T.P.E.D. : PR9
Plateforme de Télétravail
à
Système Réparti Multi-Agents Intelligents
Rapport

Suivi assuré par Monsieur Philippe BENHAMOU (ONERA)
DARCHE Marc-Aurèle PIERRE Emmanuel SALVY Frédéric
SOULIEZ Antoine VARROY Marc-Olivier

Pour le 20 juin 1998

Table des matières

Chapitre 1

Introduction

Les entreprises tendent au cours de leur développement à s'internationnaliser, à développer des projets sur plusieurs pays, et ainsi en il en va de même pour toute tâche nécessitant un travail collaboratif de plusieurs personnes.

Cela va d'une échelle locale, un groupe de développeurs, à un ensemble plus global, une multinationale.

Un autre problème est le déploiement d'un système d'information. Quand une multinationale se lance dans le déploiement d'un applicatif, cela coûte des millions de francs, en logiciel, en installation et en maintenance.

Un autre coût majeur, est le temps passé pour la transmission et la recherche d'informations.

Un des intérêts des nouvelles technologies est de simplifier ces tâches, et de les rendre plus simples, ceci au travers de programmes conviviaux, simples, et de tâches mobilisant le moins possible leurs initiateurs, en déléguant cette tâche à la partie applicative.

Il faut aussi trouver des applications fonctionnant sous tout type de plateformes, et non pas sur des systèmes propriétaires comme beaucoup de solutions existantes (comme Windows par exemple).

Un point aussi à prendre en compte est le facteur humain. Comment ces outils modifient-ils l'organisation du travail, le *workflow*, et l'appréhension vis à vis de ceux-ci?

Le but de notre projet de TPED, est de tenir compte de ces facteurs.

Passons maintenant à des considérations linguistiques. 98% des documents (livres, fichiers textes, sites internets) et 100% des programmes que nous utilisons pour ce projet sont en langue anglaise.

Cela n'a rien d'étonnant car il est facile de constater que l'Anglais est la langue de l'Informatique et des Technologies de l'Information (IT).

De ce fait, nous avons en quelque sorte "pensé" notre projet en anglais. Et c'est avec un français parsemé de termes spécialisés anglais que nous nous sommes servis pour communiquer ("le 'Broker' génère 'getException' sur 'l'Object' qui redirige le message au niveau de 'Voyager' ", "Il nous faut trouver de quoi 'mapper' notre JAVA en IDL", etc...).

Nous avons donc pris le parti de garder la langue anglaise pour tout ce qui est d'ordre technique. Ainsi, la présence de termes techniques, d'acronymes et de schémas en anglais devient tout à fait naturelle. On remarquera même un glossaire majoritairement écrit en anglais.

Chapitre 2

Présentation du projet

2.1 Sujet

Ce projet se déroule dans le cadre des T.P.E.D. (Travaux Pratiques d'Etudes et de Développement) de 5ième année.

Le but de notre T.P.E.D. est de réaliser un ensemble Client/Serveur portable et évolutif avec les spécificités suivantes :

- L'architecture est répartie sur plusieurs serveurs.
- Chaque serveur dispose d'agents intelligents et autonomes pour recueillir les informations et assurer les liaisons avec les clients.
- Les serveurs communiquent entre eux et avec les clients par le protocole http, ce qui ouvre les applicatifs à tout le Web, même derrière des Firewalls.

Les clients peuvent échanger et partager documents et Bases de Données. Ils peuvent travailler simultanément.
Un tableau interactif est disponible, pour dessiner sur des textes et images vus de part et d'autre.
Le client peut étendre et mettre à jour l'ensemble de ses outils et agents. Agents qu'il peut programmer pour exécuter des tâches spécifiques.
Le serveur est capable d'échanger ses propres agents ainsi que ceux des clients avec d'autres serveurs.

Les connaissances stockées dans des Bases de Données peuvent être échangées via le format KIF ou KQML. Tous les programmes seront développés en Java et en Perl.

Le code Java des stations clientes peut être compilé en code natif pour augmenter sa vitesse d'exécution. Java permet à chacun des membres de l'équipe de développer séparément puis d'intégrer son travail à l'ensemble (comme étudié en GLPOO).

2.2 Etude de l'existant

Notre projet utilise les toutes dernières technologies, dites 'émergeantes'. La plupart de ces technologies ne sont d'ailleurs présentes qu'en version bêta sur l'Internet et n'ont, à l'heure actuelle, fait l'objet d'aucune application commerciale.
Donc nous nous devons de regarder ce qui existe déjà, afin de ne pas 'créer' quelque chose qui existe déjà.

- En ce qui concerne les systèmes d'agents et de télétravail, Lotus Notes (IBM) est un produit qui est en expansion dans le monde entier.
  Cet environnement permet de faciliter le travail collectif et collaboratif en proposant des outils de communication et de partage de documents au travers d'un réseau.
  Cependant, le principe de *PUSH* (envoi du serveur vers le client) n'est appliqué qu'à la transmission d'informations et non aux applications.
- Castanet Tuner utilise le *PUSH* applicatif, c'est à dire l'envoi de programmes et mise à jours automatique de ceux-ci. Cette technologie est très intéressante car elle ne nécessite pour le déploiement que un client de base. Le Tuner est un programme Java utilisant des méthodes Natives (donc propres à l'architecture), et permet via Java et les applets de gérer les sécurités des programmes. Cette technologie très avancée utilise une nouvelle norme de transmission via HTTP. Au lieu d'envoyer une myriade de connections pour un programme, les paquets sont envoyés en une seule passe. D'autre part, un sytème avancé de proxy fait qu'il n'est pas nécessaire de se connecter au serveur pour obtenir les applications, mais à un de ses sites mirroirs (la redirection est transparente pour l'utilisateur). Malgré cela, les applications collaboratives existant sur cette plateforme se limitent à des forums de discussion en direct (chat).
- Les Bases de connaissances, comme on peut en trouver dans des entreprises comme Hewlett Packard, permettent de regrouper les informations nécessaire à l'entreprise.

2.3 Apports par rapport aux applicatifs existants

Notre projet est différent dans le sens où nous avons l'intention d'intégrer ces différentes approches au sein d'un même logiciel.

- Interface graphique intuitive et attractive grâce aux extensions Swing de Java.
- Téléchargement par page web de l'application client.
- Pas de poste client dédié à un utilisateur, tous sont interchangeables avant identification de l'utilisateur.
- Facilité pour l'utilisateur par *PUSH* applicatif pour choisir et charger ses outils.
- Travail collaboratif à base de vrais agents mobiles.
- Vraie existence d'agents grâce à la persistence.
- Vrais agents distribués, effectuant leur tâche sur plusieurs serveurs.
- Gestion des transactions via HTTP pour passer au travers des Firewalls.
- Outils pratiques (surveillance URL, base de connaissances, recherche d'informations...).
- Travail hors connexion, ce qui génère un gain de temps pour l'utilisateur.

2.4 Contraintes

2.4.1 Contraintes de temps et contraintes matérielles

Le temps qui nous était imparti pour réaliser notre étude de T.P.E.D. (3 semaines) a été en grande partie absorbé par des microprojets qui ont mobilisés toute notre énergie.

D'autre part, nous avions souhaité disposer d'une machine que nous aurions utilisée afin de tester les produits disponibles pour notre projet.

2.4.2 Contraintes liées à nos objectifs

Notre projet étant orienté agents et communication entre agents, il nous a fallut chercher les langages, et les environnements acceptants nos exigences.

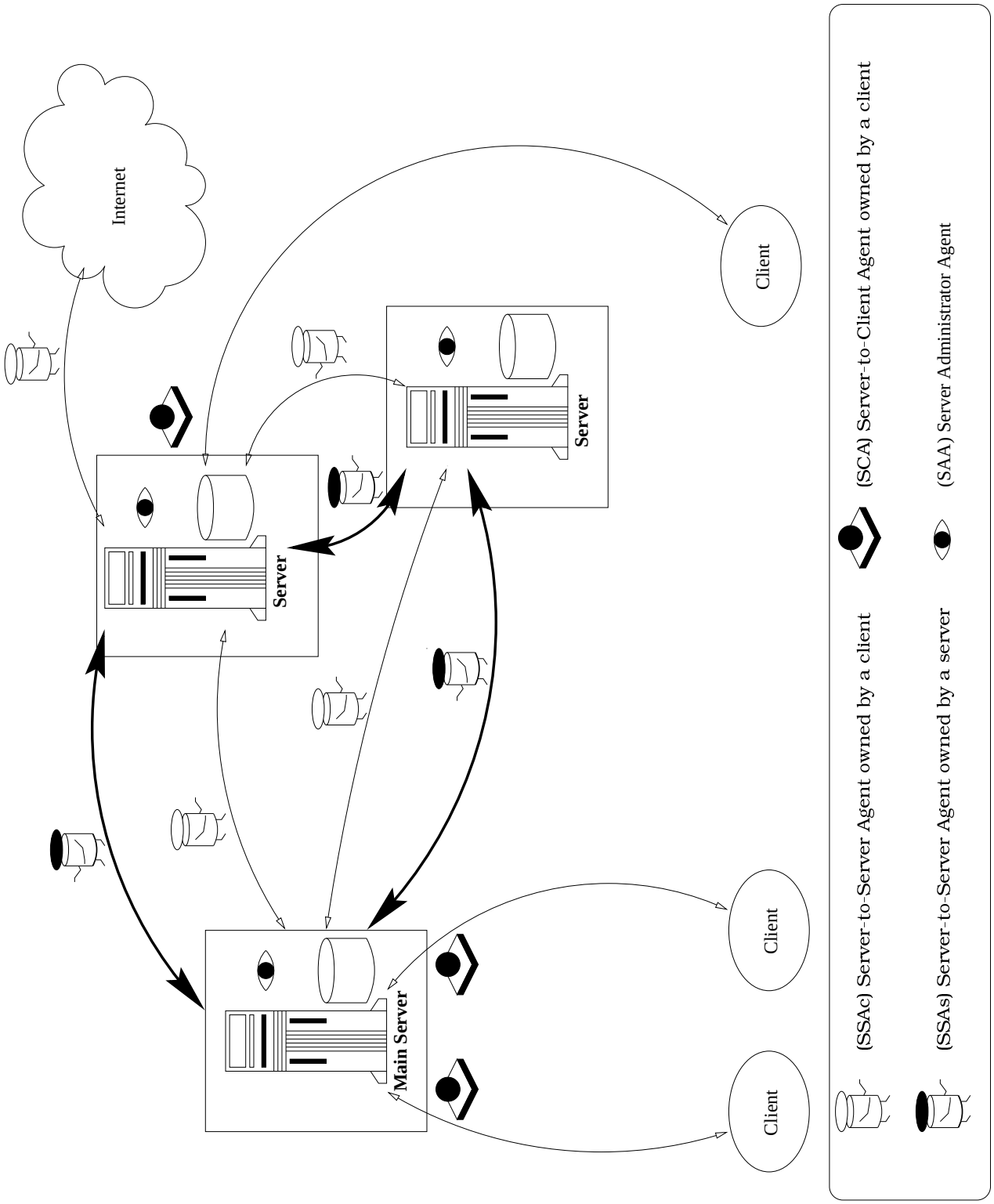
- Langages et environnements acceptant le concept d'objet (Java, C++, SmallTalk,...)
- Langage permettant le développement de moteurs d'inférence (nous avons utilisé Java lors de notre microprojet d'ARTC)
- Possibilité de communication de données entre programmes via le réseau et suivant le protocole HTTP (CORBA)

2.4.3 Contraintes liées à nos choix

Nous avons faits nos choix sur les divers produits que nous aurons à utiliser. Mais certains impliquent d'autres contraintes.

- Java Developper Kit 1.2 n'est pas encore disponible sous Linux et la version Windows est seulement en beta.
  Cette version nous permetterait de faire du CORBA, de disposer de propriété avancée sur les clients et des Java Foundation Classes (JFC) qui etendent le language.
- L'école ne dispose pas de JBuilder Client/Serveur, qui inclut une Base de données Objet (ObjectStore) et un ORB (VisiBroker).
- Il faut donc trouver un Object Request Broker (ORB), i.e. qui gère CORBA et qui fonctionne sur les plateformes voulues (Linux, Windows, HPUX, IRIX, SOLARIS,...)
  Notre choix s'est proté sur ORBacus (www.ooc.com).
- Voyager, plateforme IAD, et qui agit aussi comme ORB, nous servira pour faire voyager les agents, mais ne fonctionne que pour Java, et exclut ainsi l'utilisation d'autre language; ceci reste dans nos objectifs de protabilité.

Architecture générale



Chapitre 3

Le facteur humain

3.1 Un facteur souvent négligé

Lors de la création d'un produit le facteur humain est trop souvent négligé. Il apparaît que dans l'avenir, les systèmes d'information sont appelés à devenir une technologie complètement intégrée, naturelle pour tous. L'homme et la machine vont donc sans cesse interagir.

L'ergonomie d'un produit est souvent déterminante pour sa réussite. L'Interface Homme Machine (IHM) se situe à la confluence des sciences et des techniques informatique, considérant qu'il est essentiel pour réaliser une interface (assemblage de composants matériels et logiciels) de prendre en considération l'ensemble des phénomènes physiques et cognitifs qui interviennent dans la communication entre l'homme et la machine.
Mais l'ergonomie n'est pas tous, il faut aussi prendre en compte les besoins et surtout des habitudes et façons de penser des utilisateurs. Ce n'est pas évidemment car chaque personne est unique, mais il faut faire un effort de standardisation.

Un produit doit être convivial s'il veut s'adresser au plus grand nombre.

En effet les utilisateurs ont des habitudes et travaillent selon des
protocoles (protocoles de travail). Même si le produit demande de travailler
d'une manière différente, il faut tenir compte des habitudes passées des utilisateurs de manière à leur éviter à passer du temps dans la manipulation du nouveau produit, afin de pas réduire leur productivité.

Lorsqu'on conçoit un nouveau produit, il faut savoir que pour les utilisateurs
il y aura :

- Modification des protocoles de travail.
- Modification des habitudes.
- Modification de la productivité.

Dans le secteur informatique, la principale erreur des concepteurs est qu'ils
oublient que les utilisateurs ne sont pas, la plupart du temps, des informaticiens.
Les concepteurs oublient souvent de se mettre à la place des utilisateurs. Lorsqu'on conçoit un produit, il faut 'essayer' de partir avec un esprit vierge, de même lors des tests de convivialité du produit.

Cela nécessite une culture générale informatique et une connaissance de l'organisation des entreprises. En effet, il faut synthétiser les domaines suivants :

- les études des besoins

- l'écoute des utilisateurs et le cahier des charges
- fonctionnalités nouvelles attendues
- volume, utilisation, circulation par moi et par an
- les choix techniques, matériels et logiciels qui en découlent
- la reprise de l'existant
- les formations
- mesure et établissement d'un plan qualité

3.2 Prise en compte du facteur humain

Nous nous attachons surtout à la productivité

Notre interface sera très ergonomique et la conception de notre produit est conçue de manière à améliorer la productivité des utilisateurs en permettant une consultation et un échange plus facile et plus rapide de l'Information.

Ainsi, par exemple, la place de la souris sera matérialisée sur le bureau virtuel qu'aura chaque client.
Nous utilisons le module graphique Swing de Java qui permet d'inclure des objets garphiques standards et intuitifs : boutons, barres défilantes, onglets, etc...

Il faut accompagner et favoriser le changement de comportement des utilisateurs et savoir anticiper les différentes techniques, organisationnelles et culturelles pour assurer le succès de l'implémentation de ces types d'organisation au sein de l'entreprise.

Chapitre 4

Moyens technologiques

4.1 Inventaire des logiciels nécessaires sur chacun des postes

Logiciel/Machine	Serveur Linux	Serveur HPUX	Serveur NT	Client
JDK 1.1.6	•	•		•(Linux)
JDK 1.2	si dispo	si dispo	•	•(NT)
JBuilder Client/Serveur			•	
JFC	•	•	•	
Voyager	•	•	•	
IFC	•	•	•	
Jgl	•	•	•	
Perl 5 +libww+libnet	•	•	•	
COPE	•	•	•	
Internet Mail	•	•	•	
DataDumper	•	•	•	
ORBacus	•	•	•	
APACHE 1.3	•	•	•	
mSQL 2	•	•		
Oracle		•	•	

4.2 Le MiddleWare

Les utilisateurs de systèmes d'information ont de plus en plus besoin d'échanger et de partager des informations. Ces informations peuvent être de nature très différentes et sous des formats qui le sont tout autant (fichiers textes, images, messages, données, etc...). Et, plus récemment ces informations peuvent être situées à des endroits différents c'est à dire sur des postes distants, sur des réseaux différents ou encore sur l'Internet.

La diversité et la spécialisation des supports utilisés sont inversement proportionnelles à la facilité que les utilisateurs ont à se les échanger. De plus il existe des milliers d'applications indépendantes écrites dans des quinzaines de langages différents sur des dizaines de systèmes d'exploitation différents.

Ce sont, de ce manque de standards et du besoin vital de communiquer (surtout pour les entreprises) et de ce marché très porteur, que sont nées un nouveau type de suites logicielles: les Middleware.

Concrètement, un Middleware est un ensemble de programmes qui isole une application de l'ensemble des processus qui résultent de son lancement sur le système.

Un MiddleWare permet la communication entre des clients et des serveurs ayant des structures et une implémentation différentes. Il permet l'échange d'informations dans tous les cas et pour toutes les architectures.

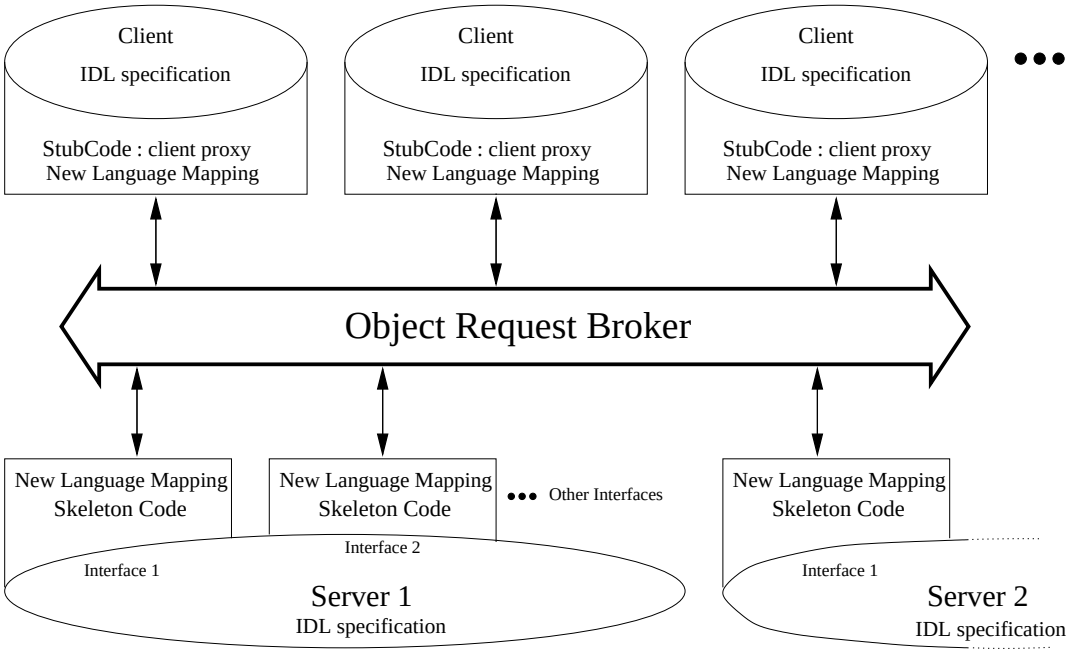
Enfin, les MiddleWare doivent fournir un moyen aux clients de trouver leurs serveurs, aux serveurs de trouver leurs clients et en général de trouver n'importe quel objet atteignable.

4.3 Présentation de CORBA

4.3.1 Vue globale

CORBA est la spécification d'une architecture orientée objet pour applications. Elle a été définie par l'OMG dans un document publié au mois de novembre 1990.

Avec CORBA, le client et le serveur sont formellement séparés, on peut alors changer l'un sans changer l'autre. Ainsi le client qui utilise une interface CORBA n'a pas besoin de connaître la façon dont le serveur va exécuter la tâche mais seulement comment faire une requête pour qu'une opération soit effectuée sur un objet.



4.3.2 Le Broker

Dans un environnement classique de client/serveur, il existe une relation de tête-à-tête entre les clients et leurs serveurs. CORBA ajoute un intermédiaire entre le client et le serveur : l'Object Request Broker, ou *Broker*. Le broker possède " l'intelligence " nécessaire pour mettre en rapport la requête provenant d'un client avec les serveurs qui peuvent y répondre.

L'apport d'un broker entraîne plusieurs améliorations :

- Le client et le serveur n'ont plus besoin de se connaître de façon directe. Ils se trouvent grâce au broker. Ainsi seul le broker a besoin de connaître la localisation et les ressources disponibles du client ou du serveur qui se trouvent sur le réseau.
- Une relation en tête-à-tête entre les clients et leurs serveurs n'est plus requise. Ainsi grâce au broker, plusieurs serveurs peuvent travailler avec un seul client, ou un seul serveur peut travailler avec plusieurs clients.
- Une application cliente peut localiser et inter-réagir avec un objet ou un serveur durant une transaction. Dans l'environnement classique client/serveur, la requête est prédéfinie alors qu'ici, le client peut invoquer une requête sur un objet durant la transaction.

4.3.3 La requête

CORBA sépare le client du serveur en restreignant la communication qui peut exister entre eux par un type de message appelé requête. Chaque interaction dans un système CORBA est :

- soit un client qui envoie une requête, c'est à dire une invocation.
- soit un serveur qui répond à une requête.

Toutes les requêtes ont la même structure:

- une indication sur l'opération que doit exécuter le serveur à la demande du client,
- une référence spécifique à l'objet sur lequel le serveur doit effectuer l'opération,
- un mécanisme qui doit permettre de retourner un message concernant la réussite ou l'échec de l'opération,
- une référence optionnelle à un objet de contexte,
- et des arguments spécifiques à l'opération à effectuer.

4.3.4 L'IDL, Interface Definition Language

Le client et le serveur ont besoin d'informations pour pouvoir travailler sur les mêmes objets. Par exemple, chaque objet doit annoncer les opérations que l'on peut effectuer sur lui.
CORBA dispose ces informations dans une interface qui définit les caractéristiques et les comportement de chaque type d'objet, y compris les opérations qu'un serveur peut exécuter sur ces objets. Pour définir une interface, on utilise l' IDL pour coder l'information dans un jeu de définitions d'interfaces. Les développeurs entreposent ces définitions d'interfaces dans des fichiers IDL.

Ainsi avant d'écrire une application cliente ou serveur, on doit d'abord définir son ou ses interface(s), c'est à dire créer un fichier IDL. C'est ce fichier qui contient les définitions des interfaces que le client ou le serveur supporte.

L'IDL est un langage de définition et non un langage de programmation grâce auquel on définit des interfaces et des structures de données et non des algorithmes. On utilise des fichiers IDL pour générer un code source pour le langage de programmation désiré. On peut donc, par exemple, écrire des applications clientes en LISP et des applications serveurs en C et faire communiquer ce client et ce serveur.

4.3.5 Les exceptions

CORBA communique des informations sur la réussite ou l'échec d'une opération. Ces informations sont considérées comme des exceptions. Toutes les requêtes doivent contenir un mécanisme qui permet de retourner des exceptions. La syntaxe dépend du langage de programmation utilisé.

Une méthode de l'application du serveur peut fournir l'information sur l'exception s'il y a une erreur ou par exemple s'il n'y a pas assez d'arguments dans la requête. Le broker lui-même peut le faire si par exemple l'accès d'un serveur lui est interdit.

L'application client peut lire mais ne peut pas modifier une exception. Par défaut, s'il n'y a pas de retour d'exception, l'opération est considérée comme réussie. Par contre s'il y en a une, l'application cliente doit vérifier si l'opération est réussie ou non.

4.4 Voyager: une plateforme multi-agents

4.4.1 Présentation

Voyager est un produit développé par ObjectSpace. Voyager implémente une plateforme pour systèmes distribués, il est 100% Java et adapté pour supporter son modèle d'objets.
De plus, il est utilisable gratuitement.

Voyager inclut un ORB supportant les objets mobiles et Agents Autonomes.
Les services fournis sont entre-autres la persistance, la communication de groupe scalaire et une gestion basique des services Directory.

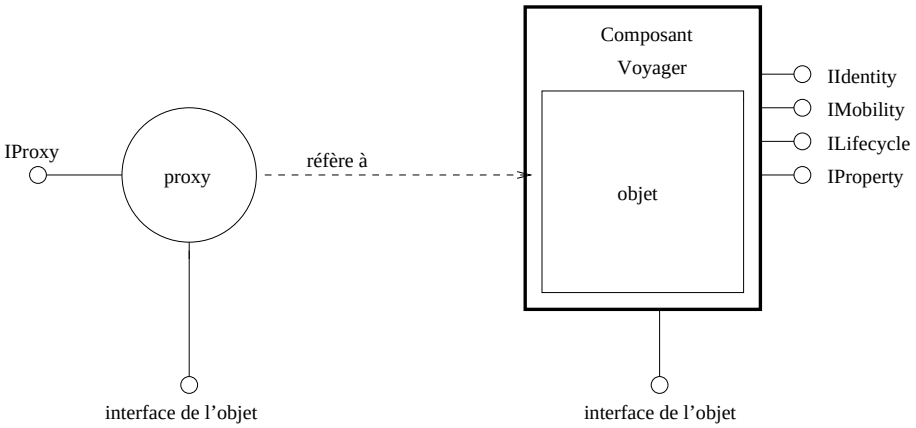
Un des avantages de Java est de charger en cours de fonctionnement des classes dans sa Machine Virtuelle (VM). Cela permet d'utiliser des objets mobiles et des agents autonomes comme outils pour construire des objets distribués.

4.4.2 Fonctionnalités

La version 2 beta 1 permet une intégration bidirectionnelle de CORBA, ce qui permet à Voyager d'être utilisé comme client ou serveur CORBA 2. Il est possible de générer une interface distante Voyager à partir de n'importe quel fichier IDL. Cette interface est utilisable pour communiquer avec n'importe quel serveur Voyager ou CORBA.

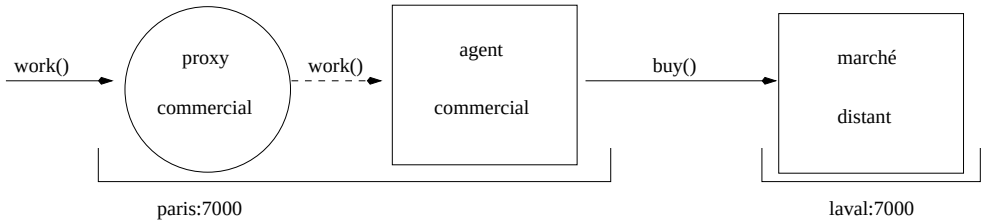
- Les composants Voyager

Ils étendent les propriétés de l'objet avec une interface spéciale (IIdentity, IMobility, ILyfecycle et IProperty).

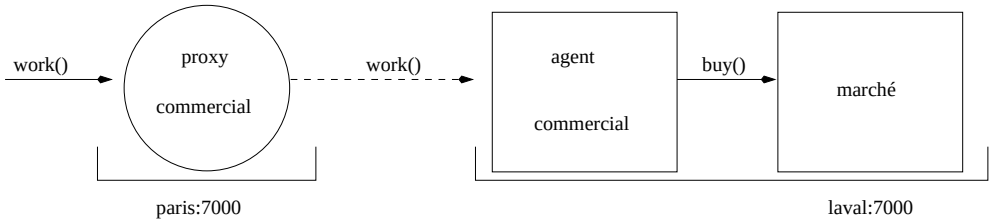


IIdentity donne un *alias* à un objet et donne accès à son adresse courante. Cela permet d'adresser ultérieurement un objet.

IMobility permet de déplacer un objet d'une VM à une autre dans un même système, ou à travers le réseau.



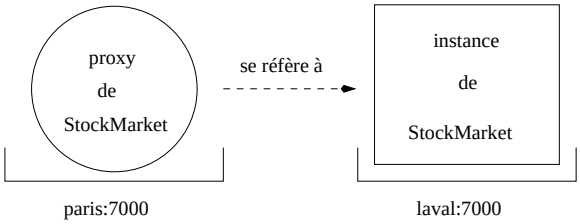
avant déplacement



ILifecycle controle le cycle de vie et la persistance d'un objet. Lorsqu'un objet atteint la fin de son cycle de vie, il "meurt", détruit par le Garbage Collector (GC). Le cycle de vie par défaut d'un objet dure tant qu'une référence régulière de Java ou d'un proxy le réfère. Celui d'un Agent est d'une journée (24 heures).

IProperty associe une paire clef/valeur à un objet.

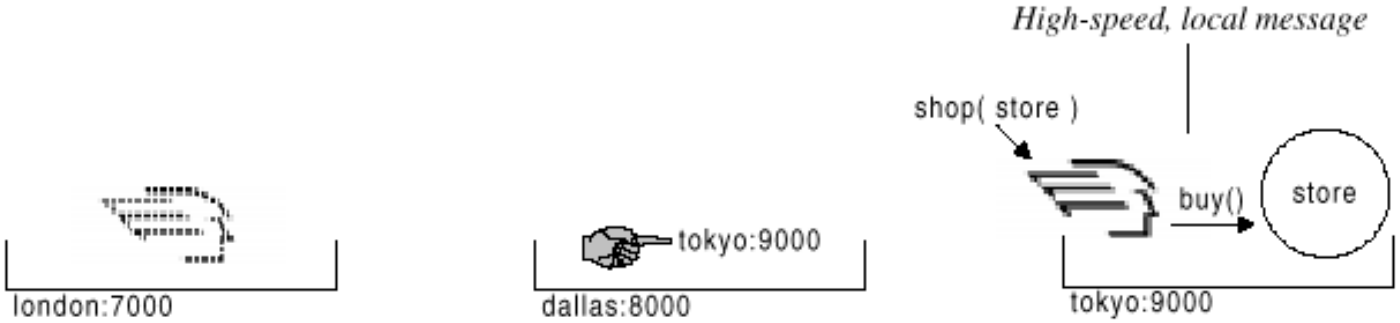
IProxy permet de modifier différents attributs d'un proxy.



Dans cet exemple, tout message envoyé au proxy à travers l'instance virtuelle locale de StockMarket est automatiquement envoyé à l'objet distant, comme s'il était local.

- mobilité : Voyager supporte la mobilité d'objets sérialisables, pas seulement les Agents. Il n'est pas nécessaire de modifier les classes pour avoir le support de mobilité. La mobilité permet :

- de rapprocher deux objets nécessitant d'avoir une discussion utilisant beaucoup de bande passante, et ainsi d'augmenter la vitesse de leur traffic et réduire la charge du réseau.



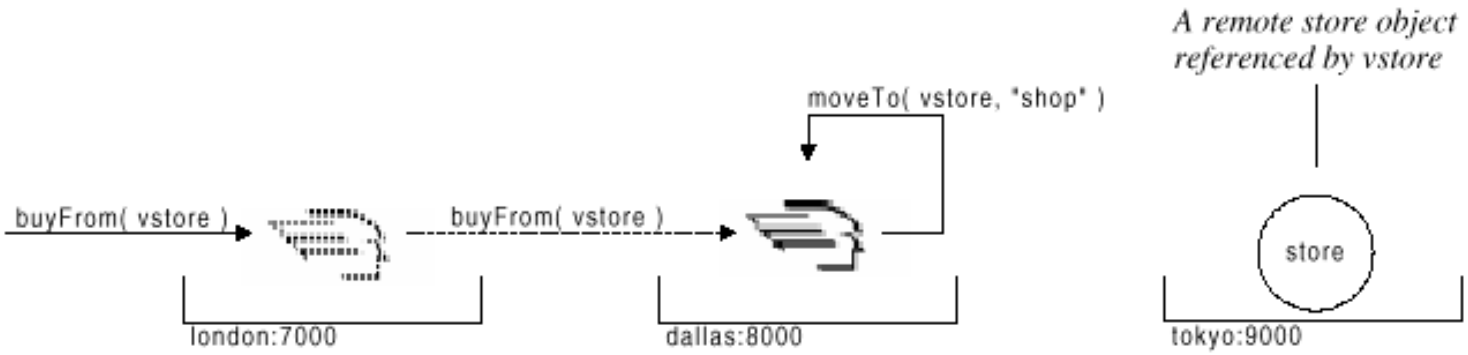
- de déplacer un objet qui a besoin de la persistance ou d'un processeur puissant, vers une machine qui a ces propriétés.

- de déplacer l'objet d'une machine qui est sur le point d'être déconnectée du réseau, et de le placer sur une autre machine et continuer sa tâche.

Il est possible de déplacer un objet vers un nouveau programme, même s'il est en train de reçevoir de

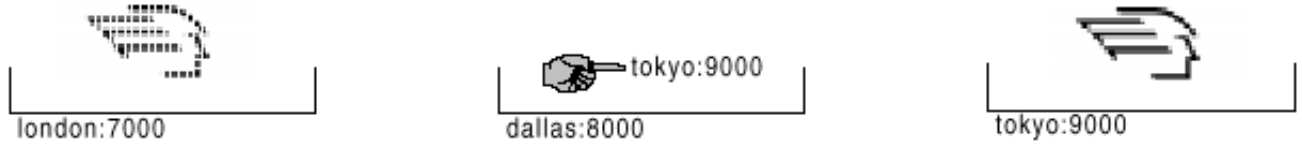
nouveaux messages, comme Voyager gère toute la synchronisation et le forwarding des messages.

- itinéraires : C'est un ensemble de tâches qui doivent être exécutées sur un ensemble de sites.

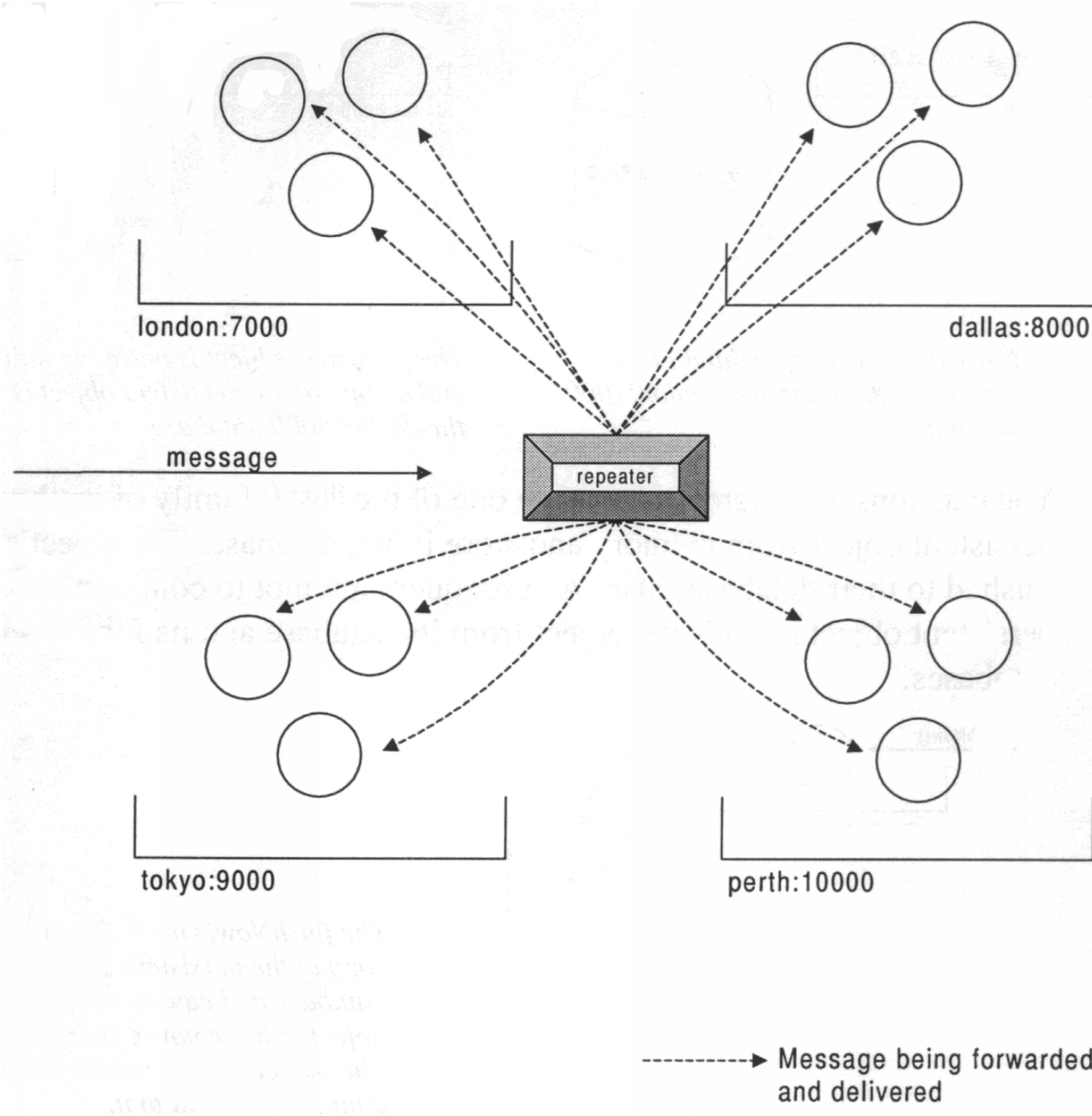


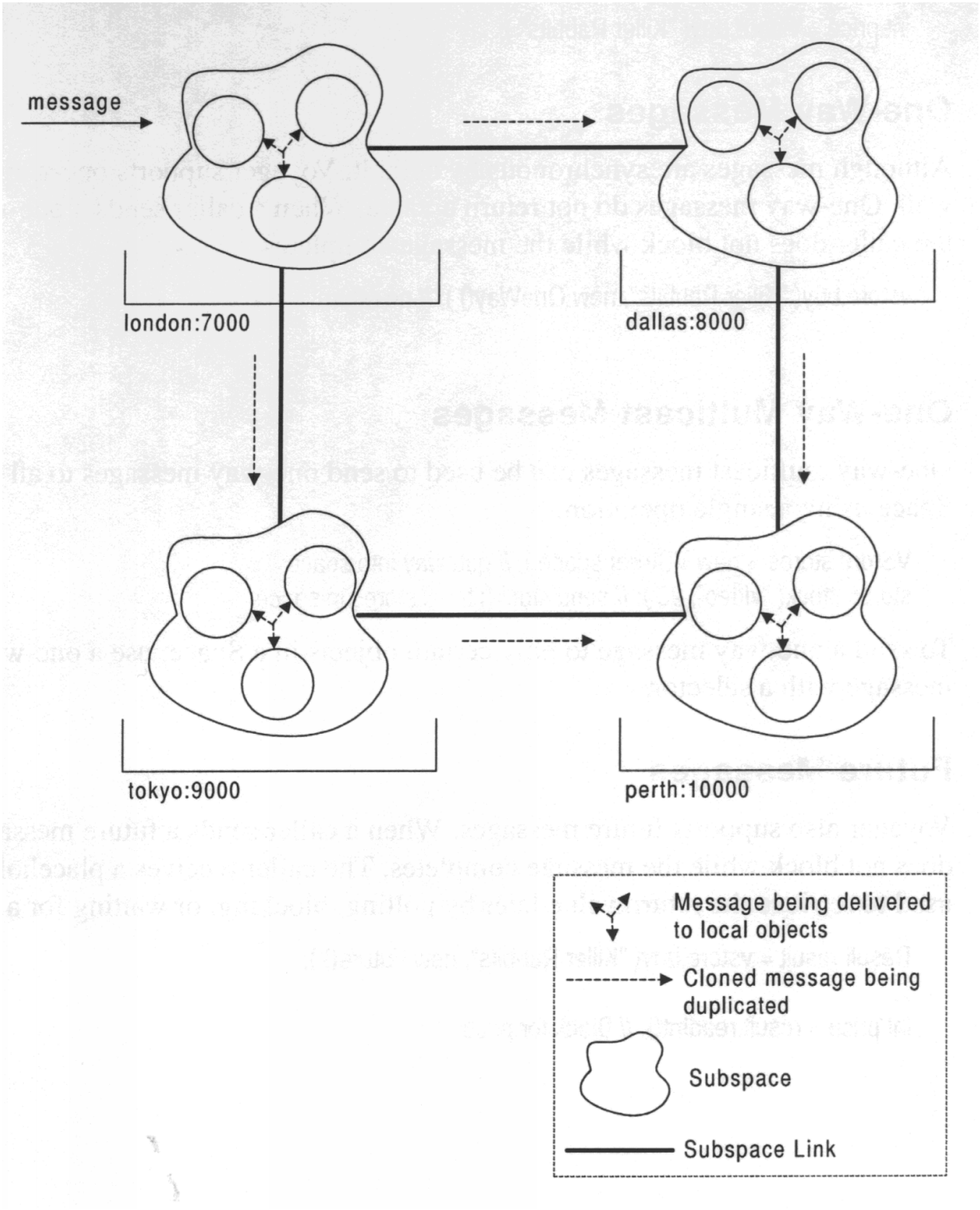
- persistance : C'est l'habilité qu'à un objet de vivre au-delà de la vie d'une application. Elle est obtenue en stoquant l'objet dans une BDD, spécifiée au démarrage de Voyager. Ensuite, un message destiné à un objet persistant qui n'est pas actuellement en mémoire, charge (automatiquement) l'objet de la BDD et l'active. En interne est utilisé VoyagerDb, mais il est possible de spécifier une autre base de donnée (via JDBC).

- agents : Un Agent est un agent spécial de Voyager qui est autonome. Un agent peut être programmé pour atteindre un ou plusieurs buts, même si l'agent se déplace et perd le contact de son créateur. L'*autonomy* d'un Agent lui permet de se déplacer de lui-même.



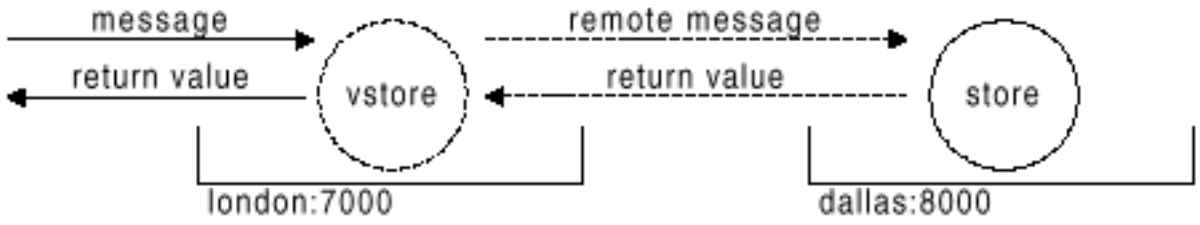
- scalabilité et espaces : Un espace logique (**Space**) consiste en un ensemble d'objets sous-espaces (*subspace*), chacun contenant un sous-ensemble des éléments de l'espace. L'architecture Space exploite le parallélisme de telle façon que les opérations sur un espace utilise le temps processeur de chaque machine qui contient un sous-espace. Chaque message envoyé à un sous-espace est cloné pour chaque sous-espace dans l'espace, ce qui fair un rapide déploiement de messages à chaque objet dans l'espace. Chaque sous-espace s'assure qu'aucun message ou évènement n'est délivré plus d'une fois.


london:7000
dallas:8000
message
repeater
tokyo:9000
perth:10000
Message being forwarded
and delivered



- services temporels : cela contient deux classes, l'une *Stopwatch* génère des statistiques d'exécution (profiling), l'autre *Timer* permet de générer des alarmes ponctuelles ou périodiques.

- types de messages :cinq types de messages sont supportés : synchrones, unidirectionnel, bidirectionnel, unidirectionnel par Multicast et future. Chaque fois qu'un objet se déplace, il laisse derrière lui un objet *forwarder* qui fait suivre les messages vers la nouvelle position de l'objet. Quand l'objet meurt, ses *forwarders* aussi. Comme Voyager traite les Agents comme des objets, on peut créer des agents distants et leur envoyer des messages pendant qu'ils se déplacent.



- messages dynamiques via IProperty, cela permet ce que l'on appelle le Publie/Souscrit (*Publish/Subscribe*); un objet souscrit à un sujet en utilisant IProperty et associant une clef au sujet. Un message Multicast peut alors être délivré à tous les objets ayant souscrit au sujet.

- Naming et Directory Service : cela permet d'associer un alias à un objet et de se connecter à cet objet via son alias plus tard, même s'il a bougé. Cela est aussi associé à la persistance de l'objet.

- garbage collector (GC) : Il s'agit de la destruction automatique d'un objet, libérant la mémoire utilisée par l'objet à la demande de la VM Java. le GC détruit aussi les objets persistants de la BDD quand cela est nécessaire.

- durée de vie *(Life Spans)* : un Agent peut avoir un cycle de vie normal, c'est à dire mourir quand il n'y a plus de référence le pointant, mais aussi une durée fixée, une date fixée, vivre et rester inactif pendant une certaine période, ou vivre indéfiniment.

- transaction de services (*Voyager Transaction Services* (VTS)) est une API basée sur Voyager compatible avec CORBA OTS et JTS de Java, permettant de gérer des séquences d'opérations au niveau d'atomes, pour gérer si la séquence entière fonctionne ou échoue.

- connectivité avec les applets : Voyager inclut un routeur logiciel minimal qui permet à un serveur de servir de passerelle (*gateway*), et donc permettant une complète connectibilité applet à applet et applet à programme. Le routeur permet aussi aux Agents Voyager de se déplacer librement entre les applets et les programmes.

- sécurité : la classe *VoyagerSecurityManager* peut etre installée et personnalisée par dessus celui de Java, et permet de restreindre les actions que peuvent faire les agents mobiles.

4.4.3 Voyager et CORBA

Le fait que Voyager soit en Java permet d'exécuter une classe distante sans modification, ce que ne permet pas la définition de CORBA.

Voyager n'est pas seulement un ORB, qui plus est un ORB CORBA, mais offre des possibilités que les solutions actuelles ne permettent pas via CORBA.

- Voyager peut prendre n'importe quel fichier IDL et créer une interface équivalente en Java et sa classe de référence pour Voyager, ou n'importe quelle classe et en extraire l'interface IDL équivalente, permettant à une classe Java d'utiliser CORBA.

- Voyager peut obtenir une référence virtuelle d'un objet CORBA et vice-versa.

- Un programme peut envoyer un message à un objet via une référence virtuelle sans savoir s'il s'agit d'un objet CORBA ou Voyager. Si l'objet est dans un ORB CORBA, Voyager utilise IIOP pour communiquer avec l'objet CORBA.

- Les référence virtuelles à des objets Voyager sont automatiquement converties en références CORBA quand elles sont envoyées à un ORB CORBA, et vice-versa.

- Voyager supporte les paramètres in, inout, et out à travers le mécanisme de gestion standard.

4.5 Le langage KQML

Le langage KQML est issu d'un projet de recherche fondé par le DARPA pour permettre à des agents cognitifs de coopérer.

Il s'agit d'un riche ensemble de types de messages et de signification à but performatif, i.e. pour faire une fonction.

Cela permet au récepteur de faire une action qui est compatible avec les autres aspects de sa fonction.

Un ensemble de performatifs est réservé, cela concerne l'émetteur, le récepteur *(sender, receiver)* ainsi que le type de langage et le type de message *(language,content)*.

Mais il est recommandé que chaque personne implémentant ce protocole étende le vocabulaire de base pour l'améliorer et l'adapter. Il s'agit en fait d'un protocole de base extensible, mais qui n'inclut pas comme KIF des échanges d'informations.

Exemple de message:

OrderAgent

(address <id><host><port>) - registered agent address. (drive (coordinate <x1><y1>)(coordinate <x2><y2)) - drive order.

Facilitator

(address <id><host><port>) - registered agent address.

TaxiAgent

(busy) - exists if the taxi is busy. (location (coordinate <x><y>)) - taxi location. (arrival-time (coordinate <x><y>) <time>) - expected arrival time. (drive (coordinate <x1><y1>)(coordinate <x2><y2)) - drive order (NOTE: not in VKB).

4.6 Stockage de données

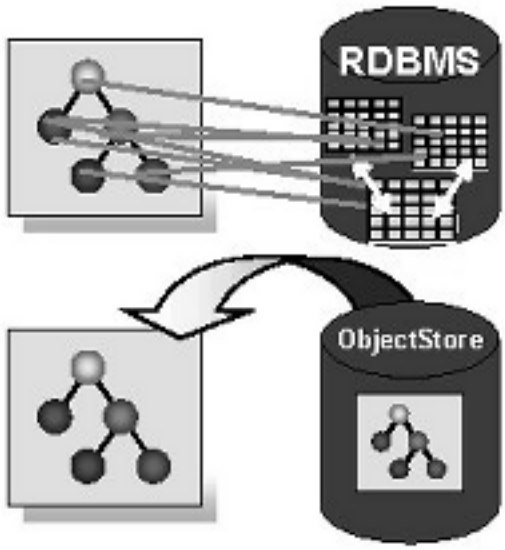
4.6.1 JGL

textbfObject Database Management Systems

Pourquoi choisir ODBMS?

Utilisation de d'objets et d'agents

La base de données orientée objet se lie naturellement avec la programmation orientée objet



OBJECTSTORE ELIMINATES MAPPING CODE AND JOINS BY MANAGING DATA IN COMPONENT-READY FORMAT—AS OBJECTS

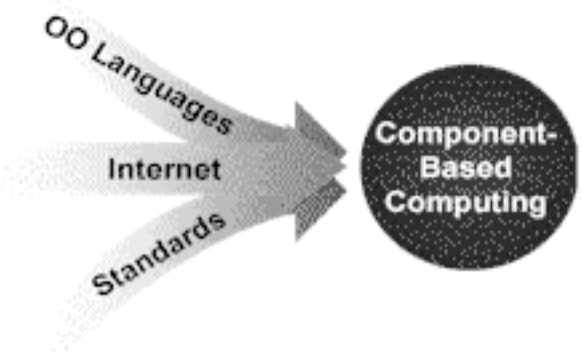
Disponibilité d'outils de développement

Données complexes avec de nombreux liens entre elles

Performances accrues par rapport au RDMS

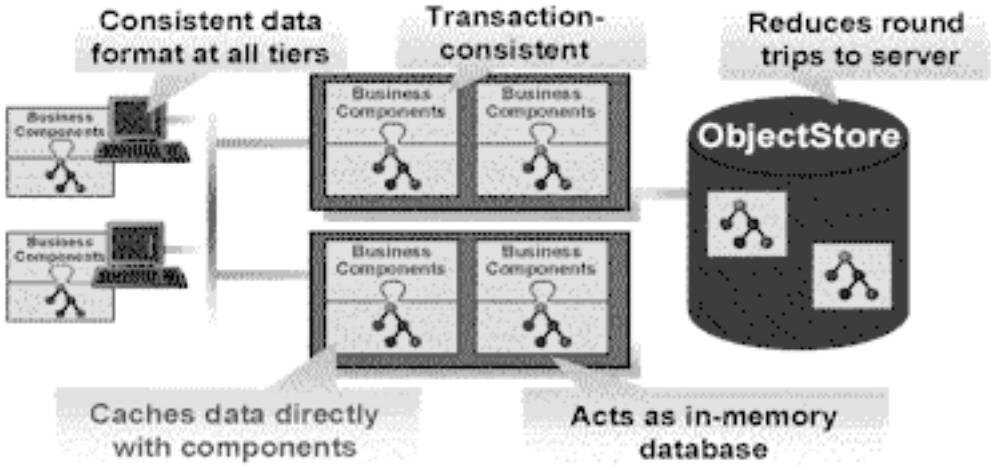
Facilités de stockage d'objets pour le web (fichiers HTML , applets Java, fichiers d'images)

Facilité d'implémentation et de maintenance par rapport à RDBMS avec technologie objet.



Object store

La méthode de cache-anticipé de object store (ObjectStore's Cache-Forward) est unique dans le sens ou elle utilise l'architecture middleware pour déporter la base de données et les applications sur le réseau.



En conséquence, les performances sont optimisées dans un environnement distribué : au lieu d'obstruer le réseau avec des requêtes SQL, seuls les objets sont transférés et stockés à la fois dans le cache du serveur et

dans le cache des clients. D'où un gain important de rapidité sur des requêtes répétitives.

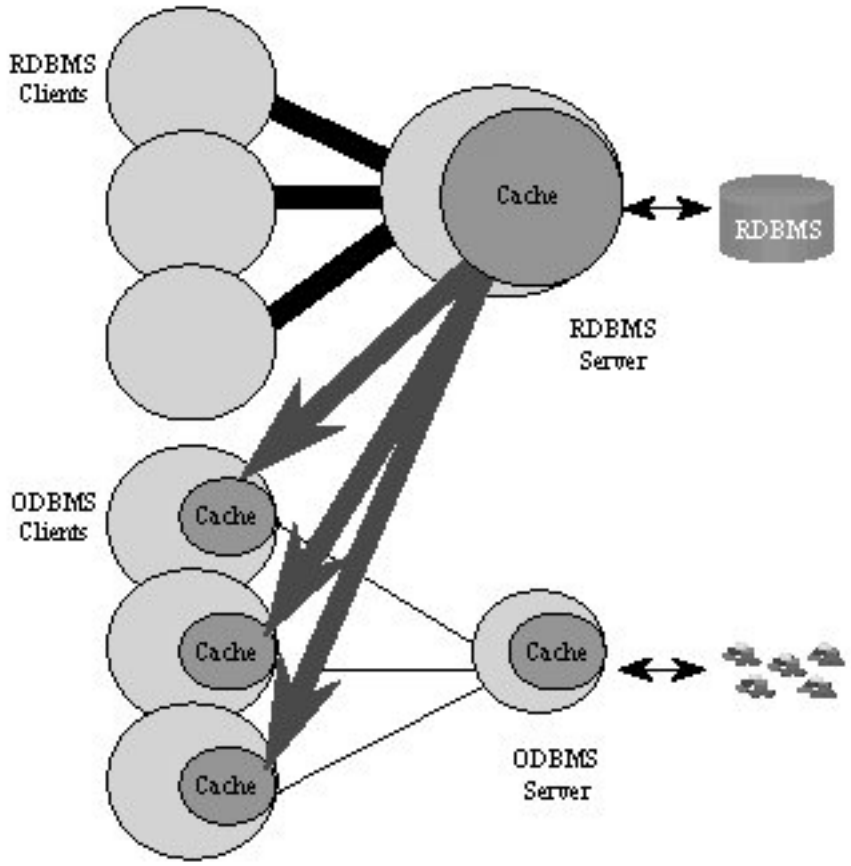


FIGURE 7 : OBJECTSTORE DISTRIBUTED CACHES PROVIDE IN-MEMORY PERFORMANCE

Pour notre projet, nous avons besoin de stocker les objets dans une bases de données orientée objet (persistance) ; mais nous avons aussi besoin d'accéder à des bases de données relationnelles. Ceci est rendu possible par les modules DBConnect et OpenAccess (cf figure). DbConnect permet d'intégrer des données relationnelle dans la base orientée objet ; alors que Open Access fait le contraire. Ces deux outils sont très importants pour les entreprises puisqu'ils permettent une transition relativement aisée à la technologie objet,

sans modification de leurs bases de données et leurs applications.

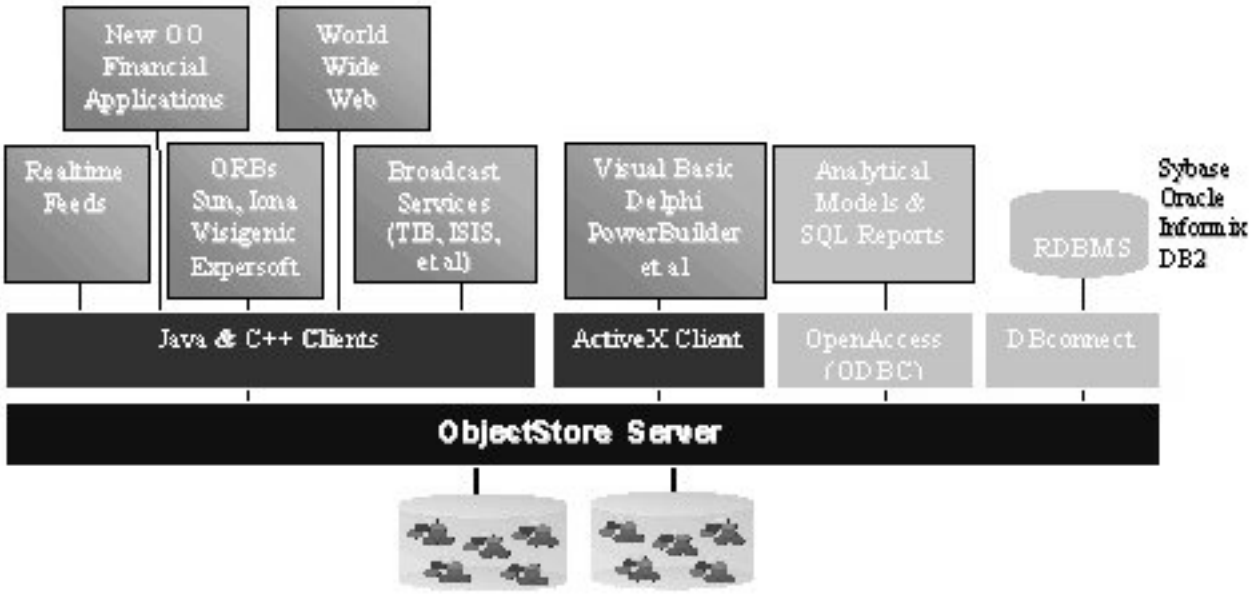
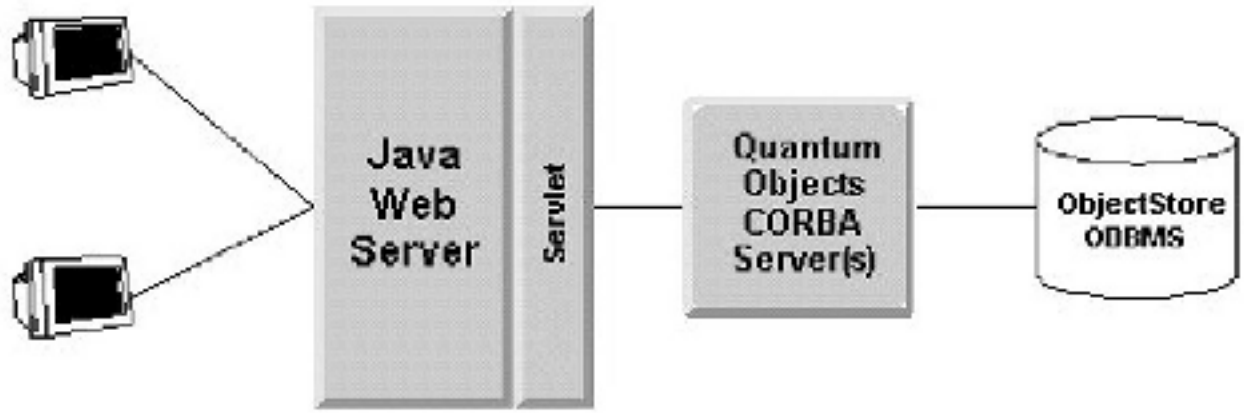
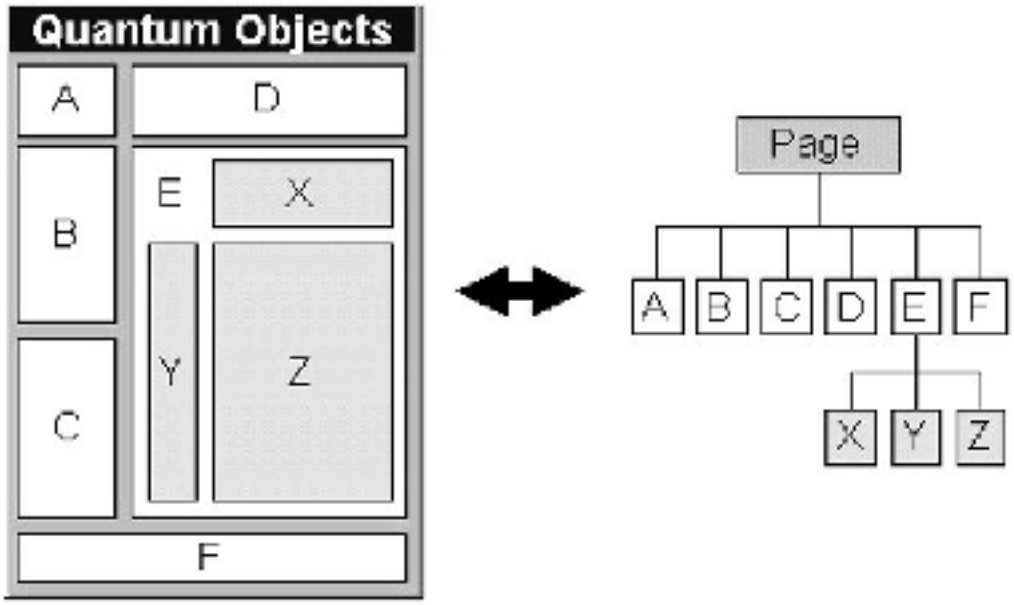


FIGURE 9: OBJECTSTORE INTEGRATES WITH LEADING ORBs, LIBRARIES & DATA SERVICES

Object Store et le web

Comme nous l'avons vu, Object Store permet l'archivage très rapide et intuitif de toutes sortes d'objet. Ce sont les caractéristiques idéales d'un serveur web dynamique avec une maintenance aisée. Deux produits existent afin de créer des pages HTML à partir d'objets. Des sociétés ont même développé un moteur automatique et dynamique de pages web à partir de composants de base (images, texte et applets). Cette technologie, Quantum Object, utilise les logiciels ObjectStore Design et le langage Java.





Du point de vue implémentation, ObjectStore offre un support naturel de Java (il n'est pas besoin de modifier les objets et les classes afin de les rendre persistants). De plus, ObjectStore est le seul logiciel de bases de données orientées objets qui permet de travailler dans les deux standards de composants objets : Microsoft DCOM/ActiveX et l'OMG de CORBA/IIOP. Le dernier point qui nous interesse est la possibilité d'utiliser des librairies de 3ème génération pour Java, nottament JGL que nous verrons par la suite.

Avec l'accroissement de l'échange d'information en tous genre, il est devenu impératif de trouver une solution de stockage et de récupération à la fois facile à implémenter, performante et flexible.

4.6.2 Base de Données Orientées Objet

Java Generic Library

ObjectSpace JGL est une librairie Java en provenance directe du Standard Template Library du C++. JGL utilise donc les puissants concepts de classes aupraravant réservée au C++ dans un environnement Java (héritage, polymorphisme, ..).

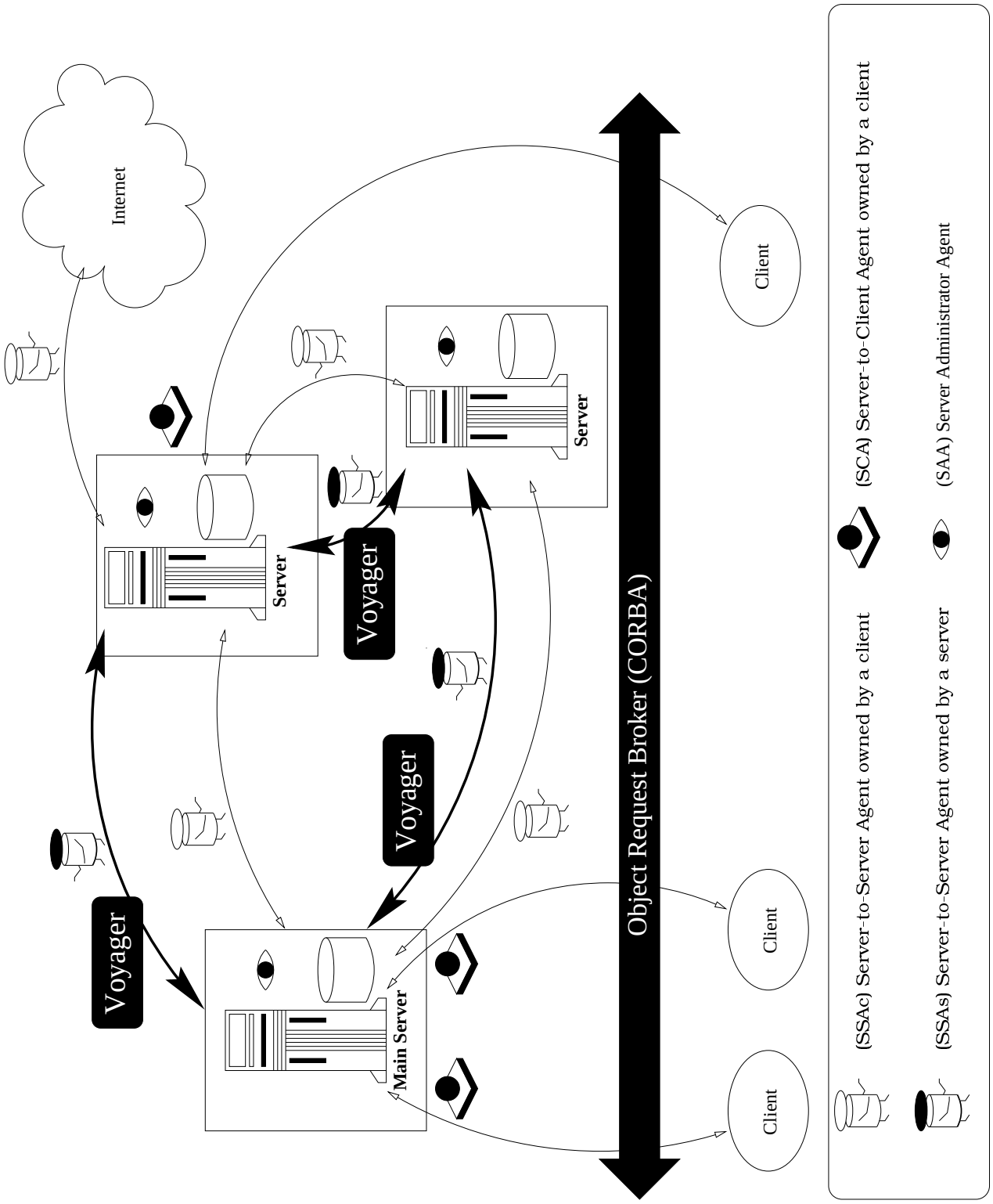
Le principe de JGL est la manipulation aisée et sous forme de classes de Type Abstraits de Données. Un bon exemple est la classe de fonction qui permet de convertir en minuscule n'importe quelle chaine de caractères (qui ce soit le nom d'une fonction, une variable, un tableau,...). Des structures de bases sont disponibles pour le développeur, à lui de les mixer comme il faut pour que la manipulations de données sont la plus aisée et puissante possible. En effet, JGL utilise toujours les mêmes méthodes pour manipuler les Types Abstraits de Données, tout en gardant un code source lisible, et facilement maintenable.

L'idée forte de JGL est de mettre à la disposition des développeurs Java la puissance du concept objet venant du C++, mais en gardant la facilité et l'universalité de Java.

Chapitre 5

Etude détaillée

Architecture détaillée avec choix technologiques



5.1 Les Agents

Sur notre système réparti, une grande partie des taches et des fonctionnalités est assurée par des agents. Chaque agent est spécialisé dans une seule tache.
Certains agents sont complètement indépendants et autonomnes, mais certaines fonctionnalités nécessitent le travail simultanné de plusieurs agents. Ces agents ont un mode de travail proche de celui des fourmis. On parle alors d'Intelligence Artificielle Distribuée.

Certains des agents du système sont mobiles. C'est à dire qu'ils se déplacent de serveurs en serveurs. Les agents mobiles sont représentés avec des pattes.

- 'Agent Personnel' (SCA)
  Il est entièrement dédié à l'utilisateur, il permet de gérer les agents utilisés pendant que l'utilisateur n'est pas connecté, il sert aussi d'interlocuteur lorsqu'il faut transmettre un message à un agent non présent sur le serveur.

- 'Agent Scheduler' ou 'agent prise de rendez-vous' (SSAc)
  Cet agent permet à un utilisateur de prendre des rendez-vous (avec des collaborateurs par exemple) en fonction de son agenda. D'autres utilisateurs peuvent demander un rendez-vous et les agents de l'utilisateur s'occupent de trouver un créneau horaire. Ces agents gèrent de manière autonomne un agenda, et sont amenés à déplacer des rendez-vous de manière intelligente. Ces agents utilisent le '**trading**'. Le trading consiste à donner une pondération à chaque information en fonction de la valeur que son possesseur lui accorde. Un exemple: "si je ne veux surtout pas déplacer une réunion avec des développeurs à 10h, je la pondère avec un coefficient 10. De cette manière, mes agents intelligents de prise de rendez-vous savent qu'ils ne peuvent déplacer cette entrevue".
  Ces agents intelligents communiquent et arrivent à **une décision par consensus**. La création de ce type d'agents fait appel à de l'**Intelligence Distribuée** et constitue une part importante de notre projet.

- 'Agent Echange d'Informations'
  Cet agent reprend un peu la structure du précédent car il utilise du 'trading'. Chaque utilisateur spécifie à ses agents "échanges d'informations" quelles sont les informations qu'il est près à échanger et à quel 'prix' il est près à les échanger. Il s'agit également ici d'Intelligence distribuée.

- 'Agent Recherche d'Informations dans les BdDs des Serveurs'
  Agent qui recherche à partir de critères des informations sur les différentes Bases de Données des serveurs.

- 'Agent Whiteboard'
  Agent avec qui communiquent au moins avec 2 utilisateurs. Possibilité de chat et de partage d'un whiteboard pour dessiner.

- 'Agent Maintenance'
  Agent qui sert à synchroniser les versions des différents agents sur les serveurs.

- 'Agent Recherche d'URL' (SSAc)
  Cet agent interroge un moteur de recherche externe qui lui renvoi des URL possibles. L'agent s'occupe ensuite de vérifier la validité des sites en cherchant des mots-clefs.

- 'Agent Post-it'
  Permet de laisser un message à un utilisateur ou à différents groupes d'utilisateurs.

- 'Agent Recherche personnes connectées'
  Recherche le status d'une(de) personne(s) connectée(s) et les informations le concernant sur le réseau. Option : ajouter un trombinoscope.

- 'Agent Base de Connaissance'
  S'occupe d'un base de données permanente de documents et permet le dialogue avec les autres agents.

5.2 Les données sur un serveur

Un serveurs possède les données suivantes :

- Liste des autres serveurs
  Un serveur possède une liste de serveurs en activité et est capable d'apprendre l'existence d'autres par les informations que transportent les agents (leur origine, leur destination,...).

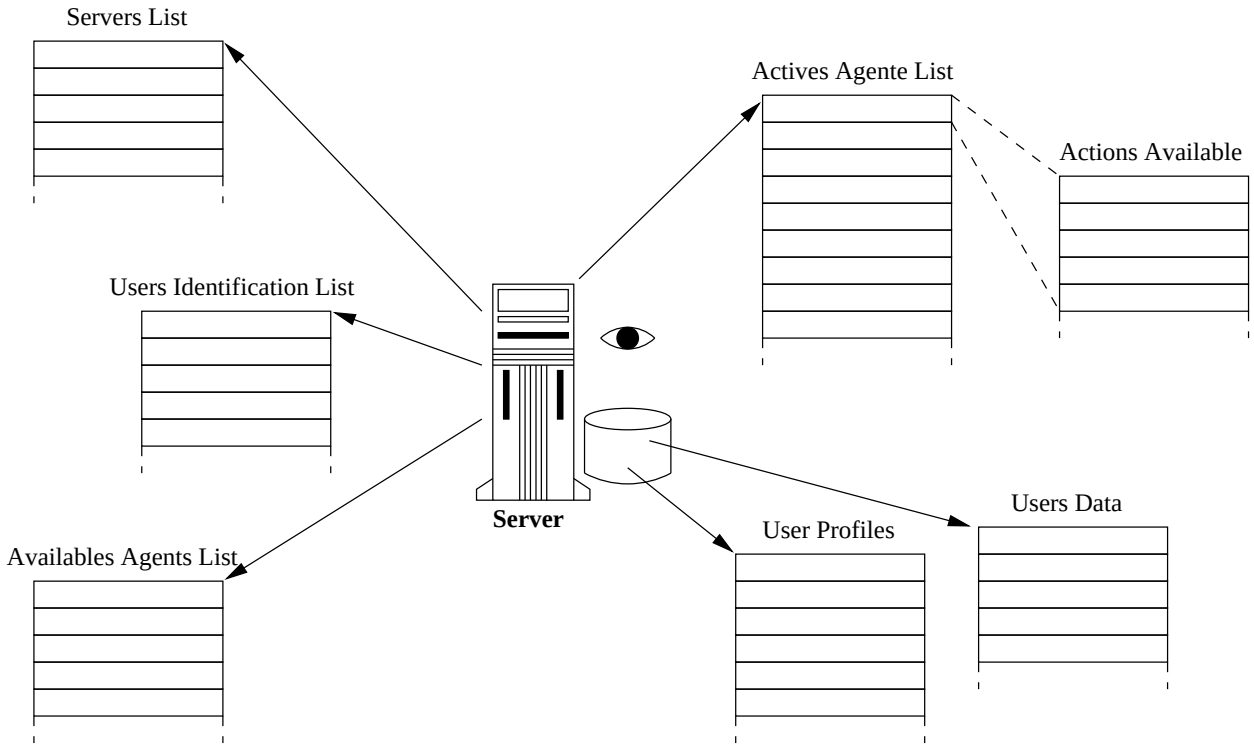
- Liste des agents actifs
  La liste des agents en activité présents chez lui.
  Liste des capacités d'action de chaque agent.

- Liste des agents disponibles
  Les agents "non occupés" dont dispose le serveur. Que ce soit des agents avec interface (Serveur-Client) ou sans interface (Serveur-Serveur).

- Un registre des identifications concernant les utilisateurs

- Base de Données des profils utilisateurs qui lui sont connectés ou qui y étaient connecté lors de leur dernier 'login'.

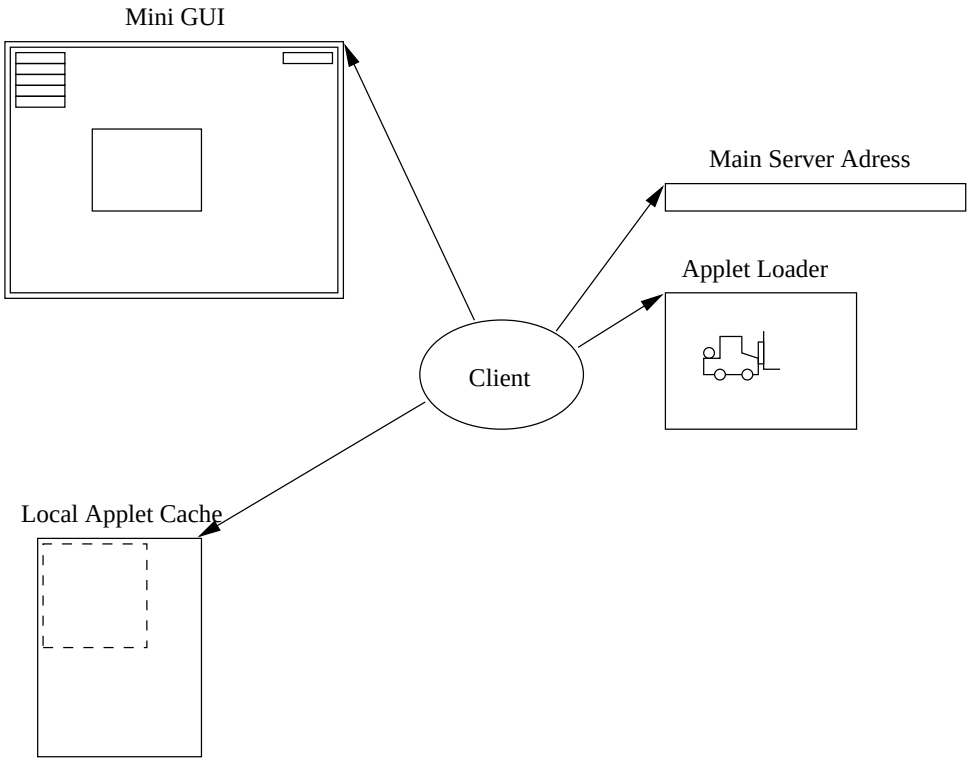
- Base de données des utilisateurs
  Données collectées/traitées par les agents pour l'utilisateur.



5.3 Le client

L'application client est le seul programme nécessaire sur un poste pour permettre à un utilisateur de se logger sur un serveur. Elle contient :

- une interface minimale pour établir une connexion (login avec password).
- l'adresse du serveur principal qui se chargera de rediriger le client vers le serveur le plus proche.
- les moyens de télécharger des applets (profil utilisateur notamment).
- Un cache local d'applets pour éviter les chargements incessants.


Mini GUI
Main Server Adress
Applet Loader
Client
Local Applet Cache

Chapitre 6

Organisation

6.1 Objectifs de chacun

On le remarque facilement, notre projet est vaste et complexe. Il faut donc planifier. C'est pourquoi nous nous sommes donné des objectifs et des échéances.

Nous nous sommes séparé les taches comme suit :

Taches/Noms	Darche	Pierre	Salvy	Souliez	Varroy
CORBA	•			•	
Voyager		•			•
Clients	•		•		
Base de Données			•		•
Agents Intelligents		•		•	•
Serveurs	•	•		•	

Grace à notre expérience du microprojet de GLPOO, nous savons qu'avec définition précise des modules et des entrées/sorties, le développement de chaque partie peut êter fait de façon séparée puis assemblé facilement.

Comme on peut le voir, nous avons toujours veillé à être au minimum deux sur chacun des domaines clés du projet. Ainsi cela nous a facilité la compréhension de chacun des domaines qui nous étaient nouveaux. Ceci permet également de faire des doubles vérifications, ce qui est un bon système d'élimination des fautes.

6.2 Echéancier

Pour la première partie du T.P.E.D., ayant lieu en Juin 1998, nous nous sommes essentiellement concentrés sur l'étude du projet, c'est à dire sa conception et sa faisabilité.

Pour la conception il nous a fallu faire une recherche sur l'existant pour voir ce qu'il se faisait du même genre. Ensuite nous avons cherché tous les outils qui nous étaient nécessaires. Puis il a fallu comprendre le fonctionnement des outils que nous avons sélectionné.

Pour la période de Juillet-Aout 1998 nous allons nous livrer à des tests sur nos machines personnelles, de maitère à être prêts à implanter des solutions sur les machines de l'école à la rentrée scolaire.

On peut retrouver les échéances précédemment évoquées regroupées dans la table suivante :

dates	échéances
20 Juin 1998	Fin de l'étude des outils informatiques
Début Septembre 1998	Fin des tests sur nos machines personnelles
Octobre 1998	Implémentation de Voyager et CORBA
Novembre 1998	Mise en place des agents personnels
Mi-Novembre 1998	Mise en place de l'agent Intelligent "prise de rendez-vous"
Décembre 1998	Mise en place de l'agent "white board"
Janvier 1999	Mise en place des autres agents
Début Février	Prise en compte des connections/déconnections de serveurs
Mi-Février 1999	Phase de tests finale

La nouveauté du programme de 5ième année risque peut-être de boulverser notre échéancier. C'est pourquoi les objectifs premiers de notre T.P.E.D. seraient remplis quand bien même nous n'irions pas au-delà du stade *Mise en place de l'agent Intelligent "prise de rendez-vous"*. Notre échéancier est donc à la fois réaliste et élastique, afin de pouvoir mieux gérer les imprévus.

Annexes

Glossaire

agents mobiles
Des agents mobiles sont des agents capables de se déplacer de serveurs en serveurs sur le réseau suivant leurs besoins.

applet
Une *applet* est une application Java qui a été compilée pour une utilisation dans un document HTML. Ces applications sont encapsulées par une couche de sécurité afin ne pas être corrompues par d'autres programmes.

attribute (IDL)
That part of an IDL interface that is similar to a public class field or C++ data member. The idltojava compiler maps an OMG IDL attribute to accessor and modifier methods in the Java programming language. For example, an interface ball might include the attribute color. The idltojava compiler would generate a Java programming language method to get the color, and unless the attribute is readonly, a method to set the color. CORBA attributes correspond closely to JavaBeans properties.

bêta (version bêta)
Une application est dite version *bêta* quand elle est en cours de développement. Elle est produite est diffusée gratuitement (le plus souvent) afin que des utilisateurs puissent la tester et apporter leur remarques et permettre un développement satisfaisant. Cela sert aussi d'argument commercial puisque'il donne aux *betatesteurs* l'envie du produit.

chat
Le *chat* est une application permettant à plusieurs personnes sur le réseau de dialoguer en ligne dans une salle virtuelle dite *chatroom*. Les personnes commmuniquent au moyen de texte frappé puis envoyé par le client au serveur de *chat*.

client
Any code which invokes an operation on a distributed object. A client might itself be a CORBA object, or it might be a non-object-oriented program, but while invoking a method on a CORBA object, it is said to be acting as client.

client stub
A Java programming language class generated by idltojava and used transparently by the client ORB during object invocation. The remote object reference held by the client points to the client stub. This stub is specific to the IDL interface from which it was generated, and it contains the information needed for the client to invoke a method on the CORBA object that was defined in the IDL interface.

client tier
The portion of a distributed application that requests services from the server tier. Typically, the client tier is characterized by a small local footprint, a graphical user interface, and simplified development and maintenance efforts.

Common Object Request Broker Architecture (CORBA)

An OMG-specified architecture that is the basis for the CORBA object model. The CORBA specification includes an interface definition language (IDL), which is a language-independent way of creating contracts between objects for implementation as distributed applications.

CORBA object
An entity which (1) is defined by an OMG IDL interface, and (2) for which an object reference is available. Object is also the implicit common base type for object references of IDL interfaces.

DARPA
Defense Advanced Research Projects Agency

data store tier
The portion of a distributed application that manages access to persistent data and its storage mechanisms, such as relational databases.

distributed application
A program designed to run on more than one computer, typically with functionality separated into tiers such as client, service, and data store.

distributed environment
A network of one or more computers that use CORBA objects. Objects are installed on the various machines and can communicate with each other.

Dynamic Invocation Interface (DII)
An API that allows a client to make dynamic invocations on remote CORBA objects. It is used when at compile time a client does not have knowledge about an object it wants to invoke. Once an object is discovered, the client program can obtain a definition of it, issue a parameterized call to it, and receive a reply from it, all without having a type-specific client stub for the remote object.

Dynamic Skeleton Interface (DSI)
An API that provides a way to deliver requests from an ORB to an object implementation when the type of the object implementation is not known at compile time. DSI, which is the server side analog to the client side DII, makes it possible for the application programmer to inspect the parameters of an incoming request to determine a target object and method.

exception (IDL)
An IDL construct that represents an exceptional condition that could be returned in response to an invocation. There are two categories of exceptions: (1) system exceptions, which inherit from org.omg.CORBA.SystemException (which is a java.lang.RuntimeException), and (2) user-defined exceptions, which inherit from org.omg.CORBA.UserException (which is a java.lang.Exception).

factory object
A CORBA object that is used to create new CORBA objects. Factory objects are themselves usually created at server installation time.

idltojava compiler
A tool that takes an interface written in OMG IDL and produces Java programming language interfaces and classes that represent the mapping from the IDL interface to the Java programming language. The resulting files are .java files.

implementation
A concrete class that defines the behavior for all of the operations and attributes of the IDL interface it supports. A servant object is an instance of an implementation. There may be many implementations of a single interface.

initial naming context
The NamingContext object returned by a call to the method orb.resolve_initial_references("NameService"). It is an object reference to the COS Naming Service registered with the ORB and can be used to create other NamingContext objects. See also: naming context

Interface Definition Language (IDL)
The OMG-standard language for defining the interfaces for all CORBA objects. An IDL interface declares a set of operations, exceptions, and attributes. Each operation has a signature, which defines its name, parameters, result and exceptions. OMG IDL does not include implementations for operations; rather, as its name indicates, it is simply a language for defining interfaces. The complete syntax and semantics for IDL are available in chapter 3 of the OMG specification at the OMG web site.

Interface Repository (IFR)
A service that contains all the registered component interfaces, the methods they support, and the parameters they require. The IFR stores, updates, and manages object interface definitions. Programs may use the IFR APIs to access and update this information. An IFR is not necessary for normal client/server interactions.

Internet InterORB Protocol (IIOP)
The OMG-specified network protocol for communicating between ORBs. Java IDL conforms to IIOP version 1.0.

invocation The process of performing a method call on a CORBA object, which can be done without knowledge of the object's location on the network. Static invocation, which uses a client stub for the invocation and a server skeleton for the service being invoked, is used when the interface of the object is known at compile time. If the interface is not known at compile time, dynamic invocation must be used.

Java IDL
The classes, libraries, and tools that make it possible to use CORBA objects from the Java programming language. The main components of Java IDL are an ORB, a naming service, and the idltojava compiler. The ORB and naming service are part of JDK1.2; the idltojava compiler can be downloaded from the Java Developer Connection (JDC) web site.

login
Le *login* est un mot anglais qui sert à désigner la demande de connection à une application. En général, on répond à un *login* en entrant un nom d'utilisateur puis un mot de passe pour autoriser l'accès à une application.

name binding
The association of a name with an object reference. Name bindings are stored in a naming context.

namespace
A collection of naming contexts that are grouped together.

naming context
A CORBA object that supports the NamingContext interface and functions as a sort of directory which contains (points to) other naming contexts and/or simple names. Similar to a directory structure, where the last item is a file and preceding items are directories, in a naming context, the last item is an object reference name, and the preceding items are naming contexts.

naming service
A CORBA service that allows CORBA objects to be named by means of binding a name to an object reference. The name binding may be stored in the naming service, and a client may supply the name to obtain the desired object reference.

object
A computational grouping of operations and data into a modular unit. An object is defined by the interface it presents to others, its behavior when operations on its interface are invoked, and its state.

object implementation See implementation.

Object Management Group (OMG)
An international organization with over 700 members that establishes industry guidelines and object management specifications in order to provide a common framework for object-oriented application development. Its members include platform vendors, object-oriented database vendors, software tool developers, corporate developers, and software application vendors. The OMG Common Object Request Broker Architecture specifies the CORBA object model. See www.omg.org for more information.

object reference
A construct containing the information needed to specify an object within an ORB. An object reference is used in method invocations to locate a CORBA object. Object references are the CORBA object equivalent to programming language-specific object pointers. They may be obtained from a factory object or from the Naming Service. An object reference, which is opaque (its internal structure is irrelevant to application developers), identifies the same CORBA object each time it is used. It is possible, however, for multiple object references to refer to the same CORBA object.

Object Request Broker (ORB)
The libraries, processes, and other infrastructure in a distributed environment that enable CORBA objects to communicate with each other. The ORB connects objects requesting services to the objects providing them.
operation (IDL) The construct in an IDL interface that maps to a Java programming language method. For example, an interface ball might support the operation bounce. Operations may take parameters, return a result, or raise exceptions. IDL operations can be oneway, in which case they cannot return results (return values or out arguments) or raise exceptions.

parameter (IDL)
One or more objects the client passes to an IDL operation when it invokes the operation. Parameters may be declared as "in" (passed from client to server), "out" (passed from server to client), or "inout" (passed from client to server and then back from server to client).

persistence
La *persistence* utilisée par *Voyager* est une méthode voisine de la *sérialisation* pour 'freezer' un objet ou une application Java de la mémoire et de la stocker sur la mémoire de masse.

PIDL (Pseudo-IDL)
The interface definition language for describing a CORBA pseudo-object. Each language mapping, including the mapping from IDL to the Java programming language, describes how pseudo objects are mapped to language-specific constructs. PIDL mappings may or may not follow the rules that apply to mapping regular CORBA objects.

pragma
A directive to the idltojava compiler to perform certain operations while compiling an IDL file. For example, the pragma "javaPackage" directs the idltojava compiler to put the Java programming language interfaces and classes it generates from the IDL interface into the Java programming language package specified.

système propriétaire
Un *système propriétaire* est un système ne fonctionnant que sous un environnement spécifique pour lequel il a été conçu. Exemple : Word, Windows, Mr Salvy, Windows Compaq pour ordinateur Compaq...

Swing
Swing est un ensemble de classes Java permettant l'utilisation d'objets graphiques connus sous les environnements classiques aujourd'hui.

proxy

Un proxy est une application qui s'occupe de la redirection de tout ce qui lui est envoyé. De cette manière, on peut toucher plusieurs entités virtuelles sans connaitre leur emplacement.

pseudo-object
An object similar to a CORBA object in that it is described in IDL, but unlike a CORBA object, it cannot be passed around using its object reference, nor can it be narrowed or stringified. Examples of pseudo-objects include the Interface Repository and DII which, although implemented as libraries, are more clearly described in OMG specifications as pseudo-objects with IDL interfaces. The IDL for pseudo-objects is called "PIDL" to indicate that a pseudo-object is being defined. servant object An instance of an object implementation for an IDL interface. The servant object is registered with the ORB so that the ORB knows where to send invocations. It is the servant that performs the services requested when a CORBA object's method is invoked.

PUSH/PULL
Les méthodes de *PUSH* et de *PULL* définissent 2 manières d'accéder à l'information. *PULL* (le plus commun) : le client va chercher les informations sur le serveur. *PUSH* : le serveur envoi l'information au client.

server
A program that contains the implementations of one or more IDL interfaces. For example, a desktop publishing server might implement a Document object type, a ParagraphTag object type, and other related object types. The server is required to register each implementation (servant object) with the ORB so that the ORB knows about the servant. Servers are sometimes referred to as object servers.

server skeleton
A public abstract class generated by the idltojava compiler that provides the ORB with information it needs in dispatching method invocations to the servant object(s). A server skeleton, like a client stub, is specific to the IDL interface from which it is generated. A server skeleton is the server side analog to a client stub, and these two classes are used by ORBs in static invocation.

service tier
The portion of a distributed application that contains the business logic and performs most of the computation. The service tier is typically located on a shared machine for optimum resource use.

stringified object reference
An object reference that has been converted to a string so that it may be stored on disk in a text file (or stored in some other manner). Such strings should be treated as opaque because they are ORB-implementation independent. Standard object_to_string and string_to_object methods on org.omg.CORBA.Object make stringified references available to all CORBA Objects.

trading
Le *trading* est un mot anglais désigant le marchandage, la négociation. Ici, le *trading* désigne l'emploi d'un algorithme de négociation entre agents pour l'échange de données. Un agent va faire du *trading* avec un autre agent pour arriver (si possible) à un commun accord (établir une date de rendez-vous dans un calendrier par exemple).

URL, Uniform Ressource Locator
Any object on an Internet/Intranet Network is referenced with an URL. It's the location on the net of this file/applet/script,... An URL is written like <*protocole*>://<*host*>.<domain>/<path>.
For example : *http://www.esiea.fr/usr/people/public_html/index.html*

Index

Bibliographie

[1] *http://copeland.smartchoice.com/ laforge/index.html.*

[2] *http://copeland.smartchoice.com/ laforge/index.html.*

[3] *http://crrm.univ-mrs.fr/mannina/mb/agent.htm.*

[4] *http://linas.org/linux/corba.html.*

[5] *http://www4.ncsu.edu/eos/info/dblab/dblab-www/wwwaj/.*

[6] *http://www.agent.org/.*

[7] *http://www.cru.fr/multimedia/.*

[8] *http://www.cs.umbc.edu/agents/.*

[9] *http://www.cs.umbc.edu/kqml/.*

[10] *http://www.isi.edu/isd/AA97/info.html.*

[11] *http://www.meangene.com/java/.*

[12] *http://www.objectspace.com/voyager/.*

[13] *http://www.orl.co.uk/omniORB/.*

[14] *http://www.oroinc.com/downloads/index.html.*

[15] *http://www.riv.be/research/events/acw.html.*

[16] *http://www.sikt.hk-r.se/isl/courses/ddv305/laboration/kqml/index.html.*

[17] *http://www.sikt.hk-r.se/isl/courses/ddv305/laboration/kqml/index.html.*

[18] *http://www.umap.com.*

[19] *http://www.veille.com.*

[20] Olivier Aubert. *Introduction à Perl.* Janvier 1998.

[21] J.Christopher Back. *KQML Users Guide, version 1.1.* August 1993.

[22] Rich Burridge. *Java Shared Data Toolkit User Guide, version 1.3.* Sun Microsystem, June 1998.

[23] Object Design. *Component Based Computing and the ObjectStore Cache-Forrward Architecture.* 1997.

[24] Christophe Bonnet et Jean-François Macary. *Technologies PUSH.* Fi System, Eyrolles, 1997.

[25] Jennifer et Joseph Bigus. *Constructing Intelligent Agents with Java.* Wiley Computer Publishing, 1998.

[26] Andreas Vogel et Keith Duddy. *JAVA Programming with CORBA (Second Edition).* Wiley, 1998.

[27] Jacques Ferber. *Les systèmes Multi-Agents.* InterEditions, 1997.

[28] Sai-Lai Lo. *The omniORB2 version 2.5 User's Guide.* Olivetti & Oracle Research Laboratory, February 1998.

[29] U. Seimet M. Laukien. *ORBacus for C++ and Java Version 3.* 1998.

[30] Bryan Morgan. *Building distributed applications with Java and CORBA.* Dr Dobb's Journal 284, April 1998.

[31] Objectspace. *Voyager Core Technology User Guide version 1.0.0.* 1997.

[32] Objectspace. *Voyager Core Technology User Guide version 2.0 Beta 1.* 1998.

[33] Objectspace. *Voyager Transaction Service Techincal Overview.* 1998.

[34] OMG. *The Common Object Request Broker: Architecture and Specification, Revision 2.2.* February 1998.

[35] J. Edwards R. Orfali, D. Harkey. *The Essential Client/Server Survival Guide, Second Edition.* Wiley Computer Publishing, 1996.

[36] Philip Rousselle. *Dynamic distributed systems in Java.* Dr Dobb's Journal 284, April 1998.

[37] Srimivasan. *Advanced Perl Programming.* O'Reilly, 1998.

[38] Joint Revised Submission. *Submission IDL/Java Language Mapping, OMG TC Document.* March 1997.

[39] The DPSsV Team. *http://www.esiea.fr:8080/ pr9.*

[40] Jay Weber Tim Finin and al. *Specification of the KQML, Agent Communication Language.* June 1991.

[41] Larry Wall. *Programmation en Perl 2e édition.* O'Reilly, 1998.

[42] Mark Watson. *Intelligent JAVA Apllications for the Internet and Intranets.* Morgan Kaufmann Publishers, 1997.

[43] Wong. *Web Client Programming.* O'Reilly, 1998.