REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS-SETIF

MEMOIRE

Présenté à la Faculté des Sciences
Département d'Informatique
Pour L'Obtention du Diplôme de

MAGISTER

Option : **Matériel et Logiciel**

Par

Mr. SAIFI ABDELHAMID

THEME

ANALYSE DES PLATEFORMES DISTRIBUEES EN VUE D'UNE APPLICATION DE E-MAINTENANCE

Soutenu le : 21/09/2011

Devant le jury

Président :	**Dr. BOUKERRAM ABDALLAH**	**M.C.A**	**U.F.A. Sétif**
Rapporteur :	**Dr. MOSTEFAI MOHAMMED**	**PROF**	**U.F.A. Sétif**
Examinateur :	**Dr. ALIOUAT MAKHLOUF**	**M.C.A**	**U.F.A. Sétif**

Remerciements

Je voudrais de profiter de cet espace pour exprimer toutes mes gratitudes et mes reconnaissances à mon encadreur et enseignant Mr MOSTEFAI mohamed qui n'a pas cessé de m'encourager et de me guider ver la bonne voie tout au long de la réalisation de ce travail.

Je tiens également à remercier M. BOUKERRAM Abdallah, Maître de Conférence à l'université de Sétif, qui m'a fait l'honneur de présider le jury,

Ainsi que M. ALIOUAT Makhlouf, Docteur à l'Université de Sétif pour avoir accepté de faire partie de ce jury.

Sans oublier, un grand merci à Mr TOUAHRIA mohamed et BENAOUDA Abdelhafid pour ces aides et ces conseils.

Enfin, je salue l'ensemble des professeurs et étudiants du département informatique.

SAIFI Abdelhamid

Dédicaces

À la mémoire
De mon chère père.

À mon épouse Sihem,
A mes enfants Aya, et Ahmed

À ma chère mère.
Et à mes soeurs et frères.

Table de matières

Chapitre IV -DCOM : Distributed Component Object Model

Deuxième partie : Applications et évaluation

Chapitre V -: Etude comparative et de synthèse

Chapitre VI -: la E- Maintenance et SMA

Chapitre VII -: Modélisation des agents en SMA

Chapitre VIII -: Implémentation et mise en ouevre

Introduction Générale

Les systèmes distribués occupent une place de plus en plus importante dans l'informatique actuelle, il est aujourd'hui possible de faire travailler de façon coopérative de plusieurs personnes distantes

La première étape de cette évolution a été un support de middleware orienté objet permettant de manipuler des objets distants de façon transparente.

Cette évolution à donner la naissance des plates-formes capables d'exploiter les ressources et les traitements distants, et de nouveaux termes sont apparus comme l'intelligence artificielle distribuée.

Le développement d'une application répartie ne se fait pratiquement jamais à partir de zéro, c'est-à-dire directement au dessus d'un système d'exploitation et de ses primitives de communications de bas niveau comme les sockets. On préfère utiliser une plate-forme logicielle (ou middleware), située entre le système d'exploitation et l'application, et qui fournit des abstractions de haut niveau définissant un modèle de programmation pour les applications.

Il existe plusieurs types de modèles de programmation, les principaux étant le modèle client serveur,basé sur l'appel de méthode à distance synchrone,ou basé sur la diffusion d'événements (messages asynchrones). Java RMI, CORBA, et DCOM sont des exemples de plates-formes industrielles qui offrent un modèle de programmation de type client - serveur

Dans ce sens , Le but de ce mémoire est d'étudier ,présenter et évaluer les différentes plates formes distribuées , sur la base de cette évaluation, nous proposons une plate forme suffisamment souple et efficace pour réaliser une implémentation d'un système multi-agents de e-maintenance

Pour atteindre cet objectif, le présent mémoire est structuré en deux parties :

La première partie, c'est un état de l'art et de synthèse concernant les plates formes distribuées, elle est divisée en quatre chapitres :

Chapitre I : pour identifier les contraintes et les besoins attendus par un middleware. Il dégage les notions fondamentales qui doivent être proposées par un tel middleware, et il identifie les principaux services nécessaires et indispensables doit être assurés.

Chapitre II : est consacré à étudier la norme CORBA, il présente ses avantages et ces concepts, et en fin la mise en place de telle plate-forme

Chapitre III : dans ce chapitre, nous allons voir les caractéristiques de la plate forme RMI, et nous illustrons le processus de conception de développement d'une application RMI

Chapitre IV : sera réservé au plate-forme DCOM proposé par MicroSoft, nous nous détaillerons les différents aspects de ce modèle, et nous finissions par donner une mise en place d'une application DCOM

La deuxième partie de ce mémoire, basée sur le choix d'une plate-forme distribuée et son application à la e-maintenance par un système multi-agents, elle est divisée en quatre chapitres :

Chapitre V : constitue le cœur de ce mémoire, il comporte une étude comparatif et de synthèse entre les différentes plates-formes de Middleware, c'est la conclusion finale de la première partie

Chapitre VI: nous consacrons ce chapitre pour les descriptions des notions de base de SMA, et le problème d'équilibre de charge dans une entreprise étendue.

Chapitre VII : consiste à donner les différente scripts des agents de notre modèle d'équilibre de charge coopératif préventif

Chapitre VIII : ce chapitre présente l'implémentation et la mise en œuvre d'une application basée sur les agents pour régler le problème d'équilibrage de stock

PARTIE I
ETAT DE L'ART

Le but de cette partie est d'étudier la mise en place de différentes politiques de middleware adaptées aux environnements répartis, caractérisées par leur grande échelle pour la structuration des applications en objets.

Pour illustrer notre démarche et proposer une plate-forme de validation, nous avons développé un état d'art après une étude bibliographique approfondie sur les différentes approches, et un test de mise en place et de performance de différentes plates formes

Mot clé :

Middleware, architecture distribuée, système réparti, réseaux, CORBA, RMI, DCOM

CHAPITRE I

LES MIDDLEWARES ETAT DE L'ART

Le but de ce chapitre est de montrer l'importance du middleware comme technologie permettant de répondre aux besoins actuels de l'industrie, ces besoins s'expriment en termes de distribution et de coopération entre applications ou entre des composants d'application

Ainsi, différentes technologies réalisant le middleware sont représentées et analysées, il apparaît que leurs utilisations influence profondément l'architecture globale d'un système informatique, ce qui implique un changement de culture puisqu'ils modifient la structure des applications.

Mot clé :

Middleware, architecture distribue, système réparti, réseaux, Client/ Serveur

Chapitre I

LES MIDDLEWARES

ETAT DE L'ART

I-1- Introduction

Une application à objet distribué :

- C'est une application découpée en plusieurs objets.
- Chaque objet peut être placé sur une machine différente.
- Chaque objet peut être s'exécuté sur un système différent.
- Chaque objet peut être programmé dans un langage différent.
- Chaque objet peut communiquer avec d'autres objets de l'application.

Cette communication est assurée par un système à objet distribué, le but de tel système est :

1- D'invoquer une méthode d'un objet se trouve sur une autre machine exactement de la même manière que s'il se trouvait au sien de la même machine.
2- L'utilisation d'un objet distant, sans savoir ou se trouve, en demandant à un service de renvoyer son adresse.
3- De pouvoir passer un objet distant en paramètre d'appel à une méthode locale ou distante.
4- De pouvoir récupérer le résultat d'un appel distant sous forme d'un nouvel objet qui aurait été crée sur la machine distante.

Pour atteindre ce but, il y a des défis à relever :

- La communication : transmettre les informations entre les composants logiciels.
- L'hétérogénéité : gérer la diversité des matériels et logiciels en interaction.
- L'intégration : introduire des nouveaux composants, et construire les liens entre les différents composants.
- L'interopérabilité : échange des données entre applications distantes.

Vu les problèmes à résoudre pour construire des applications distribuées, la solution qui était proposée et maintenue, l'utilisation d'un mécanisme fait intervenir des objets intermédiaires constituant une couche logicielle appelé middleware

I-2- Le middleware

Le middleware[1] est un bus à objets répartis, peut être constitué d'un grand nombres de programmes, fonctionnant éventuellement sur différentes machines et dans des processus distincts, assurant les communications entre les objets répartis(clients / serveurs).

C'est un environnement dans lequel ces objets fonctionnent. Cet environnement fournit tous les services de communication de base, ainsi que les services d'enregistrement, sous forme d'une couche logicielle située entre le système d'exploitation et les applications qu'il fournit :

- Des interfaces de programmation.
- Des formats d'échange de donnés.
- Un protocole d'interopérabilité.

Le middleware est [2] La couche logicielle qui se situe entre le système d'exploitation et les applications de chaque côté d'un système de calcul distribué dans un réseau.

Les composants logiciels du middleware assurent la communication entre les applications quels que soient les ordinateurs impliqués et quelles que soient les caractéristiques matérielles et logicielles des réseaux informatiques, des protocoles de réseau, des systèmes d'exploitation [3].

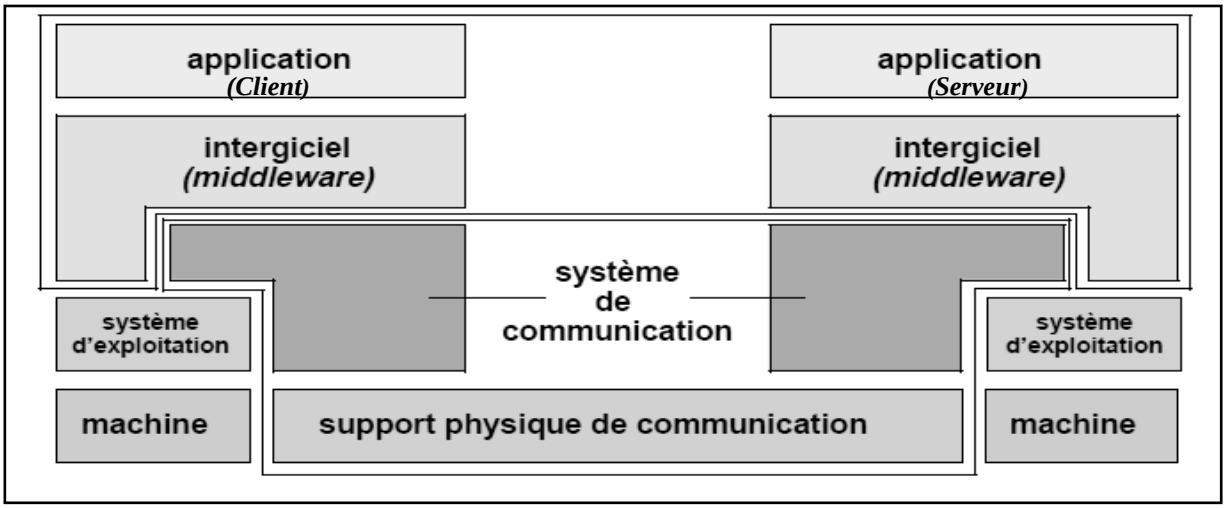


Figure 1.1 : Architecture - Client - Middleware - Serveur

La construction d'applications distribuées se fait actuellement à travers une approche largement admise : l'utilisation d'un middleware. Actuellement trois grands types de middleware qui sont disponibles : CORBA, RMI et DCOM, ils reposent tous les trois sur une architecture identique. Une couche (le middleware) assure la communication entre les entités distantes de l'application, c'est-à-dire qu'il assure le transport des appels de procédures, de méthodes ou de fonctions, le codage de l'information, la gestion du nommage, la localisation, etc..., les trois middlewares seront l'objectif de notre étude de comparaison et de synthèse dans les prochains chapitres.

I-3-Principe de fonctionnement d'un middleware

1- localiser les objets serveur qui sont basés sur :

- La procédure d'établissement de connexion.
- L'exécution des requêtes.

- La récupération des résultats.
- La procédure de fermeture de connexion.
- L'initiation des processus sur différents sites

2- L'échange des messages entre clients et serveurs en mode synchrone (requête/réponse) ou en mode asynchrone (via des fils d'attente).

3- Les mécanismes de formatage et conversion des données (requis du fait de l'hétérogénéité des plates formes (système d'exploitation)).

4- Les fonctions de sécurité (contrôle d'accès).

5- L'administration (configuration et exploitation) de l'ensemble des composants (système et application) des différentes plates formes

I-4- Avantages d'un middleware [4]

- Former une structure de communication entre des clients et des serveurs indépendants de système d'exploitation et machines, et de la nature de réseau de transmission utilisé.
- Permettre le développement rapide des applications distribuées, en permettant aux développeurs de ne pas avoir manipulé les communications de bas niveau mais directement les objets distants.
- Cacher les détails d'implantation des services et des clients, et masque la complexité des échanges inter-applications
- Fournir un environnement d'exécution aux applications, cet environnement devra bien sûr assurer l'accès aux données de l'entreprise, fournir des fonctions de distribution et permettre la notion de transaction.
- Optimiser les ressources machines, dans le but de répondre au besoin de temps de réponse garantis

1-5- Le modèle client- serveur avec un middleware

Ce modèle est basé sur l'architecture à trois niveaux (appelée architecture 3 tiers), il existe un niveau intermédiaire qui est la caractéristique la plus importante dans la réalisation de ce modèle, c'est à dire que l'on a généralement une architecture partagée entre :

- Le client

Application demandant l'exécution d'un service à une autre application serveur par envoi de message (requête) contenant le descriptif de service à exécuter et attendant la réponse à cette demande par un message en retour (réponse), le client est toujours l'initiateur de dialogue.

- Le serveur d'application (appelé aussi middleware)

Ensemble de services logiciels construits au-dessus d'un protocole de transport afin d'assurer les connexions et de permettre l'échange des requêtes et des réponses associées entre client et serveur de manière transparente.

- le serveur

Application qui implémente des services, ces services sont à la disposition des clients, le serveur accomplissant un service sur demande d'un client et transmettant la réponse à ce dernier.

Les trois parties du dialogue sont reliées par un contrat appelé interface IDL, décrivant l'échanges des messages (requête – réponse) entre le client et le serveur

- **Requête**

Message transmis par un client à un serveur décrivant l'opération à exécuter pour le compte du client.

- **Réponse**

Message transmis par un serveur à un client suite à l'exécution d'une opération contenant les paramètres de retour de l'opération.

- ***Construction du client et du serveur***

Lorsque les trois composants mentionnés précédemment sont disponibles, la construction d'un client et d'un serveur est comme suit :

- Etape 1 :

  Générer les stubs client et serveur. Ceci est réalisé grâce au compilateur IDL.

- étape 2 :

  Construire l'exécutable client. Il est nécessaire de compiler le programme principal et les stubs client, puis de lier les deux codes objets obtenus

- étape 3 :

  Construire l'exécutable serveur. Il est nécessaire de compiler le stub serveur et les services proposés puis de lier les codes objets obtenus.

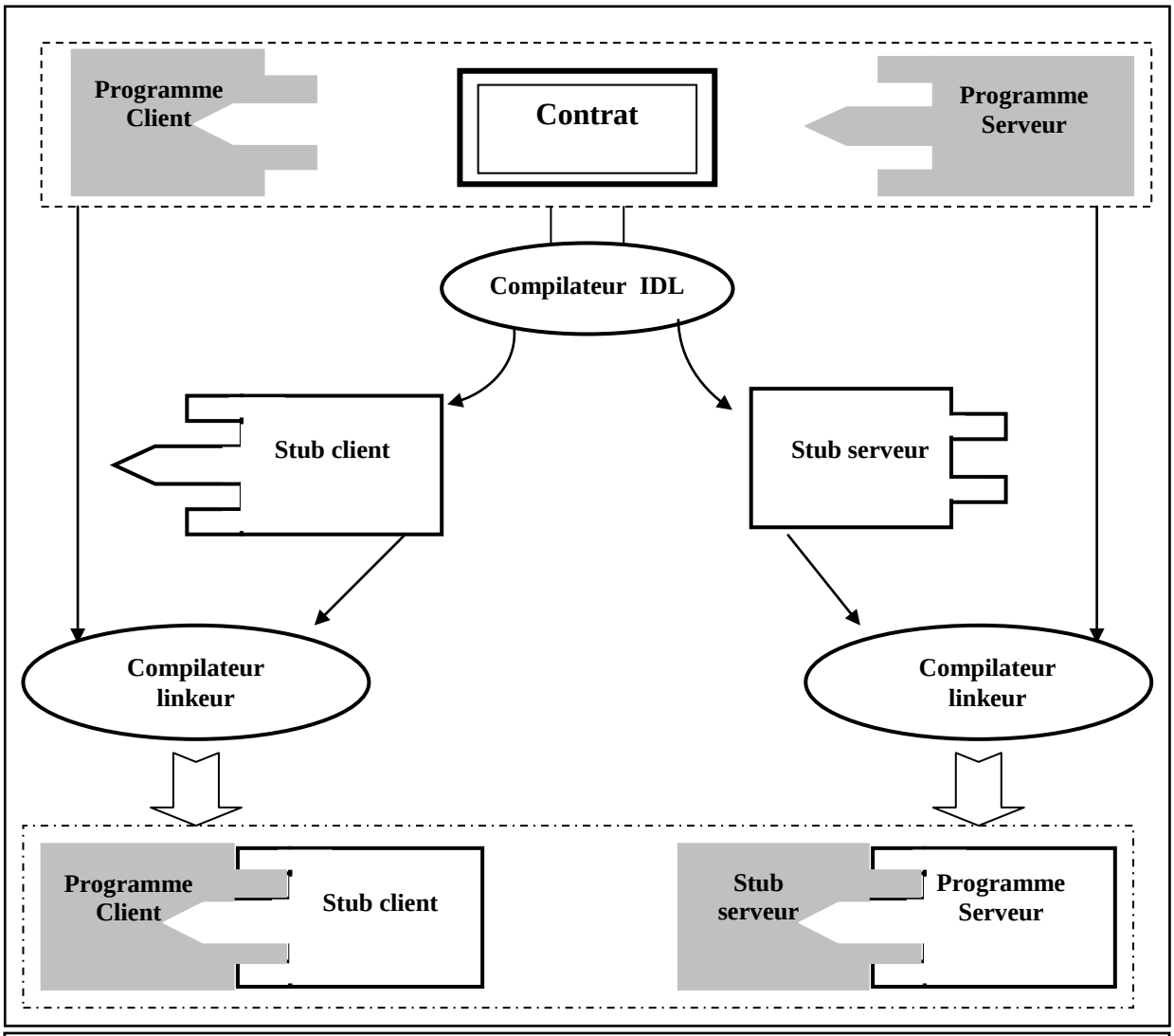


Figure 1.2 : construction d'un client et d'un serveur.

- **La connexion au serveur**

Pour que le client puisse être connecté avec le serveur, il faut avoir l'intermédiaire qui implémente un service de nom, tel service des noms sait faire correspondre à un nom logique d'entité son adresse physique, ainsi le service des noms peut être utilisé pour retrouver l'adresse physique d'un serveur.

L'utilisation d'un service des noms fonctionne comme suit [5] [6] :

1. Création du serveur
2. Le serveur s'enregistre auprès du service des noms afin que celui-ci sache son adresse physique et les interfaces qu'il offre, Le service des noms confirme au serveur qu'il est maintenant connu, le serveur se met alors en attente de requêtes.
3. Le client demande au service des noms l'adresse physique d'un serveur, en lui fournissant le nom du serveur et le numéro de l'interface, Le service des noms retourne au client l'adresse physique d'un serveur satisfaisant aux critères de recherche.
4. Le client peut alors établir une connexion avec serveur (liaison logique).
5. Le client exécute une requête pour demander un service.

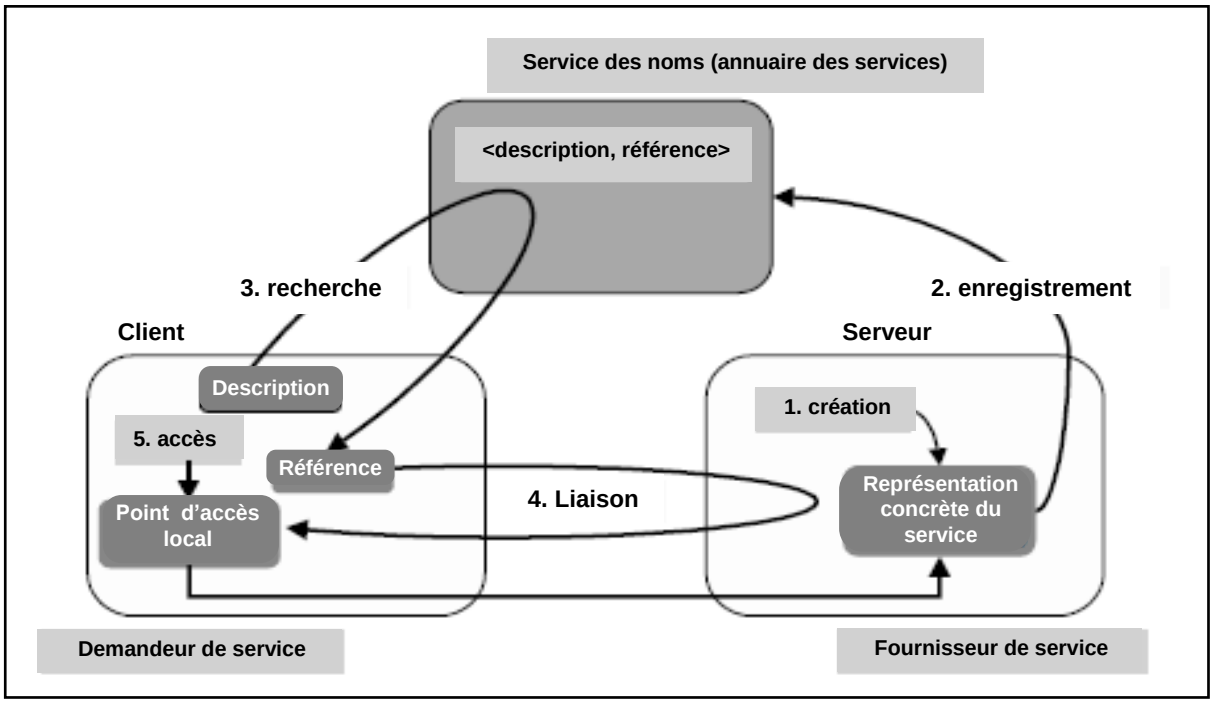


Figure 1.3 : Localisation du serveur à l'aide du service des noms

1-6- La programmation orienté objet

La technologie objet possède des propriétés très intéressantes pour la conception d'un middleware, en particulier :

- Le concept d'**encapsulation :** Ce concept permet de séparer l'aspect externe d'un objet qui forme son interface (nom des attributs et les opérations) de l'aspect interne (façon dont sont réalisés les attributs et les opérations).

L'aspect externe est défini lors de la phase de modélisation et l'aspect interne est précisé lors de la phase de programmation, ce concept permet de parler d'un objet sans se préoccuper de la façon dont il sera réalisé.

- Un système objet apparaît comme une liste d**'interfaces** derrière lesquelles se trouve le code associe aux attributs et aux opérations relatifs aux objets. Il est ainsi possible de modifier le code indépendamment des interfaces. Ceci constitue un point fondamental de la technologie objet.
- le concept d**'héritage**. Un objet peut hériter des caractéristiques (attributs et opérations) d'un autre objet. Ce mécanisme rend plus concis le modèle et facilite la réutilisation des logiciels.

Une application apparaît comme un ensemble d'objet coopérants, si on suppose que les objets sont distribués à travers le réseau, la communication inter objets correspondant à la demande d'exécuter d'une opération sur un objet. Cette demande est réalisée à l'aide d'un bus de communication spécifique appelé middleware objet

I-7- la conception d'un middleware

Le premier but d'un middleware est de résoudre le problème d'intégration des applications, l'ajout d'une application dans un environnement informatique comporte n applications peut conduire à construire n liens de communication, et 2n interfaces d'application.

Une façon de résoudre ce problème est d'introduire le concept de bus unique de communication (middleware), au quel les applications se connectant par l'intermédiaire d'une interface clairement définie.

Une telle architecture de middleware de communication qui permet d'échanger de données entre deux ou plusieurs applications offre certains nombres de services, le middleware doit permettre de [7] :

- masquer la répartition des équipements et les modes de connexion,
- masquer l'hétérogénéité des divers matériels, et protocoles,
- présenter des **interfaces homogènes** et de haut niveau aux développeurs et aux intégrateurs pour faciliter le **portage** et l'**interopérabilité**,
- présenter des **services communs** à travers de fonctions d'usage général,
- assurer le **filtrage** d'informations,
- assurer la **conservation** de l'information jusqu'à sa réception confirmée par l'application destinatrice.

En théorie, un grand nombre de services [8] peuvent être définis, mais seuls quelques uns sont obligatoires

I-7-1- Création et gestion de la durée de vie des objets

La création des objets répartis est une étape indispensable, qui est prise en charge par le service de gestion de la durée de vie des objets, et ce service qui prend en charge leur destruction, lorsqu'ils ne sont plus utiles.

Afin de permettre la création de nouveaux objets répartis, il faut disposer d'un nom unique pour chaque objet, lorsqu'un client demande la création d'un objet réparti, il doit spécifier l'objet qu'il doit être instancié à l'aide de ce nom unique, et le middleware utilise ce nom pour localiser le serveur gérant cet objet pour le créer.

Pratiquement, les serveurs créent donc les objets, et les enregistrent au niveau de middleware avec leur nom.

La gestion de la durée de vie est aussi très importante que la création des objets, en général les objets repartis restent actifs dans le système tant que quelqu'un a besoin d'eux. Ceci signifie que tous les clients d'un objet réparti sont suivis, leur connexion et leur déconnexion sont traquées pour déterminer quand un objet peut être détruit.

I-7-2- Service de connexion et de nommage

Le fait de retrouver un objet réparti suppose que l'on puisse l'identifier. Cette identification se fait naturellement par son nom, qu'il est défini de manière unique dans un espace de nommage.

Ainsi, lorsqu'un client désire accéder à un service particulier, il peut le référencer à partir de son nom, qu'il connaît, et il sait à quel objet correspond ce nom.

Bien entendu, les noms des objets doivent être valides partout dans le middleware c'est à dire qu'ils doivent désigner de manière unique un objet en toutes circonstances.

Donc un service de nommage, il faut qu'il permet de désigner un objet de manière uniforme dans tout le système [9], de telle sorte que les différents intervenants puissent parler de la même chose, et permettent de gérer les noms logiques des services, ceux ci peuvent être connus statiquement des applications ou au contraire être découverts dynamiquement grâce à un annuaire

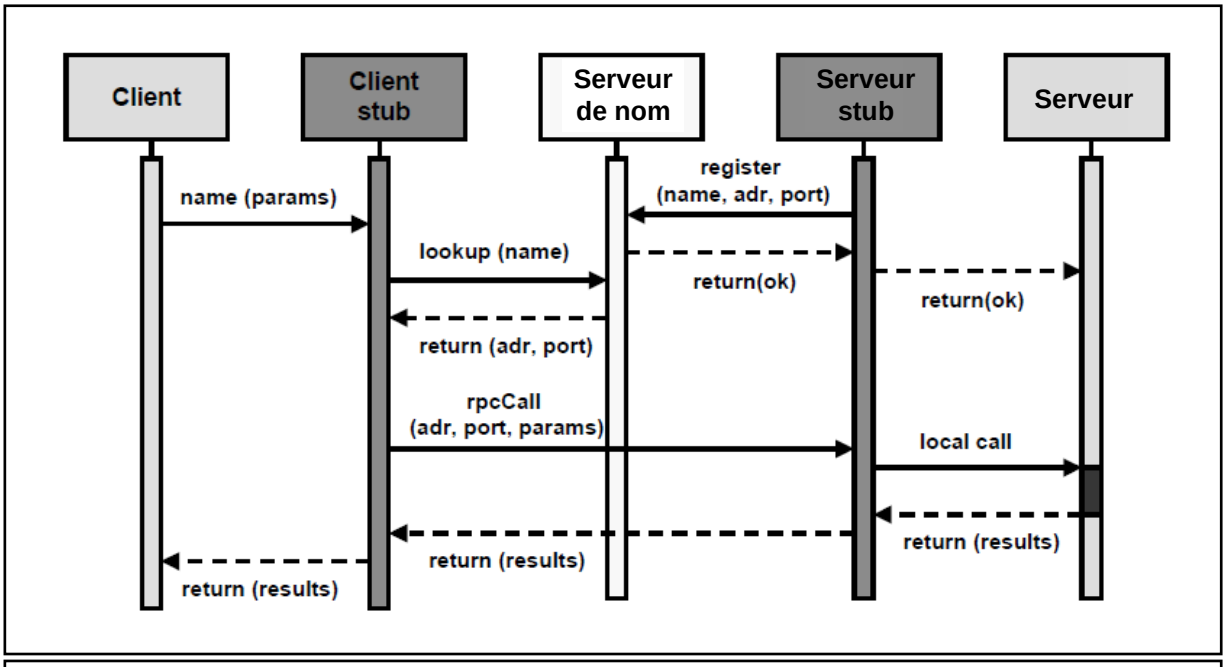


Figure 1.4 : Etablir une connexion au serveur

Le service nommage des objets en pratique doit permettre :

- d'attribuer un nom unique à un objet.
- de communiquer ce nom à un autre objet en s'assurant que ce nom sera toujours valide pour l'objet destination.
- de comparer deux noms pour savoir s'il s'agit du même objet.

- de localiser un objet dans le système à composants à partir de son nom.
- d'obtenir une référence sur l'objet à partir de ce nom (opération dite de « binding »/lookup).

Les services de connexion permettent, en fonction d'un nom, de trouver qu'elle est la machine sur laquelle va effectivement s'exécuter le service. De cette façon, la distribution peut être rendue complètement transparente pour le programmeur de l'application.

I-7-3- Notion de marshalling

Le marshalling est l'opération qui consiste à traduire tous les paramètres qu'une fonction reçoit en une représentation standard, celle-ci constitue donc une convention d'appel générique et absolument portable, qui permet aux différentes implémentations d'un middleware de parler la même langue.

Inversement, les paramètres de retour de la fonction appelée, ainsi que ses codes d'erreurs et ses exceptions éventuelles, doivent être renvoyés à l'appelant. Ils sont donc également transmis via les mécanismes standard de marshalling de middleware.

Ainsi, le marshalling assure la communication entre les clients et les objets répartis, et assure que cette communication se fera correctement, quelles que soient leurs localisations, cependant, le client et le serveur n'ont pas à se soucier de ces détails techniques, puisque les clients se contentent d'appeler une méthode d'une interface, et les objets serveurs ne voient qu'un appel d'une de leur méthode. L'opération de marshalling est donc totalement prise en charge par le middleware. Cette technique cache en réalité des problèmes très complexes :

- Premièrement, les objets répartis peuvent très bien ne pas être écrits dans le même langage que leurs clients
- Deuxièmement, ils peuvent ne pas être dans le même espace d'adressage ceci signifier que les appels de fonctions sont en fait des appels de procédure à distance.
- En fin, les objets répartis peuvent tourner sur des architectures matérielles et des systèmes d'exploitation complètement différents de ceux des clients.

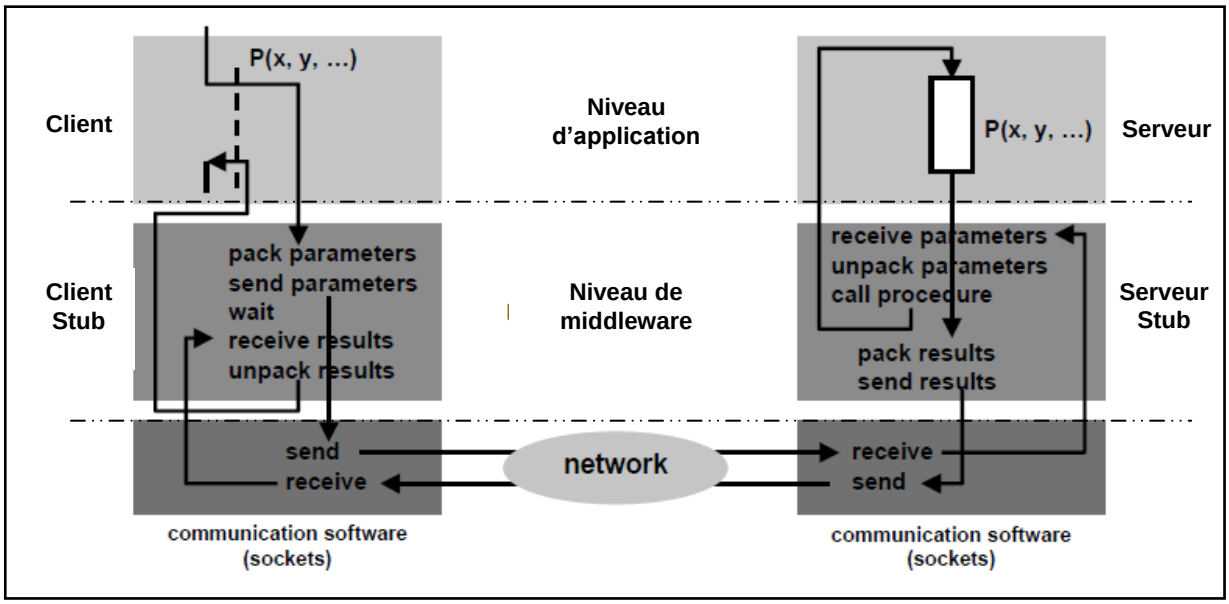


Figure 1.5 : Notion de marshalling (pack / unpack) dans un middleware

Pour assurer cette transparence, le middleware en général recours à des objets intermédiaires qui prennent en charge le marshalling, sont les souches (stub) de côté client et les squelettes coté serveur

I-7-4- l'utilisation d'une interface

Le système entier, et chacun de ses composants, remplit une fonction qui peut être décrite comme la fourniture d'un service. Selon une définition tirée de [10] « un service est un comportement défini par contrat, qui peut être réalise et fourni par tout composant pour être utilisé par tout composant, sur la base unique du contrat ».

La plus part des middleware utilisent des interfaces, pour permettre la communication entre les applications clients et les objets serveurs, il est obligatoirement de définir les interfaces des objets aux quels vous désirer accéder.

L'interface définit un contrat entre l'objet serveur et ses clients, l'interface représente la partie externe d'un objet, la liste de tous les éléments proposés par cet objet, par exemple données et méthodes, qui sont déclarer de façon publique, donc accessibles librement par d'autres objets.

L'interface représente le moyen par lequel un objet informe ses futurs clients des ses possibilités, il indique précisément les méthodes disponibles ainsi que les modalités d'appels, c'est à dire les paramètres à lui transmettre, et les paramètres à recevoir comme résultat de retour d'un appel.

I-7-5- Invocation dynamique

L'utilisation d'une interface connue à l'avance pour obtenir des informations sur les objets serveur est une idée élégante, mais n'est pas le cas si nous n'avons pas cette interface.

Il faut donc disposer d'un mécanisme dynamique, qui permettre de construire des requêtes sur les objets sans avoir connaître à l'avance leurs interfaces, ce mécanisme constitue ce que l'on appelle « invocation dynamique » des méthodes des interfaces, ce qui permet de construire des appels de méthodes sur des objets distribués à l'exécution, à partir du nom de la méthode et de l'ensemble de ses paramètres.

En pratique, le client qui désire réaliser une invocation dynamique doit commencer par récupérer les informations sur les objets, à partir de ces informations, il peut construire l'appel de n'importe quelle méthode,

Inversement, les objets répartis reçoivent les appels desquelles provenant de ces clients sous forme de paramètres indiquant le nom de la méthode à appeler et la liste des paramètres à utiliser pour effectuer cet appel, les objets répartis doivent donc interpréter la requête qu'ils reçoivent.

Bien entendu, les middlewares fournissent généralement des mécanismes standard permettant d'automatiser le travail d'interprétation des requêtes.

I-7-6- Gestion de la persistance

La persistance est l'opération qui consiste à enregistrer les objets, par exemple dans des fichiers de sauvegarde, et de lire ce fichier pour initialiser ses futures instances, en revanche,

que des objets doivent être recréés systématiquement dans le même état que celui dans lequel ils se trouvaient lorsqu'ils ont été déchargés de la mémoire.

Les middlewares fournissent souvent un service de persistance de base, que tous les composants peuvent utiliser. Ce service uniformise la gestion des objets persistants, et permet de standardiser des informations de base que tout le monde est capable d'interpréter même sans disposer du composant qui a généré ces données, pour pouvoir à un composant d'enregistrer les données de ses instances dans un flux de données standard.

L'objet n'a besoin de stocker ses données, et il ne s'intéresse absolument pas à leur devenir. Les clients des objets peuvent ensuite placer les données de ce flux dans n'importe quel support physique de données approprié.

I-7-7- Gestion de la sécurité

La plupart des applications se contentent de laisser le middleware en charge de sécurité pour éviter que par exemple :

- Un client non autorisé peut en effet chercher à faire un objet privilégié d'effectue une opération qu'il n'a pas le droit de faire.
- Inversement, un serveur peut parfaitement transférer un objet réparti sur une machine cliente, sans forcément que celle-ci ait donné son accord.

Les middlewares fournissent souvent des mécanismes de sécurité pour contrôler l'accès à des applications qui ne soucient pas de ce type de problèmes, ainsi toute application répartie est soumise implicitement à un jeu de règles de sécurité globale définies au niveau de middleware lui-même, afin d'éviter qu'une machine du système puisse être remplacée par une machine pirate.

Les objets répartis étant publics, la sécurité doit être prise en compte à différents niveaux. Les principaux points à assurer pour la sécurité sont [11] :

– **la confidentialité :** elle consiste à interdire la lecture ou la copie d'informations à des personnes qui n'ont pas été explicitement autorisées par le propriétaire de ces informations.
– **l'intégrité des données :** des données ne doivent pas pouvoir être supprimées ou altérées sans l'accord de leur propriétaire.
– **la disponibilité :** les services ne doivent pas être dégradés ou rendus inaccessibles sans autorisation.
– **la consistance :** le système doit se comporter de la façon attendue par les utilisateurs.
– **le contrôle d'accès :** l'autorisation de l'utilisation d'une fonctionnalité par un utilisateur défini doit être explicitement précisée.
– **la surveillance :** des erreurs ou des actes malveillants peuvent être commis par des utilisateurs autorisés. Dans ce cas il est important de mettre en place des mécanismes pour déterminer ce qui a été fait, par qui, et les données ou services affectés.

Il faut examiner l'authentification (comment on s'assure qu'un client est bien celui qu'il prétend être), les contrôles d'accès (comment on détermine les ressources auxquelles a droit un client) et la confidentialité (comment on protège les ressources de l'espionnage, fait par exemple à partir d'outils d'analyse de trafic réseau).

I-7-8- Gestion des licences

La répartition des objets rend le contrôle des licences de la localité d'exécution des objets un problème évident. Car un composant n'a plus besoin d'être installé sur une machine pour être utilisé, il peut être exécuté à distance, c'est pour ces raisons que les middlewares fournissent souvent un service de gestion des licences.

En général, le contrôle des licences est simplement l'autorisation ou l'interdiction d'utiliser un objet. Ce contrôle est généralement pratiqué par les middleware au niveau de l'obtention des références sur les objets. Ou bien au niveau de la création, qui peuvent exiger des informations dont seuls les utilisateurs licenciés disposent.

Enfin, chaque objet est bien entendu libre de contrôler les conditions de son exécution, par tous les moyens qu'il désire, et refuser l'exécution des requêtes provenant des clients non autorisés.

I-7-9- Fiabilité des systèmes à Objet

Premier cas, un objet réparti peut être utilisé par plusieurs clients, donc il ne faut pas qu'une faute d'un client entraîne la mort des objets qu'il utilise. Par exemple, le cas ou le client ne déconnecte pas les objets serveurs qu'il utilise, ceci peut engendre une occupation de mémoire inutile dans le meilleur des cas, ou une erreur du serveur dans le pire des cas. La solution est que le middleware lui-même ferme les connexions et libère les ressources du client lorsque celui-ci se termine

Dans le cas inverse, si un serveur provoque une erreur ou est arrêté de manière forcée, les clients qui l'utilisent encore disposeront d'une référence non valide sur ce serveur, le middleware doit signaler les erreurs et les pertes de connexion avec le serveur lorsque le client cherche à utiliser cette référence par un mécanisme dédie.

Deuxième cas, Comme chaque objet peut être à la fois client et serveur, il est tout à fait envisageable qu'un objet soit client d'un autre objet, qui lui-même est client de premier, il peut avoir d'autres objets s'intercalant entre ces deux objets, l'ensemble formant un cycle de dépendance, c'est dans ce genre de cycle que les interblocages peuvent apparaître quoi qu'il en soit, les middlewares doivent prendre en compte les interblocages, en général ils ne cherchent pas à les détecté mais plutôt à les évité.

I-7-10- Définition d'un protocole de communication

Le but d'un réseau est de faire communiquer plusieurs ordinateurs entre eux, si les hommes communiquent entre eux grâce aux différentes langues, Dans une conception client serveur, il faut avoir un protocole qui organise la communication entre le client et le serveur.

Le protocole est une description formelle de règles et de convention à suivre dans un échange d'information, que ce soit pour acheminer les données jusqu 'au destinataire ou pour que le destinataire comprenne comment il doit utiliser les données qu'il a reçues, il détermine un accord sur la façon dont les communications doivent effectuer.

La notion d'un protocole est prise en compte par la plupart des middlewares qui définissent leurs propres protocoles pour assurer la réception correcte d'échanger d'informations entre le client et le serveur.

I-7-11- Utilisation d'une file d'attente

Les interactions dans les systèmes d'information distribués sont implémentées par les modèles de communication : appels synchrones et appels asynchrones.

Dans le cas des appels synchrones entre deux processus, celui que génère la demande doit attendre l'arrivée de la réponse pour pouvoir continuer son exécution.

Dans le cas des appels asynchrones, un processus envoie l'appel et continue son exécution jusqu'au moment où il demande le résultat de l'appel.

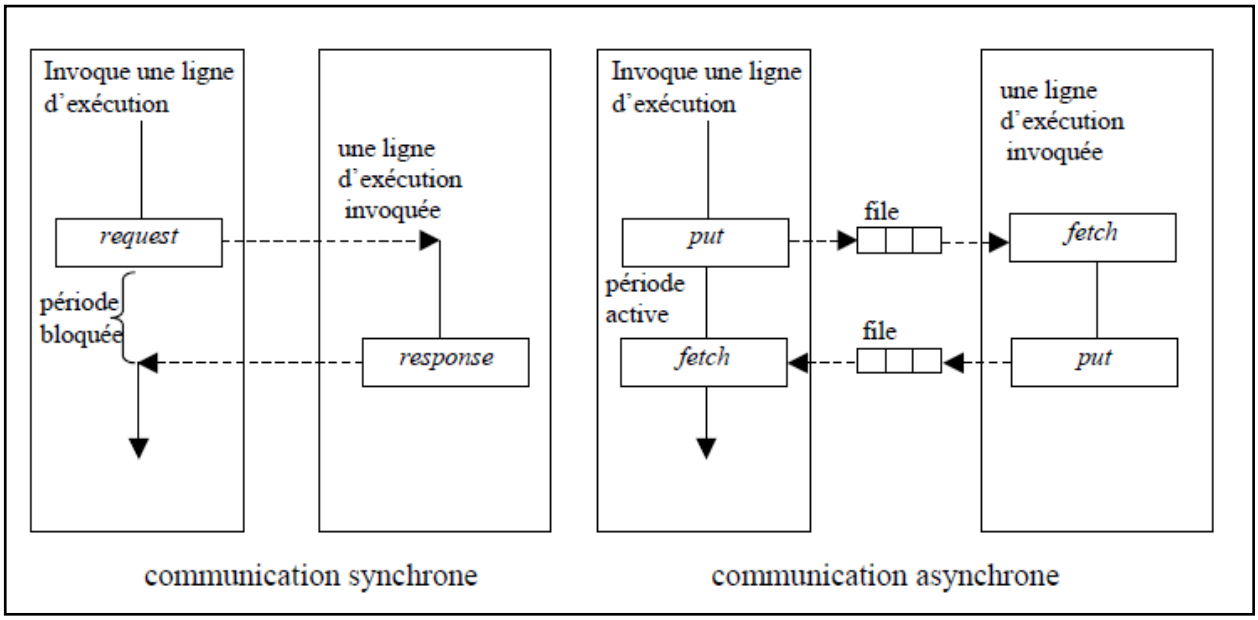


Figure 1.6 : communication synchrone et asynchrone

Le middleware par file d'attente [12] fut l'une des premières technologies à réaliser le concept de bus de communication commun à plusieurs applications afin de leur permettre d'échanger des messages.

L'échange de message peut être synchrone ou bien asynchrone, le middleware permet aux applications de communiquer de façon indépendante [13], le mécanisme asynchrone d'échange de messages permet à l'application émettrice (le client) de déposer son message dans une file d'attente et de continuer son traitement.

L'application réceptrice (le serveur), lorsqu'elle est disponible, lit le message et le traite, ainsi client et serveur fonctionnent à leur propre rythme.

Dans le mode de communication synchrone, lorsque le client a placé son message dans la file d'attente du serveur, il doit se mettre en attente de la réponse, aucun traitement en parallèle du client et du serveur n'existe.

Dans ces modes d'échange de message, à chaque application est associée deux files d'attente, et Le dialogue entre deux applications fonctionne comme suit :

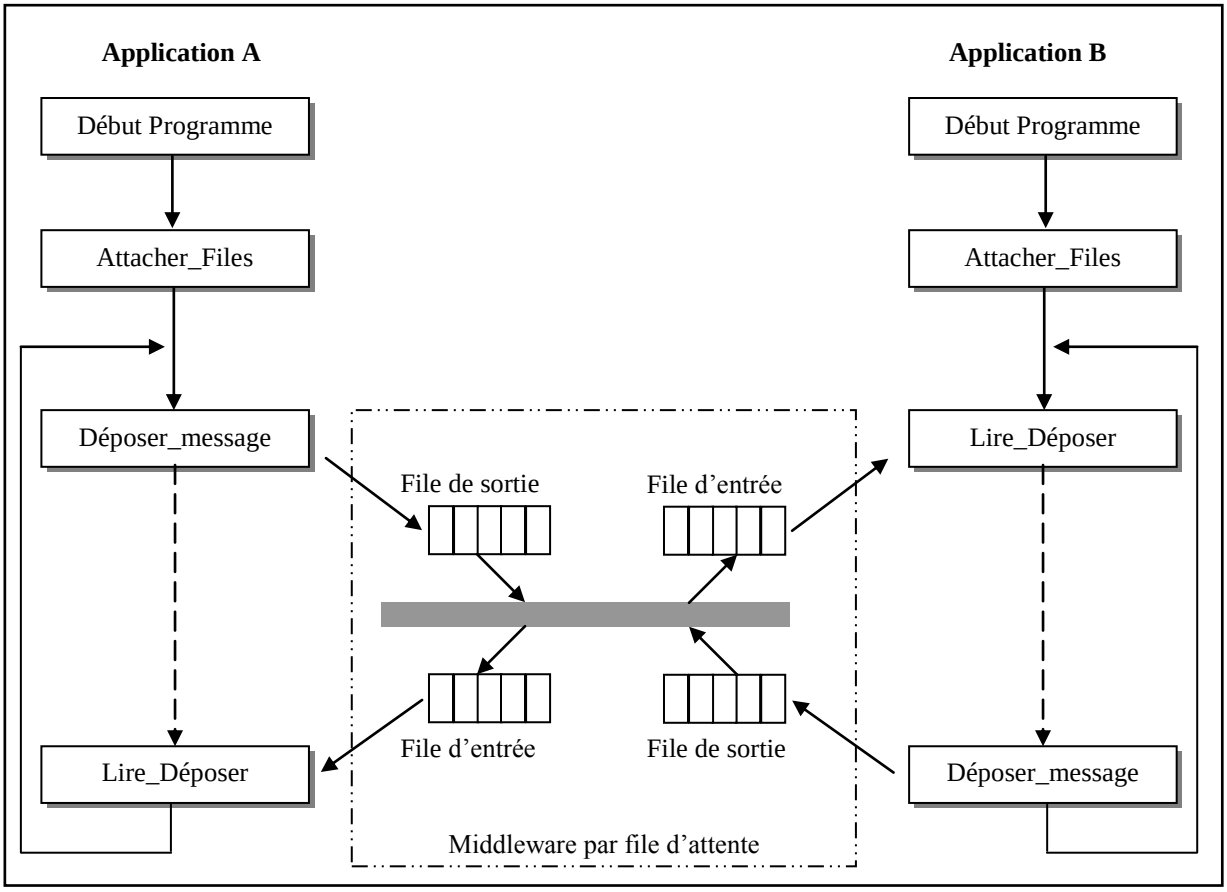


Figure 1.7 : Protocole de communication par files d'attente

1- l'application A initiatrice du dialogue, s'attache aux deux files d'attente de communication

2- l'application émettrice dépose le message qu'elle désire envoyer dans sa file de sortie, à partir de cet instant le message appartient au middleware et l'application peut, soit continuer un traitement interne (communication asynchrone), soit se mettre en attente du message réponse (communication synchrone)

3- le programme constituant le bus d'application lit le message et le transfère à travers le réseau à la file d'attente de destination

4- Apres s'être connectée à ses files d'attente, l'application destinataire est en attente de messages. Lorsqu'un message est déposé dans sa file d'entrée, elle le lit.

5- Apres traitement du message reçu, l'application réceptrice peut retourner un message réponse vers l'application émettrice, dans ce cas, elle dépose un message dans sa file d'attente de sortie

6- Le middleware récupère le message déposé et le transmet à l'application qui est à l'origine de dialogue.

7- L'application qui à initialisé le dialogue est soit en attente de la réponse et lit le message retour dès son arrivée, soit en cours de traitement et viendra lire le message plus tard.

I-8- Conclusion

Le but de ce chapitre est de montrer l'importance du middleware comme technologie permettant de répondre aux besoins actuels de l'industrie. Ces besoins s'expriment en termes de distribution et de coopération entre applications ou entre composants d'applications

Le middlewares (Intergiciel) est un logiciel de connexion formé d'un groupe de services qui permettent l'exécution de plusieurs applications sur une ou plusieurs machines connectées en réseau. Cette technologie fournit une couche d'abstraction qui permet l'interopérabilité entre différents langages de programmation, systèmes d'exploitation et architectures d'ordinateurs **[14].**

Tous les middlewares sont Construit de manière modulaire, l'approche objet est un début de solution puisqu'elle apporte l'encapsulation et la modularité, il ne reste donc plus qu'un protocole réseau pour faire communiquer des objets entre eux afin de répondre aux besoins de développeurs.

Différentes technologies réalisant le middleware suivant ce concept, leurs utilisation est influente profondément sur l'architectures globale d'un système d'information,

Il existe diverses implémentations d'architectures middleware orientées objet. Parmi le plus répondues, on peut citer CORBA de l'Object Management Group (OMG), Java RMI de Sun Microsystem, ou encore l'architecture DCOM de Microsoft.

Dans le principe, ces trois architectures fonctionnent d'une manière similaire, ainsi on y retrouve la plupart des notions évoquées précédemment. Les différences apparaissent principalement, dans la manière de spécifier les fonctionnalités du service, de générer les couches intermédiaires ou de retrouver les objets distants et le choix entre ces technologies est difficile. Ce choix est l'objet de ce mémoire.

CHAPITRE II

LE MIDDLEWARE
- CORBA -

Le but de ce chapitre est de permettre d'apprendre les principes de base concernant cette architecture, il présente la norme CORBA et tous les concepts associent.

Vous trouverez des notions générales qu'il vous faudra connaître pour commencer votre apprentissage de CORBA. Nous rappellerons les différents modes de construction d'applications, notamment l'architecture distribuée, puis nous nous pencherons sur la norme CORBA. Vous découvrirez, ainsi les raisons qui font de CORBA une solution d'avenir.

Mot clé :

CORBA, OMG ,ORB, BOA , IIOP ,STUB , SQUELETTE

Chapitre II

LE MIDDLEWARE - CORBA -

Common Object Request Broker Architecture

II-1- Introduction

Le concept de réseau est devenu une base que tout système d'informatique moderne et performant ne peut l'ignorer. Les besoins croissants des utilisateurs des systèmes d'informations distribués ont engendré le développement d'applications de plus en plus complexe, ainsi la conception, le développement et la maintenance des applications posent des problèmes difficiles à résoudre.

Ces problèmes poussent un ensemble de concepteur de décider de réunir dans un groupe appelé OMG « Object Management Group », est un groupe de travail auquel appartient la plus part des acteurs informatiques majeurs, chargé de définir les bases d'une architecture distribuée universelle permettant de développer et de distribuer aisément des applications distribuées, depuis sa création ce groupe travaille à la définition d'un ensemble de spécifications et de recommandation d'une architecture distribuée idéale, sous le sigle CORBA.

II-2- Définition

CORBA (Common Object Request Broker Architecture) est une standard ouvert développé et maintenu par L'OMG,c' est une association qui compte aujourd'hui plus de 850 membres, CORBA est apparu au début des années 90, et elle est aujourd'hui la plus répondue des infrastructures d'informatiques distribuées.

L'OMG propose cette solution globale comme middleware à l'aide des technologies orientées objets pour construire des applications réparties, résolvant les problèmes de communication, l'intégration, l'hétérogénéité et l'interopérabilité, le cœur de cette proposition est le bus à objet ORB, ce bus fournit les mécanismes de base pour la communication entre les objets distribués et hétérogènes.

CORBA utilise l'approche orientée objets pour la création des composants logiciels peuvent être réutilisés et partagés entre applications, chaque objet en capsule les détailles de ces fonctionnements interne et présente une interface bien définie, ce qui réduit la complexité de l'application.

II-3- Les caractéristiques de CORBA

1- CORBA permet une interopérabilité indépendante des langages de programmation.

Les interfaces [15] des objets CORBA sont écrites en notion indépendante de tout langage de programmation appelé IDL « Interface Definition language), le résultat est que les clients et les serveurs peuvent être écrits dans n'importe quel langage de programmation, ce qui implique que CORBA est une architecture multi-langages.

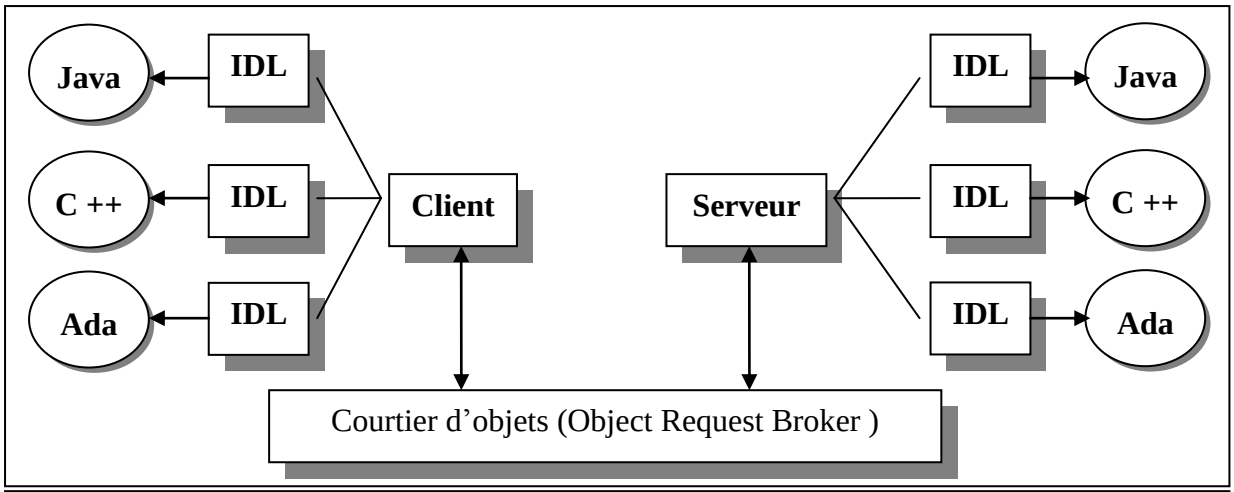


Figure 2.1 : Interopérabilité indépendante des langages de programmation

2- CORBA permet une interopérabilité indépendante des plates formes.

Le protocole utilisé par CORBA pour faire communiquer les objets distribués est spécifié par l'OMG, il s'appelle IIOP (ou plus généralement GIOP).

Des produits CORBA différents peuvent l'utiliser sur des plates formes différentes, en outre la plupart des ORBs standard ont été portés sur plusieurs plates formes logicielles et matérielles.

3- CORBA offre l'indépendance vis à vis des fournisseurs.

Les API standardistes (IDL) et le protocole de communication standardise (IIOP) offrent aux développeurs et à leurs clients une totalité indépendance vis à vis des produits et des fournisseurs CORBA.

La communication par différents éditeurs d'ORB CORBA compatibles signifier que les développeurs ne dépendant pas d'un seul fournisseur unique.

Editeur	*ORB*	*adresse Web*
- Inprise	*- VisiBroker*	*-www.inprise.com/visibroker*
- IONA	*- Orbix et OrbixWeb*	*-www.iona.com*
- IBM	*- SOMObject*	*-www.software.ibm.com/ad/somobjects*
- Object Oriented Concept	*- ORBacus*	*-www.ooc.com*
- ExpertSoft	*- CORBAPlus*	*-www.expertsoft.com/products/corbac*
- ORL	*-OmniORB2*	*-www.orl.co.uk/omniORB*

Tableau 2.1 : Interopérabilité indépendante des langages de programmation

4- CORBA offre des services suivants les besoins et les contraint de projet à réaliser.

En fonction des besoins et des contraintes des projets à réaliser certains d'ORB CORBA seront vite écartés car par exemple il n'implémente pas tel service particulier.

Selon le choix de l'ORB de fournisseur et l'ORB utilisé, CORBA présente une gamme de services,qui sont des outils destinées aux développeurs, pour diminuer les temps de développement et fiabiliser leurs applications distribuées.

L'ORB permet aussi de [16] :

- Trouver l'implémentation de l'objet serveur
- Préparer cette implémentation à recevoir la requête.
- Communiquer les données constituant la requêtes ainsi que le résultat de cette exécution

II-4- Architecture générale de CORBA

L'OMG définit une vision globale de la construction d'applications réparties : l'Object Management Architecture Guide [17]. Cette architecture globale, appelée aussi l'OMA, vise à classifier les différents objets qui interviennent dans une application en fonction de leurs rôles

Cette section présente la structure et le fonctionnement des composants qui constituent cette architecture et permettent de gérer et de créer les objets répartis.
La figure suivante [18] illustre les interfaces (composants) du bus CORBA.

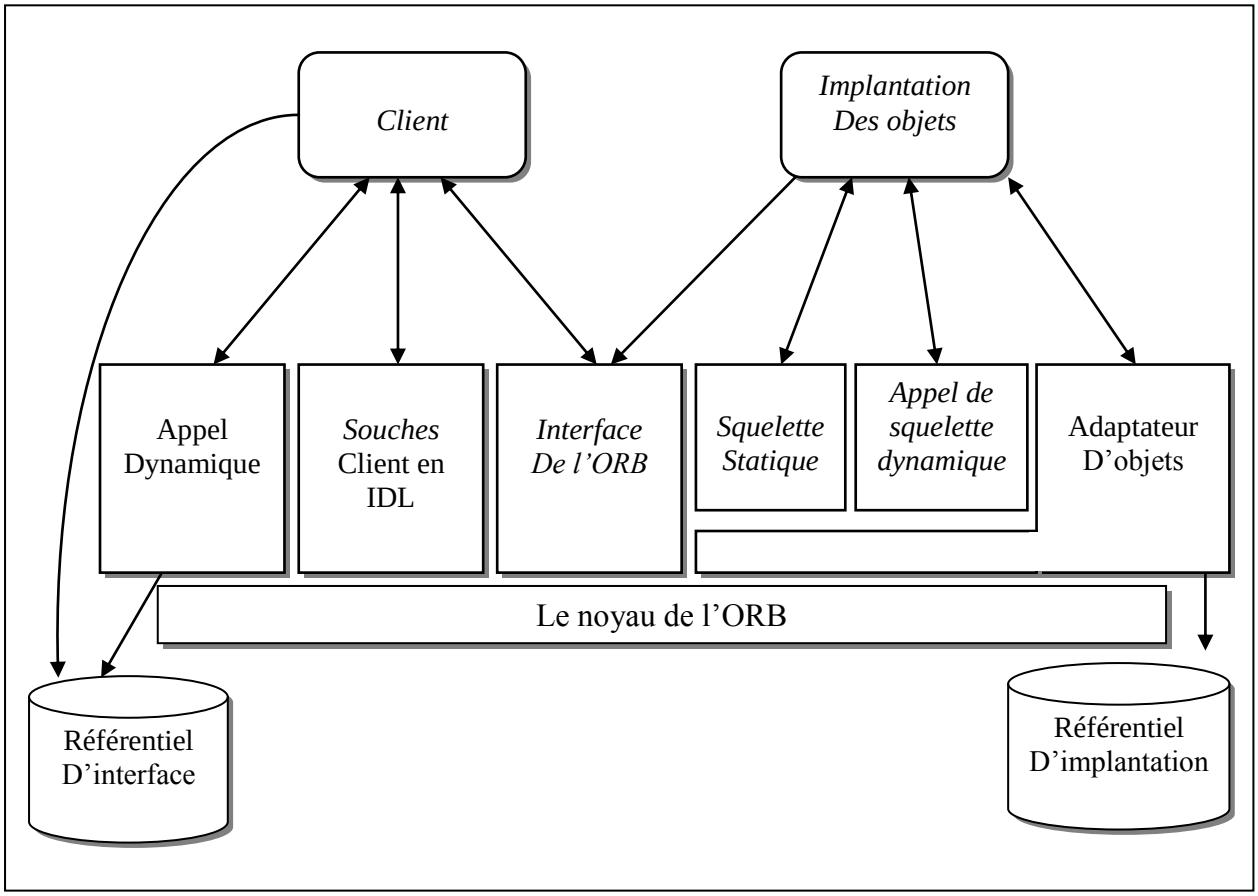


Figure 2.2 : Architecture générale de CORBA

II-4-1- ORB (Object Request Broker)

L'ORB est le noyau de transport des requêtes aux objets, souvent appelé bus logiciel, est au cœur de l'architecture CORBA.

- Il gère la localisation des objets dans l'environnement
- Il implante les protocoles de communication entre objets
- Il est accessible au travers d'un ensemble de primitives

Il intègre au minimum les protocoles GIOP et IIOP.

- General Inter-ORB Protocol (GIOP) :
  Spécification qui permet l'interopérabilité entre différents ORBs.
- Internet Inter-ORB Protocol (IIOP)
  Protocole d'interopérabilité standard pour Internet.

II-4-2- Invocation statique

L'invocation statique permet d'exécuter une méthode connue au moment de la compilation des parties serveur et client, elle est représentée par les deux composants suivantes :

- Stub (Static Invocation Interface**) coté client:**

- Partie de code générée automatiquement par un compilateur IDL vers un langage de programmation cible.
- interface précompilée statique aux services objets
- Code utilisé par le client lors des invocations statiques.
- Lien entre le client et l'ORB.
- Traduit les invocations du client en message transmissibles sur le réseau : opération "marshalling ».
- Traduit les messages qui proviennent de l'ORB : opération "unmarshalling".
- Du point de vue d'un client, le stub agit comme un appel local.
- le client possède une souche IDL pour chaque interface

- Squelette (Skeleton Static Interface**) côte serveur:**

- Partie de code générée automatiquement par un compilateur IDL vers un langage de programmation cible.
- Code utilisé par l'Adaptateur d'objet lors des invocations statiques.
- Lien entre l'ORB et l'objet d'implémentation.
- Reconstitue la requête du client de façon à invoquer la méthode requise : opération «unmarshalling ».
- Traduit les paramètres de retour en messages transmissibles sur le réseau : opération «marshalling ».

II-4-3-Invocation dynamique

Elle [19] permet aux client de pouvoir découvrir les objets pendant l'exécution et effectue une requête sur un service l'identité et la structure ne sont pas connues a la compilation du programme client, cette invocation est assurée par deux composants :

-Interface d'invocation dynamique (Dynamic Invocation Interface) côté client :

- Elle Permet la création dynamique de requêtes;

- L'utilisation du référentiel des interfaces pour récupérer les informations relatives aux interfaces IDL
- Les invocations doivent être contrôlées à l'exécution.
- **Avantage :**

  - les interfaces peuvent être découvertes dynamiquement.
  - code client générique indépendant d'une interface IDL.

- **Etapes de création d'une requête**

  - recherche et interprétation de l'interface dans le référentiel des interfaces.
  - construction d'un objet requête.
  - spécification de l'objet cible et de l'opération.
  - ajout des paramètres
  - ajout des paramètres
  - invocation de la requête.

-Interface de squelette dynamique (Dynamic Skeleton Interface**) côté serveur**

- Il permet de délivrer une requête à un objet implémentation qui est inconnu lors de la phase de compilation.
- Il interprète une requête et ses paramètres.
- Il est analogue au DII mais du côté serveur.
- Il est utilisé pour créer des ponts entre des ORBs de vendeurs différents.

II-4-4- Object adapter

L'adaptateur d'objet est un outil coté serveur qui permet l'accès à différents types d'objets, et permet aussi l'interaction entre l'objet serveur et l'ORB et autre il permet : [18]

- D'appeler des méthodes standards pour créer et détruire les objets.
- De déterminer l'état d'objet.
- De transmettre à l'objet les demandes émises par les différents clients
- D'activer un objet lorsqu'une requête lui en est destinée.

I-4-5- le référentiel des implémentations

C'est une base de données persistantes qui contient des informations permettant à l'ORB de localiser et activer l'objet, parmi les informations qui peuvent y figurer on trouve : [20]

- Le nom de l'exécutable du code de l'objet.
- le mode d'activation de l'objet.

II-4-6- le référentiel des interfaces

C'est une base de données qui permet de stocker en permanence des interfaces IDL, c'est une publication nécessaire pour les clients, pour la découverte de l'interface d'un objet.

II-4-7- L'interface de L'ORB

L'interface IDL définit le type des objets distribués en spécifiant leur interface [21]. L'IDL est donc le moyen par lequel un serveur rend publique les opérations distantes, et

comment elles doivent être invoquées par les éventuels clients. L'OMG a définit les règles de transformation (Mapping) d'une interface écrite en IDL dans les langages utilisés pour l'implémentation des objets distants.

II-5- Les services CORBA spécifiés par l'OMG

Ils ont été définis par l'OMG dans le but de simplifier et diminuer le temps des développements et de fiabiliser les applications. Et chaque service a un rôle dans une application répartie [22]

III.5.1. La recherche d'objets

Cette catégorie de services offre les mécanismes pour rechercher/retrouver dynamiquement sur le bus les objets nécessaires aux applications. Ce sont les équivalents des annuaires téléphoniques :

- Le service **Nommage** (Naming Service) [23] est l'équivalent des « pages blanches » : les objets sont désignés par des noms symboliques. Cet annuaire est matérialisé par un graphe de répertoires de désignation.
- Le service **Vendeur** (Trader Service)[24] est l'équivalent des « pages jaunes » : les objets peuvent être recherchés en fonction de leurs caractéristiques.

II.5.2. La vie des objets

Cette catégorie regroupe les services prenant en charge les différentes étapes de la vie des objets CORBA [19].

- Le service **Cycle de Vie** (Life Cycle Service) décrit des interfaces pour la création, la copie, le déplacement et la destruction des objets sur le bus. Il définit pour cela la notion de fabriques d'objets («Object Factory»).
- Le service **Propriétés** (Property Service) permet aux utilisateurs d'associer dynamiquement des valeurs nommées à des objets. Ces propriétés ne modifient pas l'interface IDL, mais représentent des besoins spécifiques du client comme par exemple des annotations.
- Le service **Relations** (Relationship Service) sert à gérer des associations dynamiques (appartenance, inclusion, référence, auteur, emploi,...) reliant des objets sur le bus. Il permet aussi de manipuler des graphes d'objets.
- Le service **Externalisation** (Externalization Service) apporte un mécanisme standard pour fixer ou extraire des objets du bus. La migration, le passage par valeur, et la sauvegarde des objets doivent reposer sur ce service.
- Le service **Persistance** (Persistent Object Service) offre des interfaces communes à un mécanisme permettant de stocker des objets sur un support persistant. Quel que soit le support utilisé, ce service s'utilise de la même manière via un «Persistent Object Manager». Un objet persistant doit hériter de l'interface «Persistent Object» et d'un mécanisme d'externalisation.
- Le service **Interrogations** (Query Service) permet d'interroger les attributs des objets. Il repose sur les langages standard d'interrogation comme SQL3 ou OQL. L'interface Query permet de manipuler les requêtes comme des objets CORBA. Les objets résultats sont mis dans une collection. Le service peut fédérer des espaces d'objets hétérogènes.
- Le service **Collections** (Collection Service) permet de manipuler d'une manière uniforme des objets sous la forme de collections et d'itérateurs. Les structures de données classiques

(listes, piles, tas,..) sont construites par sous-classement. Ce service est aussi conçu pour être utilisé avec le service d'interrogations pour stocker les résultats de requêtes.

- Le service **Changements** (Versioning Service) permet de gérer et de suivre l'évolution des différentes versions des objets. Ce service maintient des informations sur les évolutions des interfaces et des implantations. Cependant, il n'est pas encore spécifié officiellement.

1.5.3. La sûreté de fonctionnement

Cette catégorie de services fournit les fonctions système assurant la sûreté de fonctionnement nécessaire à des applications réparties.

- Le service **Sécurité** (Security Service) permet d'identifier et d'authentifier les clients, de chiffrer et de certifier les communications et de contrôler les autorisations d'accès. Ce service utilise les notions de serveurs d'authentification, de clients/rôles/droits (et délégation de droits), d'IIOP sécurisé (utilisant Kerberos ou SSL).
- Le service **Transactions** (Object Transaction Service) assure l'exécution de traitements transactionnels impliquant des objets distribués et des bases de données en fournissant les propriétés ACID.
- Le service **Concurrence** (Concurrency Service) fournit les mécanismes pour contrôler et ordonnancer les invocations concurrentes sur les objets. Le mécanisme proposé est le verrou. Ce service est conçu pour être utilisé conjointement avec le service Transactions.

1.5.4. Les communications asynchrones:

Par défaut, la coopération des objets CORBA est réalisée selon un mode de communication client/serveur synchrone. Toutefois, il existe un ensemble de services assurant des communications asynchrones.

- Le service **Evénements** (Event Service) permet aux objets de produire des événements asynchrones à destination d'objets consommateurs à travers des canaux d'événements. Les canaux fournissent deux modes de fonctionnement. Dans le mode « push », le producteur a l'initiative de la production, le consommateur est notifié des événements. Dans le mode « pull », le consommateur demande explicitement les événements, le producteur est alors sollicité.
- Le service **Notification** (Notification Service) est une extension du service précédent. Les consommateurs sont uniquement notifiés des événements les intéressant. Pour cela, ils posent des filtres sur le canal réduisant ainsi le trafic réseau engendré par la propagation des événements inintéressants.
- Le service **Messagerie** (CORBA Messaging [OMG-CM]) définit un nouveau modèle de communication asynchrone permettant de gérer des requêtes persistantes lorsque l'objet appelant et l'objet appelé ne sont pas présents simultanément sur le bus. Cela permet d'avoir des fonctionnalités proches des MOMs (« Message Oriented Middleware »).

1.5.5. Autres fonctions

- Le service **Temps** (Time Service) fournit des interfaces permettant d'obtenir une horloge globale sur le bus (Universal Time Object), de mesurer le temps et de synchroniser les objets.
- Le service **Licences** (Licensing Service) permet de mesurer et de contrôler l'utilisation des objets, et cela en vue de facturer les clients et de rémunérer les fournisseurs.

II-6- Processus de développement d'une application CORBA

Quand vous développez une application distribuée CORBA, il faut tout d'abord identifier les objets nécessaires à l'application, vous respecterez généralement les étapes suivantes :

Etape 1 : Définition de l'interface pour l'objet.

La première étape de la création d'une application CORBA consiste à spécifier vos objets et leurs interfaces en utilisant l'expert IDL(Interface Definition Language) de l'OMG, l'IDL une syntaxe semblable au C++, et peut être utilisé pour définir des modules, des interfaces, et des structures de données.

L'interface d'un objet permet de spécifier les opérations qu'il fournira, et comment elles devront être invoquées.

Exemple : *Bonjour.idl*

```
module BonjourApp
{
    interface Bonjour
   {
            string DirBonjour(in string nom , in string prenom);
    };
};
```

List 2.1 : Définition de l'interface CORBA

Etape 2 : Construction du Stub et Skeleton

A cette étape, vous utiliserez le compilateur IDL fournit avec l'ORB pour compiler l'interface, le rôle de la compilation est de :

- vérifier la syntaxe du code écrit en Idl.
- Générer le Stub coté client.
- Générer le skeleton coté serveur.

Cette étape est appelé aussi le **mapping,** qui est l'opération de projection d'un code IDL vers un langage de programmation..

Les deux modules Stub et Skeleton générés sont utilisés par le client pour invoquer des méthodes sur un objet.

Exemple :

Pour générer le stub et skeleton de notre exemple à partir de notre interface Bonjour.idl :

1- avec VisiBroker Java, vous disposez du compilateur IDL2JAVA pour convertir votre interface en utilisant la ligne commande suivante :

idl2java Bonjour.idl

2- avec VisiBroker C++, vous disposez du compilateur IDL2CPP pour convertir votre interface en utilisant la ligne commande suivante :

idl2cpp -scr_suffix cpp -hdr_suffix h Bonjour.idl

Etape 3 : Implémentation de l'objet reparti.

L'implémentation d'un objet CORBA est basé sur les codes skeleton généré lors de la deuxième étape par l'IDL, qui consiste à fournir le développement et le code source pour chaque méthode disponible dans l'interface, en utilisant un langage de programmation suivant l'ORB disponible.

exemple : *BonjourImpl.java*
Implémentation en Java de notre objet Bonjour.

```
Package BonjourApp ;
Public class BonjourImpl extends _BonjourImplBase
{       // Construction de l'objet
          public BonjourImpl(java.lang.String name)
          {
                    super(name);
          }
          public DirBonjour(String nom , String prenom )
        {
           return "Bonjour" + nom + " " + prenom ;
          }
}
```

List 2.2 : Implémentation de l'objet CORBA

Etape 4 : Ecriture et lancement d'un serveur.

La quatrième étape consiste à :

- écrire le programme de serveur qui crée des instances pour chaque objet implémenté, et enregistre ces objets au niveau de service de désignation (nommage), afin de les mettre en disposition des clients .La logique de fonctionnement d'un serveur CORBA est :

- initialiser l'ORB
- initialiser le BOA
- créer l'objet
- enregistrer l'objet
- la mise en attente.
- le lancement de serveur et de l'ORB (exe : SmartAgent pour VisiBroker)

exemple : ServeurBonjour.java

```
Package BonjourApp ;
Import java.util.*
Public class  ServeurBonjour
{
   public static void main(String[] args)
  {
     try
     {
        org.omg.CORBA.ORB orb  = org.omg.CORBA.ORB.init() ;
        org.omg.CORBA.BOA boa = org.BOA_init() ;
        BonjourApp.Bonjour  implObject = new BonjourImpl("bonjourDistant");
        boa.obj_is_ready(implobject);
        boa.impl_is_ready() ;
     }
      catch(Exception e)    {         e.printStackTrace() ;     }
   }
 }
```

List 2.3 : Ecriture et lancement d'un serveur

Etape 5 : Développement et lancement de l'application cliente.

Cette étape consiste à écrire le programme client qui localise les objets du serveur et invoque des services sur ces objets

Le fonctionnement d'un client est :

- Initialiser l'ORB.
- Connecter à l'objet.
- L'utilisation de l'objet(c à d : appels des méthodes et lecture des propriétés).

```
Package BonjourApp ;
Import java.util.*
Public class  ClientBonjour
{
   public static void main(String[] args)
  {
     try
     {
        org.omg.CORBA.ORB orb ;
       Bonjour bonjour ;
       orb  = org.omg.CORBA.ORB.init() ;
       String nom = "   xxxxx " , prenom = "yyyyyyyyyy";
       bonjour = BonjourHelper.bind(orb,"BonjourDistant");
       System.out.println(bonjour.DirBonjour(nom , prenom ));

     }
      catch(Exception e)
     {
       e.printStackTrace() ;
      }
    }
 }
```

List 2.4 : développement de Client CORBA

II-7- Conclusion

Ce chapitre présente le premiers middleware de notre étude de comparaison, nous avons vu dans ce dernier tous les concepts lies a la manipulation de cette architecture, CORBA décrit une architecture et définit les spécifications pour le traitement d'objets distribués sur le réseau

CORBA est un vrai Middleware permettant à une application cliente d'effectuer une requête à un objet distant (serveur) et de recevoir en retour le résultat de l'opération effectuée. Tout le travail de communication est automatiquement effectué par le Middleware (processus de localisation des objets distribués, la gestion des erreurs de communications, le Un/Marshaling, etc..).

CORBA, qui est un vrai concurrent dans un choix de réalisation d'une application distribuée, doit être prise en compte.

CHAPITRE III

LE MIDDLEWARE
- RMI -

Dans ce chapitre, nous allons voir les composants et les caractéristiques introduits dans RMI, et connaître le mécanisme d'invocation de méthodes distantes.

Enfin, nous aborderons les principes de base de RMI en les illustrant par un exemple d'application distribuée.

Mot clé :

RMI, Java, Sun Microsystem, Object Registry(**rmiregister)** , Remote Method Invocation

Chapitre III

LE MIDDLEWARE - RMI -

Remote Method Invocation

III-1- Introduction

Après l'apparition de CORBA et les succès qu'il à remportés à grand échelle, comme un middleware d'objet réparti, la société Sun Microsystem développé un nouveau mécanisme ou middleware d'invocation des méthodes éloigné, ce middleware est devenu un concurrent de CORBA et permet au société Sun de rejoindre l'évolution technique dans les mécanismes du système reparti, ce middleware est introduit avec le langage Java, sous le nom RMI (Remote method invocation), il constitue une nouvelle génération d'outils middleware important à étudier

III-2- Définition

RMI (Remote Method Invocation) [25] est middleware qui permet de réaliser des applications distribuées Java à Java, dans lesquelles les méthodes d'objets éloignés peuvent être invoquées à partir d'autres machines virtuelles pouvant résider dans d'autres ordinateurs.

RMI est une architecture proposée par la société Sun, comme une plate forme distribuée intégrer dans le langage Java.

L'architecture de Sun a réussi à rendre transparente l'utilisation d'objets distants. Après avoir localisé les objets appropriés, le programmeur utilise les méthodes de ces objets comme si ces dernières faisaient partie de son application. Tout le travail de codage, décodage, de vérification et de transfert est effectué par RMI.

III-3- Le langage Java

Le terme Java désigne à l'origine un nouveau langage de programmation orienté objet, inspiré du langage C++ dépouille de certaines fonctions qui en empêchaient la portabilité, la caractéristique majeure du langage Java est sa portabilité, c'est-à-dire sa capacité de fonctionner sur n'importe quel système d'exploitation, c'est pourquoi la phrase suivante est devenue l'emblème de java « Write Once, Run Anywhere »

Le langage Java a été conçu afin d'écrire du code téléchargeable par réseau sur n'importe quelle machine, ce but imposa un certaine nombre de contraintes dont les deux suivant :

- Le code à télécharger doit être aussi compact que possible afin de réduire le trafic sur le réseau
- Le langage doit pouvoir s'exécuter sur n'importe quelle machine

La première contrainte fut résolue en utilisant un langage intermédiaire appelé byte code ,ce langage se caractérise par un encombrement minimal et une complète indépendance vis-à-vis des langages machines, ainsi le code d'un programme écrit en Java est traduit(compilé) lors de sa construction dans le langage de byte code. Lors de sa phase d'utilisation, c'est le byte-code qui circule sur le réseau. Le byte code[26] est un code binaire, ce qui permet un traitement plus rapide que le code source Java, et qui rassemble (compile) tous les codes dispersés dans différents fichiers lors de l'écriture du programme. Le byte code Java [27] est l'ensemble des instructions exécutables par une machine virtuelle java

La deuxième contrainte, à savoir la portabilité d'un programme Java, a été levée en utilisant un interpréteur pour le byte code. Cette solution suppose que sur chaque système d'exploitation où l'on souhaite exécuter un programme Java existe cet interpréteur.

Ce schéma de fonctionnement impose l'existante, sur la machine réceptrice d'un interpréteur et d'un module de vérification, ces deux composants appartiennent à ce qui est appelé la machine virtuelle Java (JVM).

III-4- Structure de la machine virtuelle Java

Afin que le code Java puisse s'exécuter sur n'importe quelle système d'exploitation celle-ci doit disposer d'un environnement logiciel spécifique. Cet environnement est constitué par un ensemble de modules formant ce qui est appelé par la société Sun, la machine virtuelle Java (JVM). Celle-ci comprend [28]:

- **Le vérificateur**, ce module vérifie que le code téléchargé ne comporte pas d'opérations illégales, en particulier, il s'assure qu'aucun accès au disque local n'est possible. Il vérifie également que seules les requêtes réseaux existantes sont à destination de la machine d'où provient ce code.
- **L'interpréteur**. Le code Java est interprète localement par ce module
- **La bibliothèque**. Un certain nombre de classe Java manipulant des objets standard sont très fréquemment utilisées, afin d'éviter leur téléchargement, ces classes sont stockées localement dans cette bibliothèque.
- **Les méthodes natives**. Certaines opérations (en particulier les graphiques) sont dépendantes de la plate forme sur laquelle fonctionne la JVM, ces opérations portent le nom de méthodes natives.
- **Le ramasse miettes**. Java est un langage oriente objet dans lequel des objets sont crées puis détruits. La mémoire allouée à ces objets lors de leur création doit pouvoir être libérée lors de leur destruction.
- **Le compilateur JIT**. Ce composant est optionnel et est proposé afin d'accélérer l'exécution d'une application, ce module en fait d'abord la compilation ce qui évite la phase d'interprétation

La Java virtual machine (abrégé JVM, en français machine virtuelle Java) est une machine virtuelle permettant d'interpréter et d'exécuter le byte code Java.

III-5- Avantages

Les principaux avantages à utiliser RMI sont les suivants:

- Il permet le transfert transparent d'objets d'une application à une autre et ces objets ne subissent pas de changement lors des transferts (l'intégrité des objets transférés est respectée).

- Son utilisation est sécuritaire (les classes de gestion de la sécurité sont disponibles et le programmeur peut, s'il le désire, programmer les siennes).
- il est relativement facile et rapide d'écrire des applications Java et d'utiliser les classes RMI.
- Il permet un accès facile et simplifié aux systèmes déjà en place (utilisant des langages comme Java, C, C++, ...).
- Il élimine le coulage de mémoire par l'utilisation du ***Garbage Collection***.
- Il permet l'exécution en parallèle d'applications (locale ou distante).
- Le transfert de comportements (objets) d'une application à une autre permet d'exploiter toute la force des ***Design Patterns*** (qui s'appuient très souvent sur des comportements).

III-6- Architecture

Java RMI permet[29] la communication entre deux entités Java localisées dans deux machines virtuelles Java distinctes, Java RMI implante le modèle client/Serveur dans lequel le client est soit une application Java et le serveur est un objet appartenant à une application Java.

Afin d'accéder à l'objet serveur le client Java doit obtenir sont adresse, il obtient celle ci en le demandant à une entité appelée le Registry, le Registry agit comme un serveur de Noms, il possède une table dans laquelle à chaque nom d'objet est associée sa référence, Chaque référence contient l'interface et l'adresse de l'objet.

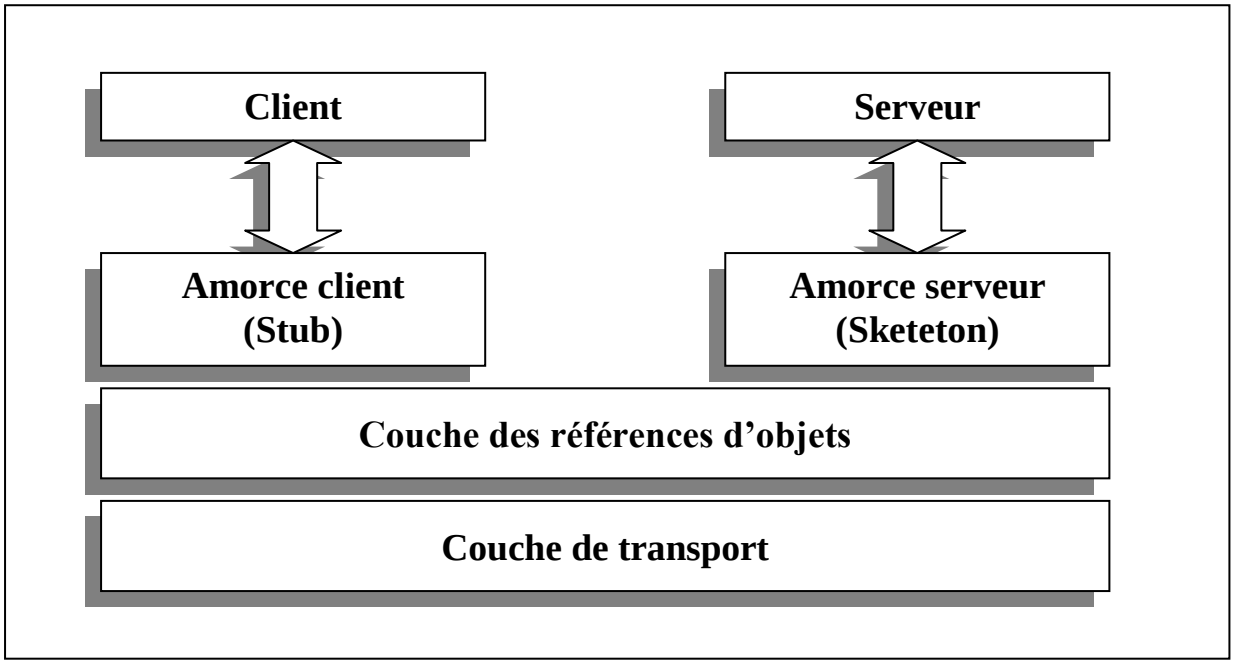


Figure 3-1 : Architecture d'un middleware RMI

Il est important de noter que chaque objet doit avoir un nom unique qui le distingue des autres objets, et pour assurer la communication entre le client et le serveur, L'architecture du RMI est divisé en plusieurs couches qui permet la modification d'une de ces couches sans affecter le reste du système, et une requête cliente doit traverser ces couches.

III-6-1- La couche d'application

La première couche est constituée par les modules client et serveur, on retrouve les appels de haut niveau pour exporter les objets, ou les accéder.

- le serveur qui expose les objets, et la manière d'accéder à ces objets.

- Le client qui initialise les appels de haut niveau, et fait les demande d'utilisation des objets exposés.

L'architecture RMI est basée sur le fait que le comportement exporté (les méthodes) est défini dans l'interface (définition des services). L'interface ne contient pas de code exécutable et elle hérite de Java.rmi.Remote. L'interface Remote est essentiellement utilisée pour déclarer aux éventuels clients l'accessibilité à distance d'un objet. Son implémentation est réalisée dans une classe du module serveur.

III-6-2- Couche proxy ou Stub

Cette couche est composée de deux parties le stub et le Skeleton [30], les deux parties sont générées lors de la compilation de l'implémentation de l'interface.

- Le Stub client :

Le stub client est le lien entre le client et l'objet distant, il a comme fonction :

- Représentant local de l'objet distant qui implémente ses méthodes « exportées »
- Transmet l'invocation distante à la couche inférieure « Remote Reference Layer »
- Il réalise le pliage (« marshalling ») des arguments des méthodes distantes, Dans l'autre sens, il réalise le dépliage (« unmarshalling ») des valeurs de retour

- Le Skeleton :

-Le skeleton est localisé coté serveur et fonctionne directement avec l'implémentation des méthodes exportés, ces fonction essentielles sont :

- Recevoir les appels de client.
- Réaliser le dépliage « unmarshalling » des arguments reçus par le flux de pliage.
- Faire un appel à la méthode de l'objet distant.
- Réaliser le pliage « marshalling » de la valeur de retour.

III-6-3- La couche de références distantes

Cette couche définit et supporte la sémantique des invocations de la connexion RMI , elle interprète et gère les références des objets distants, la couche de référence distantes est en fait une couche abstraite entre le Stub, le skeleton et les protocoles de communication, elle permet :

- l'obtention d'une référence d'objet distant à partir de la référence locale au Stub
- Ce service est assuré par le lancement du programme **rmiregister**
- à ne lancer qu'une seule fois par JVM, pour tous les objets distants à distribuer
- une sorte de service d'annuaire pour les objets distants enregistrés

III-6-4- la couche transport

C'est la couche qui effectue la connexion entre les machines virtuelles

- elle est responsable de la mise en contact des machines virtuelles (connecte les 2 espaces d'adressage JVM).
- Assurer la gestion et le suivi des connexions en cours
- Ecoute et répond aux invocation.
- Construit les tables des objets distants disponibles.
- Réalise l'aiguillage des invocations.

III-7- Principe de fonctionnement

Fonctionnement côté serveur :

- 1) L'objet serveur s'enregistre auprès du service de noms RMI via la classe Naming de sa JVM (méthode bind ou rebind)
- 2) L'objet squelette est créé, celui-ci crée le port de communication et maintient une référence vers l'objet serveur
- 3) Le Naming enregistre l'objet serveur, et le port de communication utilisé auprès du serveur de noms.

Fonctionnement côté client :

- 4) L'objet client fait appel au Naming de sa JVM pour localiser l'objet serveur (méthode lookup)
- 5) Le Naming récupère une "référence" vers l'objet serveur.
- 6) Crée l'objet souche (Proxy).
- 7) Renvoie la référence de la souche au client
- 8) Le client appelle des méthodes de l'objet serveur a travers de la souche et du squelette (skeleton).

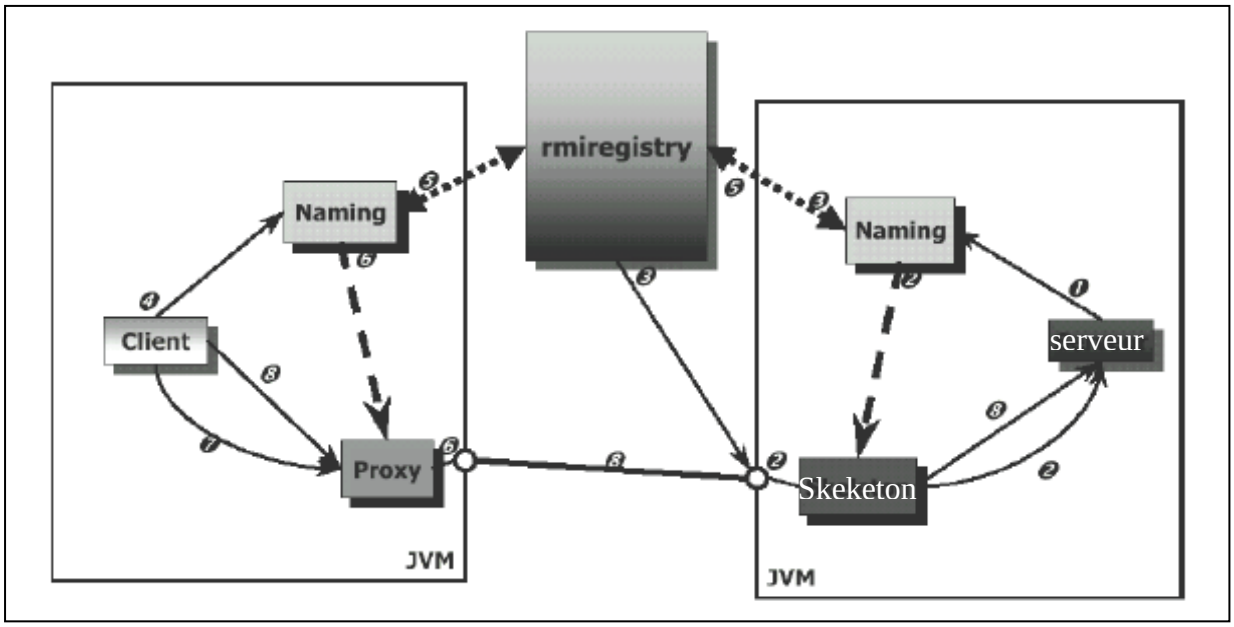


Figure 3-2 : le Principe fonctionnement d'une application RMI

III-8- Les services du RMI

-1- Service de nommage [31] :

Pour pouvoir gérer plusieurs interfaces, il est nécessaire de les identifier. Cette identification passe par un service de nommage. Ce service permet à un client de rechercher un serveur à partir d'un nom (sous forme d'une chaîne de caractères).

-2- Service d'activation d'objets :

Permet d'avoir des objets serveurs activables à la demande, pour éviter d'avoir des objets serveurs actifs en permanence

-3- Ramasse-miettes réparti :

Permet de récupérer les ressources (mémoire, ...) occupées par un objet que personne ne référence Comme RMI est désigné de telle sorte que sa sémantique soit la plus proche

possible de Java, et que Java utilise un ramasse-miette, RMI possède donc aussi un ramasse-miette mais celui-ci est distribué.

Les serveurs comptent le nombre de clients connectés à un objet exporté fournissant un service. Quand ce nombre tombe à zéro, l'objet peut être ramassé.

III-9- Processus de développement d'une application RMI

La construction de deux entités Java dont l'une « le client » souhaite émettre des demandes d'exécution de méthodes sur l'autre (le serveur) à l'aide du mécanisme RMI, se décompose en cinq étapes :

1- Première étape : définition de l'interface pour la classe distante.

Cette interface va permettre de spécifier les différentes méthodes applicables à l'objet [32]. Afin de permettre l'appel des méthodes de l'interface, celle-ci doit hériter de l'interface **java.rmi.Remote** .

Par ailleurs chacune des méthodes de l'interface doit être en mesure de prendre en compte les différentes exceptions distantes susceptibles d'être rencontrées lors de la détection d'une anomalie **java.rmi.RemoteException**.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Bonjour extends Remote
{
  public String DirBonjour(String nom,String prenom) throws RemoteException ;
}
```

List 3-1 *:* définition de l'interface pour un objet RMI

2- Deuxième étape : implantation de l'interface côté serveur.

```
import java.rmi.*;
import java.rmi.server.*;
      // Classe d'implantation BonjourImpl
public class BonjourImpl extends UnicastRemoteObject implements Bonjour
{
       // Son constructeur
   public BonjourImpl() throws RemoteException
  {
       super();
  }
      // Implémentation de la méthode à distance que le client va invoquer
  public String DirBonjour(String nom,String prenom) throws RemoteException
  {
       return "Bonjour : "+ nom+ " " +prenom;
  }
 }
```

List 3-2 *:* implantation de l'interface côté serveur RMI

Il s'agit ici de donner une implantation de l'interface définie lors de la première étape, c'est cette implantation qui permettra de rendre un objet accessible à distance par des clients.

L'implantation consiste à hériter une classe de la classe j**ava.rmi.server.UniCastRemoteObject ,** Cette classe du package **rmi** fournit des implantations de nombreuses méthodes de **java.lang.Object** adaptées aux objets distants.

La manière la plus simple de définir une classe implantant une interface pour un objet distant tel que le code précédent.

3- Troisième étape : construction du talon et du squelette.

A partir du code source qui correspond à la classe implantant l'interface, le stub et le squelette sont générés au moyen de l'utilitaire **rmic** , le nom des deux fichiers est déduit de celui de la classe implantant l'interface, et ils ont respectivement **_Stub.class** et **_Skel.class** comme suffixes.

La construction de stub et squelette correspondant au classe **CompteurImpl** à partir des différents fichiers sources java est réalisée comme suit:

```
--> javac Bonjour.java
--> javac BonjourImpl.java
--> rmic BonjourImpl
```

4- Quatrième étape : écriture et lancement d'un serveur.

Pour écrire le code source d'un serveur il faut suivir les opérations suivantes :

```
import java.rmi.*;
import java.rmi.server.*;
public class BonjourServeur
{
     public static void main(String[] args) throws Exception
    {
            // Création et installation du gestionnaire de sécurité
        System.setSecurityManager(new RMISecurityManager());
            // Création et enregistrement de l'objet à distance Serveur
        try
        {
           Bonjour serveur = new BonjourImpl();
           Naming.rebind("monObjet", serveur);
           System.out.println("Le serveur est pret");
         }
         catch(Exception e)
        {
            throw e;
         }
     }
}
```

List 3-3 : écriture et lancement d'un serveur

- Définir un gestionnaire de sécurité (objet de la classe RMISecurityManager**)** qui autorisera le chargement depuis une autre application.
- Créer les objets distants par instanciation de classes Remote.
- Exposer les objets, c'est-à-dire les rendre accessibles dans le serveur.
- Faire connaître l'existence des différents objets au serveur de noms (**Object Registry**) de la machine sur laquelle ce serveur s'exécutera.

- Lancement du serveur de noms : le serveur de noms (**Object Registry**) doit évidemment être activé s'il ne l'a pas encore été avant de pouvoir lancer le serveur d'objets. Ce lancement est réalisé par la commande **rmiregistry**.
- Lancement d'un serveur d'objets : Cette opération consiste simplement en le lancement d'une application Java.:

```
→ start rmiregistry
→ javac BonjourServeur.java
→ java BonjourServeur.java
```

5- Cinquième étape : développer et lancer de l'application cliente.

Cette étape décrie brièvement les différentes opérations que devra réaliser un client afin de pouvoir accéder à un objet distant (pour lequel évidemment un serveur du type précédemment décrit est supposé avoir été lancé avec succès).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.rmi.*;
public class BonjourClient
{
   public static void main(String[] args) throws Exception
   {
      Bonjour serveur = null;
      String nomComplet;
      nomComplet ="rmi://localhost:1099/" + "monObjet";
      try
      {
          serveur = (Bonjour) Naming.lookup(nomComplet);
         String rep = serveur.DirBonjour("Monde", "distribue");
         System.out.println(rep);
      }
     catch(Exception e)
     {
           System.out.println("exeption"); throw e;
      }
   }
}
```

List 3-4 *:* développer de l'application cliente

Le client doit tout d'abord s'adresser au serveur de noms auprès duquel l'objet auquel il souhaite accéder a été enregistré afin de récupérer un talon de l'objet (objet de la classe **Remote**). Cette opération est réalisée en invoquant la méthode statique **lookup** de la classe **Naming**. Pour cela, il doit nommer complètement l'objet concerné au travers d'une URL de la forme générale suivante :

rmi://machine:port/nom

- si le nom de machine est omis, la machine locale est considérée;
- si le numéro de port est omis, le numéro par défaut (1099) est considéré.
- Une fois le talon récupéré, un client peut invoquer sur ce talon des méthodes de l'interface comme il ferait sur un objet local

- Lancement d'un client : Cette opération consiste simplement en le lancement d'une application Java.:

```
→ javac BonjourClient.java
→ java  BonjourClient.java
```

III-10-Conclusion

Le langage Java introduit de nouveau mécanismes Java RMI permettant d'implanter des objets distribués, ce mécanisme constitue une nouvelle génération d'outils middleware importants à étudier.

Java RMI propose une méthode relativement simple et puissante pour distribuer des applications sur des machines différentes. Java RMI est un exemple de technologie orientée objet (grâce à Java) permettant de distribuer une application (grâce à RMI).

Ce troisième chapitre a été consacré à la définition de middleware Java RMI, des moyens qui l'assurent et de ses avantages. En plus, nous avons étudié son fonctionnement et les protocoles sur lesquels il est construit, et nous avons cité les différentes étapes de développer une application répartie Java RMI.

CHAPITRE IV

LE MIDDLEWARE
- DCOM -

Ce chapitre sera réservé à la plat-forme DCOM proposée par MicroSoft, nous nous détaillerons les différents aspects de ce modèle.

Et nous finirons par donner une mise en place d'une application distribuée DCOM

Mot clé :

DCOM, COM , RPC , GUID , MIDL , DCOMCNFG ,base de registres , libraire(TLB)

Chapitre IV

LE MIDDLEWARE - DCOM -

Distributed Component Object Model

IV-1- Introduction

La conception et le développement d'applications clients/serveur performantes sur des plates-formes Windows 32 bits nécessitent l'utilisation des nouvelles technologies et architectures Microsoft.

Le COM [33], (ancien modèle de Microsoft) vous permet d'écrire des composants réutilisables, mais ce qui arrive si vous voulez créer des applications distribuées, c'est-à-dire vos composants doivent être placées sur un autre ordinateur ? Microsoft a complété la technologie COM afin de permettre la répartition des composants sur un réseau. L'aspect distribué des composants COM ainsi répartis a donné le nom à cette extension de COM, que l'on appelle simplement « DCOM » (pour « Distributed COM »).

IV-2- Définition

Le DCOM [34] est un extension du Modèle d'Objet Composant COM à La structure de Microsoft qui soutient la programmation des objets. DCOM soutient l'appel à des objets distants par le protocole nommé [35] ORPC (Object Remote Procedure Call).

Le DCOM est le système d'objets distribués de Microsoft, est une interface ActiveX destinée aux applications réseau de composants distribués. Le modèle DCOM offre également une transparence au niveau du réseau et une automatisation des communications, de sorte que les communications peuvent intervenir entre des objets sans qu'un objet connaisse nécessairement l'emplacement de l'autre objet. Les objets peuvent se situer dans des processus différents sur la même machine ou sur des machines distinctes.

Le DCOM est le modèle d'architecture distribuée incontournable sur plate-forme Windows 32 bits il apporte des facilités importantes au niveau de la sécurité et de la distribution des composants mais aussi des gains notables de performances. DCOM facilite en outre le développement de composants réutilisables.

IV-3- Les caractéristiques

1- L'utilisation de DCOM cache complètement l'emplacement du composant. Cela si le composant est dans le même poste de travail ou La méthode est employée dans le poste de travail différent.

2- Le modèle COM distribué permet aux développeurs de l'entreprise de diviser une application en modules de composants qui peuvent tous s'exécuter de façon transparente *sur* différents ordinateurs. Grâce à cette transparence réseau, ces composants semblent, pour les utilisateurs et les programmeurs, être situés sur une même machine.

3- DCOM est essentiellement comme COM, mais à l'exception du fait qu'il sert sur une largeur plus grande (distribué).Le standard composant COM est exécuté sur le même ordinateur et dans le même processus que l'application.

- transparence à la localisation des objets
- activation d'objets dynamique et à distance (Service Control Manager)
- sécurité des communications (Security Account Manager)
- accès aux objets via des interfaces
- support du cycle de vie des objets (création, destruction)
- découverte dynamique des interfaces implantées par un objet
- interopérabilité binaire entre langages hétérogènes
- détection des pannes de serveurs
- invocation de méthode statique et dynamique (automation)
- mécanisme de gestion d'événements distribués

4- Une application cliente accède à un objet via un pointeur sur une interface de l'objet. Pour cette interface, COM définit une structure binaire indépendante du langage d'implémentation de l'objet.

IV-4- Les Composantes d'une application COM

Quand vous implémentez une application COM, vous fournissez ceci :

Interface COM :

Le moyen par lequel un objet expose ses services aux clients. Un objet COM fournit une interface pour chaque ensemble de méthodes (fonctions membre) et de propriétés (membres de données et/ou contenu).

Serveur COM :

Un module, EXE, DLL ou OCX, contenant le code d'un objet COM. Les implémentations d'objets résident sur les serveurs. Un objet COM implémente une ou plusieurs Interfaces.

Client COM :

Le code appelle les interfaces afin d'obtenir du serveur les services demandés. Les clients savent ce qu'ils veulent obtenir du serveur (via l'interface) ; les clients ne savent pas comment en interne le serveur fournit les services.

IV-5- L'architecture de DCOM

DCOM [36] est une extension de COM, COM définit comment les composants et des clients agissent réciproquement avec chacun d'autre.

On a trois types d'architectures, qui nous permettent de communiquer entre un client et un objet serveur suivant l'emplacement de ce dernier.

1. l'objet serveur et le client en même processus :

Les composants des serveurs fonctionnent dans le même processus que ces clients, cette architecture est appelée **serveur in-process,** dont l'interaction est définie pour que la connexion entre le client et le composant peut être fait sur n'importe quels composants intermédiaires de système, Cela permet au client faire appels à méthodes sans ajouter n'importe quelle information.

En pratique, les serveurs **in-process** sont les serveurs DLL.

L'avantage des **serveurs in-process** est qu'ils sont dans le même espace d'adressage que leur client. Ceci implique :

- qu'il est possible d'accéder directement à leurs composants, et de créer ceux-ci sans passer par DCOM ;
- que les appels des méthodes de leurs composants sont effectués directement, donc plus efficaces que pour les composants des serveurs exécutables ;
- qu'il est possible de réutiliser leurs composants à l'aide du mécanisme d'agrégation.

L'inconvénient des serveurs **in-process** est que les erreurs dans leurs composants peuvent entraîner la terminaison du client.

Figure 4-1 : Architecture d'un serveur in process

2. l'objet serveur et le client en différent processus

Dans cette architecture, l'objet serveur s'exécute dans un autre processus que le processus client. En pratique, ces serveurs sont des exécutables, qui enregistrent leurs composants lors de leur initialisation, cet objet serveur est un serveur **out-of-process**, la communication est assurée par le protocole de communication interprocessus locale(LPC).

Les avantages des serveurs **out-of-process** sont les suivants :

- les clients sont mis à l'abri des fautes des serveurs. La terminaison du serveur n'engendre pas la terminaison du client ;
- les serveurs **out-of-process** peuvent partager des données entres les différents clients.

Les inconvénients des serveurs **out-of-process** sont les suivants :

- les appels aux méthodes de leurs composants nécessite le passage des paramètres du processus client au processus serveur, ce qui est bien évidemment plus lent ;
- ils ne peuvent pas être utilisés en tant qu'objets agrégés.

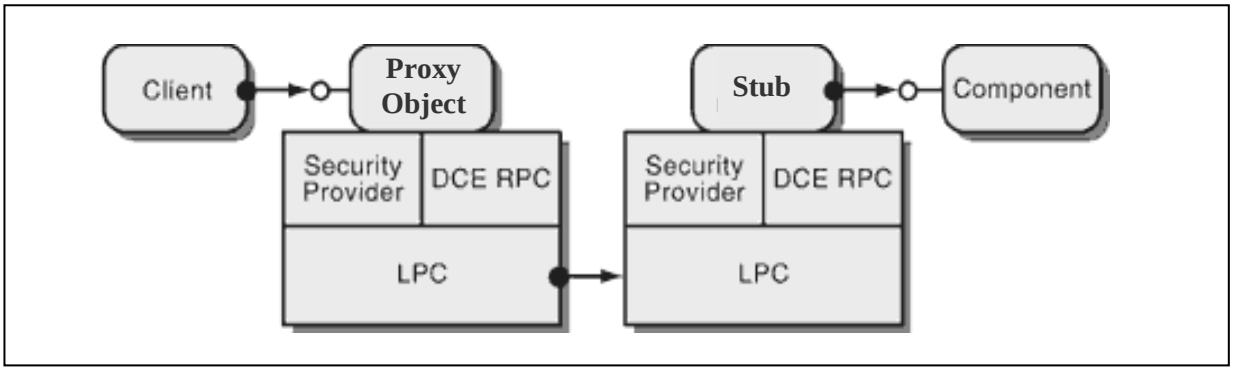


Figure 4-2: Architecture d'un serveur out-of-process

3. l'objet serveur et le client en différente machine :

Lorsque le client et les composants serveurs résident sur des machines différentes, DCOM remplace simplement la communication interprocessus locale avec un protocole réseau. Cette architecture qui nous intéresse puisque elle assure la communication entre des objets à distant

Les différents composants de cette architecture sont résumés [37] dans la figure suivante:

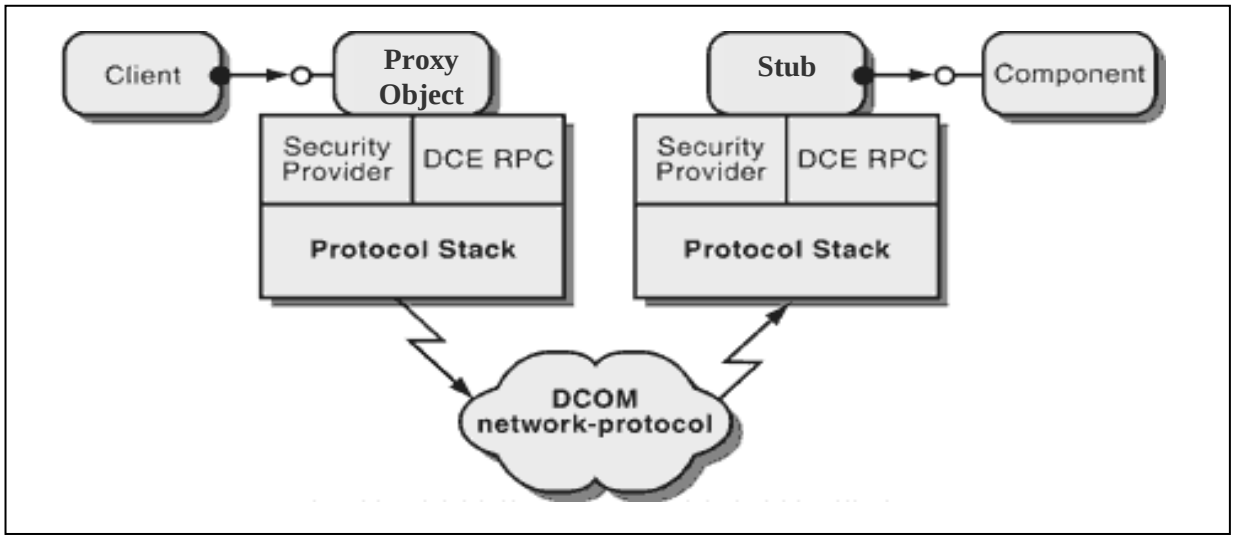


Figure 4-3: Architecture d'un serveur out-of-machine

Dans cette architecture on trouve les composants suivants :

- Proxy Object:

- Il représente l'objet coté client
- Il expose les interfaces que l'objet expose
- Il sait comment communiquer avec le stub

- Stub:

- Il représente le client coté objet
- Il expose les interfaces que le client utilise
- Il sait comment communique avec le proxy.

- DCE-RPC : (Distributed Computing Environment - Remote Procedure Call) est un ensemble d'outils, qui tournent sur un OS, servant à la création et au déroulement d'applications distribuées.

C'est un protocole réseau permettant de faire des appels de procédures sur un ordinateur distant à l'aide d'un serveur d'applications. Il permet de gérer les différents messages entre les différentes entités d'une application client et serveur.

- **Security provider** (fournisseur de sécurité) : il offre un ensemble intégré de services de sécurité doté des fonctions suivantes :

- modèle d'environnement sécurisé
- interface SSPI (Security Support Provider Interface) ;
- cryptographie ;
- authentification numérique dans le chargeur d'initialisation à distance.

Avant autorisé à une application l'accès en écriture à des parties du système, il faut vérifier la signature de cette dernière pour protéger votre système contre les applications inconnues. Cela garantit que la plate-forme Windows charge une application uniquement si elle contient une signature numérique

- Protocol Stack (pile de protocole) : La partie logicielle de communication entre les couches de réseau

IV-6- Les services fournis par DCOM

1. **interface :**

DCOM regroupe les fonctionnalités des composants par interfaces. Une interface est un jeu de fonctions, qui permettent d'utiliser un composant. Cependant, un composant peut implémenter plusieurs interfaces.

Les interfaces sont identifiées dans le système par des nombres uniques à 128 bits. Ces nombres sont appelés les GUID (« Globally Unique IDentifier », soit « Identificateur globalement unique »), ou UUID (« Universally Unique IDentifier », soit « Identificateur universel unique »). Un utilitaire, UUIDGEN.EXE, est fourni pour générer de tels nombres. Ces nombres sont calculés pour être absolument uniques dans le monde et en tout temps

2. **Transparence :**

Pour assurer la transparence au niveau du réseau, DCOM utilise les RPC (« Remote Procedure Call », ou « Appels de procédures à distance »). Il facilite ainsi énormément la gestion du réseau pour les programmeurs. En particulier, il prend en charge les communications et le marshalling, c'est à dire l'encapsulation du côté client des paramètres donnés par le client à la fonction appelée, ainsi que la reconstitution de ces paramètres du coté serveur pour l'appel de la fonction. Bien entendu, DCOM reste ouvert et permet au programmeur de réaliser son propre marshalling, ou de contrôler son propre protocole de communication.

3. **la sécurité :**

DCOM est sécurisé par les droits d'accès des utilisateurs. Ceci signifie qu'il est nécessaire de disposer d'un serveur de domaine pour donner ces droits (en pratique, ce serveur est un poste Windows NT4 Server). DCOM gère également les licences d'utilisation pour les composants commerciaux.

4. **la durée de vie :**

DCOM gère la durée de vie des objets OLE par compte de références sur ces objets. Les objets ne sont détruits que lorsque tous les clients de ces objets ont relâché leur référence qu'ils détenaient sur eux.

5. **le stockage :**

DCOM permet de stocker la description des interfaces dans ce qu'on appelle des « type libraries ». Ces librairies stockent la description des interfaces, des foncions des interfaces, de leurs paramètres et de leurs rôles. Elles permettent donc à un tout le monde d'utiliser un composant même sans en avoir la documentation.

IV-7- Principe de fonctionnement

Afin de présenter le fonctionnement d'une application répartie, il faut préciser le sens du concept client / serveur dans le modèle DCOM.

-**Le client** est n'importe quelle séquence de code faisant appel aux services d'un objet, Un client peut aussi faire des requêtes vers les services offres par DCOM afin de demander la création d'instances d'objets.

-**Le serveur** est une séquence de code associé à une classe d'objets donnée identifiée par un identifier de classe unique, le serveur sait créer des instances d'objets de cette classe et sait leur associer un pointeur d'interface.

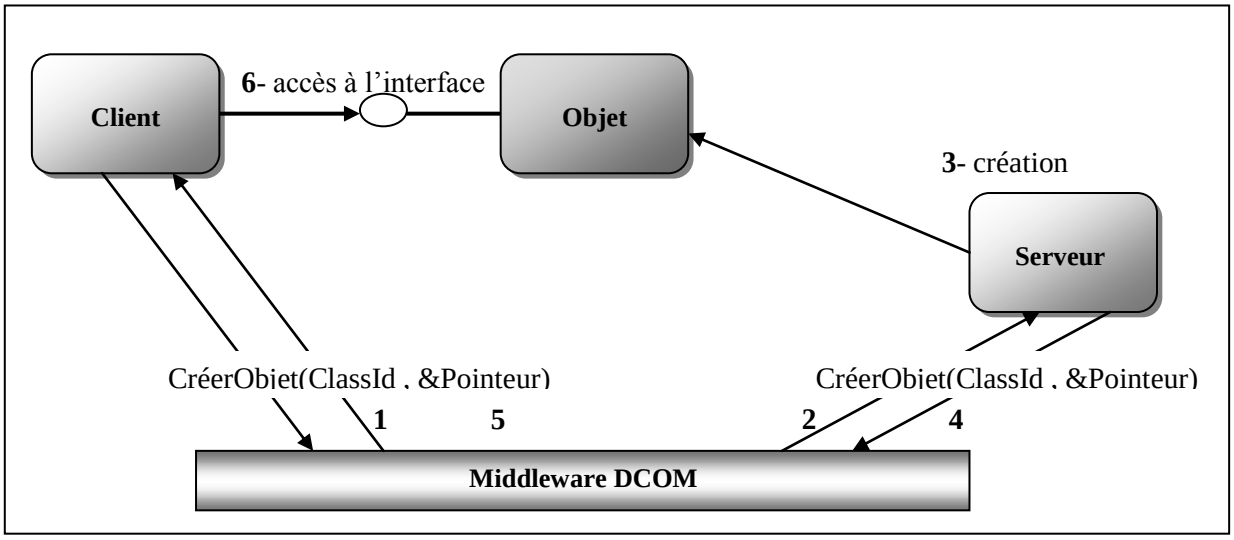


Figure 4-4 : le Principe fonctionnement d'une application DCOM

Dans ce schéma le client ne communique jamais avec un serveur, il dialogue soit avec un objet afin de demander l'exécution d'une fonction, soit avec l'infrastructure de DCOM, pour l'exécution de requêtes, par exemple :

1- Le client peut émettre une requête vers le middleware DCOM, il demande la création d'un objet d'une certaine classe

2- DCOM est en mesure de charger en mémoire et d'activer le code du serveur associé à cette classe et de lui transférer la requête du client

3- le serveur crée une instance d'objet demandé.

4- le serveur passe un pointeur d'une de ses interfaces à DCOM.

5- DCOM transmet ce pointeur au client.

6-le client peut alors accéder à l'interface de l'objet nouvellement créée.

Ainsi client et serveur ne communiquent jamais directement.

IV-8- Processus de développement d'une application DCOM

Le développement d'une application DCOM nécessite le passage par les étapes suivantes :

Etape 1 : Définition de l'interface pour l'objet.

Les objets et les interfaces de ces objets sont identifiés de manière unique dans le système par les GUID, afin de les référencer d'une manière complètement indépendante, les GUID sont des nombres à 128 bits exprimés en hexadécimal, Le format des GUID générés par l'utilitaire GUIDGEN.EXE (fournit avec VC++).

On peut donc en déduire que toute la configuration de DCOM est stockée dans la base de registre, et que DCOM est lui-même profondément intégré dans le système.

Exemple : BonjourApp.idl

```
[
        uuid(0C093940-E57B-4BEC-9ED9-D865FF0F4DF0),
        version(1.0),
        helpstring("BonjourApp (bibliothèque)")
]
library BonjourApp
{
        importlib("stdole2.tlb");
        [
              uuid(7BE98DB1-9C3E-4CDF-8B49-68417D383363),
              version(1.0),
              helpstring("Interface dispatch pour Bonjour Objet"),
              dual
        ]
        interface IBonjour: IDispatch
        {
             [
                  id(0x000000C9)
             ]
             HRESULT _stdcall DirBonjour([in] BSTR Nom, [in] BSTR Prenom, [out, retval] BSTR * Value );
         };
         [
              uuid(7BE98DB1-9C3E-4CDF-8B49-68417D383363),
              version(1.0),
              helpstring("Bonjour Objet")
         ]
         coclass Bonjour
        {
              [default] interface IBonjour;
        };
};
```

List 4-1 : Définition de l'interface objet serveur DCOM

Etape 2 : générer les fichiers de type libraire *.TLB :

Les fichiers IDL sont compilés avec le programme MIDL, fourni dans le kit de développement pour Windows (fournit avec VC++). MIDL génère alors un fichier de type libraire.

Ce fichier est le type library contenant la description des interfaces et des composants définis dans le fichier IDL, qui servira à la fois pour le proxy sur la machine où tourne le client et pour le stub sur la machine où tourne le serveur

Exemple :

La syntaxe de MIDL est relativement simple : ***MIDL BonjourApp.idl***
Et le fichier généré est BonjourApp.TLB
MIDL génère d'autres fichiers pour le langage C++

Etape 3 : implémentation de l'objet DCOM

Il utilise l'utilitaire JactiveX pour générer le skeletons coté de serveur par l'instruction

Exemple : ***JactiveX Bonjour.tlb***

Vous pouvez voir deux autres fichiers générés

Bonjour.Java
Ibonjour.Java

On peut renommer ***Bonjour.Java*** à ***BonjourImpl.Java*** et implémente le code objet comme suit :

Exemple : BonjourImpl.java

```
package bonjourapp;
import com.ms.com.*;
import com.ms.com.IUnknown;
import com.ms.com.Variant;

/** @com.class(classid=92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC,DynamicCasts)
   @com.interface(iid=7BE98DB1-9C3E-4CDF-8B49-68417D383363, thread=AUTO, type=DUAL) */

public class BonjourImpl implements IUnknown,com.ms.com.NoAutoScripting,bonjourapp.IBonjour
{
      public  String DirBonjour(String Nom, String Prenom)
      {
            return "Bonjour : " + Nom +"  " + Prenom ;
       }
}
```

List 4-2: Implémentation de l'objet serveur DCOM

Et en fin, il faut compiler le fichier d'implémentation par la commande ci-dessous pour générer les classes ***BonjourImpl.class*** et ***IBonjour.class***

Exemple : ***JVC –d *.java***

Etape 4 : l'enregistrement de l'objet DCOM serveur

Il est nécessaire d'ajouter des entrées dans la base de registres pour que le système puisse les localiser à partir de leur CLSID, Pour cela on utilise un fichier de type *.reg:
Exemple : BonjourImpl. Reg

Ou bien par la commande :
javareg.exe/register/class :BonjourImpl
/clsid :{92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC }

```
REGEDIT4

[HKEY_CLASSES_ROOT\CLSID\{92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC}]
@="Java Class: BonjourImpl "
"AppID"="{92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC}"

[HKEY_CLASSES_ROOT\CLSID\{92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC}\InprocServer32]
@="MSJAVA.DLL"
"ThreadingModel"="Both"
"JavaClass"=" BonjourImpl "

[HKEY_CLASSES_ROOT\CLSID\{92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC}\LocalServer32]
@="javareg /clsid:{92 BBAA37-DB44-4865-B7DF-1FC10DC2C4CC} /surrogate"

[HKEY_CLASSES_ROOT\CLSID\{92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC}\Implemented
Categories]

[HKEY_CLASSES_ROOT\CLSID\{92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC}\Implemented
Categories\{BE0975F0-BBDD-11CF-97DF-00AA001F73C1}]

[HKEY_CLASSES_ROOT\AppID\{92BBAA37-DB44-4865-B7DF-1FC10DC2C4CC}]
@="Java Class: BonjourImpl "
```

List 4-3: l'enregistrement de l'objet serveur DCOM

Donc maintenant l'objet **BonjourImpl** est inscrit dans la base de registres et d'autres clients peuvent y accéder.

Etape 5 : Configuration de DCOM

Ce qui différence COM et DCOM est son outil de configuration, qui permet de régler comment et ou les serveurs distants sont crées en éditant les entrées de la base de registre associes.

Cet outil de configuration permet également de contrôler comment l'objet est sécurisé, les options sécuritaires de DCOM se résument à autoriser ou non l'accès à un composant selon le nom de l'utilisateur.

Nous utilisons l'utilitaire ***dcomcnfg*** dans la boite Exécuter (menu démarrer→Exécuter)

Etape 6 : Développement et lancement de l'application cliente.

Cette étape nous permet d'exploiter les objets java par le client distant,

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
package bonjourapp;
public class BonjourClient
{
   public static void main(String[] args) throws Exception
   {
        try
        {
            IBonjourApp Bonjour = (IBonjourApp)new BonjourApp ();
            String nom = "   xxxxx " , prenom = "yyyyyyyyyy";
            System.out.println(Bonjour.DirBonjour(nom , prenom ));
        }
       catch(Exception e)
        {
                    System.out.println("exeption"); throw e;
        }
   }
}
```

List 4-4 : développement de Client DCOM

IV-9- Conclusion

DCOM est une technologie de composants distribués, introduite par Microsoft, héritée de COM afin de supporter la communication inter objets sur des ordinateurs repartis sur réseaux. Donc il n' y a pas de différences fondamentales entre COM et DCOM, puisque ce dernier ne sert finalement qu'à communiquer les composants distants.

Dans ce chapitre, nous avons détaillé la technologie DCOM en présentant en premier lieu ses composants : les caractéristiques, les avantages et les services fournies. Ensuite, nous avons cité les différents protocoles utilisés et nous avons terminé par étudier les étapes de son développement et fonctionnement**.**

DCOM est un vrai concurrent de CORBA ou RMI dans un choix de réalisation d'une application distribuée.

PARTIE II
APPLICATIONS ET EVALUATIONS

Le but de cette partie est étudié la performance des middlewares basée sur des critères descriptifs et objectifs à optimiser le temps de calcul. Par ailleurs, nous comparons sur la base de ces critères nos résultats pour choisir un middleware satisfait nos besoins.

Le second but présenté dans cette partie est une application plus concrète. Elle concerne le problème de gérer d'une façon optimale les ressources en particulier celui de pièces de rechanges dans une entreprise étendu, pour trouver un équilibrage coopératif et préventif des ressources distribuées basé sur les concepts de e-maintenance et SMA.

Mot clé :

Middleware, SMA, IAD, réseaux, CORBA, RMI, DCOM, TIC, Maintenance, UML,

CHAPITRE V

ETUDE COMPARATIVE ET DE SYNTHESE

Le but de ce chapitre est de donner les clés qui permettront de définir la solution la mieux adaptée à chaque situation rencontrée dans une application distribuée, langage de programmation, plate-forme, fournisseur, ets

Ainsi, il donne une réponse à la question essentielle posé lors du lancement de tout nouveau de projet : faut- il utiliser CORBA, RMI DCOM ?

Mot clé :

Middleware, architecture distribue, système réparti, réseaux, CORBA, RMI, DCOM, Java

Chapitre V

Etude comparative et de synthèse

V-1- Introduction

L'objectif de cette partie du mémoire est de faire une comparaison entre les plates formes distribuées afin de sélectionner une plate-forme pour l'utiliser dans la réalisation d'un système multi agents pour le e_ maintenance.

Le choix d'une architecture est primordial dans la conception et la mise en œuvre d'une application distribuée, ce choix est la résultante de nombreux paramètres tel que :

- L'environnement d'exécution (nature des réseaux et des protocoles de communication, niveau de sécurité)
- Le mode de structuration d'application
- Interopérabilité avec les applications existantes.

Bien d'autres aspects peuvent également entrer en considération.

Afin de rendre le bon choix entre ces technologies, et suivant les paramètres et les critères choisis de manière objective, on peut diviser cette comparaison en deux parties:

1- Comparaison descriptive
2- Comparaison objective.

V-2- Comparaison descriptive

Le choix dans cette étape est désigné selon les besoins des développeurs, les capacités et les moyens qu'ils disposent, on peut citer le système d'exploitation, le coût, le langage maîtrisé préalable.

Donc le choix est simple, et désigné suivant la description de chaque méthode et la présence ou l'absence d'une contrainte peut éliminer ou avantager une méthode

Cette comparaison peut être illustrée comme suit.

V-2-1- le propriétaire

En matière de middleware objet existent trois standards,

- Le standard officiel CORBA (Common Object Request Broker Architecture) établi par l'OMG (Object Management Group), près de 700 sociétés sont actuellement membres de l'OMG, dont bien des sociétés informatiques comme HP, Sun et MicroSoft.
- Et le standard de fait COM/DCOM (Distributed Component Object Model) de Microsoft, elle a développe son propre standard d'objet répartis, malgré elle est un membre de l'OMG, le standard spécifie uniquement pour et seulement pour les produits Microsoft
- Et finalement JavaRMI (java Remote Method invocation) de Sun MicroSystem.

V-2-2- la Complexité

RMI parvient au moyen d'une version simplifiée de la spécification de CORBA. Ce dernier est une spécification complexe qui permet de construire des systèmes dans des langages différents tout en maintenant la communication entre eux comme s'il s'agissait d'un seul système, et ensuite la technologie de DCOM qui est une architecture très complexe dont l'apprentissage est difficile. Donc ils évident que Java/RMI est la solution la plus rapide et plus facile.

V-2-3- Le Langage de programmation

La puissance de CORBA est de permettre à une application Java d'appeler des fonctions C dans une application C à distance tout en utilisant la syntaxe de Java, cette puissance est aussi garantie par l'architecture DCOM, tel que le choix du langage est indépendant de l'architecture On peut considérer aussi RMI comme un CORBA 100% Java. Il simplifie le développement d'applications distribuées dans des systèmes Java homogènes. Comme il fait partie intégrante du langage, Donc il apparaît que la solution CORBA et DCOM étant les meilleures solutions.

V-2-4- Le Protocole de communication

DCOM ,RMI et CORBA sont des solutions majeures pour développer des applications réparties. Chacune de ces solutions définit ses propres fonctions telles que le protocole de communication:

- ORPC (*Object Remote Procedure Call*) pour DCOM .
- IIOP (*Internet Inter-ORB Protocol*) pour CORBA .
- RMI s'appuie sur le protocole JRMP(*Java Remote Method Protocol*)

V-2-5- Le Coût du déploiement

- RMI est gratuit, il est disponible pour télécharger sur le site de Sun, il fait partie intégrante du langage Java.
- DCOM est aussi gratuit, il faut considèrer l'achat de langage VC++ par exemple.
- Par contre, avec la solution CORBA, les développeurs ne dépendent pas d'un seul fournisseur unique. Et la version *VisiBroker* est la seule version gratuite puisqu'elle est intégrée dans les produits de Borland comme C++Builder par exemple.

V-2-6- Le Langage de définition d'interface

Contrairement à CORBA, DCOM spécifie les interfaces entre les composants au niveau binaire, et non au niveau source. Ceci signifie que la manière d'utiliser les interfaces des composants n'est pas imposée au niveau des langages de programmation (il n'y a pas de « projection » du langage de description des interfaces pour chaque langage de programmation).

Par contre CORBA nécessité toujours, la projection de l'interface IDL vers un langage de programmation quelconque, cette opération est appelée le Mapping

Et comme RMI est une solution distribuée Java à Java, l'interface est écrit en Java, donc pas nécessairement de le projeter vers d'autres langages.

V-2-7-Le Plates-formes supportées

- DCOM est le modèle distribue de composants MicroSoft, donc il est le plus utilisé sur la plate-forme WINDOWS principalement.

- Etant donné que CORBA n'est qu'une spécification, elle peut être portée sur diverses plates-formes, aussi longtemps que l'implémentation des ORB sur ces plates-formes existent.

- Java/RMI fonctionne sur toutes les plates formes implantant une machine virtuelle Java

Donc, il s'avère que la solution CORBA et Java/RMI sont les bons choix.

Concepts	CORBA	DCOM	RMI
Standard	**Oui**	**Non**	**Non**
Fournisseurs	**OMG**	**Microsoft**	**Sun**
Approche objet	**Oui**	**Oui**	**Oui**
Plates-formes supportées	**Toutes ou presque**	**Microsoft**	**Toutes ou presque**
Langages utilisables	**Tous ou presque**	**Tous ou presque**	**java**
Protocole de communication	**IIOP**	**RPC/DCE**	**JRMP**
Langage de définition d'interface	**IDL**	**MIDL**	**Java**
Coût du déploiement	**Payant & Gratuit**	**Gratuit**	**Gratuit**
Complexité	**Moyenne**	**Moyenne à élevée**	**Simple**
Annuaire d'objets	**service de noms**	**Non**	**rmiregistry**

Tableau 5-1 : Comparaison entre les middlewares

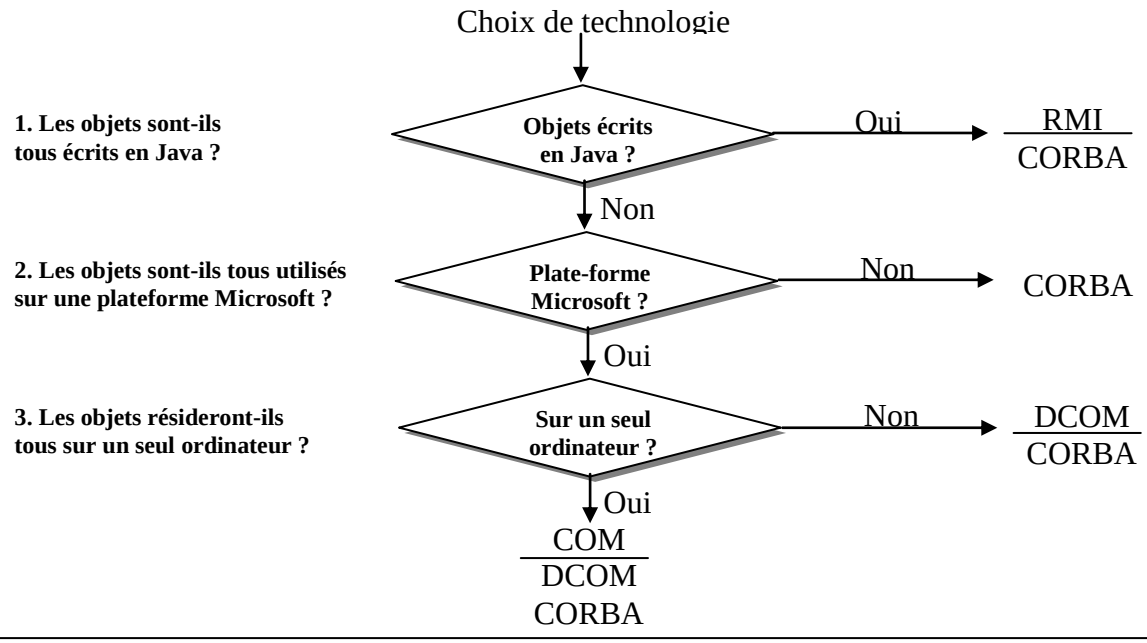


Figure 5-1 : Organigramme de choix d'un middleware

Le tableau de comparaison précédent est très utile lors du processus de prise de décision, on met en évidence les paramètres les plus Importants.

V-3- Comparaison objective

Dans cette section, les paramètres de comparaison intervenants sont égaux, tel que l'environnement d'exécution, le système d'exploitation, et le langage de programmation.

Pour choisir dans cette situation, des tests ont été effectués pour mesurer la performance des systèmes de communication choisie, en fonction de temps de transmission et le nombre d'erreurs effectuées, suivant des types des données qu'ils transmettent.

V-3-1- L'environnement d'exécution

Tous les tests de ce mémoire sont réalisés sur :

- Un ordinateur doté d'un processeur Intel **PVI** de fréquence 2.4 Ghz, et de 512 Mo de RAM comme un serveur.
- Un ordinateur doté d'un processeur Intel **PVI** de fréquence 2.4 Ghz, et de 512 Mo de RAM comme Client

Le but est d'avoir des résultats conformes à une plate-forme classique de modèle client/serveur

- Les deux ordinateurs sont reliés par un câble de réseau de type coaxial direct branché dans la carte réseau..

V-3-2- Le choix du système d'exploitation

Le choix du système d'exploitation pour faire la mise en œuvre des tests sur les trois méthodes distribuées est le Microsoft Windows Xp puisque DCOM est une spécification de Microsoft pour les systèmes Windows, et comme je compare DCOM avec d'autres méthodes donc il faut utiliser le système d'exploitation Windows.

V-3-3- Le choix du langage de programmation

Le langage Java a été sélectionné pour réaliser les tests de comparaison pour les trois raisons suivantes :

- Java RMI peut seulement être mise en œuvre employant le langage Java.
- Puisque je compare Java RMI avec d'autres technologies DCOM et CORBA, donc je dois mettre en œuvre le DCOM et les objets CORBA aussi dans le langage Java.
- Le langage Java est le meilleur langage orienté objet pour coder CORBA, et des objets COM, et cela tient la mise en œuvre simple et facile de comprendre, et plus élégant.

V-3-4- Le Principe de test

L'appel de procédure distante comprend de nombreuses étapes et il permet de mieux comprendre ce qui est effectivement mesurée lors d'un appel de méthode, De telles mesures donneraient une idée claire de ce qui est passe sous le « capot »de CORBA, RMI et DCOM

Le modèle de calcul du temps est décrit pour un client qui fait une demande à un serveur. Le client est le passage de paramètres d'entrée à l'appel et le serveur fait un travail avant de retourner un résultat au client.

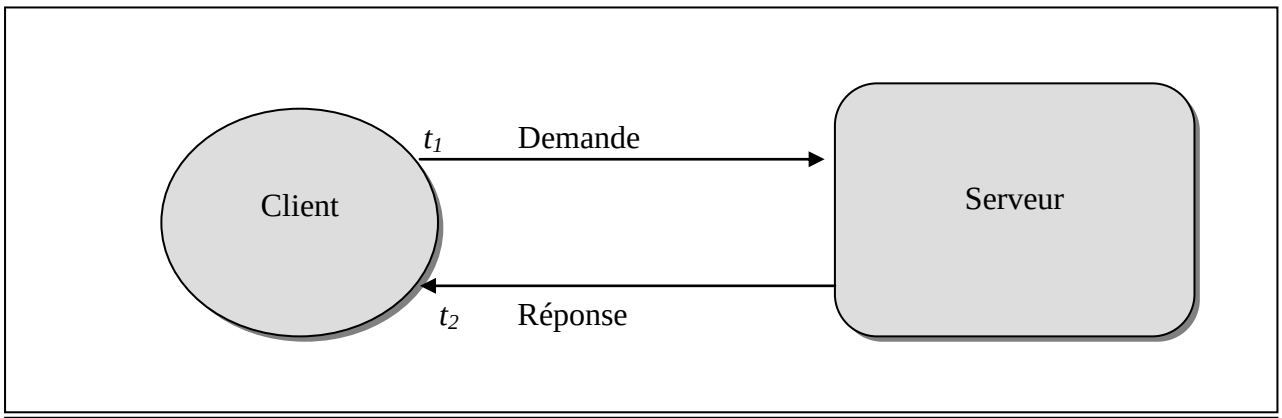


Figure 5-2: Un client fait une demande à un serveur

Suivant la figure :

- *t1* : désigne le temps juste avant que le client ne fasse la demande.
- *t2* : désigne le moment où le client a reçu une réponse à sa demande.

La différence t2 - t1 est le temps de réponse de la demande, qui est aussi appelé temps de l'aller-retour, cette différence est donnée par:

$$t_2 - t_1 = t_{\text{marshalling(demande)}} + t_{communication} + t_{\text{Unmarshalling(demande)}} + t_{execution} + t_{\text{marshalling(resultat)}} + t_{communication} + t_{\text{Unmarshalling(resultat)}}$$

- $t_{marshalling(demande)}$: Le temps nécessaire pour permettre le rassemblement et la traduction des paramètres d'entrée et de mettre le demande sur le réseau.
- $t_{communication}$: Le temps nécessaire sur le réseau pour que la demande Voyagé du client vers le serveur et vice versa
- $t_{UnMarshalling(demande)}$:Le temps nécessaire pour recevoir et faire la traduction inverse de la demande.
- $t_{exécution}$:Le temps d'exécution sur le serveur.
- $t_{Marshalling(résultat)}$:Le temps nécessaire pour que le serveur puisse rassembler les résultats et de sa mise sur le réseau.
- $t_{UnMarshalling(résultat)}$:Le temps nécessaire pour recevoir le résultat et le unmarshalling.

En fait la différence t2 - t1 est trop petite pour être mesurée sur une invocation. La solution à ce problème est de prendre le temps de plusieurs appels, ce qui signifie que l'horloge est démarrée avant la première invocation et il est arrêté après le $N^{ième}$ invocation

Chaque test à ses propres hypothèses concernant le modèle du temps en fonction et qui dépend :

- Des paramètres d'entrée. Le temps de marshalling / unmarshalling des paramètres est une grande partie de temps d'appel.
- Du temps d'exécution : qui dépend de la quantité de travail que le serveur a effectuée.
- De la valeur de retour : Le temps de marshalling / unmarshalling de résultat de retour qui dépend de savoir si quelque chose est retourné et le type de données du résultat.

Du point de vue du client, un temps aller-retour est mesuré lorsque la demande est transmise à l'agent et un résultat est retourné.

Il est également intéressant de mesurer le temps sur le côté serveur, mais que les horloges du client et du serveur ne sont pas synchronisés, il est impossible de comparer la date de transmission une demande sur le côté client avec la date de réception de cette demande sur le côté de serveur.

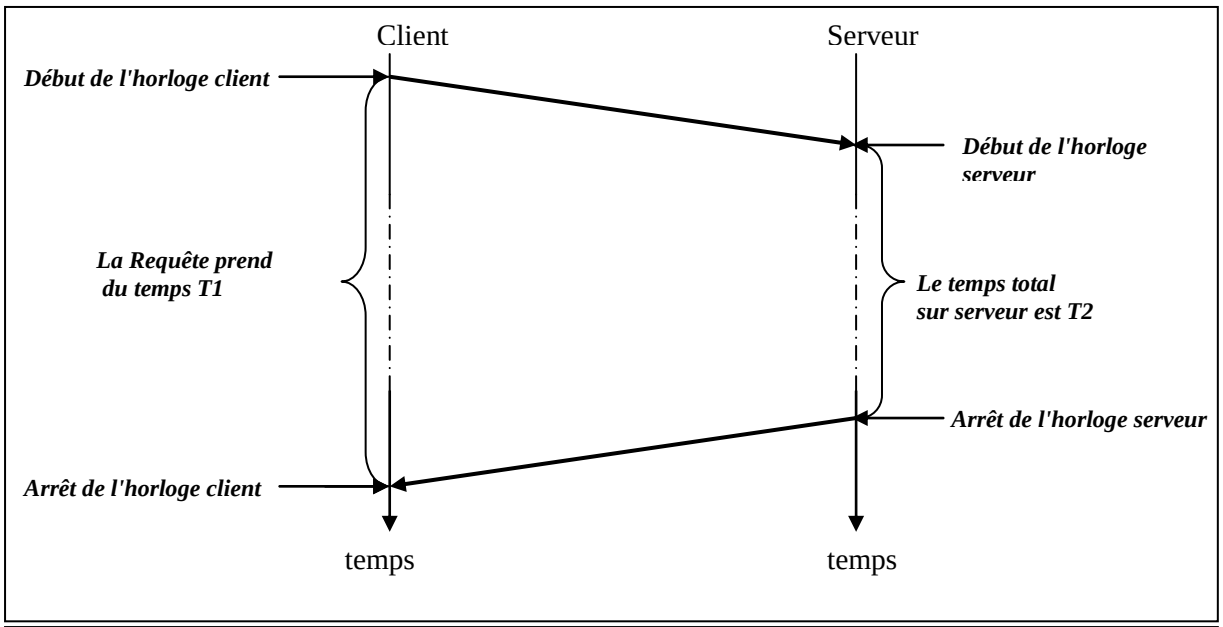


Figure 5-3: Le temps a passé sur le côté client et serveur lors d'une requête

D'après la figure précédente on déduit que :

$$T1-T2 = t_{\text{marshalling(demande)}} + t_{communication} + t_{\text{Unmarshalling(demande)}} + t_{\text{marshalling(resultat)}} + t_{communication} + t_{\text{Unmarshalling(resultat)}}$$

Le principe de test se base sur le principe d'une communication client/serveur, suivant un algorithme de test de performance, afin de mesurer le comportement des différentes middleware par rapport aux données qu'il transmet.

L'algorithme :

1- initialiser le client
2- initialiser la séquence des données à envoyer.
3- Pour i allant de 1 a nombre de tests faire
- T0 = prendre_heure() ;
- chercher serveur
- envoyer une requête de données au serveur
- attendre une réponse
- t1 = prendre_ heure() ;
- si la réponse est différente à la réponse souhaitable alors
Incriminante le nombre d'erreurs.
- sauvegarder le temps de transmission

fin pour
4- afficher la moyenne des temps.

Figure 5-4 : l'algorithme de test

Toutes les tests sont réalisés en appel distant, c'est à dire que le client et le serveur sont installés dans deux machines différentes, qui nous permet de calculer le temps nécessaire à la lecture et à l'écriture de message, le temps de calculer comprend le temps de transport nécessaire aux messages pour se déplacer d'une machine à l 'autre

Les tests sont exécutés plusieurs fois (100 fois) , et la moyenne de tous les temps est calculée et prenue en considération comme résultat final de test

Tous les tests ont été réalisés avec le même environnement et les mêmes données, afin d'obtenir des résultats comparables.

Plusieurs scénarios de test sont réalisés, pour exploiter les capacités d'appel des méthodes distantes, et extrait la performance et la réaction des différents middleware aux données qu'ils transmettent.

1. Transmission des entiers : le premier test effectué est une simple transmission, des séquences d'entiers sont envoyées au serveur, afin d'étudier la réaction des différents systèmes aux transmissions

2. Transmission des doubles : le deuxième test repose sur le même principe que le premier test, mais cette fois au lieu d'envoyées des entiers, on transmet des doubles.

3. Transmission des chaînes de caractères : le troisième test repose sur la transmission des chaînes de caractères au serveur, le test est effectue pour des tailles différentes de chaînes de caractères, ce qui nous permet d'étudier la réaction pour les grandes chaînes de caractères.

4. Transmission d'un tableau : le quatrirème test consiste à envoyer un tableau d'entiers, suivant le même principe que le test précédent, un tableau d'entiers est envoyé au serveur qui le renvoie au client, ce test permet d'étudier le comportement des différents systèmes face à des tableaux de différentes tailles.

5. L'addition : le dernier test est une simple addition de deux entiers envoyés au serveur qui renvoie au client la somme de ces deux entiers.

V-3-5- les implémentations des tests

Dans cette partie, on présente les implémentations utilisées pour exécuter chaque test de performance pour chaque middleware.

V-3-5-1-L'implémentation de CORBA

a- l'interface :

```
module TestCORBA
{
 typedef sequence <long> Tableau ;
 interface Bonjour
 {
    long     TestInteger(in long n )  ;
    long     TestAddition(in long n1 ,in long n2 )  ;
    string   TestChaine(in string ch ) ;
    double   TestDouble(in double x )   ;
    Tableau  TestTableau(in  Tableau b) ;
 };
};
```

List 5-1 : définition de l'interface CORBA de test de performance

b- implantation de l'interface :

```
Package TestCORBA;
Public class testCORBAImpl extends _ testCORBAImplBase
{
          public testCORBAImpl(java.lang.String name)    {  super(name);  }
          public double TestDouble(double x)  {    return x;    }
          public String TestChaine(String ch)   {    return ch ; }
          public int TestInteger(int n)   {     return n ;   }
          public int  TestAddition( int n1 , int n2)   {     return n1 + n2 ;   }
          public int[] TestTableau( int[] Tab )   {     return Tab ;   }
}
```

List 5-2 : l'implémentation de l'interface CORBA de test de performance

c– le serveur :

```
Package TestCORBA;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class  ServeurTestCORBA
{
   public static void main(String[] args)
  {
      try
      {
            org.omg.CORBA.ORB orb  = org.omg.CORBA.ORB.init(args,System.getProperties()) ;
            org.omg.CORBA.BOA boa  = ((com.visigenic.vbroker.orb.ORB)orb).BOA_init() ;
            TestCORBAimplObject = new TestCORBAImpl("monObjet ");
            boa.obj_is_ready(implObject);
            System.out.println(" le serveur est prêt ");
       }
       catch(Exception e)
      {
            e.printStackTrace() ;
       }
    }
 }
```

List 5-3 : le serveur CORBA de test de performance

d– le client :

```
Package TestCORBA;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class  ClientTestCORBA
{
   public static void main(String[] args)
  {
     try
     {
          org.omg.CORBA.ORB orb ;
          Bonjour serveur ;
          orb  = org.omg.CORBA.ORB.init(args,System.getProperties()) ;
          serveur = TestCORBAHelper.bind(orb," monObjet ");
          rep0 = (String)JOptionPane.showInputDialog(null,"le nombre d'element:" ,"TestInteger",
                    JOptionPane.QUESTION_MESSAGE);
                       // Déclaration de deux tableaux des entiers
          int []TableauDesEntiersEnvoyes,TableauDesEntiersRecus ;
                       // La création de deux tableaux
          n = Integer.parseInt(rep0) ;
          TableauDesEntiersEnvoyes = new int[n] ;
          TableauDesEntiersRecus   = new int[n] ;
                       // Remplir le tableau des entiers à envoyer
          for (int i = 0;i<n;i++)
                    TableauDesEntiersEnvoyes[i] = i ;
          int nbre_erreur = 0 ;// calculer le nombre des erreurs
          int nbre_test = 100 ;
                       // initialiser le temps
          float temps_moyen = 0 ;
          for (int j =1 ; j<=100 ;j++)
          {
                       int i = 0 ; // compteur
                       long start = System.currentTimeMillis() ;
                       do
                       {
                                 // appel la méthode TestInteger
                            TableauDesEntiersRecus[i] = serveur.TestInteger (TableauDesEntiersEnvoyes[i]);
                             i++ ; // incrémentation de compteur
                        }
                        while ( i!=n )   ;
                         //arreter le calcul de temps
                        long end = System.currentTimeMillis() ;
                        for(i=0 ;i<n;i++)
                        {
                             if (TableauDesEntiersEnvoyes[i] != TableauDesEntiersRecus[i] )
                                  nbre_erreur ++ ;
                        }
                        temps_moyen = temps_moyen + (end - start ) ;
          }
          System.out.println("Le test est termine,le nombre d'erreurs est :"+nbr_erreur);
          System.out.println("Le temps pour effectuer le test : " + (temps_moyen/ nbre_test) );
     }
     catch(Exception e)
     {
       e.printStackTrace() ;
      }
    }
}
```

List 5-4 : le client CORBA de test de performance (cas d'envoi des entiers).

V-3-5-2-L'implémentation de RMI

a- l'interface :

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface TestRMI extends Remote
{
  public int TestInteger(int n ) throws RemoteException ;
  public int TestAddition(int n1 , int n2 ) throws RemoteException ;
  public String TestChaine(String ch ) throws RemoteException ;
  public int[] TestTableau(int tableau[] ) throws RemoteException ;
  public double TestDouble(double x ) throws RemoteException ;
}
```

List 5-5 : définition de l'interface RMI de test de performance

b- implantation de l'interface :

```
import java.rmi.*;
import java.rmi.server.*;
public class TestRMIImpl extends UnicastRemoteObject implements TestRMI
{
 public int TestInteger(int n) throws RemoteException  {  return n;  }
 public int TestAddition(int n1, int n2) throws RemoteException  {  return n1+n2;  }
 public String TestChaine(String ch) throws RemoteException  {  return ch;  }
 public double TestDouble(double x) throws RemoteException  {  return x;  }
 public int[] TestTableau(int tableau[] ) throws RemoteException  {  return tableau  ;  }
 public TestRMIImpl() throws RemoteException  {  super();  }
}
```

List 5-6 : l'implémentation de l'interface RMI de test de performance

c- le serveur :

```
import java.rmi.*;
import java.rmi.server.*;
public class TestRMIServeur
{
    public static void main(String[] args) throws Exception
    {
       try
       {
          java.rmi.registry.LocateRegistry.createRegistry(1099)  ;
          TestRMI serveur = new TestRMIImpl();
          Naming.rebind("monObjet", serveur);
          System.out.println(" Le serveur est prêt ");
       }
       catch(Exception e)
       {
          throw e;
       }
   }
}
```

List 5-7 : le serveur RMI de test de performance

d- le client :

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.rmi.*;
import java.io.*;
public class TestRMIClient
{
   public static void main(String[] args) throws Exception
   {
      TestRMI   serveur = null;
      String nomComplet ="rmi://poste01:1099/" + "monObjet";
      serveur = (TestRMI)Naming.lookup(nomComplet);
      try
      {
            rep0 = (String)JOptionPane.showInputDialog(null,"le nombre d'element:" "TestInteger",
                  JOptionPane.QUESTION_MESSAGE);
                   // Déclaration de deux tableaux des entiers
            int []TableauDesEntiersEnvoyes,TableauDesEntiersRecus ;
                   // La création de deux tableaux
            n = Integer.parseInt(rep0) ;
            TableauDesEntiersEnvoyes = new int[n] ;
            TableauDesEntiersRecus   = new int[n] ;
                   // Remplir le tableau des entiers à envoyer
             for (int i = 0;i<n;i++)
                   TableauDesEntiersEnvoyes[i] = i ;
             int nbre_erreur = 0 ;// calculer le nombre des erreurs
             int nbre_test = 100 ;
                   // initialise le temps
             float temps_moyen = 0 ;
             for (int j =1 ; j<= nbre_test;j++)
             {
                   int i = 0 ; // compteur
                   long  start = System.currentTimeMillis() ;
                   do
                   {
                         // appel la méthode TestInteger
                        TableauDesEntiersRecus[i] = serveur.TestInteger(TableauDesEntiersEnvoyes[i]);
                        i++ ; // incrémentation de compteur
                    }
                    while ( i!=n )   ;
                          //arreter le calcule de temps
                    long    end = System.currentTimeMillis() ;
                    for(i=0 ;i<n;i++)
                    {
                          if (TableauDesEntiersEnvoyes[i] != TableauDesEntiersRecus[i] )
                               nbre_erreur ++ ;
                    }
                   temps_moyen = temps_moyen + (end - start ) ;
              }
             System.out.println("Le test est termine,le nombre d'erreurs est :"+nbre_erreur);
             System.out.println("Le temps pour effectuer le test : " + (temps_moyen/ nbre_test) );
       }
       catch(Exception e)
       {
            System.out.println("exeption"); throw e;
       }
    }
}
```

List 5-8 : le client RMI de test de performance (cas d'envoi des entiers).

V-3-5-3-L'implémentation de DCOM

a- l'interface :

```
[
  uuid(2DE37B26-F4AB-4004-90B8-0515A6C6BABC),
  version(1.0),
  helpstring("ServeurDCOM (bibliothèque)")
]
library ServeurDCOM
{
  importlib("stdole2.tlb");
  [
    uuid(ADBC91D6-37E3-4B04-9411-E4769ECEC228),
    version(1.0),
    helpstring("ServeurTestDCOM Objet")
  ]
  coclass ServeurTestDCOM
  {
    [default] interface IServeurTestDCOM;
  };
  [
    uuid(FE53186C-F7A1-4FF1-ABD1-41D26F880FCC),
    version(1.0),
    helpstring("Interface dispatch pour ServeurTestDCOM Objet"),
    dual,
    oleautomation
  ]
   interface IServeurTestDCOM: IDispatch
  {
    [
    id(0x000000C9)
    ]
    HRESULT _stdcall TestInteger([in] long a, [out, retval] long * b );
    [
    id(0x000000CA)
    ]
    HRESULT _stdcall TestAddition([in] long a, [in] long b, [out, retval] long * c );
    [
    id(0x000000CB)
    ]
    HRESULT _stdcall TestChaine([in] BSTR ch1, [out, retval] BSTR * ch2 );
    [
    id(0x000000CD)
    ]
    HRESULT _stdcall TestDouble([in] double a, [out, retval] double * b );
    [
    id(0x000000CC)
    ]
    HRESULT _stdcall TestTableau([in] long a, [in] VARIANT b, [out, retval] VARIANT * c );
  };
};
```

List 5-9 : définition de l'interface DCOM de test de performance

b- implantation de l'interface :

```
package serveurdcom;
import com.ms.com.*;
import com.ms.com.IUnknown;
import com.ms.com.Variant;

/** @com.class(classid=ADBC91D6-37E3-4B04-9411-E4769ECEC228,DynamicCasts)
   @com.interface(iid=FE53186C-F7A1-4FF1-ABD1-41D26F880FCC, thread=AUTO, type=DUAL)
*/
public class ServeurTestDCOM implements
IUnknown,com.ms.com.NoAutoScripting,serveurdcom.IServeurTestDCOM
{

  /** @com.method(vtoffset=4, dispid=201, type=METHOD, name="TestInteger", addFlagsVtable=4)
     @com.parameters([in,type=I4] a, [type=I4] return) */
  public int TestInteger(int a)
  {
       return a ;
  }

  /** @com.method(vtoffset=5, dispid=202, type=METHOD, name="TestAddition", addFlagsVtable=4)
     @com.parameters([in,type=I4] a, [in,type=I4] b, [type=I4] return) */
  public int TestAddition(int a, int b)
  {
     return a + b;
  }

  /** @com.method(vtoffset=6, dispid=203, type=METHOD, name="TestChaine", addFlagsVtable=4)
     @com.parameters([in,type=STRING] ch1, [type=STRING] return) */
  public String TestChaine(String ch1)
  {
     retrun ch;
  }

  /** @com.method(vtoffset=7, dispid=205, type=METHOD, name="TestDouble", addFlagsVtable=4)
     @com.parameters([in,type=R8] a, [type=R8] return) */
  public double TestDouble(double a)
  {
     return a;
  }

  /** @com.method(vtoffset=9, dispid=204, type=METHOD, name="TestTableau", addFlagsVtable=4)
     @com.parameters([in,type=I4] a, [in,type=VARIANT] b, [type=VARIANT] return) */
  public Variant TestTableau(int a, Variant b)
  {
       Variant  v = vararraycreate([0,a-1],varInteger);
       for(int  i = 0 ; i< a ,i++ )
            v[i] = b[i];
       return v ;
  }
}
```

List 5-10 : l'implémentation de l'interface DCOM de test de performance

d– le client :

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import serveurdcom.*;
public class ClientTestDCOM
{
   public static void main(String[] args) throws Exception
   {
       IServeurTestDCOM serveur = (IServeurTestDCOM)new ServeurTestDCOM();
       BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
       try
       {
                   System.out.println("      Entrer le nombre d'éléments pour tester les entiers  : ");
                   String rep0 = stdIn.readLine() ;
                               // Déclaration de deux tableaux des entiers
                   int []TableauDesEntierEnvoyes,TableauDesEntiersRecus ;
                              // La création de deux tableaux
                   n = Integer.parseInt(rep0) ;
                   TableauDesEntiersEnvoyes = new int[n] ;
                   TableauDesEntiersRecus   = new int[n] ;
                       // Remplir le tableau des entier a envoyé
                   for (int i = 0;i<n;i++)
                      TableauDesEntiersEnvoyes[i] = i ;
                   int nbr_erreur = 0 ;// calculer le nombre des erreurs
                       // initialise le temps
                   float temps_moyen = 0 ;

                   for (int j =1 ; j<=100 ;j++)
                   {
                            int i = 0 ; // compteur
                            long   start = System.currentTimeMillis() ;
                            do
                            {
                                     // appel la méthode TestInteger
                                TableauDesEntiersRecus[i] = serveur.TestInteger(TableauDesEntiersEnvoyes[i]);
                                  i++ ; // incrémentation de compteur
                            }
                            while ( i!=n )        ;
                                     //arreter le calcul du temps
                           long   end = System.currentTimeMillis() ;
                           for(i=0 ;i<n;i++)
                           {
                              if (TableauDesEntiersEnvoyes[i] != TableauDesEntiersRecus[i] )
                                    nbr_erreur ++ ;
                            }
                            temps_moyen = temps_moyen + (end - start ) ;
                   }
                   System.out.println("Le test est termine,le nombre d'erreurs est :"+nbr_erreur);
                   System.out.println("Le temps pour effectuer le test : " + (temps_moyen/100) );
          }
          Catch(Exception e)
          {
              System.out.println("exeption"); throw e;
          }
}
```

List 5-11 : le client DCOM de test de performance (cas d'envoi des entiers)

V-3-6- Les résultats

Ces résultats sont obtenus dans des conditions égales pour tous les systèmes de transmissions. C'est-à-dire le même environnement matériel, système d'exploitation, langage de programmation, et les mêmes tests sont effectues

V-3-6-1- Transmission des entiers

Ce test est réalisé en faisant varier le nombre d'éléments transférées de 10000 à 40000 éléments de type entier, Ce tableau exprime les temps moyens d'exécution obtenus après que le test fut effectué.

Mécanisme de transmission	CORBA	RMI	DCOM
10000 entiers	3442.00	4037.40	4461.55
20000 entiers	7000.93	7971.80	8896.81
30000 entiers	10523.11	11946.80	13426.27
40000 entiers	13740.46	16059.20	18011.91

Tableau 5-2 : les résultats de transmission des entiers

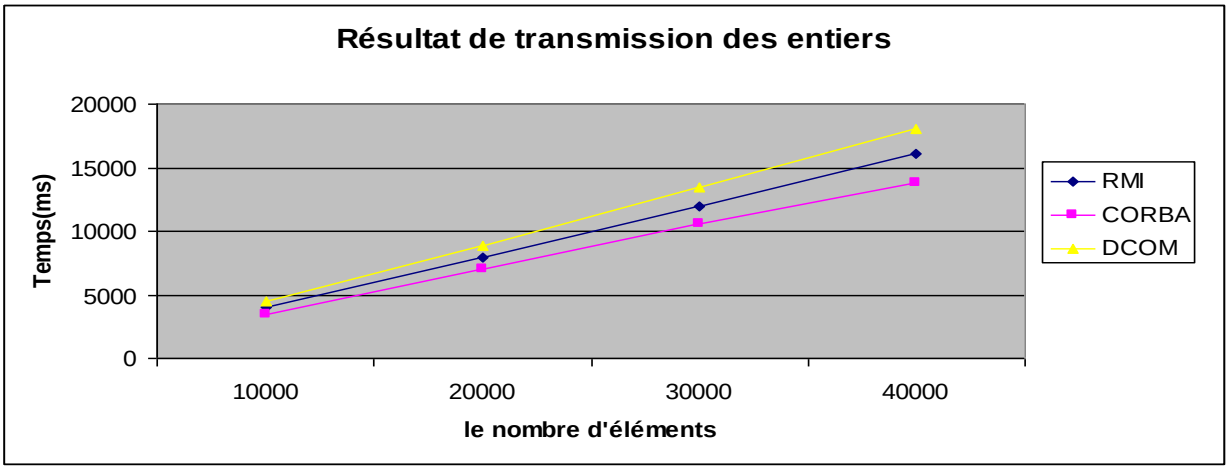


Figure 5-5 : Graphe des résultats de transmission des entiers

On peut remarquer, que la différence entre les trois middlewares est plus en plus être plus large tant que le nombre d'éléments transféré augmente, et on peut remarquer que les résultats obtenus par CORBA sont plus rapides et performants que celles de RMI, et celui-ci a obtenu des résultats plus optimales et rapides que ceux de DCOM.

V-3-6-2- Transmission des doubles

En cours, Ce teste est réalisé en faisant varier le nombre d'éléments transférés de 10000 à 40000 éléments de type double, donc on a changé la taille des éléments transférés pour avoir le comportement de chaque middleware, le tableau ci dessous exprime les temps moyens d'exécution obtenus après que le test fut effectué.

Mécanisme de transmission	CORBA	RMI	DCOM
10000 doubles	3559.83	4228.10	4462.32
20000 doubles	7235.17	8421.90	8989.42
30000 doubles	10659.39	12728.00	13540.95
40000 doubles	14320.33	16634.40	19416.25

Tableau 5-3 : les résultats de transmission des doubles

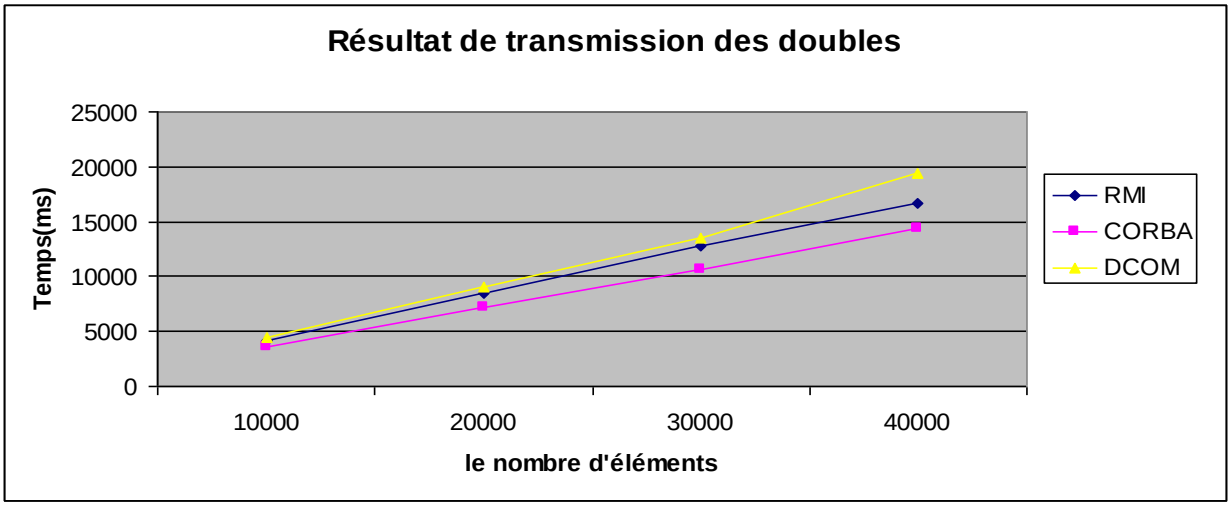


Figure 5-6 : Graphe des résultats de transmission des doubles

On peut remarquer que presque les mêmes résultats obtenus dans le test précédent sont obtenu dans ce test, mais on peut déduire, à partir des deux tests que le temps de transfert n'est pas forcement lié à la taille des éléments transférés mais au nombre des requêtes envoyées.

V-3-6-3- Transmission de chaînes de caractères

Ce test est réalisé pour avoir la réaction de chaque méthode suivant la taille des éléments envoyés, les temps moyens obtenus sont exprimés dans le tableau qui ce suit :

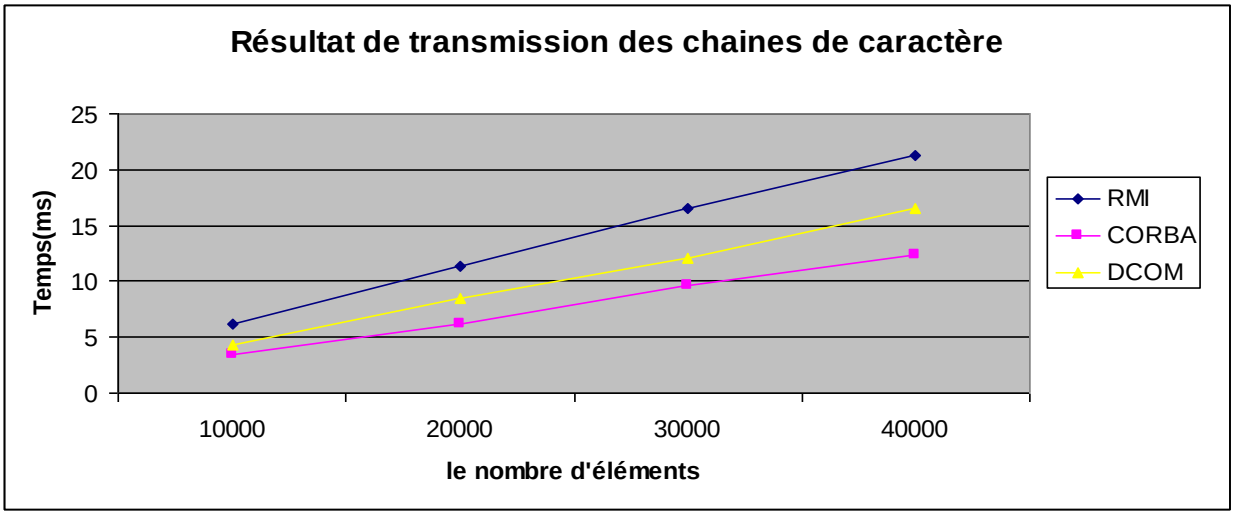


Figure 5-7 : Graphe des résultats de transmission des chaînes de caractère

Mécanisme de transmission	CORBA	RMI	DCOM
10000 caractères	3.44	6.25	4.37
20000 caractères	6.25	11.41	8.44
30000 caractères	9.56	16.53	12.04
40000 caractères	12.35	21.26	16.56

Tableau 5-4 : les résultats de transmission des chaînes de caractère

Ce test nous montre que le middleware CORBA est le meilleur toujours avec un temps de transfert minimum, et on trouve en suite DCOM, et en fin RMI, mais cette fois le temps du test dépend de la taille de message envoyés.

V-3-6-4- Transmission de tableau

Ce tableau donne les résultats de l'ensemble des tests, pour un tableau de taille varient de 10000 éléments jusqu'à 40000 éléments de type entier.

Mécanisme de transmission	CORBA	RMI	DCOM
10000 éléments	14.54	8.77	8.44
20000 éléments	27.94	17.33	16.56
30000 éléments	40.98	23.42	23.75
40000 éléments	52.35	30.47	31.57

Tableau 5-5 : les résultats de transmission des tableaux d'entiers

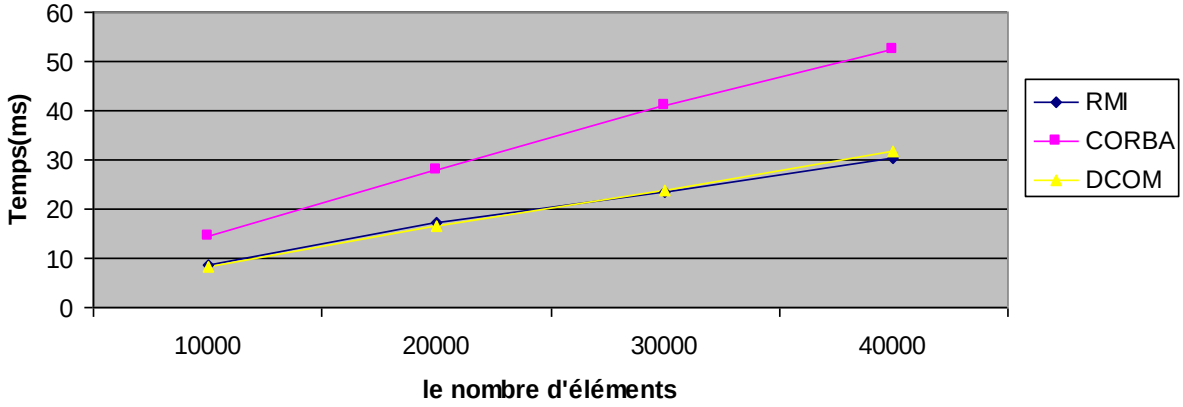


Figure 5-8 : Graphe des résultats de transmission des tableaux d'entiers

La situation dans ce teste est inversée, DCOM et RMI ont presque les mêmes résultats, et sont plus rapides que CORBA, et on peut remarquer que le temps de codage et de décodage d'un tableau est important puisqu'il influe sur le temps globale de transmission

V-3-6-5- Transmission addition

Ce tableau exprime les temps moyens d'exécution obtenus après le test addition de deux nombres de type entier, cela nous permet d'étudier le comportement des méthodes de transfert selon le nombre de paramètres envoyés.

Mécanisme de transmission	CORBA	RMI	DCOM
10000 fois	3458.76	4025.47	4461.55
20000 fois	7106.41	8050.58	8896.81
30000 fois	10688.78	12025.15	13426.27
40000 fois	13783.13	15890.80	18011.91

Tableau 5-6 : les résultats d'exécution d'une addition distant

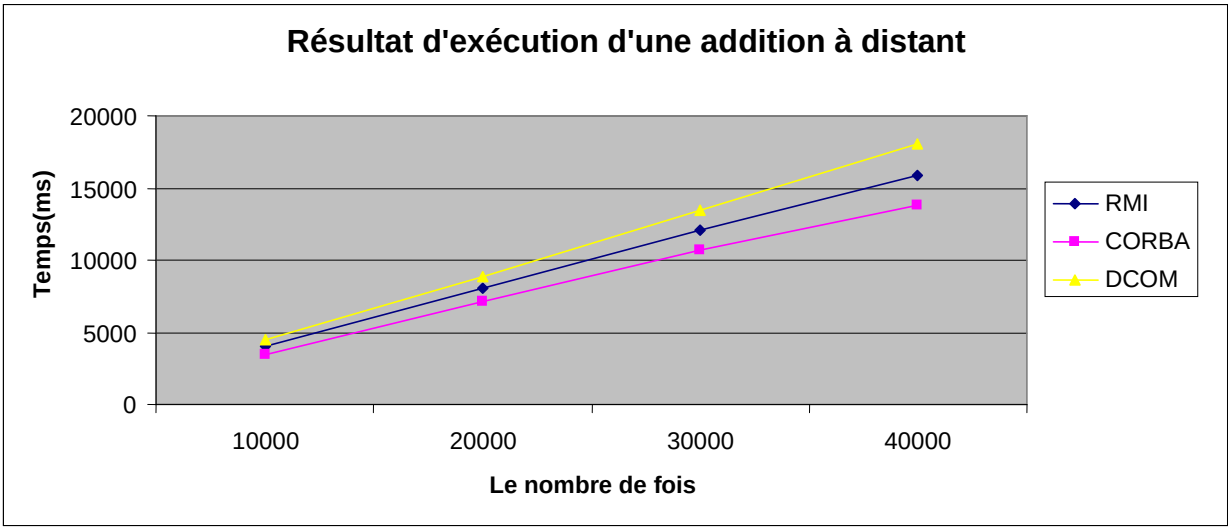


Figure 5-9 : Graphe des résultats d'exécution d'une addition distant

Pour ce dernier test comme pour les premiers tests, CORBA est toujours plus rapide, suivi de RMI qui est moins rapide, et pour finir les résultats, on trouve que le dernier DCOM est le plus lent, on peut remarquer que les résultats obtenus sont presque égaux, et on peut déduire que le temps de transmission dépend du codage et décodage des paramètres envoyés et la stratégie et le mécanisme utilisés par chaque middleware pour réaliser ces tests.

V-4- Conclusion et perspective

Les architectures de CORBA, DCOM et Java / RMI prévoient des mécanismes transparents pour l'invocation et l'accès aux objets distants distribués. Malgré les mécanismes qu'ils utilisent pour réaliser, les appels distants sont différents, les approches de chacun d'entre eux sont plus ou moins similaires.

L'idée force que nous souhaitons voir se dégager de ce chapitre est à la fois la grande variété des middlewares disponibles et surtout la grande diversité des fonctionnalités qu'ils peuvent fournir. Il faut donc bien comprendre que le choix d'un middleware, ou d'un type de middleware, doit se faire à partir des besoins exprimés.

Au début de cette partie de test, j'ai l'impression que DCOM peut être le bon choix, puisque les tests sont réalisés dans un environnement complètement Microsoft, ce qui peut aider DCOM à réaliser ces tests plus rapide que les autres méthodes.

Mais comme DCOM est une extension de COM de sorte que les anciennes applications continuent de fonctionner, empêche cette méthode d'être la meilleure.

Les tests sont réalisés pour nous permettre de choisir entre les trois middlewares, pour déterminer celui qui est le bon choix, et les tests sont valables pour comparer entre deux middlewares seulement, si le troisième est éliminé.

Ces tests sont réalisés avec une manière plus objective possible, en essayant d'utiliser tous les trucs et les astuces pour avoir un code source optimal, et que les instructions nécessaire sont adoptées

Après la réalisation de ces tests, on peut favoriser l'utilisation de CORBA comme un moyen de communication entre client et serveur à distant, et défavoriser les deux autres middleware, puisque le middleware CORBA est le plus rapide, et offre des possibilités très larges comme l'indépendance du langage de programmation, et du système d'exploitation.

CHAPITRE VI

E-MAINTENANCE ET SMA

Nous consacrons ce chapitre pour les descriptions des notions de base de SMA, nous exposerons les principaux aspects de cette approche, et nous démontrerons l'utilité de cette approche dans un équilibre coopératif de charger dans une entreprise étendue.

Afin de l'appliquer aux cas de gestion des stocks, et à l'approvisionnement des pièces de rechange

Mot clé :

SMA, Agents, Entreprise étendu, protocole, interaction, équilibre de charge, pièce de rechange, disponibilité, maintenance préventive,E-Maintenance

Chapitre VI

La E-Maintenance et SMA

VI-1- Introduction :

La maintenance joue un rôle essentiel dans la vie des entreprises, elle est obligatoire pour assurer le bon fonctionnement des machines industrielles, plusieurs solutions et méthodes sont inventées pour satisfaire cette obligation, les recherches sont appuyées sur les techniques d'intelligence artificielle pour concevoir un système plus souples et plus interactifs s'adapte mieux à l'utilisateur pour aider à maintenir en bon état du matériels, et avec l'évolution de la technologie, qui conduit à l'apparition des grands réseaux d'information, les travaux actuels s'orient vers des Environnements Informatiques artificiels distribués,et à la naissance de e_maintenance dans les entreprises étendu.

Les besoins de communication et de traitement de l'information dans ces entreprises sont de plus en plus croissants. Ils exigent la conception et le développement de systèmes complexes et intelligents. La variété des services requis, la diversité et la répartition des ressources physiques et de connaissances, ainsi que les changements dynamiques des besoins des utilisateurs, requièrent une approche nouvelle, capable de contenir toutes ces contraintes, pour résoudre ou aborder les problèmes complexes. Les systèmes d'information sont actuellement de plus en plus complexes, répartis sur un ou plusieurs sites. Ces derniers sont en interaction entre eux, ou avec des êtres humains, via des réseaux dédiés ou le réseau Internet.

Les critères de distributivité et d'interaction entre les partenaires de ce genre d'entreprises, rendent leur modélisation difficile par les outils classiques, Les Systèmes Multi-Agents (SMA) qui sont nés du besoin à répondre à cette modélisation afin de simuler et même implémenter ce type de problèmes distribués et interactifs; se veulent un substitut à ces outils.

Le but de cette partie est bien exploiter les technologies de l'informations et de communications, pour gérer d'une façon optimale les ressources en particulier celui de pièces de rechanges dans une entreprise étendu, pour trouver un équilibrage coopératif et préventif des ressources distribuées.

VI.2- Concepts de base

V1.2-1. L'entreprise étendue

L'entreprise étendue [38] est désormais une réalité. C'est celle qui s'étend au delà des frontières organisationnelles " physiques ", elle est caractérisée par une externalisation d'activités et par le développement de partenariats, par le biais des Technologies de l'Information et de la Communication (TIC) qui en sont le support.

Une **entreprise étendue** [39] dite aussi « **en réseau** », est un ensemble d'entreprises et d'acteurs économiques associés pour la réalisation de projets communs. Elle fonctionne essentiellement sur la base d'Alliances et Partenariats

Une entreprise étendue comprend en général [40] :

- une entreprise pilote maître d'œuvre / tête de réseau / chef de projet
- qui travaille en étroite coopération, tant au niveau de la conception que de la fabrication et de la distribution, avec de nombreuses entreprises partenaires, permanentes ou occasionnelles :

V1.2-2. Les N.T.I.C.

Les NTIC [41] (**N**ouvelles **T**echnologies de **l'**Information et de la **C**ommunication) sont aujourd'hui un outil majeur pour la construction de la compétitivité. Les entreprises qui gagnent sont celles qui savent établir des coopérations, travailler en réseau, produire et utiliser collectivement la connaissance sans cesse renouvelée dont elles ont besoin pour générer de la valeur.

Les NTIC (que l'on pourrait traduire par Nouvelles Technologies de l'Intelligence Collective) sont un atout irremplaçable dans la circulation rapide de l'information, l'élaboration collective de plans d'action et de nouvelles façons de faire, la coordination de l'action, la mémorisation et la capitalisation des expériences, l'accès rapide à des connaissances très diverses, l'ouverture de nouveaux services à la clientèle.
Cette contribution des NTIC à la création de valeur prend aujourd'hui plusieurs formes : intranet, internet, messageries, forums, groupware, workflow, bases de connaissances partagées, data mining, gestion documentaire, édition multimedia, commerce électronique, service à la clientèle, assistance à la formation, etc.

Ces formes vont encore évoluer, profondément et rapidement. Nous ne sommes qu'au début de la révolution informationnelle. Notre vocation est d'aider les entreprises à conduire cette mutation.

Les NTIC prennent leur sens dans une stratégie d'ensemble fondée sur l'intelligence partagée. Elles n'ont pas de but en soi, mais peuvent rendre possibles des modes d'organisation totalement nouveaux fondés sur l'innovation, la compétence collective, le partage et la capitalisation du savoir, la responsabilisation, la réactivité.

La connaissance est une construction collective, pas seulement une information à stocker. Le management de la connaissance est avant tout une dynamique d'invention et de partage. L'usage des NTIC doit être conçu pour provoquer, faciliter et démultiplier cette dynamique de création.

Les NTIC concernent toutes les activités de l'entreprise ; elles peuvent incorporer de l'intelligence tout au long de la chaîne de valeur, dans les métiers, dans les processus, dans les projets ; elles peuvent irriguer tous les réseaux de communication et toutes les catégories de personnes qui font l'entreprise : clients, collaborateurs, fournisseurs.
Les usages des NTIC doivent être élaborés en étroite concertation avec les utilisateurs, car ils modifient profondément les façons de travailler. La technologie ne doit pas prendre le pas sur les personnes, mais au contraire les aider à trouver les voies de la performance.

L'utilisation performante des NTIC nécessite l'acquisition de nouvelles compétences. Il est indispensable d'accompagner toute mise en place de NTIC par un programme volontariste de développement des compétences axé sur des objectifs opérationnels identifiés, afin d'utiliser pleinement les possibilités du système

V1.2-3. La gestion de ressources

Les stratégies d'externalisation consistent pour les entreprises à se procurer auprès des fournisseurs des services ou produits qui étaient auparavant assurés localement. Ces stratégies conduisent ces entreprises à s'associer avec d'autres entreprises dans une logique de complémentarité de ressources.

La gestion des ressources dans une entreprise étendue est à la fois:

_ Une gestion des stocks avec l'établissement de prévisions, des approvisionnements, une gestion de "un ou plusieurs magasins, appartenant à un ou plusieurs sites", des transports, gestion de nomenclatures, donc un ensemble d'activités logistiques classiques, où les règles et les algorithmes de la gestion de stock classiques peuvent être appliqués.

_Une activité technique de mise en oeuvre d'une politique de maintenance, de e_maintenance, particulièrement préventive, d'un ensemble industriel, composé d'un site ou d'un ensemble de sites interconnectés entre eux par le biais d'une plateforme de e-maintenance, de e-commerce, où l'équilibrage de la disponibilité des ressources entre ces différents sites constitue la condition nécessaire pour le bon fonctionnement de l'ensemble.

_Une politique préventive entre les différentes entités de l'entreprise qui sont directement concernées par la gestion de leurs ressources d'une façon "coopérative", en assurant un équilibrage en cette matière pouvant éviter le sous-stockage ou le sur-stockage.

VI-3- Définition de maintenance:

Selon la définition de l'AFNOR 2001, « la maintenance vise à maintenir ou à rétablir un bien dans un état spécifié afin que celui-ci soit en mesure d'assurer un service déterminé ».

Dans une entreprise, maintenir c'est donc effectuer des opérations (dépannage, réparation, réglage, révision, contrôle et de vérification) qui permettent de conserver le potentiel du matériels pour assurer la production avec efficacité et qualité

La e-Maintenance est un mode de supervision et de gestion de la maintenance à distance et en temps réel s'appuyant sur les Technologies de l'Information et des Communications (TIC).

Maintenir, c'est intervenir dans les meilleures conditions ou appliquer les différentes méthodes afin d'optimiser le coût global de possession.

On distingue deux formes de maintenance classée en fonction d'événement prévu et de l'état matériel

1- **La maintenance corrective**, qui consiste à intervenir sur un équipement une fois que celui-ci est défaillant ou dégradé de sa fonction, pour lui permettre d'accomplir une fonction requise. au moins provisoirement. Ses actions s'effectuent par étapes, dans 1'ordre suivant **[42]**:

- **test** : comparaison des mesures avec une référence,
- **détection** : déceler 1'apparition d'une défaillance,
- **localisation** : les éléments par lesquels la défaillance se manifeste,
- **diagnostic** : analyse des causes de la défaillance,
- **dépannage et réparation** : remise en etat (avec ou sans modification),
- **contrôle** : contrôle du bon fonctionnement,

- **amélioration éventuelle** : éviter la réapparition de la panne,
- **historique :** mise en mémoire de 1'intervention pour une exploitation ultérieure.

2- **La maintenance préventive**, qui consiste à intervenir sur un équipement avant que celui-ci ne soit défaillant, afin de tenter de prévenir la panne, et à exécuter à des intervalles prédéterminés ou selon des critères prescrits et destinée à réduire la probabilité de défaillance ou la dégradation du fonctionnement d'un bien

Cette maintenance se dissocie en [43] :

- **La maintenance systématique** : exécutée a des intervalles de temps préétablis ou selon un nombre défini d'unités d'usage indépendamment de 1'etat du bien. Elle repose sur le calcul du MTBF (Mean Time To Failure, Temps Moyen de Bon Fonctionnement).

- La maintenance conditionnelle : basée sur une surveillance du fonctionnement du bien et/ou des paramètres significatifs de ce fonctionnement intégrant les actions qui en découlent. L'intérêt d'une telle stratégique est de pouvoir utiliser les machines au maximum possible en diminuant le temps d'arrêt du aux opérations de maintenance corrective systématique.

- La maintenance prévisionnelle: exécutée en suivant les prévisions extrapolées de 1'analyse et de 1'evaluation de paramètres significatifs.

Le choix de la politique de maintenance dépend de plusieurs critères :

- L'évaluation du risque :

Certaines défaillances des machines peuvent causer des conséquences très importantes sur la rentabilité de l'entreprise, elles peuvent provoquer :

• Des produits de mauvaise qualité, défectueux notamment à cause des arrêts et des redémarrages.

• Des arrêts de travail provoquant des retards de livraison, ce qui peut amener à payer des pénalités.

• Des rejets qui polluent l'environnement et menacent la vie des personnes.

Suivant l'influence des pannes, on peut classer les risques de la façon suivante:

• **Risque inacceptable :** il faut supprimer les causes de panne et supprimer les effets des pannes résiduelles, on se base sur la notion de maintenance préventive conditionnelle.

• **Risque acceptable** si limité : il faut choisir entre maintenance préventive systématique et maintenance conditionnelle.

• **Risque acceptable** : dans ce cas on peut choisir la maintenance corrective.

-La charge la machine :

Selon ce critère, c'est la charge et la capacité de la machine qui comptent. L'influence d'une défaillance de la machine varie selon la charge de la machine :

• Si la machine est totalement chargée alors il est préférable d'éviter les défaillances, on utilise ici la maintenance préventive.

• Si la machine a une charge faible on utilise une politique de maintenance corrective.

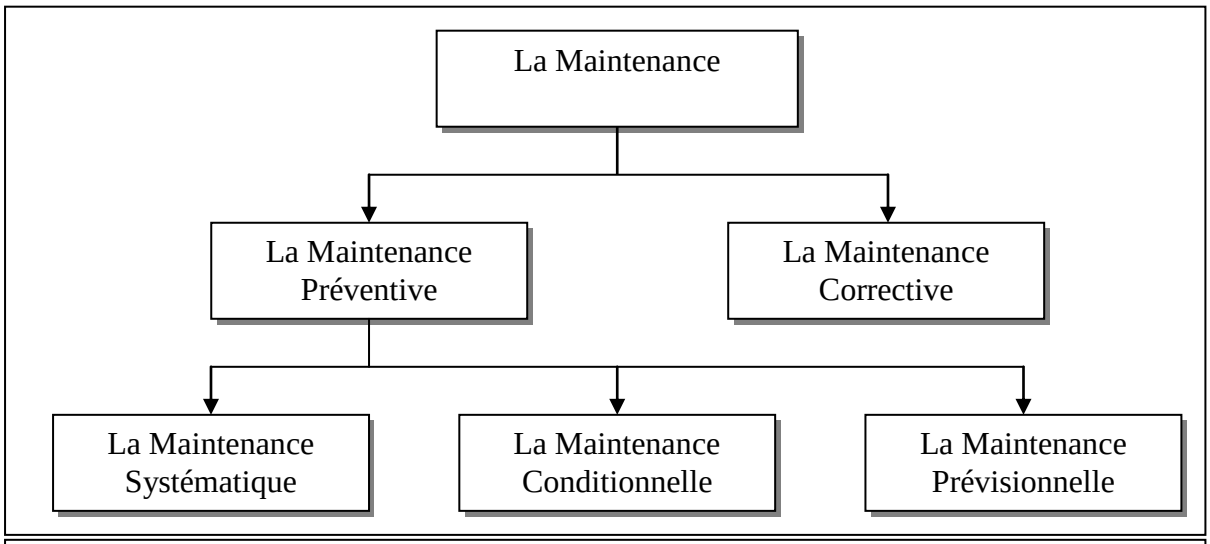


Figure 6-1 : les différents types de maintenance

VI-4- système d'information de maintenance préventive :

La maintenance préventive consiste à suivre l'évolution de l'état d'un organe, de manière prévoir une intervention dans un délai raisonnable et l'achat de pièces de rechanges nécessaire

Elaborer un système d'information de maintenance préventive, c'est décrire toutes les opérations de maintenance préventive qui devront être effectués sur chaque organe, la réflexion sur l'affectation des opération de maintenance se fait en balayant tout les organes de la décomposition fonctionnelle et en tenant compte de leur technologie, leur environnement, leur utilisation, et leur probabilité de défaillance.

Les différentes sources qui nous aident à définir les opérations de la maintenance préventive sont :

- les documents techniques de constructeurs.
- Les expériences des dépanneurs et conducteurs.
- Les historiques de la machine concernée et éventuellement celles des machines de même type.
- Les recommandations des constructeurs.
- Les valeurs de MTBF (Mean Time Between Failure).
- Les conditions d'utilisation.

La mise en œuvre de telle système nécessite de Gérer la pièce de rechange (PR), qui est une ressource des plus importantes, revient à gérer le stock en quantité stratégique juste suffisante pour répondre au besoin du bon fonctionnement de l'entreprise. Et c'est justement l'objectif que nous sommes fixés dans notre modélisation, car nous avons prévu des agents logiciels dédiés à réaliser cette stratégie.

Le « zéro stock » est un abus de langage car, le stock est indispensable, mais un stock dépassant un seuil non toléré affecte négativement l'entreprise.

Un sous-stockage provoque une rupture de stock, ce qui peut générer l'arrêt de la machine qui peut à son rôle altérer la chaîne de production.

Par contre, un sur-stockage provoque des coûts inutiles (assurances, locaux, gardiennage, vieillissement, etc.), donc n'a pas besoin de tenir au stock les pièces de rechanges si le délai normal le permet.

VI.5- les concepts de base d'un système multi agents

VI.5.1- Qu'est qu'un Agent?

Parmi les plusieurs notions d'agent, on peut définir un agent [44] comme une entité physique ou virtuelle [45]:

- qui est capable d'agir dans un environnement,
- qui peut communiquer directement avec d'autres agents,
- qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
- qui possède des ressources propres,
- qui est capable de percevoir (mais de manière limitée) son environnement,
- qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- qui possède des compétences et offre des services,
- qui peut éventuellement se reproduire,
- dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

VI.5.2- Environnement

L'environnement est communément défini par tout ce qui entoure l'agent. C'est l'endroit d'immersion dans lequel évoluent les agents. L'environnement peut être décomposé en deux parties. L'environnement social qui est composé des autres agents du système et l'environnement physique au sein duquel les agents évoluent. Un agent peut baigner dans l'un et/ou l'autre de ces environnements. L'environnement physique dépend largement de l'application traitée alors que l'environnement social dépend souvent des choix de conception du système multi-agents.

L'environnement [46] est une première classe d'abstraction qui fournit les conditions environnantes aux agents pour exister, et qui sert d'intermédiaire pour les interactions entre agents ainsi que l'accès aux ressources

Un certain nombre de propriétés pour l'environnement qui sont les classées de telle manière que l'environnement peut être :

– Accessible (par opposition à inaccessible), le système peut obtenir une information complète, exacte et à jour sur l'état de son environnement. Dans un environnement inaccessible, seule une information partielle est disponible.

– Déterministe (par opposition à non déterministe), une action a un effet unique et certain. Si le système agit dans son environnement, il n'y a aucune incertitude sur l'effet de son action sur l'état de l'environnement. L'état suivant de l'environnement est complètement déterminé par l'état courant. Dans un environnement non déterministe, une action n'a pas un effet unique garanti.

– Dynamique (par opposition à statique), l'état de l'environnement dynamique dépend des actions du système qui se trouve dans cet environnement, mais aussi des actions d'autres

processus. Ainsi, les changements ne peuvent pas être prédits par le système. Un environnement statique ne peut changer sans que le système agisse.

– Continu (par opposition à discret), le nombre d'actions et de perceptions possibles dans cet environnement est infini et indénombrable. Dans le cas discret, l'environnement est défini par un nombre d'états fini et selon un découpage régulier tel qu'une grille.

VI.5.1- Système

Nous utilisons l'approche des Systèmes Multi-Agents (SMA) comme moyen de modélisation, car ces derniers ont fait leurs preuves pour les systèmes distribués complexes, et répondent efficacement à la modélisation de ce genre de systèmes,

Et on peut définir un système Multi-agents (SMA) comme étant un système composé des éléments suivants [44] :

- un environnement E, c'est-à-dire un espace disposant généralement d'une métrique ;
- un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents ;
- un ensemble A d'agents, qui sont des objets particuliers (A. O), lesquels représentent les entités actives du système ;
- un ensemble de relations R qui unissent les objets (et donc les agents) entre eux ;
- un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler les objets de O ;
- des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.

On peut définir un système multi agents comme [47] un réseau faiblement couplé d'agent de problèmes qui interagissent pour résoudre des problèmes qui sont au-delà des capacités individuelles ou des connaissances de chaque agent.

L'utilisation d'un SMA dans une application d'environnement ouvert comme l'entreprise étendu, offre trois avantages principaux : l'extensibilité, la stabilité et l'équilibrage de charges.

– extensibilité : le système peut être mis à jour et étendu sans effets collatéraux quand il est en opération. Etant donné que les sites sont indépendants, ils peuvent être créés ou modifiés, alors que d'autres sites continuent à travailler et à fournir des services.

– stabilité : quand un site tombe en panne, d'autres sites peuvent coordonner la distribution des tâches et assurer la continuité des services.

VI.5.4- La communication dans les SMA

Tout d'abord, les agents doivent être capables, par le biais de la communication, de transmettre des informations, mais surtout d'induire chez l'autre un comportement spécifique

Communiquer est donc une forme d'action particulière qui, au lieu de s'appliquer à la transformation de l'environnement, tend à une modification de l'état mental du destinataire.

Par exemple, demander à un autre d'exécuter une tâche, tend à provoquer chez l'autre une intention d'accomplir cette tâche et constituent donc une manière de satisfaire un objectif sans réaliser la tâche soi-même

Le réseau contractuel (RC) est un exemple de système de coopération. C'est une collection de noeuds qui coopèrent pour la réalisation d'un but donné. Ces noeuds sont en principe des objets ou de simples entités capables d'agir. Ils se transforment en agents dès qu'ils adoptent un but. Les noeuds du RC comme tout réseau communiquent entre eux. Nous pouvons imaginer qu'ils sont tous connectés à un réseau de typologie quelconque

Langages de communication dans les SMA, Dans un système complexe hétérogène où collaborent des agents écrits dans différents langages et sous différentes plates-formes logicielles, la nécessité de définir un langage de communication est évidente. Ce langage doit cependant aussi bien définir le fond que la forme des échanges.

Ainsi lorsqu'un objet reçoit une information, il doit savoir de quoi il s'agit et éventuellement comment la traiter.

La définition d'un langage de communication permet, en outre, de faire coopérer facilement plusieurs versions d'un même logiciel : Une fois le langage défini, une partie du logiciel peut évoluer sans que les autres parties n'ait à évoluer de leur côté.

Ceci suppose que dès le départ, le langage de communication choisi soit le plus complet possible, mais aussi et surtout extensible. C'est en effet grâce à cela que l'évolution et l'interopérabilité des différentes briques du logiciel peuvent être garanties

L'Interaction et la Communication sont deux principes qui se côtoient souvent dans le processus d'existence d'un SMA. Nous expliquons, dans ce chapitre, que l'interaction est l'essence même de la dynamique d'un SMA, et que la communication n'est qu'un support de messages qui provoque la dynamique du système. Nous y présentons donc le principe même de l'interaction, ainsi qu'un état de l'art des différents modèles d'interaction existants.

VI.6- Eléments d'une interaction

Nous considérerons que les interactions entre agents se produisent dans le contexte de conversations, c'est à dire de séquences d'interactions élémentaires (partiellement) ordonnées par des relations causales. Chaque conversation se déroule selon les règles, conventions, etc. déterminées par un certain protocole d'interaction, qui est caractérisé par [48] :

- Les rôles que les participants peuvent jouer, la notion de rôle étant ici circonscrite à un protocole donné et non pas à l'échelle de l'ensemble du système ; par exemple, un protocole d'enchère comporte les rôles de vendeur, commissaire et d'enchérisseur.
- Le type des interventions, ou interactions élémentaires, que les participants d'une conversation peuvent effectuer les uns envers les autres.
- L'état initial, c'est à dire les conditions qui doivent prévaloir pour l'ouverture d'une nouvelle conversation.
- L'état terminal c'est à dire les conditions qui marquent la fin d'une conversation et correspondent soit à son échec soit à son succès (on suppose que chaque conversation a un objectif, qui est ce qui justifie que des agents y participent, et le succès est la caractérisation de l'atteinte de cet objectif).

- Des contraintes de distribution qui portent sur l'attribution des rôles aux agents : les conditions qu'un agent doit satisfaire pour pouvoir jouer un certain rôle, le nombre des agents

qui peuvent jouer chaque rôle, la compatibilité entre les rôles, les possibilités de changer de rôle, l'ordre dans lequel les rôles peuvent être attribués.

- Des contraintes de comportement qui portent sur les occurrences d'intervention et déterminent les séquences d'interventions qui peuvent constituer le cours d'une conversation : dans quelles circonstances un participant jouant un certain rôle peut (ou doit) réaliser une intervention d'un certain type vis à vis d'autres participants spécifiés?

Une conversation qui suit un certain protocole est donc une instance de ce protocole, c'est à dire un processus au cours duquel des agents réalisent des interventions conformes aux contraintes du protocole. La variété des conversations que l'on peut envisager est considérable, par exemple si l'on considère un protocole dans lequel il n'y a qu'un seul rôle, tout énoncé du langage naturel est un type d'intervention, toute situation correspond à un état initial et terminal, et il n'y a aucune contrainte de distribution et de comportement.

La définition et la mise en place d'un protocole mettent en jeux un grand nombre de mécanismes et de propriétés qui concernent non seulement le protocole lui-même, mais aussi les entités en interaction, le médium de communication (ou l'infrastructure, l'environnement dans lequel chacun des agents s'exécute) et les mécanismes de régulation qui peuvent exister au niveau de l'ensemble du système.

L'ensemble de tous ces éléments caractérise ce que l'on pourrait appeler un espace d'interaction ; cet espace détermine les caractéristiques des conversations susceptibles de se dérouler, et donc le type d'organisation du système. Il en découle une démarche de conception des SMA comportant les étapes suivantes [48] :

_ Choix du type d'organisation.
_ choix du type d'espace d'interaction, en fonction des protocoles mettant en œuvre cette organisation.
_ Choix techniques concernant les agents et leur environnement d'exécution.
_ Définition des mécanismes de régulation qui implantent ces protocoles en déterminant comment les agents utilisent les ressources de l'environnement d'exécution, et inversement les contraintes imposées par cet environnement sur le comportement des agents.

Les communications inter-agents ne permettent généralement pas de faire émerger aisément des enchaînements conversationnels. C'est la notion de protocole qui permet d'expliciter ces enchaînements afin d'arriver à l'émergence de l'objectif recherché.

VI.6.1- Choix d'un protocole

Le choix du protocole est très important, surtout parce qu'un protocole ou un autre peut imposer un certain comportement (préféré) aux agents. Si les agents sont coopératifs,

Le concepteur peut imposer une certaine stratégie aux agents afin d'avoir le comportement global désiré dans la société d'agents. Si les agents sont égocentrés ou compétitifs, chaque agent est libre de choisir sa propre stratégie, notamment la stratégie qui lui apporte la plus grande utilité. Dans ce contexte, il est important de connaître les critères d'évaluation des protocoles au moment de la conception ou du choix du protocole

VI.6.2-Classification des protocoles d'interaction

Les agents communiquent entre eux dans le système pour s'échanger des informations tout en ayant comme objectif d'accomplir des buts qui leurs sont propres ou partagés. La communication peut permettre aux agents de coordonner leurs actions et aboutir ainsi des comportements cohérents du système [49] .

La coordination est une caractéristique essentielle dans les SMA. Elle permet, par exemple, d'éviter le conflit sur les sections critiques, en matière de ressources, les attentes indéfinies et les inter-blockages. La coopération est la forme de coordination entre agents non antagonistes; à l'inverse, la négociation correspond à la coordination en univers compétitifs ou entre agents "égoïstes" ou concurrentiels. Typiquement, pour coopérer avec succès, chaque agent doit maintenir un modèle des autres agents, et développer également un modèle des futures interactions (planification).

L'objectif des protocoles s'inscrit dans deux types d'interactions:

_ Agents concurrents: il faut maximiser l'utilité de chaque agent.

_ Agents ayant des buts semblables ou des problèmes communs: il faut maintenir des performances globalement cohérentes sans inhiber l'autonomie de chacun. Les aspects importants de ce type d'interaction sont:

- _ déterminer les buts partagés
- _ déterminer les tâches communes
- _ éviter les conflits
- _ mettre en évidence la connaissance et la partager

1. Protocoles de coordination

La coordination se traduit par un comportement individuel visant à servir ses propres intérêts, tout en essayant de satisfaire le but global du système [49]. Cela se fait essentiellement dans des environnements à ressources limitées. Plusieurs raisons exigent la coordination entre agents d'un SMA:

_ dépendances entre actions des agents,

_ Aucun agent ne peut posséder une auto- suffisance en matière de ressources et de compétences afin d'atteindre ses buts tout seul,

_ éviter la redondance dans la résolution des problèmes.

Le contrôle distribué, sur lequel se sont concentrés les recherches récemment, concerne surtout l'autonomie des agents qui leur permettent de produire de nouvelles actions et décider des objectifs à atteindre. L'inconvénient majeur de la distribution du contrôle et des données est que la connaissance de l'état global du système est dispersée sur tous les agents où chaque agent possède une perception partielle et imprécise. Il y a plus grand degré d'incertitude au sujet des actions de chaque agent ce qui rend difficile l'atteinte d'un comportement global cohérent, la coordination[50] est caractérisée par deux aspects étroitement liés:

_ Les engagements fournissent la structure nécessaire pour des interactions prévisibles de sorte que les agents puissent prendre en compte les dépendances inter-agents des futures activités, des contraintes globales ou des conflits d'utilisation des ressources. Pendant que les situations changent, les agents doivent évaluer si les engagements existants sont encore valides.

_ Les conventions fournissent des moyens pour contrôler les engagements dans des circonstances changeantes. Elles contraignent les conditions pour lesquelles des engagements

Devraient être réévalués et indiquent les actions associées qui devraient alors être entreprises: maintenir, rectifier ou abandonner les engagements. Par exemple, si les buts sont dépendants, il est essentiel que les agents soient au courant de n'importe quel changement substantiel qui les affecte. Un autre cas, lorsque les agents s'engagent ensemble à satisfaire un but et contribuent individuellement à des tâches (participation d'une équipe entière), un changement d'engagement par un participant peut compromettre les efforts de l'équipe. par conséquent, les agents auront besoin d'être informés et de se repositionner par rapport aux engagements pris collectivement.

2. Protocoles de négociation

La négociation est une interaction entre agents basée sur la communication avec le but d'arriver à un accord. Elle peut être vu comme un processus par lequel une décision commune est prise par deux agents ou plus où chacun d'entre eux essayant d'atteindre leurs buts ou objectifs propres.

Les principaux dispositifs de la négociation sont:

_ Un langage de communication, utilisé par les agents participants

_ Un protocole de négociation suivi par les agents durant tout le processus de négociation

_ Un processus de décision par lequel un agent décide ses positions, concessions et critères pour l'accord.

La négociation peut se faire selon :

_ Négociation un -à- un

_ Négociation un -à- plusieurs

_ Négociation plusieurs -à- plusieurs

Les techniques de négociation peuvent être centrées environnement ou centré agent (égocentré):

1. Négociation centrée agent: Quelle est la meilleure stratégie pour un agent évoluant dans un environnement donné?.

2. Négociation centrée environnement: Quelles sont les règles à appliquer sur un environnement afin que les agents faisant parti de cet environnement puissent être productifs?.

Le mécanisme de négociation doit avoir les caractéristiques suivantes:

_ L'efficacité, ne pas perdre de ressources pour aboutir à un accord.

_ La stabilité qui consiste, pour un agent, de ne pas sortir de la stratégie tracé initialement.

_ La simplicité, en coût de calcul et de bande passante.

_ La distribution, qui évite l'utilisation d'un agent central superviseur.

_ et la symétrie

Les techniques de négociation peuvent être:

_ Négociation basée sur la théorie des jeux

_ Négociation basée sur des heuristiques

_ Négociation basée sur l'argumentation

Coté ingénierie de protocoles, comme chaque agent peut choisir sa propre stratégie. Ceci permet aux agents du SMA d'être conçus et implémentés différemment, tout en assurant un middleware de communication entre les agents du système. De plus, on ne peut pas assumer que les agents utilisent des stratégies "coopératives" de négociation.

3. Protocoles de coopération

Ces protocoles s'appliquent surtout à la résolution des problèmes complexes. Cela consiste à la décomposition du problème en tâches affectées à des agents distribués. Cette décomposition doit prendre en considération les capacités et les ressources de chaque agent du système. Une telle décomposition peut réduire la complexité d'un problème.

Soit un problème, même s'il ne possède pas les critères d'un problème complexe mais nous pouvons lui appliquer un protocole de coopération. Cela consiste à traiter une image complexe en lui appliquant des opérations de morphologie mathématique (érosion, dilatation, ouverture, fermeture, etc.). Cette image peut être divisée en plusieurs segments, et à chaque segment on lui affecte un agent qui assure une certaine tâche bien précise. Sachant que tous agents travaillent en coopération en traitant les frontières entre les segments [51] .

D'une façon générale, la décomposition d'un problème complexe peut être spatiale ou fonctionnelle. Une fois la décomposition faite, il faut distribuer les tâches selon les critères suivants [49] :
_ Éviter la surcharge des ressources critique.
_ Assigner les tâches aux agents ayant des capacités correspondantes.
_ Assigner les tâches interdépendantes à des agents proches spatialement ou sémantiquement pour limiter les coûts de communication et de synchronisation.
_ Réassigner des tâches pour accomplir les plus urgentes.

Plusieurs mécanismes sont utilisés pour distribuer des tâches. En voici quelques-uns:
_ Mécanisme d'élection:des tâches sont attribuées aux agents suite à un accord ou un vote.
_ Réseaux contractuels:cycles d'appels d'offres, de propositions et attribution à un agent.
_ Planification Multi-agents: Des agents dédiés à la planification des tâches.
_ Structure organisationnelle: les agents ont des responsabilités pour des tâches bien particulières.

VI.7- Équilibrage de charge dans les Systèmes Multi-agents

Chaque agent activant dans un Système Multi-agents est doté d'une charge de travail ainsi qu'une charge de ressources, à définir et à gérer selon ses capacités. Cela est caractérisé par un ensemble de tâches que cet agent est en devoir d'effectuer, durant un intervalle de temps

Comme cet agent peut être sur-chargé ou sous-chargé, cela peut affecter négativement l'ensemble des noeuds (ou sites) du système. Alors, il doit coopérer avec son environnement,c'est-à-dire l'ensemble des agents représentant les sites avec lesquels il entretient une relation, appelés généralement accointances, affin d'aider à réaliser un équilibrage de charge, pouvant assurer une meilleure émergence de solution individuelle ou globale.

Les Systèmes multi-agents sont implémentés actuellement sur des réseaux, sans lesquels ils ne peuvent pas être validés. Ces réseaux peuvent être des réseaux locaux ou carrément l'Internet. Ils présentent aussi de nouvelles opportunités pour développer des applications parallèles très performantes.

Un tel système dans lequel le nombre de ses membres augmente et les actions de ces membres nécessitent des grandes quantités de ressources. Ces dernières ne sont pas forcément centralisées.

Un service d'équilibrage de charge permet aux différents membres de ce système d'accéder et de bénéficier pleinement de ces ressources distantes afin de les utilisées au mieux de manière transparente

Le but de toute technique d'équilibrage de charge (load-balancing) est d'optimiser l'utilisation des processeurs, chargés de prendre en charge, l'exécution des tâches dédiées à ces charges tout en spécifiant ses localités.

Autrement dit, l'équilibrage de charge doit minimiser le temps d'exécution d'un ensemble donnée de tâches, ce qui revient souvent à maintenir une charge équivalente sur l'ensemble des processeurs. En général ce mécanisme est aussi utilisé pour gérer l'équité de temps d'exécution entre les tâches.

L'équilibrage de charge est également désigné sous les termes : la régulation de charge, le partage de charge ou encore l'ordonnancement distribué des charges, désigne toutes les techniques utilisées pour transférer la charge de travail des noeuds (ou sites) surchargés vers des noeuds moins chargés en respectant la capacité de chaque noeud, c'est-à-dire déterminer celui des noeuds le moins chargé et le plus apte à faire tourner l'application.

La vision nouvelle qui s'impose actuellement est la gestion et l'équilibrage des ressources, surtout en matière de logistique, dans une entreprise distribuée, dont les ressources sont distribuées et dont l'équilibrage constitue la condition de base pour la bonne gestion de l'ensemble.

VI.8- Les ressources dans une entreprise étendue

VI.8.1- Le cas de pièce de rechange

L'identification est un point crucial dans le traitement de la pièce de rechange. Dans ce cadre, la documentation est le point essentiel qui permet d'identifier cette ressource. Elle doit reposer sur des documents très fiables, informatisés ou non, tels que; dessins, photographies,

La gestion de stock de pièces de rechange pour la maintenance doit permettre [AFNOR]:

- trouver l'identification d'une pièce sur le matériel,
- identifier une pièce à vue hors matériel,
- savoir sur quels matériels est montée une pièce à partir de son identification,
- reconnaître une pièce sur le matériel à partir de son identification (repère topo-fonctionnel et éclaté),
- connaître le nombre d'exemplaires d'une pièce sur un matériel,
- connaître le nombre de matériels sur le site qui utilisent une pièce,
- connaître les pièces strictement équivalentes, interchangeables ou adaptables (avec les contraintes et limites d'utilisation),
- déterminer l'ensemble supérieur à laquelle appartient cette pièce,
- trouver éventuellement ses caractéristiques fonctionnelles, dimensionnelles économiques et techniques - y compris ses plans - particulièrement dans le cas de pièces spécifiques à un équipement réalisé par l'utilisateur,

- trouver ses caractéristiques logistiques (réparables ou non, degré et niveau de maintenance, gamme de dépose, pose, test et réparation, conditions de stockage, durée de validité),
- identifier le ou les fournisseurs de cet article et surtout les autres partenaires de l'entreprise étendue avec lesquels la coopération est possible et avoir accès à leur catalogue, particulièrement sur Internet.

L'objective visé est donc gérer le stock de PR d'une façon de chercher un stock optimal S*, au sens coût et disponibilité sur une période T, afin de ne pas exposer l'entreprise à une situation anormale. Car le problème de la gestion de la pièce de rechange est un problème de gestion de stock et non pas de tenue de stock, c'est-à-dire optimiser les quantités et les coûts de stockage d'articles de façon à prévenir ces situations.

Donc le problème est la réduction des stocks puisque les stocks représentent de l'argent qui dort (baisse de la trésorerie, augmentation des besoins en fond de roulement…), mais ils nécessitent également des moyens pour assurer leur gestion. Les coûts de gestion des stocks représentent les coûts de magasinage (salaires, loyers, chauffage, entretien, engins de manutention, informatique, assurances…).

VI.8.2- Les indicateurs de la disponibilité

Chaque fois qu'un équipement ou un sous ensemble de cet équipement tombe en panne, il faut un certain temps pour le remettre en état. La vie d'un équipement ou d'un article est donc composée de temps de bon fonctionnement et de temps de réparation. Les temps de réparation ne comprennent pas les temps de maintenance préventive.

A partir de ces définitions et en analysant la vie d'un équipement, on peut considérer des indicateurs :

_ MTBF (Mean Time Between Failure) : moyenne des temps entre deux défaillances.

_ Taux de défaillance: (inverse du MTBF) représenté par la lettre ¸ : Moyenne du nombre de défaillances par unité de temps (non compris les temps de réparation)

_ MTTR (Mean time To Repair) : moyenne des temps de réparation:

$$MTTR(enheures) = \frac{Totaltempsréparation}{Nombredepannes}$$

_ Disponibilité : aptitude du système, sous les aspects combinés de sa fiabilité et de sa logistique de maintenance, à remplir une fonction à un instant donné ou dans un intervalle de temps donné:

$$Disponibilité / \frac{MTBF}{←MTBF→MTTR←}$$

VI.8.3- Classification de la Pièce de Rechange dans une entreprise étendue

Nous pouvons appliquer la classification dite de ABC [52] , basée sur le principe de 20/80 de Pareto à cette ressource. C'est une classification qui peut s'appliquer selon les prix, les quantités utilisées, ses fréquences d'utilisation, les quantités minimales d'achat, les délais. Mais surtout leur impact en cas d'indisponibilité, etc. :

Classe A, appelée aussi classe d'Articles à Rotation Lente (SMI): Articles très chers, rares, délais longs. Ils ne doivent pas être disponibles sur tous les magasins de maintenance. Ils peuvent être mis en disponibilité commune, dans un site bien particulier, pour l'ensemble des sites de l'entreprise.

Classe B, appelée aussi classe d'Articles à Rotation Moyenne (NMI) : Articles moyennement chers, disponibilités aléatoires sur le marché. Doivent être dans des sites bien particuliers et accessibles, au moment opportun, pour l'ensemble des sites de l'entreprise étendue.

Classe C, appelée aussi classe d'Articles à Rotation Rapide (FMI) : Articles courants, généralement peu chers. Ils peuvent être disponibles dans tous les magasins de l'ensemble des sites, mais en quantités optimales.

La gestion d'un article est spécifique à la classe à laquelle il appartient. La politique de réapprovisionnement en découle étroitement. C'est en fonction de cette classification que les réponses aux questions suivantes doivent être trouvées:

_ Quoi réapprovisionner?
_ Quand réapprovisionner?
_ Combien réapprovisionner?

VI.8.4- Politiques de réapprovisionnement pour un site donné

Selon la période T et la quantité de stock S prévue. Quatre grandes politiques sont à prendre en considération :

1. T fixe et S fixe:

Cette méthode dite " calendaire " prévoit des livraisons des articles à des dates fixes. C'est une méthode qui s'applique généralement pour les articles de faible valeur de la classe C, quand la consommation est surtout régulière.

Avantages :

_ Simplicité de la gestion de l'article,
_ Gains d'échelle négociables par les acheteurs.

Inconvénients :

_ Simplicité de la gestion de l'article.

2. T variable et S fixe :

Cette méthode dite de " point de commande " consiste à définir le niveau de stock, appelé point de commande ou seuil de commande, qui déclenche l'ordre d'achat de façon à être livré juste au moment de l'utilisation du dernier article.

Avantages :

_ Méthode qui permet d'éviter la rupture de stock
_ C'est une méthode qui est adaptée surtout aux consommations partiellement irrégulière.

Inconvénients:

_ Suivi permanent des stocks.

3. T variable et S variable :

Cette une méthode qui est utilisée principalement pour les articles de la classe A, dont le prix de revient varie fortement ou dont la disponibilité n'est pas permanente. C'est une commande opportuniste.

Avantages :

_ Méthode qui permet de profiter de tarifs intéressants.

Inconvénients :

_ Faire un suivi permanent afin de guetter les prix les plus intéressants,

_ Applicable surtout pour un type d'article bien particulier, particulièrement ceux de la classe A.

VI.9- Conclusion

Aujourd'hui, les Systèmes Multi-Agents (SMA) commencent à s'imposer en tant que modèle à analyser et à structurer tout genre de phénomène sociétal, mettant en œuvre des entités interactives distribuées et surtout coopérative. Les entreprises étendues sont le meilleur exemple, car elles sont distribuées géographiquement et fonctionnellement.

Elles présentent tous les critères d'une prise en charge conceptuelle et d'une mise en oeuvre adéquate par les SMA, et ce pour leur bon fonctionnement Modéliser en SMA, c'est aussi travailler sur un modèle d'organisation qui répond au mieux au bon fonctionnement du problème posé, car nous pensons qu'il n'y pas de modèle générique pouvant satisfaire tous les critères d'organisation à un problème donné.

En outre, un modèle d'organisation est intimement lié à un modèle d'interaction qu'il faut développer et bien valider. L'expérience a montré qu'il faut partir d'un problème posé, lui appliquer un modèle d'organisation qui soit lié à des scripts d'interaction qu'il faut bien valider.

CHAPITRE VII

MODELISATION DES AGENTS EN SMA

Ce chapitre est destiné à décrire les modelés et les scripts utilisés pour formuler mathématiquement le problème d'équilibre de ressources.

L'objectif est de présenter un modèle d'organisation conceptuelle, de spécifier les scripts des agents et le modèle d'interaction, et spécifier le rôle et l'objectif de chaque agent du système.

Mot clé :

SMA, Agents,groupe , coordinateur de groupe , superviseur site , Entreprise étendu, équilibre de charge,

Chapitre VII

Modélisation des agents en SMA

VII-1- Introduction

Les politiques de réapprovisionnement que nous avons citées présentent la stratégie à adopter vis-à-vis de chaque article, après étude au préalable (la demande, coût, période, etc.), mais s'appliquent surtout pour un site autonome.

La politique de réapprovisionnement dans une entreprise étendue dépend des besoins de l'ensemble des sites interconnectés entre eux. Les articles de la classe A, par exemple, peuvent se retrouver dans tous les sites, mais les articles supposés appartenir aux classes B et C peuvent être disponibles dans des sites particuliers, considérés importants par les gestionnaires de l'entreprise étendue.

Le principe de gestion, dans ce cas, est de maintenir un équilibrage, en disponibilité en premier et en coût plus tard, entre tous les sites et pour tous les articles dont il a besoin chacun de ces sites.

Pour ce faire, il s'agit de trouver les solutions adéquates, conceptuelles et implémentatives, afin de prévenir et éventuellement d'éviter un déséquilibre de disponibilité de ces ressources dans un contexte multi-sites distribué. Avec la contrainte que chacun des sites offre et demande des ressources différentes. Sachant que chaque ressources obéit à des critères différents de coût, d'importance en cas de panne, etc.

VII-2- Pourquoi une modélisation Système Multi-agents

L'architecture d'une entreprise étendue est distribuée et communicante. Elle est composée de N sites. Chacun d'eux, à son tour, peut être composé d'un ou de plusieurs autres sous sites. Sachant que l'ensemble est concerné par l'utilisation de la pièce de rechange. Chaque site est autonome par la politique adoptée vis-à-vis de la gestion préventive des articles.

_ L'interaction entre les sites du système est évidente, afin de réaliser, par coopération, l'objectif assigné par le système.

_ Cet objectif est dans ce cas: un équilibrage de ressources du système, ce qui exige donc une approche basée sur :

- _ La coopération entre les sites du système, pour l'émergence de la solution qui est l'équilibrage de la disponibilité des ressources de l'entreprise étendue.
- _ La dynamique du système, l'équilibrage continue au fil du temps.

C'est en se basant sur toutes ces caractéristiques que la modélisation du problème posé par l'approche SMA s'impose.

VII-3- Identification des différents agents de notre système

Sur la base des types et le volume de charge des tâches que doit effectuer le système. Une répartition des rôles sur des agents logiciels du système doit être faite.

Pour cela, nous avons pris en considération les différents rôles au sein de chaque site. Sur cette base, nous avons opté pour la répartition des rôles suivants [53] :

1. Agent superviseur.
2. Agent prévention.
3. Agent gestionnaire de stock.
4. Agent Coordinateur de groupe.
5. Agent Coordinateur Inter Groupe.

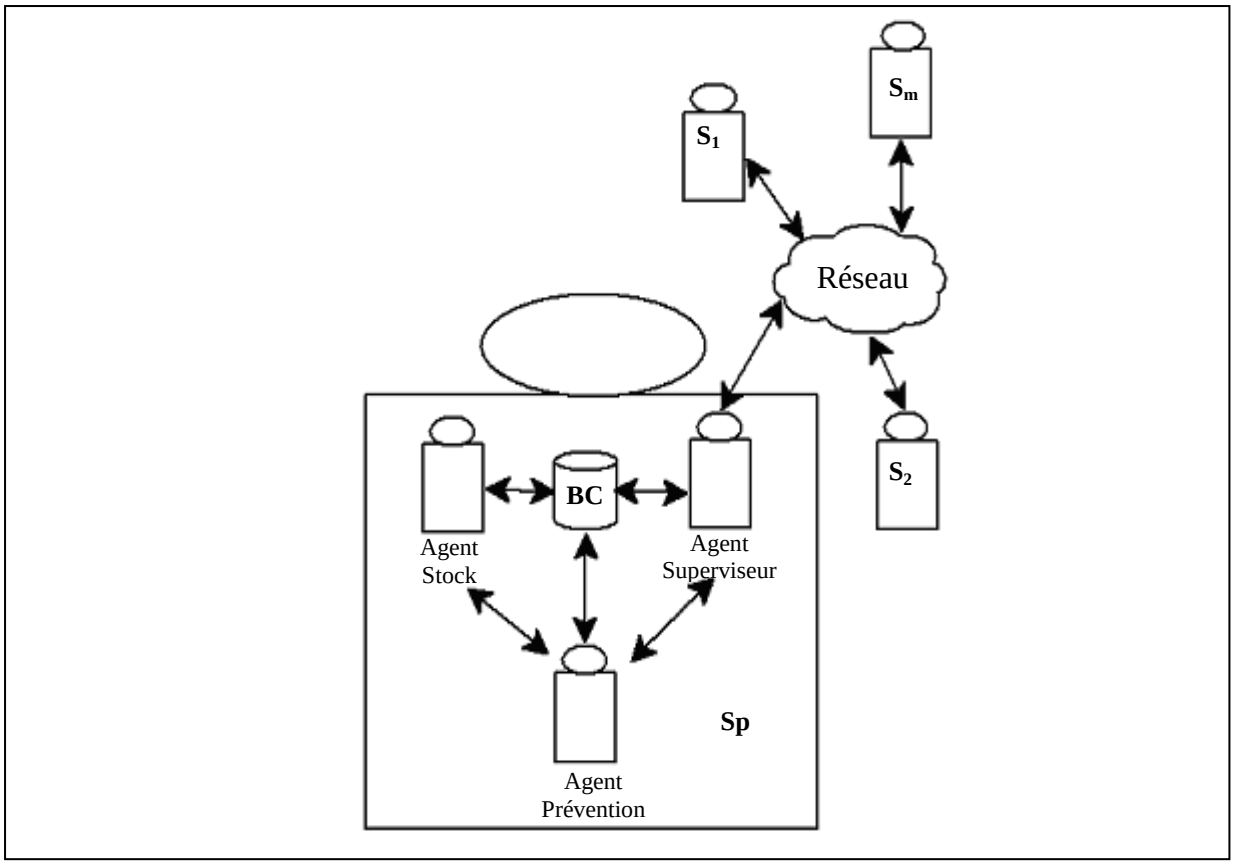


Figure 7-1 : Architecture des agents représentant un site.

Sachant que chaque site lui-même peut être considéré comme un agent. Car il sera représenté par un interlocuteur vis-à-vis des autres agents du système qui est l'agent superviseur.

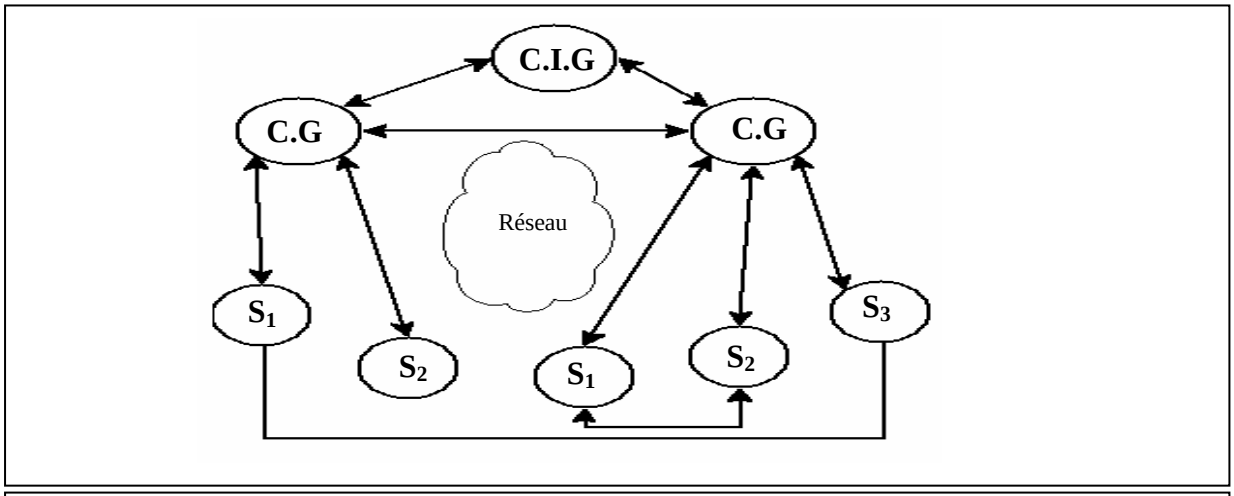


Figure 7-2 : Modèle d'organisation basée sur la notion du groupe

VII-4- Description des différents agents du système

Il en découle que le système possède des agents cognitifs et d'autres réactifs. L'agent prévention est un agent cognitif, c'est-à-dire qu'il possède une capacité de raisonner sur la base de son passé, qui est la consommation antérieure associée autres informations se trouvant dans sa base de connaissance.

1. Agent superviseur

C'est l'agent maître associé à un site, son rôle est de gérer et contrôler le fonctionnement du système au niveau de site. L'agent superviseur décrit l'ensemble des objectifs du système et observe son état.

C'est lui qui communique les quantités entrantes et sortantes d'articles. C'est lui aussi qui lance aussi les requêtes d'acquisition et d'offre des articles.

C'est le superviseur qui reçoit et envoie les messages aux autres sites. Il joue donc le rôle d'une interface de communication et de négociation par rapport aux autres sites pour acquérir une quantité de pièces de rechanges et offrir une autre quantité suivant son désir, donc Il est doté d'un mécanisme d'intelligence capable d'entrer dans des négociation avec d'autres sites:

```
//Je suis l'interlocuteur du site Sp
FAIRE_EN_PARALLELE
   SI "je reçois un message d'offre de Sm pour l'article k" ALORS
        // Entrer en négociation avec Sm
   FINSI
   SI "je reçois un message de demande de Sm pour l'article k" ALORS
        // Entrer en négociation avec Sm
   FINSI
   SI "je reçois un message de demande de Sm pour l'article k" ALORS
     // Entrer en négociation avec Sm
   FINSI
   SI "je reçois un message de sous-stockage de mon agent prévention" ALORS
     // Diffuser la demande dans mon environnement
   FINSI
   SI "je reçois un message de sur-stockage de mon agent prévention" ALORS
     // Diffuser l'offre dans mon environnement
   FINSI
FIN_FAIRE_EN_PARALLELE
```

Figure 7-3 : Script de l'agent Superviseur

2. Agent prévention

L'agent prévention est un agent intelligent, muni d'une capacité d'intelligence lui permettant de décider en fonction de sa base de connaissance (BC).

Il se base sur des algorithmes calculant les consommations antérieures et d'autres critères de disponibilité, de MTBF, de taux de défaillance, de MMTR, etc. Il a donc la capacité de prévoir les cas de sous-stock ou de sur-stock des quantités d'articles, sur la base des Tprev calculés pour chaque article. Il envoie des messages d'urgence à l'agent superviseur en lui indiquant, pour un type d'article, s'il y a sur-stockage ou sous-stockage.

Cet agent consulte donc la base de donnée d'une façon continue et périodique et guette les cas qui peuvent provoquer des situations anormales afin de prévenir et envoyer des messages d'alerte à l'agent superviseur.

Notre étude accorde plus d'importance à l'aspect prévention, donc au script de l'agent dédié à ce rôle, car cela assure une gestion optimale des ressources à l'entreprise étendue dans un cadre coopératif entre tous les sites de cette dernière. Le script de cet agent sera présenté une fois la démarche de calcul présentée

3. Agent gestionnaire de stock

C'est un agent réactif, dépourvu de toute forme d'intelligence, son rôle est la gestion de stock, au sens classique, telles que la consultation de la base de donnée et sa mise à jour. Cet agent reçoit l'ordre de l'agent superviseur.

4. Agent coordinateur de groupe CG

Un site est représenté par son superviseur, il est l'interlocuteur auprès des autres sites du système. Un groupe d'agents, selon un critère bien défini, est représenté par un agent Coordonnateur Groupe baptisé CG, auquel s'adressent toutes les requêtes émanant des superviseurs du groupe. Les agents qui appartiennent au même groupe d'agents, se reconnaissent via leurs superviseurs qui communiquent entre eux. C'est une forme d'agence immobilière regroupant toutes les informations sur les requêtes (offre et demande) des agents du groupe. En cas où la requête trouve une réponse favorable chez ce CG, donc il y a un agent répondeur. Ce CG met les deux agents en négociation directe

Le CG a, pour rôle, l'établissement d'une négociation entre agents d'un même groupe après recherche d'agents proposant des coûts optimaux. En général, un CG détient toutes les informations concernant les sur-stockages et les sous-stockages des différents articles utilisés par les membres du groupe et envoyés à leur CG périodiquement

Le rôle d'un CG associé à un groupe de sites est la coordination et la réception des besoins en quantité d'articles des différents sites du groupe, sur-stockage et sous-stockage pour chaque article. Il classe ces besoins, optimise les coûts pour chaque article et répond aux demandeurs.

Plusieurs demandes émanant de plusieurs demandeurs peuvent concerner le même article à un instant T donné, pour son meilleur coût ou sa rareté et provoquant ainsi une section critique.

La centralisation de la négociation d'un groupe au niveau d'un CG possède deux inconvénients qui sont :

_ Goulet d'étranglement.

_ risque de panne du coordonnateur du Groupe.

Elle présente néanmoins plus d'avantages que d'inconvénients:

_ Avoir le maximum d'informations disponibles à un instant T

_ Éviter les messages de diffusion (broadcast) aux sites du même groupe plusieurs fois pour l'acquisition ou l'offre d'un article quelconque.

_ Permet de déléguer des opérations aux autres groupes via le CIG.

_ La solution proposée est facilement implémentable.

5. Agent Inter-Groupe

Habituellement, un agent, en cas d'autosuffisance des ressources au sein du même groupe, reçoit ses requêtes essentiellement de la part des membres du groupe. Dans le cas

contraire, il peut toujours en recevoir d'autres, émanant d'autres sites appartenant à d'autres groupes, via le Coordonnateur Inter-Groupes, que nous avons appelé le CIG

Son rôle consiste en la contribution à l'établissement de l'équilibre des ressources au début, une fois cet équilibre atteint, il lui reste qu'à le surveiller, c'est-à-dire éviter le déséquilibre

C'est l'agent maître associé au système, jouer le rôle d'un serveur de nom, afin de garantir l'unicité de l'attribution des noms des articles et les noms des coordinateurs de groupe, et les superviseurs, qui les identifiera d'une façon unique sur tout le réseau.

Garantir l'auto- organisation de système, il inscrire les coordinateurs et les superviseurs, et assure la répartition idéal des superviseurs sur les coordinateurs de groupe suivant des critères tel que la distance, le type de pièces de rechange échanger entre les sites.

Recevoir les offres et les demandes non satisfait dans les groupes envoyés par les coordinateurs de groupe, et il est chargé de trouver un équilibre de ressource inter groupe et satisfait les demandes émanées.

VII-5- Description de notre modèle d'organisation

Nous avons développé le notre modèle, sur la base de nos besoins, il se base sur les éléments suivants :

Agent : c'est l'entité conceptuelle de base de notre organisation, pour le moment un site est représenté, vis-à-vis de son environnement, par l'agent superviseur. Et le site lui-même peut avoir une structuration interne SMA ou autre. En conséquence, un agent est:

- Autonome,
- En interaction intra-site avec les agents prévention et gestionnaire de stock et en interaction extra-site avec les autres agents superviseur du système,
- Distribué,
- possède un objectif,
- Notre agent prévention est un agent cognitif car il est doté d'une forme d'intelligence, il se base sur son passé pour prévenir son futur et agit au présent. Cet agent recherche une fonction de consommation d'articles durant les périodes précédentes afin de prévenir la durée qui reste à une rupture de stock et agit en conséquence.
- Groupe : il représente l'organisation des agents dans un groupe selon un critère bien établi. De ce fait un agent peut appartenir à plusieurs groupes.
- Interaction : Tout le système est basé sur l'interaction d'une entité vis à vis de l'autre, sans interaction, le système est statique.
- Rôle : Chaque agent à un rôle par rapport à son existence, mais exécute une ou plusieurs tâches à un instant $T_{courant}$.
- Autorité (ou privilège) : l'autorité est la règle qui régit l'organisation, l'intérêt du groupe CG doit passer avant celui de l'agent, et celui du CIG avant celui du CG, sur cette base, une stratégie de l'ensemble peut être appliquée.
- Et la tâche qui est la subdivision du rôle en séquences de tâches (en séquentiel ou en parallèle), c'est l'élément atomique par rapport aux SMA conceptuel. A l'implémentation, une tâche peut représenter un script s'exécutant sur un ou de plusieurs processus.

VII-6- Équilibrage coopératif et préventif de ressources [53]

1. L'objectif

L'objectif de notre SMA au moment des son exécution est la réalisation d'un équilibrage des ressources entre les différents sites du système. Pour ce faire, l'agent prévention de chaque site détecte les situation anormales de sur-stockage et de sous-stockage, et en informe son superviseur qui entre en interaction, par principe de coopération, avec les autres agents superviseurs des autres sites, affin d'assurer cet équilibrage. Dans ce qui suit, nous montrerons la démarche préconisée affin de réaliser cet objectif.

2. L'équilibrage de ressources

Supposons qu'un site S_1 dispose d'une quantité plus que suffisante (surplus ou sur-stockage) d'un article P sur une période T, que nous notons P+, au moment où le site S_2 accuse un défaut (déficit ou sous-stockage) pour le même article, durant la même période, que nous notons P-.

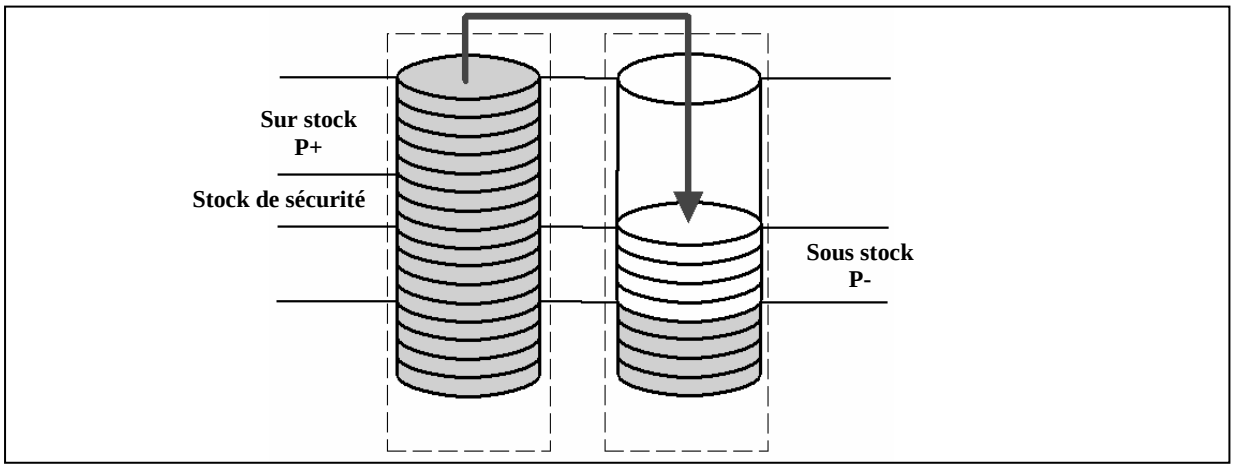


Figure 7-4 : Équilibrage entre sites pour une ressource.

Un équilibrage en cette ressource entre sites S_1 et S_2, serait simplement le transfert d'une quantité de P, à déterminer, de S_1 vers S_2.

3. L'équilibrage coopératif de ressources

Si S_1 dispose d'articles P_1et P_2 avec des quantités P_1+ et P_2- . Et S_2 dispose des mêmes articles, avec des quantités P_2+ et P_1- .

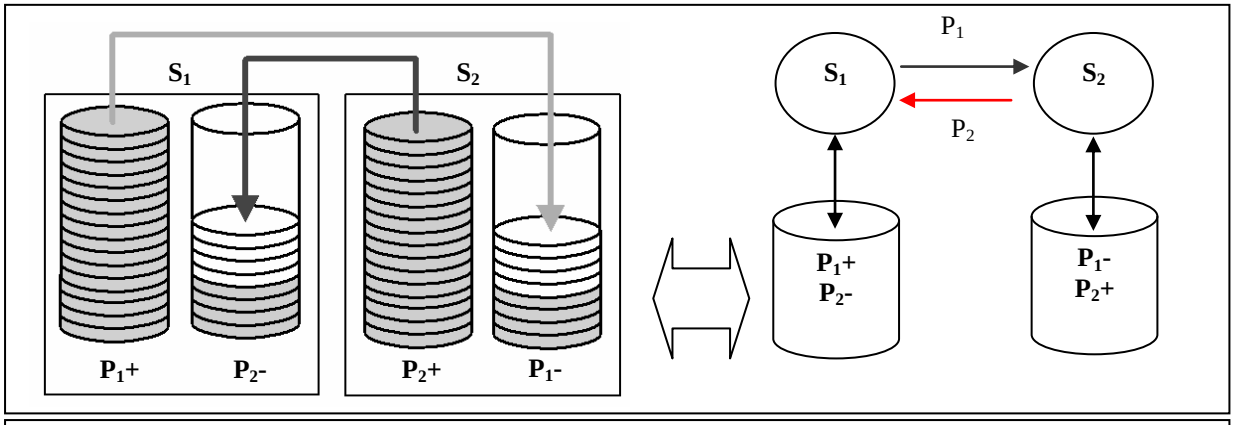


Figure 7-5 : Principe de l'équilibrage coopératif de ressources

Afin d'équilibrer les deux sites en ces ressources, S_1 doit envoyer à S_2 une quantité de P_1, selon des conditions posées par S_1 et acceptées par S_2 et peut recevoir du P_2 selon d'autres conditions.

Le même principe peut se généraliser à l'interaction d'un ensemble de N sites, chacun gérant M articles.

4. Choix sélectif de ressources

Supposons maintenant que le site S1 accuse un sous-stockage en articles P_1 et en P_2, alors que deux autres sites S_2 et S_3 possèdent des sur-stockages pour les mêmes articles, et proposent les articles P_1 et /ou P_2 au site S_1 à des coûts différents Le problème consiste à optimiser pour S_1 l'acquisition des articles P_1 et P_2. Plusieurs possibilités lui sont offertes

- acquérir P_2 du S_1 et P_1 de S_2
- acquérir P_1 du S_1 et P_2 de S_2
- acquérir P_1 et P_2 de S_1
- acquérir P_1 et P_2 de S_2

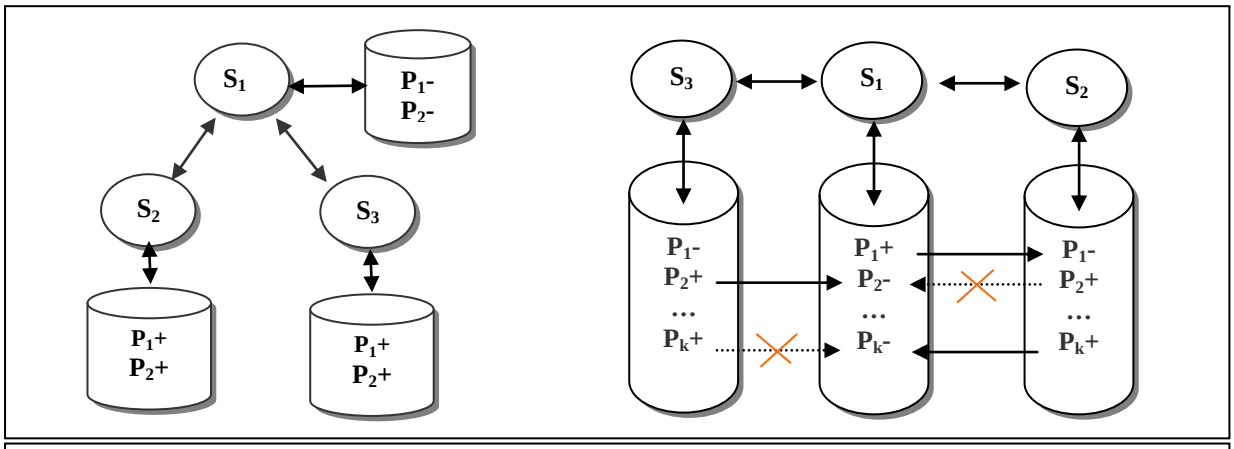


Figure 7-6 : Choix sélectif d'articles

D'une façon générale, un site peut se trouver offrant de M articles et acquéreur de P autres articles en même temps.

Sur cette base, le choix peut varier selon plusieurs critères :

_ Le choix peut être porté sur un critère de coût,

_ Le choix peut être porté sur le groupe,

_ Le choix peut être porté sur un critère géographique,

_ Le choix peut être prix sur une durée et sur la base d'un contrat de partenariat,

_ Mais, un choix peut être pris pour d'autres considérations, autres que celui du coût, au sens optimal.

5. la fonction de simulation de la consommation préventive

L'agent de prévention possède un algorithme pour détecter les situations anormales de sur-stockage et de sous-stockage suivant des consommations périodiques fixes, pour déterminer le niveau de disponibilité des ressources en quantité par rapport aux besoins future de site

Donc on simule la situation de stock pour déterminer la quantité approximative de stock en periode quelconque

La fonction de simulation est basée sur les quantités en stock pris en période fixe et bien déterminée, c'est une fonction d'approximation

Nous supposons que la loi de consommation d'un article sur un intervalle $\Delta t = T_{courant} - T_0$ qui est : $P = f(T)$ est linéaire

Comme nous disposons certainement d'un historique des niveaux de stock du produit P_i sur un cet intervalle de temps.

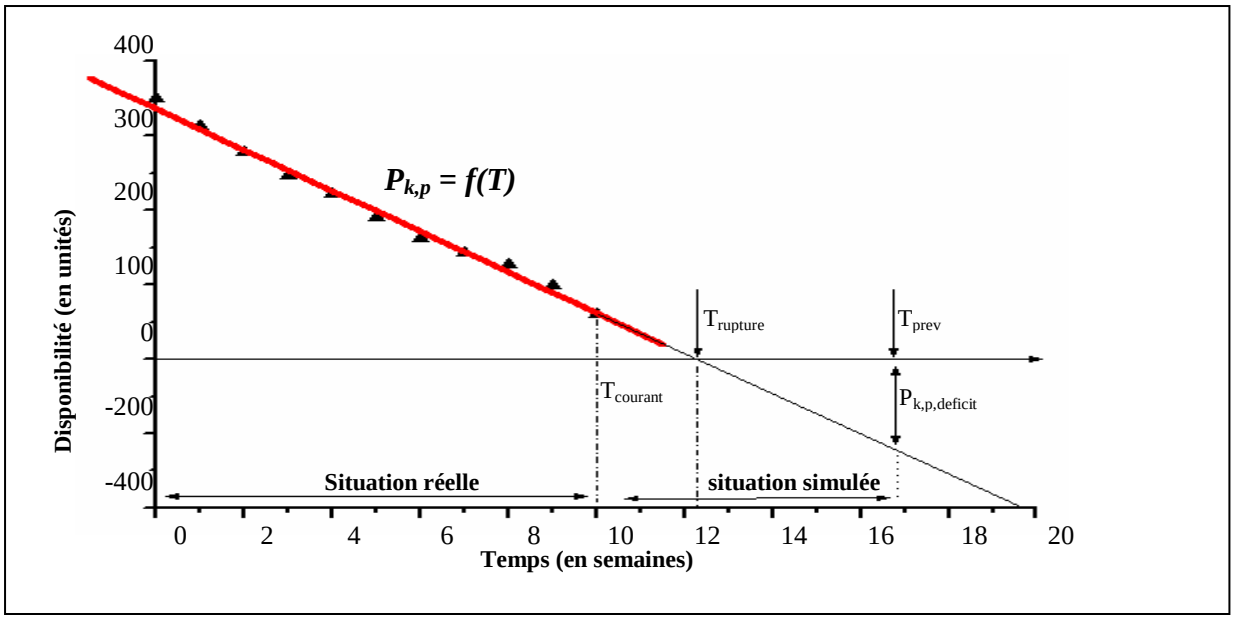


Figure 7-7 : Principe de Prévention: La rupture de Stock

Cela nous permet d'approximer la consommation en article P_k associé au Site S_p. En utilisant la méthode des moindres carrés, on obtient donc le système suivant :

$$\begin{cases} \sum_{i=1}^{n} P_i = a\sum_{i=1}^{n} T_i + nb \\ \sum_{i=1}^{n} P_i T_i = a\sum_{i=1}^{n} T_i^2 + b\sum_{i=1}^{n} T_i \end{cases}$$

C'est un système d'équations linéaires d'inconnues a et b.

La solution est immédiate, en considérant Z_1, Z_2, Z_3 et Z_4 qui simplifient les sommes, on obtient :

$$\begin{cases} z_1 = z_3 a + nb \\ z_2 = z_4 a + z_3 b \end{cases}$$

avec

$$\begin{cases} z_1 = \sum_{i=1}^{n} P_i \\ z_2 = \sum_{i=1}^{n} P_i T_i \\ z_3 = \sum_{i=1}^{n} T_i \\ z_4 = \sum_{i=1}^{n} T_{i_i}^2 \end{cases}$$

La solution est immédiate :

$$\begin{cases} a = \dfrac{z_1 z_3 - n z_2}{\Delta} \\ b = \dfrac{z_3 z_4 - z_4 z_1}{\Delta} \end{cases} avec \quad \Delta = z_3^2 - n z_4$$

Maintenant que nous connaissons l'équation de la consommation, nous pouvons approximer la quantité préventive $P_{k,p,prev}$ qui est :

$$P_{k,p,prev} = aT_{prev} + b \qquad avec \qquad T_{prev}\ donné$$

On peut détermine le niveau de disponibilité de la ressource par rapport au besoin de la tâche en voie d'exécution, qui peut être une maintenance préventive ou non et où la présence de cette ressource est indispensable, On détermine trois cas possibles

$$niv\_disp = \begin{cases} surplus & si \quad P_{k,p,prev} > 0 \\ suffisant & si \quad P_{k,p,prev} = 0 \\ déficit & si \quad P_{k,p,prev} < 0 \end{cases}$$

Par approximation, la rupture de stock arrivera donc à :

$$\boldsymbol{T_{k,p,rupture} = -b/a} \text{ quand } \boldsymbol{P_{k,p,prev} = 0}$$

L'agent prévention associé à chaque site, surveille l'état de consommation de chaque article d'une façon continue, à des périodes fixes, étudie le rythme de ces consommations sur des intervalles, à déterminer, et sur la base de la méthode d'approximation présentée, prévoit les quantités exigées à des délais calculables qui correspondent à des T_{prev} qui sont les prochains délais de réapprovisionnement, comme le montre le script suivant .

```
//Je suis l'Agent Prévention du site Sm
POUR chaque période T FAIRE
   POUR chaque pièce k de Sm FAIRE
      Calcul de TPrev()
      SI P(k;m;prev) < 0 ALORS
         Calcul de T(rupture)()
         //je veux éviter la rupture de stock avant T(rupture)
         Send (superviseurm;P(k;m;prev);demander;T(rupture))
      SINON
         SI P(k;m;prev) > 0 ALORS
            // je peux offrir à partir de T(courant)
            Send (superviseurm;P(k;m;prev);offrir;T(curant))
         FINSI
      FINSI
   FINPOUR
FINPOUR
```

Figure 7-8 : Script de l'agent Prévention

6. Etude de cas numérique

Dans ce qui suit, nous montrera, les consommations antérieures étudiées à $T_{courant}$ et respectives aux sites site1 et site2 en articles $P_{k,1,cons}$ et $P_{k,2,cons}$ de l'article k, de la classe C.

Site1				Site2			
T (sem)	$P_{k,1,cons}$ (articles)	Cumuls (articles)	$P_{k,q,disp}$ (articles)	T (sem)	$P_{k,2,cons}$ (articles)	Cumuls (articles)	$P_{k,q,disp}$ (articles)
0	0	0	100	0	0	0	100
1	15	15	85	1	2	2	98
2	4	19	81	2	3	5	95
3	12	31	69	3	5	10	90
4	4	35	65	4	6	16	84
5	8	43	57	5	6	22	78
6	7	50	50	6	11	33	67
7	4	54	46	7	5	38	62
8	6	60	40	8	4	42	58
9	10	70	30	9	4	46	54
10	5	75	25	10	6	52	48

Tableau 7-1 : la disponibilité du produit k sur deux sites

Sur la base des résultats théoriques déjà présentés, en remplaçant dans le système d'équations et en résolvant le système correspondant, on obtient les deux fonctions d'approximation de la disponibilité de l'article k pour les sites site1 et site2 sur les 10 semaines et calculé à $T_{courant}= 10^{ème}$ semaine.

Ce traitement fait parti du script décrivant le rôle des deux agents prévention, associé respectivement à chacun des sites site1 et site2, et dont les fonctions de disponibilité de l'article k sont les suivantes.

$P_{k,1,prev} = -7{,}08\ T_{k,1,prev} + 95{,}32$ pour le site1.

$P_{k,2,prev} = -5{,}64\ T_{k,2,prev} + 104{,}00$ pour le site2.

Les ruptures de stock pour le produit N° k arriveront pour les site1 et site2 à :

$T_{k,1,rupture} = 13{,}32$ quand $P_{k,1,prev} = 0$ et

$T_{k,2,rupture} = 18{,}43$ quand $P_{k,2,prev} = 0$

VII-7- Conclusion

Dans ce chapitre, nous avons posé le problème et essayer de le formuler mathématiquement. C'est un problème qui est distribué, interactif et coopératif, dont les agents, les groupes et les structures intergroupes sont autonomes. L'objectif maintenant est d'utiliser cette démarche dans les scripts des agents et spécifier le modèle d'interaction entre agents du système afin de concrétiser la modélisation du problème, de l'implémenter et le valider sur réseau

CHAPITRE VIII

IMPLEMENTATION ET MISE EN OUEVRE

Ce chapitre est destiné à décrire les différentes phases d'implémentation et de la mise en ouvre de notre modèle d'organisation et interaction, d'où nous avons utilisé l'UML pour déterminer le comportement statique et dynamique de notre modèle.

Nous présenterons les aspects de la réalisation de notre application pour savoir le choix d'ORB, ainsi que les langages et l'environnement de programmation utilisés dans l'implémentation

Mot clé :

UML, JBuilder, C++Builder, Delphi, tolerance aux panne, VisiBroker

Chapitre VIII

Implémentation et mise en ouevre

VIII-1. Introduction

Après avoir sélectionné les techniques adaptées pour traiter la situation donnée, il est nécessaire d'organiser sa mise en oeuvre pour se donner toutes les chances d'atteindre les objectifs fixés.

L'objectif de notre mémoire est de mettre en place une plate forme répartie aussi complète et aussi transparente que possible pour mieux gérer la maintenance préventive dans un environnement réparti

L'architecture de notre application est divisée en cinq agents jouant chacun un rôle bien déterminé, ces agents sont :

1. Agent gestionnaire de stock
2. Agent prévention
3. Agent superviseur.
4. Agent de coordination C.G
5. Agent de coordination C.I.G

VIII-2. Modulation par UML

UML (Unified Modeling Language) [54] est un langage standard conçu pour l'écriture de plans d'élaboration de logiciels. Il peut être utilisé pour visualiser, spécifier, construire et documenter les artefacts d'un système à fortes composantes logicielles

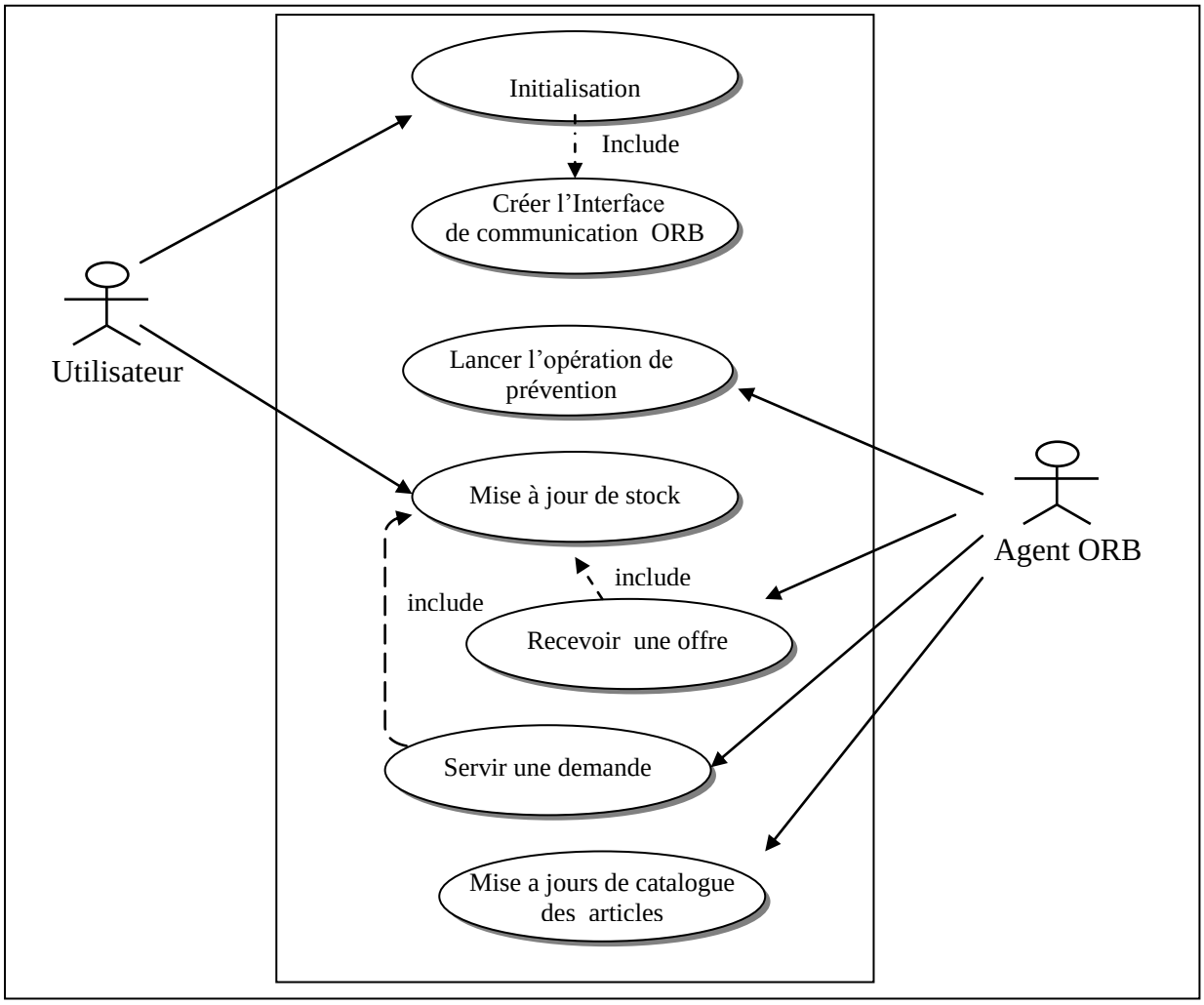
UML permet de construire plusieurs modèles d'un système, certains montrent le système du point de vue des utilisateurs, d'autres montrent sa structure interne, d'autres encore en donnent une vision globale ou détaillée. Les modèles se complètent et peuvent être assemblés. Ils sont élaborés tout au long du cycle de vie du développement d'un système

UML permet de décrire le comportement dynamique, statique et fonctionnel d'une application.

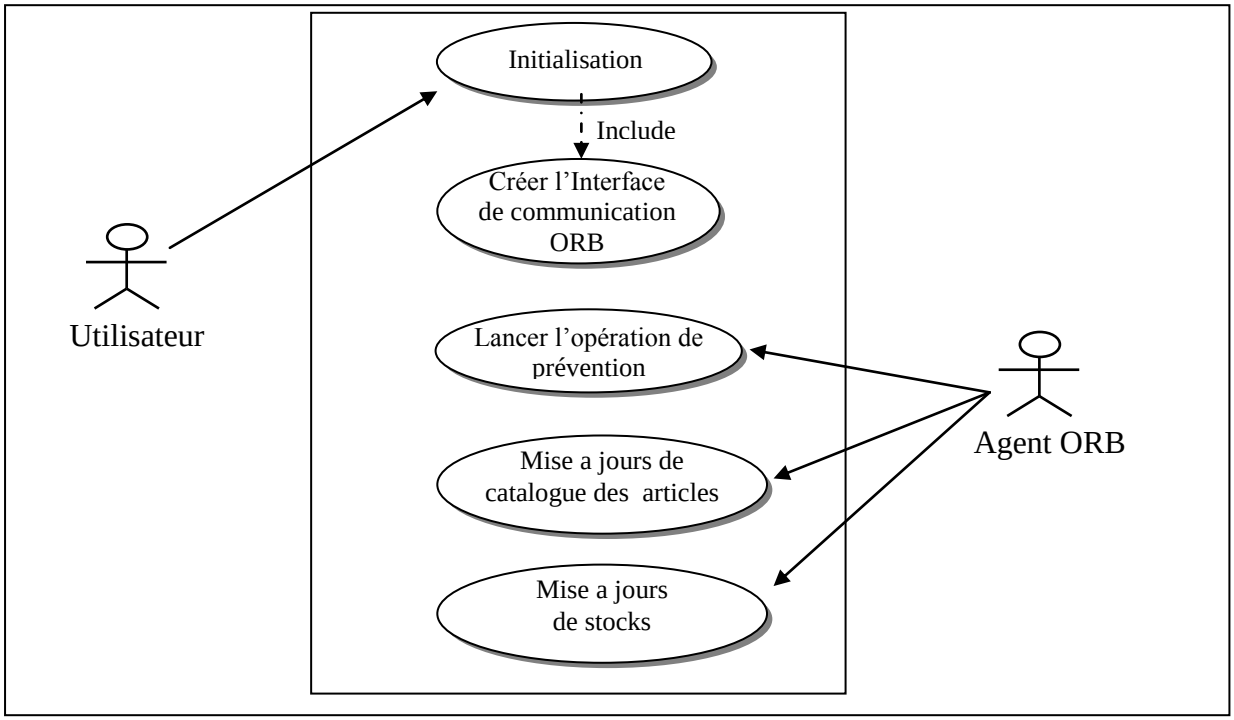
VIII-2-1. Diagramme des cas d'utilisation

Ce diagramme, décrit les fonctions du système selon le point de vue de ses utilisateurs futurs, il permet de :

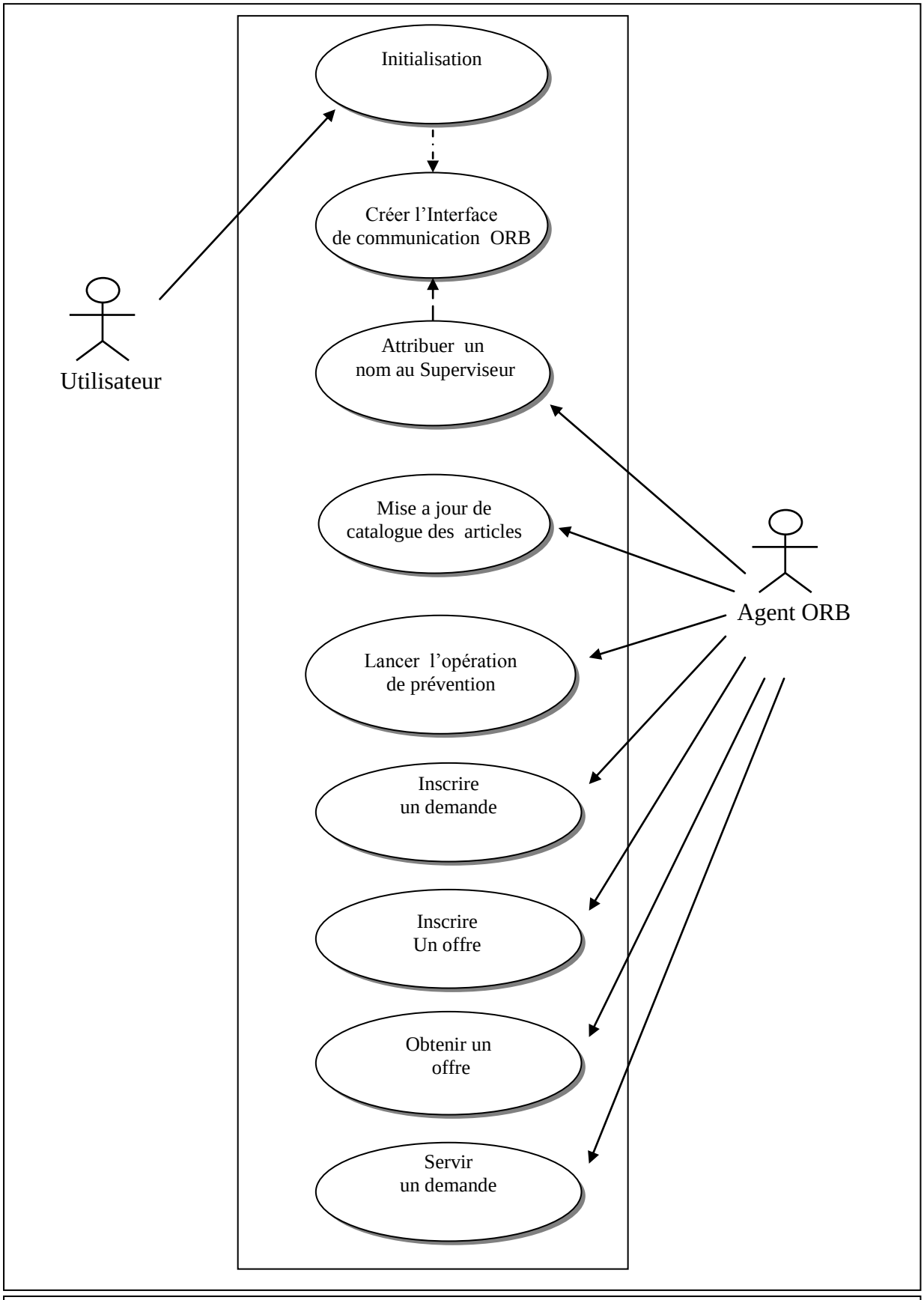
- Structurer les besoins des utilisateurs et les objectif correspondants d'un système.
- Centrer l'expression des exigences du système sur ses utilisateurs : ils partent du principe que les objectifs du système sont tous motivés.
- Identifier les utilisateurs du système (acteurs) et leur interaction avec le système, classer les acteurs et structurer les objectifs du système.



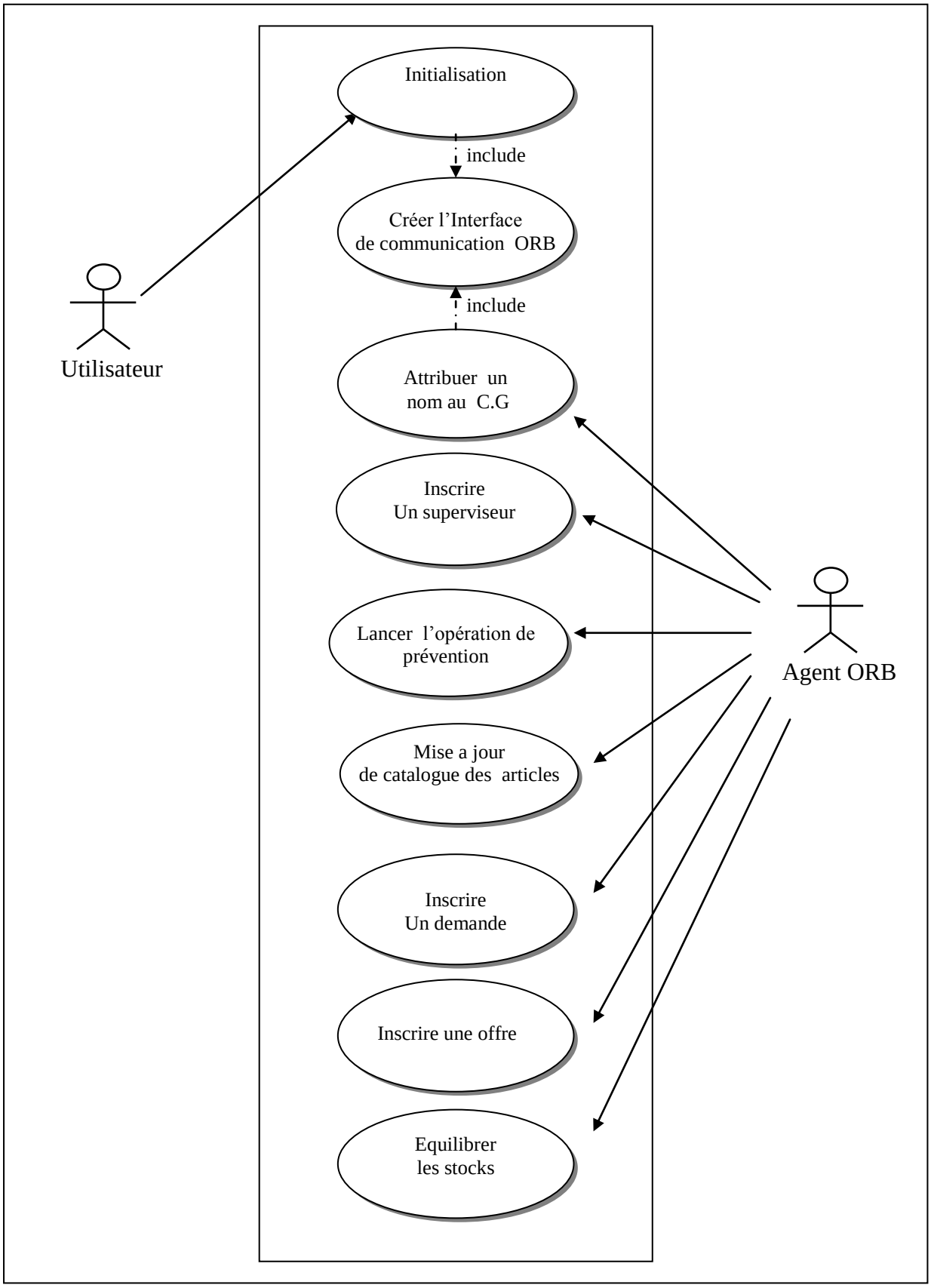
Diag 8-1 : diagramme de cas d'utilisation d'illustration pour l'agent de stock



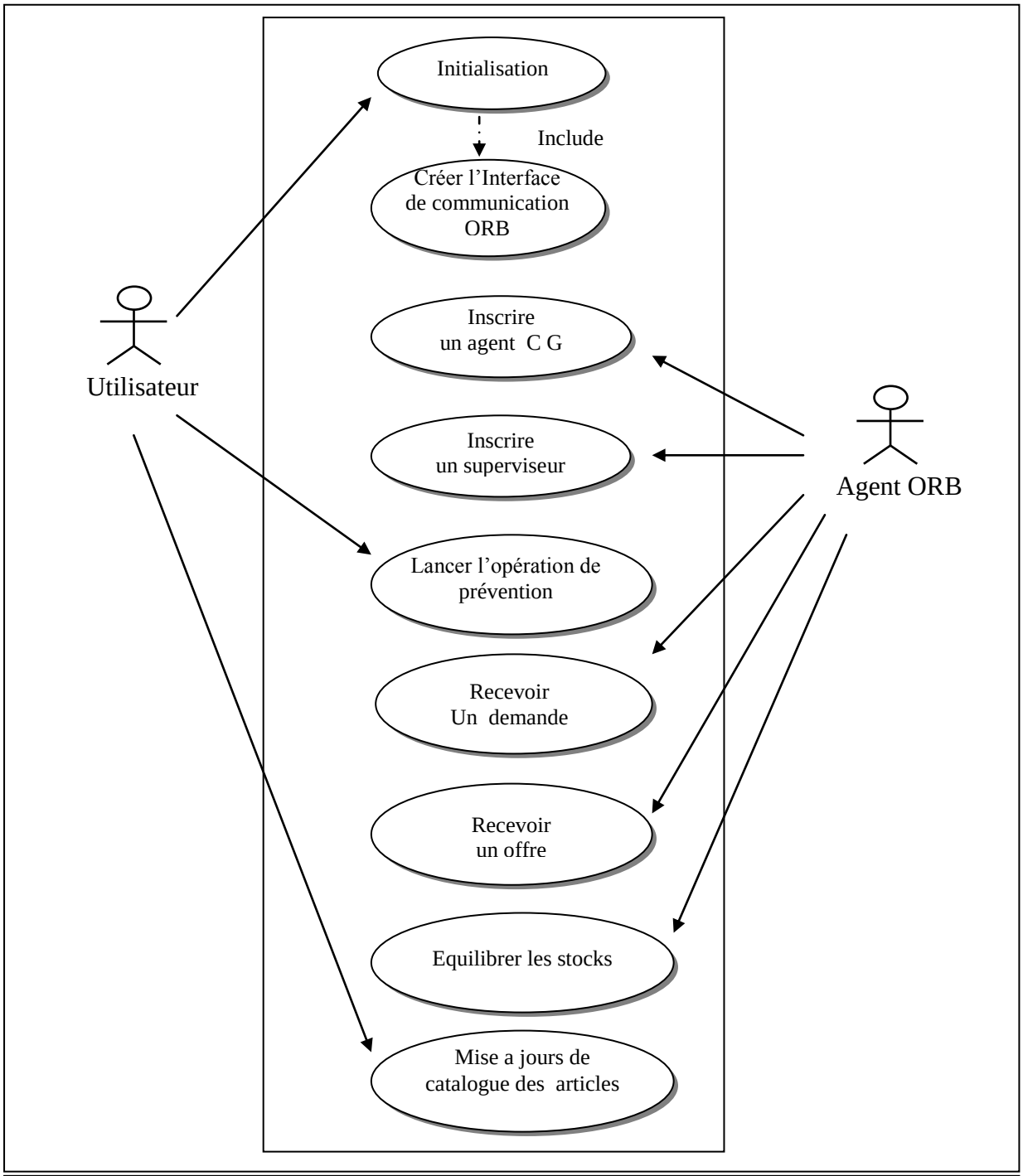
Diag 8-2 : diagramme de cas d'utilisation d'illustration pour l'agent de prévention



Diag 8-3 : diagramme de cas d'utilisation d'illustration pour l'agent superviseur



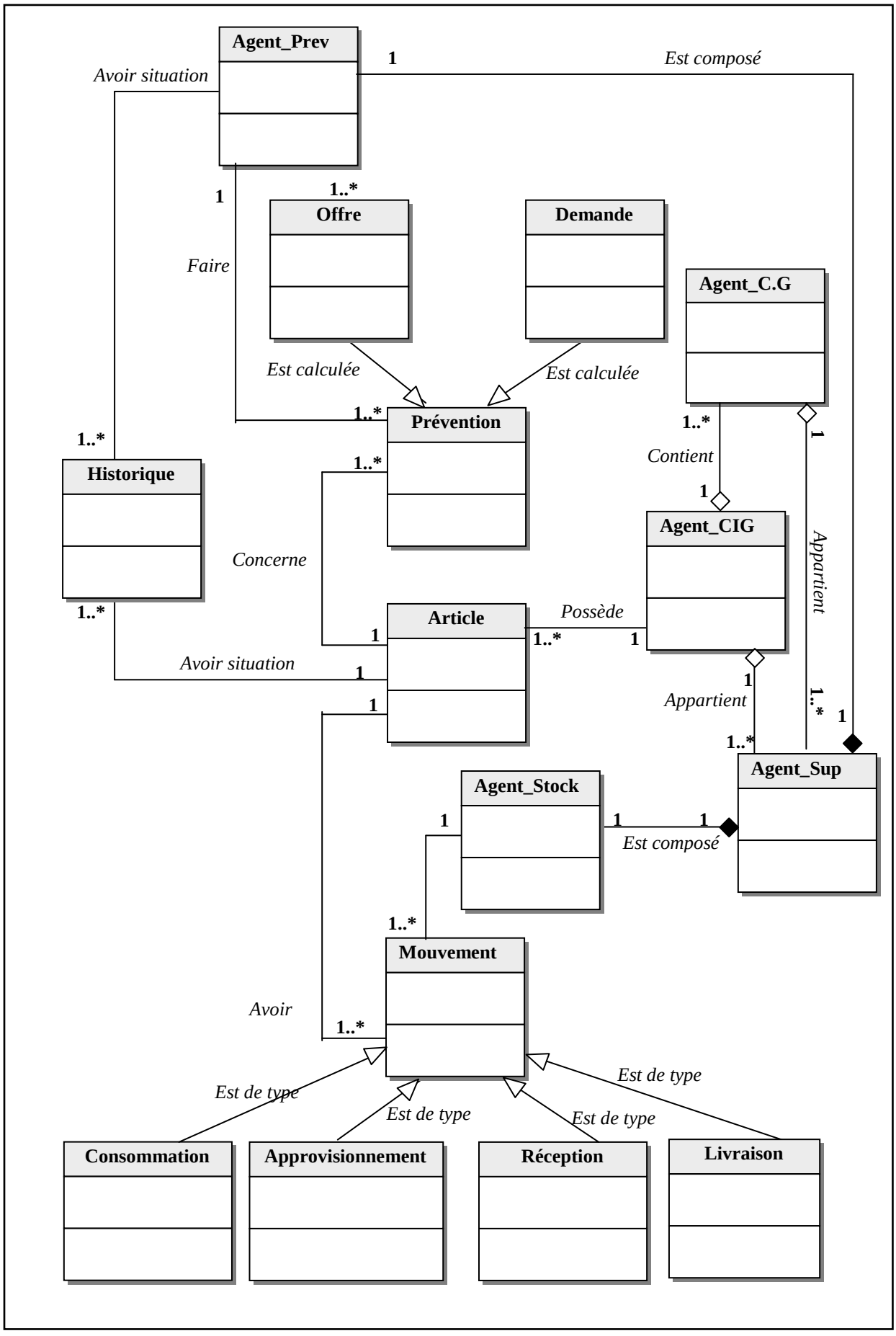
Diag 8-4 : diagramme de cas d'utilisation d'illustration pour l'agent C.G



Diag 8-5 : diagramme de cas d'utilisation d'illustration pour l'agent C.I.G

VIII-2-2. Le diagramme de classe

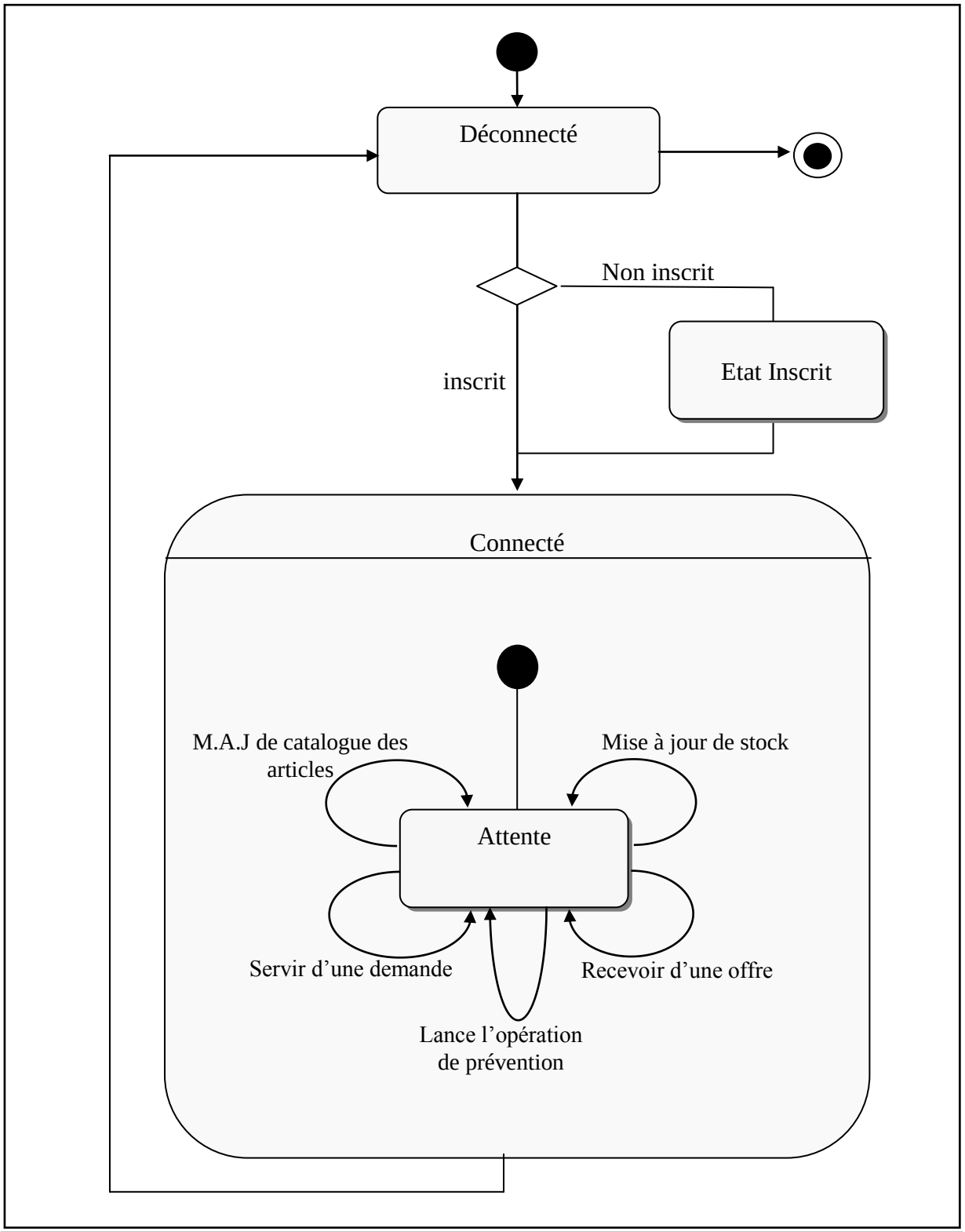
Le diagramme de classe constitue l'un des pivots essentiels de la modélisation avec UML. En effet, ce diagramme permet de donner la représentation statique du système à développer. Cette représentation est centrée sur les concepts de classe et d'association



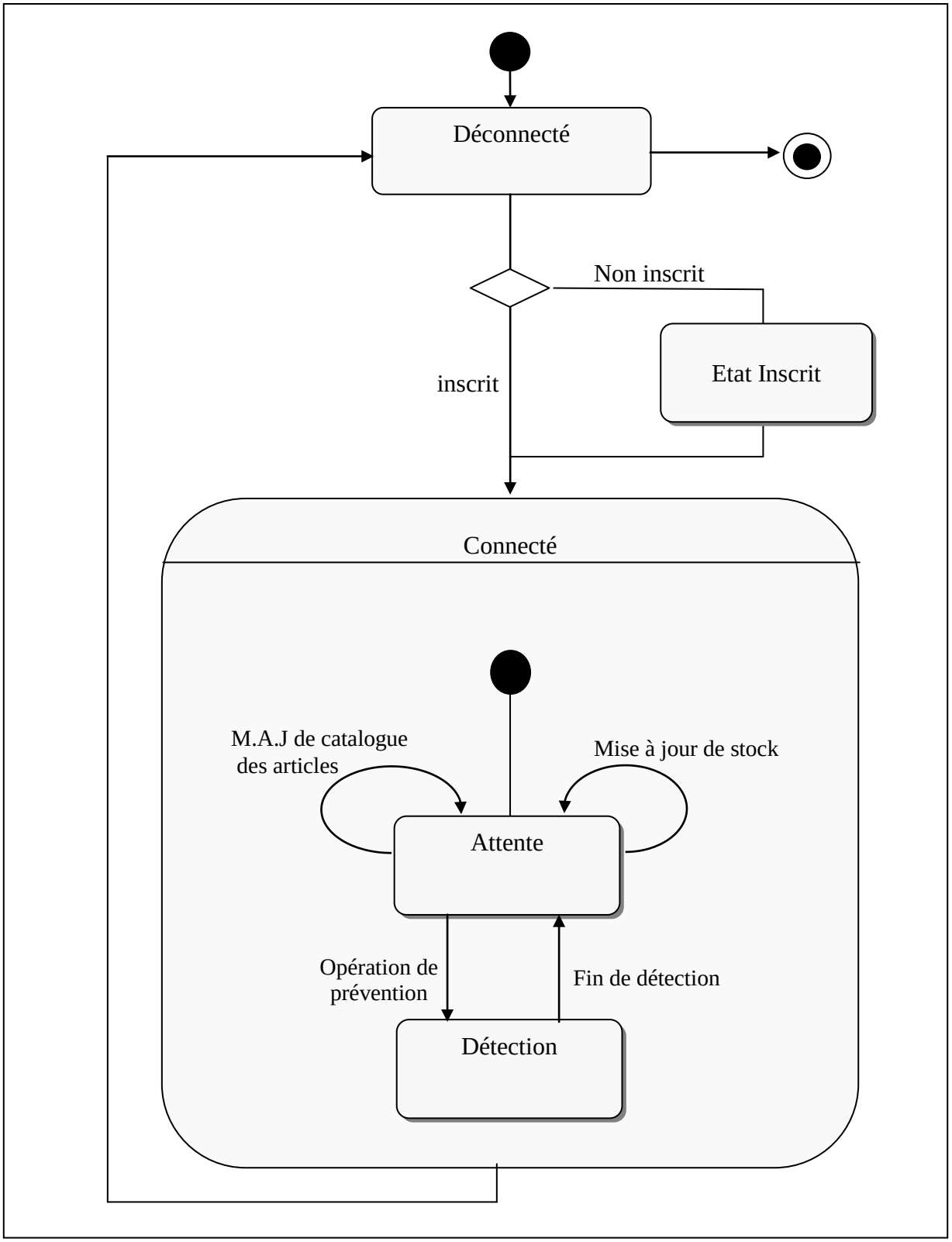
Diag **8.6 : diagramme de classes associations**

VIII-2-3. Diagramme d'état transition

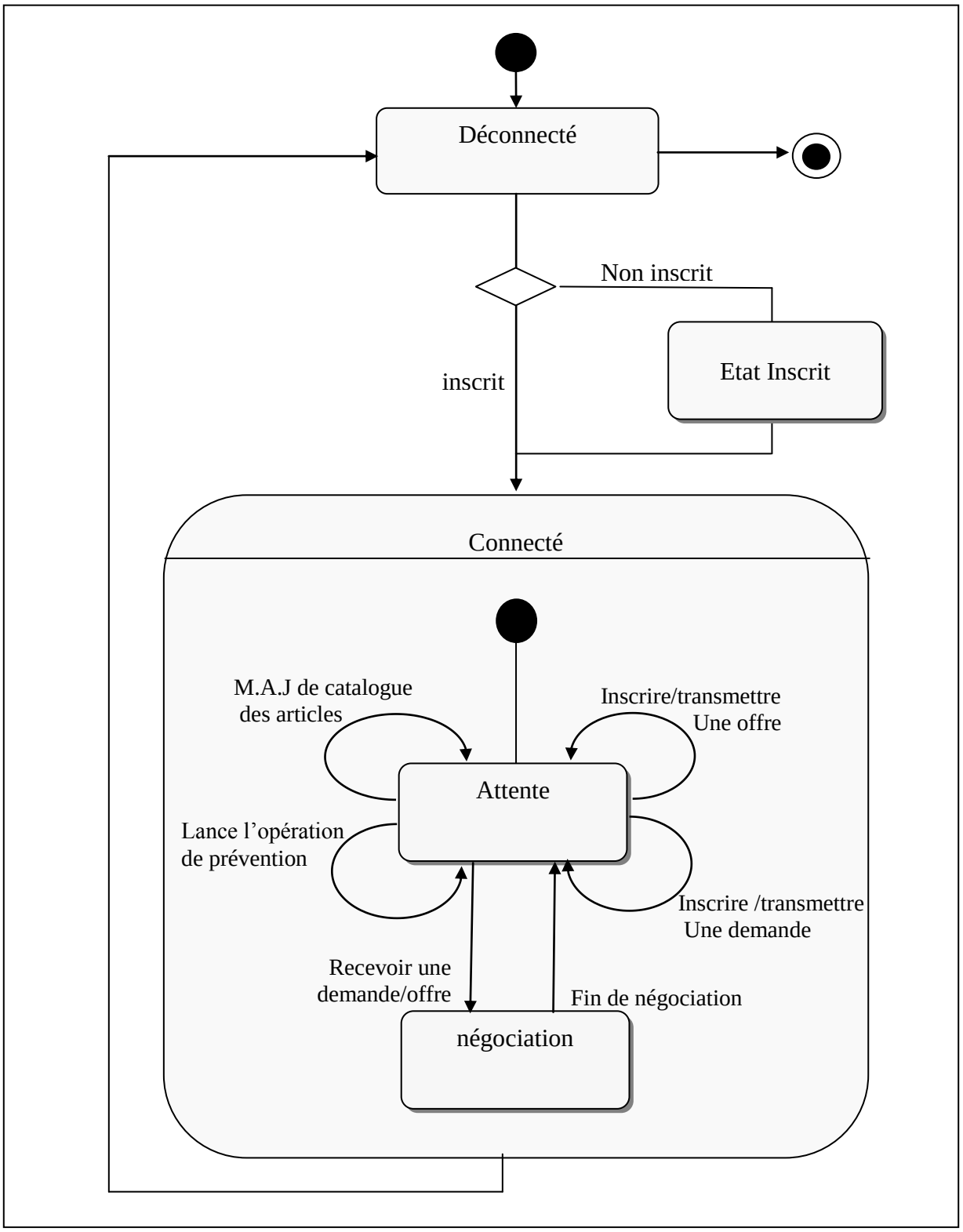
Les diagrammes d'états-transitions d'UML décrivent le comportement interne d'un objet à l'aide d'un automate à états finis. Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets.



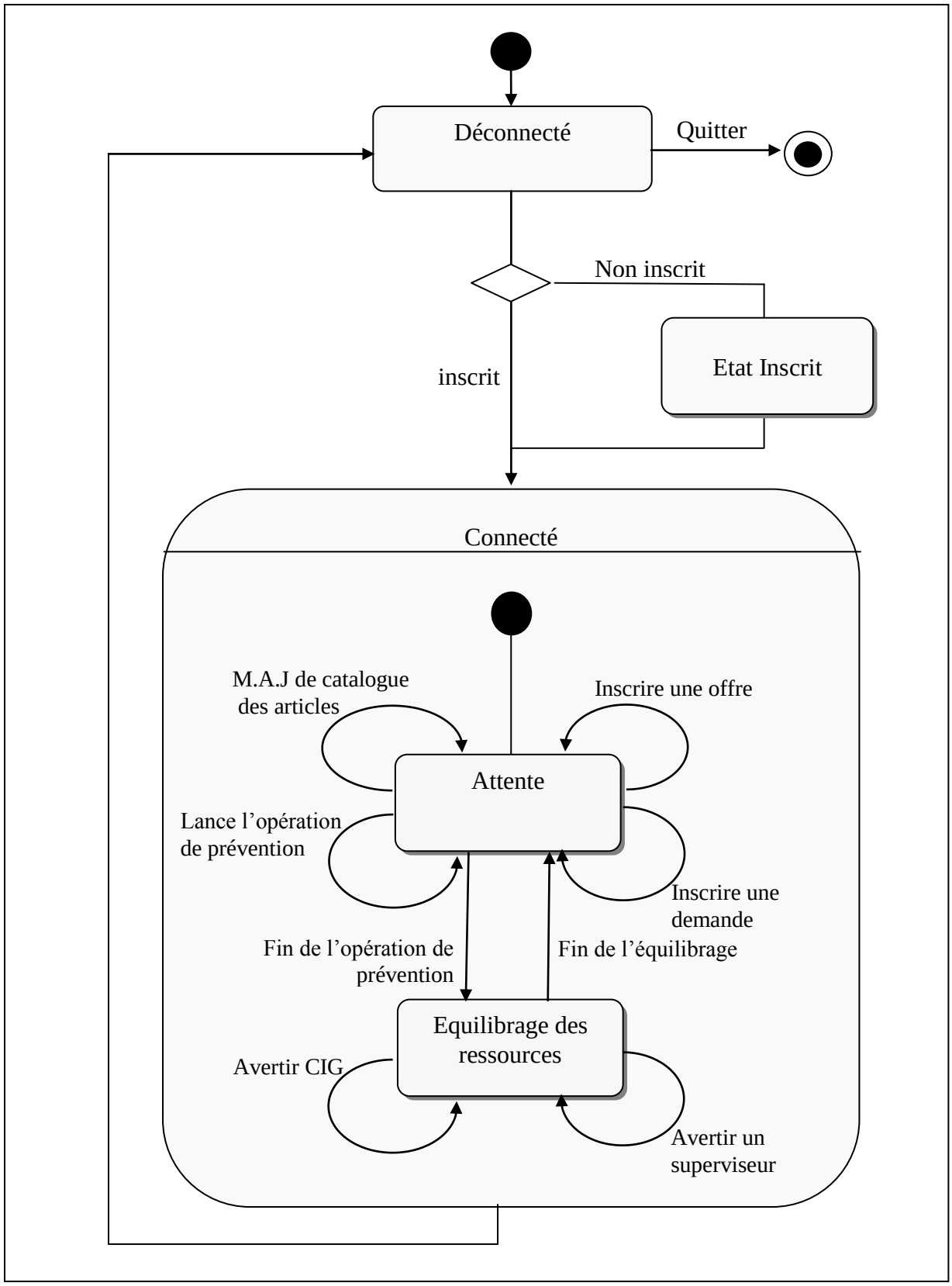
Diag 8.7:diagramme d'état transition de l'agent gestionnaire de stock



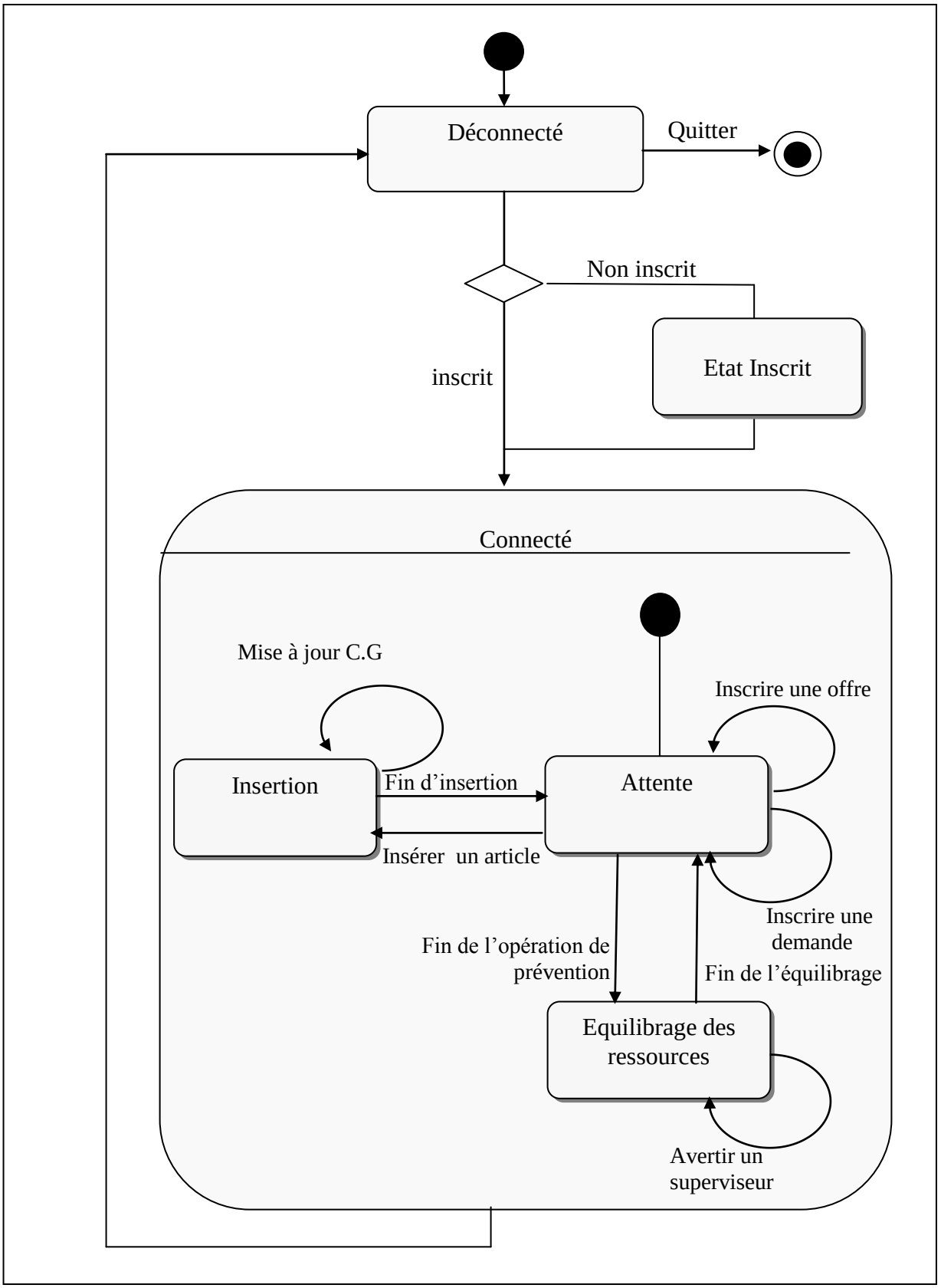
Diag 8.8:diagramme d'état transition illustrant le comportement de l'agent de prévention



Diag 8.9:diagramme d'état transition de l'agent de superviseur



Diag 8.10: diagramme d'état transition de l'agent C G

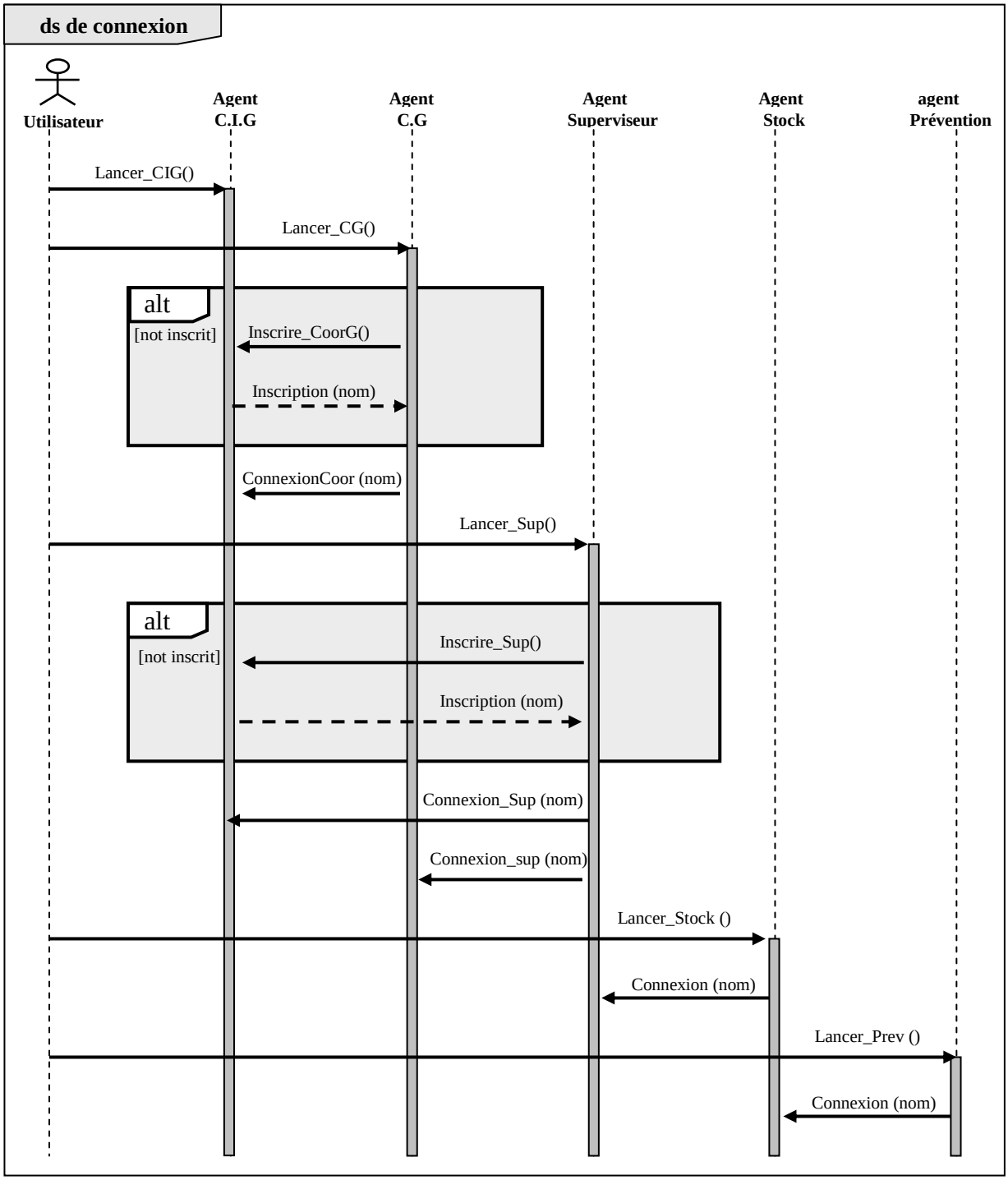


Diag 8.11: diagramme d'état transition de l'agent C I G

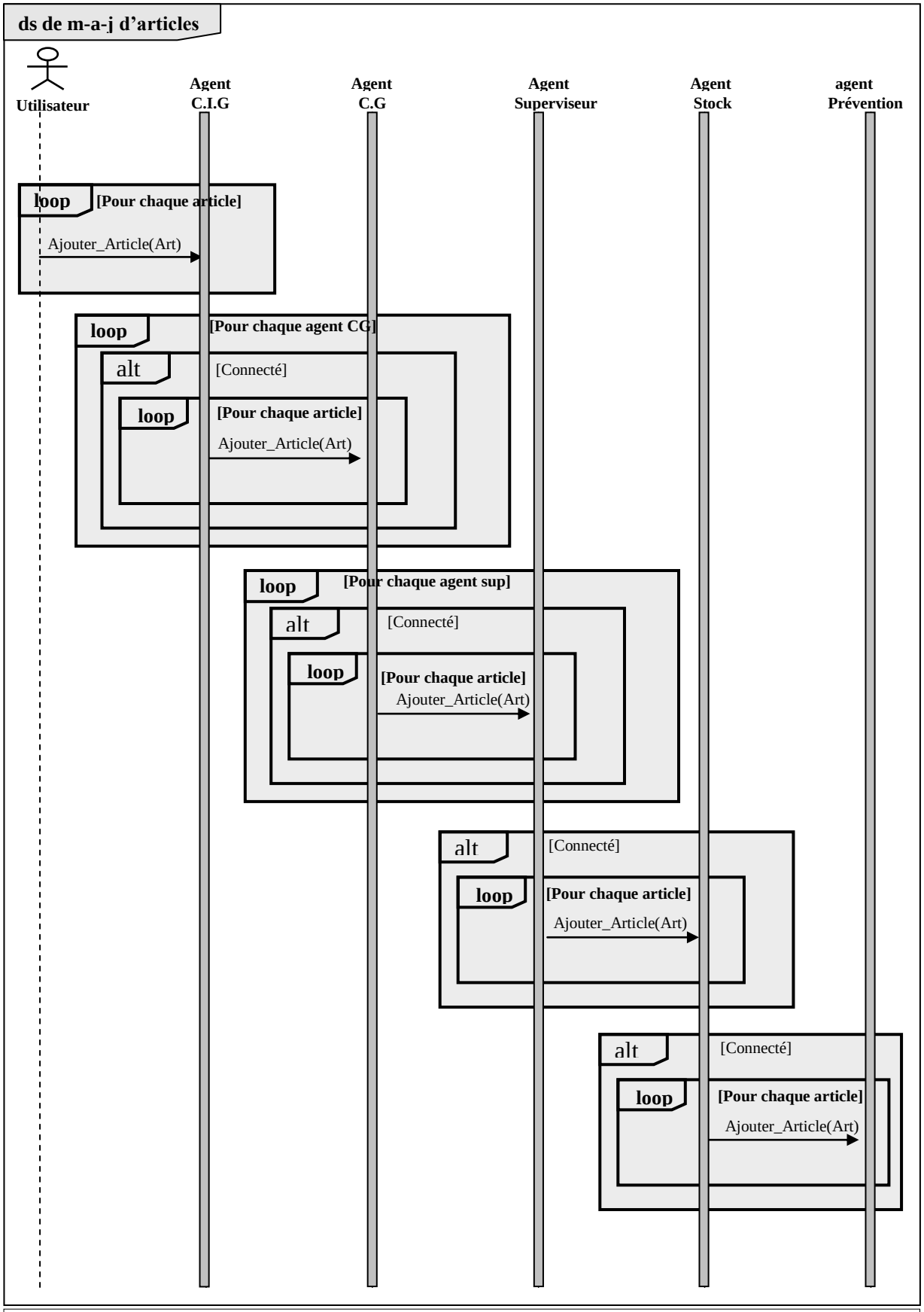
VIII-2-4. Diagramme de séquence :

L'objectif du diagramme de séquence est de représenter les interactions entre objets en indiquant la chronologie des échanges. Cette représentation peut se réaliser par cas d'utilisation en considérant les différents scénarios associés

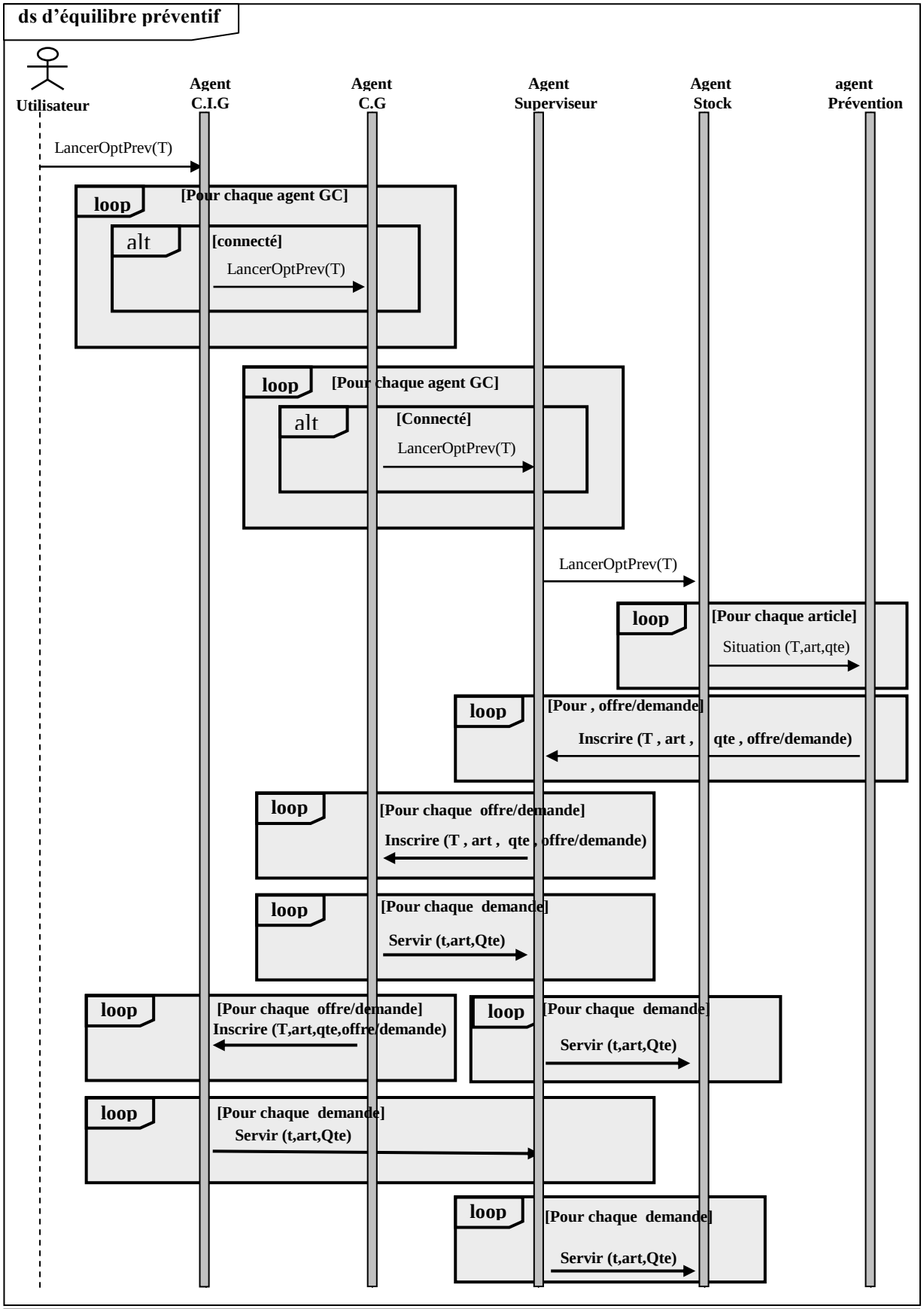
Les diagrammes de séquence représentent des interactions entre des lignes de vie. Un diagramme de séquence montre des interactions sous un angle temporel, et plus particulièrement le séquencement temporel des messages échangés entre des lignes de vie.



Diag 8.12: diagramme de séquence de connexion



Diag 8.13: diagramme de séquence de mise ajour de catalogue des articles



Diag 8.14: diagramme de séquence d'équilibre préventif

VIII-3. L'interface de coordination

Apres avoir identifié les fonctions invocables à distance, nous représentons dans cette section les interface IDL permettant de développer ces fonctions pour qu'elles soient utilisables dans un environnement réparti comme suivant :

1- Agent coordinateur intergroupe :

```
module Agent_CIG
{
 struct organisme
 {
   string nom_coordinateur ;
   string nom_superviseur  ;
 };
 interface Tache_Agent_CIG
 {
   string    inscrire_coordinateur();
   organisme inscrire_superviseur();
   void connexion_coordinateur (in string nom_coordinateur, in long nbre_article ,in long etat);
   void connexion_superviseur(in string nom_superviseur  , in long nbre_article , in long etat );
   void inscrire_offre(in string sup ,in string code_article ,in long qte );
   void inscrire_demande(in string sup , in string code_article , in long qte);
 } ;
};
```

List 8-1 : Interface IDL Pour L'agent Coordinateur inter-groupe

2- Agent coordinateur de groupe :

```
module Agent_CG
{
  interface Tache_Agent_CG
  {
     void inscrire_superviseur(in string nom_superviseur);
     void connexion_superviseur(in string nom_superviseur, in long nbre_article ,in long etat );
     void inscrire_article(in string code_article, in string designation ) ;
     void inscrire_coordinateur(in string nom_coordinateur);
     void inscrire_offre(in string sup ,in string code_article ,in long qte );
     void inscrire_demande(in string sup , in string code_article , in long qte);
  } ;
};
```

List 8-2 : Interface IDL Pour L'agent Coordinateur de groupe

3- Agent superviseur :

```
module Agent_Superviseur
{
  interface taches_agent_superviseur
  {
      void inscrire_article(in string code_article , in string designation );
      void inscrire_offre(in string code_article ,in long qte, in long periode );
      void inscrire_demande(in string code_article , in long qte , in long periode);
      long servir_demande(in string code_article , in string nom_superviseur , in long qte );
      void recevoir_offre(in string code_article , in string nom_supriveseur , in long qte ) ;
      void operation_preventive(in long t);
  };
};
```

List 8-3 : Interface IDL Pour L'agent superviseur

4- Agent de stock :

```
module agent_stock
{
  struct article_structure
  {
    string code_article ;
    string designation  ;
  };
  typedef sequence <article_structure> Tableau ;
  interface tache_agent_stock
  {
      void inscrire_article(in Tableau tableau_article,in long t ) ;
      void inscrire_offre(in long periode ,in string code_article,in long qte , in string nom_superviseur );
      void inscrire_demande(in long periode ,in string code_article,in long qte,in string nom_superviseur);
      void operation_preventive(in long t );
  };
};
```

List 8-4 : Interface IDL Pour L'agent de stock

5- Agent de prévention :

```
module agent_prevention
{
  struct article_structure
  {
    string code_article ;
    string designation ;
    long qte_stock ;
  };
  typedef sequence <article_structure> Tableau ;
  interface tache_agent_prevention
  {
    void sitaution_stock(in Tableau tableau_article,in long nbre_article, in long t );
  };
};
```

List 8-5 : Interface IDL Pour L'agent de prévention

VIII-4- Cas d'exception et Stratégies de tolérance de panne :

La tolérance de panne fait référence à la capacité d'un système à continuer à fonctionner lorsqu'une partie de celui-ci tombe en panne. Et la capacité de récupération se réfère à la capacité à restaurer le système au point duquel un échec est survenu

Pour créer un système à tolérance de panne, nous utilisons des mesures préventives pour réduire au minimum l'éventualité d'une panne du système, nous avons trois niveaux de panne qui sont localisées dans le système

VIII-4-1. Panne au niveau d'un site :

En cas de panne au niveau de site, soit par la défaillance de :

- l'agent superviseur
- l'agent de stock

- l'agent de prévention.
- Le lien entre le superviseur et le coordinateur de groupe

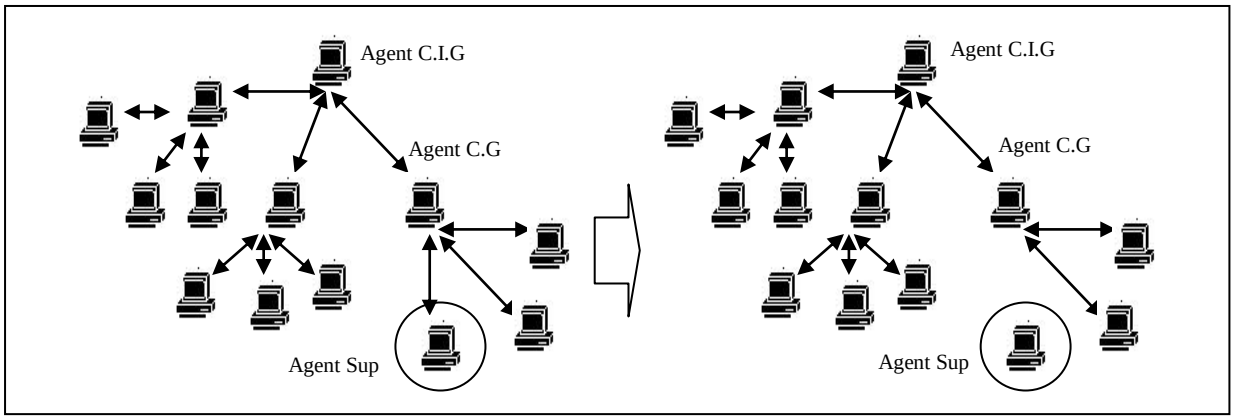


Figure 8 -1 : Cas de défaillance de site

Dans ce cas le système continue à fonctionner sans aucune intervention d'autres composants du système, la panne sera isolée au niveau de ce site.

Le coordinateur vérifie la présence de ce site périodique, s'il détecte une défaillance, l'état de superviseur de ce site devient déconnecté.

La reprise et l'intégration de site sera prise automatique dans la prochaine période de calcul préventif

VIII-4-2. Panne au niveau d'un Coordinateur de groupe :

Dans ce cas le système doit avoir une nouvelle hiérarchique, par la réinscription de tous les superviseurs liés à ce coordinateur de groupe.

Tous les superviseurs seront orientés vers un autre coordinateur soit :

a- par l'intervention du coordinateur intergroupes, celui-ci oriente les superviseurs dans le cas ou il y'a une période de calcule préventif, si les superviseurs sont connectés sans la connexion de leur coordinateur.

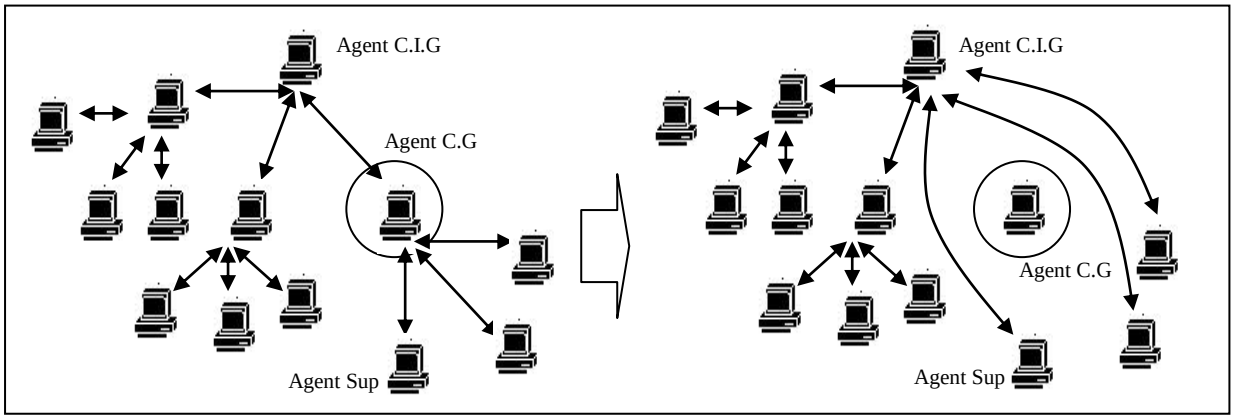


Figure 8 -2 : Cas de défaillance de l'agent CG et l'orientation vers l'agent CIG

Voici la nouvelle interface modifiée concernant le superviseur

```
module Agent_Superviseur
{
  interface taches_agent_superviseur
  {
    ……..
    void     Reinscrire_superviseur(in string nom_coordinateur) ;
  } ;
} ;
```

List 8-6 : **l'interface superviseur modifier en cas de défaillance de C.G**

b- Par l'intervention de superviseur lui-même s'il détecte que leur coordinateur de groupe n'est connecté, dans ce cas il faut faire une autre réinscription au coordinateur intergroupe

La reprise et l'intégration de coordinateur de groupe seront prises automatiquement dans la prochaine connexion de ce dernier et de leurs superviseurs.

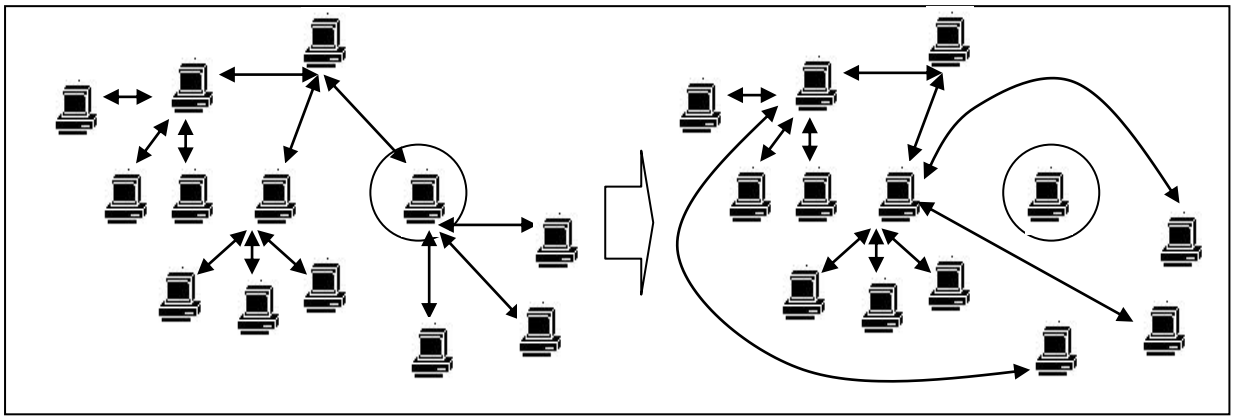


Figure 8 -3 : Cas de défaillance de l'agent CG et l'orientation vers l'agent CG

L'interface concerné par cette intervention de superviseur est comme suit :

```
module Agent_CIG
{
 interface Tache_Agent_CIG
 {
    ……..
   string    Reincrire_superviseur(in string nom_superviseur ) ;
  } ;
};
```

List 8-7 : **l'interface de C.I.G modifier en cas de défaillance de C.G**

VIII-4- 3. Panne au niveau de coordinateur intergroupe

Il est possible de conférer à notre système une certaine tolérance de panne en déployant des configurations supplémentaires qui dupliquent la configuration existante, notre système peut continuer à fonctionner à l'aide du coordinateur dupliqué

Dans ce cas nous avons installé un autre coordinateur intergroupe, il surveille le coordinateur intergroupe initial, s'il détecte l'absence de ce dernier, il lance un objet serveur portent le même nom.

Nous pouvons utiliser l'envoi périodique des mises à jours depuis le coordinateur intergroupe initial vers l'autre coordinateur intergroupe dupliqué, de manière continue. La sauvegarde permet de conserver la base de données de destination synchronisée avec la base de données source et le coordinateur dupliqué à jour.

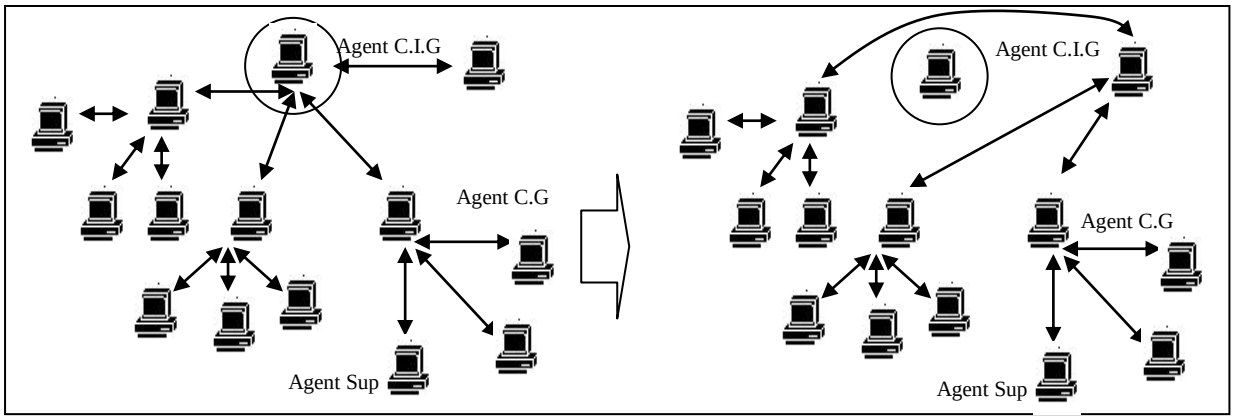


Figure 8 -4 : Cas de défaillance de l'agent CIG et l'orientation vers l'agent CIG réserve

Voici l'interface modifiée de l'agent coordinateur intergroupe, qui possède deux interfaces, une pour créer un objet serveur, et l'autre objet de contrôle de défaillance et mise a jour des données de duplication.

```
module Agent_CIG
{
 interface Tache_Agent_CIG
 {      ……..   } ;
 interface Duplique_Agent_CIG
 {
   void inscrire_coordinateur(in string coordinateur ,in long nbre_site );
   void inscrire_ superviseur(in string superviseur , in string coordinateur);
   void inscrire_article(in string code_article, in string designation );
   void operation_preventive(in long t );
   void Detection_presence() ;
 } ;
};
```

List 8-8 : Interface IDL modifiée Pour L'agent Coordinateur inter-groupe

VIII-5. Implémentation

Dans ce qui suit nous présentons les aspects de la réalisation de notre application à savoir le choix d'ORB, ainsi que les langages et l'environnement de programmation utilisés dans l'implémentation, Nous donnons un exemple pour suivre et expliquer le cycle de développement d'une application CORBA étape par étape. En plus, nous présentons quelques fenêtres concernant notre application.

VIII-5-1. Langage de programmation

Pour profiter de l'interopérabilité de CORBA, et la gratuité de VisiBroker intégré dans les outils de développement de la société Inprise [inprise 98], nous avons choisi de développer par les environnements de programmation

- JBuilder

- Delphi
- C++Builder

- JBuilder est un environnement de développement intégré pour Java, permettant le RAD, et édité par Borland. L'application est elle-même développée en grande partie en Java. JBuilder apporte certaines fonctionnalités spécifiques, disposant notamment d'une JVM propre, permettant notamment l'exécution de code Java pas à pas.

Delphi désigne à la fois un environnement de développement intégré (EDI) et implémente une version orientée objet (POO) du langage Pascal

C++Builder est un RAD conçu par Borland qui reprend les mêmes concepts, la même interface et la même bibliothèque que Delphi en utilisant le langage C++. Il permet de créer rapidement des applications Win32 ainsi qu'une interface graphique avec son éditeur de ressources. Il est compatible avec la norme ISO C++.

VIII-5-2. Choix d'ORB

Pour réaliser une application repartie avec la Norme CORBA, le premier pas à franchir est celui du choix d'ORB utilisé. Comme on vu au chapitre 2, il existe plusieurs dizaines d'ORB sur le marché CORBA, chacun respecte la norme CORBA mais à des degrés divers. Notre choix s'effectue sur le VisiBroker qui sera présenté dans ce qui suit.

VIII-5-3. Qu'est ce que le VisiBroker ?

VisiBorker est un ORB commercial de la société Inprise (Inprise). C'est l'un des ORBs les plus répandus sur le marché CORBA.

Les points prioritaires pour l'évolution et la promotion de VisiBroker sont :

- un meilleur respect des spécifications CORBA.
- une intégration de tout les éléments que nous avons présentés au chapitre 2 : protocole IIOP, compilateur IDL, BOA, etc.
- une intégration des outils capables de gérer la tolérance de panne, la répartition de charge, la translation, la sécurité.
- le développement : il s'agit des environnements Inprise tels que ceux de Delphi, de C++ Builder ou de JBuilder ainsi que d'un ensemble d'outils. Signalons que ces environnements génèrent automatiquement les interfaces d'accès aux objets distants, quelle que soit la plate-forme qui l'héberge.

VIII-5-4. Qu'est ce que le Smart Agent ?

Le Smart Agent est un service de répertoire dynamique et distribué qui offre des facilités aussi bien pour les applications clients que pour les implémentations d'objets. Quand un client invoque la méthode bind (bind n'est pas une méthode définie par la norme CORBA, c'est un ajout propriétaire à chaque ORB) pour se connecter à un objet serveur, Smart Agent localise l'implémentation de l'objet serveur et établit une connexion entre le client et cette l'implémentation. Les l'implémentations d'objets enregistrent leurs objets auprès de Smart Agent. Pour qu'ils puissent être localisés par les applications clients.

Quand un objet est supprimé, Smart Agent l'enlève de sa liste d'objets disponibles.

VIII-5-5. Relation entre client, serveur et Smart Agent :

1- L'enregistrement de serveur dans la base interne d'objet de l'agent.
2- La demande du client pour un objet particulier.
3- La réponse de l'agent au client, à savoir la référence à l'objet distant.
4- L'exploitation de l'objet par un dialogue directe client/serveur.

La figure suivante montre la relation entre le client, serveur et Smart Agent lors du lancement d'une application :

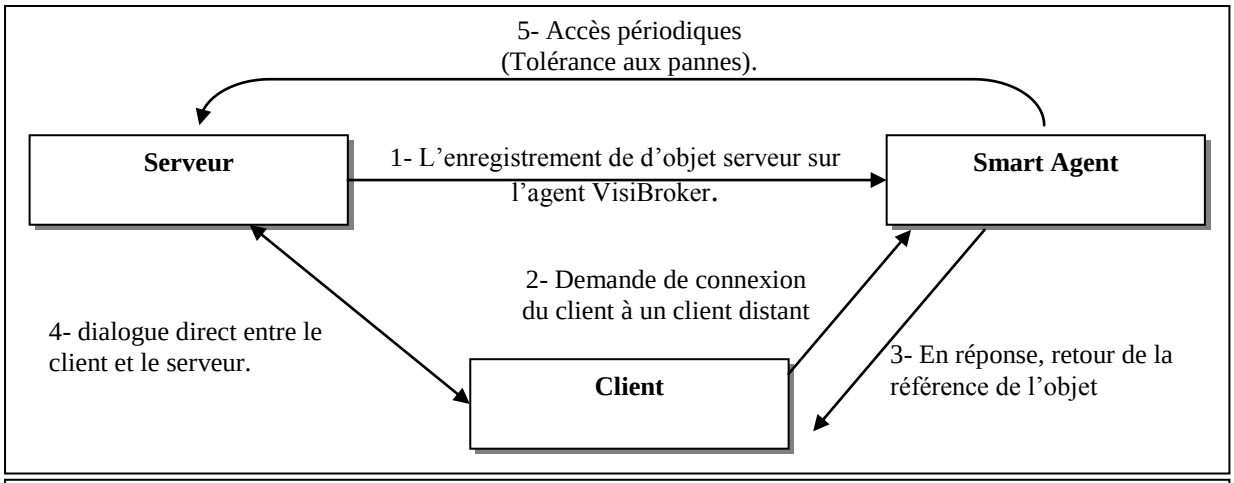


Figure 8 -5 : Relation entre client, serveur et smart Agent

VIII-5-6. Cycle de développement CORBA :

Nous allons maintenant illustrer les démarches de création d'une application CORBA à travers le cas de l'agent coordinateur intergroupe, et l'environnement de développement C++ Builder.

1. Ecriture de l'interface en IDL

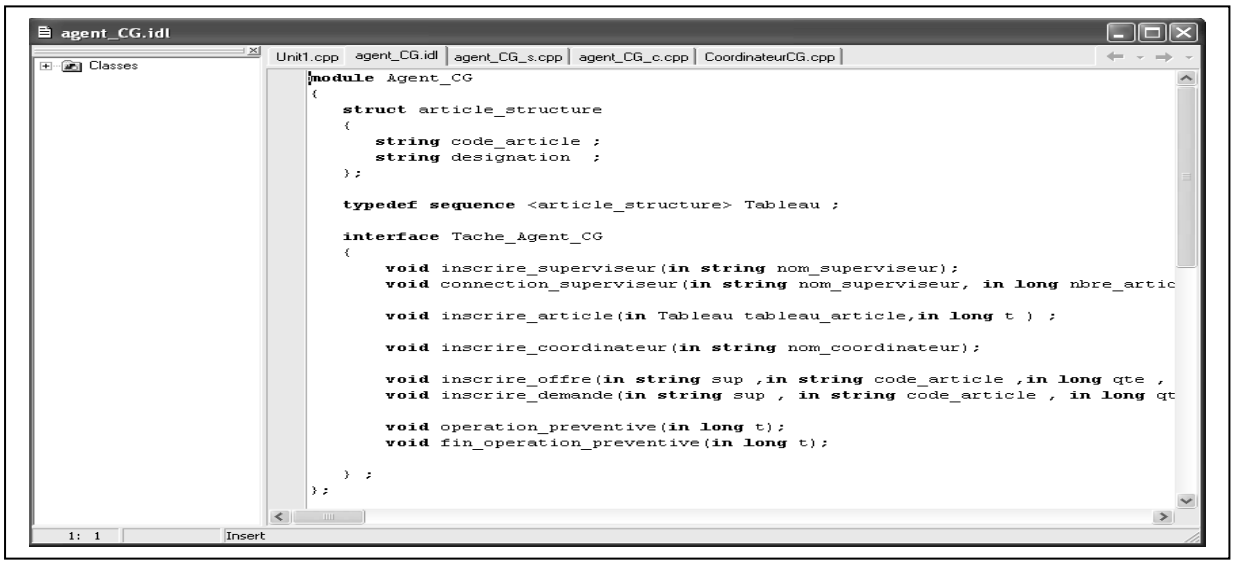


Figure 8.6 : l'interface IDL de l'agent Coordinateur de groupe

2. Compilation de l'IDL, génération des modules Stub et Skeleton

Une fois compilé, l'IDL donne naissance à deux modules: un module Stub, utilisé dans la partie cliente et un module Skeleton utilisé dans la partie serveur

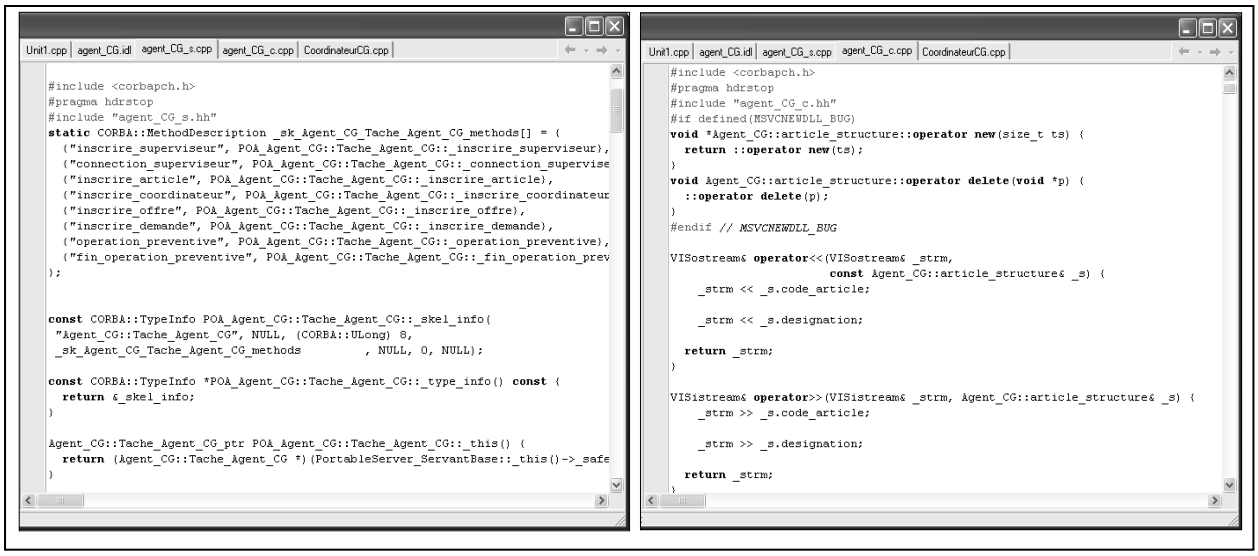


Figure 8.7 : Génération du Stub et skeleton

3. l'implémentation de l'objet Coté serveur :

Pour implémenter l'objet qui nous donne un accès à ces fonctions distantes, il faut dériver des classes à partir du skeleton généré lors de la deuxième étape, pour cela nous avons utilisé l'expert d'implémentation d'objet CORBA de C++ Builder

Pour chacune des méthodes virtuelles pures définies dans le skeleton, des méthodes vides ont été produites. Nous devrons maintenant définir les traitements de chaque objet

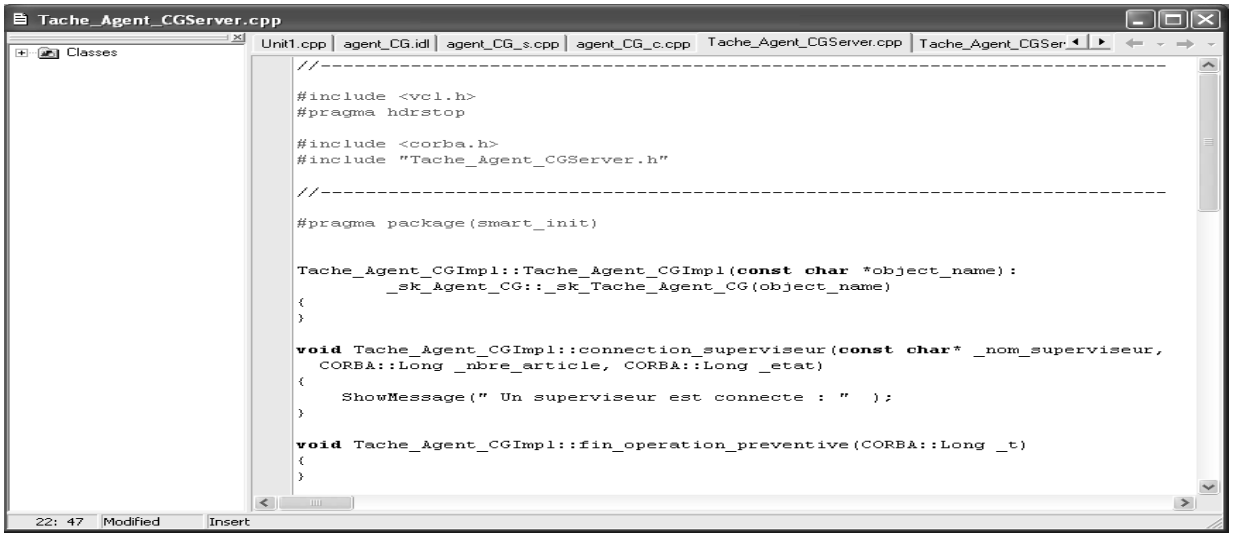


Figure 8.8 : Implémentation de la méthode connexion superviseur

4. Réalisation du serveur CORBA

Cette étape permet de mettre les objets serveurs à disposition des applications clientes, nous définissons alors, un code qui leur permettons d'écouter tout les appels clients.

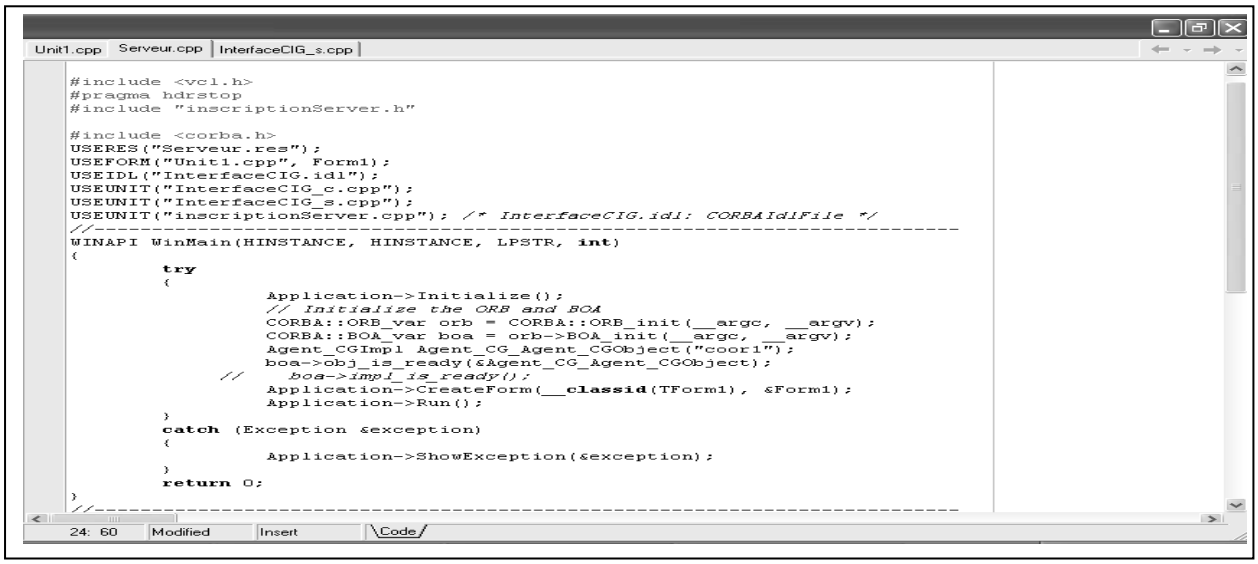


Figure 8.9 : déclaration et initialisation d'objet serveur

5. Réalisation du client

Pour pouvoir utiliser les fonctions distantes, il faut d'abord se connecter à l'objet qui les enveloppe.

L'utilisation de ces fonctions sera guidée par le contexte de leurs appels.

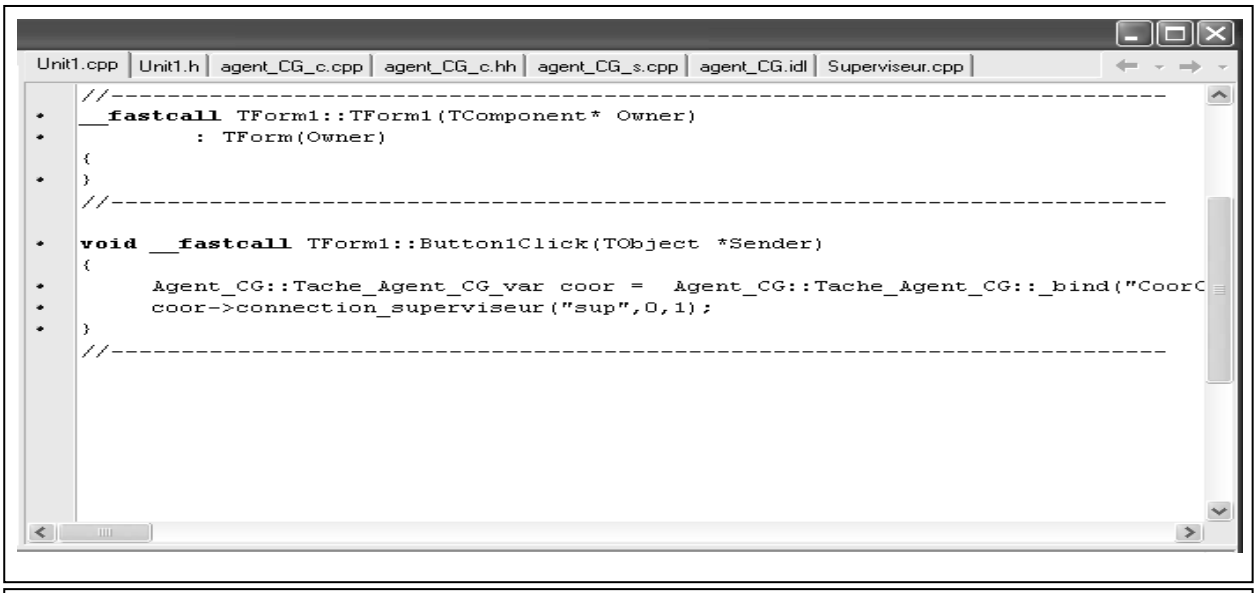


Figure 8.10 : exploiter l'objet serveur coté client

6. présentation de quelques fenêtres de notre logiciel :

Agent gestionnaire de stock :

Cet agent situé dans le site, Il est capable de stocker des informations concernant la mise à jours de stock de pièces de rechange suivant la traduction des messages reçus, et d'agir Comme un agent de ressource standards dépourvu de toute intelligence.

Il est doté d'une base de données comportent les tables suivantes :

- Une table d'article sert à donner la situation à jours de stock
- Une table des entrées sert à inscrire les offres obtenues servies par d'autres sites.
- Une table des sorties sert à inscrire les demandes servies aux d'autres sites.

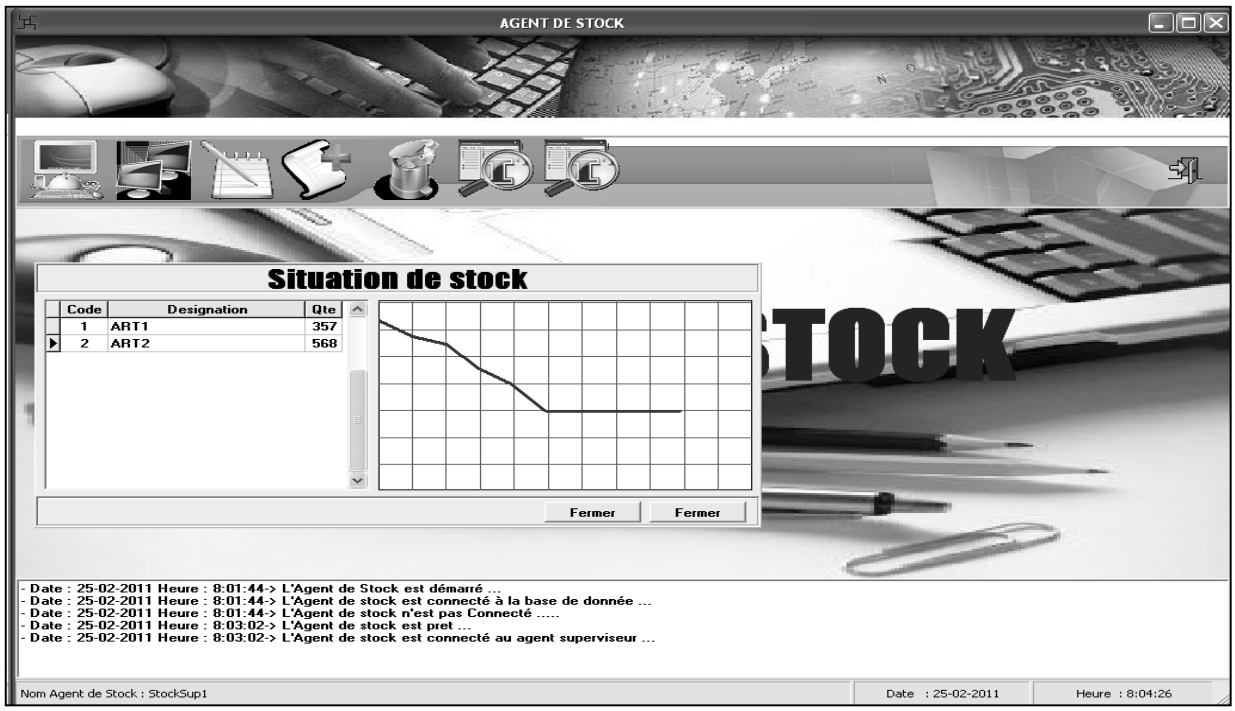


Figure 8.11 : Interface graphique d'agent gestionnaire de stock

Agent prévention :

C'est un agent capable de prévoir la quantité de pièces de rechange nécessaires durant une période bien précise, basée sur une base de connaissances concernant l'historique de consommation des pièces de rechange pendant des durées régulières.

L'agent prévention de chaque site détecte les situations anormales de sur-stockage et de sous-stockage et informe son superviseur, Il est doté :

D'une base de connaissance contient les tables suivantes :

- Une table d'historique contient la situation de stock par chaque période de prévention.
- Une table des offres sert à inscrire les offres proposées aux d'autres sites.
- Une table des demandes sert à inscrire les demandes des besoins.
- D'un mécanisme d'intelligence capable d'évoluer suivant les besoins et les contraintes, et prend des décisions en fonction de sa base de connaissances.

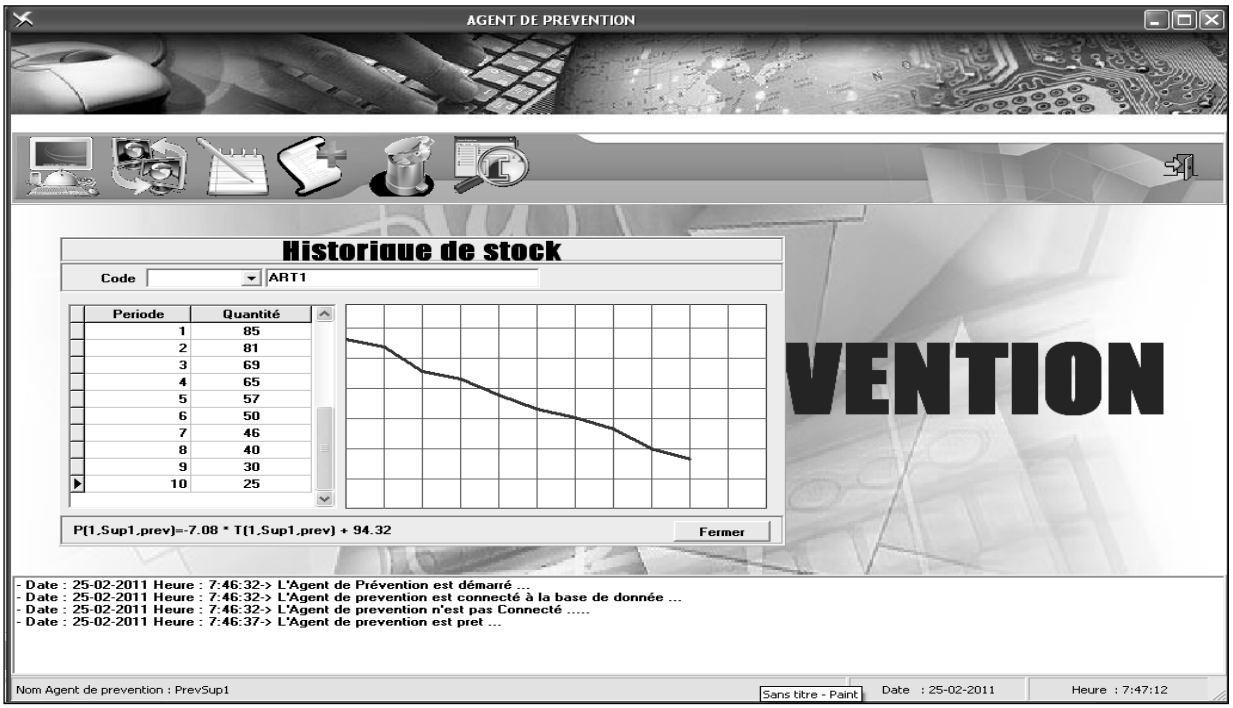


Figure 8.12 : Interface graphique d'agent de prévention

Agent superviseur :

C'est l'agent maître associé à un site, son rôle est la gestion et le contrôle du fonctionnement du système au niveau de site

C'est l'agent superviseur qui communique les quantités entrantes et sortantes d'articles. C'est lui qui lance aussi les requêtes d'acquisition et l'offre des pièces de rechange au coordinateur de groupe.

C'est le superviseur qui reçoit et envoie les messages aux autres sites, Il joue donc le rôle d'une interface de communication et de négociation par rapport aux autres sites pour acquérir une quantité de pièces de rechanges et offrir une autre quantité suivant son désir, donc Il est doté :

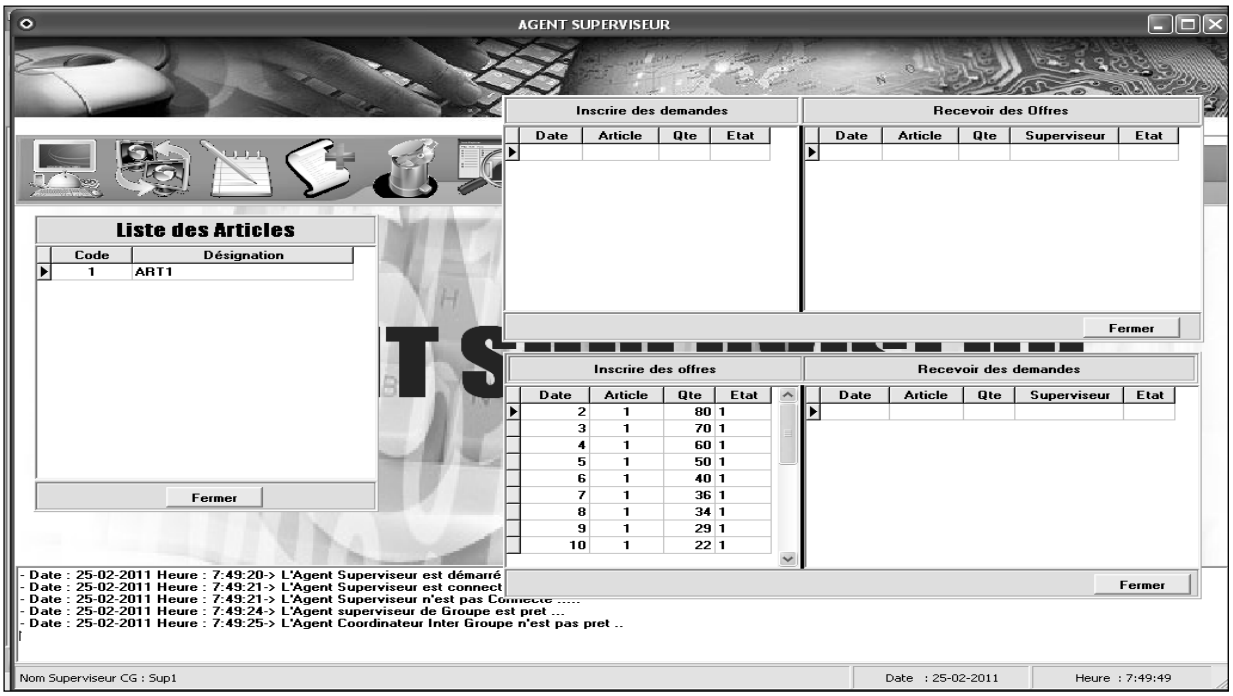


Figure 8.13 : Interface graphique d'agent superviseur de site

D'une base de donnée qui comporte les tables suivantes :

- Une table des offres sert à indiquer la situation et l'état des offres proposées.
- Une table des demandes sert à indiquer la situation et l'état des demandes établis.
- D'un mécanisme d'intelligence capable d'entrer dans des négociation avec d'autres sites.

Agent coordinateur inter groupes

C'est l'agent maître associé au système, il joue le rôle d'un serveur de nom, afin de garantir l'unicité de l'attribution des noms des articles et les noms des coordinateurs de groupe, et les superviseurs, qui les identifiera d'une façon unique sur tout le réseau.

Garantir l'auto- organisation de système, il inscrit les coordinateurs et les superviseurs, et assure la répartition idéale des superviseurs sur les coordinateurs de groupe suivant des critères tel que la distance, le type de pièces de rechange échangées entre les sites.

Recevoir les offres et les demandes non satisfaites dans les groupes envoyés par les coordinateurs de groupe, et il est chargé de trouver un équilibre des ressources inter groupe et satisfait les demandes émanées.

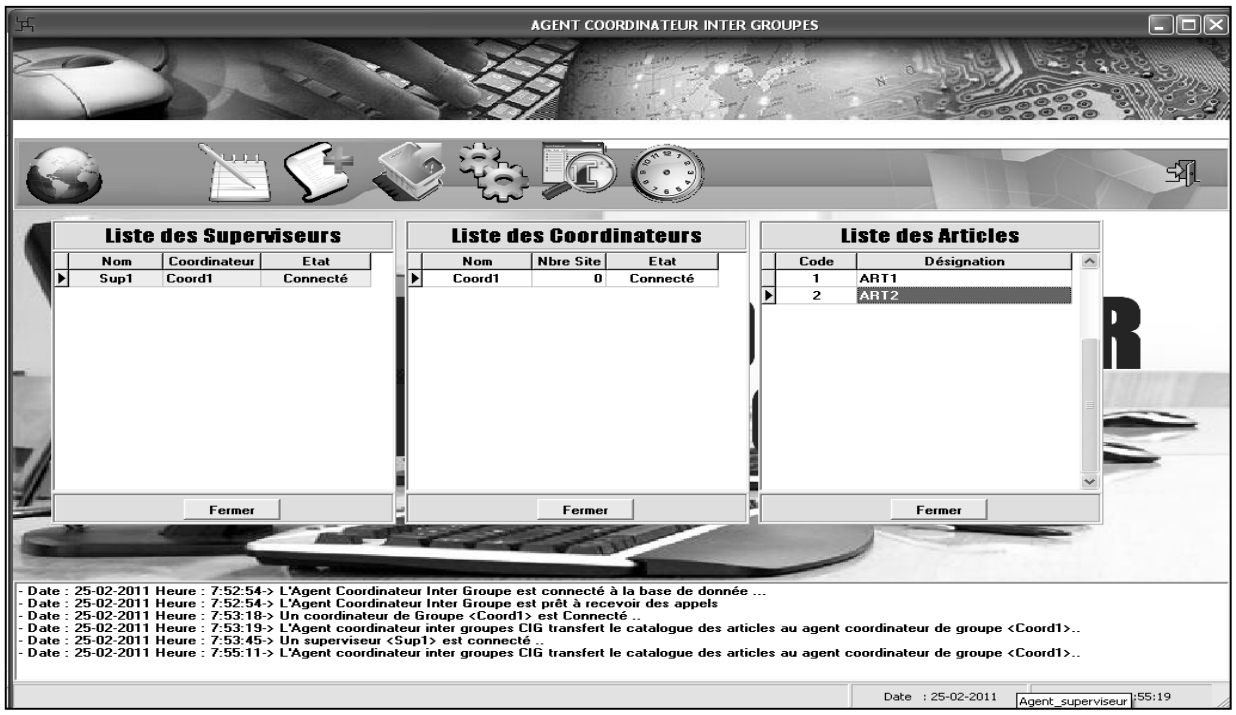


Figure 8.14: Interface graphique d'agent coordinateur inter groupe

- Liste des coordinateurs et leurs états de connexion.
- Liste des superviseurs et leurs ainsi états de connexion.
- Un catalogue des articles échangés.
- Lise des offres reçues.
- Liste des demandes reçues
- Un compteur de période pour assurer la synchronisation du système, et avoir un état global périodique.

Agent coordinateur de groupe C.G:

C'est un agent spécialisé associé à un groupe de sites, son rôle est de coordonner et la réception des besoins en quantité d'articles des différents sites du groupe, surplus et déficit pour chaque article,

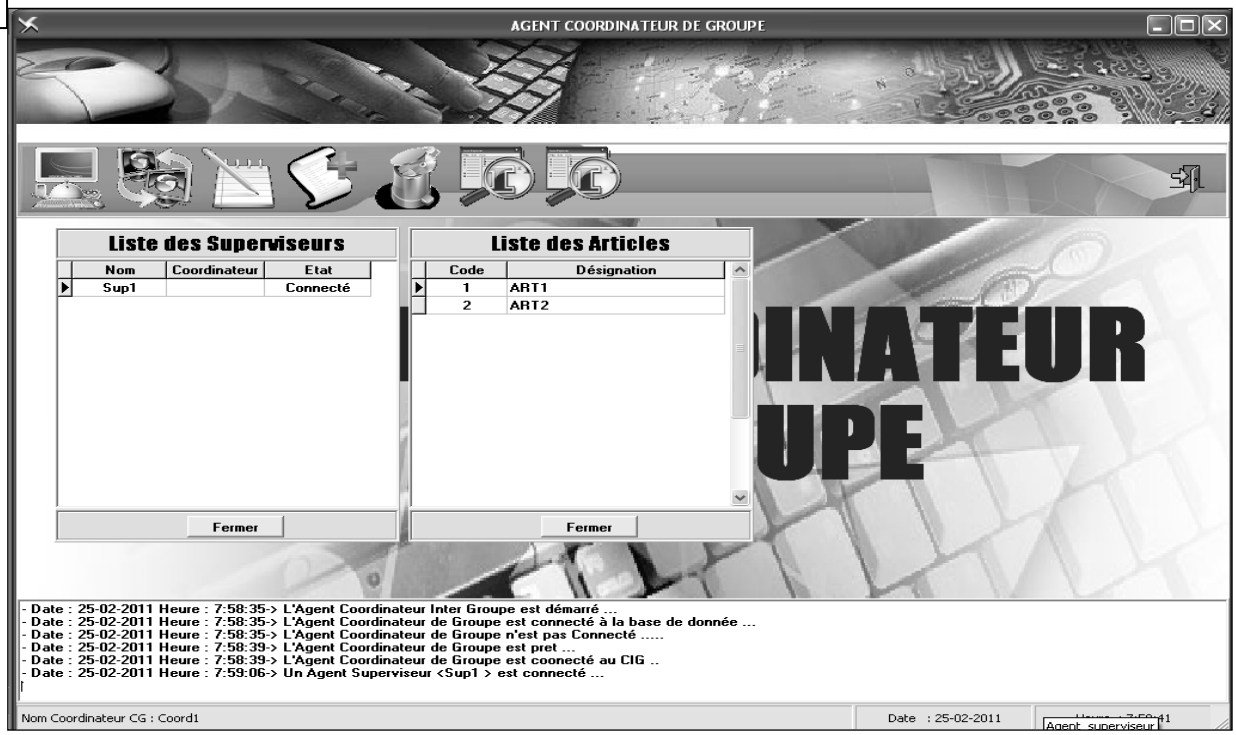


Figure 8.15 : Interface graphique d'agent coordinateur de groupe

Il est doté d'une base de données comportant les tables suivantes :

- Une table des offres sert à inscrire toutes les offres proposées par les sites de groupe.
- Une table des demandes sert à inscrire toutes les demandes de besoins établis par les sites de groupe.
- Une table pour inscrire les noms et les états de connexion des sites de groupe
- D'un mécanisme d'intelligence capable de classer ces besoins et optimiser les coûts par l'application des méthodes de recherche et répond aux demandeur par le meilleur qui convient, on peut varie les critères de sélection pour répondre aux besoins de ces demandeurs et assurer l'équilibre des stocks au niveau de groupe.

VIII-6- Conclusion

Dans ce chapitre j'ai illustré la phase d'analyse et de conception à l'aide des diagrammes du langage UML, et j'ai présenté le schéma général d'un système Multi agents qui décrit toutes les relations entre les différents agents de ce système, j'ai présenté aussi l'architecture de tel système où j'ai montré les différents modules d'envoi, de réception et de travail.

Cette architecture est basée sur cinq agents coopératifs aux fonctionnements indépendants, ce qui rend facile l'extension de leurs fonctionnalités et permet une large portabilité vers différents systèmes, et différents environnements d'exécution (elle ne dépend ni des systèmes d'exploitation, ni des machines, ni des langages de programmation).

Conclusion

Conclusion Générale

Le premier but de ce mémoire est montrer l'importance de middleware comme technologie permettant de répondre aux besoins actuels de l'industrie. Ces besoins s'expriment en termes de distribution et de coopération entre applications ou entre composants d'applications

Le middlewares (Intergiciel) est un logiciel de connexion formé d'un groupe de services qui permettent l'exécution de plusieurs applications sur une ou plusieurs machines connectées en réseau. Cette technologie fournit une couche d'abstraction qui permet l'interopérabilité entre différents langages de programmation, systèmes d'exploitation et architectures d'ordinateurs

Nous avons commencé ce travail par une étude approfondi de l'état de l'art de ce qui à été fait dans ce domaine de recherche scientifique,

Ainsi les Différentes technologies réalisant le middleware suivant ce concept, sont représentés et analysées, il apparaît que leurs utilisation est influente profondément sur l'architectures globale d'un système d'information,

Dans ce mémoire, nous avons présentées les technologies les plus répondues :

CORBA de l'Object Management Group (OMG) est une norme, décrit une architecture et définit les spécifications pour traiter les objets distribués sur le réseau, dans un environnement hétérogène (plusieurs systèmes et plusieurs langages).

Java RMI de Sun Microsystems est un middleware qui permet de réaliser des applications distribuées Java à Java dans un environnement à plusieurs systèmes d'exploitation.

DCOM est une technique propriétaire de Microsoft qui permet la communication entre des composants logiciels distribués au sein d'un réseau informatique doté un système d'exploitation MicroSoft , dans un environnement de développement multi langages

Et le deuxième but de ce mémoire est de faire une étude de comparaison et de synthèse entre les différentes technologies de middleware, nous avons proposé deux types de comparaison : descriptif et objectif dans le but de trouver le middleware optimale pour une implémentation quelconque.

Dans la comparaison descriptive, le choix est simple, et il est basé sur la disponibilité d'une contrainte ou l'absence d'elle, tel que le langage de programmation, le système d'exploitation, le fournisseur et le coût de middleware.

Dans la comparaison objective, le choix est basé sur des tests de performance, comme le temps de transmission et le nombre d'erreurs, suivant un algorithme de teste, d'où les conditions d'exécution sont égaux.

Dans cette partie on a favorisé l'utilisation de CORBA, puisque c'est une solution d'avenir multi plates-formes, multi langages et en plus c'est une norme performante d'après les testes que nous avons réalisés.

En fin comme troisième but, nous avons exposé les démarches concernant l'intégration des solutions SMA dans un environnement distribué pour résoudre le problème de e_maintenance pour l'équilibrage des ressources, le cas de pièces de rechange dans une entreprise étendue à l'aide de l'outil de communication CORBA,

Le but revient à gérer le stock en quantité stratégique juste suffisante pour répondre au besoin du bon fonctionnement de l'entreprise

Un sous-stockage provoque une rupture de stock, ce qui peut générer l'arrêt de la machine qui peut à son rôle altérer la chaîne de production.

Par contre, un sur-stockage provoque des coûts inutiles (assurances, locaux, gardiennage, vieillissement, etc.), donc n'a pas besoin de tenir au stock les pièces de rechanges si le délai normal le permet

L'équilibrage est basé sur une estimation préventive ou future de consommation et de disponibilité de pièces de rechange afin de simuler les situations déséquilibrées des stocks

En perspective de ce travail, il apparaît après analyse des modèles CORBA, DCOM et RMI, que l'un n'est pas meilleur que l'autre mais que chacun présente des avantages et des inconvénients, dans ce contexte la meilleure façon pour un constructeur de logiciels n'est pas de choisir entre l'un ou l'autre modèle mais d'offrir une passerelle entre les deux, cette passerelle doit permettre à un objet du monde CORBA de communiquer avec un objet du monde DCOM et vice versa par exemple, dans ce contexte une étude exhaustive pourra être fait pour présenter et étudier les passerelle existant.

Jusqu'à présent, nous avons développé des testes et des exemples d'applications dans un contexte distribué et dans lequel nous maîtrisions tous les éléments, nous connaissions la configuration des postes client et serveurs et généralement la nature du réseau qui transporte les données entre ces différents poste physiques.

Nous souhaitons de complexifier notre environnement d'exécution en intercalant l'Internet entre les différentes technologies de middleware, ou nous intéressons à l'exécution d'une application CORBA, DCOM ,et RMI dans un cadre Internet , et augmenter la comparaison à atteindre les services Web tel que SOAP et RPC-XML.

Etudier la mise en œuvre d'une application repartie dans un contexte Internet en explorant les différents moyens à notre disposition pour faire communiquer les différents composants de l'application répartie

Annexes

Table des figures

Chapitre –VIII- :

Table des Diagrammes

Table des Tableaux

Chapitre –II- :

Chapitre –V- :

Chapitre –VII- :

Table des Codes

Table des Abréviations

AFNOR	**A**ssociation **F**rançaise de **Nor**malisation
BOA	**B**asic **O**bject **A**dapter
CG	**C**oordinateur de **G**roupe
CIG	**C**oordinateur **I**nter **G**roupe
CM	**C**orba **M**essaging
COM	**C**omponent **O**bject **M**odel.
DCOM	**D**istributed **C**omponent **O**bject **M**odel.
DII	**D**ynamic **I**nvocation **I**nterface
DLL	**D**ynamic **L**ink **L**ibrary
EDI	**E**nvironnement de **D**eveloppement **I**ntegre
EXE	Une extension de fichier sur windows pour les fichiers **EXE**cutables
GIOP	**G**eneric **I**nter-**O**rb **P**rotocol.
GUID	**G**lobally **U**nique **I**dentifier
IDL	**I**nterface **D**efinition **L**anguage.
IIOP	**I**nternet **I**nter-**O**rb **P**rotocol.
IML	**U**nified **M**odeling **L**anguage
IOR	**I**nteroperable **O**bject **R**eference.
JVM	**J**ava **V**irtual **M**achine
MIDL	**M**icrosoft **I**nterface **D**efinition **L**anguage.
MOM	**M**essage **O**riented **M**iddleware
OD	**O**bject **D**istribute
OLE	**O**bject **L**inking And **E**mbedding
OMAG	**O**bject **M**anagement **A**rchitecture **G**uide.
OMG	**O**bject **M**anagement **G**roup.
ORB	**O**bject **R**equest **B**roker.
ORPC	**O**bject **R**emote **P**rocedure **C**all
OTS	**O**bject **T**ransaction **S**ervice.
POO	**P**rogrammation **O**rientee **O**bjet
RAD	**R**apid **A**pplication **D**evelopment
RMI	**R**emote **M**ethod **I**nvocation.
RPC	**R**emote **P**rocedure **C**all
SII	**S**tatic **I**nvocation **I**nterface
SMA	**S**ysteme **M**ulti **A**gents.
TLB	**T**ype **L**ibrary

Bibliographie

[01] [Wil 01] Wiley Son R. E. Schantz and D. C. Schmidt, Encyclopedia of software engineering, chapter middleware for distributed systems: Evolving the common structure for network-centric applications

[02] [Kra 05] Krakowiak, Sacha. "What's middleware?". ObjectWeb.org. http://middleware. objectweb.org/. Retrieved 2005-05-06.

[03] [Joh 08] John Footen et Joey Faust, The Service-Oriented Media Enterprise: SOA, BPM, and Web Services in Professional Media Systems, Focal Press, 2008, (ISBN 9780240809779), chapitre 4 definition of a middleware.

[04] [ACR 99] D. Acremann,G. Moujeard, L. Rousset "Développer avec CORBA en java et C++ ", Campus Press-réference,1999

[05] [Bir 84] Birrell, A. D. and Nelson, B. J. (1984). Implementing remote procedure calls. ACM Transactions on Computer Systems, 2(1):39-59.

[6] [Tay 90] TAY, B. H. AND ANANDA, A. L. 1990. A survey of remote procedure calls. ACM Operat. Syst. Rev. 24, 3 (July), 68–79.

[07] [Fox 98] Fox, A., Gribble, S. D., Chawathe, Y., and Brewer, E. A. (1998). Adapting to network and client variation using infrastructural proxies: Lessons and perspectives. IEEE Personal Communications, pages 10-19.

[08] [Phi 96] Philip A. Bernstein. Middleware: A model for distributed system services. Communicatios of the ACM, 39(2):86–98, 1996.

[09] [Wie 95] Wieringa, R. and de Jonge, W. (1995). Object identifiers, keys, and surrogates: Object identifiers revisited. Theory and Practice of Object Systems, 1(2):101-114.

[10] [Bie 02] Bieber, G. and Carpenter, J. (2002). Introduction to Service-Oriented Programming. http://www.openwings.org.

[11] [GS 96] S. GARFINKEL et G. SPAFFORD. Practical Unix & Internet Security. O'Reilly and Associates, 1996.

[12] [Ban 99] BANAVAR, G., CHANDRA, T., STROM, R., AND STURMAN, D. 1999. A case for message oriented middleware. In Proceedings of the 13th International Symposium on Distributed Computing (DISC 99). 1–18.

[13] [Hua 01] HUANG, Y. AND GARCIA-MOLINA, H. 2001. Publish/ subscribe in a mobile enviroment. In Proceedings of MobiDE. 27–34.

[14] [Car 97] Carnegie Mellon UNIVERSITY. Middleware, CMU/SEI-97-HB-001. Rapport technique, Software Engineering Institute - Carnegie Mellon University, 1997.

[15] [Vin 97] S. Vinoski, Corba: Intergrating diverse applications within distributed heterogeneous environments, IEEE Communications (1997).

[16] [DEV 02] F. Guerrero. "Présentation de CORBA" document tiré de « Etude des documents de l'OMG en vue de leur utilisation dans des applications multimédias distribuées.

[17] [OMA 02]. OMG. 2002a. The Common Object Request Broker: Core Specification. Object Management Group, Needham, MA.

[18] [Sie 00] Siegel, Jon. 2000. CORBA 3 Fundamentals and Programming. ISBN: 0-471-29518-3. United States of America. John Wiley & Sons, Inc

[19] [OMG 00] Object Management Group, The common object request broker: Architecture and specification revision 2.4, OMG Trchnical Document formal/00-11-07 (2000).

[20] [GEI99]. « CORBA : des concepts à la pratique » Jean-Marc Geib, Christophe Gransart et Philippe Merle.Collection InterEditions, Editions MASSON, Paris, France, 1997.ISBN : 2-225-83046-0 http://corbaweb.lifl.fr

[21] [CORBA 98] « The Common Object Request Broker : Architecture and Specification, revision 2.2 » OMG, USA, Février 1998. OMG TC Document formal/98-07-01 http://www.omg.org/corba/corbiiop.htm Appelé aussi « CORBA/IIOP 2.2 Specification ».

[22] [COS 98] « CORBAservices : Common Object Services Specification » OMG, USA, Novembre 1997.OMG TC Document formal/98-07-05 http://www.omg.org/ corba/sectrans.htm

[23] [OMG 00b] Naming object service v1.2. Specification, Object Managemen Group, Mai 2000.

[24] [OMG 00c] Trading object service v1.0. Specification, Object Management Group, Mai 2000.

[25] [Wol96] Wollrath, A., Riggs, R., and Waldo, J. (1996). A Distributed Object Model for the Java System. Computing Systems, 9(4):265-290.

[26] [Jam 95] James A. Gosling. "Java intermediate bytecodes". In Proc. ACM SIGPLAN Workshop on Intermediate Representations, pages 111–118. ACM, 1995.

[27] [Tru 01] Trusted Logic. Off-card bytecode verifier for Java Card. Distributed as part of Sun's Java ,Card Development Kit, 2001

[28] [Tim 99] Tim Lindholm and Frank Yellin. The Java Virtual Machine Specification. The Java Series. Addison-Wesley, 1999. Second edition.

[29] [SUN 02] SUN. 2002. Java Remote Method Invocation Specification. Sun Microsystems, Santa Clara, CA

[30] [Nag 03] Nagappan R., Skoczylas R. & Sriganesh R. P., (2003), "Developing Java Web Services – Architecting and developing secure Web Services using Java", Wiley Computer Publishing.

[31] [JND 03] Java naming and directory interface 1.2.1 specification. Specification, Sun Microsystems, 2003.

[32] [All 01] Allamaraju et al., (2001), "Professional Java Server Programming – J2EE 1.3 Edition", Wrox

[33] [Mic 03] Microsoft Corporation Understanding the distributed object component model (dcom) architecture, http://www.microsoft.com/ ntserver/ techresources/ appserv/ COM/ dcom_architecture.asp, January 2003.

[34] [COM 95] The Component Object Model Specification, http://www.microsoft.com/ oledev/olecom/title.htm

[35] [Edd 98] G. Eddon, H. Eddon, Au cœur de Distributed COM, Microsoft Press France, 1998

[36][Bro 98] N. Brown and C. Kindel Distributed Component Object Model Protocol DCOM : http//www.globecom.net/ietf/draft/draft-brown-dcom-v1-spec-03.html, January 1998

[37] [Dal 01] www.DalmatianGroup (2001), http://www.dalmatian.com/ com_dcom.htm , Last accessed 2006-04-14

[38] [Cap 03] Capraro Mario,Baglin Gérard « L'entreprise étendue et le développement des fournisseurs » Presses Universitaires de Lyon, 2003, (ISBN 978-2729707156)

[39] [Mar 05] Mariotti F., Qui gouverne l'entreprise en réseau, Presses de Sciences Po, Paris, 2005, (ISBN 978-2724609592)

[40] [Mei 10] Meissonier R., Organisation Virtuelle : du concept à la pratique, Editons Universitaires Européennes, 2010, (ISBN 978-613-1-51618-4)

[41] [Fra 04] Franklin Brousse, Jean-Denis Garo, Arnaud Loisel et Pascal Prot, Guide TIC des petites et moyennes collectivités , Édition Ficome, Paris, 2004

[42] [Ly 99] Ly, F. ; Simeu-Abazi, Z.; Leger, J-B. "Maintenance terminology review", Report of the Research Group of Safety Functioning Production, Grenoble France, 1999.

[43] [Ras 06] Rasovska I. (2006) Contribution à une méthodologie de capitalisation des connaissances basée sur le raisonnement à partir de cas : Application au diagnostic dans une plateforme d'e-Maintenance. Phd Thesis Université de Franche- Comté

[44] [Fer 98] Ferber J., Gutknecht A., A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. , International Conference on Multi-Agents Systems (ICMAS). IEEE CS Press, June 1998. ICMAS'98

[45] [Jen 98] JENNINGS, N., SYCARA, K. et WOOLDRIDGE, M. (1998). A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems, 1:7–38.

[46] [Wey 06] WEYNS, D. (2006). An architecture-centric approach for software engineering with situated multiagent systems. Thèse de doctorat, Katholieke Universiteit Leuven

[47] [Dur 89] DURFEE, E. et LESSER, V. (1989). Negotiating task decomposition and allocation using partial global planning. Distributed Artificial Intelligence, 2:229–244.

[48] [Sib 01] C. Sibertin-Blanc Pour une typologie des espaces d'interaction dans les SMA, IRIT / Université Toulouse1 2001

[49] [Maz 01] Mazouzi, Ingénierie des protocoles d'interaction: Des systèmes distribués aux systèmes multi-agents, Thèse de doctorat, Université Paris IX, 2001

[50] [Woo 95] Wooldridge M. et N. R. Jennings, Agent theories, architectures, and languages, Dans Wooldridge, Jennings (ed), Intelligent Agents, Springer Verlag, 1995.p.-22.

[51] [Ben 04] Benaouda A., Benaouda N., Simulation des architectures parallèles sur réseaux de PCs : Algorithmes, implémentation et Performances, CIGE'04, Sétif, 2004

[52] [Che 01] Chelbi A., Ait-Kadi D., Spare provisioning strategy for preventively replaced systems subjected to random failure, International Journal production economis, No 74, PP. 183-189, 2001.

[53] [Ben 05] A. Benaouda— N. Zerhouni — M. Mostefai : A Preventive Interaction Model Applied to the Management of Spare Parts

[54] [Mar 04] Martin Fowler et al. (2004). UML 2.0 (ISBN 2-7440-1713-2) : initiation aux aspects essentiels de la notation

Web Bibliographie

Middleware

- http://www.theadvisors.com/Client-Serveur.htm
- http://www.info.uqam.ca/~obaid/obaid_ens.html
- http://guerraoui.developpez.com/systemes-repartis/
- http://krakowiak.developpez.com/cours/middleware-adaptable/
- http://lifc.univ-fcomte.fr/
- http://apiacoa.org/contact.html.
- http://www.wisegeek.com/what-is-middleware.htm

CORBA

- http://www.omg.org/gettingstarted/corbafaq.htm
- http://www.wikipedia.com
- http://corba.developpez.com
- http://www.sylbarth.com/corba/corba.html
- http://www.iona.com/support/docs/orbix/gen3/33/html/orbix33cxx_intro/Chapter_01.html
- http://www.omg.org
- http://www.corba.org
- http://www.inprise.com/visibroker/
- http://www.inprise.com/techpubs/visibroker/
- http://www.inprise.com/devsupport/visibroker/faq/
- http://www.inprise.com/visibroker/papers/
- http://www.developpez.com/corba
- http://www.cs.wustl.edu/~schmidt

DCOM

- http://www.microsoft.com/COM/
- http://windows.developpez.com/faq/dcom/
- http://msdn.microsoft.com
- http://www.microsoft.com/oledev/olecom/title.htm .
- http://www.microsoft.com/msdn/sdk/techinfo/dcombo.doc .
- http://www.ActiveX.org .
- http://casteyde.christian.free.fr
- http://www.microsoft.com/com/resources/specs.asp
- http://www-sop.inria.fr/
- http://msdn.microsoft.com/library/default.asp

RMI

- http://java.sun.com/docs/books/tutorial/rmi/index.html
- http://java.sun.com/developer/onlineTraining/rmi/index.html
- http://developer.java.sun.com
- http://download.oracle.com/javase/1.3/docs/guide/rmi/spec/rmiTOC.html
- http://java.sun.com/j2ee/1.3/docs/tutorial/doc/index.html

- http://java.sun.com/j2ee/overview.html
- http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html
- http://www.learn-rmi.thiyagaraaj.com/

Maintenance

- http://fr.wikipedia.org/wiki/Maintenance_préventive
- http://www.concept-group.co.uk/solutions/solutions/e-Maintenance.html
- http://www.aircraftdf.com/ms_maintenance.htm
- http://www.yelp.com/biz/circle-e-maintenance-euless-2
- http://www.resolvefm.com.au/SOLUTIONS/PreventativeCorrectiveMaintenance
- http://www.kwaliteg.co.za/maintenance/corrective%20maintenance.htm

Système multi agents

- http://www.fr.wikipedia.org/wiki/Syst%C3%A8me_multi-agents
- http://www.cormas.cirad.fr/fr/demarch/sma.htm
- http://www.multiagent.com/
- http://www.emse.fr/~boissier/enseignement/sma01/references.html
- http://www.limsi.fr/~jps/enseignement/examsma/examsma.htm
- http://sma.lip6.fr/Csma/livres.php

UML

- http://uml.developpez.com/
- http://www.uml.org/
- http://uml.free.fr/index-cours.html
- http://www.abrillant.com/doc/uml/index.htm
- http://www.agilemodeling.com/artifacts/classDiagram.htm
- http://frog.isima.fr/antoine/uml.shtml
- http://www.digicomp.ch/cours/UML.html
- http://laurent-audibert.developpez.com/Cours-UML/
- http://charlie-soft.com/Modelisation/uml.php

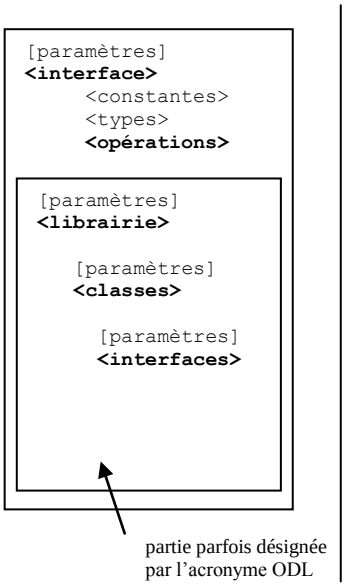
Annexe A

MIDL –Microsoft langage de description d'interfaces (Interface Definition Language)

1- Langage de description des interfaces des objets COM

- Permet de définir des interfaces, constantes, types et opérations
- Pas de variables (au sens attributs de IDL CORBA)
- Supporte l'héritage d'interfaces

1-1. Structure d'un fichier MIDL




```
[ object, uuid(9372A220-B4BD-11d1-ABE1-00207810D5FE) ]
interface IOcr : IUnknown
{
        const char * chaine = «exemple»;
        typedef float reel;
        [message] HRESULT met( [in] boolean bool,
                [in,size_is(12)] byte * p );
        HRESULT meth2( [out,string] wchar_t * ptr );
}

[ uuid(9372A2B2-B4BD-11d1-ABE1-00207810D5FE), version(1.0) ]
library OCREngineLib
{
        [ uuid(9372A2B3-B4BD-11d1-ABE1-00207810D5FE) ]

        coclass CoOcrEngine
        {
                [default] interface IOcr;
                interface ISpell;
        }
}
```

1-2. Types de données MIDL

- **`boolean` (1 octet)**
- **`byte` (1 octet)**
- **`char` (1 octet), `wchar_t` (2, Unicode)**
- **`float` (4 octet), `double` (8 octet)**
- **`small` (1 octet)**
- **`short` (2 octet)**
- **`long` (4 octet)**
- **`hyper` (8 octet)**
- **`*` (4 octets, pointeur)**
- **`[]` (tableau)**
- **`enum` (énumération)**
- **`struct` (structures)**
- **`union` (type discriminé)**
- **`VARIANT` (type indifférencié)**

1-3. Exemple

```
typedef long age;
struct personne
{
        float taille;
        long age;
};
enum couleur {rouge, vert, bleu};
```

```
union carre switch(couleur c)
{
        case rouge: short num1;
        case vert : long num2;
        case bleu : hyper num3;
};
typedef long tab [];
typedef short * ptr;
```

1-4. Paramètres de MIDL

Paramètres des interfaces
- **[object] : objet COM distant**
- **[local] : objet COM local**
- **[uuid] : l'identifiant de l'interf.**
- **[version] : n° de version de l'interf.**

Paramètres des méthodes
- **[message] : appel de méthode asynchrone (seul. param. entrée)**
- **[maybe] : idem mais sémantique de l'appel best effort**

Paramètres des arguments de méthodes
- **[in], [out] et [in,out] : entrée, sortie et entrée/sortie**
- **[size_is(n)] : taille pour un pointeur ou un tableau**
- **[string] : le pointeur est considéré comme une chaîne = zone mémoire terminée par NULL**

OMG IDL- Interface Description Language

2- Langage de description des interfaces des objets CORBA
- Utilisé pour décrire les interfaces d'Objets CORBA
- Une interface décrit les opérations et les attributs d'un Objet CORBA
- Indépendant des langages de programmation
- Projections (mapping) vers des langages de programmation orientés objet ou non

2-1. Structure d'un fichier OMG-IDL

Syntaxe : *Interface | [< héritage >] {< interface Body >} ;*

Interface →
Attribut →
Opérations →
Exception →
Opérations →

```
interface account
 {
        readonly attribute float balance;
        attribute string description;
        void makeLodgement (in float f);
        void makeWithdrawal (in float f);
};
interface currentAccount : account
{
        readonly attribute float overdraftLimit;
};
interface bank
{
        exception reject {string reason;};
        account newAccount (in string name)
                raises (reject);
        currentAccount newCurrentAccount (in string name, in float limit)
                raises (reject);
        void deleteAccount (in account a);
};
```

2-2. Types de données MIDL

Types de base
- **short**
- **long**
- **unsigned short**
- **unsigned long**

- **float**
- **double**
- **char**
- **boolean**
- **octet**
- **any**

Types construits

- **struct**
- **union**
- **enum**

Types génériques

- **array**
- **sequence**
- **string**

2-3. Exemple

```
string Name;
sequence <Name>  Names;
struct Group
{
        String Aliases[3];
        Names Users;
};
```

```
enum Sex {Male, Female};
union Status switch (Sex)
{
        Case Male: boolean Bearded;
        default: long Children;
};
```

2-4. Paramètres de OMG-IDL

- in: initialisé seulement par le client et passé à l'objet
- out : initialisé seulement par l'objet et passé au client
- inout : initialisé par le client et passé à l'objet; l'objet peut modifier la valeur avant la retourner au client

2-5. Projection IDL vers Java :

Une interface IDL est projetée en plusieurs classes Java : nom original + suffixes

Exemple : Bonjour.idl

- Bonjour.java –interface (côté client / applicatif)
- Bonjour**Operations**.java -interface (côté serveur / Obj. CORBA)
- Bonjour**Helper**.java -méthodes liées au type définit
- Bonjour**Holder**.java -emballage params out/inout
- _ Bonjour**Stub**.java -stub
- Bonjour**POA**.java -skeleton
- Bonjour**POATie**.java -skeleton (variant pour la délégation)

Annexes B

Voici un exemple de code source complète de programme client utilisé dans les testes de Performances de middleware RMI

1- ClientTestRMI.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.rmi.*;
import java.io.*;

public class ClientTestRMI
{
   public static void main(String[] args) throws Exception
   {
      Bonjour serveur = null;
      String nomComplet;
      int rep;
      BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Entrez le nom de poste du serveur /LocalHost :");
      String rep2 =  stdIn.readLine() ;
      nomComplet ="rmi://" + rep2+":1099/" + "monObjet";
      try
      {
       do
       {
        long start = System.currentTimeMillis() ;
        serveur = (Bonjour)Naming.lookup(nomComplet);
        long end = System.currentTimeMillis() ;
        System.out.println("Le temps pour effectuer la recherche du serveur est: " + (start - end)  );

                                        // le Choix du methode à executer
                                        System.out.println(" -1- La fonction : String DirBonjour(string nom , String prenom ) ");
                                        System.out.println(" -2- La fonction : int TestInteger(int n ) ");
                                        System.out.println(" -3- La fonction : int TestAddition(int n1 , int n2)");
                                        System.out.println(" -4- La fonction : String TestChaine(string ch) ");
                                        System.out.println(" -5- La fonction : int [] TestTableau(int tableau[]) ");
                                        System.out.println(" -6- La fonction : double TestDouble(double x) ");
        String rep0 = (String)JOptionPane.showInputDialog(null,"Votre choix   :" ,"RMI
        Application",JOptionPane.QUESTION_MESSAGE);
        int n = Integer.parseInt(rep0) ;
        switch (n)
        {
          case 1 : {
                    rep0 = (String)JOptionPane.showInputDialog(null,"Votre nom   :" ,"Dir Bonjour", JOptionPane.QUESTION_MESSAGE);
                    String rep1 = (String)JOptionPane.showInputDialog(null,"Votre prenom:" ,"Dir Bonjour",
                   JOptionPane.QUESTION_MESSAGE);
                    System.out.println("nom    :" + rep0) ;
                    System.out.println("prenom :" + rep1) ;
                   System.out.println(serveur.DirBonjour(rep0,rep1)) ;
                    break ;
                   }
          case 2 : {
                    rep0 = (String)JOptionPane.showInputDialog(null,"le nombre d'element:" ,"TestInteger",
                    JOptionPane.QUESTION_MESSAGE);
                    int []TableauDesEntierEnvoyes,TableauDesEntierRecus ;
                    n = Integer.parseInt(rep0) ;
                    TableauDesEntierEnvoyes = new int[n] ;
                    TableauDesEntierRecus   = new int[n] ;
                    long []TableauTime;
                    TableauTime = new long[100] ;
                    for (int i = 0;i<n;i++)
                       TableauDesEntierEnvoyes[i] = i ;
                    int nbr_erreur = 0 ;// calculer le nombre des erreurs
                    float temps_moyen = 0 ;
                     for (int j =1 ; j<=100 ;j++)
                    {
                             int i = 0 ; // compteur
                              start = System.currentTimeMillis() ;
                                 do
                                 {
                                        TableauDesEntierRecus[i] = serveur.TestInteger(TableauDesEntierEnvoyes[i]);
                                            i++ ; // incrémentation de compteur
```

```
                    }
                    while ( i!=n )     ;
                   end = System.currentTimeMillis() ;
                  for(i=0 ;i<n;i++)
                    {
                      if (TableauDesEntierEnvoyes[i] != TableauDesEntierRecus[i] )
                        nbr_erreur ++ ;
                    }
                    TableauTime[j-1] = end - start ;
                    temps_moyen = temps_moyen + (end - start ) ;
                  }
                            System.out.println("Le test est termine,le nombre d'erreur est :"+nbr_erreur);
                  System.out.println("Le temps pour effectuer le test : " + (temps_moyen/100) );
                  for(int j = 0 ; j< 100 ; j++)
                  {
                      System.out.println("test N° "+j +" = " + TableauTime[j]);
                      clavier.lireString();
                  }
                  break ;
          }
case 3 : {
          rep0 = (String)JOptionPane.showInputDialog(null,"le nombre d'element:" ,"TestAddition", JOptionPane.QUESTION_MESSAGE);
          int []TableauDesEntierEnvoyes,TableauDesEntierRecus ;
          n = Integer.parseInt(rep0) ;
          TableauDesEntierEnvoyes = new int[n] ;
          TableauDesEntierRecus   = new int[n] ;
          for (int i = 0;i<n;i++)
             TableauDesEntierEnvoyes[i] = i ;
          int nbr_erreur = 0 ;// calculer le nombre des erreurs
          float temps_moyen = 0 ;
          for (int j =1 ; j<=100 ;j++)
          {
                int i = 0 ; // compteur
                    start = System.currentTimeMillis() ;
                    do
                    {
                          TableauDesEntierRecus[i] = serveur.TestAddition(TableauDesEntierEnvoyes[i],TableauDesEntierEnvoyes[i]);
                              i++ ; // incrémentation de compteur
                    }
                    while ( i!=n )     ;
             end = System.currentTimeMillis() ;
              for(i=0 ;i<n;i++)
                    {
                      if ((TableauDesEntierEnvoyes[i]*2) != TableauDesEntierRecus[i] )
                        nbr_erreur ++ ;
                    }
                    temps_moyen = temps_moyen + (end - start ) ;
                  }
                            System.out.println("Le test est termine,le nombre d'erreur est :"+nbr_erreur);
                  System.out.println("Le temps pour effectuer le test : " + (temps_moyen/100) );
                  break ;
          }
case 4 : {
          rep0 = (String)JOptionPane.showInputDialog(null,"la chaine de text :" ,"TestChaine", JOptionPane.QUESTION_MESSAGE);
          n = Integer.parseInt(rep0) ;
          rep0 = "" ;
          for(int j = 1 ; j<= n ;j++)
            rep0 = rep0 + 'a' ;
          int nbr_erreur = 0 ;// calculer le nombre des erreurs
          float temps_moyen = 0 ;
          System.out.println("Le temps pour effectuer le test : ");
          String rep1 = "" ;
          for (int j =1 ; j<=2 ;j++)
          {
                    start = System.currentTimeMillis() ;
                     rep1 = serveur.TestChaine(rep0);
                     end = System.currentTimeMillis() ;
                    if (rep1 != rep0)
                    {
                         nbr_erreur ++ ;
                    }
                    temps_moyen = temps_moyen + (end - start ) ;
             }
            System.out.println("Le test est termine,le nombre d'erreur est :"+nbr_erreur);
             System.out.println("Le temps pour effectuer le test : " + (temps_moyen/2) );
             break ;
          }
```

```
            case 5 : {
                    rep0 = (String)JOptionPane.showInputDialog(null,"le nombre d'element du table:" ,"TestTable"
,JOptionPane.QUESTION_MESSAGE);
                    int []TableauDesEntierEnvoyes,TableauDesEntierRecus ;
                    n = Integer.parseInt(rep0) ;
                    TableauDesEntierEnvoyes = new int[n] ;
                    TableauDesEntierRecus   = new int[n] ;
                    for (int i = 0;i<n;i++)
                       TableauDesEntierEnvoyes[i] = i ;
                    int nbr_erreur = 0 ;// calculer le nombre des erreurs
                    float temps_moyen = 0 ;
                    for (int j =1 ; j<=100 ;j++)
                    {
                            int i = 0 ; // compteur
                                start = System.currentTimeMillis() ;
                                    TableauDesEntierRecus = serveur.TestTableau(TableauDesEntierEnvoyes);
                       end = System.currentTimeMillis() ;
                         for(i=0 ;i<n;i++)
                                {
                                  if (TableauDesEntierEnvoyes[i] != TableauDesEntierRecus[i] )
                                    nbr_erreur ++ ;
                                }
                                temps_moyen = temps_moyen + (end - start ) ;
                              }
                             System.out.println("Le test est termine,le nombre d'erreur est :"+nbr_erreur);
                              System.out.println("Le temps pour effectuer le test : " + (temps_moyen/100) );
                              break ;
                          }
            case 6 : {
                    rep0 = (String)JOptionPane.showInputDialog(null,"le nombre d'element:"
                           ,"TestInteger",JOptionPane.QUESTION_MESSAGE);
                    double []TableauDesEntierEnvoyes,TableauDesEntierRecus ;
                    n = Integer.parseInt(rep0) ;
                    TableauDesEntierEnvoyes = new double[n] ;
                    TableauDesEntierRecus   = new double[n] ;
                    for (int i = 0;i<n;i++)
                       TableauDesEntierEnvoyes[i] = i*0.1 ;
                    int nbr_erreur = 0 ;// calculer le nombre des erreurs
                    float temps_moyen = 0 ;
                    for (int j =1 ; j<=10 ;j++)
                    {
                                 int i = 0 ; // compteur
                                start = System.currentTimeMillis() ;
                                do
                                {
                                       TableauDesEntierRecus[i] = serveur.TestDouble(TableauDesEntierEnvoyes[i]);
                                           i++ ; // incrémentation de compteur
                                }
                                while ( i!=n )     ;
                               end = System.currentTimeMillis() ;
                                for(i=0 ;i<n;i++)
                                {
                                  if (TableauDesEntierEnvoyes[i] != TableauDesEntierRecus[i] )
                                     nbr_erreur ++ ;
                                }
                                temps_moyen = temps_moyen + (end - start ) ;
                              }
                             System.out.println("Le test est termine,le nombre d'erreur est :"+nbr_erreur);
                              System.out.println("Le temps pour effectuer le test : " + (temps_moyen/10) );
                              break ;
                          }
          }
          rep = JOptionPane.showConfirmDialog(null,"RMI Application" ,"avez vous encore des testes ",JOptionPane.YES_NO_OPTION) ;
        }
        while (rep == JOptionPane.YES_OPTION )
}
 catch(Exception e)
 {
                          System.out.println("exeption"); throw e;
 }}}
```

Résumé

Le but de ce mémoire est d'étudier ,présenter et évaluer les différentes plates formes distribuées , sur la base de cette évaluation, nous proposons une plate forme suffisamment souple et efficace pour réaliser une implémentation d'un système multi agents de e-maintenance. Notre premier objectif était de faire une étude comparative et de synthèse sur les différentes plateformes distribuées suivant des critères de comparaison et des testes de performance, pour favoriser un plate forme sera utilisé dans l'implémentation d'application de e_maintenance. Et comme deuxième objectif , c'est présenté une architecture base sur les agents visée à la e_maintenance , nous exposons les démarches concernant l'intégration des solutions SMA dans un environnement distribuée pour résoudre le problème de l'équilibrage des ressources, le cas de pièces de rechange dans une entreprise étendu à l'aide d'un plates formes distribuées, L'équilibrage est base sur une estimation préventif ou future de consommation et de disponibilité de pièces de rechange afin de simuler les situations déséquilibrer de stock

Mots clés :

Middleware, architecture distribue, système réparti, réseaux, CORBA, RMI, DCOM,SMA, e_maintenance

---

ملخص

الغرض من هده المذكرة هو دراسة و عرض و تقييم منصات التوزيع و على أساس هذا التقييم نقترح منصة مرنة و فعالة بما فيه الكفاية لتحقيق تنفيذ نظام متعدد العملاء للصيانة الشبكية, و كان هدفنا الأول مقارنة هذه المنصات و فقا لمعايير و اختبارات الأداء لاستخلاص منصة لاستخدامها في تنفيذ الصيانة الشبكية وثانيا تقديم بنية تقوم على عوامل الصيانة الشبكية و عرض جميع الخطوات لدمج نظام العملاء مع نظام منصات التوزيع لحل مشاكل موازنة قطع الغيار في مؤسسة موسعة

Abstract

The purpose of this paper is to investigate, present and evaluate the various distributed platforms, based on this assessment, we propose a platform flexible and efficient enough to achieve an implementation of a multi-agent system for e-maintenance Our first objective was to make a comparative study and synthesis on different platforms distributed according to benchmarks and performance tests, to promote a platform will be used in the implementation of application e_maintenance. And as the second target is presented an architecture based on agents referred to e_maintenance, we outline the steps for the integration of solutions in a distributed Multi-agent systems to solve the problem of balancing resource case Spare a help of an extended enterprise platforms distributed Balancing is based on an estimate preventive or future consumption and availability of spare parts to simulate the situations of unbalanced stock

Key words

Middleware, architecture distributes, distributed systems, networks, CORBA, RMI, DCOM, e_maintenance ,SMA