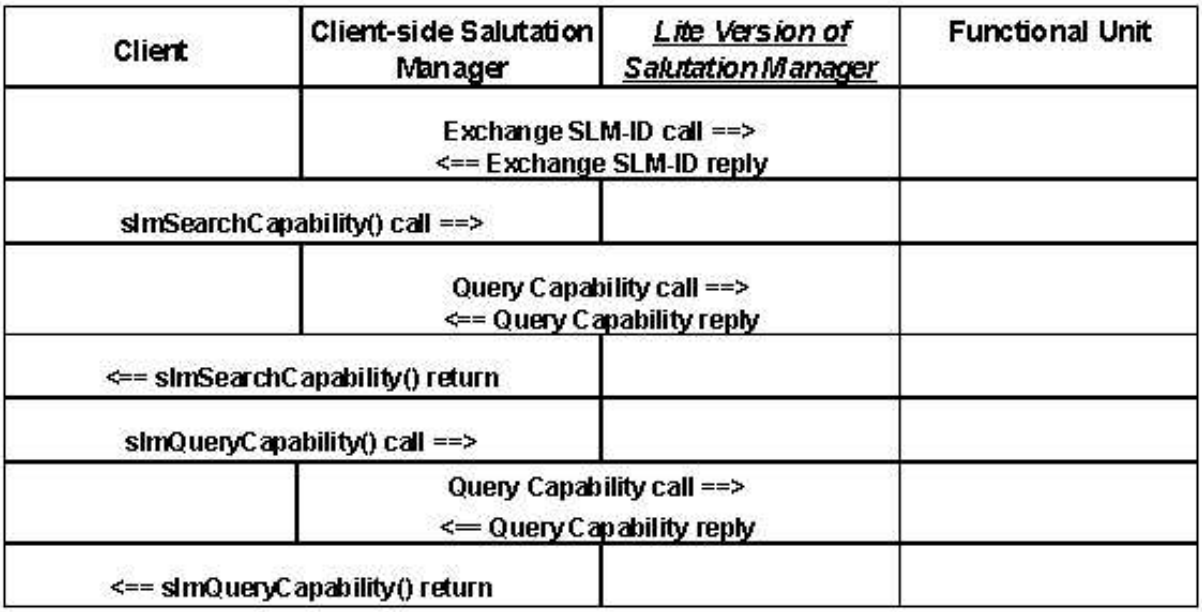


FIG. 2.8 – Découverte et utilisation de service au niveau de salutation.

2.4.6 Conclusion

Salutation se distingue par une API complètement indépendante des couches inférieures du réseau et par un protocole de communication qui lui est propre. Mais ce service de recherche est, comme Jini, limité à la racherche de services de proximité.

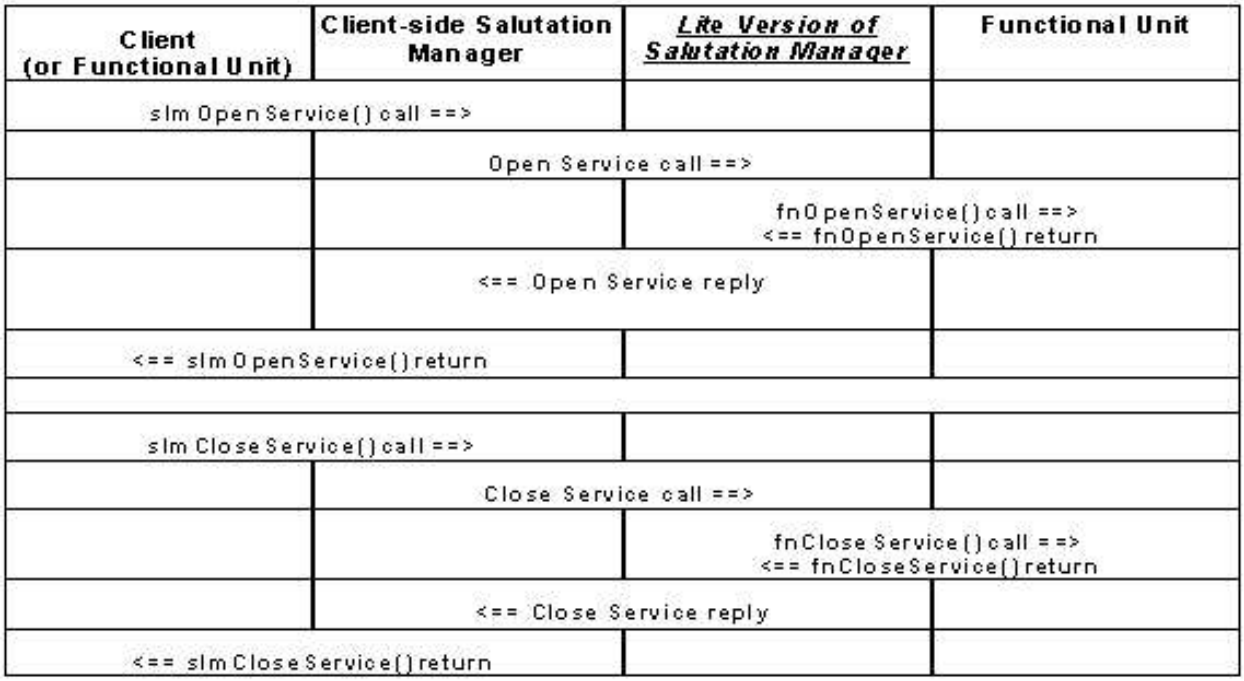


FIG. 2.9 – Ouverture et fermeture d'une session de service au niveau de salutation.

2.5 Le service de recherche de UDDI

2.5.1 Présentation de UDDI

UDDI (Universal Description, Discovery & Integration) est un projet industriel qui a été lancé en Septembre 2000 par Ariba, IBM, Microsoft et trente trois autres compagnies. Aujourd'hui, le consortium UDDI compte plus de 200 membres. UDDI est le nom d'un groupe de bases de registres web exposant des informations sur des affaires, des entreprises des vendeurs...[udd01]
Le but principal de UDDI est de faciliter l'intégration B to B [1] et offrir des mécanismes permettant de découvrir les interfaces de programmations (APIs) offertes par une entreprise pour interagir avec les autres à travers le commerce électronique.
Les informations qu'une entreprise peut enregistrer dans la base des registres sont des informations sur son nom, des contacts, des codes industriels, des classifications de produits, des URLs, l'ensemble des services offerts ainsi que des informations sur leurs interfaces techniques et leurs fonctionnements.

2.5.2 Exemple

Supposons que l'entreprise X expose un service de facturation Web afin que les fournisseurs de l'entreprise puissent envoyer des factures électroniques. De la même manière, un vendeur V

[1] Business to Business

peut offrir un service Web permettant d'envoyer des commandes d'une manière électronique. Si l'entreprise X, veut acheter des équipements informatiques à travers le web, elle cherchera tous les vendeurs d'équipements informatiques sur le web en utilisant UDDI. Supposons maintenant que l'entreprise X veuille savoir lequel de ces vendeurs d'équipements informatiques offre des services web compatibles avec ses systèmes. Par exemple, si elle supporte des commandes électroniques se basant sur SOAP, elle aura donc besoin d'un vendeur qui accepte des commandes de ce type qui soient compatibles avec son processus. Pour satisfaire ce genre de besoin, chaque entreprise enregistre au niveau de UDDI tous ses services ainsi que leurs types. Ce type de service aura un identificateur unique et fera partie d'un groupe de types de services bien connus enregistrés dans UDDI. Ces types de services sont appelés tModels. Chaque tModel a un nom, une description et un identificateur unique. Cet identificateur est appelé tmodelkey et il est noté UUID (Universal Unique Identifier).
En ayant un groupe de types de services bien définis, UDDI offre la possibilité de savoir comment faire du ebusiness avec une entreprise donnée. Ceci est l'avantage principal de UDDI par rapport aux autres pages jaunes du web telles que celles de Yahoo, Lycos, Mamma ou les autres.

2.5.3 UDDI : structure des données

UDDI utilise XML (Extensible Markup Language) [XML00] et une technologie qui lui est reliée appelée SOAP (Simple Object Access Protocol) qui est une spécification pour l'utilisation de XML dans les échanges de messages[SOA00].

Les informations enregistrées au niveau de UDDI, utilisent cinq types de structures de données. Cette division par type d'information offre une simple partition qui facilite la compréhension et la localisation des informations enregistrées. La figure 2.10 représente ces cinq types de structures de données.
Chacun des cinq types de structures de données est utilisé pour exprimer des types de données spécifiques arrangés dans une relation qui suit le modèle de la figure 2.10.

Les données gérées par UDDI sont sensibles à la relation parent/descendant . Comme le montre la figure 2.10, la structure de données BusinessEntity contient un ou plusieurs structures businessService. Une structure businessService englobe des instances spécifiques de structures bindingTemplate contenant chacune à leur tour un pointeur vers une structure tModel. Ce genre de relation s'appelle " une relation de contenance ".
Une structure bindingTemplate ne peut pas pointer vers plus qu'une structure tModel.
Aucune structure ne peut être contenue par plus d'une structure parente.

Les informations sur les affaires, leurs services et leurs informations techniques sont séparées de manière qu'ils soient accessibles individuellment à l'aide d'identifiants uniques notés UUID (Universal Unique Identifier) ou clés. Ces clés sont attribuées lorsque l'information est enregistrée pour la première fois au niveau de la base de registres de UDDI.

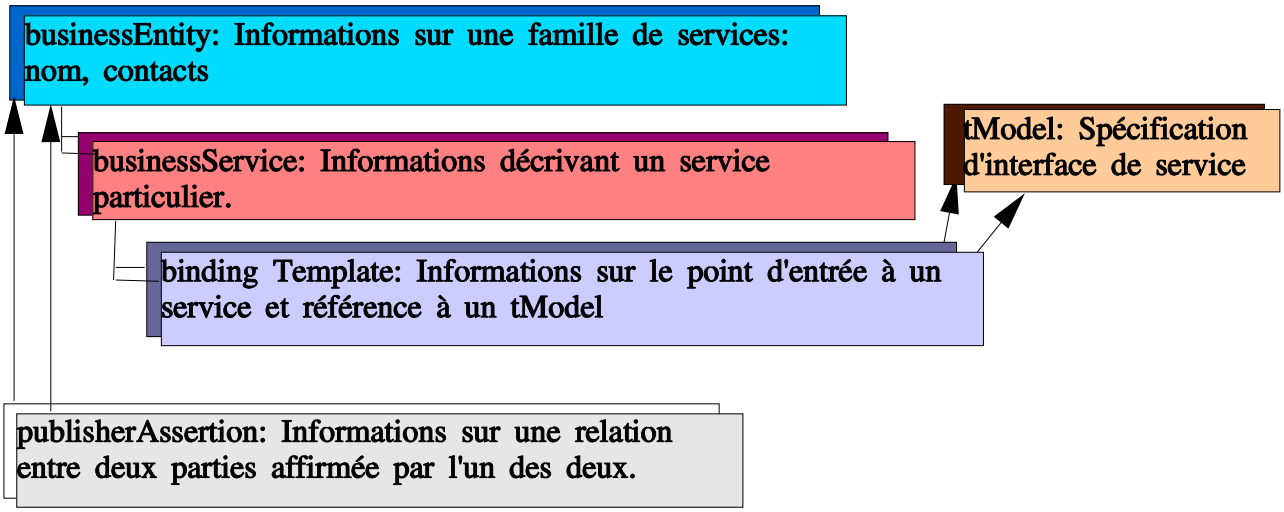


FIG. 2.10 – Structure des données UDDI.

2.5.3.1 La structure de données businessEntity

La structure businessEntity représente toutes les informations connues sur le business et les services qu'il offre :

```
<element name = "businessEntity">
<complexType>
<sequence>
<element ref = "discoveryURLs" minOccurs = "0"/>
<element ref = "name" maxOccurs = "unbounded"/>
<element ref = "description" minOccurs = "0"
maxOccurs = "unbounded"/>
<element ref = "contacts" minOccurs = "0"/>
<element ref = "businessServices" minOccurs = "0"/>
<element ref = "identifierBag" minOccurs = "0"/>
<element ref = "categoryBag" minOccurs = "0"/>
</sequence>
<attribute ref = "businessKey" use = "required"/>
<attribute ref = "operator"/>
<attribute ref = "authorizedName">
</complexType>
</element>
```

D'un point de vue XML, comme la structure businessEntity se trouve au niveau le plus haut, elle ne contiendra que des informations descriptives du business comme le nom et la description de l'industrie, des informations permettant d'avoir un contact avec des personnes responsables ou autres (emails, adresses, tel ...), des URLs, une liste contenant zéro ou plusieurs businessServices (voir section 2.5.3.2), ...
Les informations sur les services et les informations techniques sont exprimées au niveau de la businessService à travers la relation de contenance.

2.5.3.2 La structure de données businessService

Chaque structure businessService représente une classification logique de services et représente le descendant d'une seule structure businessEntity.

```
<element name = "businessService">
<complexType>
<sequence>
<element ref = "name" maxOccurs = "unbounded"/>
<element ref = "description" minOccurs = "0"
maxOccurs = "unbounded"/>
<element ref = "bindingTemplates"/>
<element ref = "categoryBag" minOccurs = "0"/>
</sequence>
<attribute ref = "serviceKey" use = "required"/>
<attribute ref = "businessKey"/>
</complexType>
</element>
```

L'identité du businessEntity parent est déterminée en examinant la valeur du champ businessKey. Mais l'identifiant unique d'un businessService est donné par le champs serviceKey. La structure d'un businessService comporte aussi un nom et une description, d'un service ainsi qu'une liste de de zéro ou plusieurs bindingTemplates (voir section 2.5.3.3).

2.5.3.3 La structure de données bindingTemplate

Les descriptions techniques des services web sont placées dans des instances des structures bindingTemplate. Chaque structure bindingTemplate est la fille d'une seule structure businessService.

```
<element name = "bindingTemplate">
<complexType>
<sequence>
<element ref = "description" minOccurs = "0" maxOccurs = "unbounded"/>
<choice>
<element ref = "accessPoint" minOccurs = "0"/>
<element ref = "hostingRedirector" minOccurs = "0"/>
</choice>
<element ref = "tModelInstanceDetails"/>
</sequence>
<attribute ref = "bindingKey" use = "required"/>
<attribute ref = "serviceKey"/>
</complexType>
</element>
```

La structure d'un bindingTemplate comporte un identifiant du service auquel appartient ces informations techniques (serviceKey) et un identifiant unique du bindingTemplate. Un bindingTemplate représente un point d'entrée aux différents services web (acessPoint). Ce point d'entrée peut par exemple représenter une URL qui permet d'accéder directement au service afin de l'utiliser.

2.5.3.4 La structure de données tModel

Le but principal de UDDI est de décrire les services web de manière à pouvoir les rechercher mais aussi d'avoir l'information nécessaire qui permet d'interagir avec le service. Les tModels sont des sortes de types de services. Ils servent à déterminer les compatibilités entre services et à les décrire d'une manière qui facilite leur recherche.

```
<element name = "tModel">
<complexType>
<sequence>
<element ref = "name"/>
<element ref = "description" minOccurs = "0"
maxOccurs = "unbounded"/>
<element ref = "overviewDoc" minOccurs = "0"/>
<element ref = "identifierBag" minOccurs = "0"/>
<element ref = "categoryBag" minOccurs = "0"/>
</sequence>
<attribute ref = "tModelKey" use = "required"/>
<attribute ref = "operator"/>
<attribute ref = "authorizedName"/>
</complexType>
</element>
```

Cette structure comporte un identificateur unique (tModelKey), le nom du tModel, le nom du site opérateur UDDI incluant dans sa base de registres le tModel, une description du tModel et des références sur des documentations techniques.

2.5.3.5 La structure de données publisherAssertion

Plusieurs grandes entreprises ne sont pas représentées par une seule businessEntity parce qu'ils offrent des services de natures diverses. Plusieurs structures businessEntity peuvent donc être publiées pour représenter une seule filiale. Deux entreprises liées utilisent donc des messages xx_publisherAssertion pour indiquer la relation existante entre les deux.

2.5.4 Interface de programmation UDDI

UDDI offre un ensemble d'APIs sous forme de services web basés sur le protocole SOAP. Pour l'instant, il y a deux sites opérateurs offrant des services web UDDI, un de Microsoft [uddb] et un autre de IBM [udda]. Ces deux sites implémentent l'API UDDI pour permettre aux différentes entreprises d'exporter et importer leurs services.

[udd01] présente une spécification détaillée des APIs UDDI, ces APIs se divisent généralement en deux catégories : des APIs d'exportation et des APIs d'importation comme le montre le tableau 2.1.

L'invocation des APIs UDDI est assurée par envoie de messages SOAP avec le contenu approprié. Par exemple pour chercher l'entreprise XYZ, il faut envoyer le bout de code XML suivant dans le corps d'un message SOAP.

```
<find_business generic='1.0' xmlns='urn :uddi-org :api'>
<name>XYZ</name>
</find_business>
```

Le message SOAP qui sera retourné à partir du service UDDI contiendra toutes les entreprises enregistrées dont les propriétés correspondent aux critères demandés par l'utilisateur.

Recherche d'une entité	Catégorie
find_business	importation
find_service	importation
find_binding	importation
find_tModel	importation
Obtenir des détails à propos d'une entité	Catégorie
get_businessDetail	importation
get_serviceDetail	importation
get_bindingDetail	importation
get_tModelDetail	importation
Enregistrement d'entité	Catégorie
save_business	exportation
save_service	exportation
save_binding	exportation
save_tModel	exportation
Effacement d'entité	Catégorie
delete_business	exportation
delete_service	exportation
delete_binding	exportation
delete_tModel	exportation

TAB. 2.1 – API UDDI

2.5.5 Conclusion

UDDI offre un servie de pages jaunes sur le Web. Sa particularité par rapport aux autres pages jaunes du web réside dans le fait qu'il offre des opérations d'interopérabilité entre les différents services offerts sur le web et enregistrés sur sa base de registres. L'inconvénients de ce service est qu'il n'offre aucune fédération de la recherche.

2.6 Etude comparative des services de recherche sur propriétés

Le tableau 2.2 présente une comparaison entre les quatres services de recherches : Trader CORBA, Jini, Salutation et UDDI.

La connexion au service de recherche Jini, présente une particularité par rapport aux autres services, dans le sens où il s'agit de commencer par chercher le service de recherche lui-même en faisant un multicast sur le réseau, une application n'est donc pas reliée à un service de recherche par défaut comme dans le cas du trader Corba et salutation.
Pour se connecter au trader Corba, une application a un fichier de configuration à partir duquel elle récupère la référence d'objet interopérable (IOR) du trader au niveau duquel elle peut exporter ou lancer une importation d'un service. De la même manière, chaque machine voulant bénéficier du service de recherche salutation, doit être reliée localement ou à distance à un Salutation Manager qui lui est propre.
Le service de recherche de UDDI est toujours offert par un site opérateur sur le web. Toute personne désirant rechercher un service peut donc aller directement sur le site et faire des recherche avec des mots clès. Mais les personnes qui désirent exporter des services doivent créer un compte sur le site. Seul Jini offre donc une détermination dynamique du service de recherche.

Concernant la structuration des données au niveau du service de recherche, le trader Corba est le seul service de recherche qui propose la notion de type de service à laquelle doit se conformer tout type d'offre de service, ce qui offre un typage fort des différentes offres. Jini offre aussi un typage fort pour les différentes offres grâce à la classe ServiceItem 2.3.5 de manière à ce que chaque offre soit une instance de cette classe.
Au niveau de salutation, une offre de service doit être sous forme de FUDR 2.4.3 et au niveau de UDDI, chacune des structures de données businessEntity, serviceEntity, bindingTemplate et tModel ont une structure bien définie en XML[spec] et chaque offre de service doit se conformer à cette structure.
La structure des données au niveau de Salutation et UDDI n'offre aucun typage fort mais sert juste à identifier les différents champs et attributs constituant les offres de service.

Le trader Corba et Salutation se distinguent par leur fédération de la recherche de services. Mais, la fédérations de recherche du trader est plus étendue et se base sur des graphes orientés de recherche.

	Trader CORBA	Jini	Salutation	UDDI
Connexion	IOR du trader dans un fichier de configuration	Multicast et recherche d'un service de recherche de proximité	Mise en place d'un SLM pour chaque entité	Création de compte pour publier un service
Structure des données	Conforme à des types de services	Instance de la classe ServiceItem	SDR, FUDR,	Format XML
Fédération	Traders reliés entre eux par un graphe orienté	Pontage entre services de recherche	RPC pour communiquer avec des SLM distants	pas de fédération de recherche
Types des propriétés	Simples ou composés, support de propriétés dynamiques, support de propriétés modifiables	sous forme de classes, support de propriétés modifiables	Simples ou composés	Format texte
Filtres de selection de services	Opérations logiques et mathématiques entre valeurs et noms de propriétés	Égalié entre propriétés	Égalié entre propriétés, union entre FUDR envoyée et enregistrée	Égalité entre propriétés
API, publication	Interface Register	ServiceRegistrer. register	SlmRegister-Capability	Publishing API
API, recherche	Interface Lookup	ServiceRegistrer. lookup	SlmSearch-Capability	Inquiry API
Particularité	Politiques de recherche	Connexion dynamique à un service de recherche	Interface indépendante du transport	Offre des informations d'interopérabilité entre services
Limitations	Pas simple à utiliser, nécessite beaucoup de lignes de code	Limité aux services de proximité	Limité aux services de proximité	Propre aux industries sur le web

TAB. 2.2 – Comparaison des services de recherche sur propriétés

Jini permet de faire le pontage avec des services de recherche d'autres types comme Salutation ou autres. Par contre, Il est impossible de rechercher un service qui a été enregistré sur un site opérateur différent de celui au niveau duquel on fait la recherche.

Chacun des services Salutation, Jini et trader Corba supportent des propriétés de types simples et composés. Au niveau de Jini, les propriétés sont sous forme de classes et au niveau de UDDI, toutes les propriétés sont au format texte.
Le trader Corba et Jini permettent tous les deux de modifier des valeurs de propriétés de services déjà enregistrés.
Le trader corba est le seul qui supporte les propriétés dynamiques.

La sélection de services au niveau de UDDI, Jini et Salutation se fait avec des tests d'égalité entre propriétés. Le trader corba est le seul qui propose un langage pour définir les contraintes et les préférences (OCL), qui peut faire des opérations logiques et mathématiques entre valeurs et noms de propriétés.

Chacun des quatre services de recherche que nous avons étudié offre une API servant essentiellement à exporter et importer les services et gérer les services enregistrés dans un dépositaire. L'API du trader Corba est la plus difficile à implémenter ; une recherche faisant appel à la méthode query de l'interface lookup nécessite une centaine de lignes de codes [LMMG01]. Mais l'API du trader Corba est la seule qui permet la définition de politiques de recherche et offre une recherche très étendue par rapport à Jini et Salutation qui se limitent à la recherche de services de proximité.
Salutation est le seul qui offre une API complètement indépendante de la couche réseau et peut être implémentée avec n'importe quel language. Les APIs de Jini ne peuvent être implémentés qu'en Java, le trader Corba se limite aux objets Corba et le service de recherche UDDI n'est valable que dans un environnement Web.
Contrairement au trader Corba qui permet de rechercher une référence sur un objet Corba, Jini permet la recherche d'objets sérialisés.
Ce qui distingue UDDI par rapport aux autres services de recherche c'est sa capacité d'offrir des informations sur l'interopérabilité et la compatibilité entre les différents services.

2.7 Conclusion

Nous avons étudié quatre services de recherche sur propriétés. Chacun de ces services présente des avantages et des inconvénients. Le trader Corba étant le plus complet, nous avons choisi de l'utiliser dans notre infrastructure de déploiement présentée dans le chapitre 4. Nous avons pu pallier à son inconvénient majeur qui est la complexité de son API en utilisant en utilisant TORBA (contrats de courtage pour TORBA) [LMMG01].

Chapitre 3

Les applications à base de composants

L'un des buts principaux de ce stage, consiste à étudier le déploiement dynamique des applications multi-composants. Il nous est donc nécessaire de présenter la notion de composants logiciels qui peut être nouvelle pour certains. Nous commencerons, dans ce chapitre, par présenter d'une manière générale les composants et les différents modèles de composants existants. Par la suite, nous décrirons le modèle de composants de Corba (CCM : Corba Component Model) en focalisant sur son apport majeur qu'est le déploiement.

3.1 Limites des applications orientées objet

Il y a eu au cours de ces dernières années, un courant de recherche très actif portant sur l'utilisation des modèles et des langages à objets pour la construction d'une application répartie. Ces modèles permettaient d'une part, d'exprimer les interfaces des entités logicielles manipulées résultat de la conception et d'autre part, de bien séparer la définition de ces interfaces de l'implantation des entités logicielles. L'objectif principal des modèles objet était d'améliorer la modélisation d'une application, d'optimiser la réutilisation du code produit, et d'adresser l'ensemble du cycle de production des applications. Cependant, l'intégration d'entités logicielles existantes peut s'avérer difficile si leur modèle d'exécution est incompatible avec le modèle imposé par le langage à objet choisi pour le développement de nouvelles entités. Par ailleurs, les modèles et langages à objets ne sont pas, en général, adaptés à la description des schémas de coordination et de communication complexes. En effet, un grand nombre de ces modèles offrent à leurs utilisateurs un mode de communication et d'exécution fondé sur des interactions synchrones de type client-serveur.

Pour pallier les défauts de l'approche objet et adresser des notions non prévues initialement, l'approche composant est apparue [MMG99].

3.2 Modèles de composants

Une application construite selon une approche orientée composant peut être vue comme une collection de logiciels indépendants, interconnectés à l'aide d'une plate-forme de communication.

Un composant est un module logiciel autonome pouvant être configuré et installé sur différentes plates-formes. Ce composant, exporte différents attributs et méthodes[MMG99].
Un composant logiciel peut être comparé à un composant électronique (circuit intégré). Pour construire un circuit électrique, il s'agit d'interconnecter un ensemble de composants électroniques vues comme des boites noires et dont on ne connaît que le principe de fonctionnement, les services rendus, quelques règles de connexion et la manière dont chacun d'eux peut communiquer avec son environnement (tension d'entrée, de sortie, ...). La construction d'applications à base de composants logiciels repose sur le même principe.

L'approche composant encourage les développeurs à construire les différentes parties de leurs applications sous forme de modules réutilisables.
Une conception soignée, doit permettre la personnalisation de ces modules dans l'environnement opérationnel. Le processus de la personnalisation d'une application dans un environnement donné s'appelle le déploiement.
Les composants sont donc les unités de base pour la construction d'applications distribuées.

Le développement à base de composants se fait généralement en trois étapes : la conception de chaque composant, leur implémentation et leur intégration (ou assemblage). La conception consiste à décrire les fonctionnalités des différents composants en spécifiants leurs ports (interfaces, événements, ...). L'étape d'implémentation consiste à écrire le code interne du composant. Par la suite, le composant peut être déployé (instancié sur un serveur) et peut être assemblé avec un autre composant à l'intérieur d'un framework d'une application distribuée. La construction d'applications par assemblage se fait à l'aide de langages de description d'architectures.

Il existe plusieurs modèles de composants :
- Modèles universitaires : Durra du CMU [BWD+93], Darwin de l'Imperial College [KM98], UniCon du CMU [SDK+95], Olan développé au laboratoire SIRAC [BBABR96] de l'INRIA-Rhône-Alpes et JavaPods [BR00] du même laboratoire.
- Modèles industriels : .net de Microsoft [Rog97], JavaBeans [Jav] et Enterprise Java Beans [EJB] de Sun Microsystems.
- Modèles de référence : ODP (Open Distributed Processing) de l'ISO et l'ITU et CCM [com] (Corba Component Model) de l'OMG (Object Management Group)

Nous avons choisi de décrire quelques concepts du modèle CCM de corba et cela, pour les raisons suivantes :
- le modèle de composants .net [1] [Rog97] est propriétaire et principalement destiné aux

[1] ou bien COM, DCOM et COM+ qui sont les prédécesseurs de .net

plates-formes Microsoft de type PC. En plus, les spécifications ne sont jamais stables.
– Le déploiement des EJB [EJB] est essentiellement mono-site, ce qui répond essentiellement aux besoins des applications de type commerce électronique s'exécutant sur des sites marchands, mais pas aux applications réellement distribuées.

Nous estimons donc que CCM est la spécification de modèle de composants la plus complète.

3.3 Le modèle de composants CCM (Corba Component Model)

La spécification du CCM [com99] est découpée en quatre modèles et un méta-modèle :

– Le modèle abstrait : ce modèle offre aux concepteurs le moyen d'exprimer les interfaces (fournies et utilisées) et les propriétés des différents types de composants ainsi que leurs gestionnaires d'instances.
– Le modèle de programmation : ce modèle spécifie le langage CIDL (Component Implementation Definition Language) à utiliser pour définir la structure de l'implantation d'un type de composant, ainsi que certains de ses aspects non fonctionnels (persistance, transactions, sécurité). L'utilisation de ce langage est associée à un framework, appelé CIF (Component Implementation Framework), qui définit la manière avec laquelle les parties fonctionnelles (programmées) et non-fonctionnelles (décrites en IDL / CIDL et générées) doivent coopérer. Il inclut aussi la manière dont l'implantation d'un composant interagit avec le container.
– Le modèle de déploiement : ce modèle définit un processus qui permet d'installer une application sur différents sites d'exécution de manière simple et automatique. Le modèle de déploiement s'appuie sur l'utilisation de paquetages déployables et composables de composants, ainsi que de descripteurs de déploiement OSD (Open Software Description) [2]. Le modèle de déploiement introduit trois types de descripteurs exprimés en OSD. Ces descripteurs contiennent des informations sur : les implantations d'un composant (et les besoins logiciels associés), les composants (interface, services, état,. . .) et la politique d'assemblage des différents composants qui forment l'application [MMG00].
– Le modèle d'exécution définit l'environnement d'exécution des instances de composants (c'est un modèle de container). Le rôle principal des containers est de masquer et prendre en charge les aspects non fonctionnels des composants qu'il n'est alors plus nécessaire de programmer.
– Le méta-modèle de composant est basé sur UML et dispose de projection vers le MOF (Meta Object Facility)

[2] OSD est un vocabulaire XML

3.3.1 Le modèle abstrait de composants de l'OMG

Les composants apportent au monde CORBA la possibilité d'avoir plusieurs interfaces associées avec un unique objet. La norme Composants regroupe sous le nom de port tout mode d'interaction qui existe par rapport à un composant. Quatre types de ports sont définis dans le contexte du CCM.

- Les facettes : une facette est une interface fournie par un type de composant et qui est utilisée par des clients en mode synchrone. Seule l'interface dont le client a besoin est fournie.
  Si on regarde un distributeur de boissons, on y voit plusieurs interfaces : une interface pour le consommateur (payer, selectionner une boisson, prendre la boisson), une interface pour le fournisseur (ouvrir distributeur, mettre des boissons, mettre de la monnaie, vider la caisse, fermer le distributeur) et une interface pour le dépanneur (ouvrir le capot arrière, fermer le capot arrière).
- Les réceptacles : un réceptacle est une interface utilisée par un type de composant en mode synchrone. Il s'agit d'un point de connexion conceptuel. Il permet donc d'assembler des composants et des objets. La composition permet de concevoir plus facilement des applications en permettant aux composants de faire de la délégation de traitement.

  Il y a deux types de ports qui traitent des événements :
- Les puits : un puit d'événement est une interface fournie par un type de composant et utilisée par ses clients en mode asynchrone. Il permet à un composant de recevoir des événements d'un certain type.
- Les sources : une source d'événements est une interface utilisée par un type de composant en mode asynchrone. Il y a deux catégories de sources d'événements : les connectables en mode un vers un, et les connectables en mode un vers n.

3.3.2 Les maisons de composants

Une maison de composants est un gestionnaire pour instances d'un même composant. Elle gère le cycle de vie des instances, éventuellement à l'aide de clés primaires. Pour cela, elle offre une fabrique d'instances de composant et des outils de recherche qui utilisent des clés. Le cycle de vie d'un composant se décompose en deux phases : la phase de configuration et la phase fonctionnelle. La configuration d'un composant repose principalement sur le positionnement des valeurs de ses attributs.

3.3.3 Le modèle de programmation

Le CIF (Component Implementation Framework) définit le modèle de programmation pour implanter des composants. Il inclut un langage déclaratif : CIDL (Component Implementation Description Language) pour décrire les implantations de composants, de leurs maisons et de l'état des composants. Le CIF utilise CIDL pour générer les squelettes et les souches. Cela automatise l'implantation de nombreux comportements de base (navigation, activation, gestion de l'état, du

cycle de vie, . . .). Le CIF est conçu pour être compatible avec le framework du POA [3], tout en masquant sa complexité.

3.3.4 Le modèle d'exécution

Un container est un environnement d'exécution pour une instance de composant CORBA. Cet environnement est implanté comme un serveur d'applications et/ou comme une plate-forme de production de telles applications.
Toutes les instances de composants, quel que soit leur type, sont créées et gérées par un container. Une instance de composant ne peut pas exister sans être supportée par un container. D'autre part, un type de container ne peut accueillir qu'un unique type de composant pour lequel il a été conçu. Deux types de containers ont été définis : des volatiles et des persistants.

3.3.5 Le modèle de déploiement

Le déploiement des composants permet d'automatiser la diffusion et la mise en place d'une application distribuée.

3.3.5.1 Etapes de production et de déploiement des applications

Les étapes de production des applications à base de composants sont les suivants :

1. Définition des types de composants à l'aide du langage IDL étendu définit par CORBA 3.
2. Projection des types de composants en IDL compatible CORBA 2 tout en produisant un descripteur de composants dont l'implantation doit être complété par le développeur.
3. Implantation du type de composant.
4. Pour être diffusable, le type de composant est packagé avec son descripteur et une configuration par défaut, au sein d'une archive.
5. Inclusion du paquetage de composant dans un assemblage.
6. Paquetage de l'assemblage de composants afin de regrouper dans une archive les implantations, le descripteur de l'assemblage et la configuration des composants pour ce contexte précis. Grâce à ce paquetage, les paquetages d'assemblage sont diffusables ou réutilisables dans une nouvelle composition.
7. Déploiement des paquetages de composants et d'assemblages et instanciation des applications sur leurs sites d'exécutions.

La figure 3.1 [MMG00], résume les étapes que nous venons de décrire. Dans ce qui suit, nous allons présenter les différents types de paquetages et de descripteurs nécessaires au déploiement, que nous venons de citer dans les étapes ci-dessus.
Dans le modèle CCM, chaque composant est placé dans un paquetage et a des descripteurs qui lui sont propres. En plus des paquetages de composants, nous disposons de paquetages d'assemblage qui permettent la connexion des différents composants.

[3] Portable Object Adapter

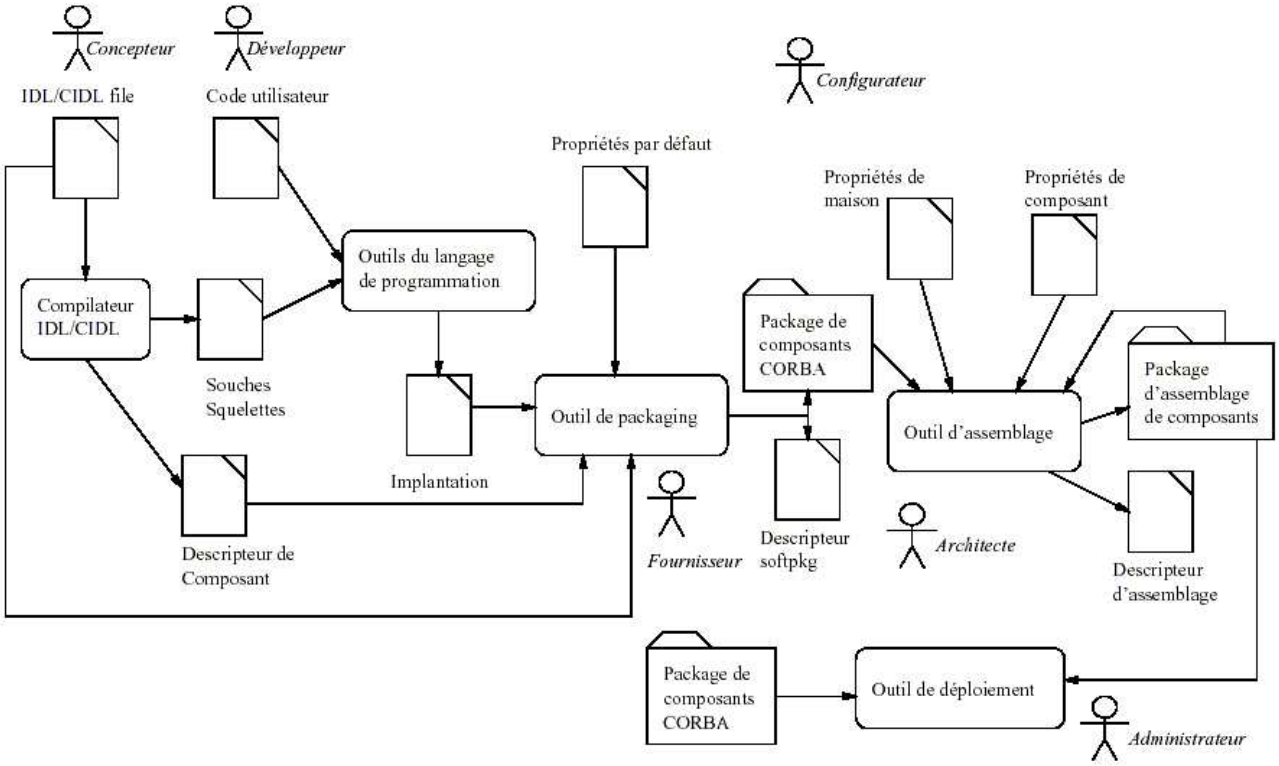


FIG. 3.1 – Etapes de production et de déploiement des applications.

3.3.5.2 Paquetages de composant

Un paquetage d'entité logicielle est l'unité de déploiementest. Il est représenté par un descripteur et un ensemble de fichiers contenant l'implantation de l'entité. Ce descripteur décrit le contenu du paquetage. Il offre des informations d'ordre général relatives à l'entité logicielle (auteur, description, interface OMG IDL, etc) ainsi qu'une description des implantations (nom, système d'exploitation cible, langage d'implantation, dépôt de code, etc). Ce descripteur regroupe aussi les dépendances du composant par rapport à l'environnement.

3.3.5.3 Descripteurs de composants

Ces descripteurs spécifient les caractéristiques d'un composant spécifiées pendant la phase de conception et la phase de développement. Il existe deux types de descripteurs de composants.

- Un descripteur de fonctionnalités contenant des informations se rapportant à la structure d'un composant : héritage, interfaces supportées, ports, ... Ce type de description permet à un outil de connecter les composants entre eux lors de la phase de déploiement.
- Un descripteur de déploiement qui contient des informations de déploiement servant à déterminer le type de structure d'accueil nécessaire au composant et fournir aux structures d'accueil des informations sur le composant. Ces informations indiquent aussi les besoins transactionnels du composant, la qualité de service des ports, des informations concernant les capacités multithread du composant et leur mode d'utilisation, ainsi que la manière d'achever la phase de configuration.

Ces descripteurs sont en partie générés par un compilateur CIDL, et en partie modifiés par un outil de packaging (pour les notions de persistance, transaction, threading,. . .)

3.3.5.4 Paquetage d'assemblage de composants

Un paquetage de composant permet de déployer un composant "seul". Un paquetage d'assemblage permet de déployer de manière simple des composants dépendants les uns des autres. Il offre un patron (ou *template*) pour instancier un ensemble de composants et les connecter les uns aux autres. Un tel paquetage regroupe un descripteur, un ensemble de paquetages de composants et un ensemble de fichiers de propriétés.
Le fichier d'assemblage est une archive regroupant le descripteur, les archives de composants et les fichiers de propriétés.
Le descripteur d'assemblage de composants décrit l'assemblage des composants, c'est-à-dire les éléments de description des composants, des connexions et du partitionnement. Le descripteur référence les fichiers de chaque composant et décrit les connexions entre composants. Ce descripteur définit des regroupements logiques pour les composants, regroupements qui seront projetés au déploiement sur des sites physiques. Le descripteur d'assemblage va servir de base au processus de déploiement.
Le descripteur de propriétés décrit le paramétrage des composants et des maisons de composants. Il est utilisé pour configurer un composant ou une maison : positionnement des attributs. Il fournit des propriétés par défaut qui peuvent être modifiées par l'utilisateur. Les informations contenues dans ce type de descripteur seront utilisées par les objets de configurations.

3.3.5.5 Déploiement

Le déploiement se fait en quatre étapes de base :

1. Le choix des sites d'installation de composants.
2. Installation des implantations où cela est nécessaire (si elles ne sont pas déjà présentes).
3. Instanciation des maisons suivie de celle des composants.
4. Connexion des composants (dans le cas des assemblages).

Seulement, les implémentations faites du modèle de déploiement du CCM restent statiques et n'utilisent pas des procédures totalement automatisées puisque le choix des sites d'installation de composants est fixé à l'avance.

3.4 Conclusion

Le modèle de déploiement CCM contribue à accroître la réutilisabilité d'entités logicielles en facilitant l'utilisation et l'intégration de composants existants. Seulement, aucune implémentation totalment automatisée, de ce modèle, n'a été proposée. Les implémentation faites jusqu'aujourd'hui ne présentent aucun aspect dynamique par rapport à l'adaptation au contexte d'exécution de l'utilisateur.

Chapitre 4

Infrastructure générale de déploiement

Jusqu'à nos jours, le déploiement des applications reste statique et n'utilise pas des moyens qui l'automatisent totalement. Le but que nous cherchons est donc d'automatiser et rendre dynamique le déploiement des applications.

Dans ce chapitre, nous allons décrire une infrastructure qui offre un ensemble d'applications accessibles par les utilisateurs à partir d'endroits différents sans avoir à les installer manuellement sur chaque terminal. Cette disponibilité des applications est rendue possible grâce à un mécanisme de déploiement d'applications à la demande qui prend en compte le contexte de connexion de l'utilisateur.
Nous allons commencer par décrire l'infrastructure de déploiement que nous proposons ainsi que les différentes étapes du déploiement automatique. Ensuite nous traiterons le problème de la terminaison de déploiement. Nous insisterons principalement sur le rôle et les fonctionnalités du serveur de déploiement. Enfin, nous terminerons par présenter une sémantique pour le déploiement en cas d'échec.

4.1 Description de l'infrastructure de déploiement

Un processus de déploiement est l'ensemble des opérations automatiques nécessaires pour qu'un utilisateur puisse accéder à un service faisant intervenir plusieurs hôtes sans avoir à l'installer manuellement. Ce processus est invoqué à chaque fois qu'un utilisateur se connecte au réseau pour utiliser un service.
Les utilisateurs peuvent se connecter aux différents services offerts à partir de terminaux et d'environnements différents : ils peuvent se connecter à partir d'un PC quelconque, d'un PDA, d'un téléphone portable ou d'un ordinateur portable et cela à partir d'endroits différents et en utilisant des systèmes différents. Il est donc important que le déploiement prenne en compte le contexte de connexion de l'utilisateur.

Les applications que nous considèrons sont construites à partir d'un ensemble de composants logiciels : une interface graphique (si le terminal de l'utilisateur est capable de l'accueillir) et d'autres composants qui coopèrent pour assurer les traitements demandés par l'utilisateur.

Les applications à base de composants répondent bien à nos besoins puisque d'une part, elles constituent un bon moyen pour construire des applications modulaires et réutilisables et d'autre part, elles permettent l'exécution de chaque composant sur un site différent. Notre unité de déploiement sera donc le composant logiciel.

Pour les applications que nous considérons, un même type de composant peut avoir des implémentations différentes. Par exemple, un composant de type interface graphique diffère d'un type terminal à un autre pour une même application (l'interface graphique n'est pas la même pour un PDA ou un ordinateur portable ne serait-ce qu'à cause de la différence de la taille de l'écran).
Chaque composant sera placé dans un paquetage. Chaque paquetage contient une version donnée du composant dans une implémentation dédiée à un environnement d'exécution particulier. Les différents paquetages des composants, constituant les différentes applications, sont placés dans des serveurs de paquetages.

Les paquetages de composants des différentes applications sont généralement téléchargés au moment de la connexion de l'utilisateur et instanciés sur des serveurs que nous appellons des serveurs de composants ou serveurs d'exécution. Ces serveurs offrent un environnement favorable pour l'installation et l'exécution de l'application. Un des serveurs de composants privilégiés pour accueillir les composants d'une application est le terminal de l'utilisateur. Si ce dernier ne comporte pas les ressources nécessaires pour accueillir la totalité des composants de l'application, ces derniers seront dispersées sur les différents serveurs de composants selon leur charge et leur capacité.

Il existe des composants logiciels qui doivent être instanciés avant la demande d'accès de l'utilisateur à l'application c'est à dire dans la phase de préparation de l'infrastructure de déploiement. Ces composants sont préinstanciés pour les raisons suivantes :
- Si le paquetage du composant a une grande taille, son téléchargement prendra beaucoup de temps.
- Il existe des composants qui doivent toujours être placés sur des serveurs fixes pour des raisons de sécurité ou parce qu'ils utilisent des ressources fixes comme les bases de données.

4.1.1 Ressources et des étapes de déploiement

Pour assurer le déploiement des applications, une infrastructure de déploiement doit être préparée à l'avance et doit contenir :

1. Un ou plusieurs serveurs de déploiement.
2. Un client de déploiement au niveau du terminal utilisateur.
3. Un ou plusieurs serveurs de recherche sur propriétés (voir section **??**).
4. Un ou plusieurs serveurs de composants.

5. Un ou plusieurs serveurs de paquetages.

L'infrastructure peut contenir un système d'équilibrage de charge entre les serveurs de composants.
En plus de l'installation de ces différents serveurs, l'infrastructure doit initialement contenir Les composants préinstanciés.

Le processus de déploiement suit les étapes suivantes (voir figure 4.1) :

1. l'utilisateur choisit l'application qu'il veut utiliser en précisant quelques critèrres de choix.
2. Le client de déploiement envoie une requête vers le serveur de recherche pour chercher l'application demandée.
3. Le client de déploiement envoie une requête vers le serveur de recherche pour chercher un serveur de déploiement de proximité pour déployer l'application.
4. Le client de déploiement collecte des informations sur le contexte utilisateur, puis envoie une requête de déploiement vers le serveur de déploiement.
5. Le serveur de déploiement demande au service de recherche de rechercher les différents paquetages des composants constituants l'application demandée par l'utilisateur ainsi que les différents serveurs de composants capables d'instancier ces paquetages. Si l'application comporte des composants préinstanciés, le service de recherche ne retournera que les serveurs sur lesquels ils ont été instanciés.
6. Le serveur de déploiement reçoit la réponse du service de recherche et envoie des ordres de déploiement vers les serveurs de composants désignés par le service de recherche.
7. Les serveurs de composants recevant ces ordres vont télécharger les paquetages des serveurs de paquetages indiqués par le service de recherche et les instancier.
8. Une fois que tous les composants sont instanciés, le serveur de déploiement connectera les différents composants et donne la main à l'utilisateur qui peut, à partir de ce moment, utiliser l'application.

Dans ce qui suit, nous allons expliquer d'une manière détaillée le rôle de chaque ressource de l'infrastructure de déploiement et plus particulièrement le serveur de déploiement, coeur de notre travail.

4.1.2 Les serveurs de paquetages

Un serveur de paquetages, représente un espace de stockage de composants logiciels avant leurs instanciation.
Un même type de composants peut exister sous forme de plusieurs paquetages. Nous disposons d'un fichier pour chaque implémentation de composant ; chaque fichier (ou paquetage) contiendra une version du composant dans une implémentation dédiée à un type de machine ou environnement d'exécution particulier.

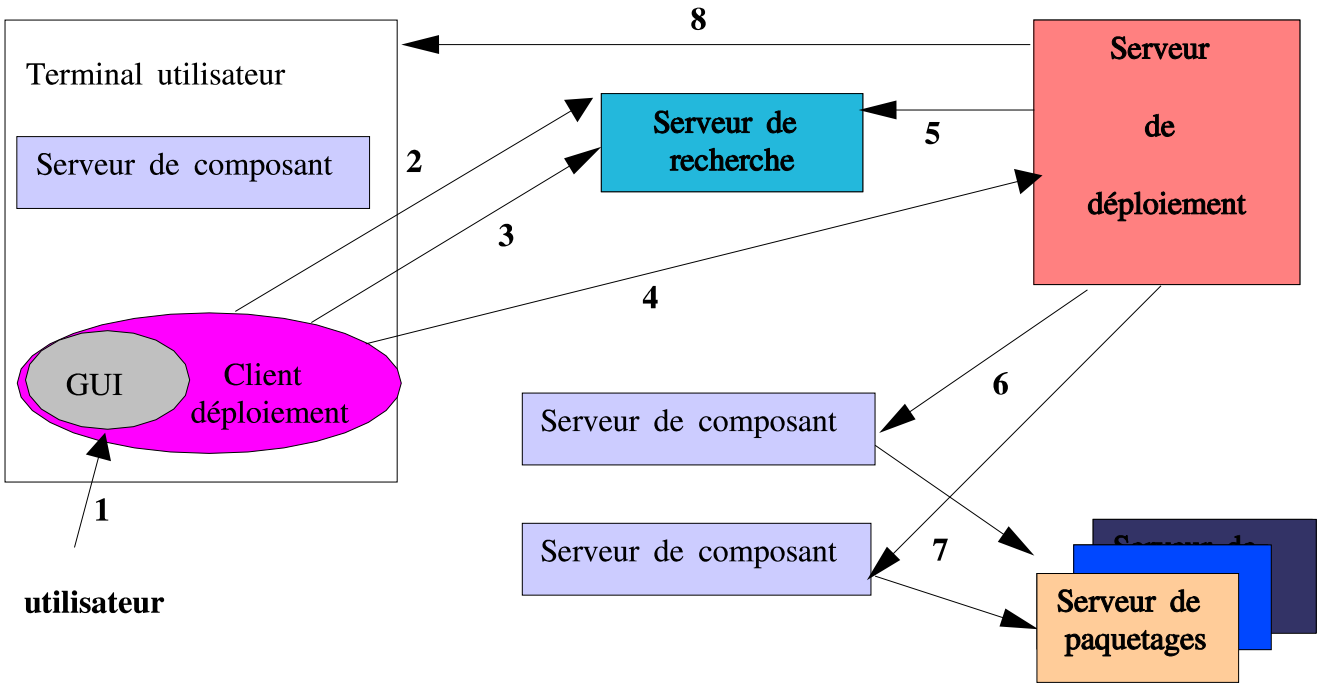


FIG. 4.1 – Infrastructure et étapes de déploiement

Chaque paquetage est mis à disposition sous la forme d'un fichier d'archive contenant : les composants logiciels, les scripts d'installation, la documentation, ...[Con00]

4.1.3 Les serveurs de composants (ou serveurs d'exécution)

Les serveurs de composants appelés aussi gestionnaires d'instances ou serveurs d'exécution, fournissent un environnement d'exécution pour accueillir les composants logiciels après leurs instanciation.
Suite à un ordre du serveur de déploiement, le serveur de composants, téléchargera le paquetage désigné par le service de recherche à partir du serveur de paquetages qui l'héberge et l'instanciera. Le téléchargement peut être fait par n'importe quel service de transfert de fichiers.
Le serveur principal d'exécution (de composants) d'une application est le terminal utilisateur. Dans le cas où le terminal est riche en ressources tel un PC, le terminal utilisateur, peut être le seul serveur d'exécution. Mais dans le cas d'un terminal pauvre en ressources tel qu'un PDA ou un téléphone portable, il faudra chercher d'autres serveurs d'exécution pour exécuter les parties de l'application qui ne peuvent pas s'exécuter sur le terminal [Con].
Le serveur d'exécution reçoit la référence d'un paquetage à télécharger et retourne la référence d'un composant instancié [PTB02].

4.1.4 Le serveur de déploiement

Le déploiement d'une application est assuré par un serveur de déploiement qu'on appelle aussi gestionnaire de déploiement. Notre infrastructure comporte plusieurs serveurs de déploiement. Néanmoins, une application esy déployée par un seul serveur de déploiement choisi par le serveur de recherche, selon des caractères de proximité, au moment où l'utilisateur accède au service.

Le rôle d'un serveur de déploiement consiste à superviser la recherche de composants préinstanciés, la recherche de paquetages et de serveurs de composants, l'instanciation et la connexion de tous les composants nécessaires pour faire fonctionner l'application et enfin le lancement de l'application au niveau de l'utilisateur final.

A chaque fois que le serveur de déploiement reçoit une liste de résultats de recherches de composants préinstanciés, de serveurs de composants, de déploiement ou de paquetages, il l'enregistre dans son journal pour une utilisation ultérieure, et choisit le premier élément de cette liste, puisque la liste retournée est triée par ordre des préférences.

Le serveur de déploiement inscrit toutes les instructions qu'il effectue dans un fichier journal. Le serveur de déploiement prend en compte les informations sur le contexte utilisateur pour déployer une application qui s'adapte à l'environnement utilisateur.
Pour assurer le déploiement des différentes applications, le serveur de déploiement utilise des descripteurs de déploiement qui servent essentiellement à donner des information d'une part sur l'architecture de l'application, à savoir les différents composants qui la constituent et leurs interconnexions et d'autre part des informations utiles pour le déploiement qui seront donnés sous forme d' attributs fonctionnels et non fonctionnels.
Dans le cadre de notre application, trois types de descripteurs seront utilisés par le serveur de déploiement lors du déploiement d'une application :

- Un descripteur de déploiement statique qui contiendra des informations sur l'architecture de l'application et des informations sur le déploiement de l'application qui sont invariables quelque soit le contexte d'exécution de l'utilisateur.
- Un descripteur de déploiement dynamique qui contient des informations de déploiement variables en fonction du contexte d'utilisation (il sera envoyé au serveur de déploiement).
- Un descripteur de déploiement concret qui décrit l'architecture et les propriétés de l'application qui a été finalement déployée (il sera retourné par le serveur de déploiement).

4.1.4.1 Le descripteur de déploiement statique

Les descripteurs de déploiement statiques se trouvent dans des fichiers de descripteurs de déploiement qui peuvent être placés sur n'importe quel serveur sur le réseau. Ces fichiers sont écrits à l'aide d'un langage de description d'architecture d'applications (ADL : Architecture Description Language).
Les descripteurs statiques contiennent les informations sur l'architecture de l'application sui-

vantes :

- nom de l'application,
- nom de chaque composant,
- l'interconnexion entre les composants.
  En plus des informations d'architecture, les descripteurs statiques contiennent des informations servant au déploiement de l'application.
- Localisation du composant après instanciation : il faut préciser si le composant une fois instancié, doit être placé sur un serveur de composant précis comme le terminal utilisateur, sur un serveur banalisé quelconque ou sur le même serveur qu'un autre composant. Nous pouvons par exemple préciser à ce niveau que le composant " interface graphique " doit être placé au niveau du terminal lors de l'instanciation.
- Le cycle de vie (ou life cycle) du composant : le cycle de vie d'un composant peut être de type entité, process ou session.
  Les composants session sont des composants avec un état volatile, et une identité qui n'est pas persistante. La durée de vie d'un tel composant est une interaction de la part du client. Les composants process sont des composants avec un état persistant géré par le composant ou par le container. Le fait que cet état soit persistant n'est pas visible pour le client. L'identité du composant est persistante, et gérée par le composant. Elle est rendue visible au client via des opérations définies par l'utilisateur. Les composants entités sont des composants semblables aux composants processus pour la persistance. Néanmoins, cette persistance est visible pour le client.
  Par exemple, l'interface graphique d'une application est un composant session. Puisque si jamais l'utilisateur ferme l'application, nous n'avons pas besoin de sauvegarder l'état de l'interface graphique pour le restituer la prochaine fois. Un compte bancaire est un composant entité, puisqu'il doit conserver son état pour chaque prochaine connexion. Un agent qui fait un virement entre deux comptes est un composant process puisque dès qu'il fait le virement demandé par l'utilisateur il disparait tout seul. Les composants session et process sont généralement sous forme de paquetages. par contre les composants entité doivent être préinstanciés.
  Le processus de déploiement diffère pour composants paquetages et préinstanciés. Pour les composants préinstanciés, le déploiement consiste à les rechercher quelque part sur le réseau et pour les composants paquetages, il s'agit de trouver le paquetage et par la suite le serveur de composants capable d'instancier ce paquetage.

4.1.4.2 Descripteur de déploiement dynamique

Le descripteur de déploiement dynamique est celui qu'utilise le serveur de déploiement pour faire le déploiement de l'application demandée par l'utilisateur. Il contient donc le descripteur statique de l'application auquel il rajoute des informations variables en fonction du contexte d'exécution et les préférences utilisateur et qui sont :

- le contexte de connexion utilisateur : Ce sont les propriétés du terminal utilisateur (ceux du serveur de composants se trouvant au niveau du terminal) : la vitesse d'horloge du processeur, la mémoire, le système d'exploitation utilisé, la charge du processeur, les langages de

programmation supportés, la taille de l'écran, ...
Ce genre d'informations peut inclure la localisation géographique de l'utilisateur qui peut être utile pour certains services comme dans le cas où l'utilisateur voudrait chercher des restaurants dans la zone où il se trouve ou veut avoir des informations sur la météo.
- les préférences et les contraintes utilisateur.
- un ensemble de services de recherche triés par ordre de proximité dont les références sont fournies par le client de déploiement.

4.1.4.3 Descripteur de déploiement concret

Le descripteur de déploiement dynamique, va être utilisé par le serveur de déploiement pour construire des requêtes de recherche vers le serveur de recherche dans le but de choisir dynamiquement les différents composants qui constitueront l'application ainsi que les serveurs sur lesquels ils vont s'exécuter. Ce choix se fera en fonction des propriétés contenues dans ce descripteur. Une fois que le déploiement de l'application sera terminé, le serveur de déploiement construira un descripteur de déploiement concret qui décrit l'architecture et les attributs de l'application qui a finalement été déployée.
Ce descripteur sert essentiellement pour le repliement (voir section 4.2) et pour des connexions ultérieures de l'utilisateur. Il contient ces informations.
- Le nom de l'application qui a été finalement déployée.
- Le nom du serveur de déploiement qui a déployé l'application.
- Le nom de chaque composant de l'application déployée.
- Le nom du serveur de composant et la référence sur ce serveur de chaque composant instancié.
- Le nom du serveur de paquetages et la référence sur ce serveur de chaque paquetage de composant téléchargé.
- l'interconnexion entre les composants.

Une fois le déploiement de l'application est terminé, le serveur de déploiement enregistre le descripteur de déploiement concret sur son journal et redonne la main à l'utilisateur en envoyant, vers le client de déploiement, le descripteur concret qu'il a construit avec les différentes listes de recherche (liste des serveurs de déploiement trouvés, liste des applications trouvées, liste des serveurs de composants trouvés et les paquetages de composants trouvés) récupérées à partir du serveur de recherche pour qu'il puisse les mettre dans le cache utilisateur qui s'en servira lors de connexions ultérieures (voir section 4.3.6).

4.1.5 Le service de recherche

Ce service est aussi important que le gestionnaire de déploiement lui-même. Il assure la partie intelligente et dynamique du déploiement en trouvant la meilleure adéquation entre le contexte du déploiement et les ressources disponibles au moment du déploiement. Notre infrastructure de déploiement comporte plusieurs services de recherches en mirroring. Chaque serveur de recherche maintient un dépositaire contenant toutes les ressources disponibles : serveurs de déploiement, applications, paquetages de composants, serveurs de composants et composants préinstanciés.

Chaque fournisseur de service gère un ensemble de ces serveurs dont il maintient les noms, les adresses et la localisation géographique dans un fichier placé sur un serveur quelconque du réseau. Ce fichier sera automatiquement rapatrié par le client de déploiement à son démarrage (voir section 4.1.6).

4.1.5.1 Exportation de services

Chaque serveur de déploiement, application, paquetage de composant, serveur de composants ou composant préinstancié nouvellement installé est enregistré auprès du service de recherche.

- Une application est décrite au niveau du dépositaire par un nom, des fonctionnalités, un ensemble de propriétés et l'emplacement d'un fichier de description de déploiement pour cette application.
- Un serveur de déploiement est décrit au niveau du dépositaire par un nom, une localisation logique (adresse IP) et une localisation physique (zone, latitude, longitude).
- Un paquetage de composants est décrit au niveau du dépositaire par le type de composant contenu dans le paquetage, une URL à partir de laquelle le paquetage peut être téléchargé et un ensemble d'informations sur l'environnement d'exécution. L'environnement d'exécution traduit des informations sur les ressources minimales nécessaires pour instancier le paquetage comme le type d'hôte et le système d'exploitation sur lesquels le composant peut s'exécuter, la vitesse minimale du processeur, la mémoire minimale, ...
- Un serveur de composants est décrit au niveau du dépositaire par un type d'hôte (serveur banalisé, PC portable, PDA, téléphone mobile,), des informations sur ses capacités et ressources (mémoire, processeur, ...), une localisation logique (adresse IP), une localisation physique (zone, latitude, longitude,), des informations sur sa charge, les logiciels supportés, ...
- Un composant préinstancié est décrit au niveau du dépositaire par un type de composant et une URL du serveur de composants sur lequel il est instancié.

4.1.5.2 Importation de services

Pour déployer une application, quatre types de recherches sont effectués :

Importation d'applications : La recherche des applications se fait à partir du type d'application demandé et des préférences utilisateur.

Importation de serveurs de déploiement : La recherche d'un serveur se base sur des critères de proximité entre lui et le client de déploiement.
La notion de proximité est difficile à exprimer : elle peut par exemple être réduite à une notion de rapidité d'accès ou à une distance minimale entre le serveur de déploiement et le client de déploiement donnée par les latitudes et les longitudes respectives.

Importation de paquetages de composants : La recherche de paquetages se base sur le type de composant souhaité et un ensemble de propriétés comportementales ou de personnalisation. Généralement, la recherche de paquetages est réalisée avant la recherche d'un serveur de composants où va être instancié le paquetage. Mais dans certains cas, comme

pour l'interface graphique, nous avons un serveur de composants (qui est dans le cas de notre exemple le terminal utilisateur) et nous devons chercher un paquetage (qui est dans ce cas l'interface graphique utilisateur) pouvant s'exécuter sur le serveur de composants. Dans ce cas, il faut rajouter aux propriétés statiques d'un composant que l'on vient de citer, des propriétés de recherche dynamiques comme les contraintes techniques d'exécution.
En effet, les sites d'exécution étant découverts lors du déploiement, leurs contraintes techniques ne peuvent être connues avant. Le paquetage sera donc recherché en fonction de l'implantation du type de composant qu'il contient qui doit être en phase avec les sites d'exécution.

Importation de serveurs de composants : Une fois les paquetages définis, il est nécessaire de rechercher des sites d'exécution pour les composants qu'ils contiennent. Ces sites doivent avoir les ressources matérielles et logicielles nécessaires pour accueillir ces composants.
Il est important de prendre en compte la charge dynamique des serveurs de composants et de trouver les moins chargés. En plus, il peut être intéressant de penser à améliorer la performance et garantir une certaine simplicité en cherchant des serveurs de composants proches du serveur de déploiement.

4.1.6 Le Client de déploiement

Nous voulons que l'utilisateur puisse accéder au service et que celui-ci soit choisi et configuré à l'aide de ses paramètres personnels et que la diversité des configurations d'applications soit gérée sans son intervention.

Pour pouvoir déployer et accéder aux différentes applications, un utilisateur doit avoir un client de déploiement installé sur son terminal. Cette entité logicielle permet essentiellement de choisir une application ou un service et d'envoyer une requête de déploiement vers le serveur de déploiement.
Le client de déploiement inclut une interface graphique permettant à un utilisateur de choisir un service parmi les différents services offerts et préciser quelques propriétés de ce service. Ces propriétés peuvent être exprimées sous forme de contraintes et préférences.
L'interface graphique du client de déploiement fournit une liste de noms de propriétés avec des champs à remplir là où l'utilisateur peut par exemple préciser la langue qu'il préfère, la monnaie qu'il veut utiliser, s'il cherche un service payant ou pas et si le service est payant, il pourra préciser la marge des prix qu'il préfère,...
Cette liste de contraintes et de préférences contient des valeurs par défaut au cas où l'utilisateur ne voudrait préciser aucune préférence. Par ailleurs, cette liste porte sur des propriétés générales qui peuvent être communes à tout type d'application mais l'utilisateur peut ajouter d'une manière dynamique ses propres contraintes et préférences à la liste.

Une fois que l'utilisateur aura saisi ces informations, il n'aura plus à intervenir, c'est le client de déploiement qui se chargera d'assurer le déploiement de l'application et assurera la transparence du déploiement au niveau de l'utilisateur.

Dès son démarrage, le client de déploiement rapatrie le fichier contenant la liste des serveurs de recherche propres à son fournisseur de service dont il connait l'adresse à l'avance. Une foie rapatriée, cette liste est triée selon un ordre de proximité des serveurs de recherche par rapport à la localisation du terminal utilisateur. Le client de déploiement fait ce trie en récupérant des informations de localisation à partir d'un localisateur installé sur le terminal utilisateur. Le client de déploiement choisira le premier serveur de recherche de la liste. Le rôle du client de déploiement consiste à envoyer trois types de requêtes à partir du terminal utilisateur. La première requête permet la recherche d'un service ou d'une application bien particulière que l'utilisateur demande, la deuxième permet la recherche d'un serveur de déploiement de proximité et la troisième, sert à déployer l'application choisie à partir du serveur de déploiement désigné.

4.1.6.1 Construction des requêtes de recherche par le client de déploiement

Le client de déploiement construit la requête de recherche avec les contraintes et les préférences qu'il faut à partir de ce qu'a saisi l'utilisateur dans les différents champs et l'envoie au service de recherche qui fait sa recherche et renvoie l'ensemble des résultats au client de déploiement. Le client de déploiement choisit la première application de la liste des résultats puisqu'il considère que c'est celle qui répond le plus aux préférences de l'utilisateur et essaie par la suite de construire la deuxième requête vers le serveur de recherche à partir des informations sur le débit supporté au niveau du terminal utilisateur ou des information de localisation géographique de l'utilisateur pour chercher un serveur de déploiement de proximité. Enfin, le client de déploiement envoie la troisième requête vers le serveur de déploiement pour déployer l'application.

4.1.6.2 Construction d'une requête de déploiement par le client de déploiement

La requête de déploiement consiste en un descripteur de déploiement dynamique (adapté au contexte utilisateur, voir section 4.1.4.2). Ce descripteur de déploiement sera construit à partir des informations suivantes :

- le descripteur de déploiement statique sera récupéré à partir d'un fichier de description de déploiement dont l'identification et l'emplacemant ont été indiqués dans l'une des propriétés de l'aplication indiquée par le serveur de recherche.
- Les informations sur la localisation géographique de l'utilisateur seront récupérées à partir d'une entité capable de déterminer la localisation de l'utilisateur (zone, latitude, longitude) que nous appelerons localisateur.
- les informations sur le contexte d'exécution de l'utilisateur seront récupérées à partir du serveur de composants du terminal utilisateur.
- les informations sur les préférences utilisateur seront extraites à partir de ce qu'a saisi l'utilisatur dans le formulaire qui lui a été fourni par l'interface graphique. Le client de déploiement ne récupèrera que les préférences que l'utilisateur a précisées et qui sont utiles à ce niveau (pour le déploiement). L'utilisateur peut saisir des contraintes et des préférences inutiles pour le déploiement, mais le client de déploiement peut extraire les propriétés utiles grâce au type de service de l'application à déployer enregistré dans le service de recherche.

En effet, le type de service contient les noms et les types de propriétés supportées par le service. Le client de déploiement ne récupérera donc que le contenu des champs du formulaire remplies par l'utilisateurs se rapportant à ces propriétés.

Une fois construite, la requête de déploiement sera envoyée vers le serveur de déploiement qui s'en servira pour déployer l'application demandée en utilisant les informations de contexte se trouvant dans cette requête.

4.2 Repliement ou terminaison du déploiement

Une fois que l'utilisateur a terminé son travail et qu'il ferme l'application, le serveur de déploiement, doit effacer les instances de composants propres aux applications de l'utilisateur qui ont été placées sur les différents serveurs de composants en effectuant des opérations inverses au déploiement : c'est le repliement.

L'événement de fermeture de l'application sera capturé par le client de déploiement qui enverra une requête de terminaison du déploiement vers le serveur de déploiement.

La requête de terminaison du déploiement contient le descripteur de déploiement concret que le client de déploiement a reçu à la fin de la phase de déploiement. Ce descripteur, permet au serveur de déploiement d'identifier, les serveurs de composants sur lesquels ont été instanciés des paquetages propres à l'application utilisateur, pour qu'il puisse envoyer des ordres vers ces serveurs de composants pour effacer ces instances.

L'utilisateur peut aussi annuler sa demande et choisir de se déconnecter avant la terminaison du déploiement de l'application sans utiliser cette dernière. Dans ce cas, cet ordre d'annulation sera capturé par le client de déploiement et envoyé vers le serveur de déploiement.

Une annulation est traitée, au niveau du serveur de déploiement, comme une déconnexion involontaire de l'utilisateur (voir section 4.3.5)

4.3 Sémantique du déploiement en cas d'échec

Cinq classes d'échec peuvent survenir au moment du déploiement de l'application :

1. déconnexion involontaire de l'utilisateur : ce genre de déconnexion peut être causé par une coupure de courant, des problèmes de réseau, un épuisement de charge de batterie, ...
2. panne du serveur de déploiement,
3. panne de l'un des serveurs de composants (à part celui du terminal utilisateur),
4. panne au niveau du serveur de recherche,
5. panne du serveur de paquetages.

4.3.1 Panne du serveur de déploiement

Le serveur de déploiement, désigné par le service de recherche pour gérer le déploiement d'une application peut tomber en panne pendant une requête du client de déploiement. Si cela

arrive, nous aurons deux cas possibles :

- Le serveur de déploiement dépasse un certain délai en panne, au bout duquel, il ne redonne pas la main au client de déploiement en lui retournant un descripteur concret, le client de déploiement suppose que ce serveur est en panne et envoie une deuxième requête de déploiement vers le serveur de déploiement suivant dans la liste qu'il a reçue à la phase de recherche. Si jamais, dans le pire des cas, il parcours toute la liste des serveurs de déploiement et il trouve que tous les serveurs sont en panne, il envoie un message d'erreur à l'utilisateur lui indiquant que l'application ne peut être déployée.
  Lorsqu'un serveur de déploiement se réveille d'une panne et trouve qu'il a dépassé le temps d'attente du client de déploiement, il saura que la requête de déploiement qu'il traitait a été redirigée et fera un repliement à partir de son journal.

- Le serveur de déploiement repart avant de dépasser le délai d'attente du client de déploiement. Dans ce cas, nous disposons de trois cas possibles selon les moments où le serveur de déploiement est tombé en panne :

  - Le serveur de déploiement est tombé en panne avant d'envoyer la requête de recherche vers le service de recherche ou avant d'envoyer les ordres de déploiement vers les serveurs de composants. Lorsque le serveur de déploiement reprend, il trouvera dans son journal ce qui a été fait précédemment. Il vérifie d'abord s'il a dépassé le temps d'attente du client de déploiement. S'il ne l'a pas dépassé, il vérifie si l'utilisateur ne s'est pas déconnecté involontairement ou s'il n'a pas annulé sa demande pendant sa panne en envoyant un signal de vérification de présence vers le client de déploiement. Si l'utilisateur est encore là, en attente de réponse, le serveur de déploiement poursuivra le déploiement en envoyant, selon le moment où il est tombé en panne, la requête de recherche ou les ordres de déploiement vers le serveur de déploiement.
    Si le serveur de déploiement ne reçoit pas une réponse du client de déploiement ou s'il a dépassé son temps d'attente, il ne termine pas le déploiement de l'application et fait un repliement à partir des informations qu'il récupère à partir de son journal.

  - Le serveur de déploiement envoie les ordres de déploiement vers les serveurs de composants ou une requête de recherche vers le service de recherche, tombe en panne et rate leurs réponses. Lorsqu'il reprend, il trouvera dans son journal qu'il a envoyé des ordres de déploiement ou une requête de recherche, sans recevoir de réponses. Il commence donc par vérifier la présence de l'utilisateur et le temps d'attente du client de déploiement. Si tout va bien, il renvoie les mêmes ordres de déploiement ou requêtes de recherche trouvés dans le journal vers les mêmes destinataires. Sinon (dans le cas ou il a dépassé le temps d'attente du client de déploiement ou l'utilisateur a annulé sa demande), s'il n'a envoyé qu'une requête de recherche, il ne poursuit pas le déploiement. Mais s'il a envoyé des ordres de déploiement vers des serveurs de composants, il fera un repliement.

– Le serveur de déploiement tombe en panne une fois que le déploiement de l'application est terminé mais juste avant de donner la main à l'utilisateur et de lui envoyer le descripteur de déploiement concret. Dans ce cas, lorsqu'il reprend, il se repèrera avec son journal et construira le descripteur de déploiement concret. Si le temps d'attente du client de déploiement a été dépassé ou l'utilisateur a annulé sa demande pendant le moment où le serveur de déploiement est en panne, ce descripteur concret sera utilisé pour annuler le déploiement déjà fait. Sinon, il sera envoyé vers l'utilisateur.

4.3.2 Panne de l'un des serveurs de composants ou du serveur de recherche

Lorsque le serveur de déploiement envoie des ordres de déploiement vers les serveurs de composants, il se met en attente de leurs réponses qui lui indiqueront que l'instanciation des différents composants s'est bien passée. Si un ou plusieurs serveurs de composants n'ont pas donné de réponse, après un certain délai d'attente, le serveur de déploiement reprend la liste de serveurs de composants reçue précédemment du service de recherche, choisit le serveur de composants qui suit le serveur qui n'a pas répondu dans la liste et renvoie l'ordre de déploiement vers le nouveau serveur de composant choisi. De la même manière, le serveur de déploiement envoie une requête de recherche vers le service de recherche. Si ce dernier n'a pas répondu, le serveur de déploiement redirige sa requête vers un autre service de recherche.
Si aucun serveur de recherche ou aucun serveur de composants ne sont trouvés, un message d'erreur sera envoyée vers l'utilisateur et un repliement sera fait.

4.3.3 Panne du serveur de recherche

4.3.4 Panne du serveur de paquetages

Chaque serveur de composants envoie une requête de téléchargement de fichier vers un serveur de paquetages. Si au bout d'un certain délai, le fichier n'a pas pu être téléchargé, le serveur de composants enverra une exception vers le serveur de déploiement lui signalant que le serveur de paquetages est en panne.
Le serveur de déploiement renvoie un autre ordre de déploiement vers le serveur de composants ayant signalé l'erreur avec un nouveau serveur de paquetages. Le nouveau serveur de paquetages choisi sera celui qui suit dans la liste de résultats de recherche le serveur de paquetages qui a été proposé précédemment. S'il n'y a plus de serveurs de paquetages dans la liste, un message d'erreur sera envoyé vers l'utilisateur.
Nous pouvons envisager cette solution, si nous voulons que toute la gestion des erreurs soit centralisée au niveau du serveur de déploiement. Mais la gestion des pannes des serveurs de paquetages sera plus simple si le serveur de déploiement envoie la liste de recherche des différents paquetages et leurs serveurs vers le serveur de composants au même moment où il lui envoie la requête d'instanciation pour qu'il s'en serve directement, sans passer par le serveur de déploiement, si jamais il y a eu une panne du serveur de paquetages.

4.3.5 Déconnexion de l'utilisateur

La déconnexion de l'utilisateur peut subvenir à n'importe quel moment de la phase de déploiement. Si la déconnexion survient avant que le serveur de déploiement n'envoie des ordres de déploiement, il s'agira juste de ne pas poursuivre le déploiement.
Sinon, le déploiement se passera normalement, le serveur de déploiement, construira un descripteur de déploiement concret comme prévu mais l'utilisera plutôt pour pour faire un repliement lorsqu'il se rendra compte que le client de déploiement ne peut pas recevoir ce descripteur.

L'utilisateur peut aussi se déconnecter involontairement après la terminaison du déploiement de l'application mais pendant l'utilisation de l'application. Dans ce cas, le problème qui se pose, c'est que la fermeture de l'application ne sera pas détectée et le repliement ne peut être réalisé. Comme solution à ce genre de problèmes, nous proposons que le serveur de déploiement se mette à vérifier la présence de l'utilisateur en lui envoyant des signaux à intervalles de temps réguliers à partir du moment où il a terminé le déploiement et qu'il a donné la main à l'utilisateur. Si le client de déploiement ne donne pas de réponse à ces signaux, il fait un repliement à partir du descripteur concret sauvegardé dans le journal.

4.3.6 Réutilisation de l'application et caches utilisateur

Lorsqu'un utilisateur se reconnecte à une même application à partir d'un même terminal, il doit trouver sur son cache les informations nécessaires permettant d'éviter de recommencer la totalité du déploiement. Pour cela, nous disposons de trois types de cache au niveau de l'utilisateur :

- Un cache descripteur : ce cache sauvegarde le descripteur de déploiement concret d'une application
- Un cache de préférences utilisateur : ce cache sauvegarde les contraintes et les préférences utilisateur.
- Un cache de recherches : ce cache sauvegarde les différentes listes de recherches faites par le serveur de recherche : la liste des serveurs de paquetages, la liste des serveurs de composants et la liste de serveurs de déploiement.

En demandant l'utilisation du même type d'application, l'utilisateur doit retrouver les préférences qu'il a saisies la dernière fois et qui seront récupérées à partir du cache de préférences utilisateur. Ces préférences seront automatiquement affichées au niveau de l'interface graphique à la place des préférences par défaut. Si l'utilisateur ne change aucune de ces préférences, le client de déploiement récupérera le descripteur concret se trouvant sur le cache descripteur et l'enverra au serveur de déploiement à la place du descripteur dynamique de déploiement, qui l'utilisera directement pour envoyer des ordres de déploiement (d'instanciation) vers les serveurs de composants sans passer par les différentes étapes de recherche. Si jamais il y a des pannes dans les serveurs indiqués par le descripteur concret du cache, ce qui est très possible puisque ces serveurs ont été attribués il y a longtemps, les listes de recherche seront récupérées à partir du cache de recherche pour rediriger les différentes requêtes vers des serveurs se trouvant sur ces

listes.

Chapitre 5

Conception du serveur de déploiement

Dans ce chapitre, nous allons proposer un modèle UML pour le déploiement. Nous essayerons de focaliser sur la conception détaillée du serveur de déploiement en définissant ses interfaces et les descripteurs qu'il utilise. Nous essayerons de donner les diagrammes de classes, d'états et d'évènements pour le modèle de déploiement que nous proposons Par la suite nous essayerons de définir en IDL les interfaces du serveur et du client de déploiement.

5.1 Modélisation UML de l'infrastruture de déploiement

La figure ... montre le diagramme de classe pour notre infrastructure de déploiement. Ce modèle comporte 17 classes : la classe DeployementServer modélise un serveur de déploiement, la classe componentServer modélise un serveur de composants, la classe PackageServer modélise un serveur de paquetages, la classe ResearchServer modélise un serveur de recherche qui utilise le trader Corba, la classe Deployment modélise l'entité déployant une application sur le serveur de déploiement, les classes DynamicDescriptor, ConcretDescriptor et StaticDescriptor modélisent respectivement les descripteurs dynamiques, concrets et statiques, les classes componentPackage et instanciatedComponent modélisent respectivement les composants sous forme de paquetages et les composants instanciés, la classe DeploymentClient modélise un client de déploiement, la classe TerminalUser modélise le terminal utilisateur, la classe User modélise un utilisateur, la classe localiser modélise un localisateur, la classe Journal modélise un journal utlisé par un serveur de déploiement et enfin, la classe Application modélise une application que l'utilisateur veut déployer. Nous allons, dans ce qui suit expliquer le rôle que joue chaque classe :

- Une classe serveur de recherche est caractérisée par le nom du serveur (donné sous forme d'URL), son adresse IP et sa localisation géographique (zone, latitude et longitude). Elle comporte quatre méthodes de recherche et quatre méthodes pour effacer les différents services enregistrés au niveau du dépositaire.
  Cette classe présente une couche au dessus du trader Corba qui utilise directement ses interfaces.
  Chacune des classes DeployementServer, componentServer, PackageServer et Application ont une relation d'association avec la classe ResearchServer leur permettant de s'enregis-

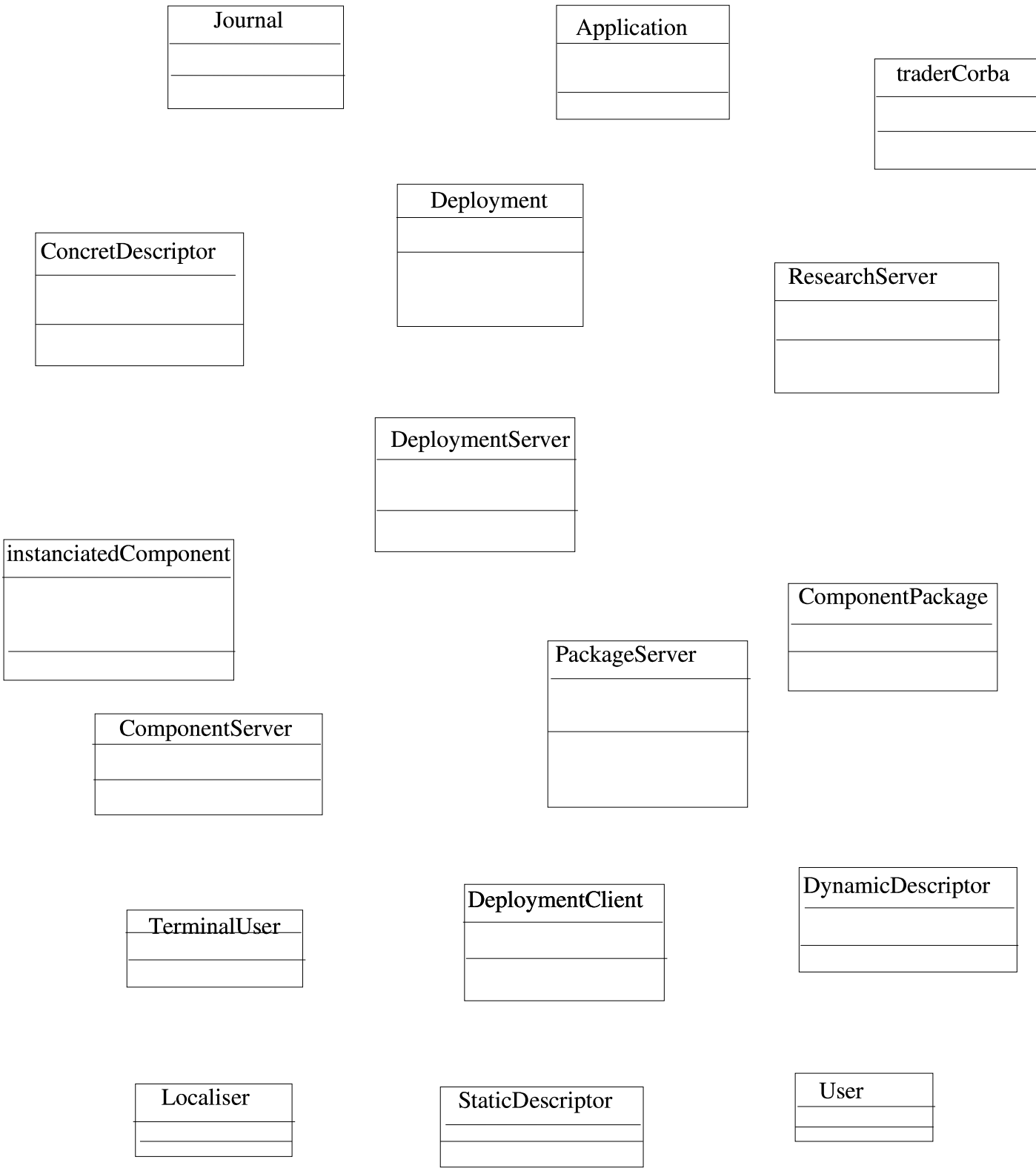


FIG. 5.1 – Modèle UML du déploiement

trer au niveau du serveur de déploiement. Chaque objet de ces classes doit s'enregistrer sur aumoins un serveur de recherche. Toutes ces associations sont donc multiples de 1 à n.

– Un serveur de composants est décrit par ses caractéristiques physiques(processeur, mémoire, système d'exploitation, ...), son nom (donné sous forme d'URL), son adresse IP et sa localisation géographique. Un serveur de composant héberge des composants instanciés. Il a donc une relation d'agrégation avec la classe instanciatedComponent.
– Un serveur de paquetages est décrit par son nom (donné sous forme d'URL) et son adresse IP.

  Les deux classes PackageServer et ComponentServer sont liés par une relation d'association de 1 à n permettant au serveur de composants de télécharger les paquetages de composants à partir des serveurs de paquetage.
– Les trois classes Entity, Process et Session (voir section 4.1.4.1) héritent de la classe InstanciatedComponent. La classe Entity modélisent les composants préinstanciés, c'est pour cette raison que cette classe comporte une associaion avec le serveur de recherche lui permettant de s'enregistrer dans le dépositaire.
– Un composant sous forme de paquetage (ComponentPackage) est décrit par une URL, le système d'exploitation sur lequel il peut s'exécuter, le type et la fréquence du processeur minimale ainsi que la taille de mémoire minimale qui lui sont nécessaires pour pouvoir s'exécuter.
– Une application utilisateur est caractérisée par un nom, des fonctionnalités et un fichier contenant son decripteur statique.
– Le terminal utilisateur peut avoir au plus un localisateur et peut contenir au plus un client de déploiement.
– Le client de déploiement a une seule méthode ListExtract permettant de choisir la première application de la liste de recherche retournée par le serveur de recherche.

  Le client de déploiement ne peut à un moment donné envoyer une requête de recherche que vers un seul serveur de recherche.Il a donc une association unaire avec le serveur de recherche. Un client de déploiement a une association avec le descripteur statique pour pouvoir en extraire des informations sur l'architecture des applications à déployer ainsi que des informations statiques sur le déploiement, une association avec le localisateur du terminal utilisateur pour pouvoir en extraire des informatins sur la localisation géographique de l'utilisateur et une association avec l'utilisateur pour extraire les contraintes et les préférences de l'utilisateur.

  Le client de déploiement a aussi une relation d'association avec la classe DynamicDescriptor lui permettant de créer un descripteur dynamique.

– Un descripteur de déploiement dynamique est décrit par
– Un descripteur de déploiement statique est décrit par
– Un descripteur de déploiement concret est décrit par

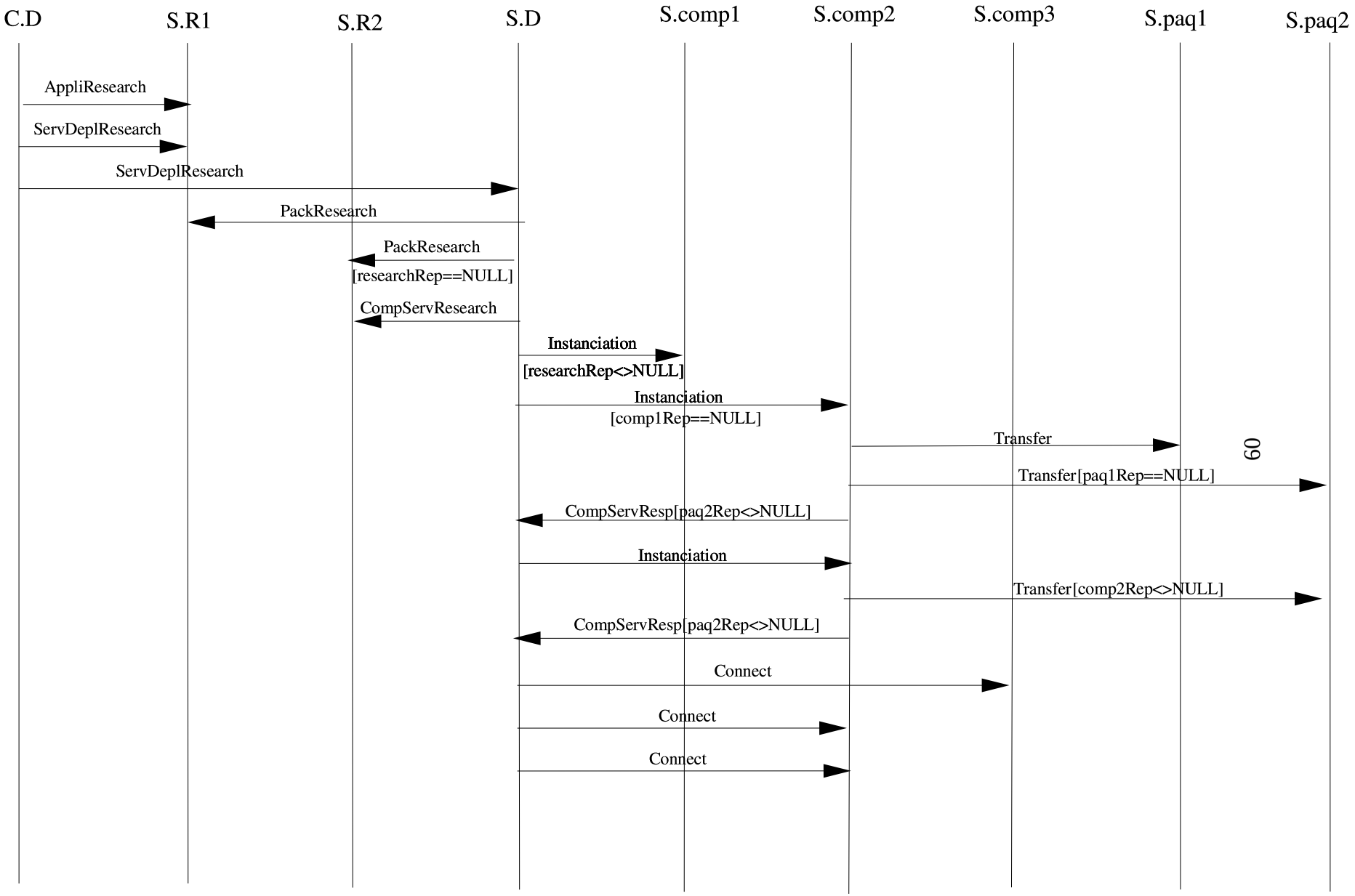


FIG. 5.2 – Diagramme d'évènements

Chapitre 6

Conclusion

Dans une première partie, nous avons présenté le contexte et ses problèmes. Puis nous avons introduit une courte présentation de CORBA base de nos solutions au problème. La partie axée sur l'étude de la littérature nous a permis d'éclaircir nos idées, nos solutions et de souligner les besoins en QoS pour les applications multimédia. Enfin nous avons décrit l'architecture Extended CORBA, en y ajoutant une proposition de protocoles pour les Ressources Managers. Notre approche a été de percevoir les nombreuses interrogations liées aux systèmes distribués, de tenter de trouver des solutions ou dans certains cas de proposer des idées nouvelles. Nous n'avons pas pu réaliser une application concrète sur Extended CORBA du fait du manque de temps. Le fruit de ces recherches est soumis sous forme d'article une conférence IEEE à Pittsburg aux Etat Unis.

Le domaine des systèmes répartis est en pleine évolution. L'internet sature alors que les utilisateurs qui sont de plus en plus nombreux commencent tout juste à l'utiliser. Les applications se veulent ouvertes vers le réseau des réseaux qui n'a déjà plus d'avenir sous la forme que l'on connaît. Son remplaçant est en test dans différents laboratoires. L'avenir risque de se trouver dans les Network Computers qui font beaucoup parler d'eux ces temps-ci. La réalisation d'une architecture d'une telle envergure a permis d'approcher et surtout de côtoyer les nombreux problèmes inhérents aux machines et aux réseaux actuels. Nous avons pu ainsi réaliser la difficulté liée à penser client/serveur, à prendre en compte les problèmes réseaux, systèmes, logiciels et même matériel. Il est évident que résoudre tout ceci n'est pas simple et l'avenir en informatique évolue trop rapidement pour miser sans erreur sur un quelconque produit. Les nombreuses questions soulevées durant ce stage méritent de trouver des réponses dans le cadre d'une ou plusieurs thèses.

Bibliographie

[BBABR96] L. Bellissard, S. Ben Atallah, F. Boyer, and M. Riveill. Distributed Application Configuration. *Proc. of the 16th IEEE International Conference on Distributed Computing Systems (ICDCS'96)*, (Hong Kong), April 1996.

[BR00] E. Bruneton and M. Riveill. JavaPod : une plate-forme à composants adaptable et extensible. *Rapport de recherche INRIA Nř3850*, Janvier 2000.

[BWD+93] M. Barbacci, C. Weinstock, D. Doubleday, M. Gardener, and R. Lichota. Durra : A Structure Description Language for Developping Distributed Applications. *IEEE Software Engeneering Journal*, (pp.83-94), Mars 1993.

[com] Composants de l'OMG. *http ://www.omg.org/library/schedule/CORBA_Component_Model_RFI*

[com99] Omg : Corba components : Joint revised submission. Object Management Group, Août 1999.

[Con] Configuration et Exécution de Services pour les Usagers Mobiles des Réseaux Etendus (CESURE). *Rapport final*.

[Con00] Configuration et Exécution de Services pour les Usagers Mobiles des Réseaux Etendus (CESURE). *Spécifications de l'infrastructure système*, 2000.

[EJB] Enterprise Java Beans : Server Component Model for Java. *http ://java.sun.com/products/ejb/white_paper.html*.

[GGM97] J.M. Geib, C. Gransart, and P. Merle. *Corba : Des concepts à la pratique*. Inter-Editions, 1997.

[HV99] M. Henning and S. Vinoski. *Advanced CORBA programming with C++*. Addison Wesley, 1999.

[Jav] Components for the Java Platform. *http ://java.sun.com/docs/books/tutorial/javabeans/index.htm*

[JIN01a] jini architecture specification. Sun Microsystems, Décembre 2001.

[JIN01b] Jini lookup service. Sun Microsystems, Décembre 2001.

[KM98] J. Kramer and J. Magee. Analysing Dynamic Change in Software Architecture : A case of study. *Proc. Of the 4th Intn'l Conference on Configurable Distributed Systems (ICCDS'98), IEEE, Annapolis MD, USA, pp.91-100*, May 1998.

[LMMG01] S. Leblanc, R. Marvie, P. MERLE, and J.M. Geib. TORBA :contrats de curtage pour CORBA. *Calculateurs parallèles*, 2001.

[MMG99] R. Marvie, P. Merle, and J. Geib. "Des objets aux composants CORBA, première étude et expérimentation avec CorbaScript". *In Proceedings of the 12th International Conference on Software and Systems Engineering and their Applications (ICSSEA'99), Paris, France,*, Décembre 1999.

[MMG00] R. Marvie, P. Merle, and J. Geib. Towards a Dynamic CORBA Component Platform. *In Proceedings of the 2nd International Symposium on Distributed Object Applications (DOA'2000), Antwerp, Belgium,*, Septembre 2000.

[OMG00] Trading Object Service Specification. OMG documents formal/00-06-27, May 2000.

[PTB02] E. Putrycz, C. Taconet, and G. Bernard. Smart Deployment Infrastructure for mobile applications. *Soumis à Mobicom*, 2002.

[Rog97] D. Rogerson. Inside COM. *Microsoft Programming Series, Microsoft Press,*, 1997.

[Sal01] Salutation Architecture Specification. The salutation consortium http ://www.salutation.org, June, 2001.

[SDK+95] M. Shaw, R. DeLine, D.V. Klein, T.L. Ross, and G. Toung, D.M.and Zelesnik. Abstraction for Software Architecture and Tools to Support Them. *IEEE Trans. Software Engineering, vol.SE-21(N.4), pp. 314-335*, Avril 1995.

[SOA00] Simple Object Access Protocol (SOAP) 1.1. W3C Note, http ://www.w3.org/TR/SOAP/, 08 May 2000.

[udda] site opérateur IBM pour UDDI. *http ://www-3.ibm.com/services/uddi/testregistry/inquiryapi*.

[uddb] Site opérateur Microsoft pour UDDI. *http ://uddi.microsoft.com*.

[udd01] UDDI Version 2.0 API Specification . UDDI Open Draft Specification http ://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf, 8 June 2001.

[XML00] XML 1.0 (Second Edition). W3C Recommandation, http ://www.w3.org/TR/REC-xml-names, 6 Octobre 2000.