SIMES - 961620
Système d'Information Multimédia
Pour l'Environnement Subsaharien

Procédés d'intégration

Deliverable number : D 4
Nature:P
Contractual Date of Delivery: 14 novembre 1998
Task WP6 : Project management

Nom du rédacteur :

Jean-Claude Derniame
INRIA
France derniame@loria.fr

Georges Edouard KOUAMOU
Universté de DSCHANG Cameroun
kouamou@loria.f
gkouamou@uycdc.uninet.cm

Abstract

Ce document décrit une approche possible pour le développement de l'architecture proposée pour SIMES . Ce travail a été réalisé pendant le séjour que Georges Edouard KOUAMOU a effectué au LORIA à Nancy de juin à septembre 1998. Il complète le document de spécifications générales et accompagne le document de conception détaillée de l'interface utilisateur.

This document describes a possible approach for the development of the architecture proposed for SIMES. This work has been realized during the stay of Georges Edouard KOUAMOU at LORIA in Nancy, June to September 1998. It makes complete the document on general specifications and go with the document on detailled conception of Man Machine Interface.

Keywords

Système d'Informations, architecture, intégration, Corba, Java

Une Architecture pour les Environnements Hétérogènes Distribués: Procédés d'intégration

1. Introduction

1. Les environnements distribués sont possibles grâce aux réseaux d'ordinateurs qui permettent la communication entre les différents composants. En général, ces environnements sont caractérisés par l'hétérogénéité, laquelle est due à la différence entre les systèmes d'exploitation et les architectures des machines, l'utilisation d'une multitude de langages de programmation et principalement des modèles de données échangés. De plus en plus, des applications sont conçues pour fonctionner dans de tels environnements. Il est donc nécessaire, voire même urgent, de s'intéresser aux problèmes d'échange d'information que pourraient poser les interactions entre les différents composants d'une application éventuellement conçus pour des tâches indépendantes.

2. On peut distinguer deux niveaux:
- le niveau transport qui s'occupe de l'acheminement des objets tout faisant abstraction de l'hétérogénéité,
- le niveau sémantique qui s'intéresse à la compréhension des informations reçues.

3. L'objectif du présent document est d'identifier les éléments nécessaires á la réalisation de tels environnements et définir une organisation. Avant d'arriver à l'architecture proprement dite, nous allons commencer par détailler tous les prérequis c'est à dire les éléments nécessaires à la conception et à la réalisation d'une architecture répondant aux questions sus citées.

4. Dans la suite du document nous commencerons par un exemple de motivation suivi des problèmes que l'on rencontre selon que l'environnement est distribué ou non. La section 4 présente de façon générale le projet SIMES avec une description fonctionnelle des données à manipuler, la section 5 définit des notions et techniques utiles pour la gestion des objets dans un
environnement distribué, lesquels sont explicitées dans le cadre plus concret d'un middleware en section 6, enfin la section 7 présente avec un minimum de détails possibles l'architecture de SIMES.

Un exemple

- Considérons deux outils qui s'échangent des objets. L'un est situé à l'ORSTOM d'Orléans et l'autre à l'Université de Dschang. Le format des données manipulées par chacun d'entre eux est le suivant:
- Outil de l'Orstom:
- O1 = [nom: string, date: string, color: string, capteur: string, rotation: integer]
- Outil de l'UDs
- O2 = [name: integer, color: integer, sensor: string, date: string]
- On note trois types de différence pour ces deux objets:
  Les synonymes. Ce sont des attributs qui ont la même signification, mais dont les types et les identificateurs sont différents (O1.nom et O2.name).
  Les homonymes. Ce sont des attributs portant les même identificateurs et pouvant être interprétés différemment (O1.date a le format français alors que O2.date a le format anglo-saxon)
  Les attributs manquants. O1.rotation n'a pas d'équivalent parmi les attributs de O2.
- On pourrait bien écrire un convertisseur capable de transformer les objets de type O1 en des objets de type O2 et réciproquement. Ce qui est très facile pour un environnement statique. Mais si l'environnement est appelé à évoluer, par exemple l'ajout de nouveaux

outils ou alors le traitement des requêtes d'un outil extérieur qui voudrait accéder aux objets O1 ou O2, cette opération devient vite fastidieuse, longue et périlleuse.

- Ceci n'est qu'une ébauche du problème, car il en existe bien plus que ceux évoqués ci-dessus, lesquels sont proportionnels à la complexité des attributs attachés aux objets.

2. Le problème

Dans la mesure où l'hétérogénéité liée aux différences de systèmes d'exploitation et d'architectures des machines est en résolue grâce aux propositions de plate-forme communément appelée middleware, l'intégration s'intéresse désormais aux échanges d'informations entre les outils. Deux cas de figure se présentent selon que les outils sont dans un environnement distribué ou no

2.1. Intégration dans un environnement autonome.

Avec la prolifération des logiciels, on ne peut s'empêcher d'acquérir un nouveau produit ou bien une nouvelle version d'un produit que l'on possède. Cet acquisition ne doit pas altérer la cohérence de l'environnement. Et le nouvel outil doit pouvoir échanger des informations avec ses prédécesseurs. En d'autres termes il doit pouvoir accéder aux objets rencontrés dans la structure d'accueil et mettre ses données à la disposition des autres.

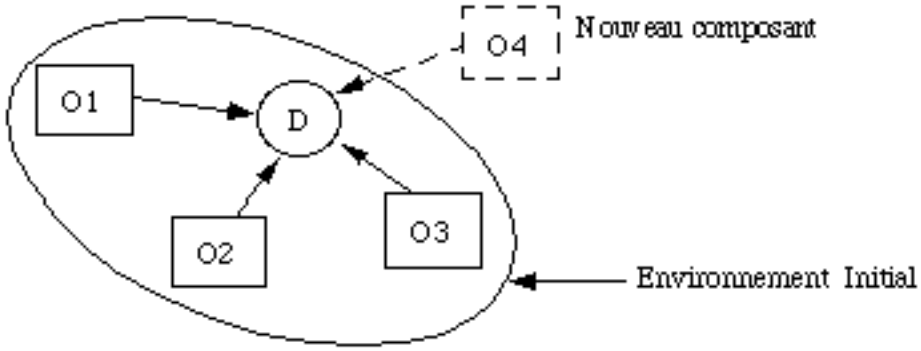


Figure 1. *Intégration d'un outil dans un environnement*

On a une structure initiale composée de trois outils (O1=Excel7, O2=FrameMaker, O3=Word7). On désire acquérir l'outil O4 qui est la version 8 de Word.

Des réflexions ont été menées sur ce sujet et une solution consiste à définir un modèle abstrait de représentation de donnéeset un ensemble de règles de transformation nécessaires pour passer d'un format donné à une représentation canonique.

Toutefois la question qui persiste est "comment mettre bout à bout cette solution avec l'accès distribué?"

2.2. Intégration dans un environnement distribué

Nous sommes en présence d'une cohabitation entre plusieurs environnements situés sur des sites différents. Des objets sont déclarés dans chaque environnement et doivent être accessibles par des outils soit du même environnement, soit d'un distant.

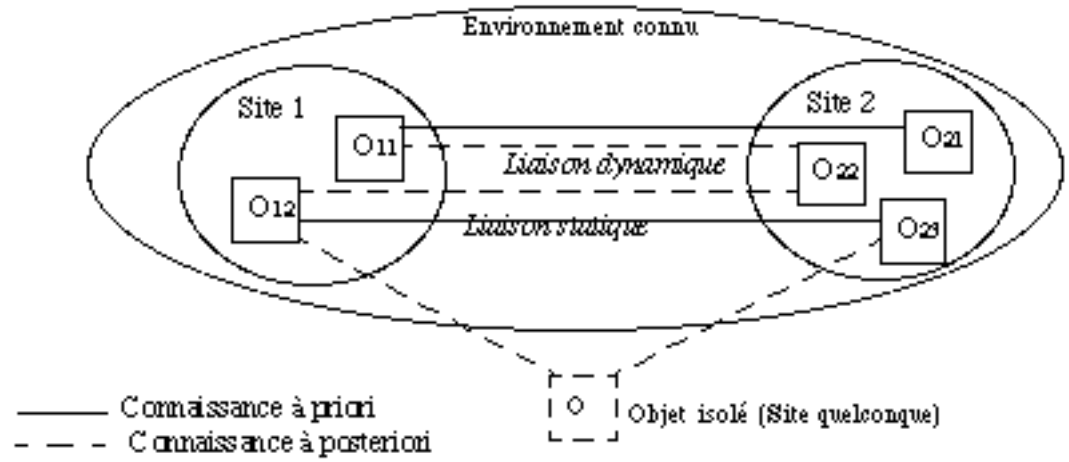


Figure 2. *Type d'invocation dans un environnement distribué.*

Pour un outil, manipulant des objets de formats O1, s'il connaît à priori un objet de format O2, alors il dispose assez d'informations pour effectuer les conversions nécessaires permettant de passer de O2 à O1. Autrement, pour passer du site d'origine au site destination l'objet devra disposer assez de connaissances pour son interprétation.

Tout d'abord, il faudrait penser aux liaisons entre les objets. Ces liaisons peuvent être statiques ou dynamiques. Les liaisons statiques sont d'éventuelles collaborations entre objets identifiées dés la conception et établies pendant la compilation des codes. Autrement elles sont dynamiques. Ces types de liaisons sont utiles pour des objets qui ne connaissent pas à priori leur partenaires, soit parce qu'ils sont nouvellement intégrés dans une structure connu de l'environnement, soit parce qu'ils sont isolés. Des mécanismes d'invocation dynamique doivent être mis en œuvre pour leur permettre l'accès aux objets. Ensuite quels types de connaissances doit contenir un objet afin de faciliter son interprétation?

3. Présentation de SIMES

Le projet SIMES comporte trois phases principales éventuellement menées en parallèle:
- la collecte, l'analyse, l'organisation et le stockage des données,
- le développement et/ou l'acquisition des outils de manipulation de ces données,
- l'utilisation.

Cette section offre une description fonctionnelle des données, suivi d'un aperçu général du développement des outils et enfin l'accès à tous ces outils.

3.1. Les données.

SIMES s'appuie sur la disponibilité des bases de données sur des thèmes variés (environnement, société, économie, santé, météorologie,...) et des données de nature diverse (qualitatives, quantitatives, images, cartes,...). Deux sites pilotes à savoir Mopti au Mali et la vallée du fleuve Sénégal sont retenus pour la collecte des données.

Les modes de collecte sont très variés et dépendent étroitement du thème d'étude. Ainsi les enquêtes sont mieux indiquées pour l'acquisition des données socio-économiques lesquels sont ensuite structurées pour en faciliter la compréhension. Pour les autres thèmes, l'utilisation des systèmes un peu plus complexes est requis à l'exemple des capteurs comme des satellites, des thermomètres qui permettent d'acquérir respectivement des images (météorologiques, sur la végétation) et des températures. En plus de tous ces différents types de données s'ajoutent les documents qui seront produits dans le cadre du projet.

Le traitement, le stockage et la restitution des informations pertinentes à partir de ces données nécessite un panel d'outils développés ou acquis sur le marché des logiciels et installés sur les différents sites retenus.

3.2. Recueil des outils

A chaque type de données correspondent des outils spécifiques destinés à leur production et à leur manipulation sous réserve d'une utilisation à d'autres fins par d'autres outils intéressés. Les principales tâches à exécuter vont de l'analyse des données brutes, en passant par le stockage, jusqu'à la production des informations pertinentes relatives aux indicateurs socio-économiques, climatologiques, environnementaux, etc.

Ainsi on aura à développere ou acquérir:
- des Bases de Données relationnels ou Orientées Objet pour le stockage des données,
- des Systèmes d'Information Géographiques pour la manipulation et stockage des données
- cartographiques,
- des outils de recalage d'image qui manipuleront les images par exemples des photos satellites et aériennes,
- des algorithmes d'analyse des données,
- des algorithmes d'extraction des indicateurs qui manipuleront des images et des données alphanumériques,
- des algorithmes de simulation qui pourront produire des données sous plusieurs formes,
- des outils de production des documents.

3.3. L'utilisation

Bien que les objets soient installés dans un environnement distribué, ils devront être intégrés dans un vaste réseau de manière à être transparent à l'utilisateur. Donc un programme qui se trouve par exemple au Burkina Faso devra éventuellement accéder à des données au Mali afin de répondre aux requêtes des utilisateurs, tout comme un outil qui est au Sénégal pourra solliciter les services d'un autre résidant à Dschang pour mener à bien sa tâche.

Sous un aspect complètement différent, les services de SIMES pourront être invoqués par un outil/utilisateur situé sur un site distinct de ceux où ont été développés les outils. On constate donc que la modification des outils est exclue d'avance d'abord parce qu'il faut en assurer la maintenance, ensuite parce que l'environnement est appelé à évoluer. De nouveaux outils pourront être acquis sur le marché des logiciels et/ou développés sur un site et intégrés à l'environnement.

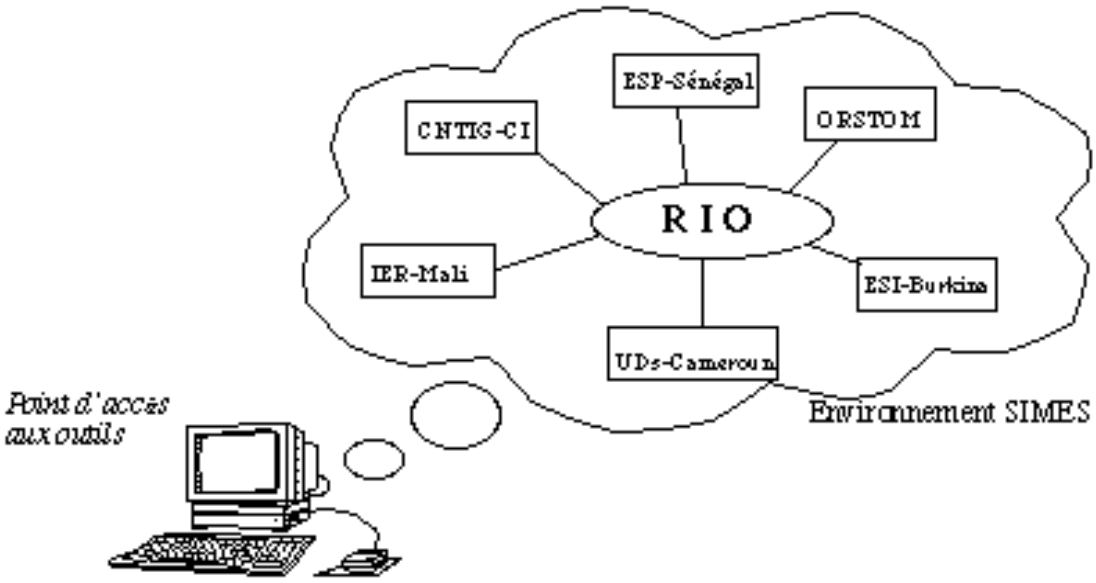


Figure 3. *Transparence d'utilisation à partir d'un poste "isolé".*

SIMES propose un ensemble d'outils génériques installés dans un environnement distribué dont les composants communiquent entre eux par l'intermédiaire du réseau RIO (Réseau Intertropical d'Ordinateurs). Un point d'accès à cette machinerie peut bien un ordinateur, intégré ou non à l'environnement, mais ayant des ressources pour accéder au réseau. L'ordinateur travaille ainsi avec un vaste environnement complètement transparent à son utilisateur.

Au delà, les outils manipuleront des objets différents de part leur nature (coupler une image satellite et les coordonnées d'une région) ou de part leur format (superposer une image Spot et une image de photo aérienne).

Concevoir une architecture pour SIMES, c'est prévoir:
une libre circulation des objets,
des possibilités de liaisons dynamiques entre les sites et donc entre les objets,
un processus de transformations des objets incluant le contrôle des tâches.

Avant d'en arriver là, nous essayerons d'établir un recueil de concepts et de techniques disponibles permettant d'assurer et de réaliser ces prévisions. En effet, l'intégration des objets dans un état brute ne facilite pas leur manipulation dans un environnement distribué.

3.4. Concepts et techniques de gestion des objets

Les objets doivent être déclarés dans l'environnement afin d'être connus des autres et ils méritent d'atteindre les services dont ils ont besoin. Pour un objet intégré, on doit pouvoir le localiser, déterminer les services dont il offre à ses clients et les moyens de les invoqués. Le processus de recherche d'un service doit être automatisé au mieux afin d'éviter à l'utilisateur des interventions inconnues et complexes.

Pour cela chaque objet devra disposer des éléments de connaissances généralement appelé meta-données, pour permettre à ses clients de l'identifier et/ou d'accéder à des ressources supplémentaires par exemple lorsqu'une transformation s'impose.
Une pratique assez courante pour ce genre d'opération est l'encapsulation.

3.4.1. L'encapsulation

L'encapsulation consiste à construire une couche supplémentaire de logiciel enveloppant un objet ou un ensemble d'objets. cette couche constitue une interface d'échange entre l'objet et l'extérieur. Elle offre éventuellement une description de la structure de l'objet et un ensemble de services accessibles depuis l'extérieur.

Considérons un texte FrameMaker dont on voudrait convertir dans trois formats différent pour en assurer la portabilité. Les trois formats retenus sont RTF pour la lecture dans Word, HTML pour l'exploiter avec un browser, PostScript pour l'afficher avec Acrobat Reader.

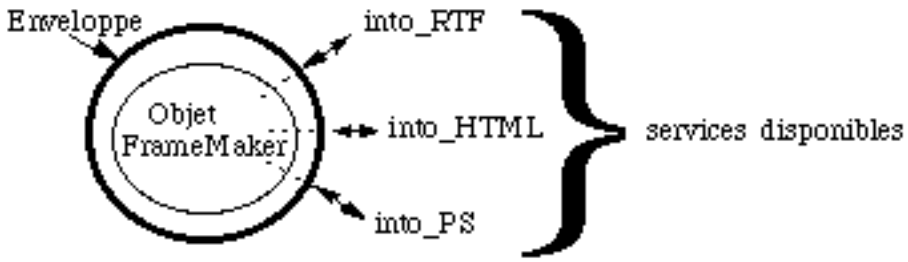


Figure 4. *Encapsulation d'un objet FrameMaker.*

1. Une façon de faire, est d'envelopper l'objet FrameMaker. A cette enveloppe on associe les opérations de conversion, into_RTF, into_HTML, into_PS, pour obtenir en sortie respectivement un objet Word, un objet HTML ou un objet Acrobat.

2. L'encapsulation est une caractéristique essentielle des environnements Orientés Objets dans la mesure où ils offrent des propriétés de contrôle permettant de limiter la visibilité de l'objet enveloppé. C'est à dire que l'accès peut se faire absolument par l'intermédiaire de l'interface ou bien il peut être directe. Dans le cas des langages comme C++ et Java, on utilise les propriétés: public, protected et private.

3. Considérons l'exemple précédent relatif à l'objet Framemaker. La définition suivante spécifie l'interface de l'enveloppe d'un tel objet:

- Class FrameMaker{
- private :
- char *objName;
- protected :
- delete();
- public :
- psObj_ptr into_PS();
- rtfObj_ptr into_RTF();
- htmlObj_ptr into_HTML();
- };

Private

l'attribut est visible seulement par les instances de la classe. Tout autre objet doit passer absolument par les opérations de l'interface pour y accéder.

Protected

l'opération est visible par les instances de la classe et celles des classes dérivées. Tout autre objet doit passer par les opérations de l'interface.

Public

Ce sont les opérations offertes á tous les objets. Il n'y a pas de restriction.
Lorsque l'objet encapsulé est une application, il serait intéressant que son action soit complètement transparente á son client et aux utilisateurs. On n'exécute pas entièrement l'application, mais juste la partie concernée par le service souhaité. Les applications adaptées sont celles qui disposent d'un interface de programmation (API). Dans ce cas, l'interface transmet les paramètres à l'application qui s'occupe du traitement et les résultats sont rendus au client par l'intermédiaire de l'interface.
Comme exemple qui est la suite du précédent, le comportement de l'opération into_HTML pourrait faire appel à la fonctionnalité de FrameMaker qui offre la création des documents HTML à partir d'un document FrameMaker.

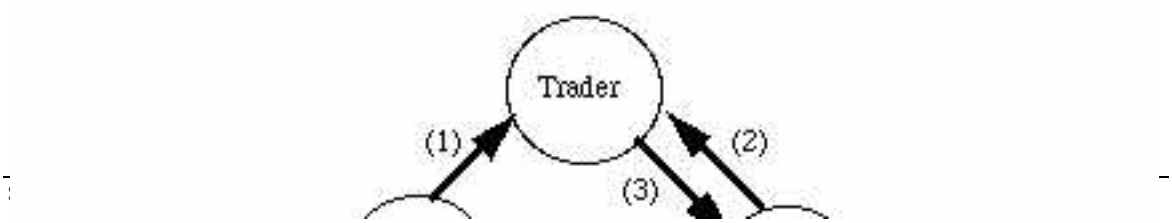
```
htmlObj_ptr into_HTML() {
/*
Appeller MIF2HTML <options> <entrée.doc> <sortie.html>
*/
}
```

3.4.2. Mécanisme de "Trading"

Le "trading" est une technique qui consiste à donner à un objet la possibilité de découvrir les services adaptés à ces besoins.
Trois types d'objets interviennent dans ce mécanisme de recherche: le "Trader", l'"Exporter", et l'"Importer".

Le trader est l'objet qui emmène les autres objets à se découvrir dans un environnement distribué, c'est le support des services de trading. L'exporter est un objet qui décrit un service, en donnant son nom et ses propriétés, et la localisation de son interface. Il peut être le titulaire du service ou alors il peut annoncer au compte d'un autre objet. L'importer est un objet qui peut demander au trader des services ayant une propriété donnée. Il peut être le client potentiel du service ou bien il peut l'importer un service au compte d'un autre objet. Le mécanisme se fait suivant les étapes ci-après:
l'exporter enregistre auprès du trader la description d'un service et la localisation de son interface,
l'importer sollicite du trader un service en lui précisant les caractéristiques,
le trader vérifie les descriptions des services dont il dispose et répond à l'importer en lui donnant la localisation du service,
l'importer interagit avec l'objet propriétaire du service.

Il est possible que les traders communiquent entre eux. Lorsque un Trader est lié à d'autres, les services de ces Traders sont implicitement disponibles pour ses clients, ainsi la requête d'un importer pourrait être traitée par un ou plusieurs traders. Les traders interconnectés forment un graphe orienté appelé graphe de trading.



4. Présentation d'un middleware: CORBA

Les architectures middleware ont pour but de résoudre les problèmes d'hétérogénéité que posent les environnement distribués. CORBA en est une proposition de ce type d'architecture. Spécifié par l'OMG, il est fondé sur le modèle objet.
Nous essayons dans cette section de présenter la structure et le fonctionnement de cette architecture et de déterminer, relativement aux technique sus-citées, les composants et les services utiles qui permettent de gérer les objets.

4.1. Structure et fonctionnement

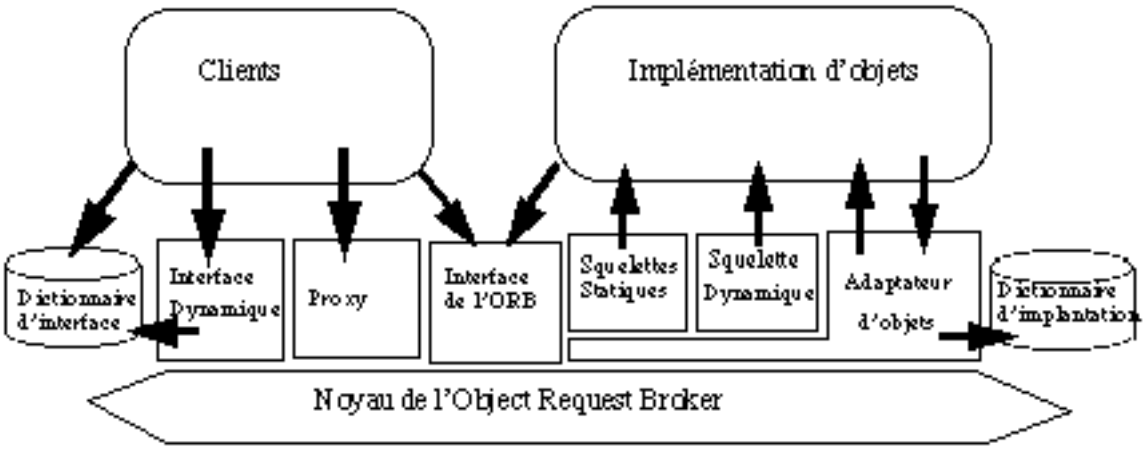


Figure 6. *L'anatomie de l'ORB*

L'Object Request Broker (ORB) est responsable de toutes les interactions entre Clients et Objets. Il doit être vu comme un ensemble logique de services plutôt qu'une bibliothèque ou un processus particulier. Un objet est une entité identifiable et encapsulée qui offre des services qui peuvent être demandées par un client.

On ne peut parler de client que par rapport à un objet dont il a accès à la référence. Il connaît uniquement la structure logique de l'objet relativement à son interface et teste le comportement de l'objet à travers les invocations. Par conséquent l'implémentation d'un objet peut être le client d'autres objets.

Pour formuler une requête, le client peut utiliser l'interface d'Invocation Dynamique ou les Proxy IDL. Il peut directement interagir avec l'ORB pour des fonctions particulières. Du côté de l'objet, il peut recevoir la requête à travers le Squelette Statique ou le Squelette Dynamique. Pendant le traitement de la requête l'implantation de l'objet fait appel à l'Adaptateur d'Objets de l'ORB.

Le client accède à la référence de l'objet pour connaître le type d'objet et les opérations offertes. Cet accès fait appel à l'un des Interfaces d'Invocation (Statique ou Dynamique). Les requêtes statiques et dynamiques satisfont la même sémantique, donc le récepteur ne peut faire aucune différence entre ces deux types de requête.

L'ORB localise l'implémentation appropriée, transmet les paramètres et transfert le contrôle à l'implémentation d'objet par le squelette dynamique ou statique. Pour traiter la requête, l'implantation d'objet sollicite des services de l'ORB par l'Adaptateur d'objet. Les résultats sont transmis au client dès que la requête est terminée.
L'implantation d'un objet offre la sémantique de l'objet, en définissant les données pour les instances et le code des méthodes de l'objet. Toutefois il est possible d'utiliser des logiciels additionnels ou d'autres objets pour implémenter le comportement de l'objet, par exemple un programme par méthode, des bibliothèques, une BDOO, etc

4.2. L'IDL et le dictionnaire d'interface (IR)

L'IDL et l'IR constituent la partie méta-donnée de CORBA. L'IDL est responsable de l'encapsulation des objets. Il est utilisé pour décrire l'interface auquel ont accès les clients.

Toutes ces informations sont stockées dans l'IR.

4.2.1. L'Interface Definition Language (IDL)

C'est un langage de spécification qui offre très peu de détails sur l'implantation mais des correspondances vers différents langages de programmation peuvent être définies. Par conséquent, le client ne peut être écrit en IDL mais dans l'un des langages pour lequel la correspondance des concepts de l'IDL sont définies. L'IDL permet l'interopérabilité entre clients et serveur d'objets écrit dans différents langages de programmation. Pour un objet implémenté en C, C++, ou Java il suffit de spécifier son interface en IDL qui est exporté vers des clients écrits dans un langage éventuellement différent.

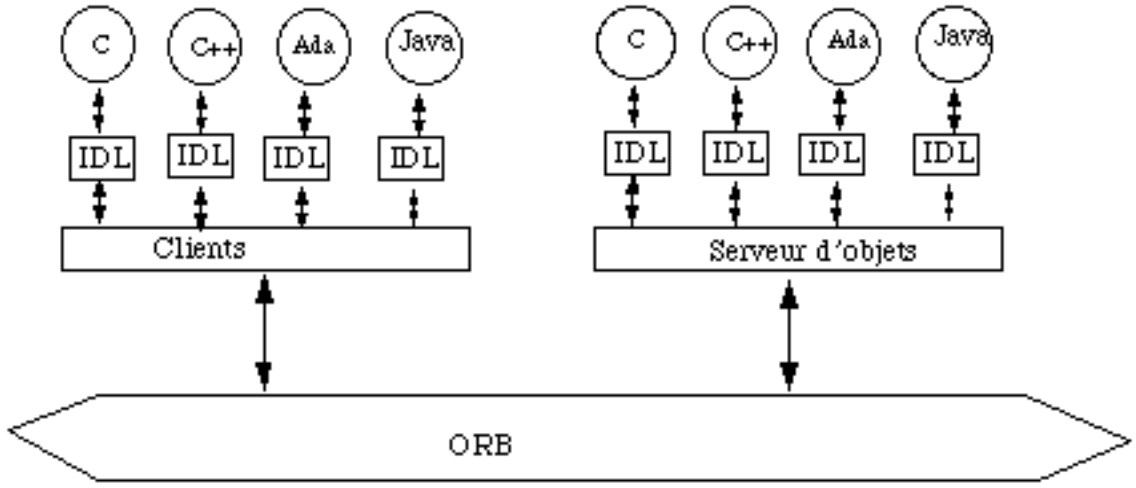


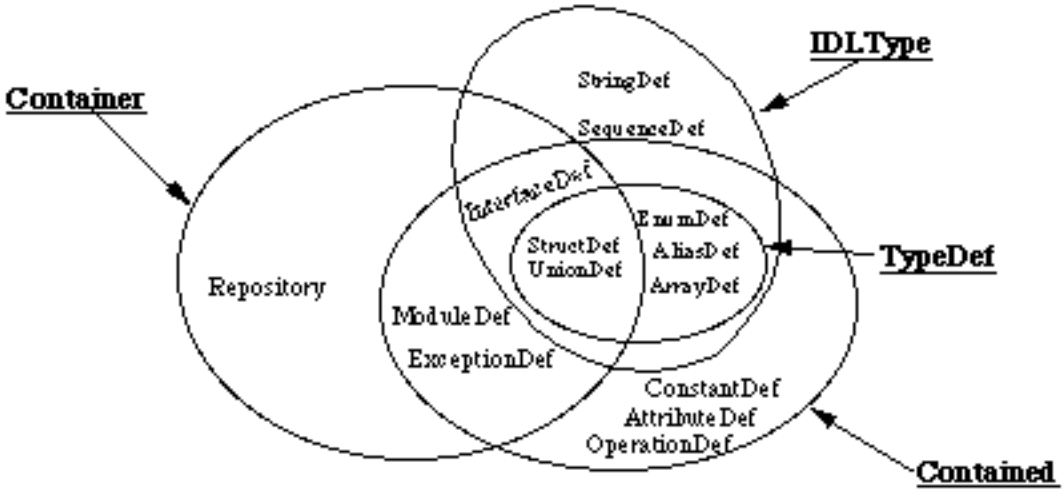
Figure 7. *L'interopérabilité Client/Serveur offert par l'IDL*

des
e de
des
opérations. Elle supporte la syntaxe des constantes, des types et de la déclaration des opérations, mais elle n'inclut pas la définition des variables et des structures algorithmiques.

4.2.2. Le Dictionnaire d'Interface (IR: Interface repository)

L'IR est la composante de l'ORB qui offre le stockage des objets persistants que représentent

les définitions d'interfaces écrits en IDL. Il contient des objets CORBA dont l'élément minimal est IRObject c'est à dire que tous les autres objets héritent de ses propriétés. On peut regrouper ces objets sous forme d'ensembles et d'éléments dont la relation d'appartenance et/ou d'inclusion symbolisent l'héritage.



4.2.3. L'Interface d'Invocation Dynamique (DII)

Cette composante n'est pas des moindres. Nous avons vu que les clients pouvaient être amenés à découvrir les objets pendant l'exécution, eh bien l'interface dynamique permet d'établir de telles liaisons.

Tout comme les proxys, l'Interface d'Invocation Dynamique (DII) est utilisé pour invoquer des requêtes sur l'objet à la seule différence que ces requêtes sont créées dynamiquement. Un client devrait préciser l'objet, l'opération à exécuter et un ensemble de paramètres pour cet opération.

Les requêtes sont définies en terme de pseudo-objet Request. Les opérations sur les requêtes sont définies comme des méthodes sur un objet.
Create_request: C'est une méthode de l'interface Object. Cette opération crée un pseudo-objet Request. Elle est exécutée sur l'objet destination i.e qui reçoit la requête. Plusieurs syntaxes cohabitent dont la plus othodoxe est la suivante:

```
Status create_request(
in Context ctx, // contexte
in Identifier operation // nom de l'opération sur l'objet
in NVList arg_list, // les arguments de l'opération
inout NamedValued result, // le résultat de l'opération
out Request request, // nouvelle requête créée
in Flags req_flags // contrôle de l'allocation mémoire
 ;)
```

Les autres opérations, définies ci-dessous, agissent sur l'objet "Request" et permettent de le modifier (ajout d'arguments et suppression de la requête), de l'envoyer ou de recevoir la réponse.
Add_arg: ajoute incrémentallement des arguments à la requête. Les arguments qui ne sont pas spécifiés pendant la création de la requête sont ajoutés en utilisant add_arg. Toutefois l'utilisation des deux méthodes à la fois pour spécifierles arguments de la même requête n'est pas encore acceptable.
Delete: détruit la requête et libère la mémoire allouée à ses arguments.
Invoque: Cette opération appelle l'ORB qui exécute la méthode appropriée. Si le résultat est

correct, il est placé dans la variable result spécifiée dans la requête. Le comportement est imprévisible si la requête a été utilisée précédemment avec invoque, send ou send_multiple_request.
Send: initie une opération en se basant sur les informations contenues dans la requête. Send retourne le contrôle au client sans attendre la fin de l'exécution de l'opération.
get_response et *get_next_response* permettent au client de savoir si l'exécution de la requête est achevée.

4.2.4. Le service Trading de CORBA

Dans la spécification du service de Trading, le comportement d'un trader CORBA est guidé par des lignes de conduite spécifiques. Elles sont représentées sous la forme de paires <nom, valeur>.

L'importer spécifie le type de service qu'il recherche en précisant les propriétés (une propriété est une paire <nom, valeur>) avec les contraintes et les préférences possibles. Une contrainte est une expression régulière écrite dans un langage de contraintes.

L'Exporter publie ses services en donnant le nom du service, la référence de l'objet qui l'assure et éventuellement des valeurs des propriétés du service.

Des interfaces fonctionnelles sont offerts pour manipuler les concepts mentionnés précédemment.
- L'Exporter (resp. l'Importer) peut interroger le Trader en utilisant les opérations offertes par l'interface Register (resp. Lookup).
- L'interface Link permet de gérer le graphe de Trading. il offre des opérations d'édition et de description des liens entre les Traders.
- Le dictionnaire des types qui est dédié à la gestion des types de service.

5. L'architecture de SIMES

SIMES est une application distribuée qui intègre des objets divers et hétérogènes. Les outils composants cette application devront être apte à s'échanger des objets et des informations de contrôle tout faisant abstraction de toute forme de différence. Ses principales caractéristiques sont: la souplesse, l'adaptabilité et l'évolutivité.

La souplesse

Répondre aux sollicitations extérieures c'est à dire accepter et traiter les requêtes soumis par un système extérieur à l'environnement.

L'adaptabilité

L'ajout d'un nouveau composant ne doit pas entraver le bon fonctionnement de l'ensemble. Le nouvel élément doit participer pleinement aux activités du groupe, accéder aux objets existants et mettre ses services à la disposition des autres.

L'évolutivité

Les outils doivent être maintenus tout comme de nouveaux outils peuvent être acquis et intégrés dans l'environnement.

La structure de base sur laquelle repose notre architecture c'est CORBA. Il est utilisé comme bus d'objets: toutes les interactions entre clients et objets, tout échange d'objets passsera par l'ORB.

Dans la couche externe, au niveau de chaque site, ce sont des agents intelligents. On pourrait ainsi disposer:

- d'un agent qui gère les images satellites avec un codage des photos satellites,
- d'un agent qui gère les bases de données,
- d'un agent qui gère les cartes avec un codage des SIG,
- d'un agent qui gère les images de prises de vue aérienne,
- d'un agent qui gère les documents,
- d'un agent qui produit les indicateurs.

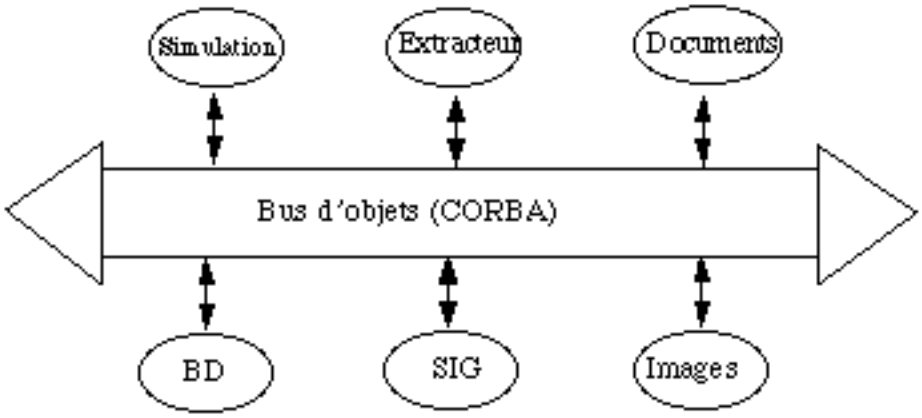


Figure 9. *Architecture générale*

Un agent est une abstraction représentant un ou plusieurs objets encapsulés, qui fournit des services déclarés dans l'environnement. Il a de la compétence/un savoir-faire qui est l'ensembles des services dont il dispose, et des connaissances sur son environnement c'est à dire l'ensemble des services offerts par les autres.

6. Conclusion et Perspectives

Nous disposons d'une implémentation de CORBA appelé MICO en deux versions: Windows NT et Solaris.

L'installation sur d'autres plate-formes est possible puisque les sources sont disponibles. Elle comprend l'ORB et ses principaux composants à savoir l'IDL, l'interface d'invocation dynamique, le squelette d'invocation dynamique, des services de gestion des objets. Pour un début, il faudra y ajouter les services utiles pour SIMES comme celui de Trading.

La prise en compte des agents requiert des notions supplémentaires sur les techniques de conception et de structuration des connaissances dans les systèmes multi-agents. L'objectif est d'étendre MICO avec les fonctionnalités des SMA.

Au niveau des agents, l'aspect donnée devra être examiné.