République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE ABDERAHMANE MIRA DE BEJAIA
Faculté des Sciences Exactes
Département d'Informatique

Mémoire de fin de cycle

En vue de l'obtention du diplôme de Master en Informatique

Option: *Réseaux et systèmes Distribués*

Thème

Evaluation des Performances d'une Application Distribuée sous Java RMI, CORBA et services Web

Réalisé par :

Melle BELHATRI Fahima

Encadré par :

Dr H. NACER Talantikit
Mme K. ADEL

Membres de jury
Président : Mr AMROUN
Examinateur : Mr ACHROUFENE Achour.
Examinatrice : Mlle BERNINE Nassima.

Année Universitaire 2012 – 2013

Dédicaces

Je dédie ce modeste travail, aux deux êtres les plus chers dans ce monde, à qui je souhaite une longue vie pleine de joie et de bonheur :

A mon père qui a cru en moi et m'a fait confiance et il m'a assuré touts les moyens pour aller très loin que possible, merci beaucoup papa.

A ma mère : qui été toujours là pour moi et qui ma beaucoup supporté durant la réalisation de ce travail et elle m'a toujours soutenu durant mon parcours, merci beaucoup maman.

A ma sœur Salima et son mari Malek et leurs fils Rafik ;

A mon oncle Houcine qui m'a soutenu durant mon parcours et il été toujours là quand j'ai besoin de lui « merci Dada » ainsi à sa femme Fatiha et leurs enfants : Yanis, Maya et notre adorable Halim que dieu le protège.

A mon frère Aziz ;

A mon neveu Rayane et ma nièce la nouvelle née Marya et leurs parents.

A mon grand père et ma grande mère ;

A tous mes cousins et cousines, oncles et tantes, et toute ma famille ;

A tous mes amis(es) et à lamine qui m'a beaucoup aider durant ce travail ;

A toutes les personnes qui me connaissent de prés ou de loin.

Remerciements

Avant tout je remercie Dieu le tout puissant de m'avoir donné la force d'accomplir ce travail et de le mener jusqu'au bout.

Je tiens à exprimer toute ma gratitude à Mme TALANTIKITE, et Mme ADEL mes promotrices, pour leurs compréhension, conseils et orientations pour la réalisation de ce travail.

Je tiens aussi à remercier infiniment M^elle BERNINE Nassima pour l'aide qu'elle ma apporté ainsi à sa sœur Cherifa.

Je tiens à remercier les membres de jury pour avoir accepté de jury mon travail.

Je remercie M^lle KHOULALEN, M^r MOHAMED.Y, Mme LEKADIR et Mr MAWLOUD.O pour leurs orientations.

Je remercie également tous mes amis qui m'ont aidé d'une manière ou d'une autre, et toute personne ayant participé, de loin et de prés, à la réalisation de ce travail.

Table de Matières

Liste des Acronymes

- **API** **A**pplication **P**rogramming **I**nterface
- **CORBA** **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- **DII** **D**ynamic **I**nvocation **I**nterface
- **DSI** **D**ynamic **S**keleton **I**nterface
- **GIOP** **G**eneral **I**nter-**O**RB **P**rotocol
- **HTTP** **H**yper**T**ext **T**ransfer **P**rotocol
- **IDL** **I**nterface **D**efinition **L**anguage
- **ImplR** **R**épertoire d'**impl**émentation
- **IFR** **I**nterface **R**epository
- **IIOP** **I**nternet **I**nter-**O**RB **P**rotocol
- **IBM** **I**nternational **B**usiness **M**achines
- **JRMP** **J**ava **R**emote **M**ethode **P**rotocol
- **JVM** **J**ava **V**irtual **M**achine
- **OMG** **O**bject **M**anagement **G**roup
- **ORB** **O**bject **R**equest **B**roker
- **POA** **O**bject **A**dapter
- **RPC** **R**emote **P**rocedure **C**all
- **RMI** **R**emote **M**ethod **I**nvocation
- **RRL** **R**emote **R**eference **L**ayer

- **SII** **S**tatic **I**nvocation **I**nterface
- **SSI** **S**keleton **S**tatic **I**nterface
- **SOA** **A**rchitecture **O**rientée **S**ervices
- **SOAP** **S**imple **O**bject **A**ccess **P**rotocol
- **TCP/IP** **T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol
- **UDDI** **U**niversal **D**escription **D**iscovery and **I**ntegration
- **URI** **U**niforme **R**essource **I**dentifier
- **W3C** **W**ord **W**ild **W**eb **C**onsortium
- **WSDL** **W**eb **S**ervices **D**escription **L**anguage
- **XML** e**X**tensible **M**arkup **L**anguage

Liste des Figures

Liste des Tableaux

Introduction Générale

Depuis son apparition, l'informatique s'est développée massivement et surtout très rapidement. Si les années 80 ont vu la banalisation de l'ordinateur personnel, les années 90 ont permis d'étendre l'utilisation de l'informatique dans tous les domaines.

De nos jours, les systèmes informatiques sont par nature distribués, l'usage d'Internet se démocratise, le commerce électronique est en plein essor, les bourses européennes et mondiales se regroupent en réseaux, etc. Ceci a fait apparaitre différentes plates formes distribuées où les applications développées sur ces plates formes ne sont pas compatibles (systèmes d'exploitation et langages de programmation différents).

Les intergiciels (middlewares) tels que CORBA (Common Object Request Broker Architecture), Java RMI (Java Remote Methode Invocation) et service Web ont été proposés pour simplifier le développement des logiciels et permettre l'interopérabilité entre les systèmes distribués et hétérogènes dans le réseau Intranet ou Internet.

CORBA est un middleware orienté objet proposé par l'Object Management Group (OMG) qui a pour objectif de faire émerger des standards pour l'intégration d'applications distribuées hétérogènes à partir des technologies orientées objet. CORBA est un bus d'objets répartis qui offre un support d'exécution masquant les couches techniques d'un système réparti (système d'exploitation, processeur et réseau) et propose un modèle orienté objet client/serveur pour la coopération entre les applications réparties.

Java RMI est un mécanisme spécifique de l'environnement Java et il ne permet la communication qu'entre objets du langage java, contrairement à la norme CORBA permettant de manipuler des objets à distance avec n'importe quel langage.

Le point commun entre les plates formes précédentes est d'utiliser le réseau Intranet pour la communication en se basant sur des intergiciels offrant des protocoles et outils de communication spécifiques à chaque technologie.

Les services Web ont été conçus pour répondre en premier lieu à des problèmes d'interopérabilité sur l'Internet en utilisant les standards SOAP, WSDL, XML.

Afin de pouvoir choisir la meilleure plate forme pour chaque application, plusieurs travaux ont été réalisés ces dernières années pour la comparaison et l'évaluation des performances de ces plates formes.

Dans la même optique, l'objectif de notre travail est d'évaluer les performances d'une application distribuée sous Java RMI, CORBA et services Web.

Nous organisons notre mémoire en quatre chapitres.

- **Le premier chapitre** présente les notions essentielles liées aux systèmes et applications distribuées et la présentation de l'architecture de chaque plate forme ainsi les protocoles de communication utilisés.

- **Le deuxième chapitre** présente un ensemble d'outils et de techniques utilisées pour l'évaluation des performances des systèmes distribués.

- **Le troisième chapitre** établit un état de l'art sur les différents travaux existants sur la comparaison et l'évaluation des performances des applications distribuées sous les trois plates formes : CORBA, Java RMI et Services Web.

- **Le quatrième chapitre** est consacré à l'évaluation des performances d'une application distribuée à savoir le produit de deux matrices sous les trois plates formes Java RMI, CORBA et services Web. Cette évaluation de performances se base sur la simulation sous l'environnement MATLAB, d'un modèle de files d'attente représentant notre application.

Afin d'enrichir et d'améliorer notre travail nous terminons par une conclusion générale en citant quelques perspectives.

Chapitre I :

Systèmes Distribués

I.1. Introduction

Dans notre vie quotidienne, les systèmes informatiques sont de plus en plus présents. Depuis les architectures centralisées, en arrivant aux architectures réparties, ces systèmes ne cessent pas d'évoluer tant au niveau logiciel que matériel. Les dernières décennies ont été marquées par une révolution dans le domaine de l'informatique distribuée. Le commerce électronique, la recherche d'informations, la gestion de réseaux ou le travail coopératif sont des exemples type d'applications développées sous ces systèmes. Elles sont devenues des outils indispensables pour plusieurs secteurs notamment la finance, l'économie, etc [1].

I.2. Systèmes distribués

I.2.1. Définition

Un système distribué est constitué d'un ensemble d'ordinateurs indépendants reliés par un moyen de communication.
Les machines sont autonomes et les utilisateurs ont l'impression d'utiliser un seul système.

I.2.2. Les caractéristiques des systèmes distribués

La construction des systèmes distribués impose certaines exigences :

- **Hétérogénéité** : un système distribué doit pouvoir gérer l'hétérogénéité concernant le matériel, les systèmes d'exploitations, les langages de programmation [8].

- **Interopérabilité** : Deux systèmes répartis sont interopérables lorsque des entités définies dans l'un et l'autre peuvent interagir afin de coopérer à la réalisation d'un but [9].
- **Sécurité** : elle permet d'assurer le bon fonctionnement des applications et de respecter les droits d'accès aux ressources disponibles sur le système [8].

- **Extensibilité :** un système distribué doit faciliter l'intégration de nouvelles entités dans le système.

I.3. Application distribuée

Une application distribuée est composée de plusieurs parties s'exécutant sur plusieurs ordinateurs et échangeant des messages contenant des données pour mener à bien une tâche [2].

I.3.1. Communication des applications distribuées

I.3.1.1. Architecture client/serveur

Le modèle client/serveur est le modèle le plus utilisé pour la construction d'applications réparties.

I.3.1.1.1 Définition

L'architecture client-serveur est un modèle de dialogue entre deux processus, le premier appelé client qui est le demandeur d'une requête, et le serveur qui est le destinataire de la requête émise. Ce dernier est chargé d'effectuer des traitements pour le compte du client, a partir de données que celui-ci lui envoie. Une fois les traitements effectués, le serveur retourne ses résultats au client [4]. (voir la figure 1).

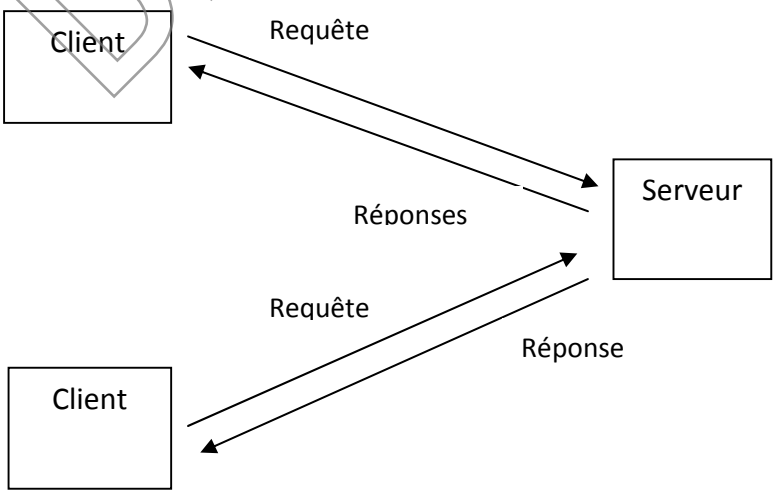

Figure -1- : Architecture client/serveur [4].

I.3.1.1.2. Les différentes architectures

Les architectures client/serveur peuvent être classées comme suit :

- **Architecture à 2 niveaux**

Ce type d'architecture (2-tier) caractérise les environnements client-serveur où le poste client demande une ressource au serveur qui la fournit à partir de ses propres ressources [5].

- **Architecture à 3 niveaux**

Dans cette architecture (3-tier) un niveau supplémentaire est ajouté [5] :

- Un client équipé d'une interface utilisateur chargée de la présentation.
- Un serveur d'application qui fournit la ressource, mais en faisant appel à un autre serveur.
- Un serveur de données qui fournit au serveur d'application les données requises pour répondre au client.

- **L'architecture n-tiers**

Dans l'architecture n-tiers, un processus Serveur peut à son tour devenir le Client d'un autre processus au cours du traitement de la requête. Avec le développement croissant des communications, en particulier du réseau Internet, ce modèle est aujourd'hui largement répandu [6]. (voir la figure -2-).

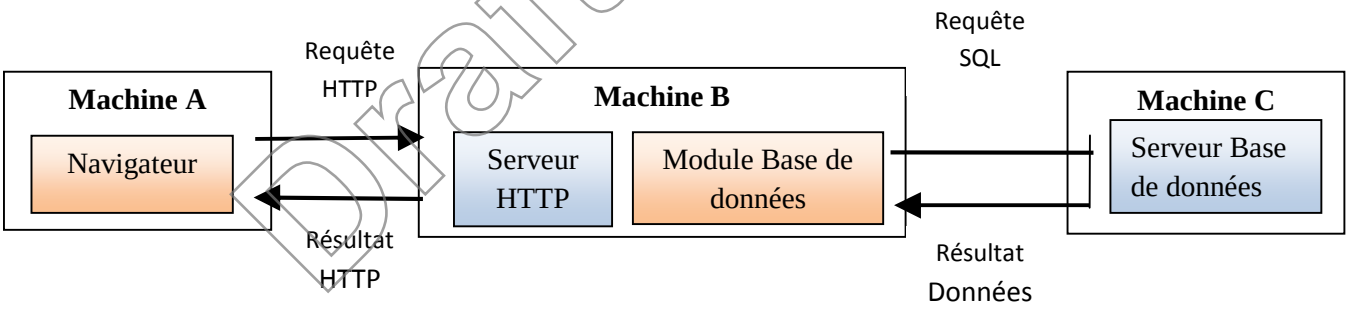

Figure -2- : Interactions dans un modèle n-tiers [6].

I.4. Définitions et concepts objet, composant et service

I.4.1. Objet distribué

Parallèlement à l'arrivée de la programmation structurée, la programmation objet est arrivée début des années 80. [27]

A. **Objet :** un objet est une entité qui encapsule des données. Un objet est identifié de façon unique, et plusieurs instances ou copies d'un même objet peuvent exister [30].

B. **Objets répartis :** un objet réparti est un objet sur lequel il est possible de déclencher à distance (à partir d'un autre nœud du système réparti que celui sur lequel il réside) une des transformations définies sur l'état de l'objet, c'est-à-dire d'invoquer à distance une méthode de l'objet [9].

Pour effectuer un appel de méthode à distance, le nœud appelant doit disposer d'une information identifiant l'objet cible de l'appel au sein de l'application.

C. **Référence distante :** Une référence distance désignant un objet dans une application répartie est une information communicable entre nœuds de l'application, qui désigne sans ambiguïté cet objet, et permet d'effectuer des appels de méthodes à distance [9].

I.4.2. Composant

Le concept de composant logiciel a vu son apparition dans l'ingénierie informatique au début des années 90. Ce paradigme a été introduit après la programmation orientée objet pour dépasser ses limites en terme de décomposition et d'autonomie.

Comme un objet, un composant est une entité logique qui contient des données et qui est capable d'exécuter des opérations (méthodes) sur ces dernières. Les méthodes peuvent être utilisées localement ou à distance comme c'est le cas avec les modèles client-serveur à base d'appel de procédure distante tels que le Remote Procedure Cal1 (RPC) et le Remote Method Invocation de JavaSoft (RMI) [28]

Un composant logiciel est une unité de composition possédant des interfaces spécifiées par contrat et des dépendances contextuelles explicites. Un composant logiciel peut être déployé indépendamment et il est sujet à la composition par un tiers. [29]

A l'exécution, un composant, selon sa granularité, représente un processus, un programme, un composant auquel fait appel un autre composant. [10]

Contrairement à un objet, un composant ne dépend pas d'un langage de programmation, d'un système d'exploitation ou d'une architecture d'ordinateur [28]

I.4.3. Service

Le service est une entité logicielle qui fournit un ensemble de fonctionnalités définies dans une description de service. Cette description comporte des informations sur la partie fonctionnelle du service mais aussi sur ses aspects non-fonctionnels. À partir de cette spécification, un consommateur de service peut rechercher un service qui correspond à ses besoins, le sélectionner et l'invoquer en respectant le contrat qui a été accepté par les deux parties [24].

Les principales caractéristiques d'un service sont décrites comme suit [22] :

- La communication avec un service est basée sur un ensemble de standards et de protocoles ouverts.
- Il est localisé à travers une adresse.
- Il possède une définition précise pour répondre aux besoins des clients.
- Il peut s'exécuter d'une manière synchrone[1] ou asynchrone[2].
- Il possède un propriétaire (fournisseur de service).
- Il met en œuvre une grande flexibilité en fournissant une granularité large des opérations.

I.5. Les intergiciels (middleware)

Un intergiciel est une couche logicielle qui se situe entre le système d'exploitation et l'application. Il fournit des solutions réutilisables à des problèmes fréquemment rencontrés lors de la construction de différentes classes d'applications ; citons par exemple les problèmes relatifs à la gestion des communications, à la fiabilité ou encore à la sécurité des applications distribuées [3].

I.5.1. Les plates formes dépendantes des intergiciels

A. CORBA

L'architecture CORBA (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture) est née en 1991 au sein du consortium de l'OMG[3] (**O**bject **M**anagement **G**roup) afin de définir l'interopérabilité entre différents langages de programmation et différents systèmes d'exploitation. [7]

A.1. Présentation de l'architecture CORBA

CORBA est un bus logiciel qui permet de construire des applications réparties selon le modèle client/serveur entre objets distribués [10].

1 : le client est bloqué en attente d'une réponse à une requête au serveur.
2 : appel non bloquant.
3: un consortium qui regroupe plus de 800 sociétés et organisations dans le monde définit des spécifications pour fournir un modèle de coopération entre objets répartis. [7]

A.2. Modèle client/ serveur orienté objet

Le bus CORBA propose un modèle orienté objet client/serveur pour la coopération entre les applications réparties [11].

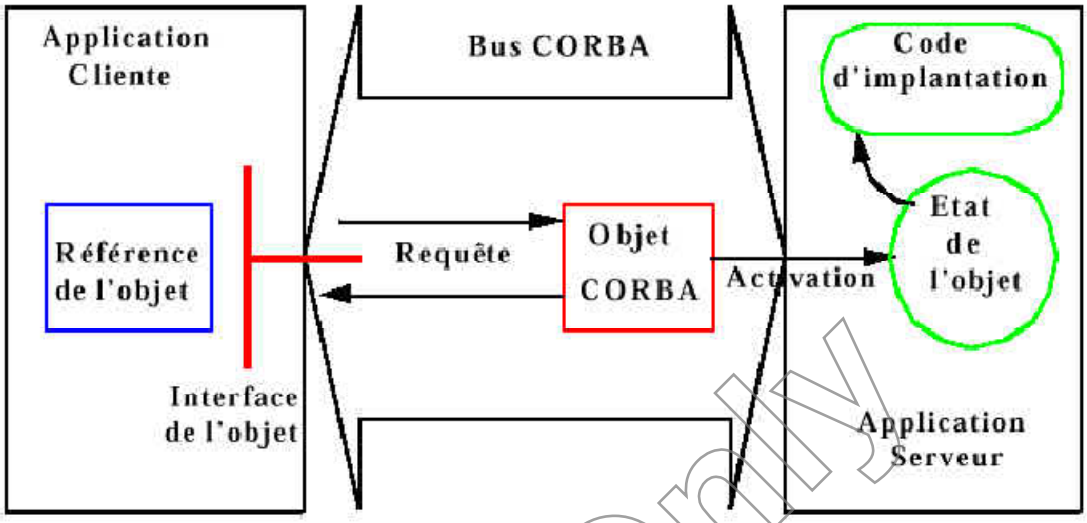

Figure -3- : Notion client/serveur avec le bus de CORBA [11].

La notion client/serveur intervient uniquement lors de l'utilisation d'un objet : l'application implantant l'objet est le serveur, l'application utilisant l'objet est le client. Dans la figure-3- on identifie les éléments suivants :

- **L'application cliente** est un programme qui invoque les méthodes des objets à travers le bus CORBA.
- **La référence d'objet** est une structure désignant l'objet CORBA et contenant l'information nécessaire pour le localiser sur le bus.
- **L'interface de l'objet** est le type abstrait de l'objet CORBA définissant ses opérations et attributs. Celle-ci se définit par l'intermédiaire du langage OMG-IDL.
- **La requête** est le mécanisme d'invocation d'une opération ou d'accès à un attribut de l'objet.
- **Le bus CORBA** achemine les requêtes de l'application cliente vers l'objet en masquant tous les problèmes d'hétérogénéité (langages, systèmes d'exploitation, matériels, réseaux).

- **L'objet CORBA** est le composant logiciel cible. C'est une entité virtuelle gérée par le bus CORBA.

- L**'activation** est le processus d'association d'un objet d'implantation à un objet CORBA.

- **L'implantation de l'objet** est l'entité codant l'objet CORBA à un instant donné et gérant un état de l'objet temporaire. Au cours du temps, un même objet CORBA peut se voir associer des implantations différentes.

- **Le code d'implantation** regroupe les traitements associés à l'implantation des opérations de l'objet CORBA.

- **L'application serveur** est la structure d'accueil des objets d'implantation et des exécutions des opérations.

A.3. Les composantes de CORBA

Les principaux composants de chaque application CORBA sont résumés par la figure suivante [12] :

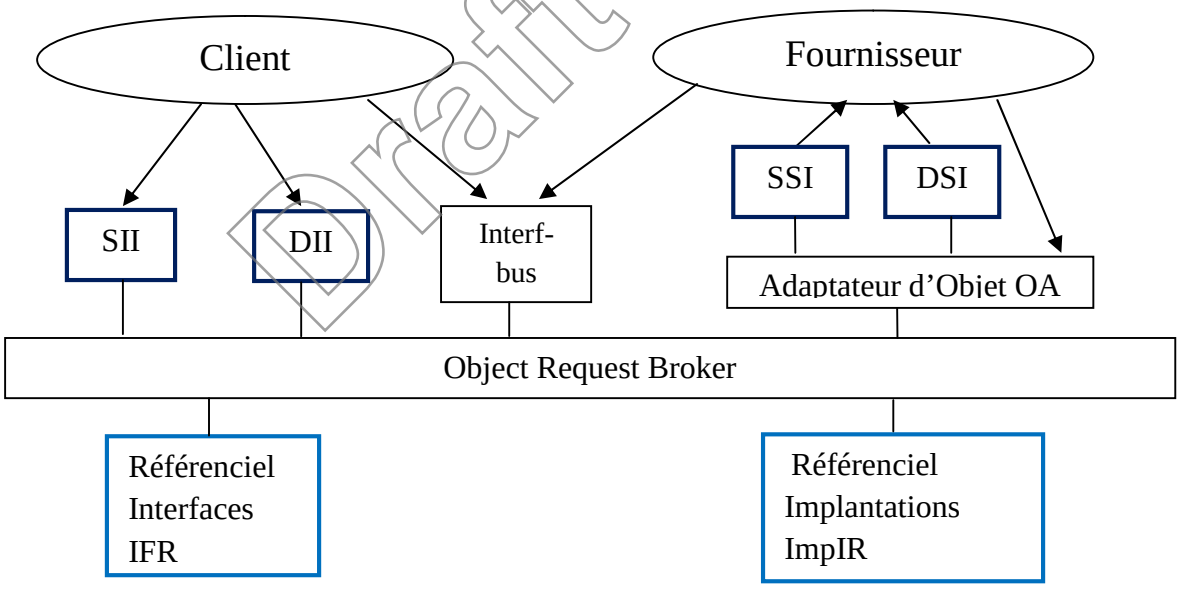

Figure -4- : Les composants de CORBA [12].

- **SII** (**S**tatic **I**nvocation **I**nterface) : est le résultat de la projection des descriptions IDL dans un langage de programmation donné. A travers cette interface, les programmes clients invoquent les opérations des objets de manière transparente.

- **DII** (**D**ynamic **I**nvocation **I**nterface) : permet de construire dynamiquement des requêtes vers les objets. (Requêtes construites seulement à l'exécution).

- **SSI** (**S**keleton **S**tatic **I**nterface) : interface de squelettes statiques qui permet à l'implantation des objets de recevoir les requêtes qui leur étant destinées. Cette interface est générée comme l'interface SII.

- **DSI** (**D**ynamic **S**keleton **I**nterface) : interface de squelettes dynamiques pour les serveurs d'objets est l'équivalent de l'interface d'invocation dynamique pour les clients.

- **L'interface ORB** l'interface commune à tous les ORBs. Elle englobe les fonctionnalités qui sont communes aux clients et aux serveurs comme la conversion de références. Contrairement à l'interface ORB, les autres interfaces (adaptateurs d'objets exclus) sont définies en utilisant IDL.

- **POA** (**O**bject **A**dapter) : est l'adaptateur d'objets qui s'occupe de créer les objets CORBA, de maintenir les associations entre les objets CORBA et d'implantations.

- **ImplR** (**R**épertoire d'**impl**émentation) est le référentiel des implantations qui contient l'information nécessaire à l'activation. Ce référentiel est spécifique à chaque produit CORBA.

- **IFR** (**I**nterface **R**epository) est le référentiel des interfaces qui contient une représentation des interfaces OMG-IDL accessible par les applications durant l'exécution.

A.4. Le bus d'objets répartis CORBA ORB (Object Request Broker)

Le noyau de CORBA est ORB (**O**bject **R**equest **B**roker).
C'est la clé de l'architecture globale de l'OMG, car il prend en charge le transport des requêtes entre tous les objets CORBA. Il donne un environnement d'exécution aux objets en masquant leur hétérogénéité en termes de langages de programmation, de systèmes d'exploitation, de processus et de réseaux.
Il a les caractéristiques suivantes [13]:

- Assurer le transport des requêtes quels que soient les langages de programmation utilisés pour implanter les objets CORBA.
- Assurer la transparence de localisation des invocations.

- Permettre l'invocation statique et dynamique.
- Assurer l'activation automatique et transparente des objets (serveur).
- Assurer l'interopérabilité entre bus CORBA implantés par différents fournisseurs.

A.5. Le langage IDL (**I**nterface **D**efinition **L**anguage**)**

CORBA dispose d'un langage de définition d'interfaces abstrayant les types de données des codes fonctionnels et permettant de décrire les fonctionnalités que peut fournir chaque composant au reste de l'application [14]. IDL est déclaratif, cela signifie qu'il ne fournit aucun détail sur l'implantation. L'IDL fournit des interfaces indépendantes du système d'exploitation et du langage de programmation pour accéder à tous les services et objets qui résident sur un bus CORBA. Cela permet aux objets clients et serveurs écrits dans des langages différents d'interagir à travers des réseaux et entre des systèmes d'exploitation différents [10].

A.6. Les protocoles de CORBA

- **GIOP** (**G**eneral **I**nter-**O**RB **P**rotocol)

C'est un protocole qui permet de mettre les messages qui transitent à travers l'ORB sous un format interopérable et indépendant du protocole de communication sous-jacent. Il définit, pour cela, un ensemble de messages qui sont utilisés, d'une part, par les clients pour envoyer leurs requêtes et, d'autre part, par les serveurs pour retourner leurs réponses [15].

- **IIOP (I**nternet **I**nter-**O**RB **P**rotocol**)**

Ce protocole spécifie comment un message GIOP est mappé afin d'être transmis en utilisant le protocole de communication TCP/IP[1] (Transmission Control Protocol/Internet Protocol). Ainsi, le protocole IIOP est une implémentation de GIOP sur TCP/IP. Son rôle par rapport à l'implémentation du GIOP est similaire à celui d'un langage de programmation par rapport à l'implémentation d'une interface IDL, c'est-à-dire il transforme le message GIOP en un message IIOP lorsque le TCP/IP est utilisé comme protocole de communication [15].

A.7. Les domaines d'applications de CORBA

CORBA de nos jours est utilisé dans de nombreuses applications tel que :
Les applications militaires et aéronautique, télécommunication et transmissions de données, équipements mobiles, transports, contrôle de processus industriel et dans la robotique, dans les applications bancaires, et dans des systèmes de commerce électronique.

1 : Protocole utilisé sur le réseau Internet pour transmettre des données entre deux machines.

B. Java RMI

La plate-forme Java-RMI (**R**emote **M**ethod **I**nvocation) définie par SUN permet l'interaction entre objets Java répartis sur des machines virtuelles différentes.

B.1. Définition

RMI (**R**emote **M**ethod **I**nvocation) est une API[1] (**A**pplication **P**rogramming **I**nterface) de communication entre objets distants de Java. Contrairement à CORBA, RMI ne permet que la communication entre programmes Java à l'exclusion de tout autre langage, par contre, il assure des échanges entre systèmes d'exploitation différents possédant une machine virtuelle Java. RMI s'appuie sur un protocole de type RPC. Il peut être utilisé également pour la communication entre deux machines virtuelles Java sur la même machine physique [7].

B.2. Le langage Java

Java est imposée comme un des meilleurs langages de programmation orienté-objet. Le langage Java, initialement développé par Sun Microsystems dans les années 90. Sa syntaxe inspirée de celle de C et C++. Le modèle de programmation, en particulier en ce qui concerne la gestion mémoire, est plus simple que celui de C++, ce qui rend le développement bien plus rapide. En particulier, le programmeur n'a pas à se soucier des désallocations mémoire, qui sont gérées de façon transparente par un ramasse-miettes qui a pour objectif de déterminer, parmi l'ensemble des objets présents en mémoire, lesquels sont encore vivants et lesquels sont devenus inaccessibles, de façon à recycler l'espace occupé par les objets morts.
De plus, ce succès est dû notamment à la réputation de grande portabilité du code binaire Java (le bytecode), qui n'est pas exécuté par le matériel, mais qui est interprété par une machine virtuelle [16].

B.3. L'architecture de RMI

Lorsqu'un objet instancie sur une machine cliente désire accéder à des méthodes d'un objet distant du serveur, une suite de six opérations (schématisées dans la figure -5-) est lancée [17]:

1. Le client localise l'objet distant grâce à un service de désignation : le registre RMI,

2. Il obtient dynamiquement une image virtuelle de l'objet distant (appelée stub ou souche). Le stub possède exactement la même interface que l'objet distant,

1 : ensemble normalisé de classes, des méthodes ou des fonctions offertes par une bibliothèque logicielle ou un service web.

3. Le stub transforme l'appel de la méthode distante en une suite d'octets (en utilisant la sérialisation) puis les transmet au serveur sous forme de flot de données. On dit que le stub <marshalise> les arguments de la méthode distante,

4. Le skeleton instancie sur le serveur <désérialise> les données envoyée par le stub (on dit qu'il les <démarshalise>), puis appelle la méthode en local,

5. Le skeleton récupère les données renvoyées par la méthode (type de base, objet ou exception) puis les marshalise,

6. le stub démarshalise les données provenant du squelette et les transmet à l'objet qui a fait appel à la méthode distante.

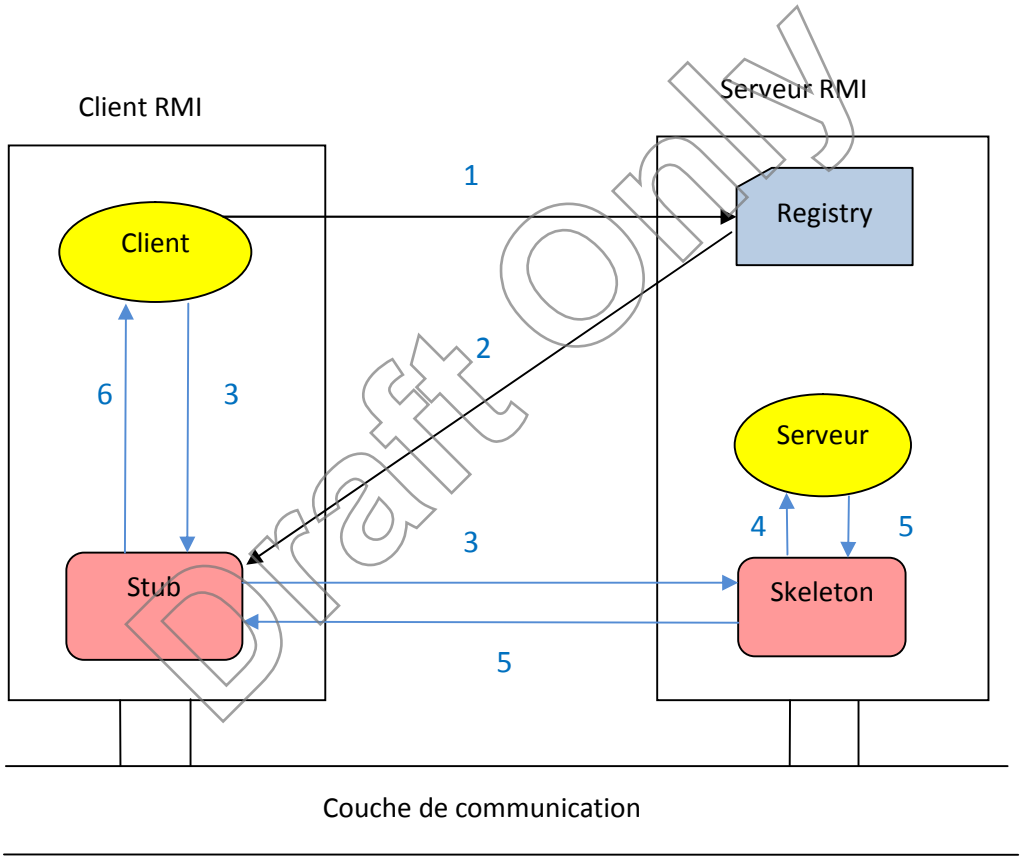

Figure -5- : Architecture RMI. [17]

B.4. L'architecture interne de RMI (Structure des couches RMI)

L'architecture interne de Java RMI est présentée par la figure -6- [18] :

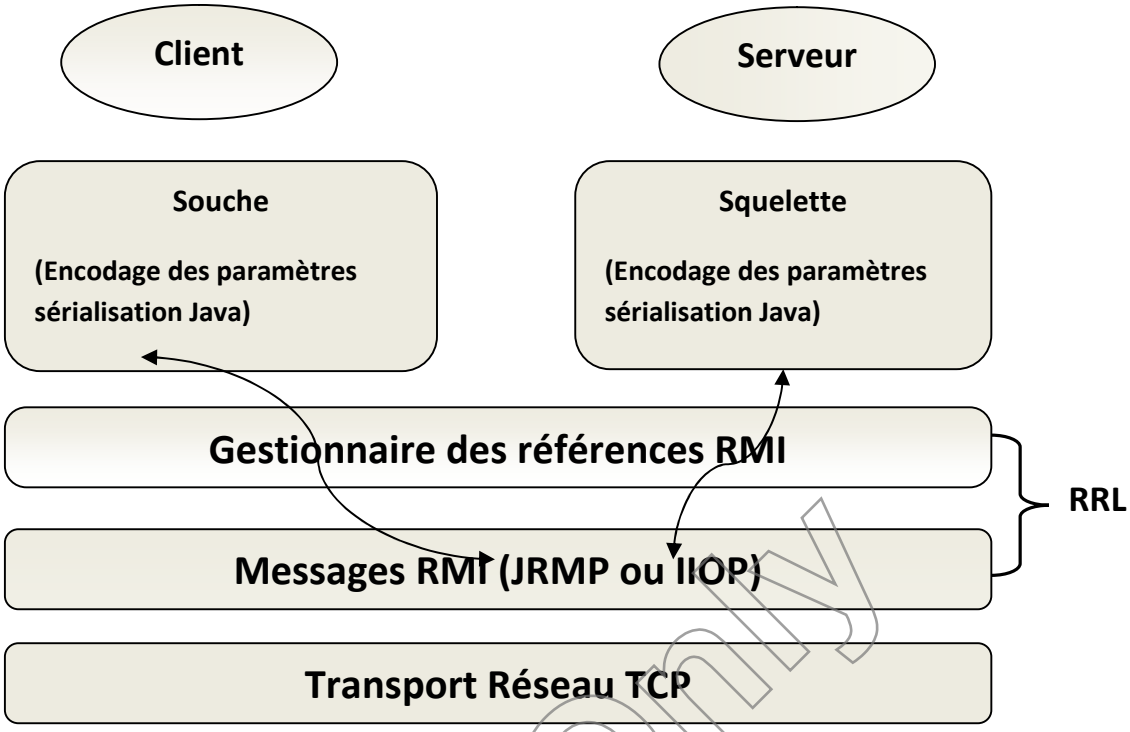

Figure -6- : Architecture interne de RMI [18]

- **Souche/squelette (stub/skelton) :** un objet client ne communiquait pas directement avec un objet serveur, il devait passer par l'intermédiaire d'interfaces. Ce sont ses interfaces qui constituent le niveau souche/squelette. La génération d'interfaces effectuées par le compilateur conduit à la création de ces derniers.

- **Gestionnaire de références (RRL**, **R**emote **R**eference **L**ayer) : il est chargé du système de localisation afin de fournir un moyen aux objets d'obtenir une référence à l'objet distant. Elle est assurée par le package java.rmi.Naming. RMI peut utiliser plusieurs services d'annuaire, il inclut lui même le registre RMI.

- Les connexions et les transferts de données dans RMI sont effectuées par Java sur TCP/IP grâce à un protocole propriétaire **(JRMP J**ava **R**emote **M**ethode **P**rotocol**)** sur le port 1099. A partir de Java 2 version 1.3, les communications entre client et serveur s'effectuent grâce au protocole **RMI-IIOP** (Internet-Orb protocol), un protocole normalisé par OMG.

- **Transport** (Transport Layer) : la couche transport est basée sur les connexions TCP/IP entre les machines. Elle fournit la connectivité de base entre les 2 JVM (**J**ava **V**irtual **M**achine). Elle construit une table des objets distants disponibles. Elle écoute et répond aux invocations.

B.5. Domaine d'utilisation

Java RMI est utilisé pour la mobilité (des cartes à puces (Java Card RMI)), la création des jeux vidéo et des applications de chat.

I.5.2. Les plates formes indépendantes des intergiciels

A. Service Web

La technologie de services Web est aujourd'hui un nouveau paradigme des architectures logicielles pour l'interconnexion des applications sur des machines distantes.

L'une des tendances historiques qui a conduit à l'apparition des services Web est l'utilisation de l'architecture par composants. Ce type d'architecture a engendré un développement rapide et évolutif d'applications distribuées et complexes.

A.1. Définition des services Web

Plusieurs définitions des services Web sont apparues :

W3C (**W**ord **W**ild **W**eb **C**onsortium), le consortium principal des activités du Web, ce groupe établit la définition suivante :

« Un service Web est un système logiciel identifié par un identificateur de ressources URI[1] (**U**niforme **R**essource **I**dentifier), dont les interfaces publiques et les liens sont définis et décrits en XML. Sa définition peut être découverte dynamiquement par d'autres systèmes logiciels. Ces autres systèmes peuvent ensuite interagir avec le service Web d'une façon décrite par sa définition, en utilisant des messages XML transportés par des protocoles Internet ». [19]

IBM donne la définition suivante des services Web :

« Les services Web sont la nouvelle vague des applications Web. Ce sont des applications modulaires, auto contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le Web. Les services Web effectuent des actions simples requêtes à des processus métiers complexes. Une fois qu'un service Web est déployé, d'autres applications (y compris des services Web) peuvent le découvrir et l'invoquer». [20]

Une définition plus précise des services Web est fournie par [21] :

L'auteur définit un service Web comme "une manière standardisée d'intégration des applications basées sur le Web en utilisant les standards ouverts XML, SOAP, WSDL, UDDI et les protocoles de transport de l'Internet. XML est utilisé pour représenter les données,

1 : une courte chaîne de caractères identifiant une ressource sur un réseau (par exemple une ressource Web).

SOAP pour transporter les données, WSDL pour décrire les services disponibles, et UDDI pour lister les fournisseurs de services et les services disponibles ”.

A.2. Les caractéristiques d'un service Web :

Les principales caractéristiques d'un service Web sont [22]:

- Interopérabilité grâce à son indépendance par rapport aux systèmes d’exploitation et des langages de programmations.
- Publiable et accessible en utilisant le langage XML, les protocoles ouverts et les standards du Web.
- Vise à exposer une ou plusieurs fonctionnalités.
- Basés sur le protocole HTTP, les services Web peuvent fonctionner au travers de nombreux pare-feu sans nécessiter des changements sur les règles de filtrage.

A.3. Architecture des services Web

W3C a mis en place une architecture de référence pour la technologie service Web pour proposer un schéma directif et une vision sur le fonctionnement et la dynamique de cette nouvelle technologie [26].

Cette architecture a été proposée afin de promouvoir l'interopérabilité et l'extensibilité des services Web et de préserver ces deux objectifs lors des évolutions technologiques successives. Cette architecture est connue sous le nom SOA (**A**rchitecture **O**rientée **S**ervices) Elle propose une perspective globale pour le développement, la gestion et le fonctionnement des services Web et elle s'articule autour des trois rôles suivants [26]:

- **Le fournisseur :** correspond au propriétaire du service. Il fournit une plateforme d'accueil du service. Il a pour rôle de créer un service, de l'héberger et de le publier pour le mettre à la disposition des clients ou des partenaires.

- **Le client :** correspond au demandeur du service. Il est constitué par l'application qui va rechercher et invoquer un service. L'application cliente peut être elle même un service Web.

- **L'annuaire :** correspond à un registre de descriptions de services. Il offre des facilités de publication de services à l'intention des fournisseurs ainsi que des facilités de recherche de services à l'intention des clients.

Les interactions de base entre ces trois rôles incluent les opérations de transport, de description et de découverte et de transport qui sont assurées respectivement par les standards SOAP, WSDL et UDDI.

Nous décrivons, par la Figure-7-, un scénario type d'utilisation de cette architecture

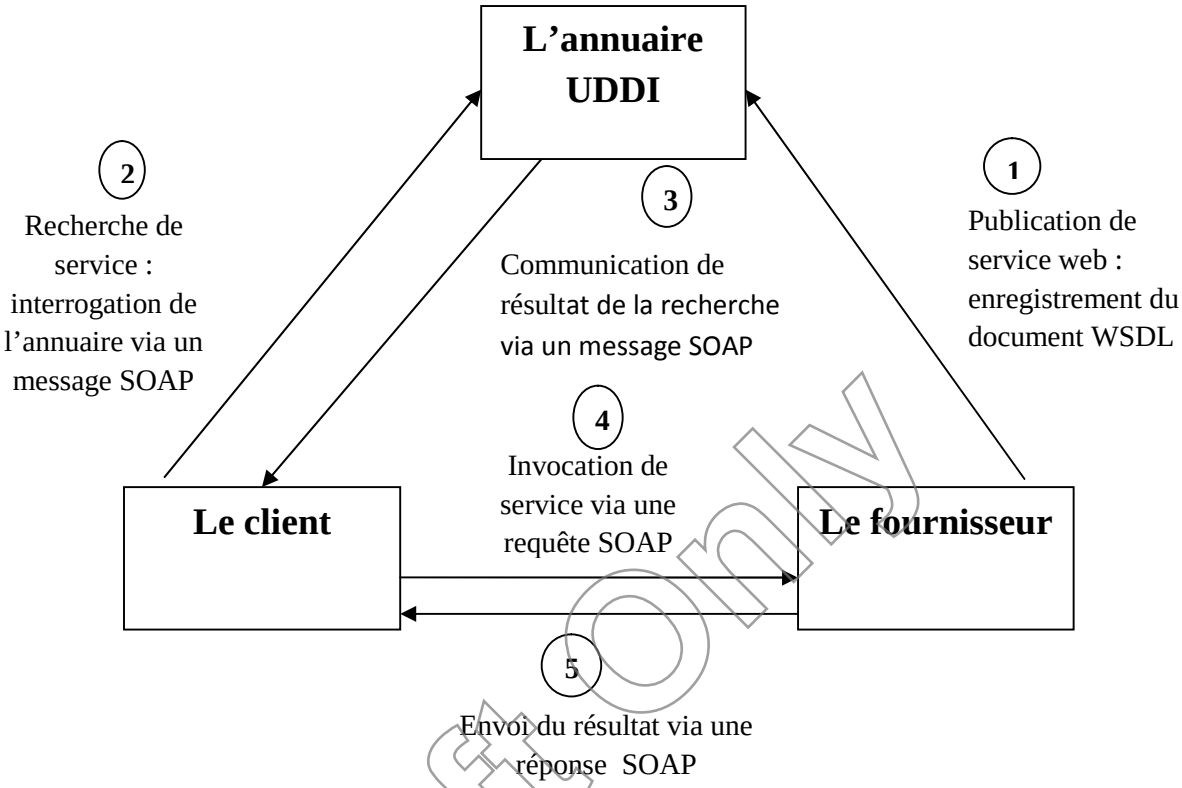

Figure -7- : Architecture de référence des services Web [26].

Ce scénario se déroule en plusieurs étapes qui sont les suivantes :

1. Le fournisseur définit la description de son service dans un document WSDL et la publie dans l'annuaire UDDI.
2. Le client, désirant trouver un service, interroge l'annuaire UDDI via un message SOAP.
3. L'annuaire retourne, via un message SOAP aussi, une liste de services qui répondent à la requête du client. Le client n'a qu'à choisir un parmi la liste.
4. Le client récupère le document WSDL du service choisi. Ensuite, il examine ce document afin de récupérer les informations nécessaires lui permettant de se connecter au fournisseur et d'interagir avec le service considéré. Enfin, il invoque l'opération désirée par le biais d'une requête SOAP renfermant les paramètres d'entrée de l'opération.
5. Le service, du côté du fournisseur, reçoit la requête, la traite, formule la réponse SOAP et l'envoie au client.

A.4. L'infrastructure des services Web : les standards en jeu dans l'architecture

A.4.1. XML (eXtensible Markup Language)

XML est un langage assurant la représentation des données et des documents structurés. Il s'agit d'un langage de balisage générique, sous forme textuelle, défini par le W3C en 1996.

XML est un langage utilisé pour décrire les messages échangés entre applications. Il permet une représentation textuelle des données. Dans les services Web, c'est plus particulièrement XML schéma qui est mis en œuvre et qui permet la description des structures de données en XML [23].

A.4.2. SOAP (Simple Object Access Protocol)

SOAP est un protocole dont la syntaxe est basée sur XML. Son but est de permettre des échanges standardisés de données dans des environnements distribués. Il permet d'accéder à des services distants indépendamment de la plate-forme d'exécution [24].

SOAP a été initialement défini par une initiative conjointe entre Microsoft et IBM en fusionnant XML et HTTP (**H**yper**T**ext **T**ransfer **P**rotocol) dans un protocole commun pour transporter les messages d'une façon structurée et assurer plus d'interopérabilité [22].

Le message SOAP est un document XML présenté sous une forme normalisée. Il est composé d'une enveloppe, contenant un entête (facultatif) et le corps du message (voir la figure -8-) :

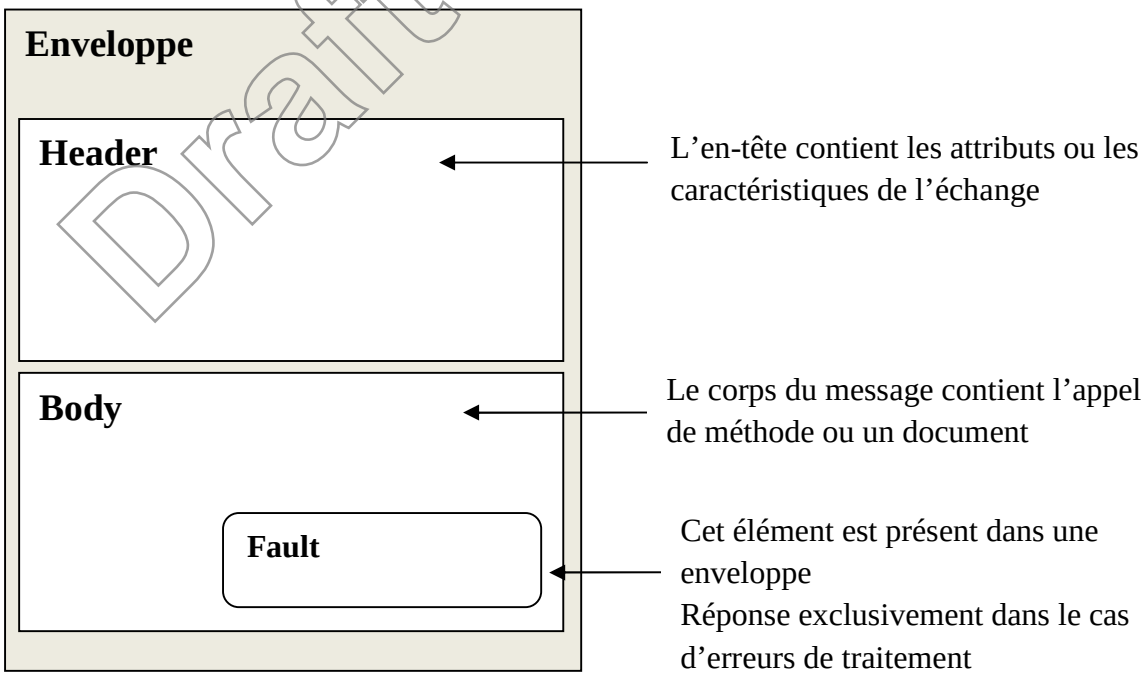

Figure -8- : structure de l'enveloppe SOAP [4]

A.4.3. WSDL (Web Services Description Language)

Le succès des services Web repose en partie sur le faible couplage qui existe entre les consommateurs de services et les fournisseurs de services.

La description dans un langage standard des différentes fonctionnalités du service par le fournisseur de service permet aux consommateurs de s'abstraire des langages de programmation utilisés pour réaliser les services Web. Seules les fonctionnalités du service sont présentées dans le fichier de description ; ainsi les détails techniques propres au fournisseur de service ne sont pas dévoilés aux utilisateurs [24].

La description d'un service Web est faite dans le langage WSDL. Un fichier WSDL comprend une description des fonctionnalités d'un service, mais il ne se préoccupe pas de l'implantation de celles-ci. Le fichier WSDL est un fichier XML qui se divise en deux parties (voir la figure -9-) [24].

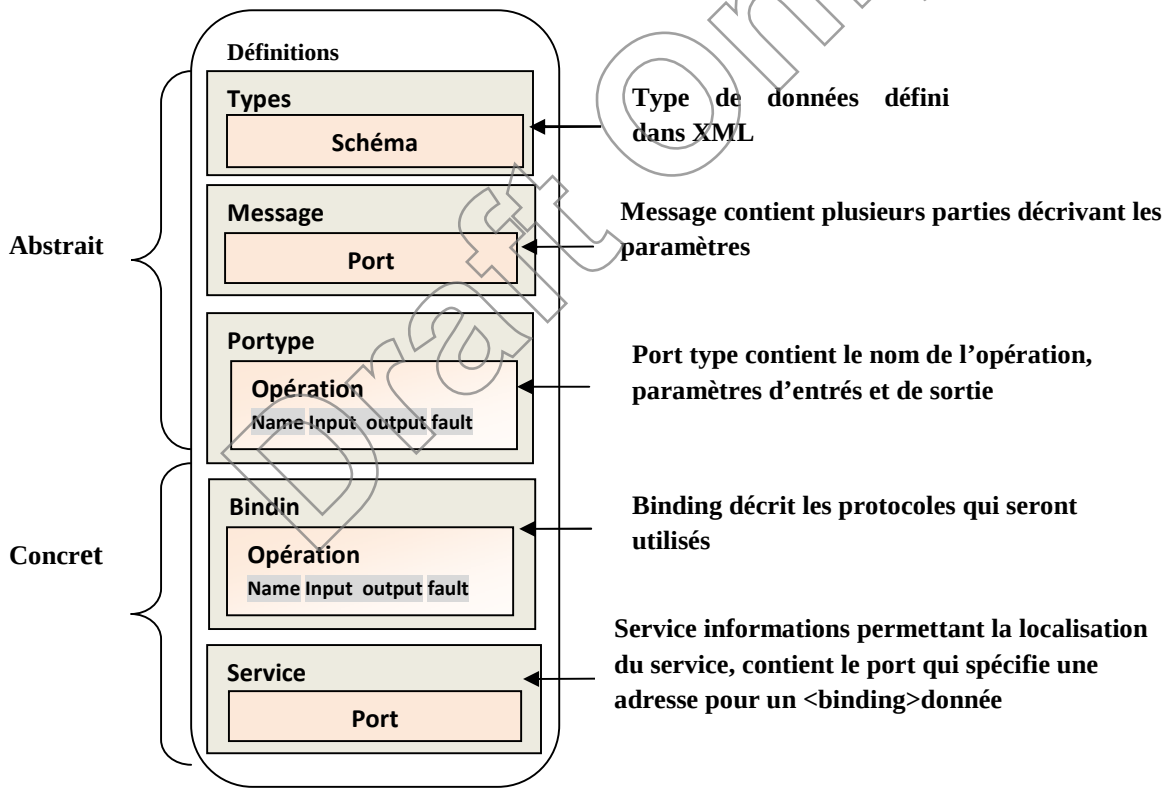

Figure -9- : structure de fichier WSDL d'un service [25].

A.4.4. UDDI (Universal Description Discovery and Integration)

UDDI est un protocole fondé sur la technologie XML et proposé initialement par une collaboration entre Ariba, IBM et Microsoft en 2000. Il permet d'automatiser les communications entre les fournisseurs de service et les clients et de localiser les services disponibles sur internet [22].

UDDI offre un mécanisme fondé sur des normes pour classer, cataloguer et gérer les services Web de tel sorte qu'ils peuvent être découverts et utilisés par d'autres applications [25].

UDDI se comporte aussi comme un service Web qui dispose d'un ensemble d'opérations comme la recherche, l'ajout et la suppression de services. En outre, il permet aux développeurs de découvrir les détails techniques d'un service Web ainsi que les informations métiers à travers le fichier WSDL.

UDDI offre des fonctions regroupées dans les trois composants suivants : les pages blanches, les pages jaunes et les pages vertes (voir la figure-10-).

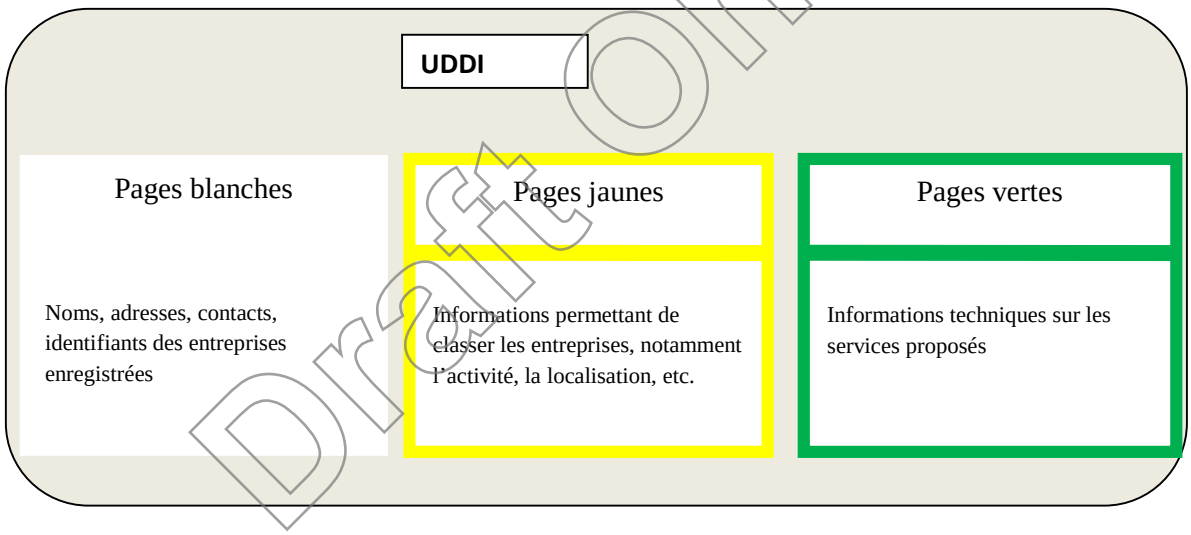

Figure -10- : structure de données de l'annuaire UDDI [18].

- **Page jaunes :** indique ce que fait le service.
- **Pages blanches :** donne des informations sur le fournisseur du service.
- **Pages vertes :** donne des détails sur comment utiliser techniquement le service Web.

A.5. Architecture étendue

Actuellement, SOAP, WSDL et UDDI sont les trois standards qui constituent l'architecture des services Web. Ensemble ils résolvent les problèmes de l'hétérogénéité des systèmes pour l'intégration d'application en ligne [31].

L'architecture étendue est constituée de plusieurs couches se superposant les une sur les autres, d'où le nom de pile des services Web. La figure -11- décrit un exemple d'une telle pile.

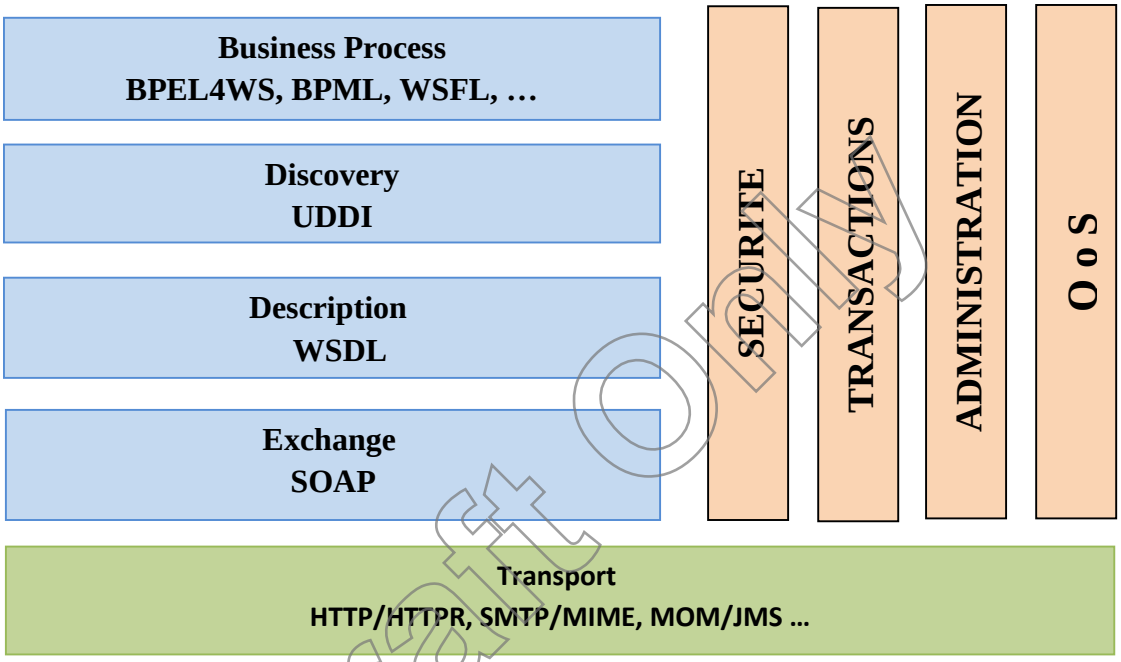

Figure -11- : Architecture en pile (étendue) [31].

A.6. Domaine d'utilisation

Les services Web permettent à une entreprise d'exposer via le réseau Internet ses compétences et son savoir faire ou encore d'ouvrir de nouveaux marchés et de nouveaux supports à la vente.

I.6. Conclusion

Dans ce chapitre nous avons présenté un ensemble de définitions liées aux systèmes et applications distribués ainsi que les architectures des trois plates formes dans lesquelles notre application distribuée sera modélisée afin d'évaluer ses performances. Le chapitre suivant se concentrera sur les concepts d'évaluation de performance pour les applications distribuées.

Chapitre II :

Evaluation des Performances

II.1. Introduction

Les systèmes informatiques et les réseaux de communication (réseau Internet, réseaux mobiles, réseaux ad hoc, réseaux de capteurs,…) ont récemment vu une période de croissance comme jamais dans l'histoire. Leur évolution autorise la mise en œuvre d'architectures parallèles et distribuées de plus en plus sophistiquées, qui intègrent à la fois des traitements répartis et des échanges d'information entre ces traitements. Ces réseaux étant de plus en plus complexes, la modélisation et l'évaluation de performance joue un rôle crucial dans leur processus de conception pour assurer leur déploiement et leur exploitation efficaces dans la pratique.

Les études de performance sont nécessaires pour fournir des réponses aux questions de coût, de performance, de qualité de service et de sécurité, surgissant durant la vie d'un système.

Dans ce chapitre, nous présentons les différentes notions liées à l'évaluation de performances, la modélisation et la simulation.

II.2. Evaluation de performances

Il existe deux approches d'évaluation pour un système, l'approche qualitative et l'approche quantitative.

A. **L'approche qualitative :** consiste à définir les propriétés structurelles et comportementales du système. Il s'agit de vérifier quelques propriétés qualitatives : l'absence de blocage, l'exclusion mutuelle, le comportement fini ou borné…etc.

B. **L'approche quantitative :** concerne le calcul des mesures que l'on veut effectuer sur un système informatique permettant de quantifier ces performances : débit, temps de réponse, taux d'utilisation de ses ressources… etc.

II.2.1. Le rôle de l'évaluation de performances

Le rôle essentiel de toute application informatique est d'accomplir les fonctions pour lesquelles elle a été conçue, et d'offrir une performance adaptée à un coût raisonnable. La performance d'une application est donc un facteur clé de son succès [50].

L'évaluation des performances peut se faire à deux niveaux [32] :

- En conception : le système n'existe pas encore et il s'agit de le créer en respectant le cahier des charges.

- En exploitation : le système existe mais on souhaite le tester ou le modifier de telle manière à améliorer son fonctionnement.

II.2.2. Le concept de l'évaluation de performances

Pour faire de l'évaluation de performances, nous devons disposer de deux éléments :

- **Le système :** c'est l'entité dont on évalue les performances. Il est considéré comme étant un ensemble de ressources partagées entre différentes tâches. La caractéristique commune pour de tels systèmes est la présence d'un temps d'attente pour l'accès à ces ressources partagées [33].
- **La charge :** la charge du système est une description sans ambigüité des inputs qui contiennent suffisamment de détails qui permettront l'évaluation de performances du système considéré [33].

La bonne connaissance de ces deux éléments permet l'évaluation de performances du système.

II.2.3. Les étapes d'évaluation de performances

Les différentes étapes d'évaluation de performances sont [32]:

A) Substituer le système à un modèle que l'on pourra résoudre soit par des méthodes analytiques (mathématiques) soit par des simulations.
B) Appliquer une méthode de résolution ou de simulation (au modèle donné). L'objectif étant d'obtenir les performances du système (temps d'attente, temps de réponse…etc.)
C) Analyser les résultats obtenus par une méthode quelconque.

Le dimensionnement du système sera obtenu en exploitant les résultats par l'intermédiaire des scénarios et en modifiant éventuellement certains éléments du modèle.

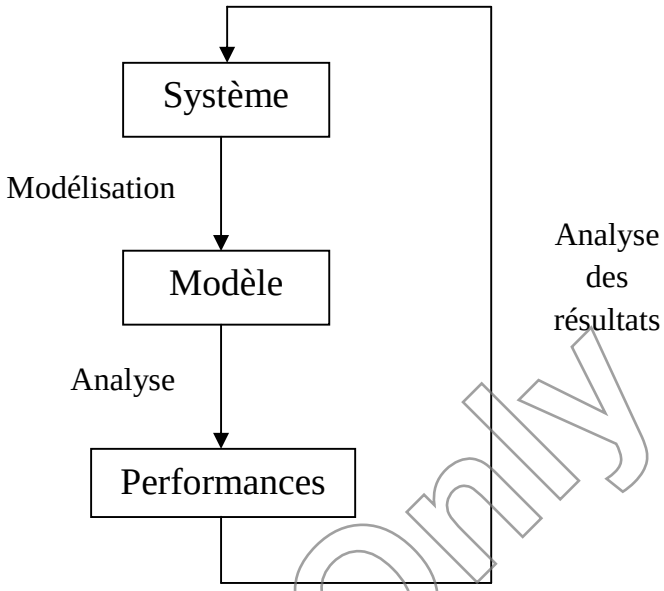

Figure -12- : Etapes de l'évaluation de performances d'un système.

II.2.4. Méthodes et techniques d'évaluation de performances

L'évaluation de performances peut se réaliser par deux manières :

A. Méthodes analytiques

Elles consistent à réduire le système en un modèle mathématique et à l'analyser numériquement.

Un modèle analytique est une relation fonctionnelle entre les paramètres du système et un critère de performance choisi en termes d'équations pouvant être résolues d'une manière analytique [18]. Il existe de nombreux outils mathématiques permettant l'évaluation de performances :

Les réseaux de pétri, Réseaux de files d'attente, les chaines de Markov, les automates…. etc.

B. La simulation

Il s'agit d'implémenter un modèle simplifié du système à l'aide d'un programme de simulation adéquat. Elle présente l'avantage de traduire de manière plus réaliste le comportement du système [32].

C'est une technique largement utilisée pour l'évaluation des performances des systèmes informatiques et réseaux de communications.

II.3. La modélisation

La modélisation est la substitution d'un système réel par un modèle.
En d'autres termes, la modélisation est l'opération par laquelle un modèle d'un phénomène est établi, afin d'en proposer une représentation, interprétable, reproductible et simulable. » [35].

Les réseaux de files d'attente constituent un formalisme de modélisation, largement utilisé pour l'évaluation des performances des systèmes à événements discrets tels que les systèmes informatiques, les réseaux de communication et les systèmes de production.

II.3.1. Les files d'attente

La notion de files d'attente est vu comme un ensemble d'individus, appelés clients, qui viennent, suivant un processus quelconque (le plus souvent aléatoire) acquérir un service auprès d'un autre individu dit serveur. Le processus d'attente, ou la constitution de la file commence à se manifester dés que le taux des arrivées excède le taux de service (on entend par taux, le nombre moyen de clients arrivant ou servis par unité de temps) [33].

Un système de files d'attente peut être représenté par la figure -13-

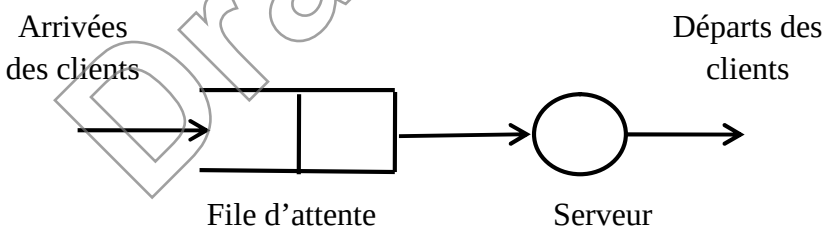

Figure -13- : Représentation graphique d'un système de files d'attente simple [37].

II.3.1.1. La notation de Kendall

Pour identifier une file d'attente, la notation de Kendall est utilisée [38] :

A/S/n/K/D

Où les lettres identifient :

- **A** : la loi du processus des arrivées (la distribution des inter-arrivées) ;
- **S** : la loi du processus de service (la distribution du temps de service) ;
- **n** : le nombre de serveurs ;
- **K** : la taille (capacité) de la file ;
- **D** : la discipline de service.

Les valeurs de A et S sont données par :

- M : loi exponentielle (Markovienne)
- D : loi déterministe ;
- E_K : loi "Erlang-k" ;
- H_K : loi "hyper- exponentielle" d'ordre k ;
- GI : loi générale; les variables successives étant indépendantes ;
- G : loi générale; la suite des variables successives étant stationnaire.

D peut prendre les valeurs suivantes :

- **FIFO** (**F**irst **I**n **F**irst **O**ut) : les clients sont servis dans l'ordre des arrivées (Premier Arrivé Premier Servi) ;
- **LIFO** (**L**ast **I**n **F**irst **O**ut) : le serveur sert le client arrivé le plus récemment (Dernier Arrivé Premier Servi) ;
- **PS** (**P**rocessor **S**haring) : si il y a n > 0 clients dans la file, chacun est servi avec un taux 1/n (processeur partagé) ;
- **Priorités** : il y a plusieurs classes de clients, chacune avec un niveau de priorité. Le client qui a la priorité la plus haute est servi en premier.

II.3.1.2. Files d'attente markoviennes

Ce sont les files d'attente dont le flux des arrivées est Poissonnien (donc les inter-arrivées sont de la loi exponentielle) et la durée de service est exponentielle. La propriété sans mémoire de la loi exponentielle permet de décrire l'évolution du nombre de clients dans le système comme une chaine de Markov [33].

II.3.1.2.1. File d'attente M/M/1

La file M/M/1 est la file la plus simple et la plus utilisée pour modéliser les systèmes informatiques. L'utilisation de cette file est motivée par l'ensemble de ses résultats permettant de déterminer les paramètres de performances moyens. Elle est définie par le processus stochastique suivant [35] :

- le processus d'arrivée des clients est distribué selon un processus de Poisson de paramètre λ.
- le processus de temps de service est indépendant du processus d'arrivée et suit la loi exponentielle de paramètre μ.

- un seul serveur.

Une file d'attente M/M/1 est représenté comme suit :

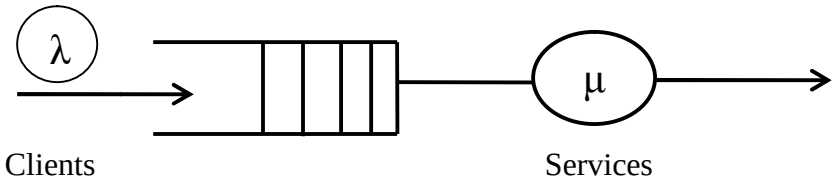

Figure -14- : File d'attente M/M/1 [39].

II.3.1.2.2. La file M/M/S

Dans un système d'attente M/M/S, le processus d'arrivée des clients suit toujours la loi de Poisson de taux λ. C'est un système à S serveurs et les S serveurs sont indépendants et la loi de service est exponentielle de paramètre μ. La capacité de la file est infinie et la discipline de service est FIFO [47].

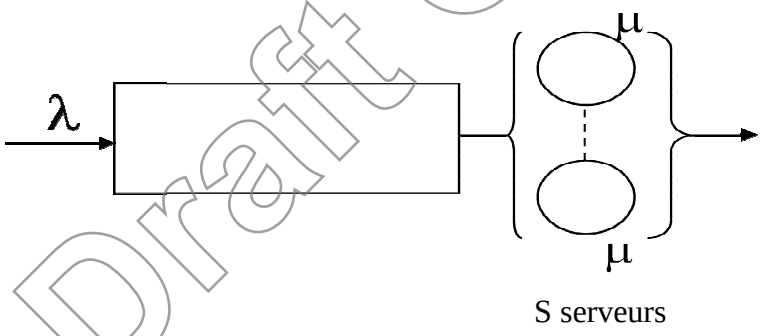

Figure -15- : Modèle de file d'attente M/M/S [47].

II.3.1.3. La file d'attente M/G/1

La file M/G/1 est une file à capacité illimitée ayant un seul serveur. Son processus d'arrivée suit la loi poissonienne de taux λ tandis que le temps de service est distribué selon une variable aléatoire générale G. Dans ce cas particulier, il est possible de déterminer tous les paramètres de performances moyens, même si son service ne vérifie pas la propriété sans mémoire. Ces paramètres sont déterminés grâce à la méthode de la chaîne de Markov induite.
Cette méthode consiste à ramener l'étude à une chaîne de Markov à temps discret en considérant des instants d'observation particuliers (instants de début de service ou instants de fin de service) [35].

II.3.2. Réseaux de files d'attente

Un réseau de files d'attente est composé d'un ensemble de stations de service et d'un ensemble de clients. Ce système est caractérisé par le processus représentant l'arrivée des clients aux réseaux, le temps de service des clients à la station, le cheminement des clients d'une station à l'autre et les disciplines de services des clients au niveau de chaque station (voir la figure -16-).

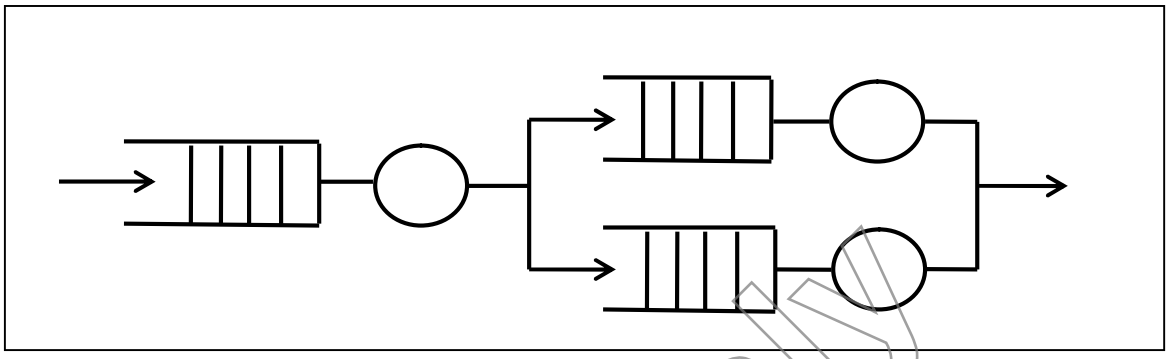

Figure -16- : Exemple de réseau de files d'attente. [36]

II.3.3. Les processus stochastiques

Un processus stochastique décrit l'évolution temporelle de l'état d'un système à l'aide de variables aléatoires et de lois de probabilités.

Un processus stochastique {Xt, t € T} est une collection de variables aléatoires indexées par un paramètre t et définies sur un même espace de probabilités. Le paramètre est généralement interprété comme le temps et appartient à un ensemble ordonné T.
Il existe deux classes de processus stochastiques en fonction de l'ensemble T :
• processus stochastique à temps discret si T C **N** (entiers naturels).
• processus stochastique à temps continu si T C **R**+ (réels positifs).

II.3.4. Chaines de Markov

Une chaine de Markov est un modèle fournissant un outil simple de modélisation et d'analyse de performances d'une classe particulière de système à événements discrets. [32]

Un processus stochastique vérifiant la propriété suivante est appelé un processus de Markov ou processus markovien. [51]

La propriété : pour toute séquence d'instant $t_{n+1} > t_n > t_{n-1} > t_{n+2} > \ldots > t_0$ et tout sous ensemble d'états I € S, il est vrai que

Prob $\{Xt_{n+1} \in I \mid Xt_n = i_n, Xt_{n-1} = i_{n-1}, Xt_{n-2} = i_{n-2}, \ldots, X_{t0} = i_0\} = Prob\{Xt_{n+1} \in I \mid Xt_n = i_n\}$.
L'évolution de l'état d'un système peut être modélisée par un processus markovien si, pour tout instant t, l'état courant X_t résume, à lui seul, tout l'historique du système susceptible d'influencer son évolution future [54].

II.3.5. Les réseaux de Petri

Les réseaux de Pétri constituent un outil mathématique de modélisation développé au début des années soixante par le mathématicien allemand Carl Adam Petri. Les réseaux de Petri décrivent des relations existantes entre des conditions et des événements et ils modélisent le comportement des systèmes dynamiques à événements discrets.
Les réseaux de Petri présentent des caractéristiques intéressantes à savoir le parallélisme, la synchronisation, le partage des ressources,… [52]

II.3.5.1. Définition des réseaux de Petri [53]

A. Réseaux de Petri simples : On appelle Réseau de Petri simple places-transitions le quadruplet Q=<P;T;I;O> où [53] :

- P est un ensemble fini non vide de places;
- T est un ensemble fini non vide de transitions;
- $P \cap T = \emptyset$;
- I(Ti) représente l'ensemble de places d'entrée de la transition Ti (transitions d'entrée) ;
- (Ti) représente l'ensemble de places qui sont en sortie de la transition Ti (sotie des places).

B. Réseaux de Petri marqués : On appelle Réseaux de Petri marqués, le couple R=<Q;M_0>, où M_0 est le marquage initial du réseau et tel que la fonction de marquage M est définie de **P** dans l'ensemble des entiers naturels [53].

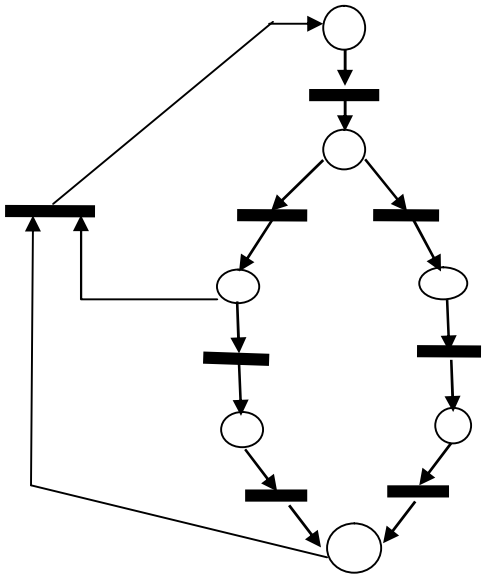

Figure -17- : Un réseau de Petri comportant 7 places, 6 transitions et 15 arcs orientés. [53]

II.3.6. UML

UML (**U**nified **M**odeling **L**anguage), que l'on peut traduire par "langage de modélisation unifié) est une notation permettant de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existant auparavant, et est devenu désormais la référence en termes de modélisation objet. [57]

UML est à présent un standard défini par l'Object Management Group (OMG). La dernière version diffusée par l'OMG est UML 2.4.1 depuis août 2011. [5]

Le langage UML est défini sous forme d'un méta-modèle, organisé en plusieurs paquetages. Chaque paquetage introduit des concepts que nous pouvons exprimer par des notations graphiques à travers des diagrammes. Ces diagrammes peuvent être regroupés en deux classes principales : celle relative à la modélisation structurelle et celle relative à la description du comportement [56].

L'objectif d'UML en proposant autant de diagrammes et concepts est de fournir un langage de modélisation généraliste qui couvre une grande partie du domaine du logiciel.
Cependant, il arrive parfois que ce langage ne soit pas adapté à capturer certaines propriétés particulières à un domaine spécifique. C'est le cas par exemple des systèmes temps réel, pour lesquelles il est nécessaire de spécifier des propriétés temporelles (échéance, période, etc.), ce qui n'est pas supporté par la notation standard d'UML [56].

II.4. La simulation

II.4.1. Les étapes de la simulation

Les différentes étapes à suivre pour faire une simulation d'un système sont [55] :

1. **Formulation du modèle :** cette étape consiste à identifier et analyser le problème, en déterminant ses composantes, leurs relations et les frontières entre le système et son environnement.
2. **Elaboration du modèle :** cette étape consiste à extraire un modèle aussi fidèle que possible du système réel.
3. **Identification du modèle et collecte de données :** la collecte de données est indispensable pour l'estimation des paramètres du modèle. Ceci requiert une connaissance des méthodes statistiques et des tests d'hypothèses.
4. **Validation du modèle :** cette étape consiste à évaluer les performances du modèle puis les comparer à celles du système réel.
5. **Exécution de la simulation :** pour permettre à l'épreuve le modèle. Le concepteur doit effectuer plusieurs exécutions et recueillir les résultats.
6. **Analyse et interprétation des résultats :** une fois les résultats obtenus, le concepteur passe à l'analyse et l'interprétation de ces résultats pour donner des recommandations et des propositions.
7. **Conclusion :** cette dernière étape consiste à évaluer les perspectives d'exploitation du modèle pour d'autres préoccupations.

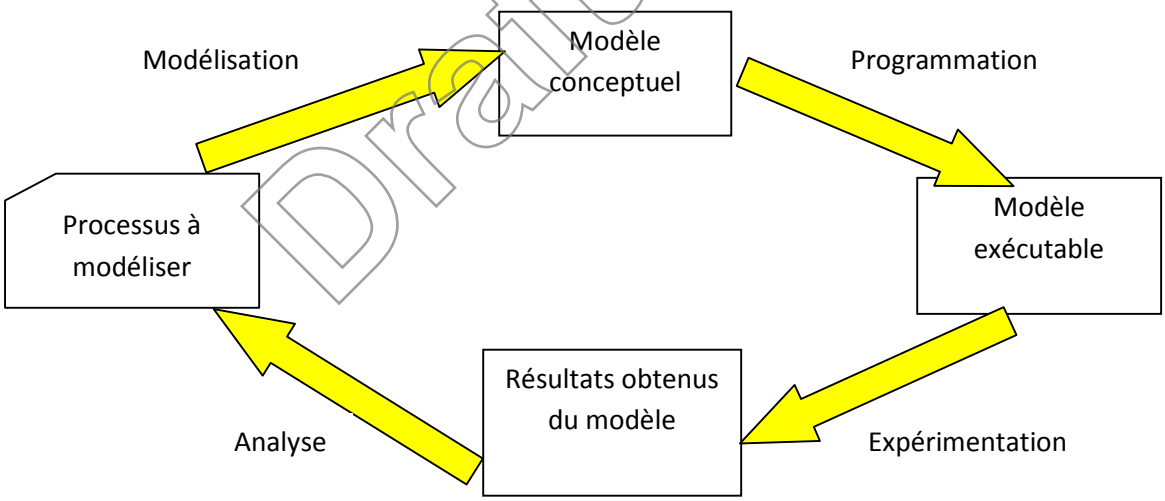

Figure -18- : Processus simplifié de simulation. [34]

II.4.2. Les techniques de simulation

On distingue trois techniques de simulation :

A. Simulation à événements discrets

Cette approche représente typiquement la simulation d'un modèle de files d'attente qui est dirigé par une séquence de nombres aléatoires (ou semi-aléatoires) dont la distribution est spécifiée par l'utilisateur. Ces séquences aléatoires sont utilisées pour obtenir les temps de réponse, les probabilités de branchement (routage,…).

Ce type de technique a été largement utilisé pour l'évaluation de performances des ordinateurs. La simulation à événements discrets représentant une expérimentation statistique, les résultats fournis nécessitent une interprétation statistique effectuée avec soin. [55]

B. Simulation conduite par trace

Cette technique doit être utilisée avec une grande précaution. Elle est très appropriée quand on veut faire des comparaisons entre deux alternatives dans le même environnement.

Si le système existe et s'il peut être observé, les entrées du modèle à simuler peuvent provenir d'échantillons obtenus directement du système par des techniques de mesure au lieu d'employer des valeurs aléatoires. Cela peut donner un peu plus de certitude quant aux résultats obtenus. [49]

C. Simulation continue

Si le modèle du système contient en grande partie des variables continues qui évoluent dans le temps, ce modèle conduit à un système d'équations différentielles. Fréquemment, il se trouve que le système d'équations ne peut pas être résolu analytiquement. On utilise alors la simulation dite continue, où des combinaisons de méthodes numériques essaient d'aboutir à la solution du modèle. [49]

II.4.3. Domaines d'application de la simulation

Les domaines d'application de la simulation sont nombreux et variés, nous trouvons, entre autres: [48]

- Concevoir et analyser des systèmes industriels
- Evaluer du hardware et du logiciel pour un système d'exploitation d'ordinateur
- Déterminer des politiques d'ordonnancement pour un système de production
- Concevoir des systèmes de communications et leurs protocoles
- Evaluer des politiques de gestion pour les organisations de service
- Analyser des systèmes financiers ou économiques.

II.5. Conclusion

Dans ce chapitre, nous avons présenté les outils d'évaluation de performances des systèmes informatiques ainsi que le concept de modélisation et de simulation. Dans le chapitre suivant, nous présenterons les différents travaux existants concernant l'évaluation des performances des plates distribuées Java RMI, CORBA et service Web.

Chapitre III :

Etat de l’Art sur les Travaux Existants

II.1. Introduction

Ce chapitre présente un état de l’art sur les différents travaux existants sur la comparaison et l’évaluation de performance des applications distribuées sous trois plates formes : CORBA, Java RMI et Services Web.

Dans la littérature plusieurs travaux ont été réalisés et nous nous focalisons sur les travaux présentés ci-dessous.

III.2. Article1: Evaluation de CORBA et les services Web dans les Applications distribuées [40]

Cet article présente une étude comparative entre les deux technologies pour le développement des systèmes distribués CORBA et les services Web.
La comparaison est faite par la mise en œuvre de deux applications :

•Une simple opération arithmétique.
• L'envoi d'une chaîne de caractères.

Ces deux applications ont été choisies car elles sont simples pour la mise en œuvre et elles sont la base des transactions et des calculs complexes. Les mesures ont été effectuées dans un laboratoire des essais pour chacune des technologies en isolement, de sorte que le processeur, la mémoire et le temps de latence dans les messages n'ont pas été interrompues par des processus autres que cette expérience.

Les tests ont été réalisés sur un réseau LAN, et les mesures suivantes doivent être effectuées: l'espace mémoire utilisé par les clients, utilisation du processeur client et serveur, le temps de latence à une simple requête et aux multiples requêtes, le nombre total d'octets transférés et le nombre de paquets.
Les résultats sont présentés dans les tableaux suivants :

Technologie	Temps de latence total en secondes	Nombre total de paquets	Nombre total d'octets
Service Web	0.11 s	16	3338
CORBA	0.5 s	8	1111

Tableau -1- : Résultats de la latence et du trafic durant une simple requête.

Exemple	Technologie	Total de packages	Total en octets
Opérations arithmétiques élémentaires	Service Web	48,931	10,360,814
	CORBA	10,007	1,400,851
Chaîne passage	Service Web	55,617	16,053,312
	CORBA	10,007	2,236,661

Tableau -2- : Analyse du trafic sur le réseau.

Exemple	Opérations arithmétiques élémentaires		Chaîne passage	
Technologie	CPU client	CPU serveur	CPU client	CPU serveur
Service Web	15.0 s	6.0 s	22.8 s	8.4 s
CORBA	2.3 s	0.8 s	4.0 s	1.1 s

Tableau -3- : CPU et mémoire utilisées.

Les auteurs concluent que pour les environnements locaux (intranet), la solution la plus viable est l'utilisation de CORBA, car elle offre les meilleures performances et avec moins en terme d'utilisation d'espace mémoire, traitement et la bande passante. Maintenant, si les besoins des systèmes distribués sont orientés vers l'utilisation de l'Internet, les services web est la solution qui donne les meilleurs résultats, et leur scalabilité et flexibilité sont supérieurs à CORBA.

III.3. Article 2 : L'efficacité du traitement parallèle distribué en utilisant Java RMI, les sockets et CORBA [41]

Les auteurs de cet article évaluent les performances des protocoles de communication sockets, RMI et CORBA en réalisant une application de tri.

Trois programmes ont été mis en œuvre de manière cohérente en utilisant les fonctionnalités offertes par chacun des sockets, RMI et CORBA. Les données ont été distribués à partir du client à chacun des plusieurs serveurs, Les serveurs traitent les données et retournent les résultats au client.

La recherche évalue les performances des sockets, RMI et CORBA en utilisant 2, 4 et 8 serveurs sur des gammes de 20.000 à 400.000 entiers de données. Les serveurs font tout le travail alors que le client distribue et reçoit des données depuis et vers les serveurs. Tous les tests ont été exécutés exactement en mêmes conditions pour chacun des protocoles de communication. Le tableau suivant résume les Résultats.

communication et le temps de calcul									
Nombre serveurs			8		4			2	
Données	Socket	RMI	CORBA	Socket	RMI	CORBA	Socket	RMI	CORBA
20000 entiers	1295 s	1270 s	1500 s	2441 s	1616 s	1652 s	4373 s	2305 s	2548 s
40000 entiers	2524 s	2547 s	2951 s	4698 s	3301 s	3225 s	8171 s	4776 s	5054 s
60000 entiers	3788 s	3823 s	4732 s	7294 s	4900 s	4817 s	12271 s	7173 s	7583 s
80000 entiers	4998 s	5103 s	5800 s	9748 s	6511 s	6327 s	16357 s	9242 s	10032 s
100000 entiers	6285 s	6357 s	7313 s	12284 s	8101 s	6379 s	23232 s	11460 s	12529 s
200000 entiers	13184 s	12954 s	14870 s	24681 s	15910 s	16128 s	46181 s	23506 s	24829 s
400000 entiers	25251 s	25491 s	29934 s	49108 s	31543 s	32251 s	95167 s	46220 s	49616 s

Tableau -4- : Communication et le temps de calcul des sockets, RMI et CORBA.

D'après les résultats les auteurs ont conclut que CORBA est toujours plus long que RMI et sockets en moyenne de 17%.

Les auteurs ont conclut que Java rend la programmation très facile avec les sockets mais les données doivent être sérialisées et transférées comme un paquet.

RMI et CORBA nécessitent d'enregistrer les méthodes avec un serveur de noms, pour RMI, le rmiregistry doit être exécuté sur chaque serveur et pour CORBA le serveur de nommage doit être seulement exécuté sur le réseau.

Pour la facilité de programmation, Java est un très bon choix. Les sockets nécessitent plus d'instructions à exécuter à chaque fois un envoie ou un recevoir est fait. RMI est facile à programmer et nécessite le temps de configuration de la communication, mais il est seulement de Java pour Java.

CORBA est plus difficile à programmer mais il offre la possibilité de communiquer entre les applications mises en œuvre dans différents langages. CORBA exige le moindre accord des IDL et subit une baisse de performance de cette généralité.

III.4. Article 3 : Performance des middlewares Java RMI, JAXRPC et CORBA [42]

Le but de l'auteur de cet article est de montrer l'impact de la nature des données de réponse sur la performance des technologies middleware Java (Java RMI, JAXRPC et CORBA).
Pour cette étude, l'auteur a utilisé trois sortes de serveur qui se caractérisent chacun par le type de données renvoyé où l'auteur a testé la performance avec une simple application d'invocation d'un serveur.
Le premier serveur définie une opération getString () car l'effet de la longueur de la chaîne sur la performance était un problème, la performance a été testé avec des chaînes de différentes tailles dans la gamme de 64 à 4096 caractères.
Les résultats obtenus sont présentés par le tableau suivant :

Technologie	La taille de la chaine retournée	Paquets (10,000 appels)	Nombre total d'octets transférés (10.000 appels)	Le temps stabilisé pour 1000 appels	Portion données envoyées/ données de contenu réels
CORBA	64	20,034	3,374,400	1.29 s	5.3
	512	20,034	7,854,420	1.50 s	1.5
	4096	80,032	48,434,269	2.86 s	1.2
RMI-JRMP	64	20,100	2,412,400	0.77 s	3.8
	512	40,130	8,404,000	1.19 s	1.6
	4096	100,129	48,345,000	2.04 s	1.2
RMI-HTTP	64	40,965	7,948,300	11.42 s	12.4
	512	40,968	12,438,400	11.85 s	2.4
	4096	61,198	49,390500	13.57 s	1.2
JAXRPC	64	100,816	19,868,900	4.84 s	31.0
	512	100,816	24,348,800	5.04 s	4.8
	4096	130,869	61,830,000	7.03 s	1.5

Tableau -5- : Les performances des différentes technologies pour des tailles de chaines de caractères de taille différentes.

Le tableau suivant présente les données de performance pour l'exemple où le serveur retourne un petit tableau de chaines simples :

Technologie	**La taille du tableau**	**Paquets (10,000 appels)**	**Nombre total d'octets transférés (10.000 appels)**	**Temps stabilisé pour 1000 appels**
CORBA	4	28,264	13,161,991	1.9 s
	32	130,397	88,930,600	4.1 s
RMI-JRMP	4	51,679	15,431,012	2.3 s
	32	293,246	99,885,950	5.8 s
RMI-HTTP	4	41,071	18,857,000	13.4 s
	32	91,668	89,463,665	18.3 s
JAXRPC	4	130,835	72,015,000	14.0 s
	32	427,584	437,890,943	78.5 s

Tableau -6- : Performance des technologies pour un serveur qui retourne un petit tableau de chaines.

Une version simplifiée des solutions du système de client / serveur a été exécuté pour la JAXRPC et RMI-HTTP. Les réponses du serveur sont représentées par des tableaux de différentes tailles. Les résultats sont indiqués dans le tableau suivant :

Technologie		La taille du tableau= 1	La taille du tableau= 2	La taille du tableau= 4	La taille du tableau= 8	La taille du tableau= 16
RMI-http	Temps	1.3 s	1.36 s	1.41 s	1.5 s	1.66 s
	Nombre total d'octets de données	2,663,145	2,977,242	3,607,242	4,973,958	7,597,547
JAXRPC	Temps	0.8 s	1.0 s	1.31 s	2.2 s	3.8 s
	Nombre total d'octets de données	7,106,609	9,860,018	15,401,886	27,087,637	48,912,368

Tableau -7- : Performance de JAXRPC et Java RMI pour des structures de données complexes.

D'après les résultats de comparaison des middlewares Services Web (JAXRPC) et RMI-http et CORBA montrent également que la performance relative dépend des types de données de réponse. Java-RMI est plus efficace en ce qui concerne le traitement des paquets individuels que la plupart des implémentations CORBA. Il est possible qu'une solution de CORBA

s'exécute mieux dans la pratique qu'une solution Java-RMI. L'auteur conclut que les Services Web n'effectuent pas systématiquement mieux que RMI- http si les données de réponse sont sous la forme d'un tableau ou d'une séquence de structures, ou d'un serveur qui récupère des données à partir d'une base de données. Le volume de données, et par conséquent le nombre de paquets transférés augmente considérablement. En ces circonstances, RMI- HTTP peut être plus rapide qu'une solution Service Web.

III.5. Article 4: Les services Web peuvent s'exécuter aux côtés de CORBA et RMI? [43]

De point de vue des développeurs, les services Web peuvent être considérés comme un nouveau modèle de calcul distribué, à partir de ce point de vue, l'auteur de cet article a écrit un benchmark[1] comparant XML-RPC avec CORBA et RMI.

L'auteur a créé des méthodes de service qui sont définies par une interface Java, où chaque méthode de service correspond à une mise en œuvre simple d'une charge de travail. Chaque charge de travail est exécutée plusieurs fois, pour saisir le temps d'exécution.

Le tableau suivant montre les valeurs moyennes et montre également dans les trois colonnes les plus à droite, les relations entre les trois technologies :

[ms]	CORBA	RMI	JAXRPC	JAXRPC / CORBA	JAXRPC / RMI	CORBA / RMI
Connexion	2 445,8 ms	793,0 ms	649,2 ms	3,1 ms	3,8 ms	1,2 ms
Temps max	1 051.2 ms	62.0 ms	15.8 ms	17.0 ms	66.5 ms	3.9 ms
Temps min	14.4 ms	1.2 ms	0.9 ms	12.4 ms	15.2 ms	1.2 ms

Tableau -8- : Valeurs moyennes et les relations.

Les relations montrent clairement que même dans des conditions optimales (temps min), JAXRPC est encore 15 fois plus lent que Java RMI (JAXRPC/RMI temps min = 15.2 ms) et 12 fois plus lent que CORBA (JAXRPC/CORBA temps min = 12.4 ms). Dans le cas pire (temps max) JAXRPC est 66 fois plus lent que RMI et 17 fois plus lent que CORBA.

L'auteur a calculé le temps de configuration de connexion sur chaque plate forme et le nombre d'appels de service qui peuvent être effectués par seconde comme le montre le tableau suivant :

1 : Benchmark désigne une série d'essais effectués sur une machine ou un système afin d'en qualifier les performances à des fins de comparaisons.

Connexion et appel de service par secondes	JAXRPC	CORBA	RMI
Connexion	0.4 s	1.3 s	1.5 s
Temps max	1.0 s	16.1 s	63.3 s
Temps min	69.3 s	857.0 s	1056.6 s

Tableau -9- : Les connexions et les appels par seconde.

L'auteur a conclut qu'il existe de nombreux arguments en faveur de l'utilisation des services Web, mais la performance n'est pas l'un d'eux.

XML-RPC est un très bon candidat pour étudier et comparer à CORBA lorsque l'interopérabilité et l'accès facile des opérations commerciales sont les éléments clé de l'entreprise. D'autre part, si la performance comme une latence faible, un grand nombre d'appels de service par unité de temps et une large bande passante sont les aspects clés pour une application d'entreprise, XML-RPC n'est pas le premier choix.

III.6. Article 5 : Java RMI, tunnel RMI et Service Web : Comparaison et analyse de performances [44]

Les auteurs dans cet article comparent entre Java RMI, tunnel RMI[1] et les services Web selon deux approches : la première est de mesurer l'impact de pare-feu sur l'exécution des applications réparties qui doivent communiquer à travers ce dernier et analyser les scénarios d'exécution des différentes technologies, alors les auteurs ont mesuré les temps de réponse à une invocation à distance des méthodes et les temps d'instanciation. La deuxième approche est de mesurer le temps de réponse à une invocation pour huit types de données différents : int, short, long, float, double, boolean, byte, string afin de comprendre comment les différents types de données influencent sur le résultat.

Pour la mise en œuvre des approches, ils ont utilisé deux ordinateurs, le premier agissant autant que client et le deuxième autant que serveur. Les auteurs ont effectué un nombre d'essais pour obtenir les résultats souhaités, pour chaque essai, mille invocations ont été mesurés pour obtenir la résolution nécessaire.

D'abord les auteurs ont mesuré l'exécution locale pour tous les scénarios, où tous les essais ont été exécutés sur le même ordinateur (le client et le serveur se trouvent sur la même machine) pour éviter le réseau.

1 : une méthode qui utilise RMI pour faire des appels à travers un pare-feu local.

Les résultats sont montrés par le tableau suivant :

Technologies / Types De données	**RMI**	**http to port**	**http to servlet**	**Services Web**
Instanciation	**0,686 ms**	**19,070 ms**	**19,483 ms**	**0,616 ms**
La moyenne des Types simples	**0,157 ms**	**5,052 ms**	**7,663 ms**	**2,336 ms**
Int	0,157 ms	5,066 ms	7,659 ms	2,330 ms
Short	0,156 ms	5,061 ms	7,689 ms	2,356 ms
Long	0,156 ms	5,050 ms	7,622 ms	2,319 ms
Float	0,157 ms	5,052 ms	7,680 ms	2,375 ms
Double	0,161 ms	5,066 ms	7,689 ms	2,342 ms
Boolean	0,155 ms	5,059 ms	7,670 ms	2,316 ms
Byte	0,155 ms	5,050 ms	7,630 ms	2,313 ms
String	**0,172 ms**	**5,105 ms**	**7,658 ms**	**2,353 ms**

Tableau -10- : Résultats des exécutions locales.

Pour inclure le temps de communication et des échanges, les auteurs ont répété les essais sur deux ordinateurs reliés (le client et le serveur se trouvent sur des ordinateurs distincts).

Technologies / Types De données	**RMI**	**http to port**	**http to servlet**	**Services Web**
Instanciation	**0,600 ms**	**17,283 ms**	**23,352 ms**	**0,620 ms**
La moyenne des Types simples	**0,251 ms**	**9,051 ms**	**7,117 ms**	**2,224 ms**
Int	0,254 ms	9,112 ms	7,305 ms	2,189 ms
Short	0,248 ms	9,050 ms	7,053 ms	2,225 ms
Long	0,250 ms	9,039 ms	7,048 ms	2,224 ms
Float	0,252 ms	9,042 ms	7,067 ms	2,247 ms
Double	0,248 ms	9,042 ms	7,125 ms	2,253 ms
Boolean	0,250 ms	9,042 ms	7,108 ms	2,222 ms
Byte	0,261 ms	9,033 ms	7,114 ms	2,208 ms
String	**0,261 ms**	**9,078 ms**	**7,138 ms**	**2,206 ms**

Tableau -11- : Résultats d'exécution à distance.

Les auteurs ont conclut que les résultats offerts par Java RMI sont meilleurs que les résultats offerts par les autres technologies car RMI utilise la sérialisation binaire, les services Web, d'autre part, passent beaucoup plus de temps pour la sérialisation et la désérialisation XML, qui est utilisée pour créer des messages SOAP. Pour les deux alternatives tunneling RMI : HTTP-to-port et HTTP-to-servlet la raison pour la dégradation des performances est qu'ils nécessitent beaucoup de communications supplémentaires (routage, transmission de requêtes, marshaling/ demarshaling, le codage / décodage des messages JRMP binaires).

Les résultats du tableau11 sont légèrement différents des résultats obtenus dans l'exécution locale, le tunnel http-to-servlet est maintenant considérablement plus rapide que http-to- port. La raison peut être trouvée dans le fait que http-to- port fonctionne presque 80% plus lent une fois utilisé en réseau.

Une analyse détaillée des résultats d'exécution en réseau et en local montre la dégradation considérable d'exécution dans le scénario du réseau pour les technologies qui ne sont pas basées sur les services Web.

Cet article a traité des paramètres qui n'ont pas été déjà abordés dans les articles précédents.

III.7. Article 6 : Comparaison des Protocoles de Communication dans les Environnements Distribués [45]

Les auteurs de cet article ont comparé les protocoles binaires RMI et JMS au protocole non-binaire XML-RPC en termes de temps de réponse et les données transférées sur le réseau.

Les auteurs ont construit une application distribuée en utilisant les protocoles mentionnés ci-dessus et de mesurer la quantité de données transférées et le temps de réponse. L'application distribuée construite est d'effectuer la multiplication de matrices en divisant le calcul entre le nombre de nœuds et la différence entre les trois implémentations est le moyen où les données sont transférées entre les nœuds du réseau distribué (maître et esclaves).

Pour la conception de l'application le maître est chargé de générer les deux matrices d'entrée à multiplié et il prend en entrée la taille de la matrice et le nombre d'esclaves à utiliser pour calculer la matrice résultante et il calcule le temps total nécessaire pour multiplier les deux matrices. Par contre l'esclave est responsable de l'exécution de la multiplication réelle des matrices et après le calcul, chaque esclave renvoie sa colonne de résultat au maître.

Le tableau ci-dessous montre la quantité de données transférées pour une matrice de taille (1000 x 1000) fonctionnant sur un nombre variable d'esclaves pour les trois protocoles.

Nombre d'esclaves	RMI	JMS	XML-RPC
1	9 Mo	62 Mo	21.3 Mo
2	12.8 Mo	62 Mo	27 Mo
3	16.7 Mo	62 Mo	36.7 Mo
4	20.6 Mo	62 Mo	47.3 Mo
5	24.4 Mo	62 Mo	57.9 Mo
6	28.4 Mo	62 Mo	68.5 Mo

Tableau -12- : Quantité de données transférées.

Les auteurs ont tracé deux graphes à partir des données collectés :

Dans le premier graphe la taille de la matrice est constante et le nombre d'esclaves varie. Ils ont enregistré le temps de réponse en seconde.

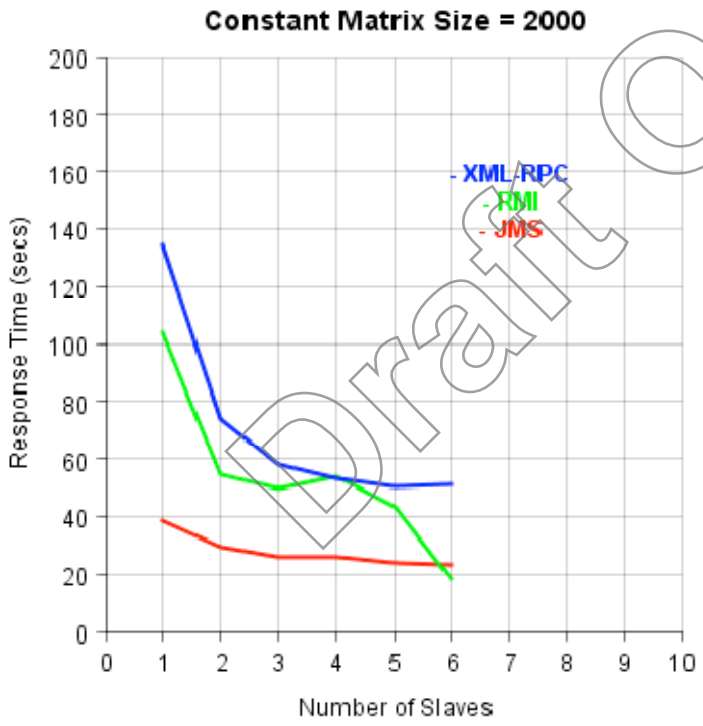

Figure -19- : Le graphe du temps de réponse en variant le nombre d'esclave.

Le graphe montre que les protocoles sont performants avec l'augmentation du nombre d'esclaves. Comme le nombre de nœuds augmente, la quantité de communication augmente.

Dans le deuxième graphe, le nombre des esclaves constant à 6 et la taille de la matrice varie.

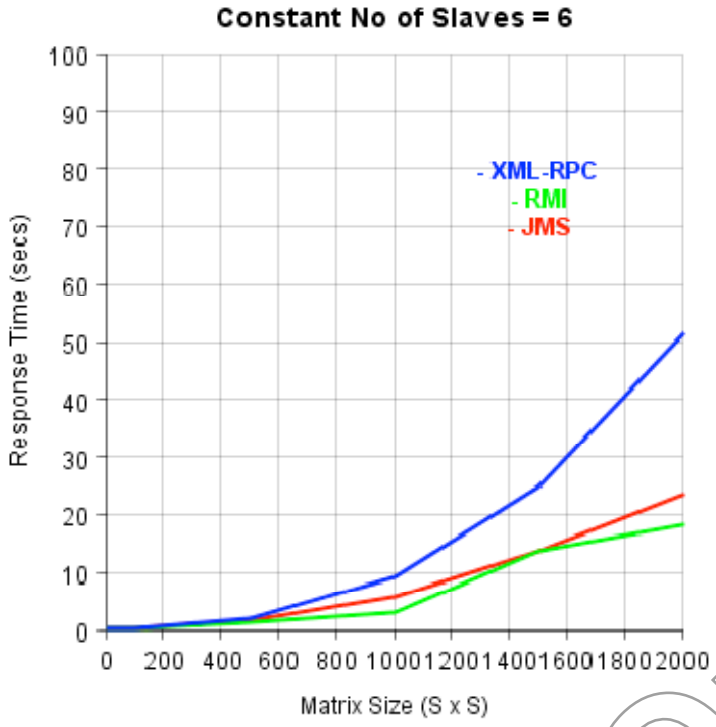

Figure -20- : Le graphe du temps de réponse en variant la taille de la matrice.

Les graphes montrent que le temps de réponse pour les protocoles binaires où les données sont transférées sous forme sérialisée (RMI et JMS) est inférieur au temps de réponse du protocole non binaire XML-RPC.

III.8. Article 7 : Choix obligatoire de temps- déploiement pour améliorer la performance des applications distribuées [46]

Dans cet article les auteurs proposent une architecture générale, dite Bleu Pencil, afin d'optimiser le temps de déploiement des applications distribuées à savoir le protocole SOAP des services Web et RMI.

Les auteurs examinent les performances des deux plates formes RMI et service Web en utilisant deux implémentations différentes de la plate forme J2EE. La première, JBoss Application Server, qui est une implémentation Open Source, et la seconde WebSphere Application Server qui représente une version commerciale.

Les auteurs étudient la charge de travail en termes de temps de réponse (en milliseconde) et de débit (requêtes /secondes) de deux services Web avec les caractéristiques différentes. Le premier concerne le service Web de Google qui permet de rendre un ensemble de pages résultats pour des mots clés spécifiques. Le résultat est paramétré par le nombre de pages générées afin de varier la quantité de données transférée. Le deuxième concerne les services Web d' IBM WSBench qui modélisent des transactions financières. Les résultats obtenus sont groupés dans deux tableaux différents (tableau 11 et 12).

Le tableau suivant montre les résultats obtenus dans le cas JBoss Application Server.

Application	Paramètres	Temps de réponse (msec)					Débit (Requêtes / Sec)				
		SOAP	RMI (JRMP)	RMI (IIOP)	**SOAP/RMI**		SOAP	RMI (JRMP)	RMI (IIOP)	**RMI/SOAP**	
					JRMP	**IIOP**				**JRMP**	**IIOP**
Google	10 pages	1232.4	122.6	48.0	**10.0x**	**25.7x**	1.62	561.8	515.5	**346x**	**318x**
	50 pages	4946.8	486.3	241.9	**10.2x**	**20.4x**	0.40	216.9	213.7	**542x**	**534x**
	100 pages	9322.7	895.2	454.1	**10.4x**	**20.5x**	0.21	146.0	116.6	**680x**	**555x**
WSBench	10KB	5801.6	286.3	/	**20.3x**	/	0.34	137.0	/	**397x**	/
	50KB	27735.4	2539.3	/	**10.9x**	/	0.07	38.4	/	**549x**	/
	100KB	62953.8	7697.7	/	**8.2x**	/	0.03	16.0	/	**503x**	/

Tableau -13- : Performance de SOAP et RMI pour JBoss Application Server.

À partir de ce tableau, les auteurs remarquent que la technologie RMI est plus rapide que SOAP.

En ce qui concerne l'environnement WebSphere Application Server, les auteurs ont obtenu les résultats suivants :

Application	Paramètres	Débit (Requêtes / Sec)			Débit (Requêtes / Sec)		
		SOAP	RMI (IIOP)	**SOAP/RMI-IIOP**	SOAP	RMI (IIOP)	**RMI-IIOP /SOAP**
Google	10 pages	227.6	84.7	**2.69x**	219.7	590.3	**2.96x**
	50 pages	599.7	276.5	**2.17x**	83.4	180.8	**2.17x**
	100 pages	1185.0	516.2	**2.30x**	42.2	96.9	**2.30x**
WSBench	10KB	313.5	452.2	**0.69x**	159.5	110.6	**0.69x**
	50KB	981.2	1768.1	**0.52x**	54.5	28.3	**0.52x**
	100KB	1734.1	3476.9	**0.50x**	28.8	14.4	**0.50x**

Tableau -14- : Performance de SOAP et RMI pour WebSphere Application Server.

Contrairement aux résultats précédents, dans le cas de l'environnement WebSphere Application Server, la performance de protocole SOAP est meilleure que dans le cas JBoss en termes de temps de réponse et le débit malgré l'augmentation de la charge de travail. Ceci est dû à l'utilisation des technologies accélérant l'analyse des documents XML et améliorant le protocole SOAP.

Les auteurs ont conclut que les caractéristiques de performance de SOAP et RMI dépendent fortement d'un environnement de serveur d'application particulier et ils ont conclut que les techniques d'optimisation qui tiennent compte à la fois des informations de configuration et les caractéristiques de la charge de travail fournit un grand avantage.

III.9. Tableau comparatif

D’après la comparaison des travaux précédents le point commun entre les différents travaux étudiés est d’établir une comparaison entre les plates formes distribuées Java RMI, CORBA et services Web à base d’évaluation de performances des applications exécutées sous ces plates formes.

Après l’analyse des articles précédents le tableau -15- résume la différence entre ces travaux selon les différents paramètres de performance ainsi que le type d’application utilisée.

Les paramètres de performances sont :

- Temps de réponse : désigne la durée d’exécution d’une opération dans le système.
- Latence : le délai qu'il y a entre une requête d'un client et la réponse du serveur.
- Type de données : définit les valeurs que peut prendre une donnée : int, float, boolean, string, … etc.
- Nombre d’octets transférés en une unité de temps.
- Nombre de paquets transférés en une unité de temps.
- Impact de pare-feu : l’effet du pare-feu sur les données.
- Débit : la quantité d'information traitée ou transmise durant un intervalle de temps donné.
- Type de données de réponse : le type qu’une réponse peut prendre par exemple un tableau.
- Charge de travail : ensemble des contraintes subies au cours d'une tâche déterminée.
- Espace mémoire utilisé par le client ou le serveur.
- Temps configuration de connexion : le temps nécessaire pour configurer une connexion.
- Nombre appels de service : le nombre d’appels entre le client et le serveur.

		Article1	Article2	Article3	Article4	Article5	Article6	Article7
La méthode utilisée	Exécution des applications	X	X	X	X	X	X	X
Paramètre des performances	Temps de réponse		X	X	X	X	X	X
	Temps de latence	X						
	Types de données		X	X		X	X	
	Nombre d'octets transférés	X		X			X	
	Nombre de paquets transférés	X		X				
	L'impact de pare-feu					X		
	Débit							X
	Type de données de réponse			X				
	Charge de travail				X			X
	Espace mémoire utilisé	X						
	Temps configuration de connexion				X			
	Nombre appels de service				X			
Les types d'applications utilisées	Invocation à distance de méthodes					X		
	Google[1]							X
	WSBench[2]							X
	Invocation d'un serveur			X				
	Opération arithmétique	X						
	Envoi chaine de caractères	X						
	Benchmark[3]				X			
	Application de tri		X					
	Multiplication de matrice						X	
La plate forme la plus performante	Java RMI		X	X		X		
	Web Service	X			X		X	X
	CORBA	X						

Tableau -15- : Tableau comparatif des travaux d'évaluation de performances des plates formes distribuées (CORBA, Java RMI et Service Web).

1 : moteur de recherche sur internet.

2 : Web service d'IBM qui modélise les transactions financière

3 : désigne une série d'essais effectués sur une machine ou un système afin d'en qualifier les performances à des fins de comparaisons.

III.9. Conclusion

Dans ce chapitre nous avons présenté les différents travaux existants sur la comparaison et l'évaluation de performances des applications distribuées sous les trois plates formes Java RMI, CORBA et services Web et l'analyse de ces travaux nous a permis de dégager un ensemble de critères essentiels pour l'évaluation de performances de notre application distribuée sous les trois plates formes.

Chapitre IV :

Simulation et Analyse des Performances de Java RMI, CORBA et services Web

IV.1. Introduction

Dans ce chapitre nous allons présenter l'application distribuée de produit de deux matrices et son exécution sur les trois plates formes CORBA, Java RMI et service Web ainsi que la modélisation de l'application sous les trois plates formes et nous présentons les résultats de la simulation.

IV.2. Plan de travail

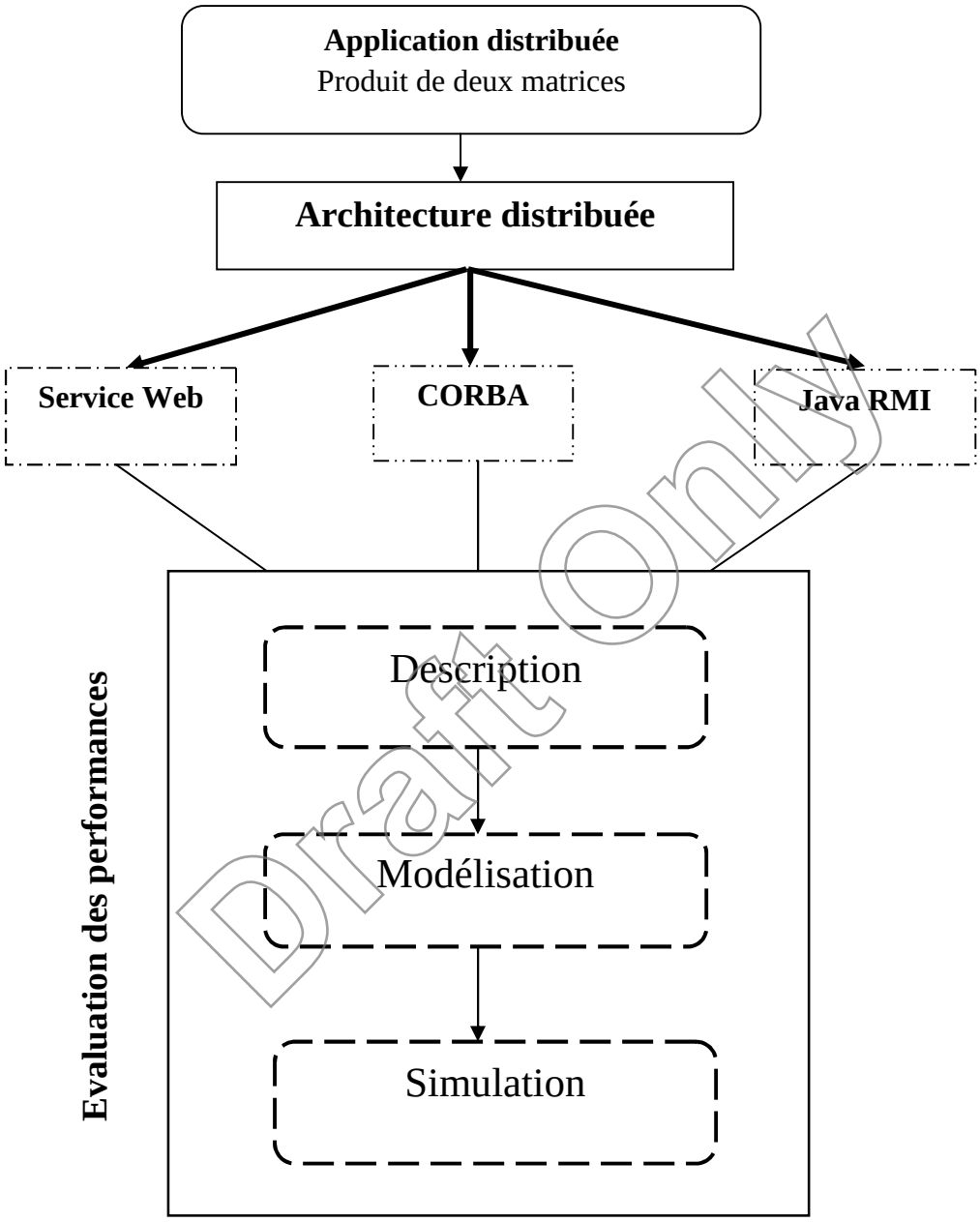

Figure -21- : Démarche du travail.

IV.3. Présentation de l'application

Nous présentons dans cette partie notre application distribuée via Internet qui permet de calculer le produit de deux matrices avec le principe client / serveur.

- Le produit matriciel n'est définit que si le nombre de colonnes de la première matrice dans la multiplication doit être égal au nombre de ligne de la deuxième matrice.

le produit de la matrice **A** (n × m) par la matrice **B** (m × p) est la matrice **C** (n × p) telle que l'élément C_{ij} est égal au produit scalaire de la ligne i de la matrice **A** par la colonne j de la matrice **B**.

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \qquad i = 1..n \quad j = 1..p$$

- **Exemple :**

$$\begin{bmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{bmatrix} \bullet \begin{bmatrix} 5 & 1 \\ 2 & 3 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 9 & 7 \\ 23 & 9 \end{bmatrix}$$

On a en effet, en effectuant les produits ligne par colonne :

$$\begin{bmatrix} 1 & 2 & 0 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \\ 3 \end{bmatrix} = 1 \times 5 + 2 \times 2 + 0 \times 3 = 9 \qquad \begin{bmatrix} 1 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} = 1 \times 1 + 2 \times 3 + 0 \times 4 = 7$$

$$\begin{bmatrix} 4 & 3 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \\ 3 \end{bmatrix} = 4 \times 5 + 3 \times 2 - 1 \times 3 = 23 \qquad \begin{bmatrix} 4 & 3 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} = 4 \times 1 + 3 \times 3 - 1 \times 4 = 9$$

Le client envoie la requête (les deux matrices d'entrée) au serveur en lui demandant le service et il attend le résultat.

Le serveur reçoit les données, met le client en attente du résultat et commence le traitement (le calcul de produit de deux matrices). Voir la figure -22- .

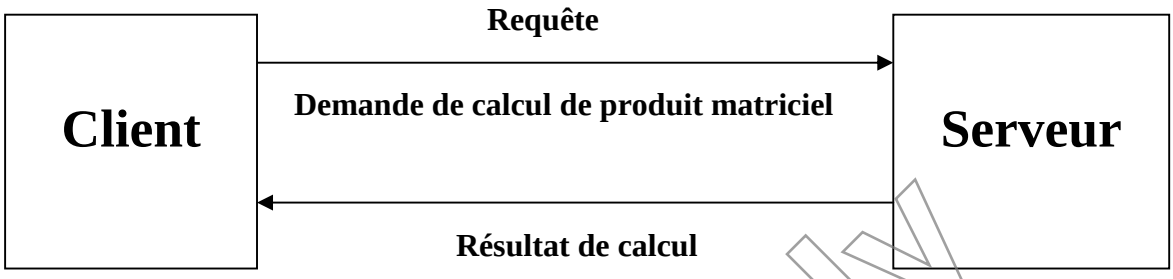

Figure -22- : Schéma général de l'application.

Le serveur distribue les éléments de chaque matrice sur plusieurs serveurs qui effectuent des traitements différents.

Le service 1 regroupe plusieurs serveurs effectuent le produit d'une ligne de la première matrice fois la colonne de la deuxième matrice ensuite le service 2 qui regroupe plusieurs serveurs effectuent la somme de ces produits et retourne le résultat.

IV.4. Etude de l'exécution de l'application sur les trois plates formes

Nous allons présenter et expliquer dans cette section le schéma d'exécution de notre application distribuée de calcul de produit de deux matrices sous chaque plate forme : Java RMI, CORBA et service Web.

IV.4.1. Java RMI

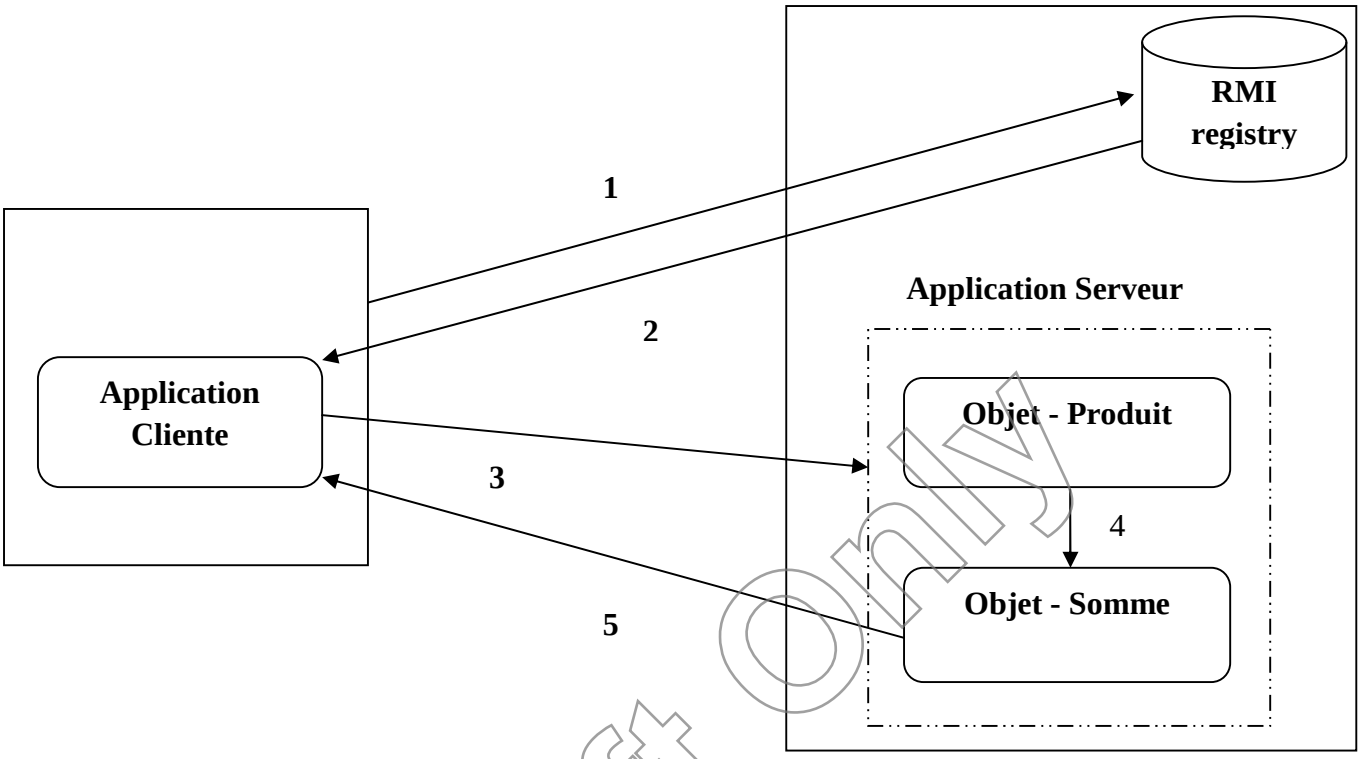

Figure -23- : le schéma global de l'exécution de l'application distribuée sous Java RMI.

1. Le client recherche la référence de l'objet distant (le produit matriciel) dans le registre RMI.
2. Le registre RMI retourne la référence au client.
3. Le client invoque la méthode à exécuter avec ses paramètres (les deux matrices).
4. L'objet produit après avoir fait le produit ligne par colonne il transmit les résultats à l'objet somme.
5. L'objet somme construit la matrice résultante après avoir fait la somme et envoie le résultat au client.

IV.4.2. CORBA

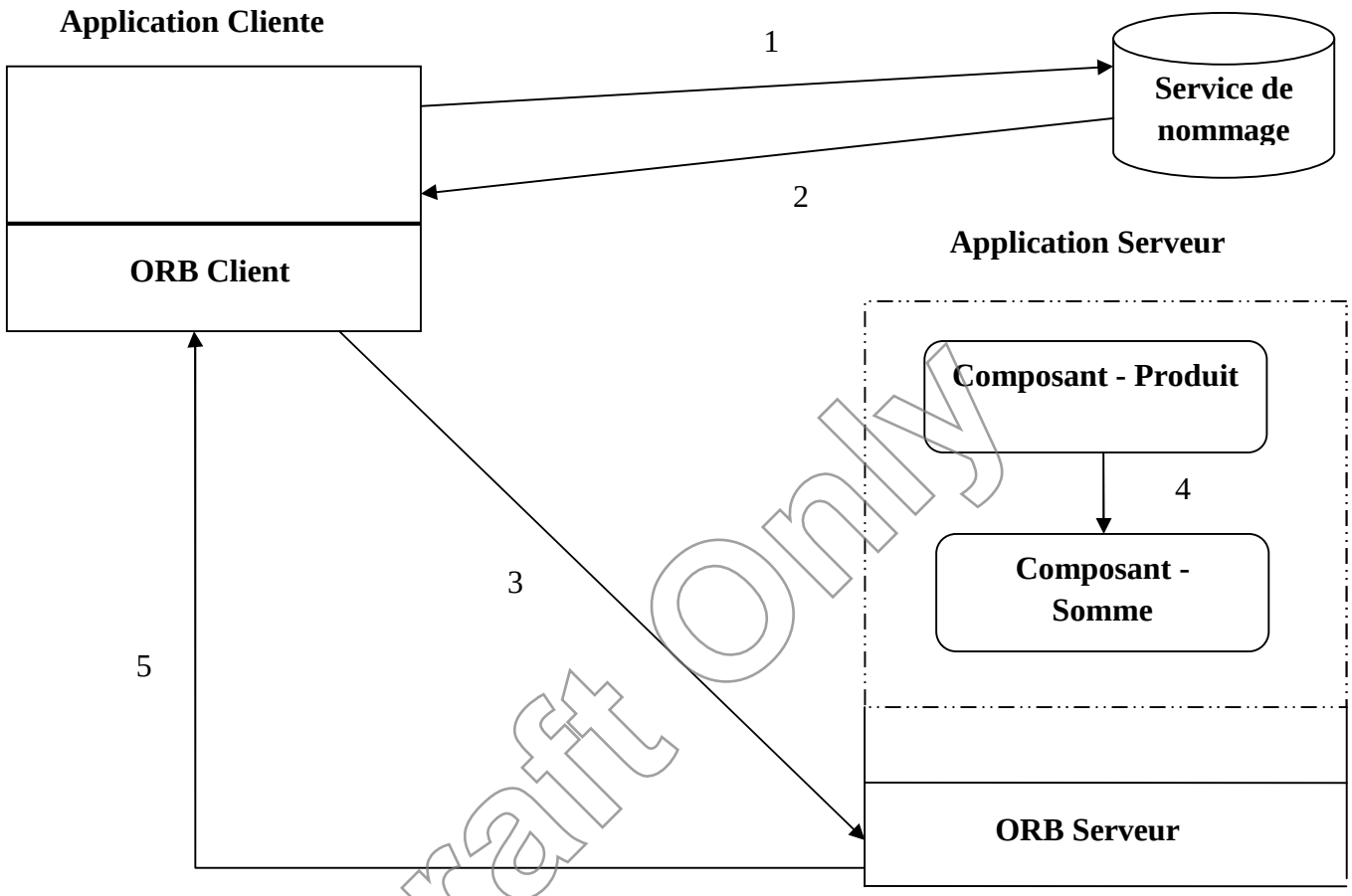

Figure -24- : le schéma global de l'exécution de l'application distribuée sous CORBA.

1. Le client recherche la référence de l'objet distant dans le service de nommage.
2. Le service de nommage lui retourne la référence.
3. Le client invoque la méthode à exécuter avec ses paramètres (les deux matrices) à travers l'ORB Client.
4. Le composant produit après avoir fait le produit ligne par colonne il transmit les résultats au composant somme.
5. L'objet somme construit la matrice résultante après avoir fait la somme et envoie le résultat au client à travers l'ORB serveur.

IV.4.3. Services Web

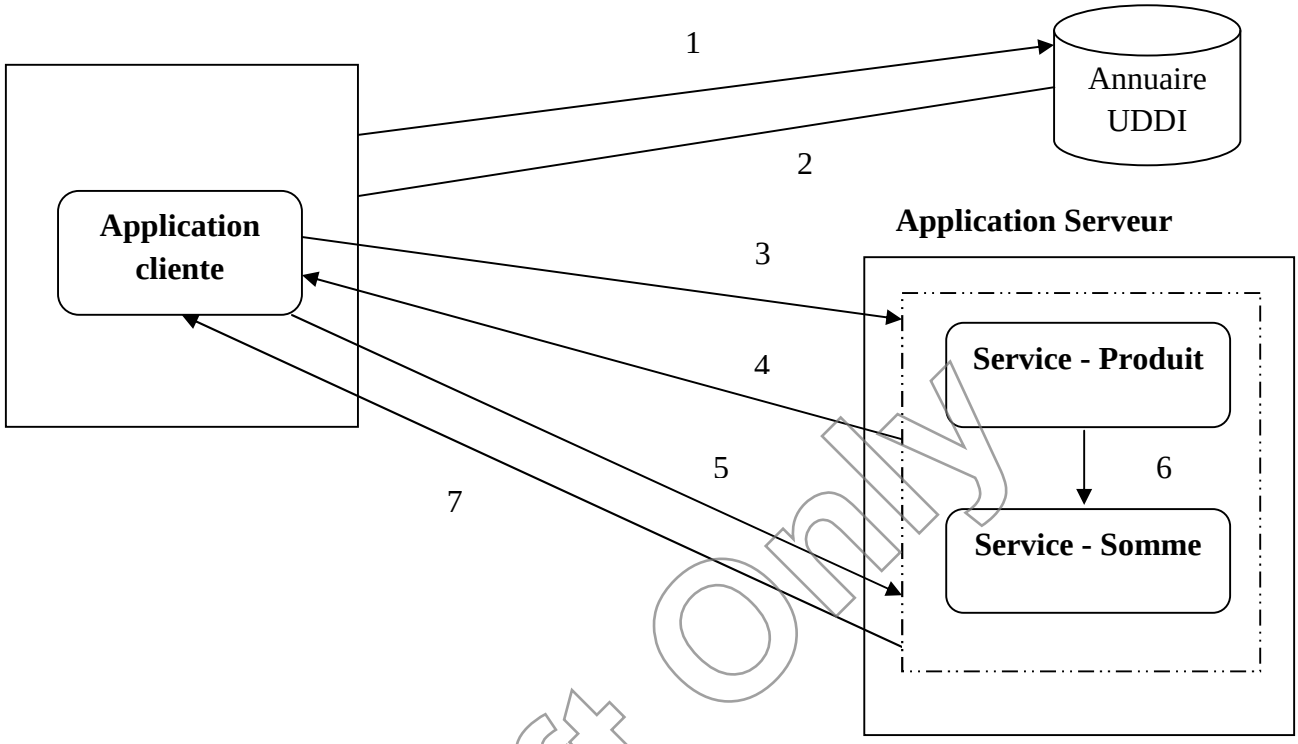

Figure -25- : le schéma global de l'exécution de l'application distribuée sous les services Web.

1. Le client recherche un service Web de calcul de produit de deux matrices dans l'annuaire UDDI.
2. L'annuaire UDDI retourne au client le serveur hébergent le service Web.
3. Le client demande le format d'appel du service.
4. Le serveur lui retourne un contrat WSDL.
5. Le client invoque le service souhaité.
6. Le service produit après avoir fait le produit ligne par colonne il transmit les résultats au service somme.
7. L'objet somme construit la matrice résultante après avoir fait la somme et envoie le résultat au client.

IV.5. La modélisation de l'application distribuée sous les trois plates formes

IV.5.1. Java RMI

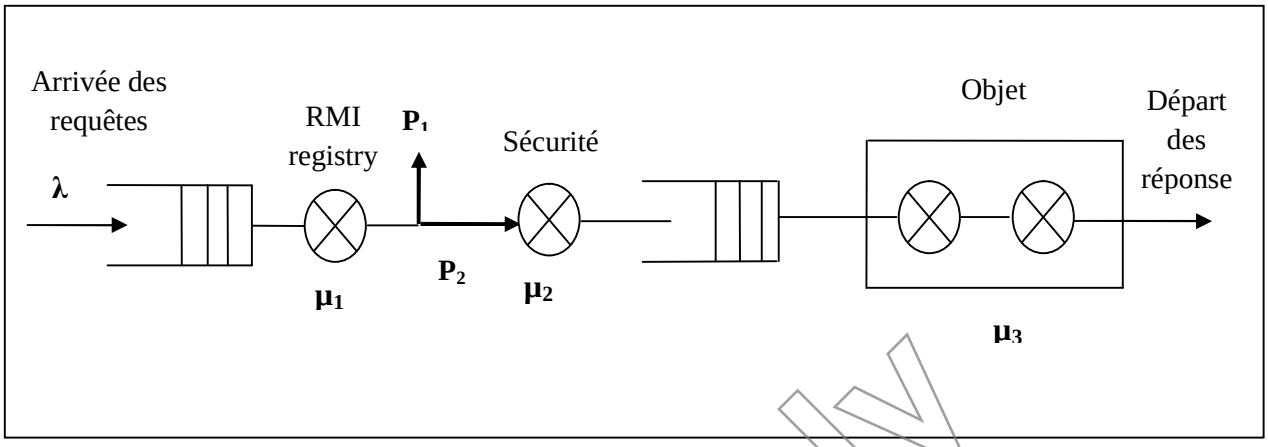

Figure -26- : Modèle files d'attente d'exécution de l'application distribuée sous Java RMI.

Dans ce modèle de réseaux de files d'attente, les clients sont des requêtes simples, les inter-arrivées sont supposées exponentielles, le temps moyen de service (μ) suit la loi exponentielle, et la discipline de service est FIFO.

Les serveurs sont : Registre RMI, Serveur de Sécurité, Objet qui regroupe le serveur qui effectue le produit et le serveur qui effectue la somme et retourne le résultat.

Une requête qui arrive accède directement au **Registre RMI,** si ce dernier est libre sinon elle se met dans une file d'attente, afin de rechercher la référence de l'objet distant désiré et dans notre cas la référence est l'application de produit de deux matrices. Si la requête n'est pas satisfaite elle sort avec une probabilité $\mathbf{P_1}$ dans le cas d'erreur, sinon elle rentre dans un serveur de **Sécurité** (afin d'éviter la perte des requêtes) avec une probabilité $\mathbf{P_2}$ ($P_1 + P_2 = 1$). La requête est ensuite affectée à l'**Objet** (où l'objet désiré se trouve), ce dernier regroupe le serveur qui effectue le produit entre un élément d'une ligne de la première matrice et l'élément de la colonne de la deuxième matrice et le serveur qui effectue la somme après avoir les résultats de produit et enfin il retourne le résultat.

IV.5.2. CORBA

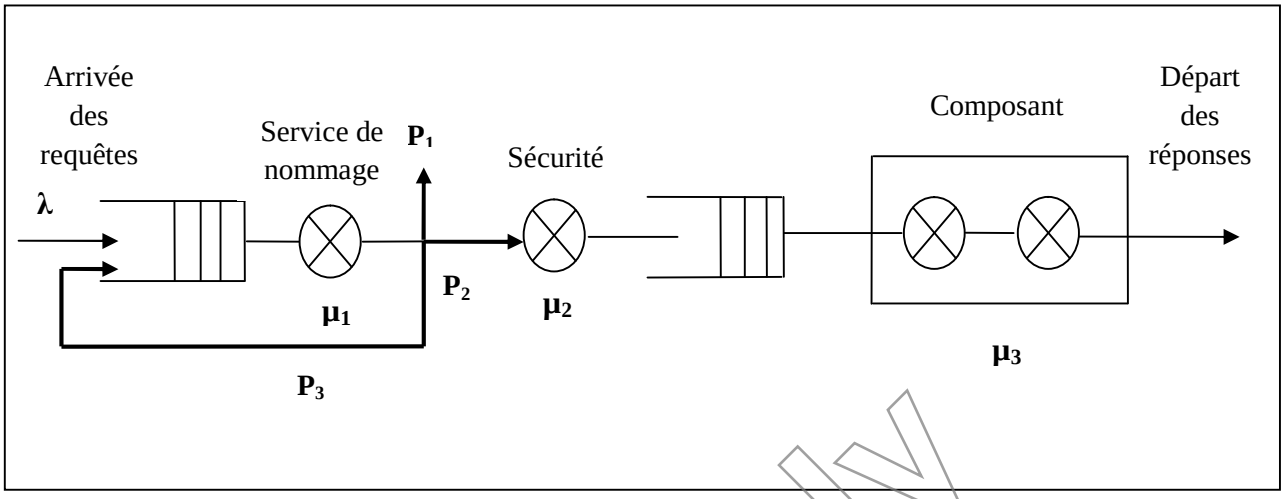

Figure -27- : Modèle files d'attente d'exécution de l'application distribuée sous CORBA.

Dans ce modèle de réseaux de files d'attente, les clients sont des requêtes simples, les inter-arrivées sont supposées exponentielles, le temps moyen de service (μ) suit la loi exponentielle, et la discipline de service est FIFO.

Les serveurs sont : Registre RMI, Serveur de Sécurité, Objet qui regroupe le serveur qui effectue le produit et le serveur qui effectue la somme et retourne le résultat.

Une requête qui arrive accède directement au **Service de nommage**, si ce dernier est libre sinon elle se met dans une file d'attente, afin de rechercher la référence de l'objet distant désiré et dans notre cas la référence est l'application de produit de deux matrices. Il ya trois cas possibles : Si la requête n'est pas satisfaite et dans ce cas elle retournera à la fille d'attente de serveur de nommage avec une probabilité $\mathbf{P_3}$ afin d'effectuer une autre tentative de recherche. Dans le cas d'erreur elle quitte le système avec une probabilité $\mathbf{P_1}$, sinon elle rentre dans un serveur de **Sécurité** (afin d'éviter la perte des requêtes) avec une probabilité $\mathbf{P_2}$ ($P_1+P_2+P_3=1$). La requête est ensuite affectée au **Composant** (où l'objet désiré se trouve), ce dernier regroupe le serveur qui effectue le produit entre un élément d'une ligne de la première matrice et l'élément de la colonne de la deuxième matrice et le serveur qui effectue la somme après avoir les résultats de produit et enfin il retourne le résultat.

IV.5.3. Service Web

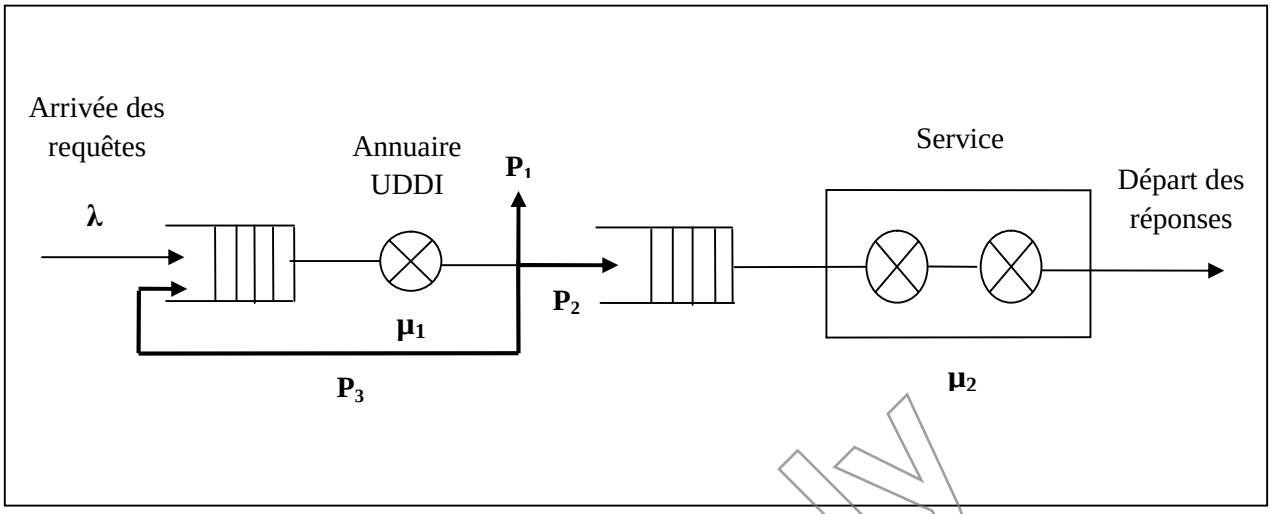

Figure -28- : Modèle files d'attente d'exécution de l'application distribuée sous les services Web.

Dans ce modèle de réseaux de files d'attente, les clients sont des groupes de requêtes, les inter-arrivées sont supposées exponentielles, le temps moyen de service (μ) suit la loi exponentielle, et la discipline de service est FIFO.

Les serveurs sont : Annuaire UDDI, Serveur de Sécurité, Service qui regroupe le serveur qui effectue le produit et le serveur qui effectue la somme et retourne le résultat.

Une requête qui arrive accède directement à l'**Annuaire UDDI** pour rechercher le service Web désiré et récupère son URL si l'annuaire est libre, sinon elle est placée dans une file d'attente. Si la requête n'est pas satisfaite elle retournera à la file d'attente de l'annuaire UDDI avec une probabilité $\mathbf{P_3}$ afin d'effectuer une autre tentative de recherche, dans le cas d'erreur elle sort avec une probabilité $\mathbf{P_1}$, elle passera au **Service** avec une probabilité $\mathbf{P_2}$, ce dernier regroupe le serveur qui effectue le produit entre un élément d'une ligne de la première matrice et l'élément de la colonne de la deuxième matrice et le serveur qui effectue la somme après avoir les résultats de produit et enfin il retourne le résultat.

IV.5.4. Interopérabilité Java RMI – CORBA

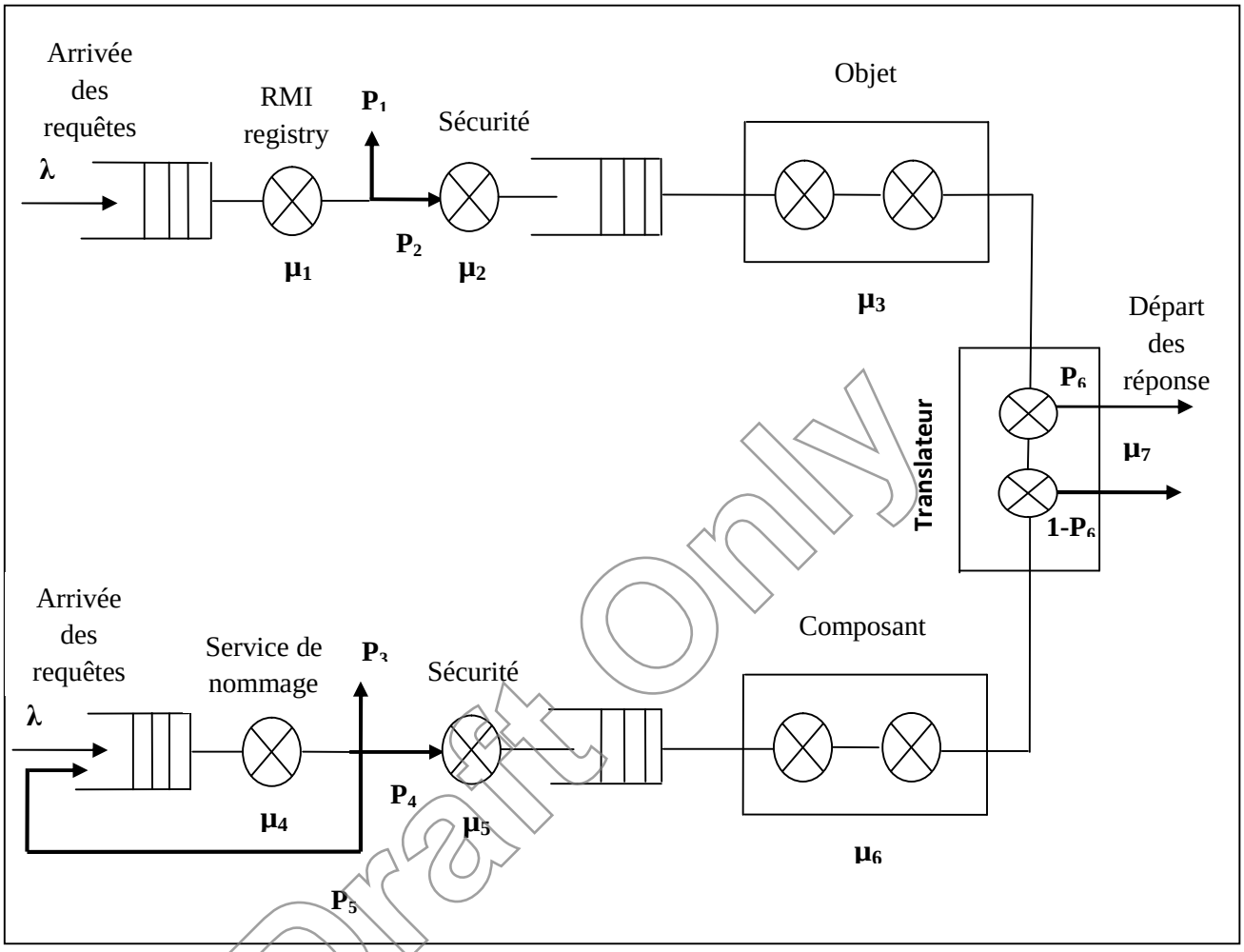

Figure -29- : Modèle files d'attente d'exécution de l'application distribuée sous Java RMI - CORBA.

Dans ce modèle de réseaux de files d'attente, les clients sont des requêtes simples, les inter-arrivées sont supposées exponentielles, le temps moyen de service (μ) suit la loi exponentielle, et la discipline de service est FIFO.

Les serveurs sont : Registre RMI, Serveur de nommage, Serveurs de Sécurité, Objet, Composant ces deux derniers regroupent le serveur qui effectue le produit et le serveur qui effectue la somme et retourne le résultat au Translateur qui transforme le résultat de Java RMI en CORBA ou de CORBA en Java RMI.

Les requêtes arrivent au niveau de Java RMI et CORBA et suivent le chemin déjà expliqué dans IV.5.1 et IV.5.2 et on aura deux résultats différents qui passent par le Translateur qui transforme un résultat Java RMI en CORBA et vice versa pour avoir un seul résultat à la fin.

IV.6. La simulation

Après la modélisation de l'application distribuée de produit de deux matrices sous les trois plates formes : Java RMI, CORBA et service Web, nous avons implémenté un programme de simulation pour chaque modèle sous Matlab.

IV.6.1. Description de l'environnement du travail

Matlab est un logiciel de calcul et de visualisation produit par MathWorks dont les entités de base sont des matrices, Matlab est une abréviation de Matrix Laboratory. Matlab est un langage simple, très efficace et interprété : il propose des facilités de programmation et de visualisation, ainsi qu'un grand nombre de fonctions réalisant diverses méthodes numériques.

Pour le calcul numérique, Matlab est beaucoup plus concis que les vieux langages (C, Pascal, Fortran, Basic). On peut traiter la matrice comme une simple variable.

Matlab contient également une interface graphique puissante, ainsi qu'une grande variété d'algorithmes scientifiques. On peut enrichir Matlab en ajoutant des boites à outils (toolbox) qui sont des ensembles de fonctions supplémentaires, profilées pour des applications particulières (traitement de signaux, analyse statistiques, optimisation,…….).

IV.6.2. Les résultats de la simulation

Les performances des modèles calculés par le programme de simulation avec la méthode Monté Carlo, nous permettre de faire une comparaison entre les trois architectures.

➢ Les paramètres de simulation

Les résultats de cette simulation établis, après avoir saisi les différents paramètres.
On définit le taux d'arrivées des requêtes qui suit la loi exponentielle, par : λ : : 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35.

- Le temps maximum de simulation : $\mathbf{T_{max}}$ = 100000 µs.
- RAM 4 Go.
- Vitesse 2,53 GHZ.
- Monté Carlo : 40 (le nombre d'itération).
- µ : le temps moyen de service qui suit la loi exponentielle.

➢ Organigramme de la simulation

Time : compteur.

Tsim : Durée de simulation.

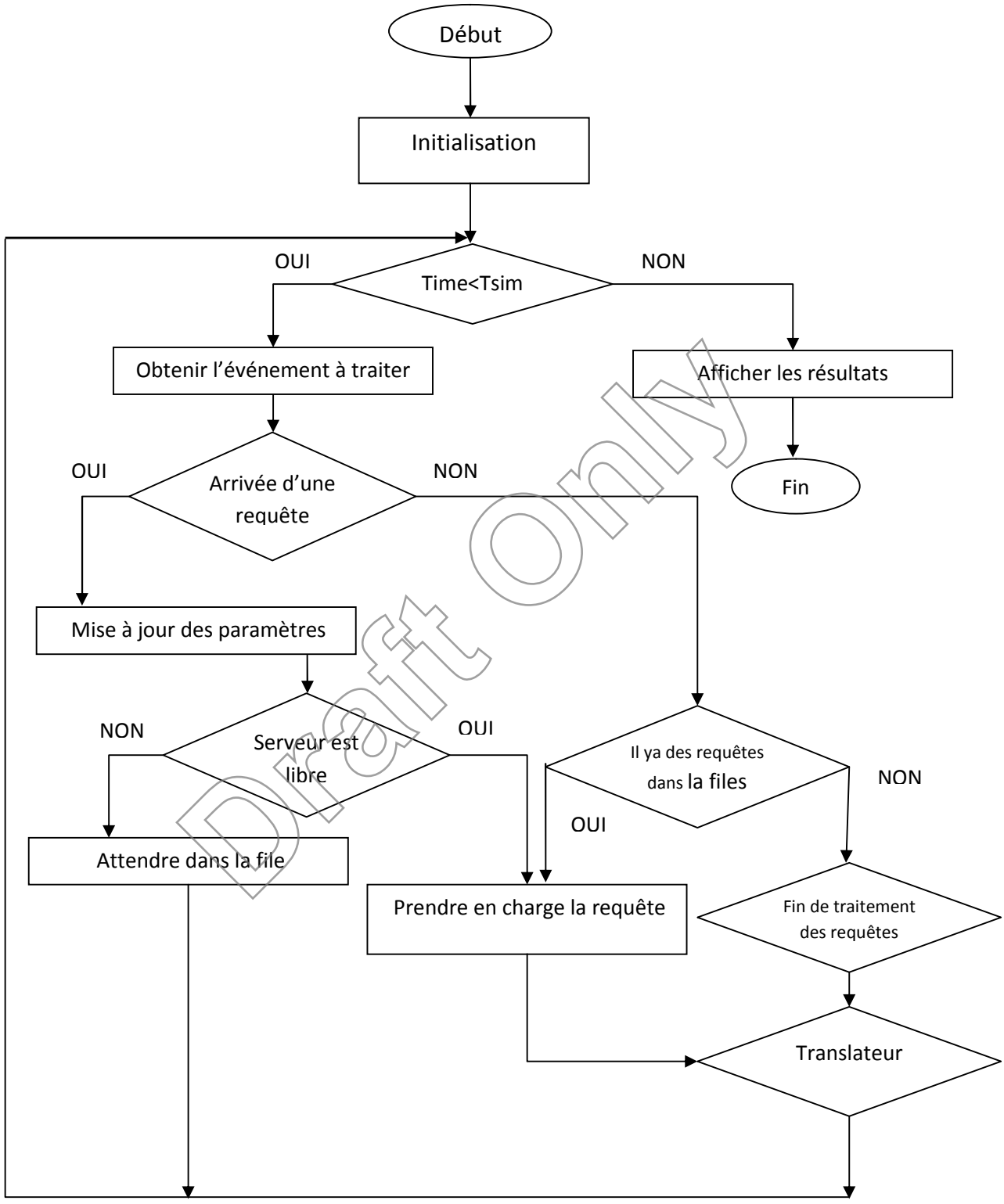

Figure -30- : Diagramme de la simulation.

A. Résultat 1 : Débit du système en fonction de taux de service

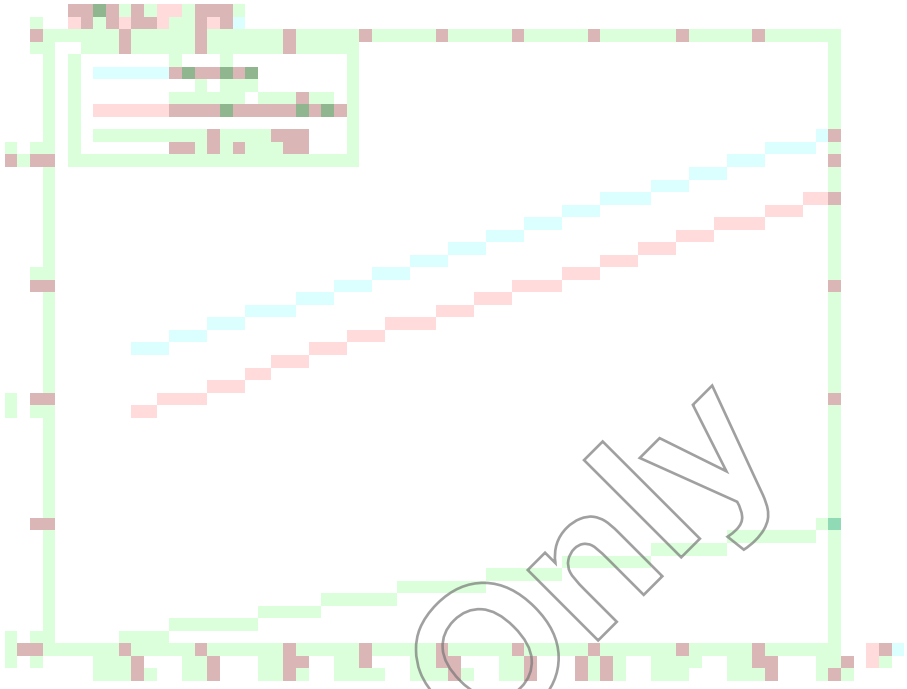

Figure -31- : Variation du débit en fonction de taux de service.

Dans cette figure, les courbes montrent les variations de débit en fonction du taux de service. Nous remarquons que le débit, dans les trois plates formes Java RMI, CORBA et services Web augmente en augmentant le taux de service. Java RMI est plus performant que les deux autres plates formes.

La différence entre ces trois résultats revient à l'architecture des différentes plates formes.

B. Résultat 2 : Temps moyen de réponse du système en fonction de taux de service

Le temps de réponse permet de préciser la durée moyenne nécessaire entre l'arrivée d'une requête et son départ final. Il représente la durée moyenne passée par chaque requête dans le système (file d'attente + les serveurs).

La figure suivante montre le temps moyen de réponse en fonction de taux de service dans les trois plates formes.

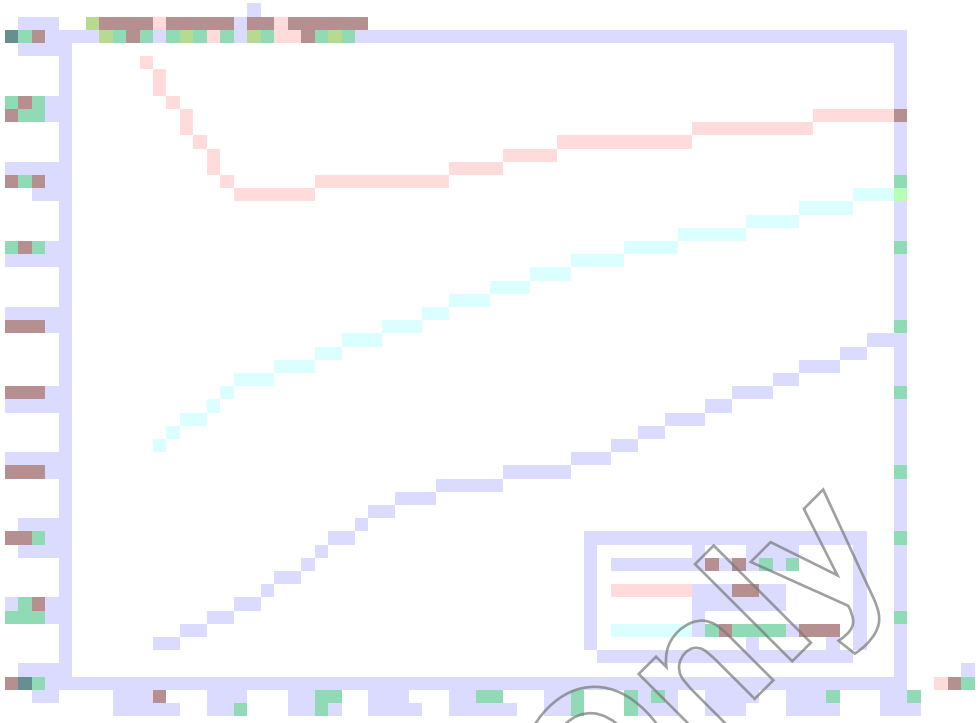

Figure -32- : Temps moyen de réponse en fonction de taux de service.

Cette figure montre que Java RMI est plus performant que les autres plates formes.

Le taux de service influe sur le temps de réponse total car le temps de réponse augmente avec l'augmentation de taux de service. Pour CORBA le temps de réponse diminue mais à partir de $\mu_{3=}$ 0.4 augmente avec l'augmentation de taux de service.

La différence est due à la simplicité de java RMI, la notion des paquets dans les services Web et la complexité dans CORBA comme nous avons déjà expliqué.

C. Résultat 3 : Nombre de requêtes satisfaites

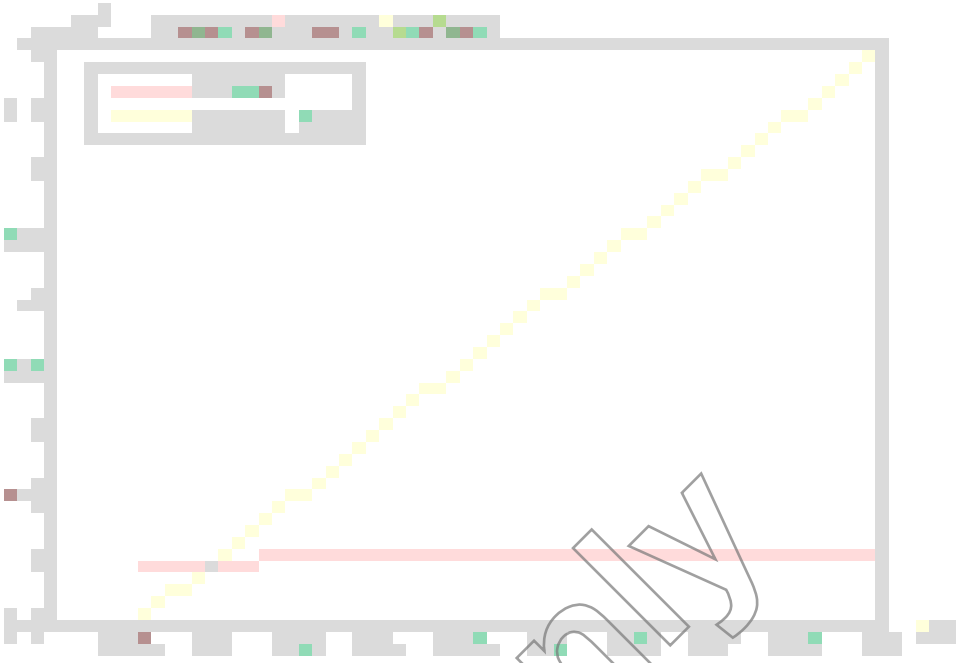

Figure -33- : Nombre de requêtes satisfaites pour CORBA et services Web.

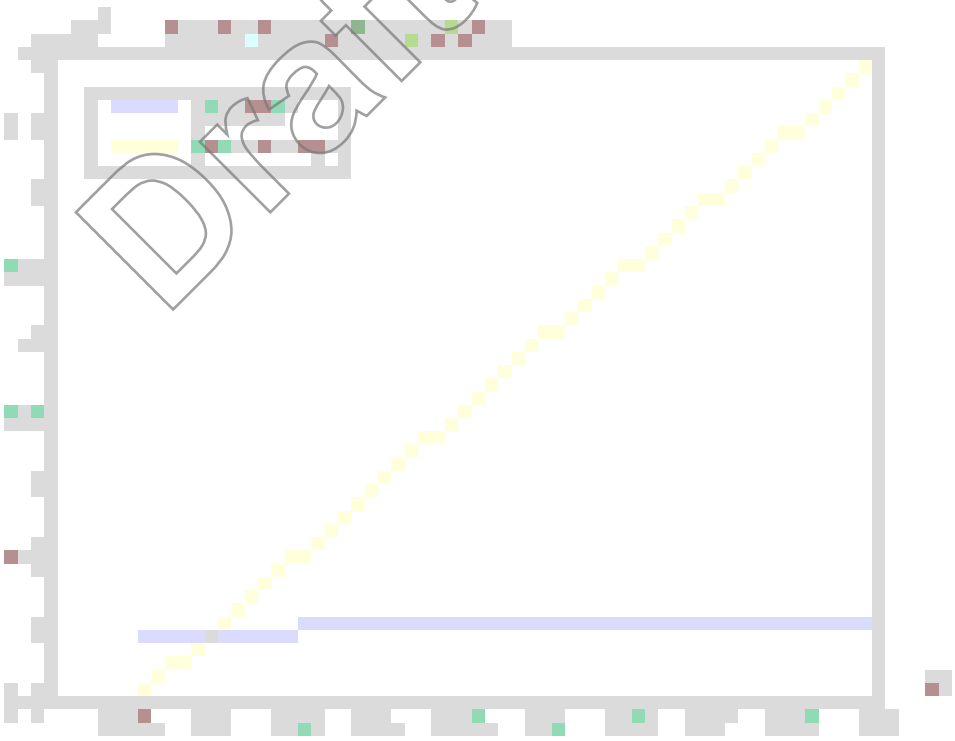

Figure -34- : Nombre de requêtes satisfaites pour Java RMI et services Web.

Les figures -33- et -34- montrent que services Web est plus performant que les deux autres plates formes.

Le nombre de requêtes satisfaites dans les services Web augmente en augmentant le taux de service et par contre pour java RMI et CORBA le nombre de requêtes est stable.

IV.6.3. Interopérabilité Java RMI - CORBA

A. Résultat 1 : Débit du système en fonction de taux d'arrivée

La figure ci-dessous montre la variation de débit en fonction de taux d'arrivée.

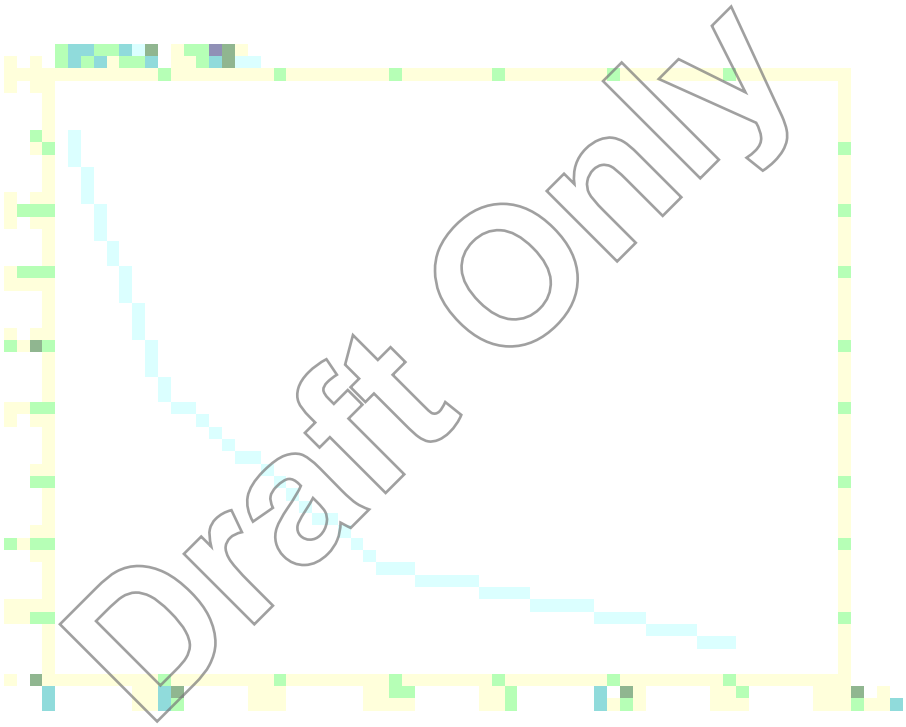

Figure -35- : Variation du débit en fonction de taux d'arrivée.

La figure montre que le débit diminue en augmentant le taux d'arrivée (le temps moyen entre les inter-arrivées diminue) d'où le nombre de requêtes augmente et le débit diminue.

B. Résultat 2 : Temps moyen de réponse du système en fonction de taux d'arrivée

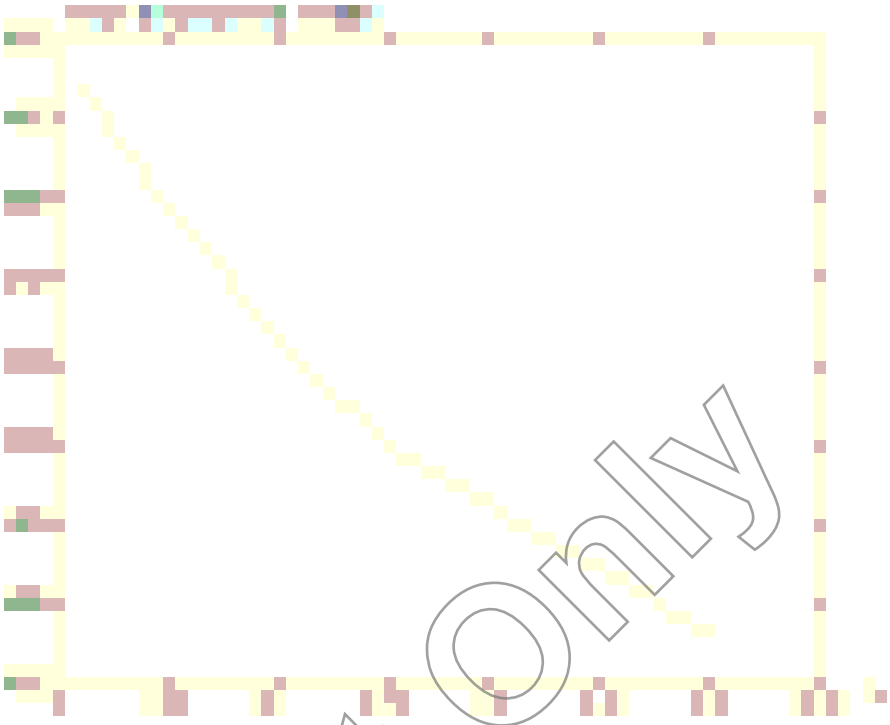

Figure -36- : Temps moyen de réponse en fonction de taux d'arrivée.

Le temps de réponse dans le cas Java RMI – CORBA diminue en augmentant le taux d'arrivée. Car le nombre de requêtes augmente d'où le temps de service augmente et le temps de réponse diminue.

C. Résultat 3 : Nombre de requêtes satisfaites

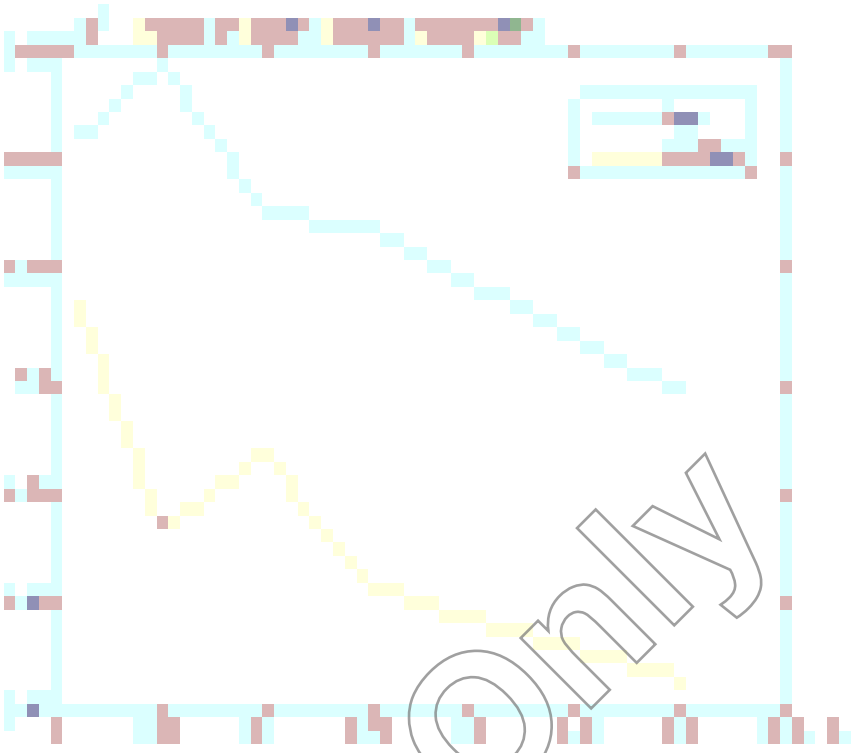

Figure -37- : Nombre de requêtes satisfaites pour Java RMI - CORBA.

On remarque que le nombre de requêtes satisfaites par java RMI est plus grand que le nombre de requêtes satisfaites dans CORBA et dans Java RMI le nombre de requêtes augmente en premier temps après il diminue en augmentant le taux d'arrivée et pour CORBA le nombre de requêtes diminue et après à $\lambda = 0.05$ il augmente après il continue à se diminué. Ceci est dû au type de requêtes (la taille de la matrice).

IV.7. Conclusion

A partir des résultats de la simulation que nous avons obtenus, et d'après les résultats des auteurs cités auparavant on pourra dire que la plate forme Java RMI est plus performante que celle de CORBA et services Web en temps moyen de réponse et le débit du système et pour le nombre de requêtes satisfaites la plate forme services Web est plus performante que Java RMI et CORBA. Pour l'interopérabilité Java RMI – CORBA le débit et le temps de réponse diminue en fonction de taux d'arrivée et nombre de requêtes satisfaites dans Java RMI est plus grand que CORBA.

Conclusion Générale et perspectives

Afin d'évaluer les performances des applications distribuées et déduire la plate forme la plus performante, plusieurs travaux ont été établis dans la littérature, soit en se basant sur des modèles (les files d'attente), soit en exécutant des applications réelles.

L'objectif principal de notre travail est l'évaluation de performances d'une application distribuée via Internet qui consiste à calculer le produit de deux matrices sous chacune des trois plates formes distribuées Java RMI, CORBA et services Web.

Cette évaluation est réalisée à base d'une simulation à travers des modèles de file d'attente afin d'évaluer les performances de ces plates formes suivant un ensemble de critères inspirés principalement des travaux présentés dans l'état de l'art.

Comme tout travail de recherche, qui nécessite une continuité et des améliorations, nous proposons pour de futurs travaux, les perspectives suivantes :

- ✓ Supporter des matrices de grande taille.
- ✓ Analyser l'impact de pare-feu sur chaque plate forme.
- ✓ Comparer le temps de réponse pour des types de données différents.

Bibliographie

[1] M. Tounsi. **Preuves d'algorithmes distribués par raffinement.** Thèse Doctorat. Université de Bordeaux. France. Juillet 2012.

[2] A. Contes. **Une architecture de sécurité hiérarchique, adaptable et dynamique pour la grille.** Thèse Doctorat. Université de Nice. France. Septembre 2005.

[3] P. Laumay. **Configuration et déploiement d'intergiciel asynchrone sur système hétérogène à grande échelle.** Thèse Doctorat. Institut National Polytechnique de Grenoble. France. Mars 2004.

[4] N. Baraham, N. Hadji. **Qualité de Service et Evaluation des Performances dans les Architectures Distribuées Java RMI, CORBA, services Web.** Mémoire Master2, Université Abderrahmane Mira, Bejaia Algérie. 2011.

[5] http:// www.wikipedia.com Avril 2013.

[6] D. Durand. **Gestion de la Qualité de Service dans les Applications Réparties sur Bus Middleware Orientés Objet Approche Dirigée par les Modèles**. Thèse Doctorat. Université de Picardie Jules Verne. France. Novembre 2006.

[7] A. Fron. **Architectures Réparties en Java RMI, CORBA, JMS, sockets, SOAP, services Web.** Edition Dunod. 2008.

[8] B. Haidar. **Services d'Indexation Multimédia Distribués.** Thèse Doctorat. Université Paul SABATIER Toulouse III. France. Septembre 2005.

[9] T. Quinot. **Conception et réalisation d'un intergiciel schizophrène pour la mise en œuvre de systèmes répartis interopérables.** Thèse Doctorat. Université Paris VI — Pierre-et-Marie-Curie. France. Mars 2003.

[10] M-C. Pellegrini. **Reconfiguration d'applications réparties : application au bus logiciel CORBA.** Thèse Doctorat. Institut National Polytechnique de Grenoble. France. 2000.

[11] M. jaber. **Architecture de Système d'Information Distribué pour la Gestion de la Chaîne Logistique : Une Approche Orientée Services.** Thèse Doctorat. Institut National des Sciences Appliquées de Lyon**.** France. février 2009.

[12] J. Geib. C. Gransart. **CORBA : des concepts à la pratique.** 2eme Edition DUNOD. 2000.

[13] P. Loc**. Adaptation des composants centrés sur l'utilisation.** Thèse Doctorat. Institut national Polytechnique de Toulouse. France. Novembre 2004.

[14] A. Turki. **Un modèle pour la composition d'applications de visualisation et d'interaction continue avec des simulations scientifiques.** These Doctorat. Université D'Orléans. France. Mars 2012.

[15] M..Bennani. **Tolérance aux Fautes dans les Systèmes Répartis à base d'Intergiciels Réflexifs Standards.** Thèse Doctorat. Institut national des sciences appliquées de Toulouse. France. Juin .2005.

[16] G. Salagnac. **Synthèse de gestionnaires mémoire pour applications Java temps-réel embarquées.** Thèse Doctorat. Université Joseph Fourier – Grenoble I. France. Avril 2008.

[17] K. Mazouzi. **JACE : un environnement d'exécution distribué pour le calcul itératif asynchrone.** Thèse Doctorat. Université de Franche-Comté UFR Sciences et Techniques. France. Décembre 2005.

[18] C. Bernine, S. Bouchetaoui. **Evaluation des Performances d'une Application Distribuée sous Java RMI, CORBA et Services Web.** Mémoire Master2, Université Abderrahmane Mira, Bejaia Algérie. 2012.

[19] http:// www.w3.org/2003/talks/0521-hh-wsa. Avril 2013.

[20] http:// www.IBM.com Mai 2013.

[21] http:// www.Webopedia.com Mai 2013.

[22] W. Sellami. **Une approche formelle pour la vérification des propriétés non fonctionnelles d'orchestration des services web.** Mémoire Master. Université de Sfax. Tunis. Juillet 2011.

[23] M. Lallali. **Modélisation et Test Fonctionnel de l'Orchestration de Services Web.** Thèse Doctorat. L'Institut National Des Télécommunications. Novembre 2009.

[24] S. Chollet. **Orchestration de Services Hétérogènes et Sécurisés.** Thèse Doctorat. Université Joseph Fourier Grenoble I. France. Décembre 2009.

[25] C.L. Velasco. **Sélection et composition de Services Web pour la génération d'applications adaptées au contexte d'utilisation.** Thèse Doctorat. Université Joseph Fourier Mathématiques. France. Novembre 2008.

[26] M. Driss. **Approche multi-perspective centrée exigences de composition de services Web.** Thèse Doctorat. Université de Rennes 1. France. Décembre 2011.

[27] S. Rampacek. **Sémantique, interactions et langages de description des services web complexes.** Thèse Doctorat. Université de Reims Champagne-Ardenne. France. Novembre 2006.

[28] S. Bernier. **Conception et implantation basées sur des composants répartis d'une station terrestre virtuelle de communication satellite.** Mémoire. Université de Sherbrooke. Canada. Septembre 2000.

[29] S. Chardigny**. Extraction d'une architecture logicielle à base de composants depuis un système orienté objet Une approche par exploration.** Thèse Doctorat .Université de Nantes École des Mines de Douai. Octobre 2009.

[30] H. Cervantes. **Vers un modèle à composants orienté service pour supporter la disponibilité dynamique.** Thèse Doctorat. Université Joseph Fourier. France. Mars 2004.

[31] Z. Chouiref. **Un système de programmation fonctionnelle pour la composition de service Web.** Mémoire de Magister. Université M'hamed Bougara Boumerdes. Algérie. 2008.

[32] L. Bouallouche. **Modélisation et simulation des systèmes informatiques et réseaux de télécommunication.** Cours. Université Abderrahmane Mira, Bejaïa. Algérie. 2012.

[33] M. Hassani**. Evaluation de Performance des Systèmes d'Attente.** Mémoire d'ingénieur. Université Abderrahmane Mira de Bejaia. Algérie. 2008.

[34] G. Habchi. **Conceptualisation et Modélisation pour la simulation des Systèmes de production.** Thèse Doctorat. Université de Savoie. Décembre 2001.

[35] A. Harbaoui. **Vers une modélisation et un dimensionnement automatiques des applications réparties.** Thèse Doctorat. Université de Grenoble. France. Octobre 2011.

[36] K. Cong. **Etude et Amélioration de l'organisation de la Production de Dispositifs Médicaux Stériles.** Thèse Doctorat. Université Joseph Fourier – Grenoble I. France. Mars 2009.

[37] Y. Arda. **Politiques d'approvisionnement dans les systèmes à plusieurs fournisseurs et Optimisation des décisions dans les chaînes logistiques décentralisées.** Thèse Doctorat. L'Institut National des Sciences Appliquées de Toulouse. France. Janvier 2008.

[38] D. Thu Ha. **Les files et les réseaux zéro-automatiques.** Thèse Doctorat. Université Paris Diderot. France. décembre 2007.

[39] M. Messegmine, I. Tikhmarine. **Modélisation de flux dans un réseau: étude et simulation de files d'attente.** Mémoire Master2, Université Abou Bakr Belkaid– Tlemcen**.** Algérie. Juillet 2012.

[40] S. Artemio, B .Leonardo, J .Hugo, M.J. Carlos et al. **Evaluation of CORBA and Web Services in Distributed Applications.** In Proceedings of 22nd International Conférence Electrical Communications and Computers (CONIELECOMP). Pages 97-100, Mexique. Février 2012.

[41] R. Eggen, M. Eggen. **Efficiency of Distributed Parallel Processing using Java RMI, Sockets, and CORBA.** In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. Pages 888-893. Las Vegas, USA. Juin 2001.

[42] N. A. B. Gray, **Performance of Java Middleware - Java RMI, JAXRPC, and CORBA.** In Proceedings of The 6th Australasian Workshop on Software and System Architectures (AWSA). Pages 31-39. Australie. Mars 2005.

[43] J. Riboe, **Can Web Services run along side CORBA and RMI.** Journal. RiboComments, Web Service performance. 2009.

[44] M. B. Juric, B. Kezmah, M. Hericko et al. **Java RMI, RMI Tunneling and Web Services Comparison and Performance Analysis**. Journal. ACM SIGPLAN Notices, Volume 39, Issue 5, page 58-65. 2004.

[45] V.Narang, P. Pulusani, H. Thakkar, **Comparison of Communication Protocols in Distributed Environment.** Distributed Warriors. 2012.

[46] S. Lee, K.W. Lee, K.D.Ryu et al. **Deployment-time Binding Selection to Improve the Performance of Distributed Applications.** Journal. Computer Science, IBM Research Report, RC23861. Volume 218. Pages 1-19. 2006.

[47] S. Loustau. **Performances statistiques de méthodes à noyaux.** Thèse Doctorat. Université d'Angers. France. Novembre 2008.

[48] A. Korichi, M. Seddiki. **Investigation sur la possibilité de l'évaluation de performance multicritères d'une entreprise par l'exploitation de la simulation sur ordinateur.** Journal. Gestion et Simulation, volume 3, pages 17-25. 2004.

[49] J. Diéguez. **Contributions à la modélisation et à la simulation accélérée de réseaux de communication.** Thèse Doctorat. Université de Rennes1. France. Mars 2001.

[50] M. Himdi. **Performances des Systèmes Distribués : Proposition d'une Plateforme Fédératrice de la Supervision.** Thèse Doctorat. Université de Rennes 1. France. Décembre 2007.

[51] O. Salem. **Modélisation Algébrique et Évaluation de Performances des Mécanismes de Gestion de la Qualité de Service dans les Réseaux Ad Hoc.** Thèse Doctorat. Université Toulouse III - Paul Sabatier**.** France. Octobre 2006.

[52] A. Mkhida. **Contribution à l'évaluation de la sûreté de fonctionnement des Systèmes Instrumentés de Sécurité intégrant de l'Intelligence.** Thèse Doctorat. Institut National Polytechnique de Lorraine**.** France. Novembre 2008.

[53] M. Bahri. **Une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles.** Thèse Doctorat. Université Mentouri Constantine**.** Algérie. Juillet 2010.

[54] M. Zidouni. **Modélisation et analyse des performances de la bibliothèque MPI en tenant compte de l'architecture matérielle.** Thèse Doctorat. Université Joseph Fourier – Grenoble. France. Mai 2010.

[55] S. Kendi, S. Touati. **Evaluation de Performances d'un Réseau Informatique.** Mémoire d'Ingénieur. Université Abderrahmane Mira, Bejaia Algérie. Juin 2006.

[56] A. Benyahia. **Contribution à la mise-en-œuvre d'un moteur d'exécution de modèles UML pour la simulation d'applications temporisées et concurrentes.** Thèse Doctorat. Ecole doctorale, Sciences et Technologies de l'Information des Télécommunications et des Systèmes**.** France. Novembre 2012.

[57] H. Malgouyres. **Définition et détection automatique des incohérences structurelles et comportementales des modèles UML - Couplage des techniques de métamodélisation et de vérification basée sur la programmation logique.** Thèse Doctorat. L'Institut National Des Sciences Appliquées de Toulouse**.** France. Novembre 2006.

Résumé

Les systèmes d'information ont évolué d'un contexte centralisé et monolithique à des environnements distribués et hétérogènes. Dans ce nouveau contexte, les intergiciels ont pris une place importante afin d'assurer l'interopérabilité entre les différents systèmes distribués. Plusieurs intergiciels sont apparus comme Java RMI, CORBA et services Web. Ceci est dû à la prise en compte de plusieurs critères et facteurs : interopérabilité, transparence, efficacité et sécurité, … Notre travail s'inscrit dans le cadre des plates formes distribuées, plus précisément à évaluer les performances d'une application distribuée qui calcule le produit de deux matrices sous Java RMI, CORBA et services Web à travers une simulation de modèles de réseaux de files d'attente.

Mots clés

Systèmes distribués, Intergiciels, Java RMI, CORBA, Services Web, Evaluation de performances, Réseau de files d'attente, Simulation.

Abstract

Information systems have evolved from a centralized and monolithic to distributed and heterogeneous environments. In this new context, middleware has taken an important place to ensure interoperability between different distributed systems. Several middleware have appeared as Java RMI, CORBA and Web services. This is due to the inclusion of several criteria and factors: interoperability, transparency, efficiency and security ... The scope of our work is distributed platforms, specifically to evaluate the performance of a distributed application that is the calculation of product of two matrixes in Java RMI, CORBA and Web services through queues networks models simulation.

Keywords

Distributed Systems, middleware, Java RMI, CORBA, Web Services, Evaluation of performances, Queues Networks, Simulation.