Table des matières

Table de figures

Introduction

Un service Web désigne essentiellement une application (ou un programme) mise à disposition sur Internet par un fournisseur de service, et accessible par les clients à travers des protocoles Internet standards. Cependant, pour certains types d'application, il est nécessaire de combiner un ensemble de services Web en services plus complexes (services Web agrégés ou composites) afin de répondre à des exigences plus complexes.

La qualité de service (QoS) représente un ensemble de propriétés opérationnelles du service que l'on doit constater dans la réalisation de la prestation (le débit, la disponibilité et le temps de réponse, …etc.). Ces propriétés (ou mesures) représentent un ensemble d'exigences concernant la mise en œuvre du service. La dégradation de ces mesures peut engendrer de sérieuses conséquences. Pour garantir ces propriétés, une analyse fine du comportement du système est nécessaire pour identifier les problèmes et les résoudre. Il est donc fondamental de disposer de méthodes et d'outils permettant d'analyser et de comprendre le comportement des services, afin de répondre à des questions de performance et de coût.

L'évaluation de performance des services Web est un problème difficile à cause du fait que les méthodes d'analyse ne montrent que peu d'intérêt lorsque l'on prend en compte leur fonctionnement. Dans ce mémoire de DEA, nous nous sommes intéressés à l'évaluation de performances des services Web composites en les modélisant par les réseaux d'automates stochastiques. De ce fait, nous commençons par un état de l'art sur les technologies des services web (SOAP, WSDL, UDDI et BPEL), puis nous faisons une étude sur les formalismes d'évaluation de performance (en particulier le réseau d'automate stochastique).

Dans la première partie de ce mémoire, nous étudierons l'architecture orientée service, ensuite nous commencerons par présenter les technologies des services web qui sont les moyens d'implémentation de cette architecture.

Dans la deuxième partie, nous présenterons les Réseaux d'Automates Stochastiques (RAS) que nous utiliserons pour la modélisation. Dans cette partie, nous étudierons la représentation formelle des RAS et les méthodes itératives utilisées pour la résolution.

Dans la dernière partie, nous allons faire l'implémentons du modèle proposé, et le calcul des distributions stationnaires afin de dégager et analyser les mesures de performances.

Partie I : L’architecture orientée services

I. Architecture orientée services

Les technologies des services web (SOAP, WSDL, UDDI, BPEL,…etc.) sont présentées comme le moyen d'implémentation des architectures orientées services. Cette architecture introduit la relation de service et les rôles de clients et de prestataires joués par les applications participantes.

L'architecture orientée services (AOS) est le terme utilisé pour désigner un modèle d'architecture pour l'exécution d'applications logicielles réparties. Dans ce modèle d'architecture, le concept service joue le rôle primaire, contrairement aux autres environnements reparties (CORBA, DCOM,…etc.) qui sont généralement des architectures par composants logiciels repartis plutôt que des architectures orientées services [**4**].

Rappelons de deux objectifs des environnements repartis, tel que CORBA, DCOM, qui sont :

- La standardisation du mécanisme d'invocation de traitements distant.
- La transparence de la localisation des composants dans un système réparti.

Le terme « service » est généralement absent de leur terminologie (sauf, par exemple, dans CORBA où l'on parle de services CORBA à propos de fonctions offertes par la plate-forme middleware aux composants applicatifs).

Construire une architecture orientée services signifie d'abord concevoir une architecture en réseau de relations de services, entre applications réparties. La description d'une relation de services est formalisée par un contrat de services. Ce contrat décrit les engagements réciproques du prestataire et du client du service.

A. Qu'est-ce qu'un service

La notion de service est le produit d'une démarche d'abstraction par rapport au logiciel qui l'implémente, au processus qui l'exécute et au port qui le localise. La réalisation d'un service passe par des messages échangés, des transitions d'état et des actions que l'application cliente suppose assurés de la part de l'application prestataire du service [**7**].

Pour faire partie du club des services il y a quatre critères à satisfaire :

Frontières explicites

Un service définit clairement ses points d'entrée. Un Service ne peut pas partager l'état avec un autre service . Ainsi, un service ne permet pas qu'on s'intègre à lui en partageant sa base de données, en appelant directement ses implémentations interne (sous forme de composants

métier ou autres), ou de quelque autre manière. Il définit clairement ses points d'entrées, et bloque toute tentative de pénétration par un quelconque autre accès.

Échanges par contrat

Les échanges se font par contrat : Un service ne donne aucune indication sur la manière dont il est implémenté, de la technologie avec laquelle il est créé, ou de la plateforme sur la quelle il est déployé. Il expose simplement un contrat qui stipule se que sont les formats de données échangés, les points d'entrée, quelles informations sont acceptée en entrée pour chaque point d'entrée, et quelles informations sont données en réponse.

Autonomie

Un service doit être autonome. Si les frontières explicites forcent déjà qu'un service ne communique pas de façon transversale avec un autre service, et qu'il ne partage aucun état avec un autre service, l'autonomie va encore plus loin. Un service doit "posséder" ses propres données. Cette possession est exclusive.

Données propres

La compatibilité est négociée en fonction de stratégies : Un service se doit de définir toutes les méthodes et protocoles d'échange qu'il permet en termes de stratégies. Si le service permet des séquences d'appels successifs, ils doivent être définis dans ces stratégies. Ainsi, un autre service, en utilisant simplement les contrats et les stratégies, peut définir s'il peut appeler ce service, et quelles méthodes et protocoles d'échange utiliser.

B. Les éléments du service

Les applications orientées services Peuvent jouer au moins l'un des deux rôles impliqués dans une relation de service [**4**] :

- **Prestataire de service** : c'est l'application qui exerce une activité dont les résultats sont directement ou indirectement exploitables par d'autres applications, éventuellement réparties sur un réseau.
- **Client** : les applications qui bénéficient de la prestation de service.

Les activités d'un prestataire de service peuvent être classées en trois groupes :

- **Informations** : c'est les résultats de l'application prestataire, qui transmis à l'application cliente sous forme de messages qui peuvent être par exemple des résultats d'un calcul complexe ;

- **Etats** : l'application prestataire gère les états et les changements d'états des ensembles de données. Les états peuvent être volatiles, persistants (les données sont stocker sur un mémoire secondaire) et durables (la possibilité de contrer les défaillances). Le changement d'état devrait être toujours durable.
- **Effets de bord** : c'est les interactions de l'application prestataire avec son environnement. par exemple des dispositifs d'entrées-sorties. Les effets de bords sont, à l'inverse des changements d'état, irréversibles par définition.

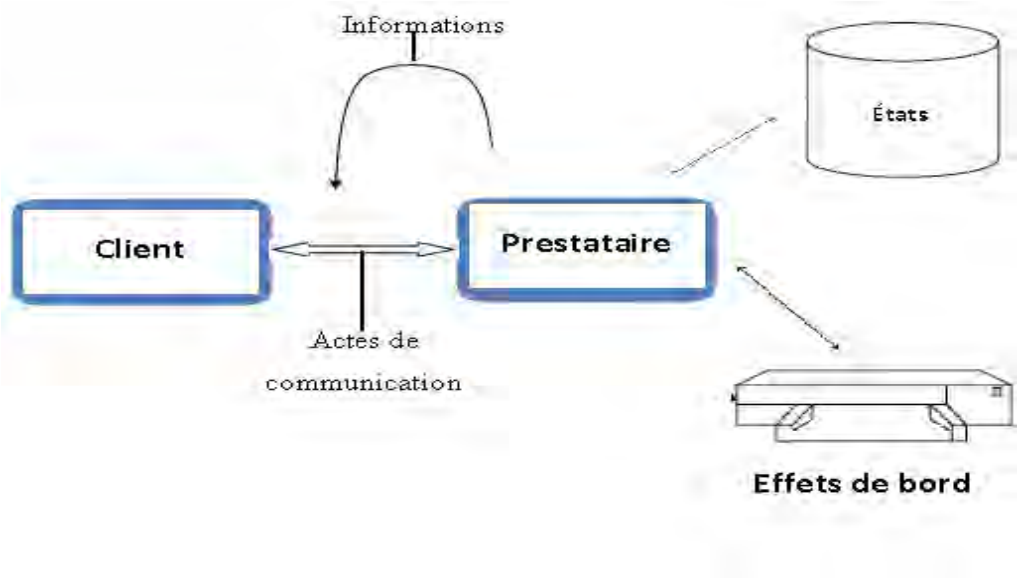


Figure 1 : Eléments d'une prestation de service.

C. Contrat de service

La description d'une relation de service est formalisée par un contrat de service. Ce contrat décrit les engagements réciproques du prestataire et du client du service. Toute application pouvant satisfaire les engagements du prestataire peut interpréter le rôle de prestataire (Idem pou le Client). Ce contrat doit être facilement extensible, aussi doit être lisible par des agents humains et exploitable par des agents logiciels (généré, interprété, agrégé, indexé, mémorisé,…, etc.). Donc le choix d'un format universel et extensible de structuration de documents comme XML s'impose.

Les éléments d'un contrat de service sont organisés en six parties [**4**] :

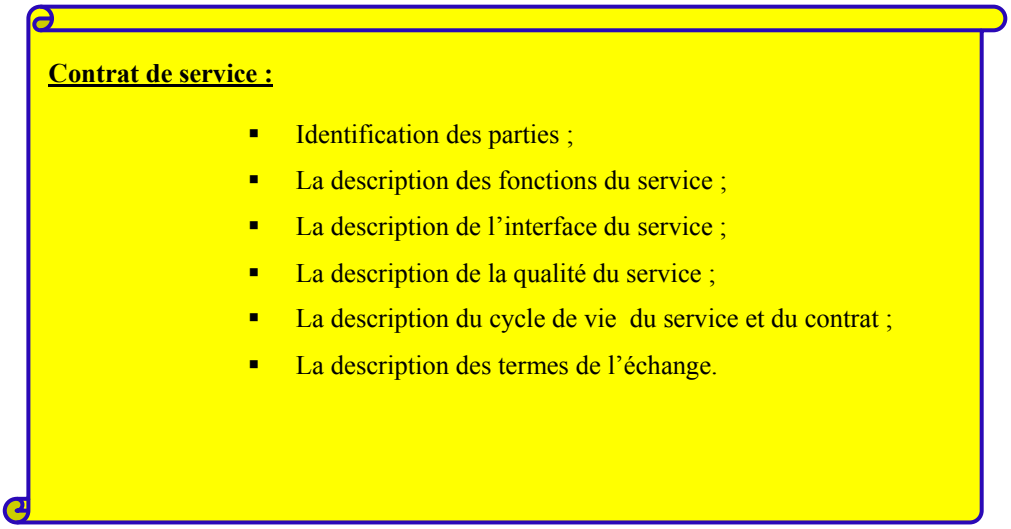


Figure 2 : Contrat de service.

- **Identification des parties** : L'identification de toutes les parties n 'est p as obligatoire. Par exemple identification du prestataire est en générale nécessaire, alors que les clients peuvent rester anonymes.
- **Fonctions du service** : Un système en générale peut être décrit en termes d'objectifs (à quoi il sert), de moyens d'interactions (comment on s'en sert), d'informations et de règles de fonctionnement du service (connaissance permettant au prestataire de service d'accomplir sa tache).

  La définition des fonctions du service est la définition de l'objet du contrat. C'est-à-dire ce que l'application prestataire fait en termes de restitution d'information, de gestion d'états, de gestion des effets de bord, d'échange d'actes de communication) et ce que les données qu'elle échange signifient [**4**].

  Dans le modèle fonctionnel, le système utilise les informations et les règles en sa possession pour sélectionner les actions qui lui permettent d'accomplir ses objectifs. Ce modèle est différent de celui de l'implémentation qui est une description technique de l'application logiciel qui met en œuvre les fonctions du service. Il faut noter qu'il est incorrect d'inclure les modèles d'implémentation des applications prestataires dans les contrats de service. L'implémentation de l'application prestataire est donc une boîte noire par rapport au contrat de service, sauf pour ce qui touche l'implémentation de la communication [**4**].

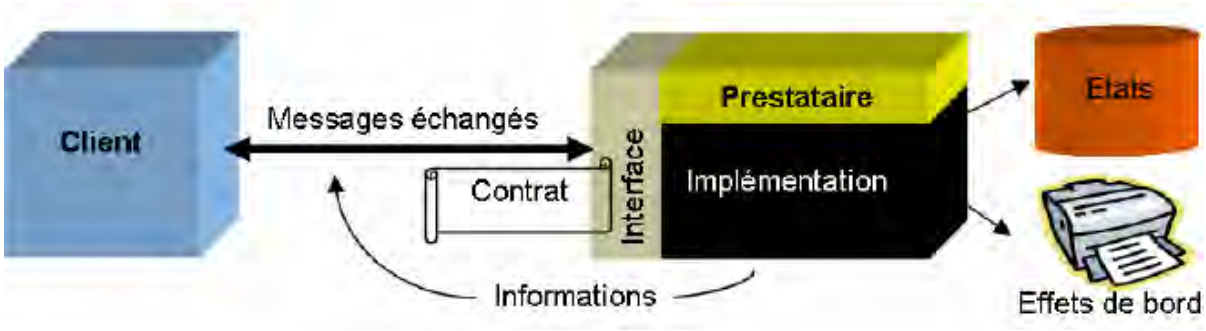


Figure 3 : Architecture boîte noire avec une interface transparente.

- **Interface du service** : Une architecture orienté services se caractérise par le fait que les interfaces de service constituent les seuls moyens de communication entre agents participants. La description de l'interface du service s'articule en deux parties :
  - **Description de l'interface abstraite** : La description de l'interface du service est donc d'abord la description des actes de communication entre le client et le prestataire dans le cadre de la réalisation de la prestation de services. L'interface abstraite qui représente l'ensemble des actes de communication entre le client et le prestataire, indépendamment des moyens utilisés.
  - Description de **l'implémentation de l'interface** : cette description comprend quatre parties :
    - **L'interface concrète** : une interface abstraite peut être implémenté par plusieurs interfaces concrètes. Ces derniers permettent de définir les différents styles d'échange et les différents formats de messages.
    - **Liaison** : une interface concrète est mise en œuvre sur une infrastructure de communication (liaison). une infrastructure de communication est constituée d'une convention de codage du contenu du message et d'un protocole de transport.
    - **Désignation des ports de réception** : Désignation du ou de s adresses des prestataires via Leurs URI et leurs n uméros de ports. A la place de la désignation des ces adresses, des mécanismes d'indirection qui permettent de découvrir ces adresses au moment de la réalisation de la prestation (par exemple un annuaire). Le langage WSDL 1.1 pe rmet de spécifier les adresses de communication via les éléments port.

Chaque port établit une association entre une liaison (implicitement, une interface) et une seule adresse de communication toujours sous format URI.

- **Chaines d'acheminement (routing) :** décrit la chaine d'intermédiaires qui joue des rôles de prestataires de services secondaires (par exemple de sécurité). Les informations nécessaires à l 'acheminement des messages de la part des intermédiaires sont notamment c ontenues dans l'en-tête, alors que l e corps du message étant destiné au récepteur final. Le chemin du m essage peut être construit de façon dynamique pendant la transmission du message.

❖ **Qualité de service** : La qualité de service est un ensemble de propriétés opérationnelles du service que l'on doit constater dans la réalisation de la prestation. Ces propriétés représentent un ensemble d'exigences concernant la mise en œuvre du service. Ces propriétés peuvent être réparties en six groupes :

- **Le périmètre de la prestation** : la partie du contrat sur le périmètre de la prestation précisent :
  - Limites de la prestation : il s'agit de préciser le caractère optionnel de certaines fonctions, de préciser explicitement les exclusions;
  - Droits et obligations du client ;
  - Conformité aux normes et aux standards : gestion des versions et la configuration de ces normes et standards qui sont par définition évolutifs.
- **La qualité de fonctionnement** : c'est un ensemble de propriétés opérationnelles qui caractérisent la réalisation des fonctions :
  - Dimensionnement : il s'agit de deux mesures de type nombre d'objets et de taille d'objets manipulés ;
  - Exactitude et précision : l'exactitude est une mesure de la dérivation de la valeur véhiculée par le prestataire par rapport à l a valeur théorique exacte (erreur standard). Alors que la précision est la mesure de la finesse de la valeur ;
  - Performance : il s'agit des mesures des critères de performances tel que le délai et le débit ;

- Accessibilité : est la probabilité d'obtenir avec succès la prestation à un instant T ;

- **La sécurité** : le contrat de service formalise les exigences de sécurité de la prestation de services (authentification, contrôle d'accès, confidentialité, intégrité, non-répudiation) ;
- **La robustesse** : c'est la résistance aux défaillances. La robustesse est l'ensemble de propriétés opérationnelles du service qui définissent par exemple:
  - Les taux d'exposition aux défaillances (fiabilité, disponibilité) du prestataire. Pour la fiabilité on peut distinguer : la fiabilité des échanges, la fiabilité fonctionnelle du service et la fiabilité des serveurs. La fiabilité de la transmission est la probabilité de transmission d'un message dans son intégralité, éventuellement dans la séquence d'émission, éventuellement sans répétition. La fiabilité fonctionnelle est une mesure de la conformité entre l'implémentation des fonctions du service de la part du prestataire et leur définition dans le contrat. La fiabilité des serveurs est une mesure de durée de service ininterrompu. la disponibilité se mesure comme la probabilité d'un prestataire d'être en service. Il faut noter que pour augmenter la disponibilité, il faut augmenter la fiabilité et diminuer le temps de rétablissement du service.

❖ **Cycle de vie du service et du contrat** : Le déroulement de la prestation de service peut être géré de manière explicite par:

- Service de gestion du cycle de vie du service primaire (activation, suspension, redémarrage, arrêt) ;
- Service de pilotage du service primaire via la modification dynamique des paramètres de la prestation ;
- Service d'interrogation de l'état du service primaire ;
- Service de journalisation des activités du service primaire ;

❖ **Description des termes de l'échange** : Décrit si le service est gratuit, Payant ou troqué. Dans un service troqué, la prestation est exécutée en échange de prestations de services compensatoires et cet échange est formalisé dans le contrat.

D. Le base de l'AOS

La base d'une AOS repose sur des services répondant, notamment, aux critères suivants :

- **Faiblement couplée** : Un service est en couplage faible quand il n'est pas autorisé à appeler directement un autre service. Il délègue cette responsabilité à un traitement spécialisé que l'on nomme fonction d'orchestration. Grâce au couplage faible, on peut réutiliser un service sans devoir reprendre des services qui lui sont liés. A partir de cette première définition, on ajoute d'autres principes techniques qui contribuent à découpler les services entre eux :
  - Un service est activable indépendamment de sa t echnologie. Pour ce faire, l'activation se réalise par l'envoi (et la réception) d'un message XML.
  - Un service peut être activé suivant un mode asynchrone. Dans ce cas, le service s'abonne à un événement auprès de la fonction d'orchestration.
- **Distribution** : Là où l'entreprise devait maintenir certaines fonctions et données comme les tarifs de fournisseurs, elle pourra interroger directement, via son applicatif, le service du fournisseur sans se préoccuper de sa mise à jour.
- **Invocation et publication** : Les services doivent être invocables et publiables quels que soient les systèmes utilisés.
- **orientation métiers** : Les services permettent de gérer les applications avec une approche fonctionnelle, par l'intermédiaire de processus métiers intégrés à l'architecture, permettant de piloter les applications sans développement conséquent.

E. L'agrégation et la dissémination de service

Une architecture orientée services peut être conçue par une approche incrémentale, résultat de la combinaison de deux démarches de base, présentées ci-dessous, qui sont [**10**] :

- **L'agrégation de services** : Plusieurs applications se r épartissent les tâches et les publient sous forme de services. Chaque service est issu de la composition (ou de l'orchestration) de services atomiques.

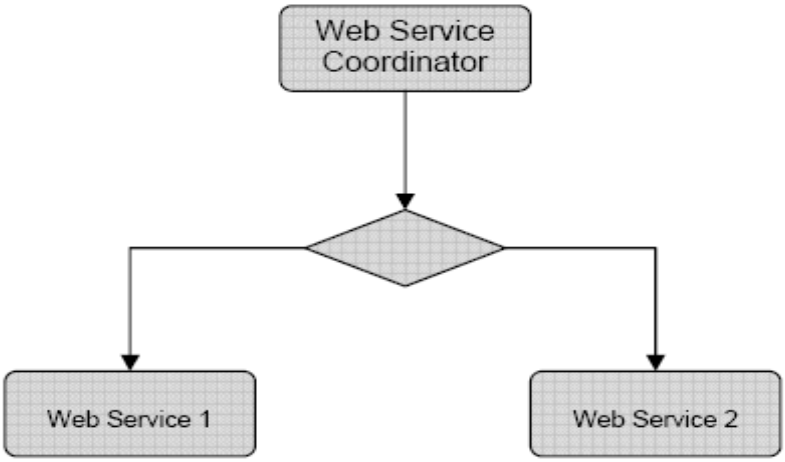


Figure 4 : L'agrégation de services.

- **La dissémination de services** : La dissémination de services permet d'effectuer la décentralisation des données. Donc on pe ut avoir plusieurs applications qui sont prestataires du même service sur des données différentes.

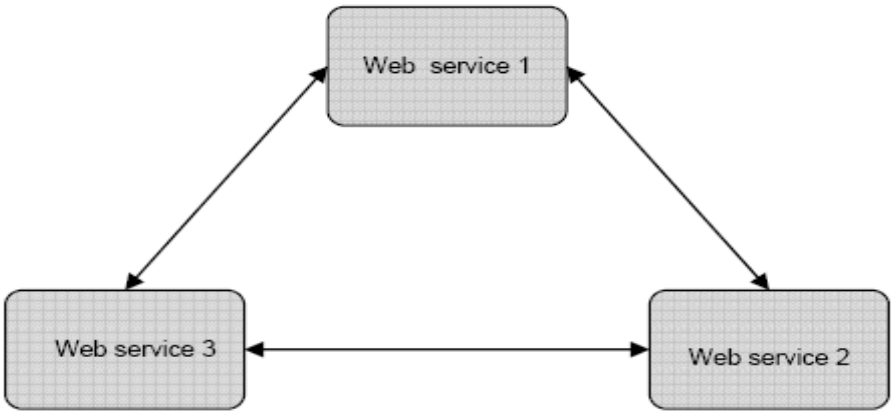


Figure 5 : La dissémination de services.

F. L'architecture orientée services && les services Web

La cible des technologies de service Web est bien les architectures orientées services. Un service Web est naturellement une application orientée services. Mais l'inverse ce n'est pas vrai. Selon la définition du W 3C, une application orientée services peut être qualifiée de service Web si elle exhibe le profil technologique suivant :

- Identification : le service Web est identifié par un URI (Uniform ressource Identifier). Un URI c'est un mécanisme permettant aux utilisateurs de localiser leurs ressources.
- Description : les interfaces et les liaisons d'un service Web sont décrites (et donc peuvent être définies et découvertes) au moyen d'un langage XML ;
- Message : un service Web communique avec les autres agents logiciels au moyen de messages au format XML ;
- Transport : les messages sont transmis via des protocoles Internet.

Identification	URI
Description	Language XML (WSDL)
Message	Format XML (SOAP, XML)
Transport	protocoles Internet (HTTP, SMTP, …etc.)

Figure 6 : profil technologique d'un service Web

Dans la mise en place des architectures de service Web, la fonction du contrat est essentiellement portée par le document SWDL. Un document WSDL permet de définir l'implémentation de l'interface, à savoir :

- Les styles d'échange ;
- Les formats de messages ;
- Les conventions de codage ;
- Les protocoles de transport ;
- Les ports de réception.

Pour approfondir la notion d'architecture des services Web, voyons la façon dont cette architecture est mise à contribution. La figure suivante illustre une utilisation typique des composants de cette architecture :

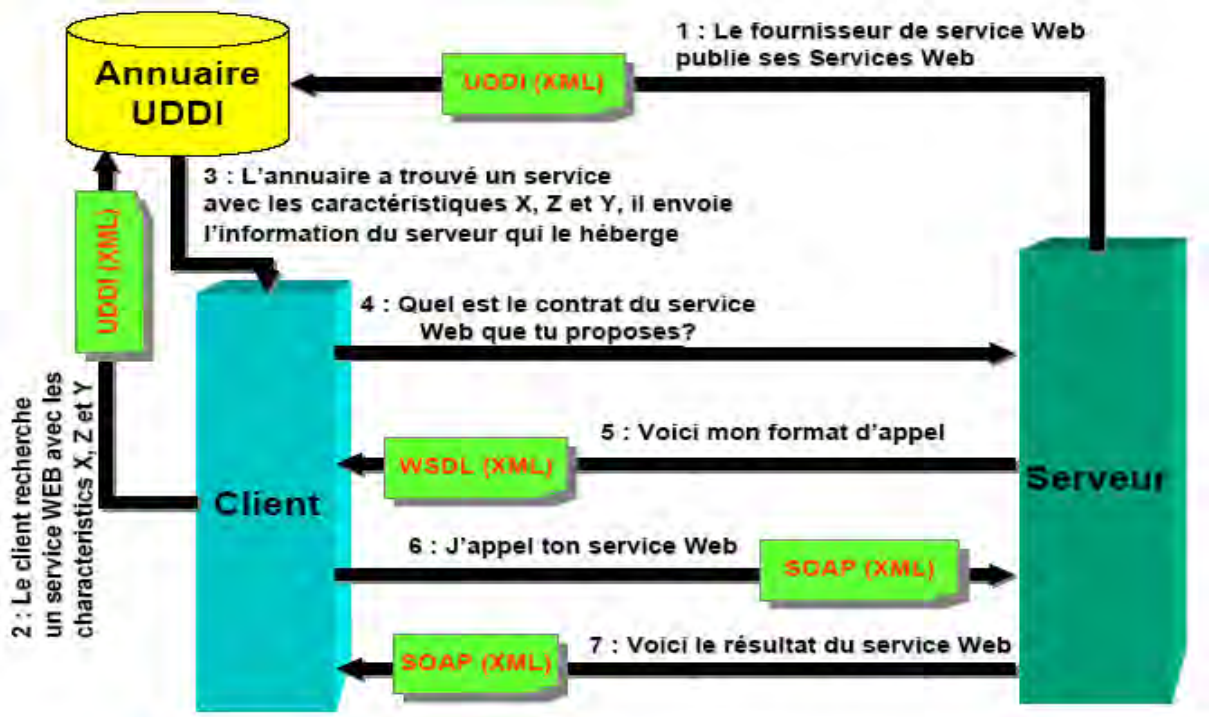


Figure 7 : Architecture générale de Service Web.

1. Tout d'abord, le fournisseur crée le service Web. Ce service doit s'inscrire auprès d'un référentiel UDDI en indiquant plusieurs de ses caractéristiques (l'interface ainsi que les informations d'accès au service), y compris sa description WSDL ;
2. L'application cliente va consulter l'annuaire de service (un référentiel UDDI) afin de sélectionner le service Web qu'elle souhaite utiliser. L'annuaire de service rend disponible l'interface du service ainsi que ses informations d'accès pour n'importe qu'elle cliente potentielle ;
3. Une fois que le service Web est sélectionné, on peut consulter sa description WSDL pour savoir comment interagir avec celui-ci. Cette description permet de connaitre quel message SOAP est à envoyé et quelle message SOAP sera reçu en retour du traitement souhaité ;
4. Il faut à présent contacter le service Web, afin de demander le contrat du service Web que le serveur (fournisseur) propose, et donc établir la communication avec ce dernier. Cette connexion à lieu lors de l'envoi du premier message SOAP ;
5. Le message reçu du coté de service Web indique le format d'appel ;

6. Le client utilise la description du service Web pour invoquer ce service ; Si aucune erreur ne se produit, une réponse est émise à l'application cliente.

G. Conclusion

L'AOS est l'aboutissement de plusieurs années de corrections des erreurs rencontrées dans les applications. Ce n'est pas à proprement parler une révolution technologique. Elle permet de créer de manière incrémentale de nouvelles fonctions, de les combiner aux services existants pour créer des applications composites.

Dans cette partie nous avons abordé en détail les différents éléments clé et concepts utilisés dans cette architecture. Nous avons également étudié le rapport entre les technologies de service Web et les architectures orientées services.

Dans la partie suivant on va détailler les technologies des services Web unitaire, à savoir SOAP, WSDL et UDDI.

Partie II : Introduction aux services Web

II. Introduction aux services Web

De plus en plus, avec l'essor d'Internet, le développement tend vers les technologies du Web. Il n'est pas évident de distinguer les différents logiciels qui sont de plus en plus intégré au WEB. Les Web Services sont des applications modulaires basées sur Internet qui exécutent des tâches précises et qui respectent un format spécifique.

On peut trouver également une autre définition montrant qu'un service Web correspond à une extension du Web le faisant passer d'une infrastructure proposant des services aux personnes à une infrastructure offrant des services à des logiciels qui cherchent à coopérer avec d'autres logiciels [**1**].

A. Historique des services Web

Les services web prennent leur origine dans les systèmes repartis et dans l'avènement du Web. Les systèmes repartis datent des années quatre-vingt avec la migration progressive de systèmes centralisées (où tous les traitements, incluant la gestion des données et celle des calculs, sont exécutés depuis une unique et volumineuses application) vers des systèmes repartis.

Le but de la répartition est de permettre à une application sur une machine d'accéder à une fonction d'une autre application sur une machine distante et ce, de la même manière que l'appel d'une fonction locale, indépendamment des plates-formes et des langages utilisés. Donc on peut imaginer que l'application est divisée en plusieurs morceaux, chaque morceau peut être développé dans un langage de programmation différent et exécuté depuis un système d'exploitation différent. Egalement, il est important de noter que chaque morceau joue un rôle différent pour les traitements du système d'information.

Actuellement trois standards sont utilisés pour mettre en pratique les applications reparties, chacun de ces st andards offrent des mécanismes de récupération d'espace mémoire, de sécurité, de gestion du cycle de vie des objets. Il s'agit du standard CORBA/IIOP (*Common Object Request Broker Architecture / Internet Inter-ORB Protocol*) spécifié par l'OMG (*Object Management Group*), de la standard DCOM (*Distributed Component Object Model*) de Microsoft et RMI (*Remote Method Invocation*) de Sun. Le choix entre ces standards est lié à la plate forme employée. DCOM s'impose rapidement comme la solution de développement pour la plate forme Windows. Par contre, CORBA s'impose comme la solution inter pates-formes (Unix, Windows).

Mais le problème qui se pose est comment faire collaborer des environnements reparties (établir des liens entre ces environnements). Ce problème peut arriver en pratique dans le cas d'une fusion entre deux entreprises ayant fait dans le passé un choix différent en terme d'environnement reparti.

Avec l'apparition des termes B2B (*Business to Business*) et B2C (*Business to Customer*) qui désignent respectivement des plates formes de gestion des échanges commerciaux entre entreprises ou d'une entreprise vers ses clients par l'intermédiaire de l'internet, il était clair aux yeux des entreprises la nécessité de tels technologies, qui permettent non seulement de faire des liens de communications entre ces entreprises, mais également qui limitent la nécessité d'un système de middleware réparti commun pour pouvoir communiquer. La solution c'est un modèle d'architecture solide basé seulement sur des protocoles et des technologies de communication interopérables sur Internet.

C'est l'apparition des services Web. L'apport principal des services web est la solution au problème d'hétérogénéité par rapport aux plates-formes et langages des applications.

En fin, les services permet non seulement des conserver les caractéristiques des architectures distribuées (comme CORBA ou RMI) qui est indispensable pour répondre aux besoins en terme de complexité des applications d'entreprises, mais permet également de satisfaire les contraintes imposées par le WEB, à savoir : client léger qui est souvent réduit à un simple navigateur et la mise en œuvre du protocole HTTP.

B. Définition des services Web

W3C définit un service Web comme une application logicielle identifiée par un URI dont les interfaces et les liaisons sont définies, décrites et découvertes en XML et supporte une interaction directe avec les autres applications logicielles en utilisant des messages XML via un protocole Internet [**2**].

C. Caractéristiques

Le service web se caractérise en effet par une standardisation des implémentations, par une localisation à distance des services et par une récupération de l'interface d'accès permettant l'exécution du traitement correspondant. Mais les caractéristiques principales sont :

- **Réutilisable** : Une fonctionnalité développée sous forme de Web services peut dorénavant être réutilisée et recombinée à une suite d'autres fonctionnalités pour composer une nouvelle application.

- **indépendamment de la plate-forme** (UNIX, Windows,…), de leur environnement d'implémentation (Java, C++, Visual Basic,…) et de l'architecture sous-jacente (.NET, J2EE,…) ;
- **Web based** : les Web services sont basés sur les protocoles et les langages du Web, en particulier HTTP et XML.
- **Self-described, self-contained** : le cadre des Web services contient en lui-même toutes les informations nécessaires à l'utilisation des applications, sous la forme de trois fonctions : trouver, décrire et exécuter.
- **Modular**: les Web services fonctionnent de manière modulaire et non pas intégrée. Cela signifie, qu'au lieu d'intégrer dans une seule application globale toutes les fonctionnalités, on crée (ou on récupère) plusieurs applications spécifiques qu'on fait inter-coopérer entre elles et qui remplissent chacune une de ces fonctionnalités.

D. Les spécifications de services Web

Les éléments qui constituent la mise en œuvre d'un service web sont :

- Le protocole d'accès ;
- La description du service ;
- L'annuaire des services.

1. Protocole d'accès (SOAP)

Pour accéder aux services web l'architecture est basée sur le protocole SOAP (en anglais, *Simple Object Access Protocol*). Le protocole SOAP est un standard qui permet la communication entre applications pour l'accès aux services web. Il identifie le réseau internet comme le réseau utilisé pour accéder aux services web, mais il n'identifie pas nécessairement le protocole de niveau application qui sera utilisé pour réaliser cette communication. En effet, plusieurs protocoles de communication sur l'Internet peuvent être utilisés avec SOAP comme par exemple HTTP, SMTP), mais ce standard recommande l'utilisation du protocole HTTP comme protocole de communication principal des services Web. Cette recommandation vient du fait que le protocole http est largement accepté, et qu'il a montré sa capacité à gérer les très nombreux échanges sur Internet.

SOAP est développé conjointement par Microsoft, IBM, Lotus Development (une division d'IBM), DevelopMentor et UserLand Software sous les auspices du W3C. SOAP 1.1 fit

l'objet d'une note soumise au W3C en mai 2000, et SOAP 1.2 d' un document de travail (*working draft*) en juillet 2001 [**3**].

Ce protocole est indépendant des langages de programmation ou des systèmes d'exploitation employés pour l'implémentation des services Web. Il repose sur XML et sur quelques standards dérivés, les NameSpaces et XML Schema, en particulier.

a. Terminologie SOAP

La structure de la spécification SOAP peut être organisée à plusieurs niveaux comme le montre à la figue suivante [**4**] **:**

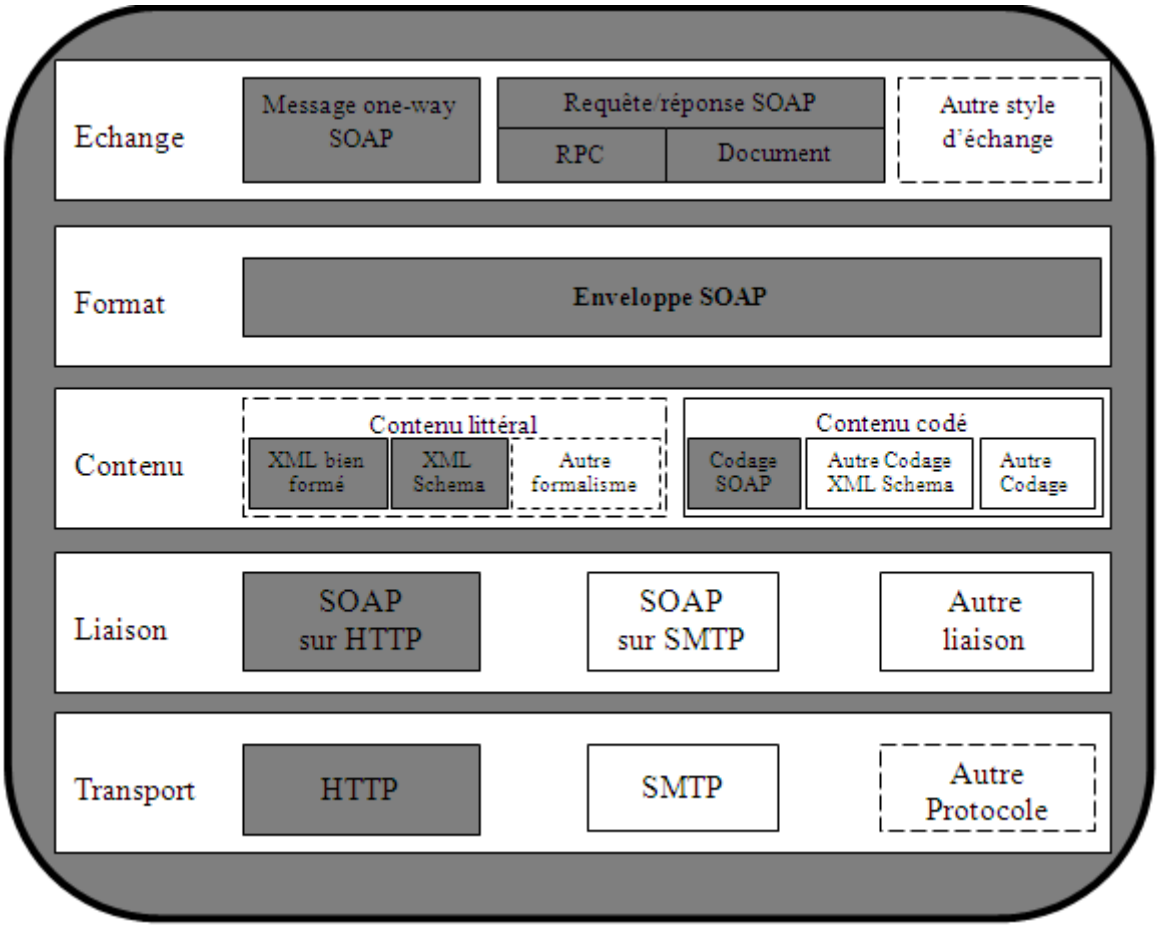


Figure 8 : Structure de la spécification par niveaux de SOAP

Style d'échange : Différents styles d'échange peuvent être utilisés dans l'architecture orienté service, SOAP permet de mettre en œuvre plusieurs styles d'échange :

- Message à sens unique : il s'agit d'un échange unidirectionnel d'un message SOAP. C'est le style de base ;
- Requête/réponse : dans ce mode l'envoie d'un requête SOAP est suivi d'un message de réponse. Ce style est en fait définit implicitement (via la liaison générique

SOAP/HTTP). Ce style s'applique à la mise en place de l'appel de procédure distante (RPC : Remote Procedure Call) et le style document qui est définit implicitement comme le complément du RPC ;

- **RPC (*Remote Procedure Call*)** : appel via un réseau d'une procédure ou d'un programme exécuté sur une machine distante (ou dans un espace d'adresses différent sur la même machine). Ce style présente de nombreux avantages. Mais l'avantage essentiel est son caractère intuitif par apport à des habitudes de programmation généralement acquises. Ce style est offerts par un plusieurs éditeurs et fournisseurs de systèmes d'exploitation. Par exemple ce style est présent dans DCE, CORBA, RMI.

  Pour mettre en œuvre ce style de requête/réponse, la requête doit contenir une représentation de l'invocation de la procédure. La réponse contient soit une représentation du compte rendu de réussite de l'exécution de la procédure, ainsi qu'éventuellement des données résultat de l'exécution de la procédure. Soit un compte rendu d'échec, sous forme de message d'erreur.

  L'exécution de l'appel de procédure distante (RPC) peut être synchrone ou asynchrone. En RPC synchrone, le thread à l'origine de l'appel RPC bloque le client jusqu'à ce que l'appel RPC soit terminé. Le mode RPC asynchrone permet au thread ayant émis l'appel de poursuivre son exécution et de récupérer les résultats ultérieurement. Lorsque les temps de latence de la livraison des messages ne peuvent pas être contrôlés, l'emploi de messages asynchrones devient nécessaire.
- **Style document** : Pour comprendre le style document on doit d'abord aborder le passage d'objet par référence et celui par valeur. Pour le passage par référence l'émetteur doit d'abord connaitre l'identifiant unique de cette objet sur le réseau. Puis utilise l'une de deux approches (service ou objet). Dans l'approche objet l'application client obtient l'identifiant technique de la copie en mémoire du serveur de l'instance d'objet visé. Ensuite l'application cliente peut invoquer les méthodes (procédures) qu'elles possèdent via son identifiant. Dans l'approche service un identifiant technique d'une instance de la class qui fait office de représentant de service est utilisé. Cet identifiant est utilisé comme le point d'entrée du service. Pour finir sur le passage par référence, celui-ci peut être effectué entre programmes mis en œuvre par des langages de programmation différents.

Le passage d'objet par valeur est nécessaire dans la mesure où le passage par référence est déconseillé (et ceux-ci pour des raisons de performance, de fiabilité et de robustesse de l'application). Dans ce cas, ce qui est véhiculé est bien un structure de données et non pas un objet. Pour mettre en œuvre l'invocation des méthodes distant par le passage par valeur l'homogénéité des environnements de mise en œuvre et le partage de code entre les participants (Client/serveur) est nécessaire. Sans ces deux conditions, la structure passée est une structure de données, et les procédures d'encodage/décodage et les méthodes de manipulation sont différentes entre client et serveur. Cette dernière approche est appelée approche *document*.

Dans cette approche (*document*) la structure de données dans la requête et la réponse est un doc ument qui est encodé décodé, manipulé de façon indépendante par le client et le serveur [**4**].

Format du message : Un message SOAP est un item d'information de type document XML qui contient trois éléments : une enveloppe <Envelope>, un en-tête <Header> et un corps <Body>. La Structure d'un message SOAP se divise en quatre parties :

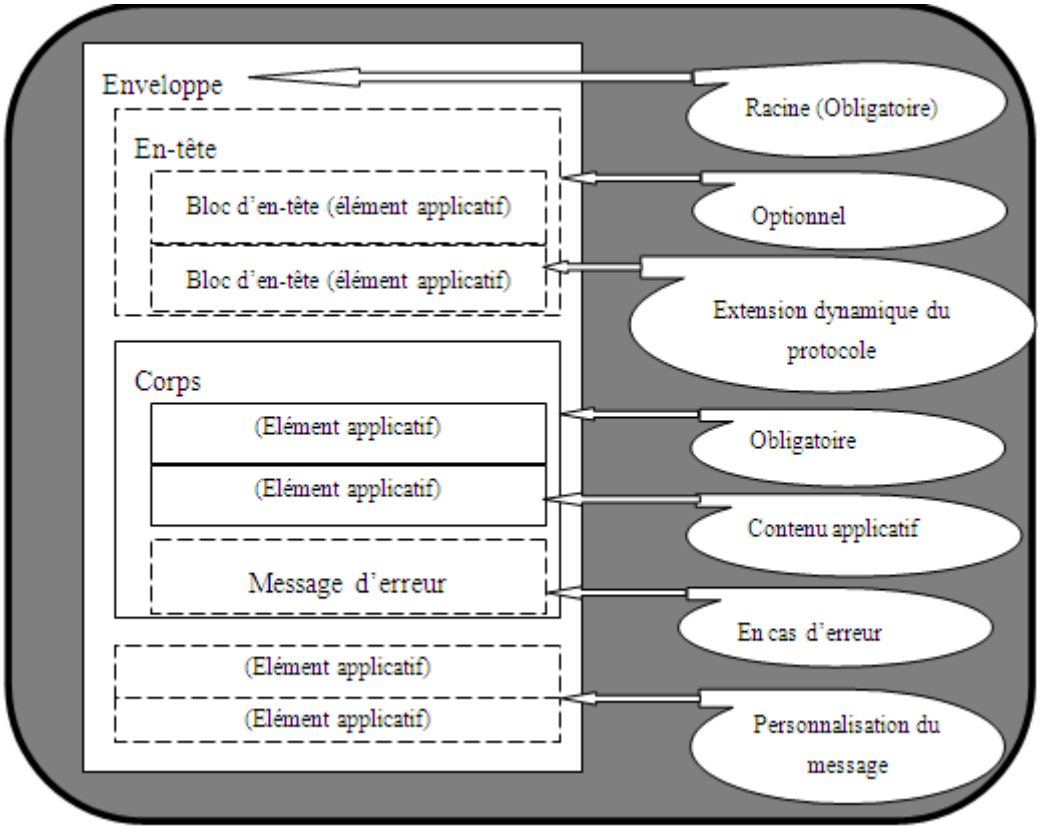


Figure 9 : Structure d'un message SOAP.

- **L'enveloppe SOAP** : Les messages sont emballés dans une enveloppe SOAP. L'enveloppe est l'élément racine du message SOAP. La déclaration de cette enveloppe SOAP doit spécifier l'espace de noms dont l'URI précise la version supporté de SOAP (1.1 ou 1.2). Si aucune version du protocole n'est indiquée dans l'enveloppe un nœud SOAP doit le rejeter et éventuellement d'envoyer immédiatement un message d'erreur (dont le code est SOAP-ENV:VersionMismatch). On doit également spécifier le format de données échangées qui sera défini par un modèle d'encodage. La déclaration du modèle d'encodage est optionnelle.

  L'enveloppe SOAP contient un élément d'en-tête optionnel et un élément corps obligatoire.

- **L'en-tête de message SOAP** : L'en-tête du message SOAP est un élément optionnel. Son but est de permettre la mise en place d'extensions aux mécanismes de base définis par SOAP. En effet, Il peut être utilisé pour véhiculer des informations liées à la gestion de sécurité ou à la gestion des transactions. Par exemple cet en-tête peut utiliser pour transmettre des informations d'authentification ou de gestion de session. Il est important de noter que SOAP ne traite pas du contenu de l'en-tête, ni du routage des messages entre les nœuds, …cela dépend juste de l'application en question.

  L'en-tête peut contenir plusieurs éléments descendants. Chaque élément enfant de l'en-tête est nommé bloc d'en-tête. SOAP définit plusieurs attributs bien connus qui peuvent servir à indiquer qui est concerné par le bloc d'en-tête (*actor*) et si le traitement est optionnel ou obligatoire (*mustUnderstand*). L'utilisation des ces attributs est recommandé par exemple par SOAP1.1.

- **Le Corps de message SOAP** : Le corps d'une enveloppe SOAP est toujours exploité par le destinataire final (il fournit un mécanisme de transmission d'information vers ce récepteur). En revanche, les en-têtes SOAP peuvent être traités par les intermédiaires comme par le destinataire final. L'élément Corps est obligatoirement présent dans un message SOAP 1.1 comme descendant direct de l'élément enveloppe, suivant immédiatement l'élément en-tête (si présent) et suivi éventuellement par les éléments propriétaires.

  L'élément Corps peut contenir un ensemble d'éléments descendants. Ces éléments sont appelés entées du corps. Des éléments particuliers pour indiquer que des erreurs sont produites lors de traitement peuvent exister dans l'élément Corps. Avec SOAP 1.2 chacun de ces éléments doit avoir une propriété [namespace name] avec une valeur (C'est-à-dire doit être qualifié par un espace de nommage).

Codage de contenu du message && SOAP 1.1 : Les services Web sont mis en œuvre par des programmes écrits à l'aide de différents langages de programmation. Les données manipulées par ces langages sont toujours typées (de façon statique ou dynamique). L'objectif de SOAP est de mettre en œuvre un mécanisme d'échange indépendant des choix d'implémentation et notamment des langages de programmation. Ce mécanisme doit permettre de définir des représentations (codage) dans les messages SOAP des données manipulées par ces langages et de mettre en œuvre les règles associées d'encodage et de décodage.

SOAP définit une représentation en XML de son modèle de données. Cette représentation et éventuellement les règles d'association sont appelés style de codage (encoding style). L'objectif est que le développeur puisse continuer à travailler dans la représentation de données propre au langage de programmation qu'il utilise, sans poser de questions au sujet du format et du codage des données dans le message SOAP.

Le problème qui se pose est comment SOAP va permettre de transporter les fichiers binaires ? SOAP utilise un mécanisme qui permet de transférer les fichiers binaires sous forme des pièces jointes pour résoudre ce problème.

Pour approfondir, l'objectif principal de SOAP 1.1 était d'apporter une meilleure intégration entre les technologies des objets et des composants répartis (DCOM, CORBA et RMI) avec les technologies Internet (XML et http). C'est pour cela que SOAP 1.1 a fournit une représentation universelle. Ce format permet à chaque langage de programmation de mettre en œuvre une procédure d'encodage vers ce format (coté Serveurs) et une procédure de décodage (coté clients) à partir de cette représentation. Ce format universel est nécessaire lorsqu'il s'agit de garantir l'interopérabilité entre applications faiblement couplées est mis en œuvre dans des différents langages de programmation. En plus, pour des raisons de performance, il peut être nécessaire d'utiliser une représentation spécifique lorsque le même langage de programmation est utilisé dans les deux coté (Client et serveur).

Dans le service une interface publiée peut avoir plusieurs liaisons (bindings). Par exemple une liaison avec un style de codage universel et autre avec un style spécifique lorsque le même langage est utilisé dans les deux coté.

Le style de codage spécifié par SOAP 1.1 est précisé par l'URI http://schemas.xmlsoap.org/soap/encoding. Qui doit apparaitre explicitement dans tous messages respectant ce format de style.

Liaison && Indépendance du transport : SOAP est défini indépendamment du mécanisme sous-jacent de transport des messages. Il autorise l'emploi de différents moyens de transport pour l'échange de messages, et permet à la fois des transferts et des traitements synchrones ou asynchrones. Mais même avec cette indépendance générale vis-à-vis du transport, la plupart des services Web communiquaient via HTTP, l'un des principaux canaux de communication inclus dans les spécifications SOAP. HTTP est un protocole bidirectionnel synchrone, formé par le couple indissociable requête/réponse HTTP.

2. La description du service (WSDL)

C'est une description basée sur le langage XML, qui indique quelles sont les opérations disponibles?, comment on y accède (adresse, protocole, …) ?, quel est le format de messages échangés entre le client et le serveur :

- pour invoquer le service ?;
- pour interpréter les résultats ?

La description est réalisée en se basant sur le langage WSDL (*Web Service Description Language, en français un langage XML de description de service Web*) qui est une grammaire dérivée de l'XML. Cette description permet de cacher les détails de l'implémentation du service, ce qui permet d'offrir une utilisation indépendante de la plate forme et du langage utilisés.

Le langage WSDL est une proposition conjointe des sociétés Ariba, IBM et Microsoft auprès du W3C dont la première spécification fut publiée en septembre 2000 [**1**]. Le W3C vient de finaliser la version WSDL 2.0 (27 juin 2007) prenant complètement en charge le principal protocole Web, HTTP, ainsi que le protocole de services Web le plus couramment utilisé, SOAP. WSDL 2.0 intègre les correctifs apportés à WSDL 1.1 par le profil WS-I Basic, offre des fonctions améliorées de description de fautes et d'erreurs, d'héritage et d'importation plus nombreuses, et prend en compte HTTP et SOAP.

a. Terminologie WSDL

Dans la mise en place des architectures de Services Web aujourd'hui, la fonction de contrat de service (ce concept sera abordé dans la partie suivante) est portée par le document WSDL. Le document WSDL peut être divisé en deux parties. Une partie abstraite qui est constituée des définitions abstraites (les types, les messages, et les types de port), et une partie concrètes qui contient les descriptions concrètes (les bindings ou liaisons, les services). La partie abstraite

définit les messages SOAP d'une façon totalement indépendante de la plate-forme et de la langue. Cela facilite la définition d'un ensemble de services pouvant être implémenté par différents sites Web. La partie concrète propose une ou plusieurs réalisations de la partie abstraite (par exemple, un type de port peut être réalisé par SOAP+RPC+HTTP et/ou SOAP+RPC+SMTP).

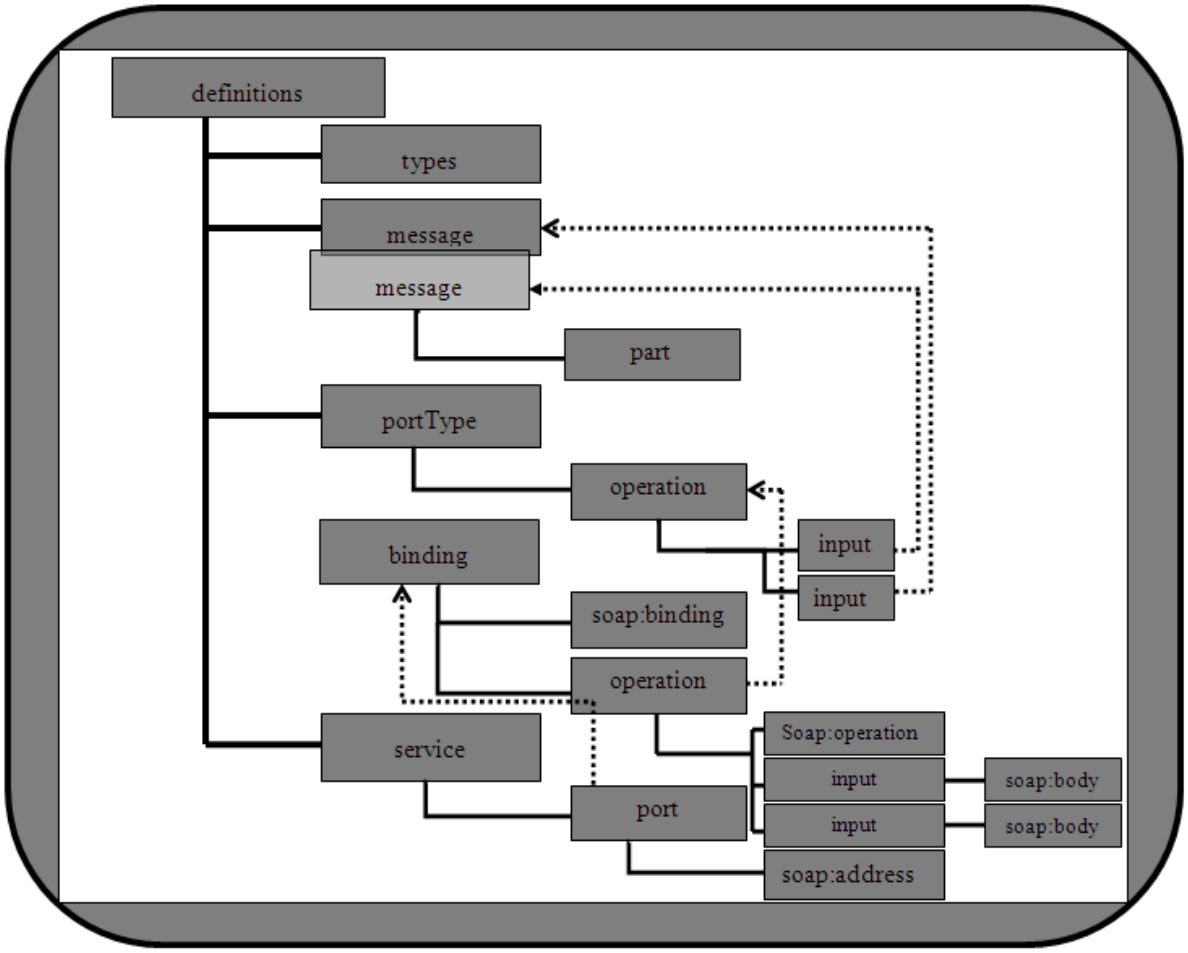


Figure 10 : Structure de la spécification par niveaux de WSDL.

Nous allons maintenant présenter en détail chaque composant majeur d'un document WSDL. Le fichier WSDL est principalement composé de 6 éléments, avec d'autres éléments optionnels [**7**] :

- **Definitions** : élément racine du document, il donne le nom du service, déclare les espaces de noms utilisés, et contient les éléments du service (Obligatoire). Il peut être associé à un espace de noms particulier optionnel via l'attribut targetNamespace. Cette attribut est de type URI et doit être obligatoirement absolu.

  Avant d'aborder le deuxième élément, il est important de noter, qu'il est possible d'importer un ou plusieurs fragments de documents dans un document WSDL. Cela

est réalisé via l'élément import. L'utilisation de l'élément import permet de séparer les différents éléments d'une définition de service en documents indépendants, qui peuvent ensuite être importés si nécessaire. Cette technique offre le moyen d'écrire des définitions de services plus claires, en séparant les définitions en fonction de leur niveau d'abstraction.

- **Types** : décrit tous les types de données utilisés entre le client et le prestataire. WSDL n'est pas exclusivement lié à un système spécifique de typage, mais utilise par défaut la spécification XML Schema (Définition abstraite).
- **message**: décrit un message unique, que ce soit un message de requête seul ou un message de réponse seul. L'élément définit le nom du message et peut contenir (ou pas) des éléments « part », qui font référence aux paramètres du message ou aux valeurs retournées par le message (Définition abstraite).
- **portType** : combine plusieurs messages pour composer une opération. Chaque opération se réfère à un message en entrée et à des messages en sortie. (Définition abstraite). La spécification WSDL prend en charge quatre types d'opérations :
  - One-way (l'interaction à sens unique): Le client envoie une requête SOAP au fournisseur du service mais n'attend pas de réponse en retour ;
  - Request-response (la requête/réponse) : Le client envoie une requête SOAP au fournisseur du service et attend une réponse en retour ;
  - Solicit-response (demande de réponse) : Cette fois, c'est le fournisseur du service qui envoie une requête SOAP au client et qui attend une réponse en retour ;
  - Notification (notification) : Le fournisseur du service envoie une requête SOAP au client.
- **Binding** : décrit les spécifications concrètes concernant la manière dont le service sera implémenté: protocole de communication et format des données pour les opérations et messages définis par un type de port particulier. La spécification WSDL définit deux liaisons standards à des protocoles de transport :
  - la liaison avec le protocole SOAP ;
  - la liaison avec le protocole http GET/POST.
- **Service**: défini les adresses permettant d'invoquer le service donné, ce qui sert à regrouper un ensemble de ports reliés. La plupart du temps, c'est une URL invoquant un service SOAP (Définition concrète). Un port définit un nœud de communication,

et donc un URI, pour une liaison particulière. Cette liaison est repérée via l'attribut binding du port.

Liaison avec SOAP : WSDL inclut une liaison pour les points finaux SOAP, qui reconnaît la spécification des informations spécifiques au protocole suivantes :

- Une indication qu'une liaison est liée au protocole SOAP;
- Une façon de spécifier une adresse pour un point final SOAP ;
- L'URI du header SOAPAction HTTP pour la liaison HTTP de SOAP ;
- Une liste de définitions des entêtes qui sont transmises dans l'Enveloppe SOAP ;
- Une façon de spécifier les racines SOAP dans XSD.

WSDL possède des extensions internes pour définir des services SOAP; de ce fait, les informations spécifiques à SOAP se retrouvent dans cet élément. Ces éléments d'extension sont :

- **soap:binding :** cet élément est obligatoire lorsque l'on utilise une liaison SOAP dans le document WSDL. Le but de cet élément est de signaler que la liaison est liée au format du protocole SOAP : Envelope, Header et Body (à l'une de ces éléments). Cet élément ne demande rien de particulier pour le codage ou le format du message.

```
<definitions…>
        <binding…>

                <soap:binding transport="uri"? style="rpc|document"?>

        </binding>
</definitions>
```

Rappelons « dans la partie SOAP » que SOAP définit deux types de format possibles pour le transport des données entre applications : Les messages de type RPC et ceux de type Document. Le message est soit un message directement au format XML dans le cas du modèle "Document", soit les arguments d'entrées-sorties des fonctions encodées au format XML dans le cas d u modèle "RPC". L'attribut « style » s'applique par défaut à l'ensemble des opérations incluses dans la liaison. Si celui-ci n'est pas précisé, il prend la valeur « document » par défaut.

L'attribut « transport » est obligatoire et la valeur de l'URI précise le protocole de transport réel utilisé par SOAP pour la communication. L'URI http://schemas.xmlsoap.org/soap/http correspond à la liaison HTTP dans la

spécification WSDL. D'autres URI peuvent être utilisées ici pour indiquer d'autres transports (tels que SMTP, FTP, etc.).

- **soap:operation** : Cet élément se présente ainsi dans la structure du document :

```
<definitions…>
        <binding…>

                <operation…>
                        <soap:operation soapAction="uri"? style="rpc|document"?>
                </operation>

        </binding>
</definitions>
```

L'attribut « style » prend les mêmes valeurs que celui de l'élément « soap:binding » vu précédemment. Si l'attribut n'est pas spécifié, il prend par défaut la valeur spécifiée dans l'élément soap:binding. Si l'élément soap:binding ne spécifie pas un attribut Style, on suppose que c'est "document".

L'attribut « soap:Action » spécifie la valeur du header SOAPAction pour cette opération. Cet URI est obligatoire en cas de description d'une liaison du protocole sur HTTP, dans ce cas cette valeur URI doit être utilisée directement comme valeur pour l'entête HTTP SOAPAction. Pour les autres liaisons du protocole SOAP, elle peut ne pas être spécifiée, et l'élément soap:operation peut être omis.

- **soap:body :** l'objectif de cet élément est de décrire la structure du corps du message SOAP (**soapenv:Body**). L'élément soap:body est utilisé pour définir à la fois les messages orientés RPC et les messages orientés document. Si le style de l'opération est "RPC", chaque partie logique est un paramètre ou une valeur de retour qui apparaît à l'intérieur d'un élément Wrapper dans Body. Si le style est "document", chaque partie logique est un document qui apparaît directement à l'intérieur de Body.

```
<definitions…>
        <binding…>
                <operation…>

                        <input>
                                <soap:body parts="nmtokens"? use="literal|encoded"?
                                        encodingStyle="uri-list"? namespace="uri"?>
                        </input>

                        <output>
                                <soap:body parts="nmtokens"? use="literal|encoded"?
                                        encodingStyle="uri-list"? namespace="uri"?>
                        </output>
                </operation>
        </binding>
</definitions
```

L'attribut « parts » optionnel de type nmtokens indique quelles parties apparaissent à l'intérieur de la portion soap:body du message. S'il n'est pas spécifié, toutes les parties du message sont incluses dans le corps du message SOAP.

Les parties d'un message peuvent être soit des descriptions des schémas concrètes, soit des définitions de types abstraits.

L'attribut « use » obligatoire indique si les parties du message sont codées en utilisant des règles d'encodage, ou si les parties définissent le schéma concret du message.

Si la valeur encoded est spécifiée, chaque partie du message fait alors référence à un type abstrait en utilisant l'attribut type. Ces types abstraits sont utilisés pour produire un message concret en appliquant un codage spécifié par l'attribut encodingStyle.

L'attribut « encodingStyle » contient une liste d'URI séparés par une espace. Chaque URI correspond à un encodage utilisé dans le message. Les noms de parties, les types et la valeur de l'attribut « namespace » sont tous des entrées du c odage, bien que l'attribut « namespace » s'applique uniquement au contenu non e xplicitement défini par les types abstraits.

Si Use est literal, chaque partie fait alors référence à un schéma concret en utilisant un attribut element pour les parties simples ou un a ttribut type pour les parties composites.

- **soap:fault** : cet élément permet d'exprimer comment sera codé l'élément detail, sous-élément de soapenv:Fault du message d'erreur SOAP.

```
<definitions…>
        <binding…>
                <operation…>
                        <fault>*
                                <soap:fault name="nmtoken"  use="literal|encoded"
                                        encodingStyle="uri-list"? namespace="uri"?>
                        </fault>
                </operation>
        </binding>
</definitions>
```

L'attribut name relie le soap:fault au WSDL fault défini pour l'opération. Le message d'erreur doit avoir un seul Part. Les attributs use, encodingStyle et namespace sont tous utilisés de la même manière qu'avec soap:body.

- **soap:header :** L'élément soap:header permet de définir les headers qui seront transmis à l'intérieur de l'élément Header de l'Enveloppe SOAP.

```
<definitions…>
        <binding…>
                <operation…>
                        <input>
                                <soap:header element="qname"  fault="qname"?>*
                        </input>
                        <output>
                                <soap:header element="qname"  fault="qname"?>*
                         </output>
                </operation>
        </binding>
</definitions>
```

L'attribut element fait référence au schéma concret de l'élément header. Le schéma d'un header ne peut pas inclure de définitions pour les attributs soap:actor et soap:mustUnderstand.

Un attribut fault optionnel autorise la spécification du schéma concret du header en cas d'une erreur appartenant au header.

- **soap:address** : La liaison soap:address est utilisée pour donner une adresse (une URI) à un port. Un port utilisant la liaison SOAP doit spécifier exactement une adresse.

```
<definitions…>
        <port…>
            <binding…>
                <soap:address location="uri"/>
            </binding>
        </port>
</definitions>
```

3. L'annuaire des services (UDDI).

Une fois le service décrit, il faut mettre en œuvre des mécanismes à la disposition des utilisateurs, pour que ces derniers puissent utiliser ces services. Cette phase s'appelle la publication. La publication des services nécessite une sorte d'annuaire dans lequel on va pouvoir le référencer.

L'annuaire des services correspond au lieu où les services web sont enregistrés. L'architecture utilise le standard UDDI (en anglais, *Universal Description, Discovery and Integration*) qui représente le protocole pour l'enregistrement et la découverte des services web. L'annuaire des services inclut à la fois des annuaires publics, gérés par des opérateurs UDDI, et des annuaires privés à usage interne de l'entreprise ou à usage entre entreprises pour la mise en œuvre d'échanges B2B.

La première spécification d'UDDI date le 6 septembre 2000 [**1**]. C'est les sociétés Ariba, IBM, Microsoft qui sont plus précisément à l'origine de cette spécification.

b. Terminologie UDDI

Grâce à UDDI, les entreprises peuvent enregistrer des données les concernant, des renseignements sur les services qu'elles offrent et des informations techniques sur le mode d'accès à ces services. Ces données sont enregistrées dans un annuaire qui est le cœur de

standard UDDI. Cette approche par annuaire est intéressante lorsque les informations sont stockées dans des emplacements bien connus. Une fois que l'annuaire est localisé, il reste à l'interroger pour obtenir les informations souhaitées. L'emplacement de l'annuaire UDDI est généralement défini par l'administrateur du système.

Les fournisseurs de services Web présentent diverses options pour déployer les registres UDDI. Les scénarios de déploiement se répartissent en trois grandes catégories : déploiement public, inter-entreprise et intra-entreprise. Le déploiement public se fait par l'intermédiaire d'un annuaire public alors que les deux autres se font par l'intermédiaire des annuaires privés [**1**]:

- **Les annuaires publics** : ces sont des annuaires hébergés par des sociétés comme IBM ou Microsoft (UBR : UDDI Business Registry). UBR est un registre UDDI public répliqué dans de nombreuses entreprises. Il sert à la fois de ressources pour les services Web présents sur Internet et de test pour les développeurs de services Web. Ces annuaires sont moins développés que les annuaires privés parce qu'ils ne sont pas suffisamment sécurisés.
- **Les annuaires privés** : peuvent être hébergés par une société quelconque sur un réseau privé ou sur internet. un contrôle plus étroit sur les types d'informations enregistrées est alors possible. Ces registres privés peuvent être dédiés à une seule entreprise ou à des groupes de partenaires commerciaux.

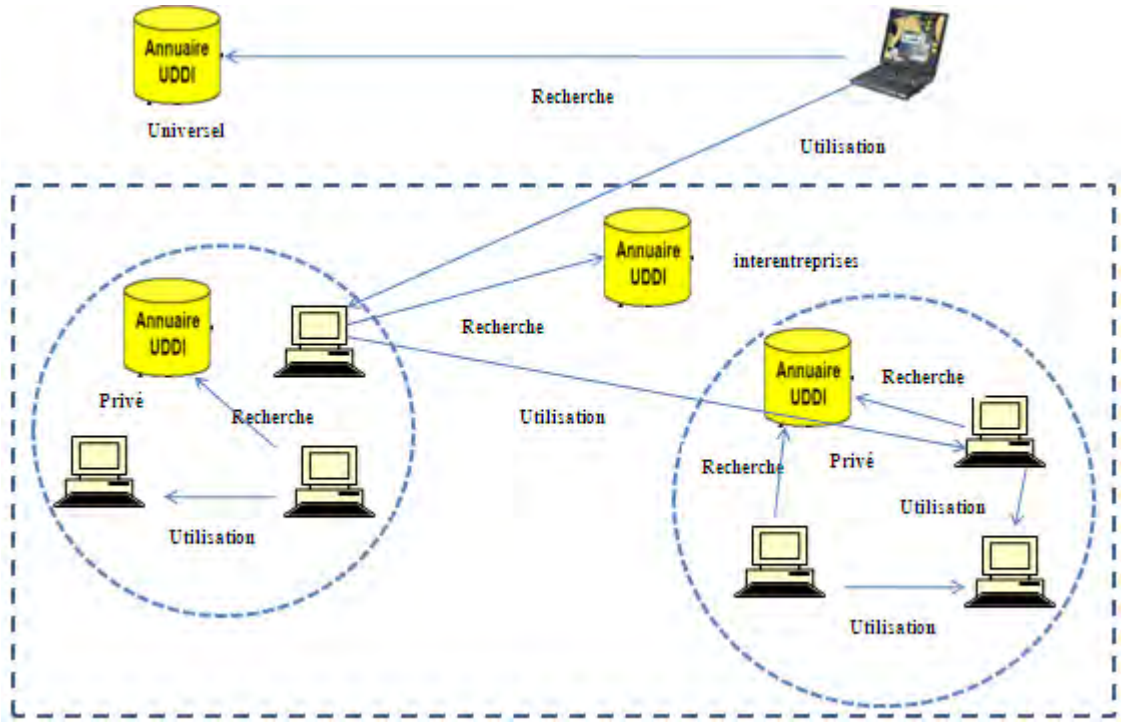


Figure 11 : les divers types de référentiels UDDI.

Les annuaires UDDI contiennent des informations détaillées concernant les services Web et leurs emplacements. Une entrée d'annuaire UDDI se compose de trois parties principales [8]:

- **Des pages blanches (BusinessEntity)** : annuaire classique par nom. Elles contiennent les informations les plus générales qui décrivent le fournisseur du service. Elles ne sont pas destinées à un logiciel mais à un développeur ou à un installateur qui peut avoir besoin de contacter directement quelqu'un responsable du service. Donc elles recensent les entreprises et contiennent des informations telles que le nom de l'entreprise, ses adresses, des descriptions accessibles aux clients, et d'autres détails administratifs.
- **Des pages jaunes (BusinessService)** : annuaire thématique. Dans ces pages La liste des services Web disponibles est stockée dans une entrée de type BusinessService. Les services peuvent être organisés en fonction de leur utilisation prévue : ils peuvent être groupés en domaines d'application, par régions géographiques, ou tout autre schéma approprié. Une information de service stockée dans un registre UDDI inclut simplement une description d'un service et un pointeur vers les implémentations de service Web qu'il contient. Il est aussi possible d'enregistrer des liens vers des services hébergés par d'autres fournisseurs. Ces liens se nomment « projections de service ».

- **des pages vertes** : annuaire technique. Il fournit des informations sur le mode d'exploitation des services en proposant des fichiers WSDL;

NB : UDDI ne contient que des références associées à des services, et non les services eux-mêmes.

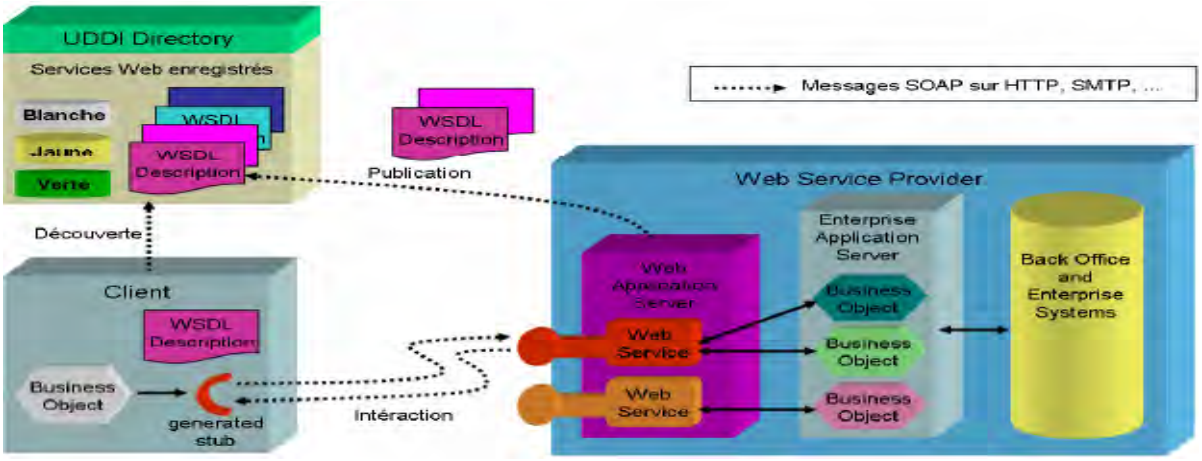


Figure 12 : Publication et découverte.

L'organisation structurelle d'un référentiel UDDI est essentiellement composée de quatre éléments :

- **BusinessEntity (entité commerciale)** : Décrites sous la forme d'un schéma XML. Elles contiennent les éléments relatifs à l'entreprise qui propose le service (nom, coordonnées, secteur d'activité, l'adresse du site web…). Une entité commerciale (BusinessEntity) est associé à un identifiant unique UUID (Universal Unique Identifier) BusinessKey. A ces information s'ajoutent plusieurs liens vers les autres constituants du référence UDDI (par exemple : le BusinessEntity peut comporter plusieurs offres de services).
- **BusinessService (les offres de services) :** Décrites aussi sous la forme d'un schéma XML. C'est un ensemble de services proposés répondant à un besoin métier spécifique. Donc elles Contiennent des informations relatives au métier de l'entreprise. Elles énumèrent plusieurs liaisons UDDI qui vont formaliser les points d'accès aux services offerts. Une entreprise peut avoir plusieurs métiers et donc plusieurs businessService. Le BusinessService est identifié par un nom et est associé à un UUID service Key.
- **bindingTemplate (Liaisons UDDI)** : cet élément doit décrire les informations techniques nécessaires à l'utilisation d'un service Web (décrire les informations

destinées à décrire les points d'accès aux services). Le bindingTemplate, également, est identifié par un UUDI bindingKey.

- **tModel** : La définition détaillée du protocole est fournie par une entité UDDI nommée (tModel). Dans de nombreux cas, le tModel référence un fichier WSDL qui décrit l'interface SOAP du service Web, mais les entités tModel sont suffisamment flexibles qu'elles peuvent servir à décrire pratiquement n'importe quel type de ressource.

E. Conclusion

Dans cette partie, nous avons répondu aux questions qui portent sur l'historique des services Web et le rapport entre le service Web et les normes utilisées pour concevoir les architectures distribués. Nous avons également étudié en détail les différents éléments indispensables au développement des services Web unitaires que sont les spécifications SOAP, WSDL, UDDI.

Dans la partie suivante, nous allons aborder en détail les différentes techniques et technologies utilisées pour faire la composition des services Web unitaire.

Partie III : Les Services Web Composites

III. Les Services Web Composites

Après avoir définit les éléments indispensables au développement des services Web unitaires que sont les spécifications SOAP, WSDL, UDDI, il devient impératif de formaliser la façon dont ces services Web peuvent être combinés entre eux : le développement d'applications à base de services web qui s'appuie sur la composition de ces derniers. Cela a relancé les travaux de recherche sur les problèmes d'interopérabilité et de coordination de services web. La solution proposée par la plupart de ces travaux consiste à modéliser le logiciel à réaliser par un procédé (un processus métier). Ce procédé gère la coordination des tâches et les opérations fournies par ce logiciel. Dans cette section nous allons présenter une étude sur les services web composites, orchestration et chorégraphie et les langages pour représenter ces concepts.

A. Description et fonctionnement

Un service web est dit composé ou composite lorsque son exécution implique des interactions avec d'autres services web afin de faire appel à leurs fonctionnalités. La composition de services web spécifie quels services ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'exception [**13**].

La composition des Services Web en processus métier peut se faire de deux manières: orchestration et chorégraphie :

1. Chorégraphie

La *chorégraphie* trace la séquence de messages pouvant impliquer plusieurs parties et plusieurs sources, incluant les clients, les fournisseurs, et les partenaires. La chorégraphie est typiquement associée à l'échange de messages publics entre les services web, plutôt qu'a un procédé métier spécifique exécuté par un seul partenaire [**14**].

Une *chorégraphie* décrit, d'une part un ensemble d'interactions qui peuvent ou doivent avoir lieu entre un ensemble de services (représentés de façon abstraite par des rôles), et d'autre part les dépendances entre ces interactions.

2. Orchestration

L'*orchestration* définit les interactions entre les applications qui participent au processus métier et l'enchainement de ces interactions. Les interactions sont décrites en termes de

messages (envoyés et reçus) et de traitement métier associées à l'émission (préparation) ou à la réception de ces messages [**4**].

L'*orchestration* de services permet de définir l'enchaînement des services selon un canevas prédéfini, et de les exécuter à travers des "scripts d'orchestration". Ces scripts sont souvent représentés par des procédés métier ou des workflows inter/intra-entreprise. Ils décrivent les interactions entre applications en identifiant les messages, et en branchant la logique et les séquences d'invocation [**14**].

Il y a une différence importante entre l'orchestration et la chorégraphie de services web :

- L'orchestration se base sur un procédé métier exécutable pouvant interagir avec les services web internes ou externes. L'orchestration offre une vision centralisée, le procédé est toujours contrôlé du point de vue d'un des partenaires métier.

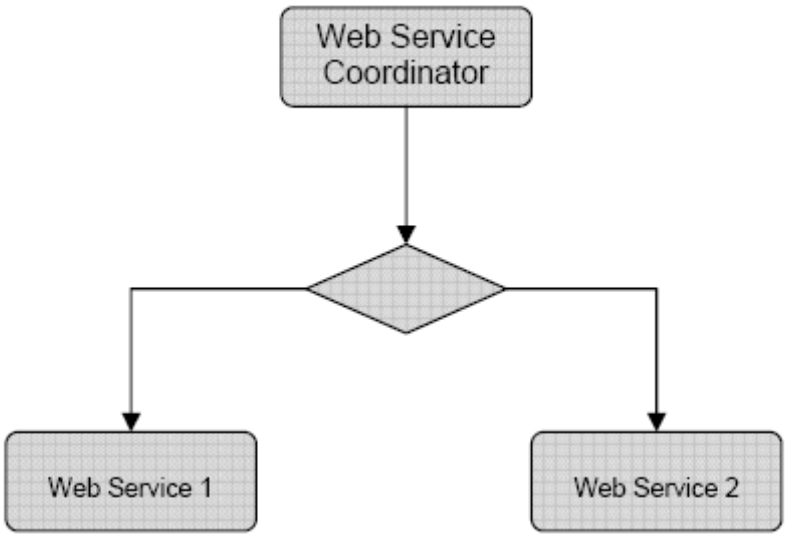


Figure 13 : Web Services Orchestration

- Contrairement à l'orchestration, la chorégraphie n'a pas un coordinateur central. Chaque service web mêlé dans la chorégraphie connaît exactement quand ses opérations doivent être exécutées et avec qui l'interaction doit avoir lieu (La chorégraphie est de nature plus collaborative, chaque participant impliqué dans le procédé décrit le rôle qu'il joue dans l'interaction).

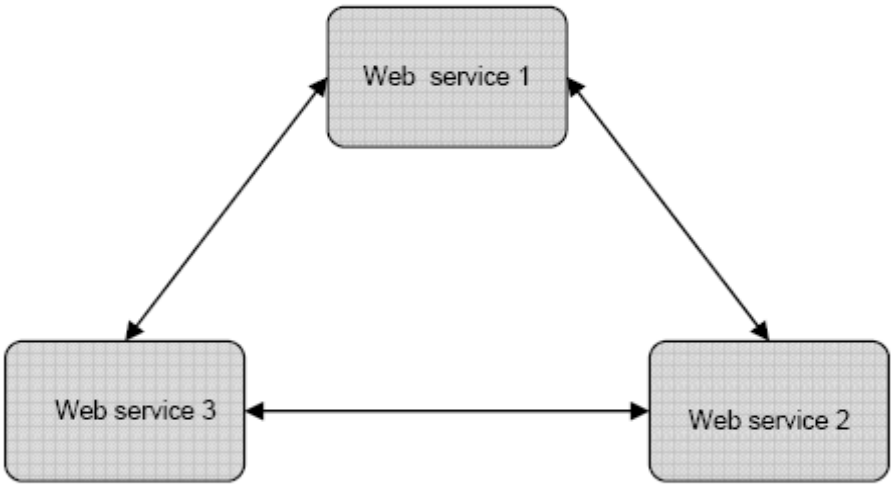


Figure 14 : Web Services chorégraphie

c. Langages d'orchestration et de chorégraphie de services web

Ils existent plusieurs langages de définition pour la composition de services web. Dans ce qui suit nous allons présenter brièvement quelques uns d'entre eux :

BPML

La spécification BPML (*Business Process Modeling Language*) est un métalangage de modélisation des processus métier dont le but est de fournir un modèle d'exécution pour les processus métier collaboratifs et transactionnels.

Le langage BPML a ét é développé par l'organisation BPMI.org *(Business Process Management Initiative)* dont l'objectif est de standardiser la gestion des processus métier qui couvrent de multiples applications et partenaires métier en intranet et sur Internet. Cette organisation est une organisation indépendante comprenant Intalio, Sterling Commerce, Sun, CSC, et d'autres.

Schématiquement, BPML divise un processus métier en trois parties [**4**]:

- Une interface publique commune aux partenaires de description du processus. Cette interface peut être prise en charge par des protocoles tels que WSDL, ebXML ou autres ;
- Deux implémentations privées, propres à chacun des deux partenaires concernés.

WSCI

WSCI *(Web Services Choreography Interface)* est une spécification proposée par BEA, Intalio, SAP et Sun Microsystems pour prendre en compte la collaboration d'application à application. La spécification WSCI a été soumise au W3C sous forme d'une note datée du 8 août 2002.

La spécification WSCI définit un langage de description des échanges collaboratifs entre services Web. Cette spécification concerne essentiellement les aspects liés à la chorégraphie des échanges entre les services web. Ainsi, les interfaces statiques des services web sont décrites en WSDL, et la chorégraphie entre eux est décrite en WSCI. Elle ne traite pas les questions d'orchestration des processus métier. Elle ne décrit que le comportement du service que l'on peut observer de l'extérieur, ainsi que les règles d'interaction entre le service et son environnement.

WSCI s'ajoute à WSDL et SOAP (standards sur lequel les services web sont construits), pour donner une description des types des messages, de leur ordre ainsi que de quelques autres attributs afin de permettre aux applications d'agir en collaboration avec succès [**14**].

WSCL

La spécification WSCL *(Web Service Conversation Language)* a été proposée en mai 2001 par une équipe de Hewlett-Packard. Elle a été adressée au W3C le 4 février 2002, et enregistrée sous forme d'une note datée du 14 mars 2002.

WSCL propose de décrire à l'aide de documents XML les services Web en mettant l'accent sur les *conversations* de ceux-ci. En outre, les messages à échanger sont pris en compte. WSCL a été pensé pour s'employer conjointement avec WSDL.

Les définitions WSDL peuvent être manipulées par WSCL pour décrire les opérations possibles ainsi que leur chorégraphie. En retour, WSDL fournit les concrétisations vers des définitions de messages et des détails techniques pour les éléments manipulés par WSCL[**11**].

BPEL (Process Execution Language for Web Services)

BPEL4WS (BPEL for Web Service), ou simplement BPEL définit la coordination de services web pour atteindre un objectif métier. Un processus BPEL est un document XML généré en utilisant un outil de conception graphique (comme par exemple ActiveBPEL® Designer).

Les processus BPEL sont exécutés par des moteurs d'exécutions. Plusieurs implémentations industrielles ont été réalisées pour offrir un moteur d'exécution de procédés décrits en BPEL, parmi ces implémentations Oracle BPEL et BSOA Orchestra et ActiveBPEL.

Pour développer un processus d'agrégation complet avec le langage BPEL, trois étapes sont nécessaires :

- ❖ Le premier consiste à formaliser les besoins d'agrégation dans le langage BPEL sous forme d'un fichier XML d'extension «.bpel » conforme à la structure BPEL. Ceci pourra être réalisé avec un outil de conception graphique tel que «ActiveBPEL® Designer».
- ❖ Ensuite, il faut déployer ce fichier dans un moteur d'exécution.
- ❖ Une fois exécuté, le processus a besoin d'un outil pour visualiser son avancement. Par exemple, ActiveBPEL propose une console qui permet de déclencher et de gérer une ou plusieurs instances d'un processus BPEL et d'assurer son suivi lors de l'exécution.

BPEL emploie WSDL pour modéliser un processus et ses partenaires en tant qu'interfaces abstraites WSDL. De même, l'abstraction des messages définie par WSDL se voit employée dans BPEL. Les activités de processus peuvent être définies à l'aide d'´éléments orientés flots (itérations, répétitions, séquences).

Structure d'un processus BPEL [9]

```
<process name="NCName» targetNamespace="anyURI"
   queryLanguage="anyURI"?
   expressionLanguage="anyURI"?
   suppressJoinFailure="yes|no"?
   exitOnStandardFault="yes|no"?
   xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

   <extensions>?
      <extension namespace="anyURI" mustUnderstand="yes|no" />+
   </extensions>

   <import namespace="anyURI"?
      location="anyURI"?
      importType="anyURI" />*


   <partnerLinks>?
      <!-- Note: At least one role must be specified. -->
      <partnerLink name="NCName"
         partnerLinkType="QName"
         myRole="NCName"?
         partnerRole="NCName"?
         initializePartnerRole="yes|no"?>+
      </partnerLink>
   </partnerLinks>

   <messageExchanges>?
      <messageExchange name="NCName" />+
   </messageExchanges>
```

```
<variables>?
   <variable name="BPELVariableName"
      messageType="QName"?
      type="QName"?
      element="QName"?>+
      from-spec?
   </variable>
</variables>


<correlationSets>?
   <correlationSet name="NCName" properties="QName-list" />+
</correlationSets>

<faultHandlers>?
   <!-- Note: There must be at least one faultHandler -->
   <catch faultName="QName"?
      faultVariable="BPELVariableName"?
      ( faultMessageType="QName" | faultElement="QName" )? >*
      activity
   </catch>
   <catchAll>?
      activity
   </catchAll>
</faultHandlers>
   <eventHandlers>?
      <!-- Note: There must be at least one onEvent or onAlarm. --
      >
      <onEvent partnerLink="NCName"
         portType="QName"?
         operation="NCName"
         ( messageType="QName" | element="QName" )?
         variable="BPELVariableName"?
         messageExchange="NCName"?>*
         <correlations>?
            <correlation set="NCName" initiate="yes|join|no"? />+
         </correlations>
         <fromParts>?
            <fromPart part="NCName" toVariable="BPELVariableName"
            />+
         </fromParts>
         <scope ...>...</scope>
      </onEvent>
      <onAlarm>*
         <!-- Note: There must be at least one expression. -->
         (
         <for expressionLanguage="anyURI"?>duration-expr</for>
         |
         <until expressionLanguage="anyURI"?>deadline-expr</until>
         )?
         <repeatEvery expressionLanguage="anyURI"?>
            duration-expr
         </repeatEvery>?
         <scope ...>...</scope>
      </onAlarm>
   </eventHandlers>
   activity
</process>
```

Figure 15 : Structure d'un processus BPEL

L'attribut ***queryLanguage*** définit le langage de requête XML utilisé pour la sélection des nœuds lors de diverses opérations (par défaut Xpath 1.0: dans BPEL 2.0 c'est : *urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0*).

L'attribut ***expressionLanguage*** définit le langage de requête XML utilisé pour exprimer des expressions utilisées dans le processus (par défaut Xpath 1.0).

L'attribut ***suppressJoinFailure*** : est utilisé dans les mécanismes de liens pour préciser s'il est possible de supprimer l'instruction en cas d'inter blocage d'un processus métier exécutant du code en parallèle.

L'attribut ***exitOnStandardFault*** : Si cette propriété est placée à « yes » alors le processus sort immédiatement comme si une activité de sortie « exit » a été atteinte, quand une erreur standard de WS-BPEL autre que le bpel:joinFailure est produite.

BPEL Namespace : Indique le namespace BPEL pour chaque type de deux processus BPEL (exécutable ou abstrait). Par défaut cet espace de noms est pour spécifier les processus exécutables (http://docs.oasis-open.org/wsbpel/2.0/process/executable).

PartnerLinks : un processus BPEL décrit l'enchaînement des interactions entre le processus et les services Web utilisés. Chaque interaction décrit quel est le rôle que le processus et les services Web jouent, et quel sont les données qui peuvent être manipulées par les différentes parties dans ce r ôle. La collection des partenaires (***partners***) énumère les tiers qui interviennent durant le déroulement du pr ocessus. Ceux-ci sont modélisés sous forme de services Web. Chaque lien de partenaire est typé par un ***partnerLinkType***, qui est chargé de définir le rôle que joue chacun des deux partenaires dans une conversation (Voir la figure suivante) :

Le processus BPEL a un état, cet état est maintenu par des *"variables"* contenant des données. Ces données sont combinées afin de contrôler le comportement du pr océdé, pour cela nous utilisons les *"expressions"*. Les expressions permettent d'ajouter des conditions de transition ou de jointure au flot de contrôle.

Il ya quatre façons d'utiliser des variables :

- La variable d'entrée recevant le message envoyé par un partenaire ;
- La variable de sortie contenant le message envoyé à un partenaire ;

- La variable d'erreur retournée par une activité gérant les erreurs ;
- La variable temporaire utilisée pour des traitements intraprocessus.

CorrelationSets : la collection des ensembles de corrélation est utilisée pour permettre les interactions asynchrones.

BPEL définit également les gestions de :

- fautes (***FaultHandlers) :*** la collection des gestionnaires d'exceptions fournit les activités à exécuter en cas d'erreurs dans le fonctionnement du processus.
- événements (***eventHandlers*** : messages, alarmes) ;
- Les compensations (il s'agit en fait d'une liaison vers une activité qui sera dite de compensation : c'est-à-dire lorsqu'il est nécessaire d'annuler une action déjà exécutée et terminée) ;

Les activités représentent les actions exécutées dans le cadre du processus métier modélisé. Ces activités peuvent être *basiques* ou *structurées*.

- Les activités basiques sont :
  - *"invoke"* pour invoquer une opération dans un service web ;
  - *"receive"* Mise en attente d'un message d'un partenaire;
  - *"reply"* Envoi d'un message de réponse à un message reçu (par un receive) de la part d'un partenaire ;
  - *"wait"* pour attendre un certain temps ;
  - *"assign"* Affectation de nouvelles valeurs aux données d'un conteneur (par copy) ;
  - *"throw"* pour lancer une erreur d'exécution ;
  - *"exit"* terminaison du processus;
  - et *"empty"* qui ne fait rien (utile pour la synchronisation des activités concurrentes).
- Les activités structurées sont composées d'autres activités basiques et structurées. Les types d'activités structurées sont :
  - *"sequence"* pour définir un or dre d'exécution (Bloc d'activités à exécuter en séquence);
  - *"switch"* pour l'acheminement conditionnel (sélection d'une activité à exécuter parmi ensemble);
  - *"while"* pour les boucles ;

- *"pick"* attente bloquante d'un message d' un partenaire ou de l'expiration d'un délai ;
- *"flow"* pour l'acheminement parallèle ;
- *"scope"* pour regrouper les activités afin qu'elles soient traitées par le même gestionnaire d'erreur ;
- et *"compensate"* pour invoquer les activités de compensation par le gestionnaire d'erreur, pour défaire l'exécution déjà complétée d'un regroupement d'activité.

Pour illustrer toutes ces explications, voyons comment s'applique BPEL4WS dans un exemple simple mais bien correct : prenons l'exemple d'un Web Service de réservation d'une agence de voyages. Le client invoque le service de réservation en fournissant un message de demande qui contient ses coordonnées (personnels et bancaires) et le séjour choisi. En retour, il obtient un numéro de réservation et recevra sa facture par courrier dans les jours suivants.

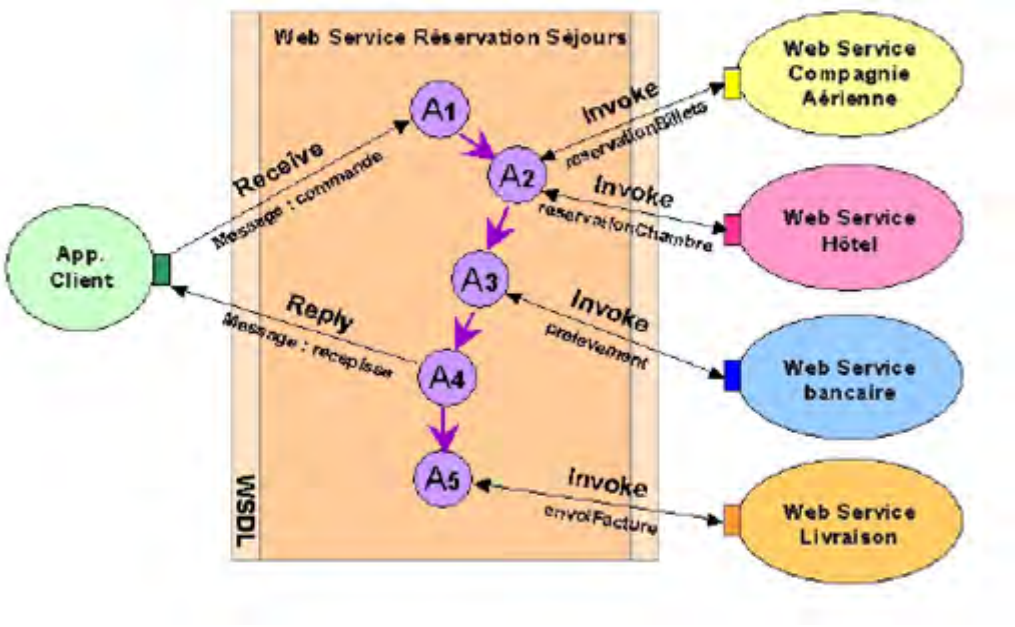


Figure 16 : Web Service de réservation d'une agence de voyages.

Ce Web Service de réservation peut être modélisé sous la forme d'un processus métier en combinant différents Web Services élémentaires. Ainsi, à la réception de la demande de réservation, le processus d'orchestration invoque en parallèle le Web Service d'une compagnie aérienne et celui d'une chaîne d'hôtels pour effectuer les réservations. Ensuite, il effectue le prélèvement bancaire auprès du Web Service d'une banque et donne au client son numéro de réservation. En fin, il invoque un Web Service de livraison pour l'envoi de la facture papier au domicile du client.

Pour les aspects d'illustration nous allons limier à une partie du processus métier qui permet à un client de réserver dans un hôtel. Le comportement de ce processus sera le suivant : obtenir un message, invoquer le service Web (Banque pour vérifier la carte Bancaire), puis invoquer le service Web Hôtel pour faire la réservation, et finalement envoyer la réponse à l'application cliente. Pour ces actions, nous allons donc respectivement nous servir des activités <receive>, <Assign>, <invoke> et <reply>.

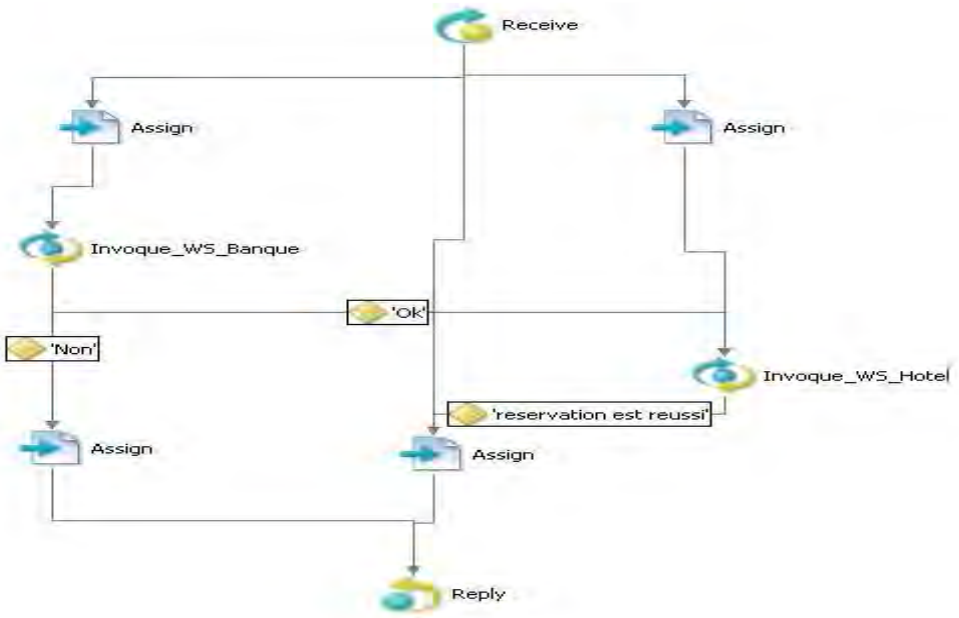


Figure 17 : Processus métier modélisant le Service Web composite (Agence de voyage)

Les fichiers WSDL contiennent une caractéristique importante pour l'expression du procédé dans le langage BPEL, cette caractéristique est la définition des partnerLinkTypes. Un partnerLinkType fournit une relation de conversation entre deux services à partir de la définition des rôles et de la spécification du portType. Cette information sera utilisée pour c onstruire la section <partnerLink> du fichier BPEL.

Le fichier WSDL en générale ne contient pas la déclaration des partnerLinkTypes. Il ya deux possibilités pour déclarer les partnerLinkTypes : soit en créant un nouveau fichier ou e n insérant les lignes des déclarations dans le même fichier WSDL. Pour la dernière approche on doit respecter la structure suivante :

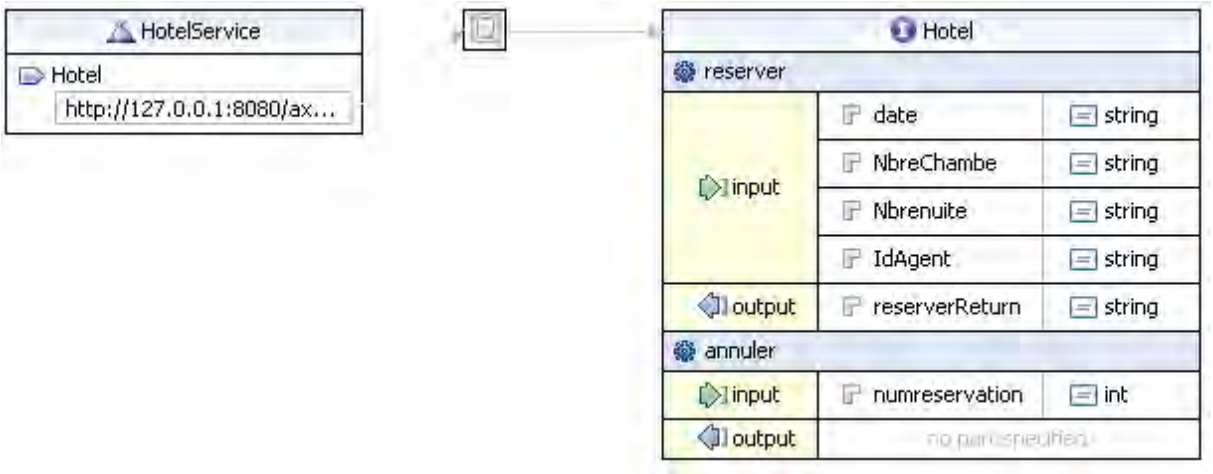

```
<wsdl:definitions xmlns:impl="1  " … >
      <plnk:partnerLinkType xmlns:plnk="http://docs.oasis-
open.org/wsbpel/2.0/plnktype"  name="2">
   <plnk:role name="3" portType="impl:4"/>
</plnk:partnerLinkType>
……..

</wsdl:definitions>
```

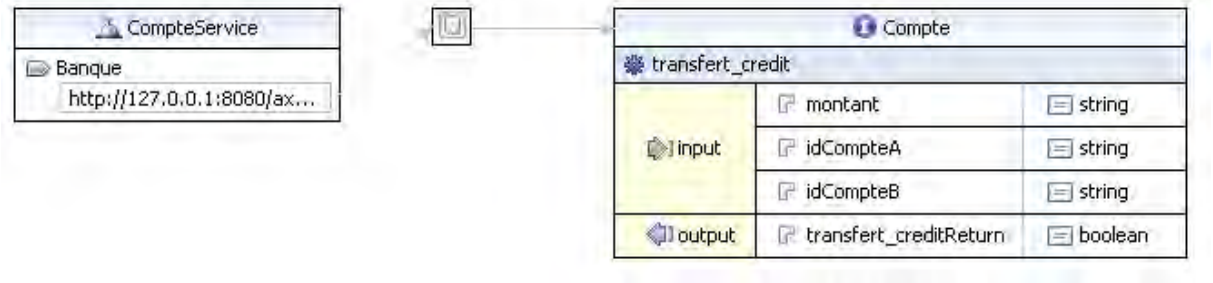
Les valeurs identifiées pour (1) et (4), sont recherchées dans les fichiers WSDL de la manière suivante :

- La valeur de l'attribut xmlns:impl est générée automatiquement par le serveur de déploiement de service Web et vient avec le contrat WSDL.
- La valeur de l'attribut portType de l'élément role (4) à partir de la balise <*portType*> (valeur introduite dans le fichier au numéro 4) ;
- Les valeurs 2 et 3 respectivement peuvent être obtenu en ajoutant PLT et Role à la valeur 4 ;

Dans cet exemple nous avons développé deux services Web. Chacune de ces services contient une opération.

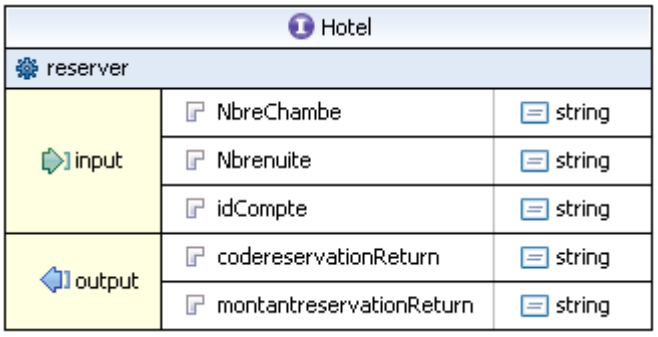



```
<plnk:partnerLinkType xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype" name=
"HotelPLT">
   <plnk:role name="Hotel" portType="impl:Hotel"/>
</plnk:partnerLinkType>
```




```
<plnk:partnerLinkType xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype" name=
"BanquePLT">
   <plnk:role name="Banque" portType="impl:Compte"/>
</plnk:partnerLinkType>
```

Dans cet exemple, un client envoie une requête qui contient trois entrées. Ces entrées seront modélisées sous forme des messages dans un fichier WSDL séparé.




```
<plnk:partnerLinkType xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype" name=
"AgencePLT">
   <plnk:role name="Agence" portType="impl:Hotel"/>
</plnk:partnerLinkType>
```

Sections générales du fichier BPEL

Un processus BPEL peut être divisé en quatre sections : *attributs supérieurs*, la section *partnerLinks*, la section *variables* et la section d'*orchestration*.

- Pour les ***attributs supérieurs*** : c'est sont les attributs queryLanguage, expressionLanguage,….etc. Pour savoir plus vous pouvez voir la section « Structure d'un processus BPEL » ;

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasisopen.org/wsbpel/2.0/process/executable"
              xmlns:ns1="http://127.0.0.1:8080/activebpel/services/AgenceVoyagePLTService"
              xmlns:ns2="http://127.0.0.1:8080/axis/services/Hotel"
              xmlns:ns3="http://127.0.0.1:8080/axis/services/Banque"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="AgenceVoyage"
              suppressJoinFailure="yes" targetNamespace="http://AgenceVoyage">
```

- **Les *partnerLinks*** : sont les partenaires qui interviennent durant le déroulement du processus. Ceux-ci sont modélisés sous forme de services Web. Chaque lien est typé par partnerLinkType qui est chargé de définir le rôle que joue chacun des deux partenaires dans une conversation.

```
<bpel:partnerLinks>
        <bpel:partnerLink myRole="Agence" name="AgencePLT" partnerLinkType="ns1:AgencePLT"/>

        <bpel:partnerLink name="HotelPLT" partnerLinkType="ns2:HotelPLT" partnerRole="Hotel"/>

        <bpel:partnerLink name="BanquePLT" partnerLinkType="ns3:BanquePLT" partnerRole="Banque"/>
</bpel:partnerLinks>
```

- ***Les variables*** : les variables permettent à un processus BPEL de garder l'état entre des interactions.

```
<bpel:variables>
        <bpel:variable messageType="ns1:clientRequest" name="clientRequest"/>
        <bpel:variable messageType="ns1:clientResponse" name="clientResponse"/>
        <bpel:variable messageType="ns2:reserverRequest" name="reserverRequest"/>
        <bpel:variable messageType="ns2:reserverResponse" name="reserverResponse"/>
        <bpel:variable messageType="ns3:transfert_creditRequest" name="transfert_creditRequest"/>
        <bpel:variable messageType="ns3:transfert_creditResponse" name="transfert_creditResponse"/>
</bpel:variables>
```

- **Orchestration** : Décrit le comportement du processus métier lui-même en utilisant différents opérateurs dit de programmation.(voir partie Annexe) . pour un e xemple d'activité :

```
<bpel:receive createInstance="yes" operation="reserver" partnerLink="AgencePLT" variable="clientRequest">
    <bpel:sources>
        <bpel:source linkName="L1"/>
        <bpel:source linkName="L2"/>
        <bpel:source linkName="L6"/>
    </bpel:sources>
</bpel:receive>
<bpel:reply operation="reserver" partnerLink="AgencePLT" variable="clientResponse">
    <bpel:targets>
        <bpel:target linkName="L8"/>
        <bpel:target linkName="L10"/>
        </bpel:targets>
</bpel:reply>
<bpel:invoke inputVariable="reserverRequest" name="Invoque_WS_Hotel" operation="reserver"
             outputVariable="reserverResponse" partnerLink="HotelPLT">
    <bpel:targets>
        <bpel:joinCondition>$L5 and $L4</bpel:joinCondition>
        <bpel:target linkName="L4"/>
        <bpel:target linkName="L5"/>
        </bpel:targets>
        <bpel:sources>
            <bpel:source linkName="L7">
                <bpel:transitionCondition>$reserverResponse.reserverReturn
                                !='imposible de faire la reservation'</bpel:transitionCondition>
            </bpel:source>
        </bpel:sources>
</bpel:invoke>
```

B. Conclusion

Dans cette partie, nous avons étudié les différentes approches utilisées pour faire la composition des services Web (orchestration et chorégraphie). Nous avons également étudiés les différents langages utilisés pour représenter ces approches.

La combinaison de cette partie avec les deux parties précédentes, donne une vision approfondie de différentes techniques utilisées actuellement, dans le domaine de l'informatique distribué et en particulier le domaine de service Web. Dans la partie suivant, nous commençons à attaquer le deuxième objectif qui porte sur les aspects qualité de service et en particulier l'étude de performance.

Partie IV : Réseaux d'automates stochastiques

IV. Réseaux d'automates stochastiques

A. Introduction

L'évolution rapide des systèmes informatiques et des réseaux entraine actuellement des problèmes critiques de performances, par exemple la qualité de service. La qualité de service représente un ensemble de propriétés opérationnelles du service que l'on doit constater dans la réalisation de la prestation (ces propriétés représentent un ensemble d'exigences concernant la mise en œuvre du service). Pour garantir ces propriétés, une analyse fine du comportement du système est nécessaire pour identifier les problèmes et les résoudre. Il est donc fondamental de disposer de méthodes et d'outils permettant d'analyser et de comprendre le comportement de services, afin de répondre à des questions de performance et du coût.

Cette section présente la partie qui concerne la modélisation des services web composites. Et plus précisément l'utilisation des Réseaux d'Automates Stochastiques pour modéliser le comportement d'un processus métier.

Mais avant de passer aux aspects liés à ce formalisme on va faire une petite introduction sur l'historique des méthodes d'évaluation de performance utilisées pour modéliser les comportements des applications informatiques en générale.

B. Méthodes d'évaluation de performances

Le modèle est une représentation de la réalité dans un formalisme. Il est développé pour répondre à des questions déterminées et comporte certaines limitations, c'est une abstraction du système réel. Ce système réel n'existe pas forcément lors du processus de modélisation, il se peut que ce soit un système que nous cherchons à développer. Pour cela nous désirons effectuer une étude préliminaire de ses performances.

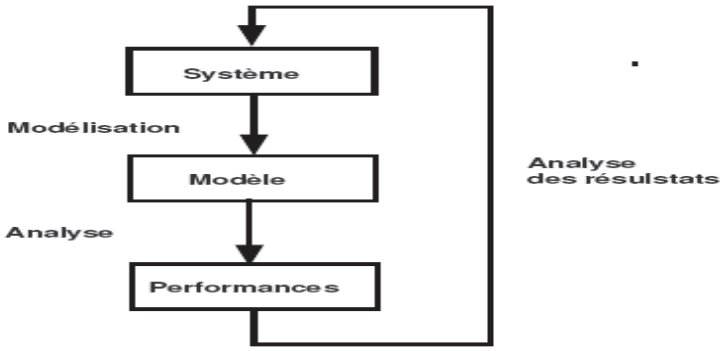


Figure 18 : Processus de Modélisation.

La modélisation permet de suivre une ou plusieurs trajectoires (c.à.d. couple (espace, temps)) de manière à déterminer l'évolution des systèmes complexes. Ce qui permet de déduire les propriétés suivantes :

- Des mesures de performances : début, temps de réponse, taux de perte,…. ;
- Fiabilités : temps moyen entre deux pannes ;
- Spécification : un état est il atteignable ?
- ...etc.

Lors de la modélisation d'un tel système, la principale difficulté consiste à modéliser les délais, à savoir le temps que l'on passe dans chaque état en fonction des statistiques sur l'environnement.

Dans la plupart des cas, nous supposons que le système est markovien, donc sans mémoire. L'état futur ne dépend alors que de l'état présent, et non du passé. Cependant, lorsque le nombre d'états est important (de l'ordre du million d'états), il est quasiment impossible d'envisager une modélisation à plat, à la fois pour la phase d'abstraction du système, mais aussi en prévision des techniques d'analyse utilisées pour calculer des indices de performances sur le système. Raison pour laquelle plusieurs formalismes à haut niveau ont été proposés pour faire la modélisation de chaînes de Markov très grandes et très complexes de façon compacte et structurée. Ces formalises ont été implémentés dans des logiciels pour permettre de générer l'espace d'états et le générateur infinitésimal de la chaîne de Markov, ainsi que pour calculer les solutions stationnaires et transitives. Nous pouvons citer ici par exemples :

- **Les réseaux de files d'attente** : c'est une approche orientée « ressources consommées par des clients ». ce formalisme a plusieurs extensions comme par exemple les réseaux

de files d'attente hiérarchiques, qui utilisent l'algèbre de Kronecker pour structurer le modèle [**REF** 18] ;

- **Les réseaux de Petri** : le problème de la modélisation avec les réseaux de files d'attente est qu'elle manquait de précision dans la description des comportements complexes et en particulier le problème d'introduction de la synchronisation. Donc une approche basée sur le formalisme réseaux de Petri a été adoptée. Ce formalisme permet une analyse fine des synchronisations. Pour les extensions de ce niveau formalisme les premières approches ont consisté à introduire des temporisations (réseaux de Petri temporisés). Les réseaux de Petri stochastiques ont ensuite été définis pour introduire le comportement probabiliste des systèmes. Au niveau de la structuration des modèles, les réseaux de Petri stochastiques bien formé, les réseaux de Petri stochastiques généralisés, et les réseaux de Petri stochastiques généralisés coloré ont été introduits [REF 21].

Il existe d'autres approches qui n'ont pas la notion d'entités et de flot (respectivement clients et routage dans les réseaux de files d'attente ou jetons et arcs dans les réseaux de Petri). En revanche, elles offrent une vision compositionnelle de sous-systèmes qui interagissent entre eux (concept de modèles modulaires) :

- **Les algèbres de processus** : composition concurrente, exécution parallèle ;
- **Les réseaux d'automates** : intégration des synchronisations au modèle état-transition.

Nous nous intéressons aux Réseaux d'Automates Stochastiques (**RAS**). Comme il a été dit précédemment, ce formalisme fait parti des outils de modélisation qui reposent sur l'utilisation de chaînes de Markov. La résolution consiste en générale, dans un premier temps, à déterminer la chaîne associée ainsi que son générateur Q puis à calculer la distribution stationnaire de la chaîne en résolvant le système linéaire suivant :

$$\begin{cases} \pi Q = 0 \\ \pi e = 1 \end{cases} \quad \text{en temps continu.} \qquad \textbf{(1)}$$

Où Q est le générateur (matrice carrée de taille n), π est le vecteur de la distribution stationnaire de taille n, et e est le vecteur colonne de normalisation de taille n (e est un vecteur dont tous les éléments sont égaux à 1).

Dans ce q ui suit nous allons présenter en détail le formalisme des réseaux d'automates stochastiques.

C. Réseaux d'automates stochastiques

Les réseaux d'automates stochastiques (RAS) sont basés sur l'idée que le système modélisé peut se décomposer sous la forme de plusieurs sous-systèmes. Chacun de ces sous-systèmes peut évoluer indépendamment et peut interagir avec les autres en se synchronisant ou en faisant varier la valeur de ses transitions en fonction des états des autres sous-systèmes, ce comportement étant typiquement celui des services Web composites. C'est du fait que les Web services fonctionnent de manière modulaire et non pas intégrée (c.-à-d. qu'au lieu d'intégrer dans une seule application globale toutes les fonctionnalités, on crée (ou on récupère) plusieurs applications spécifiques qu'on fait inter-coopérer entre elles sous forme d'un processus métier.

1. Terminologie

Les RAS font partie des systèmes de transitions étiquetées (Labeled Transition System – LTS) qui sont des modèles permettant de représenter l'évolution d'un système par un graphe incluant des états reliés par des transitions étiquetées par des actions. Les LTS sont basés sur les notions suivantes :

- **Les états :** Un état représente un état du système prenant en compte les évènements de ce système que l'on souhaite modéliser, c'est-à-dire une étape (visible ou non) à instant donné du système réel.
- **Les transitions** : Les transitions relient deux états et indiquent les actions qui provoquent les changements d'états du LTS (et donc, par extension, du système). Ainsi, à un instant donné, l'évolution du système se traduit par le passage d'un état à un autre en franchissant une des transitions reliant ces deux états.

Avant de donner la liste des concepts utilisés dans les réseaux d'automates stochastiques nous commencerons par un petit rappel qui donne quelques définitions d'algèbre tensorielle, employées pour exprimer le générateur de la chaîne de Markov associée à ce formalisme. Et en particulier les trois propriétés suivantes :

- **les produits tensoriels** : Le produit tensoriel de deux matrices A et B, de dimensions respectivement (n_1, m_1) et (n_2, m_2), est une matrice de dimensions $(n_1 \times n_2, m_1 \times m_2)$.

On note par $\otimes$ le produit tensoriel, $\times$/+ le produit/somme matriciel traditionnel et I_n la matrice identité de dimension **n** :

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \qquad B = \begin{pmatrix} b_{1,1} & \cdots & b_{1,4} \\ \vdots & \ddots & \vdots \\ b_{3,1} & \cdots & b_{3,4} \end{pmatrix}$$

Le produit tensoriel est défini par $C = A \otimes B$ est égal à :

$$C = \begin{pmatrix} a_{1,1}B & a_{1,2}B \\ a_{2,1}B & a_{2,2}B \end{pmatrix}$$

$$C = \left(\begin{array}{cccc|cccc} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,1}b_{1,3} & a_{1,1}b_{1,4} & a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,1}b_{1,3} & a_{2,1}b_{1,4} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,1}b_{2,3} & a_{1,1}b_{2,4} & a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,1}b_{2,3} & a_{2,1}b_{2,4} \\ a_{1,1}b_{3,1} & a_{1,1}b_{3,2} & a_{1,1}b_{3,3} & a_{1,1}b_{3,4} & a_{2,1}b_{3,1} & a_{2,1}b_{3,2} & a_{2,1}b_{3,3} & a_{2,1}b_{3,4} \\ \hline a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,1}b_{1,3} & a_{2,1}b_{1,4} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} & a_{2,2}b_{1,3} & a_{2,2}b_{1,4} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,1}b_{2,3} & a_{2,1}b_{2,4} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} & a_{2,2}b_{2,3} & a_{2,2}b_{2,4} \\ a_{2,1}b_{3,1} & a_{2,1}b_{3,2} & a_{2,1}b_{3,3} & a_{2,1}b_{3,4} & a_{2,2}b_{3,1} & a_{2,2}b_{3,2} & a_{2,2}b_{3,3} & a_{2,2}b_{3,4} \end{array}\right)$$

Donc :

$$C_{[i,k],[j,l]} = a_{i,j}\, b_{k,l} \; avec \; i \in [1 \ldots . n_1] \, , j \in [1 \ldots . m_1] \; , k \in [1 \ldots . n_2] \; et \; l \in [1 \ldots . m_2] \qquad \textbf{(2)}$$

- **Facteurs normaux** : Un cas spécifique de produit tensoriel est le produit tensoriel d'une matrice carrée par une matrice identité.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \qquad I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Le facteur normal $A \otimes I_3$ est égal à :

$$\left(\begin{array}{ccc|ccc} a_{1,1} & 0 & 0 & a_{1,2} & 0 & 0 \\ 0 & a_{1,1} & 0 & 0 & a_{1,2} & 0 \\ 0 & 0 & a_{1,1} & 0 & 0 & a_{1,2} \\ \hline a_{2,1} & 0 & 0 & a_{2,2} & 0 & 0 \\ 0 & a_{2,1} & 0 & 0 & a_{2,2} & 0 \\ 0 & 0 & a_{2,1} & 0 & 0 & a_{2,2} \end{array}\right)$$

Le facteur normal $I_3 \otimes A$ est égal à :

$$\left(\begin{array}{cc|cc|cc} a_{1,1} & a_{1,2} & 0 & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & a_{1,1} & a_{1,2} & 0 & 0 \\ 0 & 0 & a_{2,1} & a_{2,2} & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & a_{1,1} & a_{1,2} \\ 0 & 0 & 0 & 0 & a_{2,1} & a_{2,2} \end{array}\right)$$

- **les sommes tensorielles** : La somme tensorielle de deux matrices carrées A et B est définie comme la somme de facteurs normaux de chacune des matrices selon la formule :

$$A \oplus B = (A \otimes I_n) + (I_n \otimes B)$$

Vous pouvez remarquer que le produit tensoriel est défini pour des matrices non carrées, alors que la somme tensorielle n'est définie que pour les matrices carrées.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \qquad B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}$$

La somme tensorielle définie par $C = A \oplus B$ est égale à

$$C = \left(\begin{array}{ccc|ccc} a_{1,1} & 0 & 0 & a_{1,2} & 0 & 0 \\ 0 & a_{1,1} & 0 & 0 & a_{1,2} & 0 \\ 0 & 0 & a_{1,1} & 0 & 0 & a_{1,2} \\ \hline a_{2,1} & 0 & 0 & a_{2,2} & 0 & 0 \\ 0 & a_{2,1} & 0 & 0 & a_{2,2} & 0 \\ 0 & 0 & a_{2,1} & 0 & 0 & a_{2,2} \end{array}\right) + \left(\begin{array}{ccc|ccc} b_{1,1} & b_{1,2} & b_{1,3} & 0 & 0 & 0 \\ b_{2,1} & b_{2,2} & b_{2,3} & 0 & 0 & 0 \\ b_{3,1} & b_{3,2} & b_{3,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & 0 & 0 & b_{2,1} & b_{2,2} & b_{2,3} \\ 0 & 0 & 0 & b_{3,1} & b_{3,2} & b_{3,3} \end{array}\right)$$

$$C = \left(\begin{array}{ccc|ccc} b_{1,1}+a_{1,1} & b_{1,2} & b_{1,3} & a_{1,2} & 0 & 0 \\ b_{2,1} & b_{2,2}+a_{1,1} & b_{2,3} & 0 & a_{1,2} & 0 \\ b_{3,1} & b_{3,2} & b_{3,3}+a_{1,1} & 0 & 0 & a_{1,2} \\ \hline a_{2,1} & 0 & 0 & b_{1,1}+a_{2,2} & b_{1,2} & b_{1,3} \\ 0 & a_{2,1} & 0 & b_{2,1} & b_{2,2}+a_{2,2} & b_{2,3} \\ 0 & 0 & a_{2,1} & b_{3,1} & b_{3,2} & b_{3,3}+a_{2,2} \end{array}\right)$$

Donc :

$$C_{[k,k][j,l]} = a_{i,j}\delta_{k,\square} + \delta_{i,j}\, b_k,\ avec\ i,j \in [1 \ldots . n_A], et\ k,l \in [1 \ldots . n_B] \quad \textbf{(3)}$$

2. Automates, Événement Synchronisant et Transition Fonctionnelle

Les RAS sont un formalisme pour la définition et la solution des systèmes complexes à grand espace d'états. Il est constitué d'un ensemble d'automates, Chaque automate est constitué d'un ensemble d'états, chaque état a un ensemble d'arcs sortant (transitions) et chaque arc est étiqueté par un ensemble d'événements. Les états internes, ou états locaux d'une automate correspondent aux différents états possibles du sous-système modélisé par cet automate. L'état global est défini comme la combinaison de tous les états internes de chaque automate. L'occurrence d'un événement peut modifier l'état interne d'une ou de plusieurs automates. Lorsqu'un seul automate est impliqué, l'événement est dit local. Sinon, nous parlons d'événement synchronisant. Dans tous les cas, nous avons un changement de l'état global d'un RAS.

Pour représenter graphiquement un RAS, nous pouvons ainsi associer un graphe à chaque automate. Les nœuds du graphe correspondent aux états locaux de l'automate, et les arcs sont étiquetés par une liste d'événements. Il se peut en effet que plusieurs événements distincts produisent le même changement d'état local dans une automate. Dans ce cas, nous avons formellement une transition par événement, et l'arc du graphe représente toutes ces transitions en même temps.

Description Formelle des RAS

Pour donner une représentation formelle d'un RAS on va considérer un RAS composé de :

- **N** automates stochastiques $A^{(i)}, i \in [1 \ldots N]$. Chaque Automate $A^{(i)}$ est constituée d'un ensemble d'états locaux $S^{(i)}$ et d'un ensemble des transitions $T^{(i)}$. On note par $n_i = |S^{(i)}|$ le nombre des états locaux de l'automate $A^{(i)}$ et par $x^{(i)} \in S^{(i)}$ un état local de l'automate $A^{(i)}$;
- Un ensemble d'événements $\mathcal{E}$;

❖ Et la fonction d'atteignabilité **F**, qui est une fonction de $\widehat{S}$ (l'ensemble des états globaux) dans $\{0,1\}$. Selon cette définition, un état global x est accessible si et seulement si $F(x) = 1$. L'ensemble des états accessibles du RAS est noté $S \subset \widehat{S}$.

Les concepts utilisés dans le RAS seront abordés en détail dans la partie suivante :

1. **Les événements** : Un événement $e \in \mathcal{E}$ est défini par :
   - un taux de franchissement λ_e, qui est une fonction de $\widehat{S}$ dans $\mathbb{R}^+$;
   - un ensemble d'automates impliqués par cet événement $\mathbf{O}_e$ (l'occurrence de e entraîne une modification de l'état local de chacune de ces automates);
   - un automate maître $A^{(\eta_e)} \in \mathbf{O}_e$;

   L'événement e est dit local à l'automate $A^{(\eta_e)}$ si $\mathbf{O}_e$ est réduit à l'automate $A^{(\eta_e)}$. Sinon, l'événement e est synchronisant. Selon cette définition on peut dire que pour tout état global $x = (x^{(1)}, \dots, x^{(N)}) \in \widehat{S}$, l'événement **e** est dit possible si et seulement si, pour toute automate $A^{(i)} \in \mathbf{O}_e$, il existe au moins une transition $t^{(i)} \in T^{(i)}$ telle que $x_{t^{(i)},dept} = x^{(i)}$ et $evt_{t^{(i)}} = e$;

2. **Les transitions** : Pour chaque automate $A^{(i)}$, une transition $t^{(i)} \in T^{(i)}$ est définit par :
   - un état de départ $x_{t^{(i)},dept} \in S^{(i)}$ et un état d'arrivée $x_{t^{(i)},arr} \in S^{(i)}$;
   - un événement associé à la transition $evt_{t^{(i)}} = e \in \mathcal{E}$;

- une probabilité de routage $\pi_{t^{(i)}}$, qui est une fonction de $\widehat{S}$ dans l'intervalle $[0\ 1]$. Si $\epsilon \widehat{S}$, $\pi_{t^{(i)}}(x)$ est la fonction évaluée pour l'état x;

3. **Probabilité de routage $\pi_{t^{(i)}}$:** Il se peut qu'un même événement e, en partant du même état, puisse mener dans plusieurs états différents. Dans ce cas, il faut définir la probabilité d'aller dans chacun des états d'arrivée, et la somme de ces probabilités doit être égale à 1 . Ces probabilités sont appelées les probabilités de routage. Formellement, quand un événement **e** a lieu avec un taux de franchissement λ_e et les deux états départ/arrivé respectivement sont $x = (x^{(1)}, \dots, x^{(N)}\ \epsilon \widehat{S}$, alors :

   Pour toute $A^{(i)}\ \epsilon\ O_e$, pour chaque transition $t^{(i)} \epsilon\ T^{(i)}$ telle que $x_{t^{(i)},dept} = x^{(i)}$ et $evt_{t^{(i)}} = e$ la probabilité que $x'_{t^{(i)}} = x_{t^{(i)},arr}$ est $\pi_{t^{(i)}}(x)$;

4. **Élément fonctionnel :** Un élément fonctionnel $f(A^w)$ est une fonction de $\prod_{i \epsilon w} S^{(i)}$ dans $\mathbb{R}^+$ où w est un sous-ensemble de $[1 \dots N]$

5. **Automate :** Chaque automate $A^{(i)}$ est définit par :

   - $S^{(i)}$ est l'ensemble des états de l'automate $A^{(i)}$;
   - $Q^{(i)}$ est la fonction de transition de l'automate $A^{(i)}$ définie de $S^{(i)} \times S^{(i)}$ dans $T^{(i)}$.

Descripteur Markovien

L'un des avantages apporté par le formalisme des réseaux d'automates stochastiques est la facilité de décrire le générateur de la chaîne de Markov associée au modèle complet de façon compacte, grâce à u ne formule mathématique appelée descripteur **[16, 17].** Cette formule

mathématique décrit, à partir de la description de chaque automate, le générateur infinitésimal de la chaîne de Markov associée au SAN.

Soit $\mathbf{E} \subset \mathcal{E}$ l'ensemble des événements synchronisant. Le comportement de chaque automate $\boldsymbol{A^{(i)}, i = 1 \ldots N}$, est décrit par un ensemble de matrices carrées, toutes de dimension $\boldsymbol{n_i}$. Ces matrices peuvent être regroupées en deux familles :

- une matrice regroupant tous les taux de transitions locaux correspondant aux événements locaux de l'automate $\boldsymbol{A^{(i)}}$, appelée la matrice de transition locale $\boldsymbol{Q_l^{(i)}}$;
- $\boldsymbol{2E}$ matrices regroupant tous les triplets de synchronisation pour les événements $\boldsymbol{e}$ de l'ensemble $\boldsymbol{E}$ appelées $\boldsymbol{Q_{e^+}^{(i)}}$ et $\boldsymbol{Q_{e^-}^{(i)}}$:
  - $\boldsymbol{Q_{e^+}^{(i)}}$ représente la matrice de synchronisation positive de $\boldsymbol{A^{(i)}}$, qui représente l'événement synchronisant $\boldsymbol{e}$, et ses taux de franchissement si $\boldsymbol{A^{(i)}}$ est le maître de l'événement ;
  - $\boldsymbol{Q_{e^-}^{(i)}}$ la matrice de synchronisation négative de $\boldsymbol{A^{(i)}}$, qui correspond à une mise à jour des éléments diagonaux pour l'événement $\boldsymbol{e}$.

Soit :

$\boldsymbol{Q_j^{(i)}(x^{(i)}, y^{(i)})}$: L'élément de la matrice $\boldsymbol{Q_j^{(i)}}$ dans la ligne $\boldsymbol{x^{(i)}}$ et la colonne $\boldsymbol{y^{(i)}}$, avec $\boldsymbol{i \in [1 \ldots N]}$ et $\boldsymbol{j \in \{l, e^+, e^-\}}$.

$\boldsymbol{I_{n_i}}$: La matrice identité de taille $\boldsymbol{n_i}$ avec $\boldsymbol{i \in [1 \ldots N]}$.

1. **La matrice de transition locale** : Les éléments de la matrice de transition locale $Q_l^{(i)}$ de l'automate $A^{(i)}$ sont définis par :

   - $\forall x^{(i)}, y^{(i)} \in S^{(i)} \mid \forall x^{(i)} \neq y^{(i)} \; Q_l^{(i)}(x^{(i)}, y^{(i)}) = \tau_l[x^{(i)}, y^{(i)}]$
   - $\forall x^{(i)} \in S^{(i)}$

     $Q_l^{(i)}(x^{(i)}, x^{(i)}) = -\sum_{y^{(i)} \in S^{(i)},\ x^{(i)} \neq y^{(i)}} (\tau_l[x^{(i)}, y^{(i)}])$

2. **La matrice de synchronisation positive** : Les matrices des transitions synchronisant représentant l'occurrence de l'événement $e \in E$ sont définies par :

   - $\forall A^{(i)} \notin O_e$

     $Q_{e^+}^{(i)} = I_{n_i}$;

   - $\forall A^{(i)} \in O_e, \forall x^{(i)}, y^{(i)} \in S^{(i)}$

     $Q_{e^+}^{(i)}(x^{(i)}, y^{(i)}) = \tau_e[x^{(i)}, y^{(i)}]$;

$\tau_e[x^{(i)}, y^{(i)}]$ Représente Le taux synchronisant de transition associé à l'événement e de $x^{(i)}$ vers $y^{(i)}$. Ce taux est défini par :

- ✓ S'il existe une transition $t^{(i)} \in T^{(i)}$, telle que $x_{t^{(i)},dept} = x^{(i)}$ et $x_{t^{(i)},arr} = y^{(i)}$ et $evt_{t^{(i)}} = e$ alors :
  - Si $i = \eta_e(A^{(i)}$est l'automate maître),$\tau_e[x^{(i)}, y^{(i)}] = \lambda_e \times \pi_t^{(i)}$;

- Sinon, $\tau_e\left[x^{(i)}, y^{(i)}\right] = \pi_t^{(i)}$;

✓ Sinon, s'il n'existe pas de transition étiquetée par l'événement synchronisant e allant de $x^{(i)}$ vers $y^{(i)}$ dans l'automate $A^{(i)}$, $\tau_e\left[x^{(i)}, y^{(i)}\right] = 0$;

3. **la matrice de synchronisation négative** : représentant l'ajustement arithmétique nécessaire pour construire un générateur. pour $e \in E$ sont définies par :

➡ $\forall A^{(i)} \notin O_e$

$$Q_{e^-}^{(i)} = I_{n_i} ;$$

➡ $\forall x^{(\eta_e)} \in$

$$S^{(\eta_e)} Q_{e^-}^{(\eta_e)}\left(x^{(\eta_e)}, x^{(\eta_e)}\right) = -\sum_{y^{(\eta_e)} \in S^{(\eta_e)}, x^{(\eta_e)} \neq y^{(\eta_e)}} \tau_e\left[x^{(\eta_e)}, y^{(\eta_e)}\right] * \pi_e\left[x^{(\eta_e)}, y^{(\eta_e)}\right]$$

➡ $\forall A^{(i)} \in O_e$ tel que $i \neq \eta_e, \forall x^{(i)} \in S^{(i)}$

$$Q_{e^-}^{(i)}\left(x^{(i)}, x^{(i)}\right) = -\sum_{y^{(i)} \in S^{(i)}, x^{(i)} \neq y^{(i)}} \pi_e\left[x^{(i)}, y^{(i)}\right]$$

➡ $\forall A^{(i)} \in O_e, \forall x^{(i)}, y^{(i)} \in S^{(i)}$ et $x^{(i)} \neq y^{(i)}$

$$Q_{e^-}^{(i)}\left(x^{(i)}, y^{(i)}\right) = 0 ;$$

4. **Le générateur Markovien** :correspondant à la chaîne de Markov associée à un RAS est défini par la formule tensorielle appelée Descripteur Markovien :

$$Q = \bigoplus_{i=1}^{N}{}_{g} \; Q_l^{(i)} + \sum_{e \in E} \left(\bigotimes_{i=1}^{N}{}_{g} \; Q_{e^+}^{(i)} + \bigotimes_{i=1}^{N}{}_{g} \; Q_{e^-}^{(i)} \right) \quad \textbf{(4)}$$

Étant donné que toute somme tensorielle est équivalente à une somme de produits tensoriels particuliers, le descripteur peut être décrit par :

$$Q = \sum_{j=1}^{N+2|E|} \bigotimes_{i=1}^{N}{}_{g} \; Q_j^{(i)} \text{ Avec,}$$

$$Q_j^{(i)} = \begin{cases} I_{n_i} & \text{pour } j \leq N \text{ et } j \neq i; \\ Q_l^{(i)} & \text{pour } j \leq N \text{ et } j = i; \\ Q_{(j-N)^+}^{(i)} & \text{pour } N < j \leq N + E \\ Q_{(j-(N+E))^-}^{(i)} & \text{pour } j > N + E \end{cases}$$

Donc, les matrices de transition nécessaires à l'écriture de cet équation peuvent être résumées dans le tableau suivante :

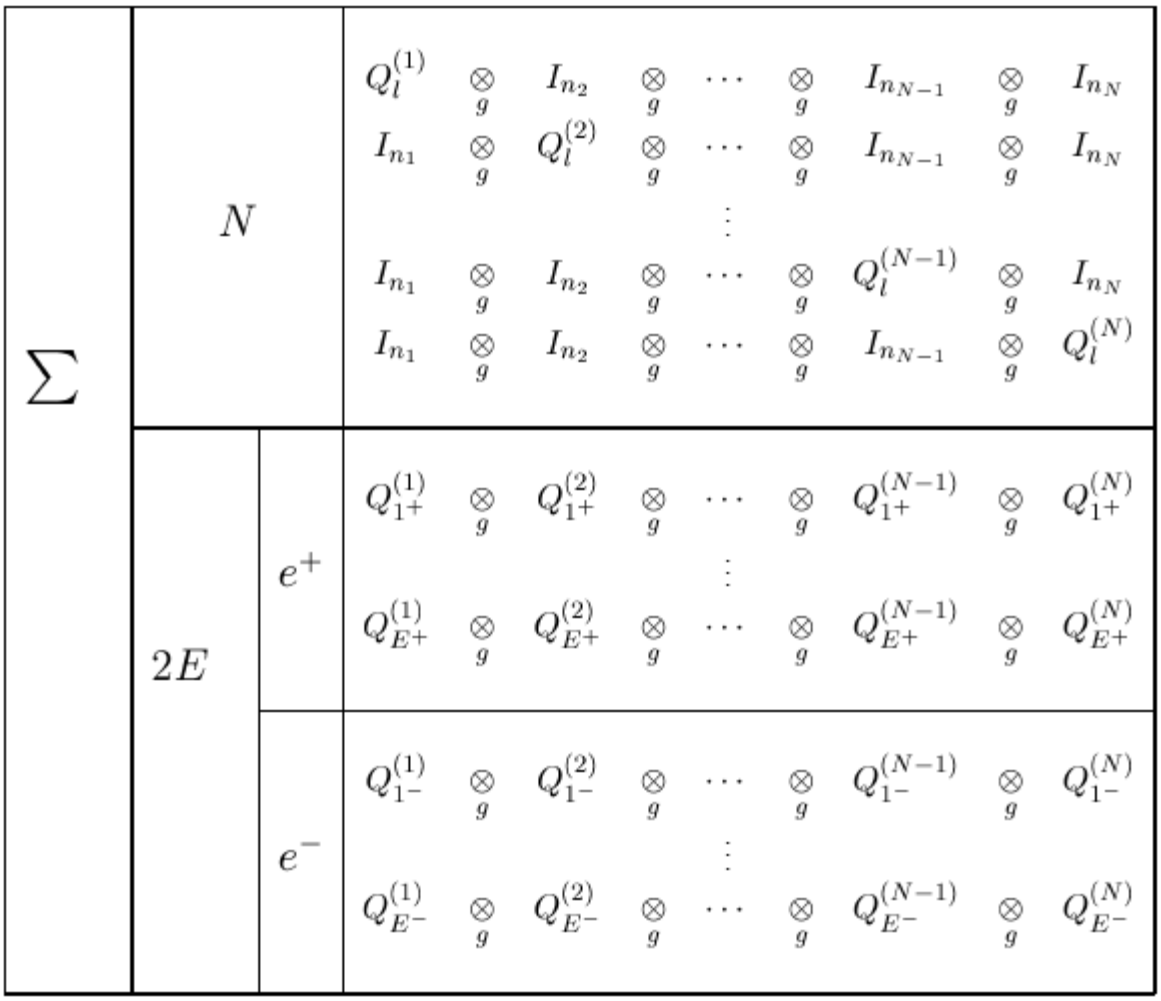


Figure 19: Descripteur Markovien.

D. PEPS

PEPS (Performance Evaluation of Parallel Systems) est un outil informatique qui permet à la fois la définition et la solution de modèles utilisant le formalisme des réseaux d'automates stochastiques et algèbre tensorielle. Sa première version fût proposée par Plateau, Fourneau et Lee**.** Actuellement il est dans sa version PEPS 2007**.**

PEPS reçoit la description d'un modèle RAS et calcule le vecteur de probabilité associé à la chaine de Markov correspondant au modèle RAS décrit. Pour cela il utilise un formalisme textuel pour décrire les modèles, qui conserve l'élément cléf du formalisme RAS, à savoir sa spécification modulaire [**16].** Cette description textuelle est simple, extensible et flexible :

- **simple** parce qu'il y a peu de mots réservés, juste assez pour délimiter les différents niveaux de modularité ;
- **extensible** car la définition des modèles SAN est faite de façon hiérarchique ;
- **flexible** grâce à l'inclusion de structures de réplication, qui permettent d'une part la réutilisation d'automates identiques, et d'autre part la construction d'automates ayant des blocs d'états de comportement identique répétés, comme on le trouve souvent dans les modèles de réseaux de file d'attente.

Entre le format d'entrée du modèle RAS et la génération du vecteurs de la probabilité un certains nombre des structures de données intermédiaires peut être généré. Par la suite on va décrire le format d'entrée et l'ensemble des étapes nécessaires pour la résolution du modèle.

1. Format textuel

identifiers

Le premier bloc **identifiers** contient les déclarations de tous les paramètres : valeurs numériques, fonctions, ou ensemble d'indices (domaines) qui seront utilisés pour les répliquas dans la définition du modèle.

event

Ce bloc définit chaque événement du modèle en précisant pour chacun :

- son type : événement local loc ou synchronisant syn;
- son nom $< event\_name >$;
- son taux de franchissement $< rate >$ qui peut être une valeur numérique ou une fonction définie précédemment dans le bloc **identifiers.**

Nous pouvons également répliquer les événements en utilisant des ensembles d'indices $< replication\_domain >$. Ceci est notamment utilisé lorsqu'un ensemble d'automates possèdent des événements avec un même taux de franchissement.

reachability

Ce bloc définit la fonction d'accessibilité du modèle RAS. C'est une fonction boolean, qui renvoie une valeur non nulle pour les états accessibles $\widehat{S}$, et la valeur 0 pour les états non accessible $S - \widehat{S}$.

network

Ce bloc représente la partie principale du SAN, il a une structure hiérarchique. Cette structure hiérarchique représente un RAS constitué d'un ensemble d'automates, Chaque automate est constitué d'un ensemble d'états, chaque état a un ensemble d'arcs sortant (transitions) et chaque arc est étiqueté par un ensemble d'événements.

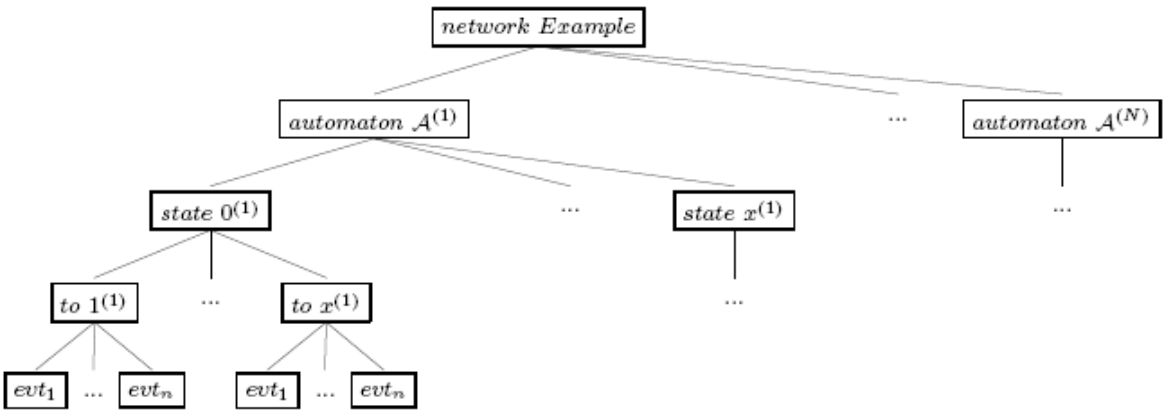


Figure 20 : structure hiérarchique d'un RAS.

Dans le format textuel, La première ligne du bloc « **Network** » contient des informations générales sur le RAS tel que le nom du modèle et le type d'échelle de temps du modèle, qui peut être **"continuous"** ou **"discrete"** suivant que le modèle soit à temps continu ou à temps discret.

Le mot-clé **aut** sert à délimiter les automates et $< replication\_domain >$ désigne la possibilité d'utilisé la réplication. Dans ce cas, si $i \in [replication\_domain]$, et **A** est le nom de l'automate (aut_name), alors $A[i]$ est l'identifiant de l'un des automates répliquées.

```
identifiers
      < id_name >=< exp >;
      < dom_name >= [i ...j];
event
       //sans replication
                      loc  < event_name > (< rate >)
                      syn  < event_name > (< rate >)
      //avec replication
                      loc  < event_name > [ replication_domain] (< rate >)
                      syn  < event_name > [ replication_domain](< rate >)
{partial} reachability = < exp >
Network < net_name > (< type >)
       out < aut_name > {[replication_domain]}
          stt < stt_name > {[replication_domain]}{(reward)}
             to (< stt_name > {[replication_domain]}{/f _cond})
                 < event_name > {[replication_domain]}{(< prob >)}{/f _cond}
                 ...
                 < event_name > {[replication_domain]}{(< prob >)}{/f _cond}
                 ...
                 ...
           from < stt_name >
                to (< stt_name > {[replication_domain]}{/f _cond})
                   < event_name > {[replication_domain]}{(< prob >)}{/f _cond}
                   ...
          stt < stt_name > {[replication_domain]}{(reward)}
             to (< stt_name > {[replication_domain]}{/f _cond})
                 < event_name > {[replication_domain]}{(< prob >)}{/f _cond}
                 ...
        out < aut_name > {[replication_domain]}
        ...
Results
        < res_name >=< exp >;
```

Figure 21 : Structure modulaire du format textuel de PEPS 2007.

Le mot-clé *stt* annonce la définition d'un état. *stt_name* est le nom de l'état, et nous pouvons utiliser [*replication_domain*], pour répliquer l'état. Le mot clé *reward* représente la possibilité d'utiliser les récompenses.

Une description de chaque arc sortant de cet état est donnée par la définition d'une section *to*() . L'identifiant *stt_name* dans la parenthèse indique l'état d'arrivée de l'arc. Dans un groupe d'états répliqués, l'expression d'autres états du groupe peut être faite par des références aux positions : l'état courant (==), précédent (– –) ou suivant (++).

f_conf permet de définir une condition sur la transition. Cette condition est une fonction qui est généralement définie dans le bloc **identifiers.** Normalement, les valeurs retournées sont en fonction de l'automate actuelle et / ou de l'état courant/cibles.

Finalement, chaque arc est associé à un ensemble d'événements qui peuvent déclencher la transition. Ils sont exprimés par leur nom *event_name*, et éventuellement (si différente de 1) par la probabilité de routage associée à cette transition *prob* .Les différents événements sont séparés par des espaces ou des sauts de ligne.

La section *from* est relativement semblable à la section *stt*, sauf qu'elle ne définit pas d'état local. Elle sert à rajouter des arcs qui ne peuvent pas être définis dans une section *stt*. Typiquement, nous nous en servons pour définir un arc sortant d'un état particulier d'un groupe d'états répliqués pour aller dans un état en dehors du groupe. Une file d'attente ayant des états initiaux ou finaux particuliers peut utiliser ce type de définition d'arcs.

Enfin, les fonctions utilisées pour calculer des indices de performances sur le SAN sont définies dans le bloc **results.**

2. Structure de données

L'un des avantages apportés par le formalisme des réseaux d'automates stochastiques est la facilité de décrire le générateur de la chaîne de Markov associée au modèle complet de façon compacte, grâce à une formule mathématique appelée descripteur, et bien sur la possibilité de calculer le produit vecteur-descripteur sans jamais exprimer la matrice de façon explicite

(approche Kronecker). Cette approche fait partie d'un ensemble des techniques et algorithmes qui portent sur la manière dont les données sont stockées, (la représentation du **générateur** et les **vecteurs de probabilité),** ce qui influe largement sur le calcule du vecteur des probabilités stationnaires. Les deux méthodes implémentées dans PEPS 2007 sont les suivantes [**16**] :

- **Stockage explicite, matrice en format creux** : consiste à stocker le descripteur de façon explicite en ne gardant que les éléments non nuls et leur position dans la matrice (ceux-ci est le format Harwell-Boeing). Les vecteurs de probabilité sont alors de la taille de l'espace d'états accessibles.
- **Approche de Kronecker** : consiste à u tiliser un ensemble de petites matrices de dimension $n_{i,}$ pour stocker un grand modèle. n_i est le nombre d'états locaux de l'automate $A^{(i)}\ avec\ i \in [1 \dots N]$. Cette approche sera décrite ultérieurement.

3. L'agrégation

L'utilisation de deux techniques précédemment définies n'est pas toujours suffisante pour pallier au problème de l'explosion du nombre d'états. Pour surmonter ce problème, on peut jouer sur le fait que de nombreux systèmes contiennent un grand nombre de composants identiques. Donc, on peut alors exploiter ces réplications de composants pour générer une chaîne de Markov réduite, en effectuant une agrégation.

L'agrégation est basée sur le fait que, grâce aux répliquas, certains états globaux du SAN ont la même probabilité stationnaire. Pour cela, tous les états du SAN initial équivalents par une permutation de P sont groupés en un unique état de la chaîne de Markov agrégée. Chaque état particulier représente une classe d'équivalence.

Pour finir, l'agrégation peut être utilisée avec les deux techniques citées précédemment. Avec la technique de stockage explicite, en stockant la matrice en format creux agrégé. Pour la deuxième approche, il existe une expression tensorielle de la matrice de la chaîne de Markov agrégée.

4. Méthodes Itératives de Résolution des RAS

Les cibles principales des RAS sont les problèmes à g rands espaces d'états et pour ces problèmes les méthodes itératives sont les plus adéquates. La solution qui nous intéresse est

un vecteur de probabilités, donc normalisé de façon à ce que la somme des éléments soit égale à un. Le système linéaire à résoudre est:

$$xQ = 0$$

Trois méthodes de résolution sont implémentées dans PEPS. Ces méthodes sont les suivantes :

- Méthode de la puissance;
- Méthode d'Arnoldi;
- Méthode GMRES;

Ces trois méthodes de résolution peuvent être appliquées non seulement aux matrices en format explicite mais peuvent aussi être employées pour les matrices en format descripteur.

A titre d'exemple on va détailler la première méthode (méthode de la puissance), et les lecteurs intéressés pourront voir les détails des deux autres méthodes dans la [17].

Mais avant de décrire cette méthode, on va expliquer comment on peut utiliser les propriétés de l'algèbre tensorielle pour faire la résolution sans calculer explicitement la matrice du générateur Markovien (L'Algorithme du shuffle) :

$$xQ = x\left(\bigoplus_{i=1}^{N}{}_{g}\ Q_{l}^{(i)} + \sum_{e \in E}\left(\bigotimes_{i=1}^{N}{}_{g}\ Q_{e^{+}}^{(i)} + \bigotimes_{i=1}^{N}{}_{g}\ Q_{e^{-}}^{(i)}\right)\right)$$

d. L'Algorithme du shuffle :

Dans L'Algorithme du shuffle **[17]**, on remarque que l'opération de base lors du calcul de la solution stationnaire d'une chaîne de Markov en utilisant des méthodes de résolution itératives appliquées sur le format Kronecker est le produit d'un vecteur et d'un terme produit tensoriel généralisé :

$$x \bigotimes_{i=1}^{N}{}_{g}\ Q^{(i)}$$

C'est l'utilisation de la propriété d'algèbre tensorielle suivante qui a permis d'appliquer ces trois méthodes au format descripteur :

$$\begin{aligned} Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-1)} \otimes Q^{(N)} = \\ & Q^{(1)} \otimes I_{nright_1} \\ & \times I_{nleft_2} \otimes Q^{(2)} \otimes I_{nright_2} \\ & \times \cdots \\ & \times I_{nleft_{N-1}} \otimes Q^{(N-1)} \otimes I_{nright_{N-1}} \\ & \times I_{nleft_N} \otimes Q^{(N)} \end{aligned}$$

Où

$\boldsymbol{n_i}$ la dimension de la i-ème matrice d'une suite;

$\boldsymbol{nleft_i}$ le produit des dimensions de toutes les matrices à gauche de la i-ème matrice d'une suite, $\prod_{k=1}^{i-1} n_k$.

$\boldsymbol{nright_i}$ le produit des dimensions de toutes les matrices à droite de la i-ème matrice d'une suite, $\prod_{k=i+1}^{N} n_k$.

$\bar{n}$ le produit des dimensions de toutes les matrices sauf la i-ème matrice d'une suite, $\prod_{k=1,k\neq i}^{N} n_k$.

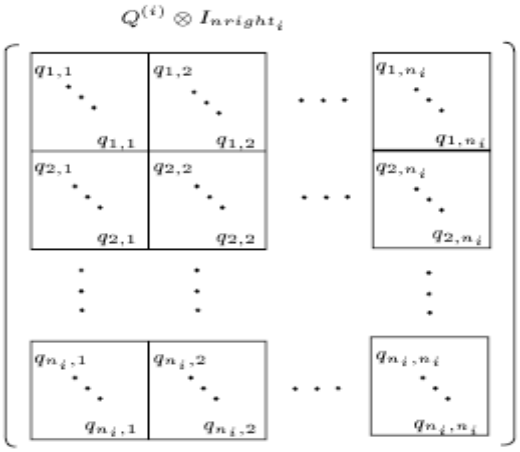
Donc, dans cette méthode, pour calculer la multiplication d'un vecteur $\boldsymbol{x}$ par le terme $\bigotimes_{g=1}^{N} Q^{(i)}$ il est suffisant de savoir multiplier ce vecteur par un facteur normal i avec $i \in \{1 \ldots N\}$:

$$x \times I_{nleft_i} \otimes Q^{(i)} \otimes I_{right_i}$$

Etant donné que,

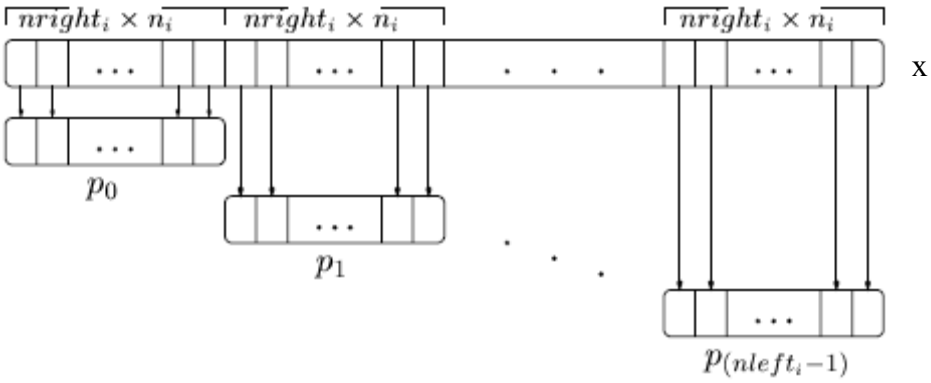
$$Q^{(i)} \otimes I_{nright_i} = \begin{pmatrix} q^{(i)}_{1,1} I_{nright_i} & q^{(i)}_{1,2} I_{nright_i} & \cdots & q^{(i)}_{1,n_i} I_{nright_i} \\ q^{(i)}_{2,1} I_{nright_i} & q^{(i)}_{2,2} I_{nright_i} & \cdots & q^{(i)}_{2,n_i} I_{nright_i} \\ \vdots & \vdots & \ddots & \vdots \\ q^{(i)}_{n_i,1} I_{nright_i} & q^{(i)}_{n_i,2} I_{nright_i} & \cdots & q^{(i)}_{n_i,n_i} I_{nright_i} \end{pmatrix}$$

Donc, chaque matrice $q^{(i)}_{j,k} I_{right_i}$ $(j, k \in [1 \ldots n_i)$ est une matrice diagonale de taille $nright_i$ dont tous les éléments de la diagonale sont égaux à $q^{(i)}_{j,k}$.



Donc, la matrice $I_{nleft_i} \otimes Q^{(i)} \otimes I_{right_i}$ est une matrice bloc diagonale dont les blocs sont constitués de la matrice $Q^{(i)} \otimes I_{right_i}$. Les différents blocs peuvent être traités de façon indépendante, ce qui suggère la possibilité d'introduire du parallélisme.

Il y a $nleft_i$ blocs de matrices, et chacun d'entre eux doit être multiplié par une portion différente de vecteur x :



Ainsi, l'algorithme du shuffle consiste en une boucle sur les $nleft_i$ l-portions, et à chaque itération il calcule la l-portion résultat :

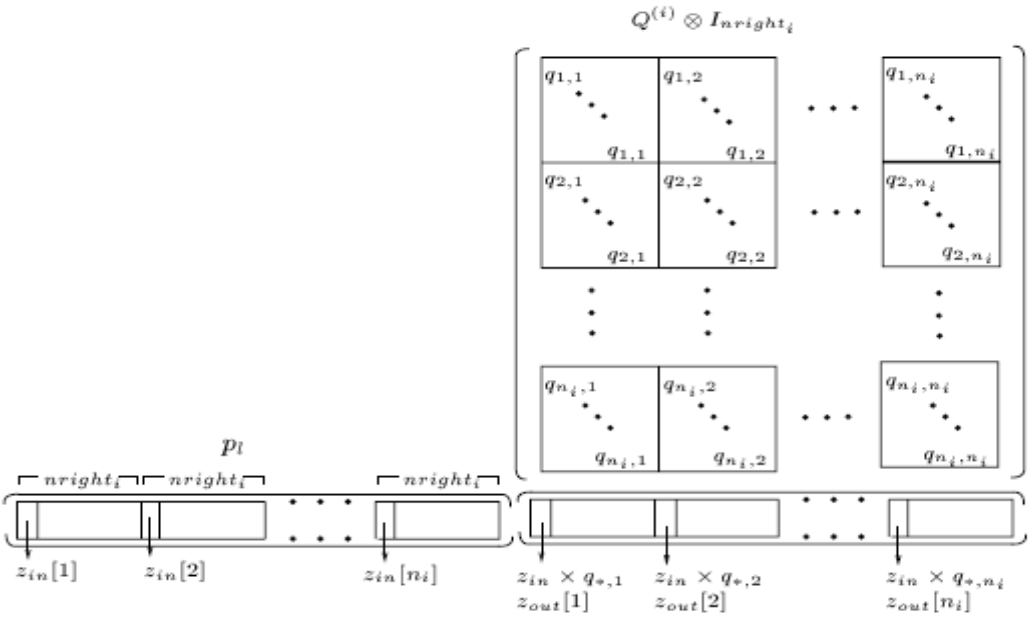
$$r_l = p_l \times Q^{(i)} \otimes I_{right_i}$$

Le calcul d'un élément de r_l correspond à la multiplication de p_l par une colonne de la matrice $Q^{(i)} \otimes I_{right_i}$. Or, chaque colonne de cette matrice est composée d'éléments d'une seule colonne de $Q^{(i)}$. Les autres éléments sont nuls. La multiplication revient donc à extraire de façon répétée des éléments de p_l (distants de $nright_i$), à former un vecteur appelé z_{in} à partir de ces éléments, puis à multiplier z_{in} par une colonne de la matrice $Q^{(i)}$.

L'extraction d'un z_{in} revient à accéder au vecteur p_l et à choisir les éléments espacés de $nright_i$ positions dans le vecteur.

Une fois obtenu, la multiplication d'un z_{in} par la matrice $Q^{(i)}$ en entier donne ainsi plusieurs éléments du résultat. Ces éléments constituent un sous-vecteur appelé z_{out}, et les positions des éléments de z_{out} dans r_l correspondent aux positions des éléments de z_{in} dans p_l.

La figure suivante illustre la multiplication qui a été expliquée précédemment :



Cet algorithme peut être décrit sous la forme suivante :

```
1   for i = 1, 2, ..., N            // boucle sur les facteurs normaux
2   do  base = 0 ;          // base : indice du début de la portion dans le vecteur
3       for l = 0, 1, ..., nleft_i − 1          // boucle sur les portions
4       do  for r = 0, 1, ..., nright_i − 1 // détail de la portion : boucle sur les z_in
5           do  index = base + r ;
            // index : indice de l'élément courant du z_in dans le vecteur
6               for k = 0, 1, ..., n_i − 1 // extraction du z_in n° r (sauts de nright_i)
7               do  z_in[k] = π̂[index] ;
8                   index = index + nright_i ;
9               end do
10              multiply    z_out = z_in × Q^(i) ;          // multiplication
11              index = base + r ;
                // index : indice de l'élément courant du z_out dans le vecteur
12              for k = 0, 1, ..., n_i − 1 // stockage du résultat z_out (sauts de nright_i)
13              do  π̂[index] = z_out[k] ;
14                  index = index + nright_i ;
15              end do
16          end do
17          base = base + ( nright_i × n_i ) ; // on passe à la portion suivante (saut dans π̂)
18      end do
19  end do
```

Dans le cas où notre réseau d'automate stochastique contient un élément fonctionnel, on doit faire une évaluation des éléments fonctionnels de chaque matrice $Q^{(i)}$ avant leur multiplication par une tranche z_{in} du vecteur $\mathbf{x}$.

Pour conclure, cet algorithme permet de calculer le produit vecteur-descripteur sans jamais exprimer la matrice de façon explicite. Cependant, cet algorithme requiert l'utilisation de

vecteurs de probabilité de la taille de l'espace d'états produit S. Ces vecteurs sont appelés **vecteurs étendus.** Des améliorations ont été proposées dans [16]. L'idée consiste à calculer dans un premier temps l'espace d'états accessibles (PEPS le permet), puis à résoudre le modèle en utilisant des vecteurs d'itération x ne contenant des entrées que pour les états accessibles (vecteurs de la taille de S). Par opposition aux vecteurs étendus, ces vecteurs sont appelés **vecteurs réduits.**

Pour savoir plus sur les techniques utilisées pour faire l'évaluation des éléments fonctionnels de chaque matrice et comment on précède pour introduire l'amélioration par vecteur réduit vous prouvez se rapporter à ce document [16].

e. Méthode de la Puissance

La méthode de la puissance est la solution itérative la plus simple pour un système linéaire définie par:

$$x\,Q = 0$$

Cette solution doit être précédée par l'uniformisation du descripteur, *i.e.*, la transformation d'un générateur infinitésimal Q en matrice stochastique P. Le descripteur doit être normalisé et une matrice identité doit être ajoutée :

$$P = I_{n_Q} + \frac{Q}{\alpha}$$

Où α est le plus grand élément en module du descripteur Q :

Cette transformation du système linéaire à résoudre correspond aux équivalences:

$$0 = x\,Q$$

$$\Leftrightarrow 0 = x\,\frac{Q}{\alpha}$$

$$\Leftrightarrow x = x + x\,\frac{Q}{\alpha}$$

$$\Leftrightarrow x = x(I_{n_Q} + \frac{Q}{\alpha})$$

Ce système est résolu par la méthode de la puissance avec le schéma itératif suivant:

$$x^{(k+1)} = x^{(k)}(I_{n_Q} + \frac{Q}{\alpha})$$

Où $x^{(k)}$ représente la solution après la k-ème itération.

L'algorithme suivant représente l'application de la méthode de la puissance pour un descripteur Q en partant d'un vecteur initial v et en arrivant à la solution x.

```
1  x^(0) = v
2  Q = I_{n_Q} + Q/α
3  for k = 0, 1, ... ∞
4  do multiply   x^(k+1) = x^(k) Q
5     if   convergence test
6        stop
7  end do
8  x = x^(k+1)
```

Le test de convergence exprimé dans la ligne 5 est exprimé dans PEPS par une comparaison simple des vecteurs des deux dernières itérations. La valeur α par $\alpha = max_{i=1...N}(Q(i,i))$ (la plus grande valeur de tous les éléments de la diagonale du descripteur).

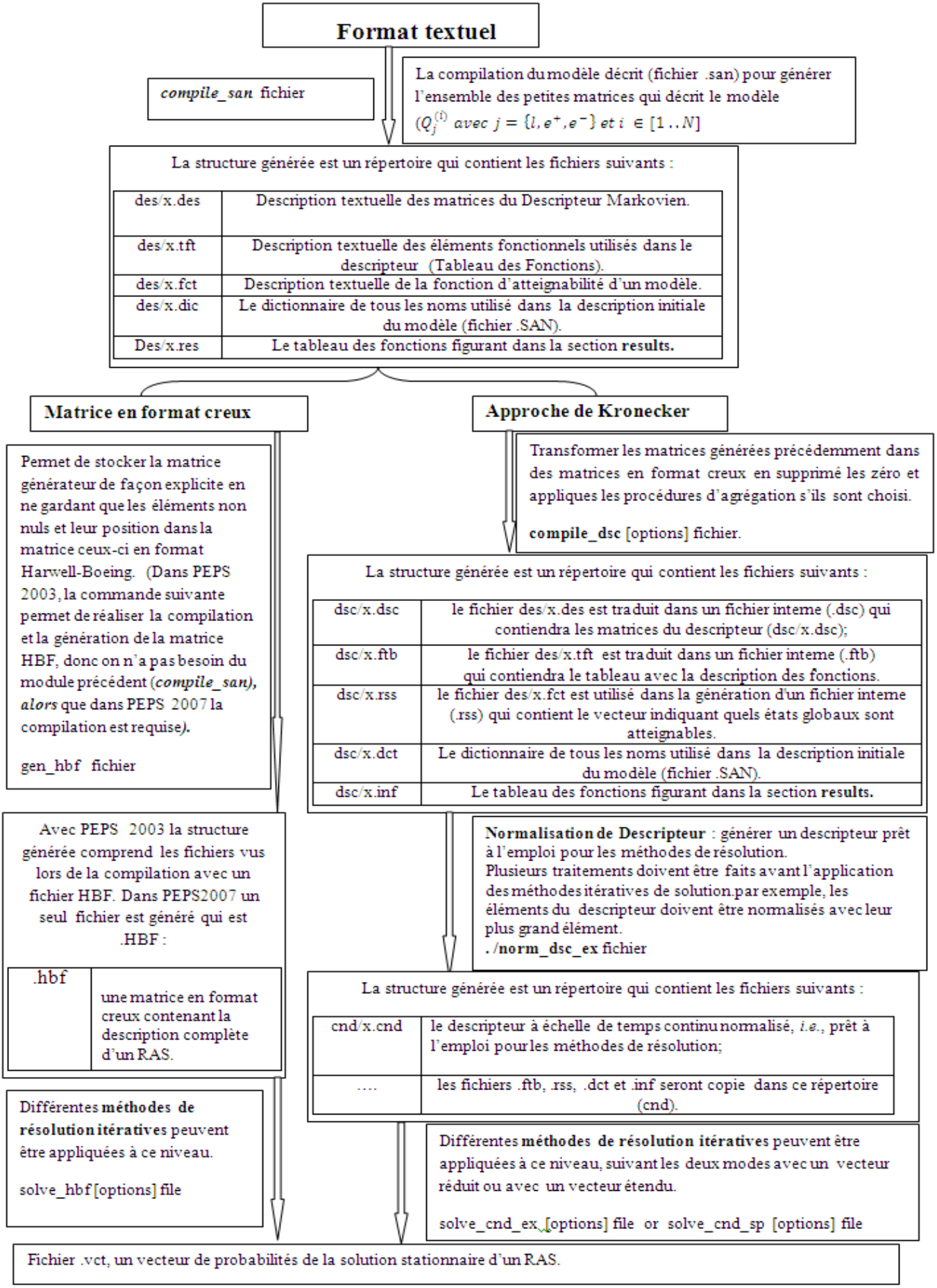
5. LA structure des fichiers générés par PEPS

PEPS reçoit la description d'un modèle RAS et calcule le vecteur de probabilités associé à la chaîne de Markov correspondant au modèle RAS décrit. Entre le format d'entrée du modèle RAS et la génération du vecteur de probabilités un certain nombre des fichiers de données intermédiaires peut être générés. Quatre structures de cet ensemble des fichiers sont les plus importants. Ces Quatre fichiers correspondent aux structures qui décrivent le modèle RAS et leur résolution :

- le tableau des fonctions contenant tous les éléments fonctionnels qui peuvent apparaître dans le modèle;

- le descripteur Markovien qui décrit l'ensemble de matrices contenant les éléments constants (nombres réels) et les éléments fonctionnels (identifiés par le tableau des fonctions);
- une fonction d'atteignabilité décrivant les états atteignables de l'espace produit du modèle.
- La dernière structure correspond à un fichier qui contient un vecteur de probabilité associant une probabilité à ch aque état de la chaîne. Cette probabilité représente la proportion de temps que la chaine de Markov reste dans chacun des états sur une trajectoire de durée infinie.

Le schéma suivant représente un processus complet pour la compilation, normalisation et la résolution d'un modèle RAS :


Format textuel
compile_san fichier
La compilation du modèle décrit (fichier .san) pour générer l'ensemble des petites matrices qui décrit le modèle
$(Q_j^{(i)}$ avec $j = \{l, e^+, e^-\}$ et $i \in [1..N]$
La structure générée est un répertoire qui contient les fichiers suivants :
des/x.des
Description textuelle des matrices du Descripteur Markovien.
des/x.tft
Description textuelle des éléments fonctionnels utilisés dans le descripteur (Tableau des Fonctions).
des/x.fct
Description textuelle de la fonction d'atteignabilité d'un modèle.
des/x.dic
Le dictionnaire de tous les noms utilisé dans la description initiale du modèle (fichier .SAN).
Des/x.res
Le tableau des fonctions figurant dans la section results.
Matrice en format creux
Approche de Kronecker
Permet de stocker la matrice générateur de façon explicite en ne gardant que les éléments non nuls et leur position dans la matrice ceux-ci en format Harwell-Boeing. (Dans PEPS 2003, la commande suivante permet de réaliser la compilation et la génération de la matrice HBF, donc on n'a pas besoin du module précédent (compile_san), alors que dans PEPS 2007 la compilation est requise).
gen_hbf fichier
Transformer les matrices générées précédemment dans des matrices en format creux en supprimé les zéro et appliques les procédures d'agrégation s'ils sont choisi.
compile_dsc [options] fichier.
La structure générée est un répertoire qui contient les fichiers suivants :
dsc/x.dsc
le fichier des/x.des est traduit dans un fichier interne (.dsc) qui contiendra les matrices du descripteur (dsc/x.dsc);
dsc/x.ftb
le fichier des/x.tft est traduit dans un fichier interne (.ftb) qui contiendra le tableau avec la description des fonctions.
dsc/x.rss
le fichier des/x.fct est utilisé dans la génération d'un fichier interne (.rss) qui contient le vecteur indiquant quels états globaux sont atteignables.
dsc/x.dct
Le dictionnaire de tous les noms utilisé dans la description initiale du modèle (fichier .SAN).
dsc/x.inf
Le tableau des fonctions figurant dans la section results.
Avec PEPS 2003 la structure générée comprend les fichiers vus lors de la compilation avec un fichier HBF. Dans PEPS2007 un seul fichier est généré qui est .HBF :
.hbf
une matrice en format creux contenant la description complète d'un RAS.
Normalisation de Descripteur : générer un descripteur prêt à l'emploi pour les méthodes de résolution.
Plusieurs traitements doivent être faits avant l'application des méthodes itératives de solution.par exemple, les éléments du descripteur doivent être normalisés avec leur plus grand élément.
./norm_dsc_ex fichier
La structure générée est un répertoire qui contient les fichiers suivants :
cnd/x.cnd
le descripteur à échelle de temps continu normalisé, i.e., prêt à l'emploi pour les méthodes de résolution;
....
les fichiers .ftb, .rss, .dct et .inf seront copie dans ce répertoire (cnd).
Différentes méthodes de résolution itératives peuvent être appliquées à ce niveau.
solve_hbf [options] file
Différentes méthodes de résolution itératives peuvent être appliquées à ce niveau, suivant les deux modes avec un vecteur réduit ou avec un vecteur étendu.
solve_cnd_ex [options] file or solve_cnd_sp [options] file
Fichier .vct, un vecteur de probabilités de la solution stationnaire d'un RAS.

Trois des fichiers cités précédemment seront détaillés par la suite :

Tableau des Fonctions

Le tableau des fonctions est décrit dans le fichier (`.tft`). Cette structure définit les expressions arithmétiques correspondant à tous les éléments fonctionnels du modèle :

```
#domain  <le nombre d'automates> <le nombre d'états locaux de chaque automate>
#functions <le nombre d'éléments fonctionnels distincts>
F0 :    <la définition arithmétique du premier élément fonctionnel distinct>;
F1 :    <la définition arithmétique du deuxième élément fonctionnel distinct;
        ........
#rewards
0  0 :   <la récompense associée au premier état du premier automate>
0  1 :   <la récompense associée au deuxième état du premier automate...>
........
1  0 :   <la récompense associée au premier état du deuxième automate>
.....
```

Fonction d'atteignabilité

La fonction d'atteignabilité est une structure d'informations composées par:

- le nombre d'automates et leur taille (nombre d'états locaux de chaque automate);
- la définition arithmétique d'une fonction qui retourne une valeur nulle pour les états globaux non atteignables.
- Le format générique du fichier (.fct) que décrit la fonction d'atteignabilité est:

```
#domain      <le nombre d'automates> <le nombre d'états locaux de chaque automate>
#function    < la définition arithmétique de la fonction d'atteignabilité>;
```

Descripteur Markovien

Le descripteur Markovien est construit à l'instar de son équation :

$$Q = \bigoplus_{i=1}^{N}{}_{g} \; Q_l^{(i)} + \sum_{e \in E} \left(\bigotimes_{i=1}^{N}{}_{g} \; Q_{e^+}^{(i)} + \bigotimes_{i=1}^{N}{}_{g} \; Q_{e^-}^{(i)} \right)$$

Pour plus d'information rapportez-vous à la section qui traite le RAS.

L'ensemble des ces petites matrices est décrit dans un fichier d'extension .des dont le format est le suivant :

#type	<0 pour des modèles à temps discret ou 1 pour modèles à temps continu>
#automata	<le nombre d'automates>
#events	<le nombre d'événements synchronisants>
#sizes	<le nombre d'automates<le nombre d'états locaux de chaque automate>
#locals	<les matrices de transition locale>
#event 0	
#positive	<les matrices avec les taux d'occurrence du premier événement>
#negative	<les matrices avec les taux d'ajustement diagonal du premier événement>
#event	1
……	

E. Conclusion

Dans cette section nous avons fait une petite introduction sur l'historique des méthodes d'évaluation de performance utilisées pour modéliser les comportements des applications informatiques en générale. Nous avons également étudié le formalisme des réseaux d'automates stochastiques, leurs description formelle, les différentes méthodes itératives utilisées pour la résolution du modèle décrit en RAS et nous avons présenté l'outil de simulation qui sera utilisé pour la représentation du modèle et les calcules des mesures de performances. Dans la partie suivante on attaquera la partie modélisation.

Partie V : Modélisation des services Web composite

V. Modélisation des services Web composite

L'étude de performance représente l'utilisation d'un ensemble d'outils et méthodes pour faire une analyse fine du comportement du système. Ce qui permet de prévoir et d'identifier les problèmes qui peuvent se produire et éventuellement de leurs proposer des solutions.

Dans la partie précédente, nous avons présenté les différents formalismes de modélisation qui peuvent être utilisés pour répondre à des questions de performance et de coût. Nous nous intéressons aux réseaux d'automates stochastiques(RAS), par ce que étant les mieux adapté à l'évaluation de performance des services Web [17].

Pour faire la modélisation des services Web composite, une bonne compréhension du fonctionnement de ces services est nécessaire. Dans cette partie, on va faire un rappel sur le fonctionnement des services Web composite. On décrira également le comportement du système à modéliser et leur description en RAS.

A. Principe de fonctionnement des services Web composite

Un service web est dit composé ou composite lorsque son exécution implique des interactions avec d'autres services web afin de faire appel à leurs fonctionnalités. Cette interconnexion des services est vu par l'utilisateur (ou l'application cliente) comme un unique service.

La modélisation de cette interconnexion se fait sous forme d'un processus métier. Ce processus spécifie quels services ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'exception.

Plusieurs implémentations ont été réalisées pour offrir un moteur d'exécution de ces processus. Dans le cadre de ce mémoire, nous avons utilisé le moteur d'orchestration ActiveBPEL [21] pour déployer un exemple de processus métiers décrit en utilisant BPEL [9].

Le déploiement d'un processus métier sur le moteur d'orchestration se fait sous forme d'un service Web (ce service est le service qui représente le service Web composite). Ce service est décrit par un contrat WSDL [7], qui est une description basée sur le langage XML, qui indique quelles sont les opérations disponibles?, comment on y accède (adresse, protocole, …) ?, quel est le format de messages échangés entre le client et le moteur, pour invoquer le service et pour interpréter le résultat.

Une fois que le service Web composite est déployé, il faut mettre à la disposition des utilisateurs des mécanismes pour que ces derniers puissent utiliser ces services. Cette phase s'appelle la publication [8].

La publication des services Web nécessite une sorte d'annuaire dans lequel ils seront référencés.

Maintenant, le client ou l'utilisateur peut effectuer des recherches, et l'annuaire va lui répondre en lui renvoyant un ensemble de liens qui correspondent à ses critères. Le client choisi un parmi eux. Donc le client connait l'adresse du serveur et ce que peut faire le service, mais il ne sait pas comment ?

Pour pouvoir connaitre comment il peut invoquer ce service, le client et le moteur vont participer à un dialogue en quatre étapes : deux requêtes et deux réponses.

Dans la première phase, l'utilisateur va demander au moteur (serveur) quelles sont les méthodes qu'il expose et quels sont les paramètres qui doivent être passés à chacune de ces méthodes. Cette requête peut être réalisée sous forme d'une requête Get HTTP.

Le serveur répond en envoyant son contrat WSDL qui décrit les fonctions exposées et comment les clients peuvent les utiliser.

A cet instant, l'utilisateur sait les noms des méthodes exposées par le serveur, et leurs paramètres, mais il ne sait pas comment va-t-il indiquer au serveur qu'il veut appeler la fonction X par exemple.

Pour construire et formater les messages qui seront échangés avec le serveur le client va utiliser le protocole SOAP [2]. Ce protocole va permettre au serveur, une fois récupéré les messages envoyé par les clients, d'analyser les contenues de ses requêtes, de les exécuter et de renvoyer les résultats.

Donc ce processus de fonctionnement peut être résumé comme suit:

Une fois l'utilisateur a pu retrouver le service qu'il recherche, il peut récupérer toutes les informations qu'il lui faut pour développer une application cliente en utilisant un langage de programmation de son choix.

Ensuite l'application cliente peut invoquer les fonctions exposées. Différents styles d'échange peuvent être utilisés :

One-way (l'interaction à sens unique): Le client envoie une requête SOAP au serveur, mais n'attend pas de réponse en retour ;

Request-response (la requête/réponse) : Le client envoie une requête SOAP au serveur et attend une réponse en retour ;

Solicit-response (demande de réponse) : Cette fois, c'est le serveur qui envoie une requête SOAP au client et qui attend une réponse en retour ;

Notification (notification) : Le fournisseur du service envoie une requête SOAP au client.

Mais comme la plupart des services Web communiquent via HTTP, qui est un protocole bidirectionnel synchrone, formé par un couple indissociable requête/réponse http, nous allons nous limiter à ce type de style d'échange.

Pour conclure, l'interaction d'une application cliente avec le serveur (moteur d'orchestration) peut être décrite par le schéma suivant :

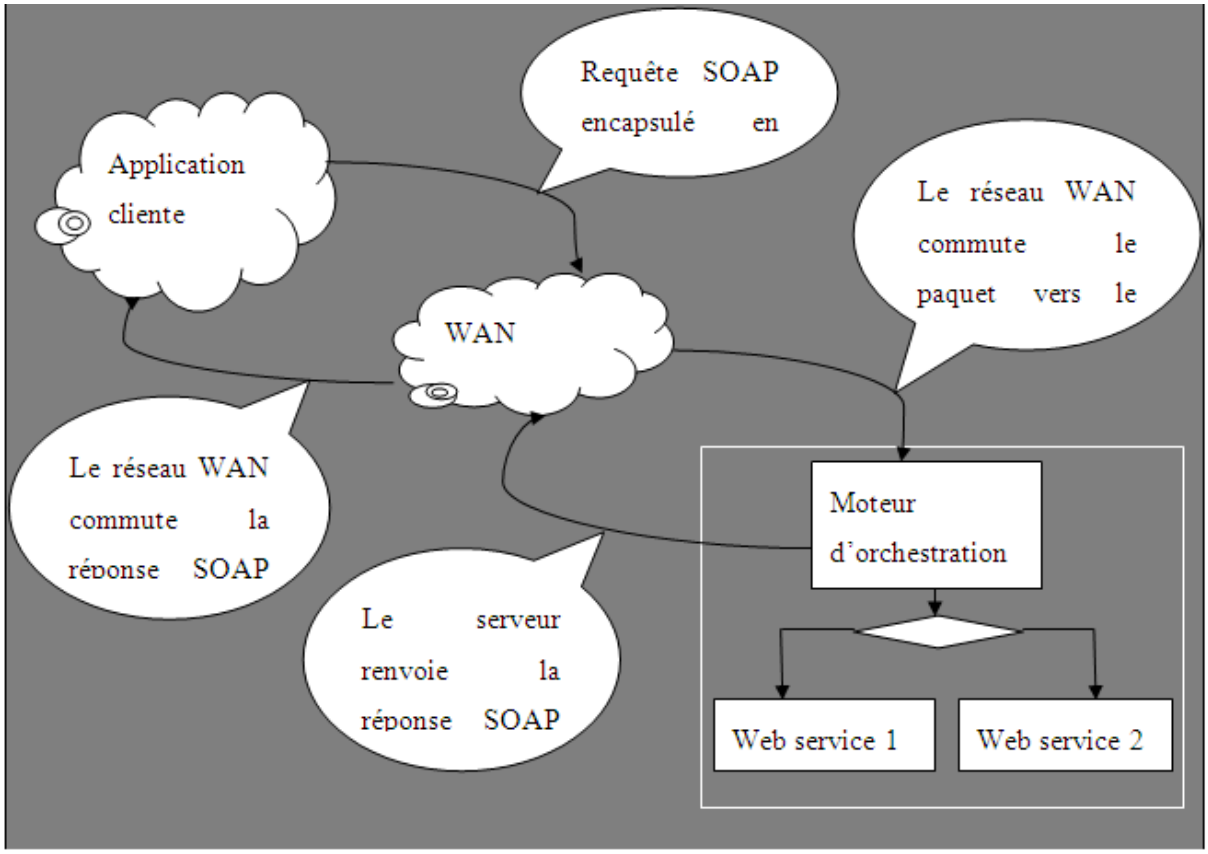


Figure 22 : Architecture générale à modéliser.

B. Modèle d'essai

Nous pouvons envisager que nous sommes dans un cas où une entreprise possède deux sites (A et B) connectés à l'aide d'une liaison série. Il arrive que cette entreprise ait besoin de développer une nouvelle application dans le site A pour répondre à des nouvelles exigences. Lors de la conception, le concepteur a remarqué qu'une partie de l'application qu'il doit développer tourne déjà sous forme d'un service Web sur le site B. Le concepteur sait que grâce aux technologies des services WEB, il est devenu possible de faire une composition des ces derniers en utilisant un langage d'orchestration. Il décide donc, de concevoir et de développer un nouveau service mais en ne considérant que les besoins qui ne sont pas pris en compte lors du développement du service Web B. Ensuite, il crée un processus métier qu'il déploie dans un moteur d'orchestration sous forme d'un service Web composite.

Enfin, la société décide de faire une étude de performance de sa nouvelle architecture pour assurer le bon fonctionnent du système et pour assurer encore que les exigences de performance sont toujours respectées.

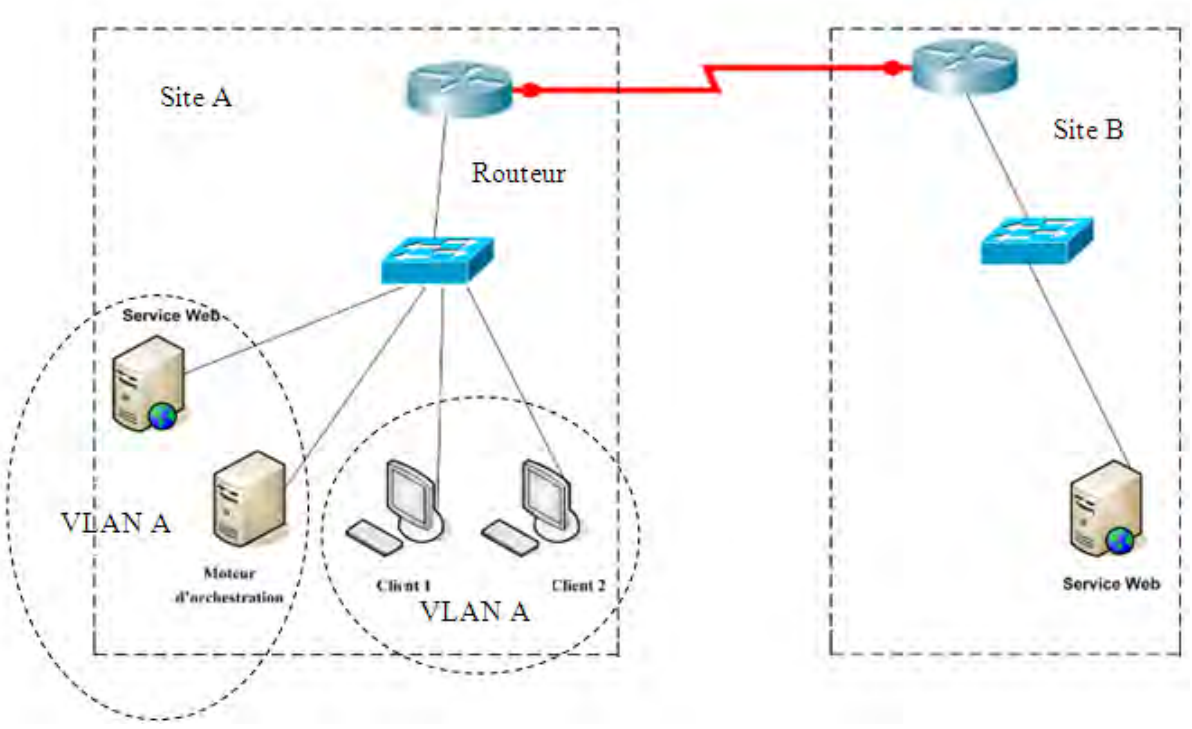


Figure 23 : Modèle d'essai

C. Les hypothèses

Dans notre modèle d'essai nous supposons que toutes les configurations sont statiques. Nous supposons également que les messages des requêtes des clients ont une taille de 12 koctets, et que les réponses envoyées par le moteur d'orchestration est de 40 koctets. Les requêtes envoyées par le moteur et leurs réponses envoyées par les services Web unitaire ont la même taille (8 koctets). Pour calculer les paramètres d'entrées nous supposons également que le Moteur (CPU + Disk) peut traiter 1000 requêtes/s pour le CPU et 500 requêtes/s pour le Disk. Les serveurs qui traitent les requêtes du moteur peuvent traiter 600 requêtes/s pour le CPU et 500 requêtes/s pour le Disk.

Paramètres du modèle

Les paramètres du modèle sont les suivants :

- Taux d'arrivées des requêtes des clients ;
- Taille des messages envoyés et reçus par chaque nœud du réseau ;
- Taux de services des différents nœuds du réseau ;
- Les probabilités de routages des messages qui doivent être envoyés par chaque nœud ;
- Les capacités des buffers pour chacun des routeurs ;
- Les débits des différentes liaisons.

Le taux de services des routeurs est calculé en fonction du temps de transmission T_t selon la formule suivante :

$$\mu = \frac{1}{Tt}$$

Le temps de transmission T_t est le délai qui s'écoule entre le début et la fin de la transmission d'un message sur une ligne, ce temps est donc égale au rapport entre la longueur du message et le débit de la ligne. Ici nous utilisons des interfaces FastEthernet et des interfaces série dont les débits sont respectivement 100Mbits/s et 1544 kbit/s.

Donc, les taux des services du routeur A peuvent être calculés en fonction de la taille du message selon la formule suivante :

Pour les requêtes des clients

$\mu_{ra\_fast} = 1/(8 * (\frac{(0...1460 + entete\ TCP + enteteIP + enteteethernet + tag\ (vlan))}{debit\ de\ l'interface fastEthernet}))$ Pour chaque trame Ethernet.

$$\mu_{ra\_req\_client} = 1/(8 * \left(\frac{(1000 + 20 + 20 + 18 + 4)}{100000000}\right)) = 11770$$

Pour les requêtes envoyées par le moteur

vers le service WEB B (site distant)

$\mu_{ra_{ssr}\_req} = 1/(8 * (\frac{(0...1460 + entete\ TCP + enteteIP + enteteHDLC)}{debit\ de\ la\ ligne}))$ Pour chaque paquet HDLC.

$$\mu_{ra_{ssr}\_req} = 1/(\left(\frac{8*(800+20+20+6)}{1544000}\right)) = 228$$

Pour les réponses envoyées par le serveur distant

$$\mu_{rafast\_res} = 1/\left(\frac{8 * (800 + 20 + 20 + 18 + 4)}{100000000}\right) = 14501$$

Pour les réponses envoyées par le moteur

$$\mu_{ra\_fast\_res} = 1/(8 * \left(\frac{(1400 + 20 + 20 + 18 + 4)}{100000000}\right)) = 8549$$

Pour le moteur d'orchestration

Vers les différents services WEB

$$\mu_{mtrreq} = 1/(\left(\frac{1}{1000} + \frac{1}{500}\right) = 334$$

Pour le routeur RB :

Le taux de traitement de requêtes envoyées par le moteur est :

$$\mu_{rbfast\_req} = 1/(8 * \left(\frac{(800 + 20 + 20 + 18)}{100000000}\right)) = 14568$$

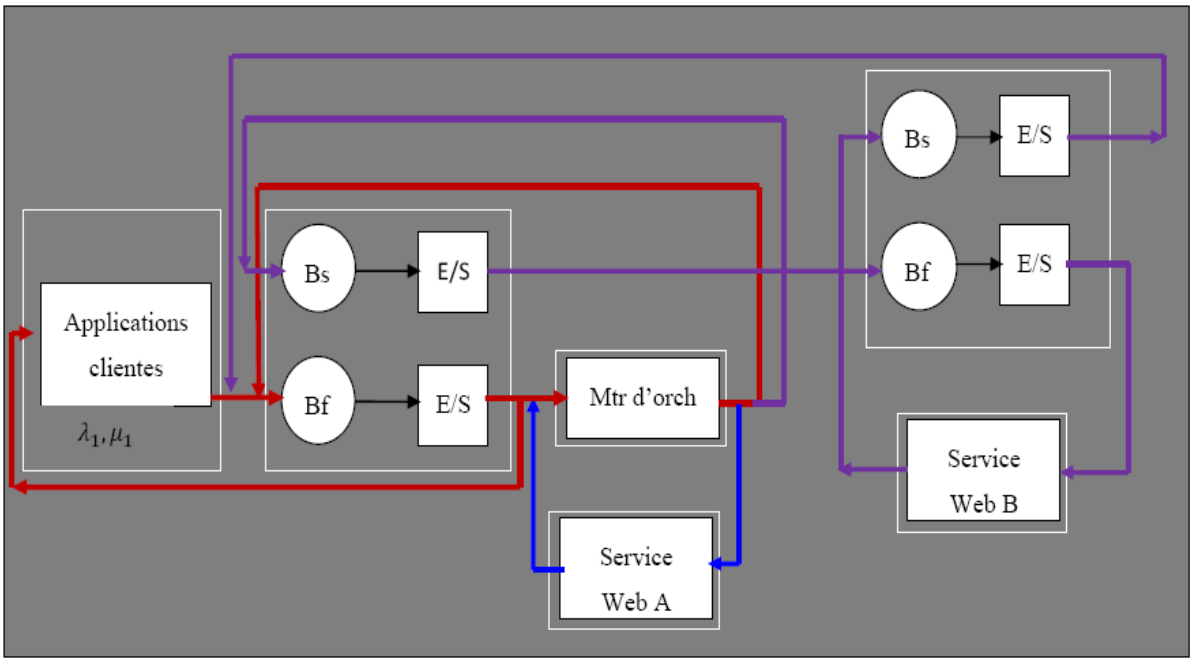
Le taux de traitement de réponses envoyées par le serveur distant est :

$$\mu_{ra_{ser\_res}} = 1/(8 * \left(\frac{(800+20+20+6)}{1544000}\right)) = 228$$

Pour les serveurs qui exécutent les requêtes du moteur

$$\mu_{SWA} = \mu_{SWB} = 1/(\frac{1}{1000} + \frac{1}{500}) = 334$$

D. Modélisation



Ce système peut être modélisé par un ensemble des automates synchronisés en utilisant des éléments fonctionnels.

Comme les réseaux d'automates stochastiques (RAS) sont basés sur l'idée que le système modélisé peut se décomposer sous la forme de plusieurs sous-systèmes, chacun de ces sous-systèmes peut évoluer indépendamment et peut interagir avec les autres en se synchronisant ou en faisant varier la valeur de ses transitions en fonction des états des autres sous-systèmes, Nous pouvons profiter de cette caractéristique de RAS pour découper notre système en deux parties :

Dans la première partie, le moteur d'orchestration et son interaction avec les deux services WEB A et B sont vus par le client comme une boite noire. Donc on va se limiter à la modélisation des interactions des clients avec le moteur.

La deuxième partie traite les interactions entre le moteur d'orchestration et les différents services WEB unitaires.

Mais avant de décrire les composants de la première partie nous allons donner des définitions des différents composants apparus dans le **Modèle d'essai** et leurs fonctionnements**.**

Routeur : Le principe de fonctionnement d'un routeur standard est relativement simple : il reçoit un paquet sur une interface. Il consulte sa table de routage afin de déterminer l'interface

de sortie à utiliser. Il ajoute le paquet à transmettre dans la file d'attente de cette interface. Si la file est pleine, il supprime le paquet.

Modélisation des interactions liées aux requêtes et aux réponses des requêtes des clients

La première partie peut être résumée par le schéma suivant :

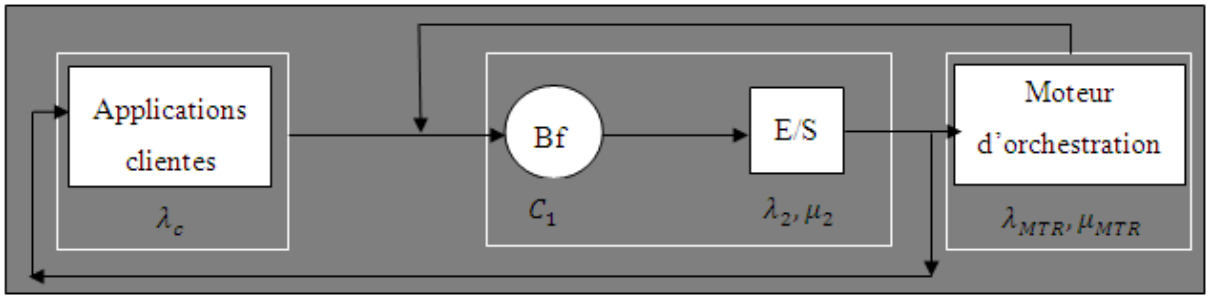


Figure 24 : La première partie du modèle**.**

L'arrivée des requêtes des clients et leurs commutations au niveau de ***RouteurA*** sont modélisées par une automate ***RA1*** dont les nombres d'états correspondent à la taille du buffer de l'interface FastEthernet.

Les différents événements pouvant avoir lieu:

- ***arr_c_ra*** l'arrivée des requêtes des clients au niveau des ***RouteurA***. Cet événement est un événement local. Sont taux d'occurrence est égale à λ_{c_ra}.
- ***comm_c_mrt*** La commutation des requêtes de clients vers le moteur d'orchestration : La synchronisation entre le Moteur et le routeur, en ce qui concerne l'envoie des requêtes des clients au Moteur, est assuré par un liste des triplets de synchronisation. Il faut remarquer ici que le taux d'occurrence associé à ces événements n'est pas le taux de service de l'interface FastEthernet du routeur A μ_{ra_fast}, mais ce taux de service multiplié par la probabilité de routage correspondante $p_{c,mtr}$.

La fonction f qui apparaît dans les triplets de synchronisation (triplets des arcs supérieurs des automates **RA1**) permet de faire une restriction des arrivés selon la capacité de la file

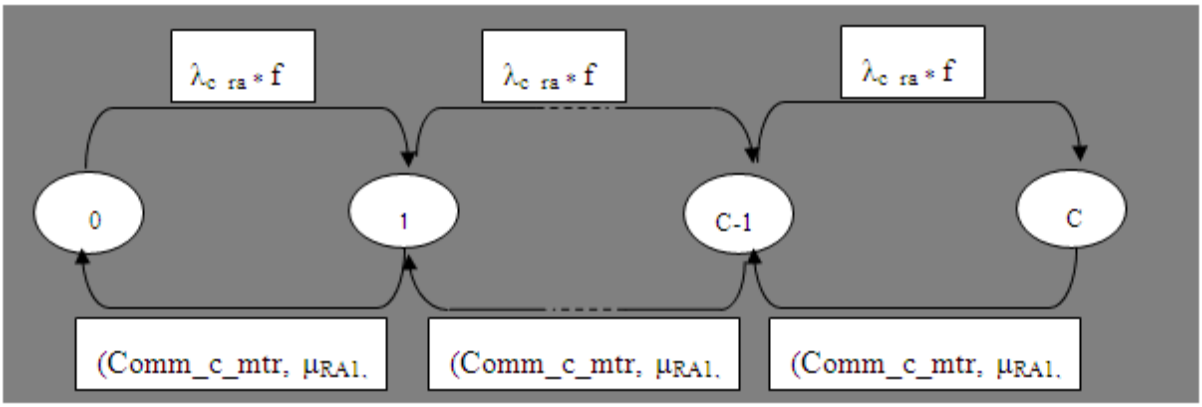
d'attente C de l'interface FastEthernet $RouteurA$ (contrôle de flux). Cette fonction peut être définie comme suit :

$$f = \delta(Nbre\ des\ \text{réponses envoyées}\ par\ le\ moteur + nbre\ de\ \text{requêtes envoyées}\ par\ les\ clients + Nbre\ des\ \text{réponses envoyées}\ par\ le\ serveur\ distant\ < C)$$

Où $\delta(b)$ est une fonction booléen qui renvoie 1 si la condition b est vraie et 0 dans le cas contraire. Ici la condition qu'on doit vérifier c'est de regarder si le buffer est plein ou pas. Si la condition n'est pas vérifiée (buffer est pleine, f=0) le routeur va rester dans le même état.

$$p_{c,mtr} = nbre\ de\ \text{requêtes envoyées}\ par\ les\ clients/\text{nombre des paquets dans le buffer}$$

$$\mu_{RA1} = \left(\mu_{ra_{fast}} * p_{c,mtr}\right) * \left(\mu_{ra_{fast}} * p_{c,mtr} \leq \mu_{NIR}\right) + \left(\mu_{ra_{fast}} * p_{c,mtr} > \mu_{NIR}\right) * \mu_{NIR}$$

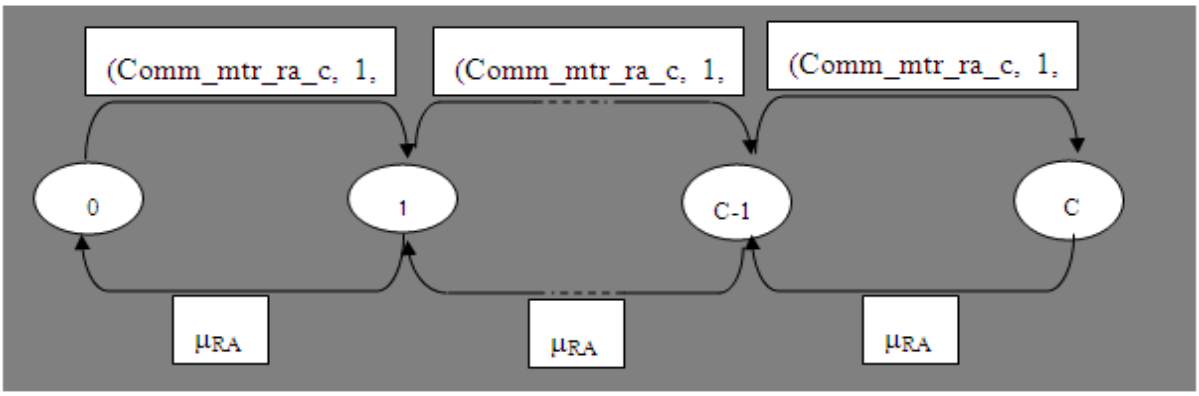


Lorsque le routeur reçoit les réponses envoyées par le Moteur d'orchestration, il les place dans le buffer de l'interface FastEthernet dans le but de les commuter vers les clients. Ce processus peut être modélisé par un automate $RA2$ dont le nombre d'états correspond à la taille du buffer de l'interface FastEthernet.

Les différents événements pouvant avoir lieu:

- $comm\_mtr\_ra\_c$: l'arrivé des réponses envoyées par le moteur. L'automate maître de cet événement est l'automate modélisant le moteur d'orchestration. Le ta ux d'occurrence de l'automate **RA2** est donc 1.

- $comm\_ra\_c$: la commutation des réponses envoyées par le moteur vers les clients. Cet événement modélise le départ des réponses vers l'extérieur (les clients), donc cet événement est un événement local.



Le taux d'occurrence de l'événement $comm\_ra\_c$ n'est pas le taux de service de l'interface FastEthernet du r outeur A μ_{ra_fast}, mais ce taux de service multiplié par la probabilité de routage correspondante $p_{ra,c}$.

$p_{ra,c} = nbre\ de$ réponses envoyées $par\ le\ moteur$
/ nombre des paquets dans le buffer de l'interface FastEthernet

$$\mu_{RA2} = \mu_{ra\_fast} * p_{ra,c}$$

Modélisation des interactions liées aux requêtes envoyées par le moteur vers le service Web distant et ses réponses

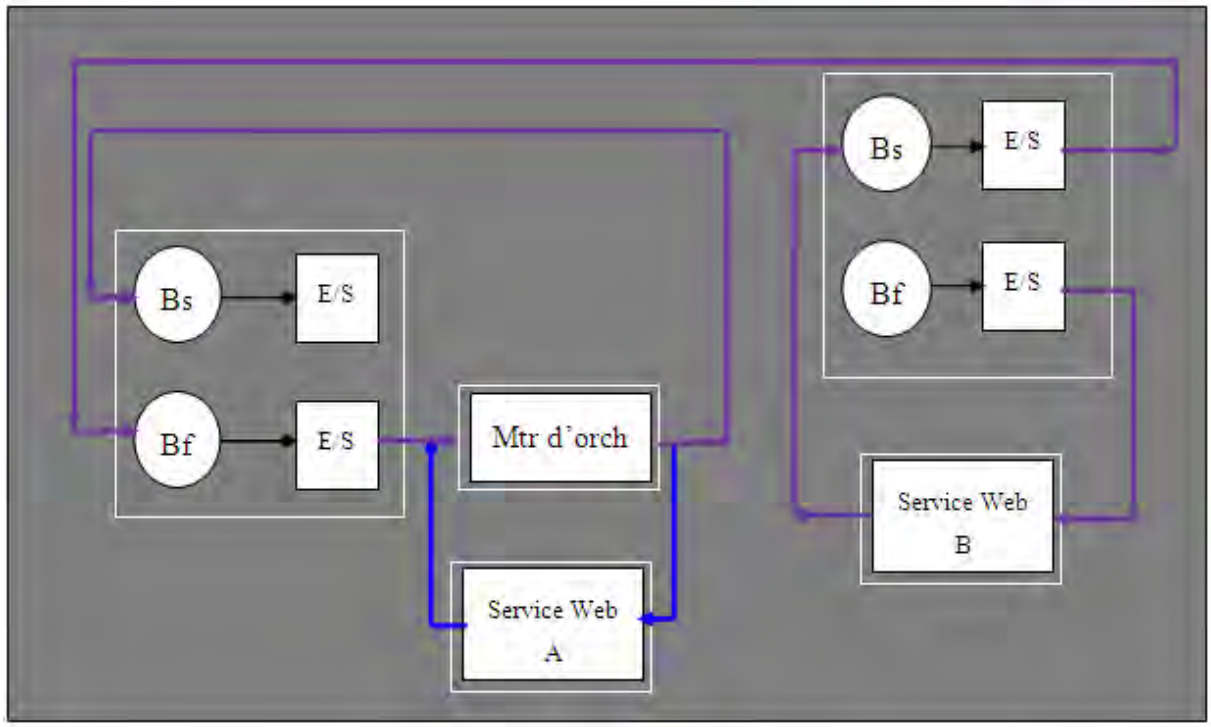


Figure 25 : Deuxième partie du modèle.

Lorsque les requêtes des clients arrivent au niveau du moteur d'orchestration, ce dernier doit invoquer l'un de deux services Web pour pouvoir répondre à ces requêtes. Donc le moteur va interroger l'un de ces services Web, en lui envoyant un message SOAP. Pour le service WEB B les requêtes doivent traverser les deux routeurs pour arriver au serveur distant et les réponses vont suivre le même chemin mais en retour. Mais avant de décrire ce processus pour le service WEB B en RAS on va donner une description de deux automates modélisant le Moteur et le service WEB A.

Moteur d'orchestration

Le moteur d'orchestration peut être modélisé par une automate à cinq états : le moteur peut être soit :

- Dans l'état ***prés*** *(*il attend les requêtes des clients qui peuvent arriver à tous moment) ;
- dans l'état ***examiné*** *(*il analyse les requêtes reçues, pour pouvoir extraire les paramètres des méthodes exposées par les différents Web Services à invoqués*) ;*

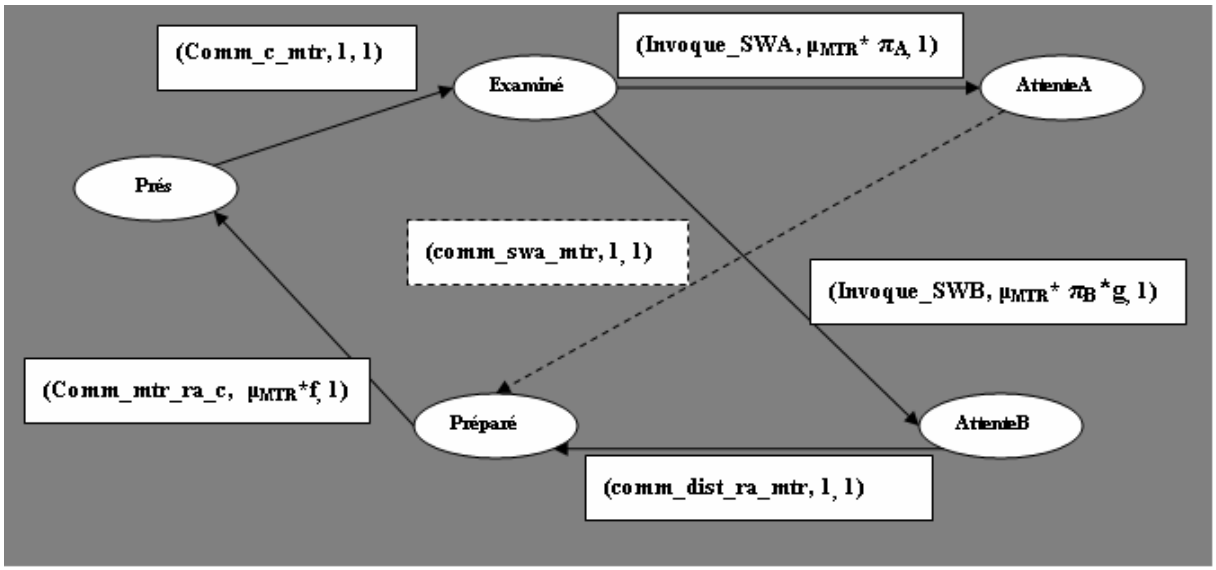
- dans l'état $attente$**A** *(*le moteur attend les réponses du service WEB A qu'il a invoqué) ;
- dans l'état $attente$**B** *(*le moteur attend les réponses des services WEB B qu'il a invoqué) ;
- dans l'état $préparer$ *(*construire les réponses des requêtes des clients et des envoyées*)*. On nome cette automate ***MTR.***

Les différents événements pouvant avoir lieu sont :

- $comm\_c\_mrt$: cet événement modélise la commutation des requêtes des clients vers le moteur, ce qui veut dire que cet automate sera esclave de cet événement. Le taux d'occurrence pour cet événement est donc 1. Cet événement permet de passer de l'état $prés$ vers l'état $examiné$.
- $invoque\_sw$**A** : cet événement modélise l'invocation du service Web A. Cet événement permet de passer de l'état $examiné$ vers l'état $attente$A. Son t aux d'occurrence est μ_{MTR} multiplié par la probabilité de routage π_A, qui correspond à la proportion des requêtes qui seront envoyées vers le service WEB A (le moteur d'orchestration, le serveur A ont les mêmes caractéristiques).
- $invoque\_sw$**B** : cet événement modélise l'invocation du service Web B. Cet événement permet de passer de l'état $examiné$ vers l'état $attente$**B**. Son t aux d'occurrence est μ_{MTR} multiplié par la probabilité de routage π_B, qui correspond à la proportion des requêtes qui sera envoyés vers le service WEB B. La fonction g qui apparaît dans le triplet de cet événement permet de faire une restriction des arrivées selon la capacité de la file d'attente Cs de l'interface série au niveau de $RouteurA$ (voir la figure).

$$g = \delta(Nbre\ des\ \text{requêtes envoyées}\ par\ le\ moteur\ dans\ cette\ file\ d'attente < Cs)$$

- ***comm_dist_ra_mtr*** : cet événement modélise l'arrivée des réponses du service WEB distant. Son taux d'occurrence est égal à 1 (l'automate maître de cet événement est **RA3**).
- ***comm_dist_ra_mtr*** au niveau du routeur A modélise la commutation des réponses envoyées par le serveur distant vers le moteur d'orchestration. Cet événement permet de passer de l'état ***attente*****B** vers l'état ***préparer***.
- **Comm_swa_mtr :** cet événement modélise l'arrivée des réponses du service WEB A. Son taux d'occurrence est égal à 1 (l'automate maître de cet événement est **SWA**). Cet événement permet de passer de l'état ***attente*****A** vers l'état ***préparer***.
- ***comm_mtr_ra_c*** : cet événement modélise l'envoie des réponses vers les clients. Son taux d'occurrence est μ_{MTR}. Cet événement permet de passer de l'état ***préparer*** vers l'état ***prés***. La fonction f qui apparaît dans le triplet de cet événement permet de faire une restriction des arrivées selon la capacité de la file d'attente ***Cs*** de l'interface FastEthernet au niveau de ***RouteurA*** (contrôle de flux).

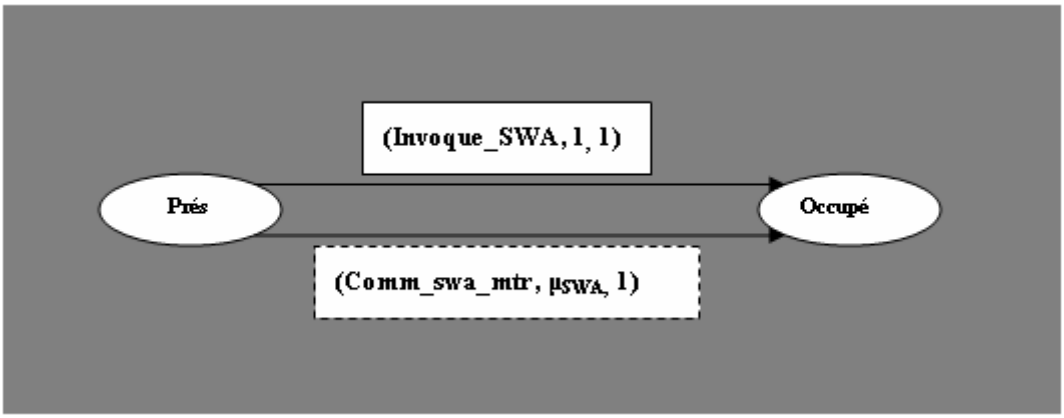


Service WEB A

Le service WEB A tourne sur un serveur qui existe sur le même VLAN que celui du moteur. Ce serveur est modélisé par une automate ***SWA*** à deux états : soit dans l'état ***prés*** (il attend les requêtes des Moteur qui peuvent arriver à tous moment), soit dans l'état ***occupé*** (il analyse les requêtes, faire les traitements demandés et construire les réponses).

Les différents événements pouvant avoir lieu sont :

- ***invoque_swA*** cet événement modélise l'invocation des deux services Web par le moteur, ce qui veut dire que cet automate est esclave de cet événement. Cet événement a un taux d'occurrence égal à 1. Cet événement permet de passer de l'état ***prés*** vers l'état ***occupé***.
- **Comm_swa_mtr :** cet événement modélise l'envoie des réponses par le service WEB A et en même temps modélise l'arrivé de ces réponses au niveau du Moteur. Cet automate est l'automate maître de cet événement synchronisant. Cet événement permet de passer de l'état ***occupé*** vers l'état ***prés*** . le taux d'occurrence de cet événement est μ_{SWA}.



Processus d'invocation du service Web Distant

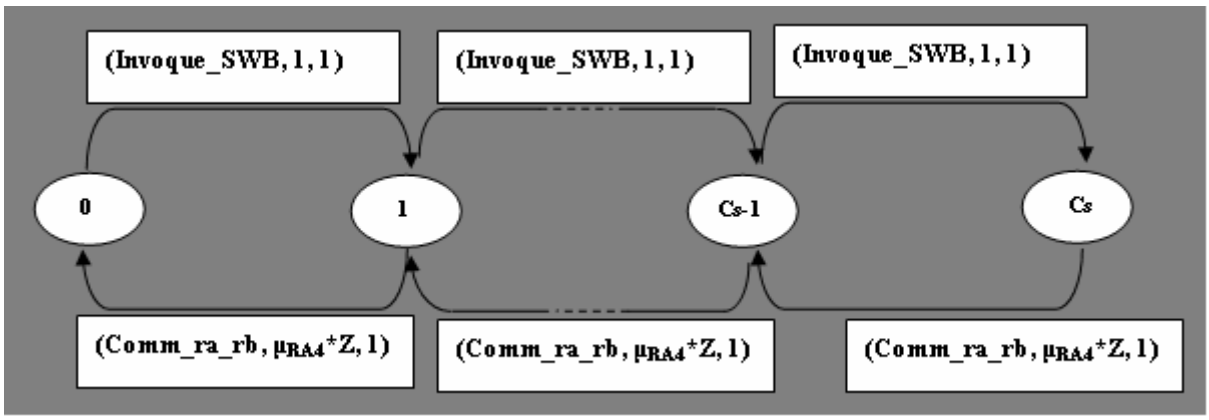
Les requêtes envoyées vers le serveur distant passent par le ***routeurA*** qui les place dans le buffer de l'interface série dans le but de les commuter vers le ***routeurB***. Lorsque ces requêtes arrivent au niveau du ***routeurB*** ce dernier les place dans le buffer de l'interface

FastEthernet pour qu'il envoie vers le serveur. Le **serveur** récupère les requêtes, fait les traitements demandés et envoie les réponses vers le routeur B. le ***routeurB*** place les réponses dans le buffer de l'interface série Lorsque ces requêtes arrivent au niveau du ***routeurA*** ce dernier les place dans le buffer de l'interface FastEthernet pour qu'il l'envoie vers le Moteur d'orchestration.

L'arrivée des requêtes et l'arrivée des réponses au niveau du ***routeurA*** sont modélisées par deux automates **RA3** et **RA4**. **RA4** modélise l'insertion des requêtes dans le buffer de l'interface série et leurs commutations, alors que **RA3** modélise le même comportement mais avec l'interface FastEthernet.

Le nombre d'états de l'automate **RA4** correspond à la taille du buffer de l'interface série. Les différents événements pouvant avoir lieu dans cet automate sont :

- ***invoque_swB*** : cet événement modélise l'invocation du service Web B par le moteur, ce qui veut dire que cet automate est esclave de cet événement. Donc cet événement a un taux d'occurrence égal à 1.
- ***comm_ra_rb*** : la commutation des requêtes envoyées par le moteur vers les le service Web distant.



Le taux d'occurrence de l'événement ***comm_ra_rb*** est le taux de service de l'interface série du routeur A μ_{ra_serial}. La fonction **Z** qui apparaît dans la liste des triplets des arcs inférieurs de

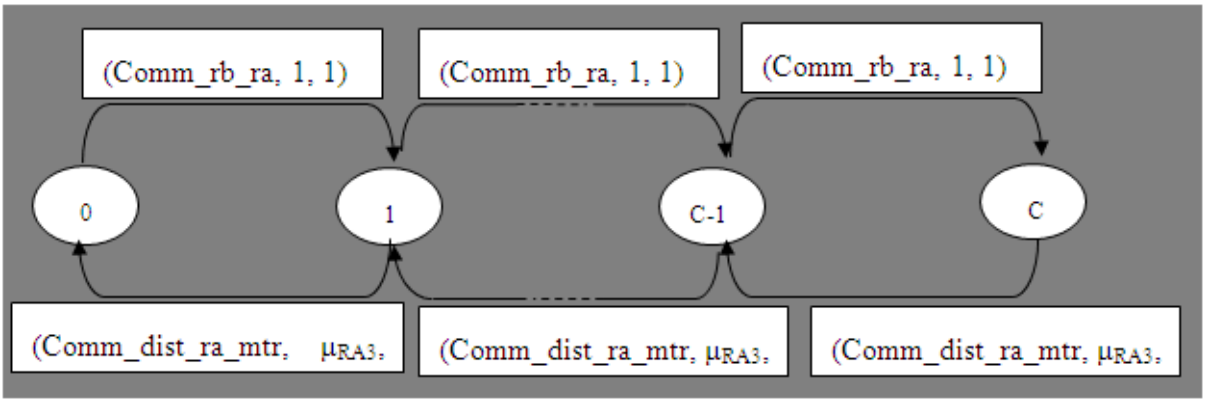
cet automate permet de faire une restriction des arrivées selon la capacité de la file d'attente **C** de l'interface FastEthernet du ***routeurB***.

$\mu_{RA4} = \mu_{ra\_serial}$

Z = **δ** (Nbre des requêtes dans l'interface FastEthernet du routeur B < C).

Le nombre d'états de l'automate **RA3** correspond à la taille du buffer de l'interface FastEthernet du routeur A. Les différents événements pouvant avoir lieu dans cet automate sont :

- ***comm_rb_ra*** : cet événement modélise l'arrivée des réponses du serveur distant au niveau du ***routeurA*.** Le taux d'occurrence de cet événement est 1.
- ***comm_dist_ra_mtr*** : cet événement synchronise les deux Automates MTR et RA3. Cet événement permet la commutation des réponses du serveur distant vers le Moteur (l'automate maître est l'automate **RA3)**, mais il modélise également l'arrivée de ces réponses au niveau du Moteur d'orchestration dans l'automate ***MTR***.



Le taux d'occurrence de l'événement ***comm_dist_ra_mtr*** :

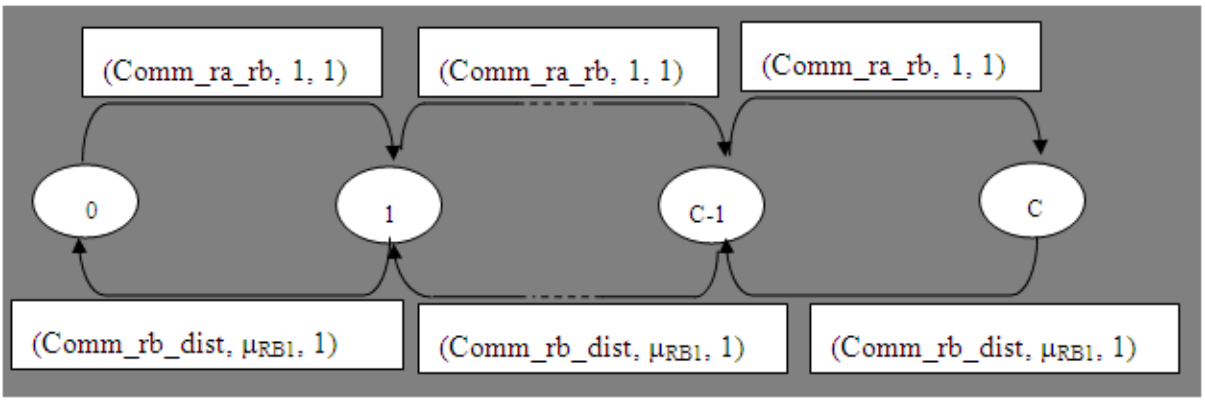
$\mu_{RA3} = ((\mu_{ra\_fast} * p_{rb\_ra\_mtr}) * (\mu_{ra\_fast} * p_{rb\_ra\_mtr} < \mu_{MTR})) + ((\mu_{ra\_fast} * p_{rb\_ra\_mtr} > \mu_{MTR}) * \mu_{MTR})$

L'arrivée des requêtes et l'arrivée des réponses au niveau de ***routeurB*** sont modélisées par deux automates **RB1** et **RB2**. **RB1** modélise l'insertion des requêtes dans le buffer de l'interface FastEthernet et leurs commutations, alors que **RB2** modélise le même comportement mais avec l'interface série.

Le nombre d'états de l'automate **RB1** correspond à la taille du buffer de l'interface FastEthernet. Les différents événements pouvant avoir lieu dans cet automate sont :

- ***comm_ra_rb*** : cet événement modélise la commutation des requêtes envoyées par le routeur A vers les le routeur B. L'automate **RA4** est l'automate maître de cet événement. Le taux d'occurrence de cet événement est donc 1.
- ***comm_rb_dist*** : cet événement modélise la commutation de ces r equêtes vers les le serveur distant. Son taux d'occurrence peut être calculé par la procédure suivante:

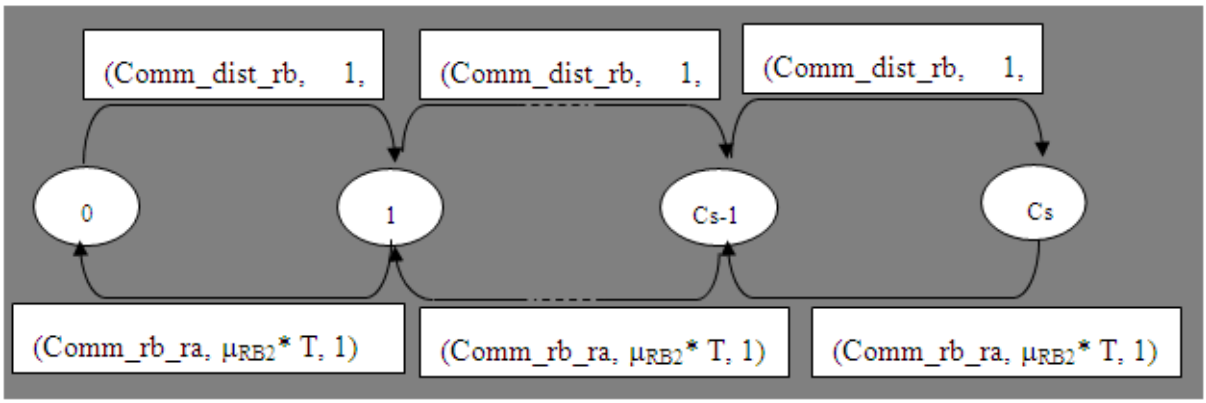
$$\mu_{RB1} = \left(\mu_{rb\_fast}\right) * \left(\mu_{ra_{fast}} \leq \mu_{SRV\_dist}\right) + \left(\mu_{rb\_fast} > \mu_{Srv\_dist}\right) * \mu_{Srv\_dist}$$



Le nombre d'états de l'automate **RB2** correspond à la taille du buffer de l'interface série. Les différents événements pouvant avoir lieu dans cet automate sont :

- ***comm_dist_rb*** cet événement modélise l'arrivée des réponses du s erveur distant au niveau du ***routeurB.***
- ***comm_rb_ra*** : dans l'automate ***RA3*** cet événement modélise l'arrivée des réponses du serveur distant au niveau du ***routeurA***, mais il modélise également, dans cet automate, la commutation des réponses reçues du serveur distant vers le routeur A. Ce qui veut dire que cet automate est l'automate maître. Le taux d'occurrence de cet automate est$\mu_{RB2} = \mu_{rb_serial}$. La fonction **T** qui apparaît dans la liste des triplets des arcs inférieurs de cette automate permet de faire une restriction des arrivées selon la capacité de la file d'attente **C** de l'interface FastEthernet du ***routeurA.***

T = δ (Nbre des messages dans l'interface FastEthernet du routeur A < C).



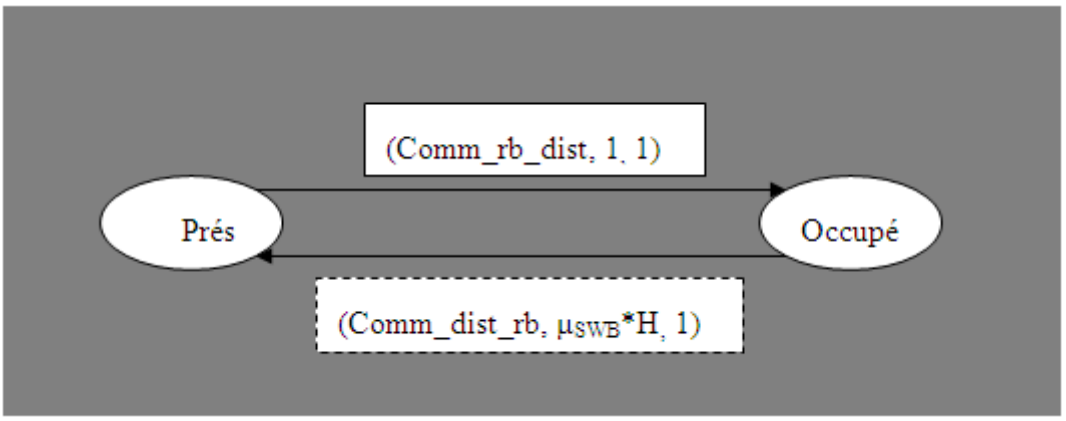
Service WEB B

Le service WEB B tourne sur un serveur qui existe sur le site distant. Ce serveur est modélisé par une automate ***SWB*** à deux états : soit dans l'état ***prés*** (il attend les requêtes du Moteur qui peuvent arriver à tous moment), soit dans l'état ***occupé*** (il analyse les requêtes, faire les traitements demandés et construire les réponses).

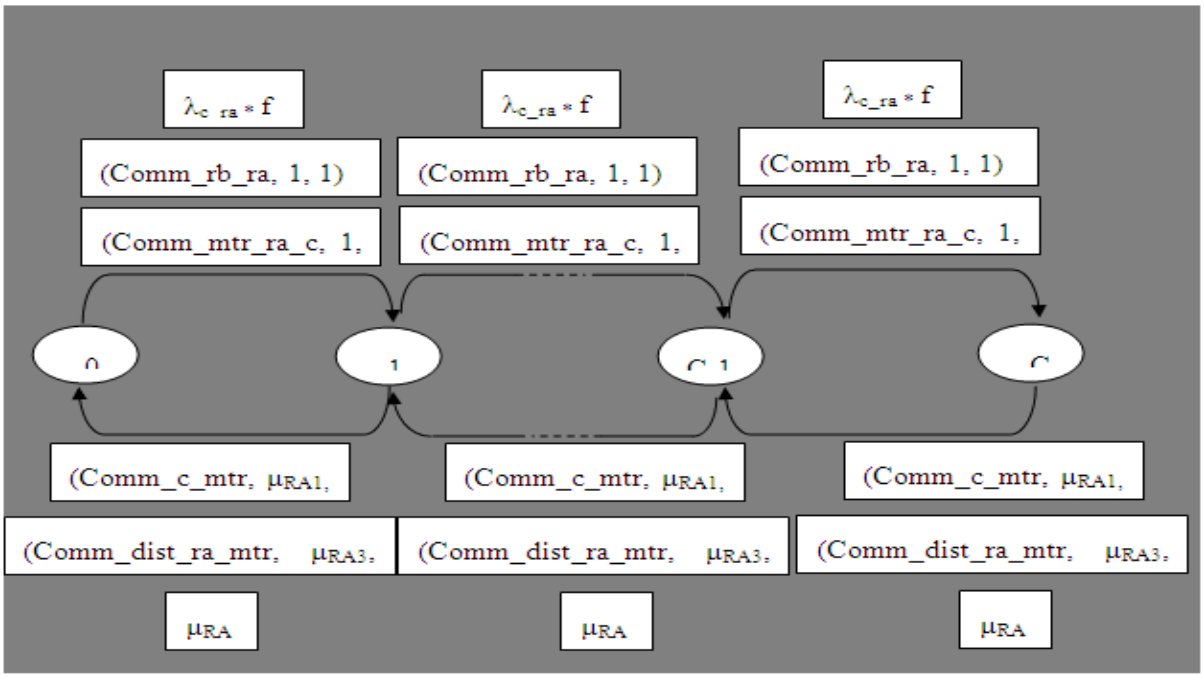
Les différents événements pouvant avoir lieu sont :

- ***comm_rb_dist*** : cet événement modélise la commutation de ces r equêtes p ar le routeur B vers le serveur distant. Mais il modélise également l'arrivée de ces requêtes au niveau de ce serveur. Son taux d'occurrence est donc 1.
- ***comm_dist_rb*** : cet événement modélise l'envoie des réponses par le serveur distant vers le moteur d'orchestration. Sont taux d'occurrence est μ_{SWB}. $\mu_{RB2} = \mu_{rb\_serial}$. La fonction **H** qui apparaît dans la liste des triplets des arcs inférieurs de cette automate permet de faire une restriction des arrivées selon la capacité de la file d'attente **Cs** de l'interface série du ***routeurB***.

H = δ (Nbre des messages dans l'interface série du routeur B < Cs).



Pour conclure, les trois automates RA1, RA2 et RA3 peuvent être regroupées pour former un seul automate (on nome cet automate RA1 : figure 14). Chacun des arcs supérieurs de cet automate est étiqueté par trois événements : l'arrivé des requêtes des clients, l'arrivé des réponses du serveur distant et l'arrivé des réponses envoyées par le moteur d'orchestration. Ces trois événements ont les mêmes taux d'occurrences décrits précédemment. Même chose pour les événements des arcs inferieurs, sauf que cette fois ci on va considérer que les probabilités de routage sont égaux (1/3).



E. Les critères de performances

Les critères de performances qu'on cherche à évaluer dans cette modélisation sont en fonction du nombre de requêtes des clients et qui sont le nombre moyen de requêtes dans chaque buffer de deux routeurs, le taux d'utilisation de deux routeurs ainsi que pour le moteur d'orchestration, et le temps de réponse moyens du système. Le programme suivant montre la traduction du modèle décrit précédemment en format interne exécutable par PEPS avec comme nombre d'arrivé 3000 requêtes/s.

```
identifiers

 lamda_c_ra =3000;
 mu_intf_fast_ra1 = 11770;
 mu_intf_fast_ra2 = 8549;
 mu_intf_fast_ra3 = 14501;
 mu_inf_ser_ra4 = 228;
 mu_inf_fast_rb= 14568;
 mu_inf_ser_rb = 228;
 mu_MTR = 334;
 mu_SWB = 334;
 mu_SWA = 334;
 pu_SWA = 60/100;
 pu_SWB = 40/100;
 C = 14;
 CS= 25;
 C1 = [0..14];
 CS1 = [0..25];
 b= st RA1;
 f = b < C;
 lamda_RA1 = lamda_c_ra * f;
 prob_c_mtr = 1/3;
 muRA1 = ((mu_intf_fast_ra1) * (mu_intf_fast_ra1 <=
mu_MTR)) + ((mu_intf_fast_ra1  > mu_MTR)*mu_MTR);

 g = (st RA2 < CS );
 mu_invoque_SWB = mu_MTR * g;
 mu_reponse_MTR = mu_MTR * f;
 z = (st RB1 < C);
 muRA4= mu_inf_ser_ra4 * z;
 muRB1 = (mu_inf_fast_rb*(mu_inf_fast_rb <=mu_SWB ))
+((mu_inf_fast_rb >    mu_SWB )* mu_SWB);
 h = (st RB2 < CS);
 mu_dist_rb = mu_SWB * h;



 muRB2 = mu_inf_ser_rb * f;

 muRA3 = ((mu_intf_fast_ra3 )*(mu_intf_fast_ra3 <=
mu_MTR)) + ((mu_intf_fast_ra3 *prob_c_mtr >
mu_MTR)*mu_MTR);

 prob_mtr_c =1/3;
 muRA2 = (mu_intf_fast_ra2);
events
    loc arr_c_ra (lamda_RA1)
    loc comm_ra_c (muRA2)
    syn comm_c_mrt (muRA1)
    syn invoque_swA (mu_MTR)
    syn invoque_swB (mu_invoque_SWB)
    syn comm_swa_mtr (mu_SWA)
    syn comm_mtr_ra_c (mu_reponse_MTR)
    syn comm_dist_ra_mtr (mu_SWA)
    syn comm_ra_rb (muRA4)
    syn comm_rb_dist (muRB1)
    syn comm_dist_rb  (mu_dist_rb)
    syn comm_rb_ra (muRB2)
    syn comm_dist_ra_mtr (muRA3)

reachability =(st RA1 <= C) && (RA2 <= CS) && (RB1
<= C) && (RB2 <= CS);

network DEA (continuous)

   aut RA1
        stt n[C1]  to(++)  arr_c_ra comm_rb_ra
comm_mtr_ra_c
         to(--) comm_c_mrt(prob_c_mtr)
comm_dist_ra_mtr (prob_c_mtr)  comm_ra_c (prob_mtr_c)

   aut MTR
        stt pres to(examiner) comm_c_mrt
     stt examiner to(attenteA) invoque_swA (pu_SWA)
              to(attenteB) invoque_swB (pu_SWB)
     stt attenteA to(preparer) comm_swa_mtr
     stt attenteB to(preparer) comm_dist_ra_mtr
     stt preparer to(pres) comm_mtr_ra_c

   aut SWA
        stt pres to(occupe) invoque_swA
         stt occupe to(pres) comm_swa_mtr

   aut RA2
     stt n[CS1] to(++) invoque_swB
         to(--) comm_ra_rb


   aut RB1
     stt n[C1] to(++) comm_ra_rb
         to(--) comm_rb_dist

   aut SWB
     stt pres to(occupe) comm_rb_dist
     stt occupe to(pres) comm_dist_rb

   aut RB2
     stt n[CS1] to(++) comm_dist_rb
         to(--) comm_rb_ra
results

RAlibre = (st RA1 == n[0]) && (st RA2 ==n[0]);
RBlibre = (st RB1 ==n[0]) && (st RB2 == n[0]);
libre = (st RA1 == n[0]) && (st RA2 ==n[0]) && (st RB1
==n[0]) && (st RB2 == n[0])&& (st MTR == pres) &&
(st SWA == pres) && (st SWB == pres) ;
```

Figure 26 : représentation du modèle en format PEPS.

Après l'exécution de notre programme «voir la figure 26 » sous le Logiciel PEPS, nous avons finalement abouti à des vecteurs de probabilités, chaque vecteur donne les probabilités de rester dans chacun des états locaux de chacune des automates modélisant le système. Ces probabilités correspondent aux proportions de temps que les messages restent dans chacun des états.

La méthode itérative qui a été utilisée pour la résolution du système est celui d'Arnold, comme le montre la figure suivante :

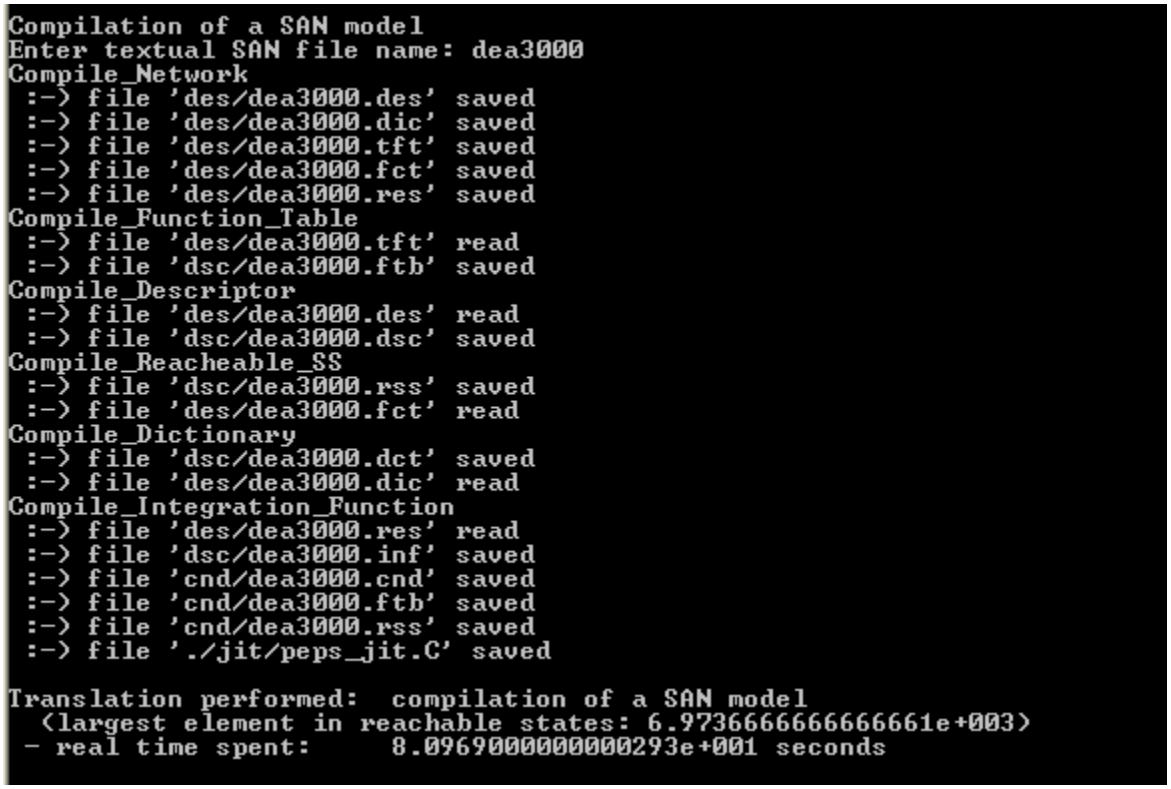


Figure 27 : Compilation du modèle décrit.

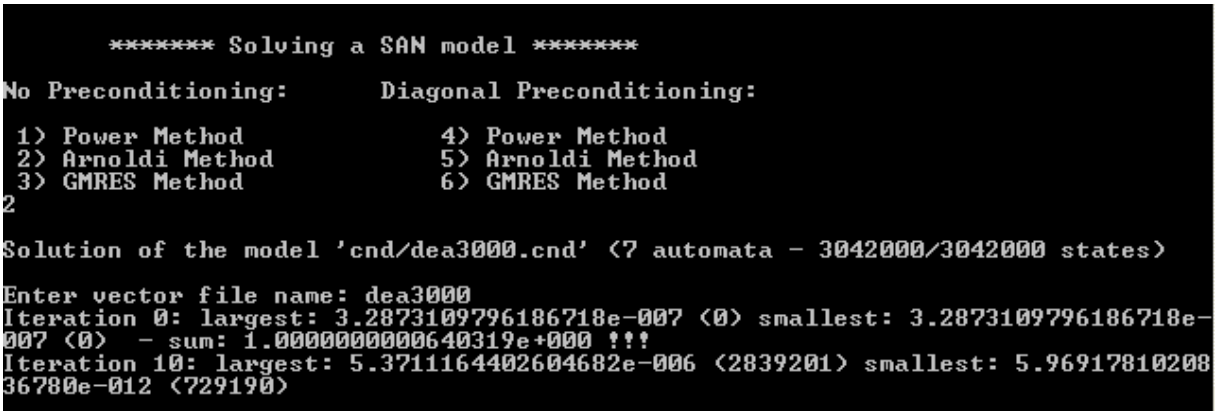


Figure 28 : résolution du système

Le tableau suivant donne quelques valeurs probabilistes calculées à p artir de notre programme :

Taux d'arrivées des requêtes des clientes	Taux d'utilisation de deux routeurs		Probabilité que le Serveur Web soit à l 'état repos (prés)		
----------	RouteurA	RouteurB	Srv locale	Moteur	Srv distant
300	0,120	0,028	0,41	0,478	0,01

600	0 ,235	0,053	0,413	0,509	0,021
900	0,346	0,075	0,418	0,536	0,030
1200	0,454	0,094	0,428	0,559	0,038
1500	0,577	0,215	0,432	0,653	0,078
1800	0,660	0,128	0,437	0,600	0,052
2100	0,796	0,142	0,439	0,623	0,057
2400	0,845	0,156	0,44112	0,632	0,063
2700	0,915	0,168	0,44112	0,644	0,066
3000	0,955	0,363	0 ,44162	0,706	0,123
3500	0,990	0,195	0,44211	0,663	0,068

1. La Longueur des files d'attente au niveau des différents routeurs

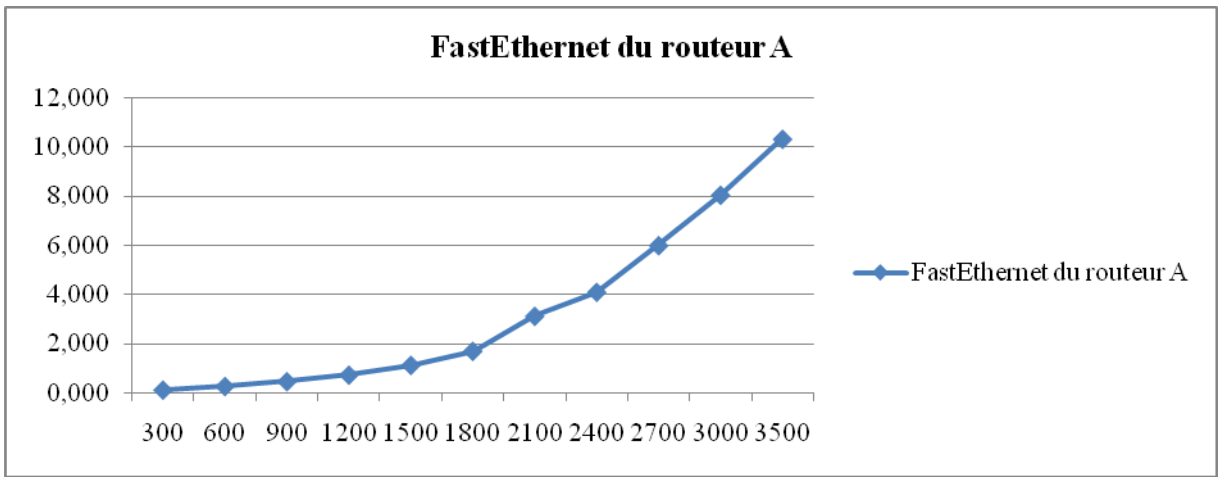


Figure 29: Longueur de file d'attente au niveau de l'interface FastEhernet du Routeur A

Voyons comment la longueur de la file d'attente varie en fonction de nombre des requêtes des clients. Comme notre modèle est un réseau ouvert avec blocage (si le buffer de l'interface est plein on bloque les trafics venant de l'extérieur) et en même temps le nœud qui reçoive les paquets de l'extérieur n'envoie au moteur que le nombre des requêtes qu'il peut traiter par unité de temps (idem pur les autres interfaces), donc il est normale que la longueur de file d'attente au niveau de l'interface FastEthernet du routeur croit rapidement (l'arrivé des requêtes de l'extérieur, l'arrivé des réponses envoyées par le serveur distant et l'arrivé des réponses envoyées par le moteur d'orchestration). Lorsque cette file d'attente devient plein (pour 6000 requêtes), non seulement les requêtes venant de l'extérieurs seront perdu mais encore ça va provoquer le blocage du système dans ça globalité. La figure suivant montre la longueur de file pour les autres interfaces de deux routeurs.

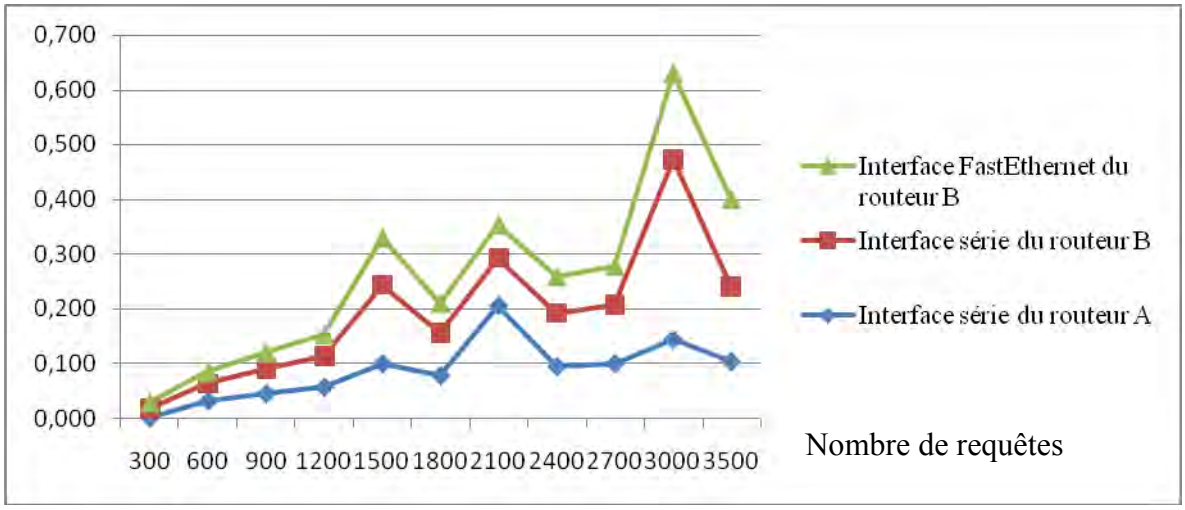


Figure 30 : Longueur des files d'attente au niveau de différentes autres interfaces.

2. Taux d'utilisation des différents routeurs

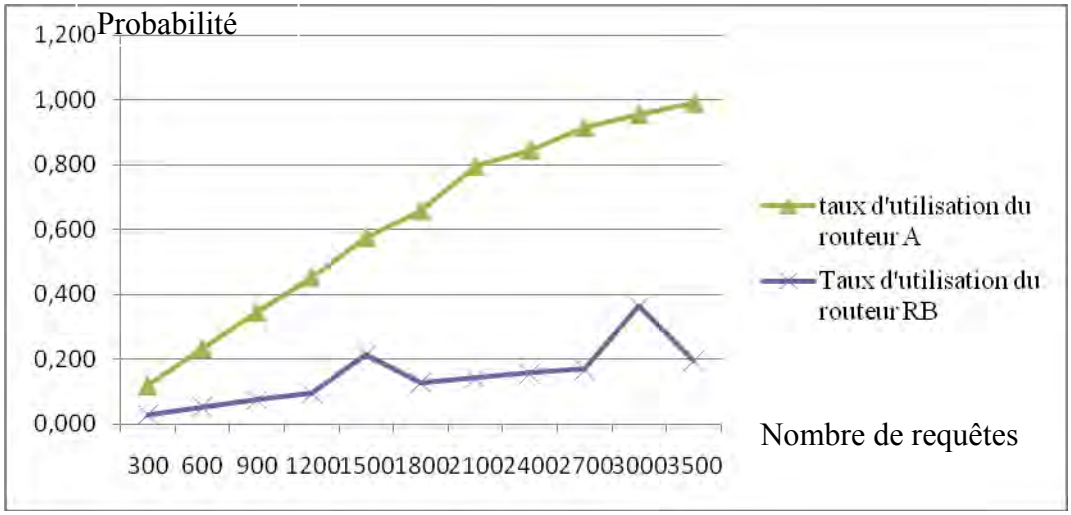


Figure 31 : taux d'utilisation des deux routeurs.

3. Charge du system

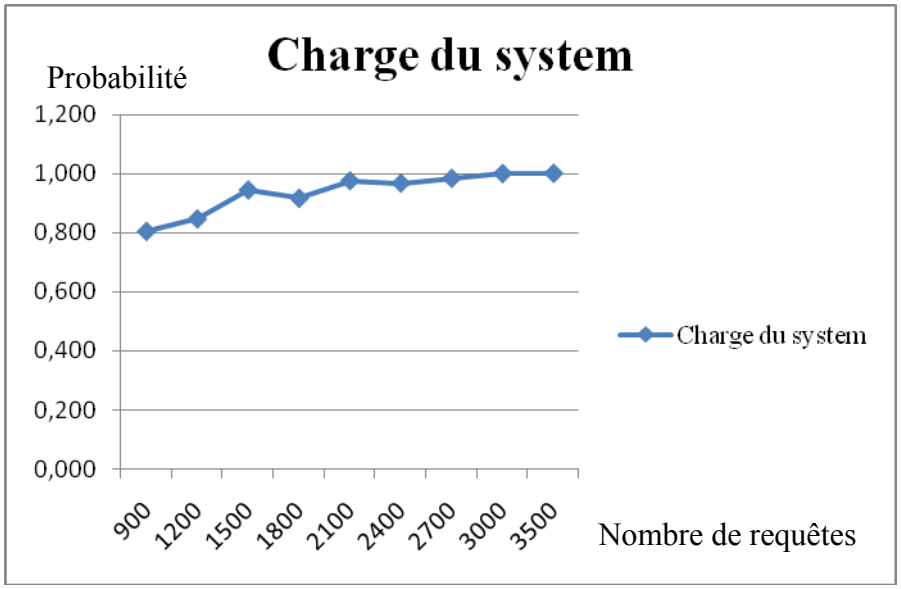


Figure 32 : Charge du système en fonction de nombre des requêtes.

Cette figure montre le rapport entre le taux d'utilisation du système en fonction du nombre de requêtes des clients. Le charge du système est définit par la probabilité que le système soit utilisé (il y'a au moins un des équipements du système qui

4. Taux d'utilisation des différents WEB services

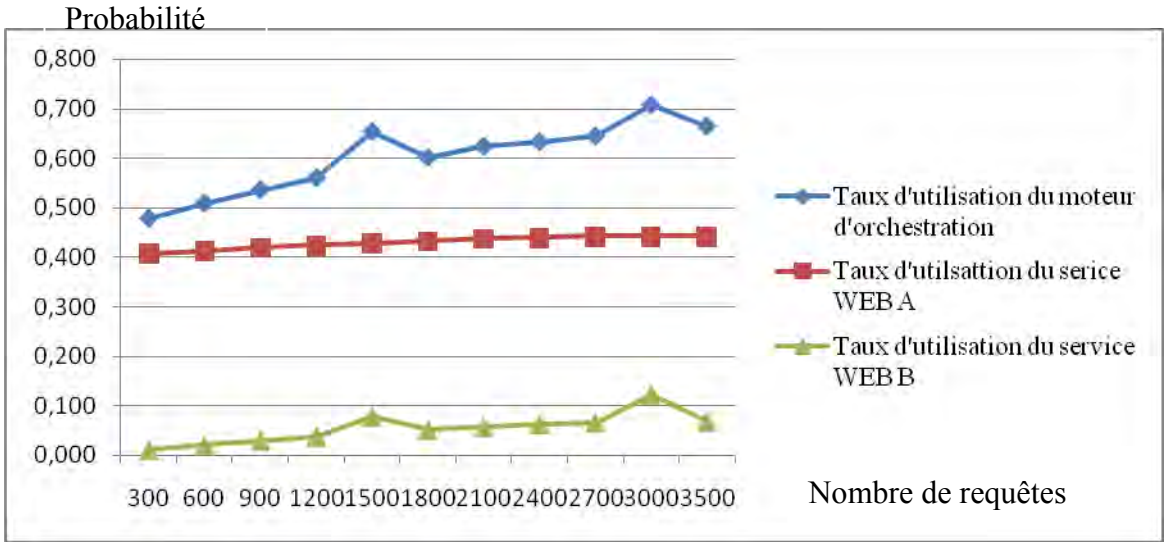


Figure 33 : taux d'utilisation des différents WEB services.

Cette figure montre le taux d'utilisation des différents services Web en fonction du nombre de requêtes des clients.

VI. Conclusion et perspectives

Dans ce mémoire de DEA, nous nous sommes particulièrement intéressés à l'évaluation des services Web composites par les Réseaux d'Automates Stochastiques (RAS). Nous avons dans un premier temps étudié les technologies des services WEB que sont les spécifications SOAP, WSDL, UDDI et BPEL qui constituent une infrastructure de base permettant la mise en œuvre des architectures orientées services.

L'architecture orientée service (SOA) est un standard émergeant utilisé pour développer des applications sur Internet. Ces applications, qui sont généralement de type B2B (business to business) ou B2C (business to customer), sont principalement basées sur l'interaction de services web et font transiter d'énormes quantités de données sérialisées en XML.

Les entreprises, qui souhaitent utiliser cette architecture, veulent avant tout pouvoir faire confiance aux services web qu'elles utilisent, et pour ce faire, elles ont besoin de tester ces services web. Le test permet de s'assurer et de montrer que les services web et les applications qui les utilisent, sont valides. Donc, les entreprises doivent avoir des réponses sur les questions qui portent sur les performances, la sécurité, la composition automatique, …etc.

Dans la deuxième partie de ce mémoire, nous avons traité les aspects liés aux études de performance des services Web composites. Dans cette partie nous avons étudié le formalisme du Réseau d'Automate Stochastique (RAS) en temps continu, et la manière dont il est utilisé pour faire la modélisation. Nous avons également décrit les méthodes itératives utilisées pour le calcule des distributions stationnaire.

Dans la dernière partie nous avons utilisé, comme outil de calcule, le logiciel PEPS pour la représentation du modèle décrit en RAS et pour le calcul des indices de performances.

Comme perspectives, nous comptons continuer dans les aspects performances, et en particulier sur le réseau d'automates stochastiques en temps discret.

Pour les aspects liés aux Services Web composites, nous comptons étudier les Protocoles liées aux gestions de transaction et leur influence sur la performance de Services Web.

VII. Annexe

A. Processus BPEL correspondant à l'exemple d'Agence de voyage

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasisopen.org/wsbpel/2.0/process/executable"
              xmlns:ns1="http://127.0.0.1:8080/activebpel/services/AgenceVoyagePLTService"
              xmlns:ns2="http://127.0.0.1:8080/axis/services/Hotel"
              xmlns:ns3="http://127.0.0.1:8080/axis/services/Banque"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="AgenceVoyage"
              suppressJoinFailure="yes" targetNamespace="http://AgenceVoyage">

              <bpel:import importType="http://schemas.xmlsoap.org/wsdl/" location="../wsdl/AgenceVoy
namespace="http://127.0.0.1:8080/activebpel/
services/AgenceVoyagePLTService"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
location="../wsdl/Hotel.wsdl"
namespace="http://127.0.0.1:8080/axis/services/Hotel"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
location="../wsdl/Banque.wsdl"
namespace="http://127.0.0.1:8080/axis/services/Banque"/>

<bpel:partnerLinks>
        <bpel:partnerLink myRole="Agence" name="AgencePLT" partnerLinkType="ns1:AgencePLT"/>

        <bpel:partnerLink name="HotelPLT" partnerLinkType="ns2:HotelPLT" partnerRole="Hotel"/>

        <bpel:partnerLink name="BanquePLT" partnerLinkType="ns3:BanquePLT" partnerRole="Banque"/>
</bpel:partnerLinks>
<bpel:variables>
        <bpel:variable messageType="ns1:clientRequest" name="clientRequest"/>
        <bpel:variable messageType="ns1:clientResponse" name="clientResponse"/>
        <bpel:variable messageType="ns2:reserverRequest" name="reserverRequest"/>
        <bpel:variable messageType="ns2:reserverResponse" name="reserverResponse"/>
        <bpel:variable messageType="ns3:transfert_creditRequest" name="transfert_creditRequest"/>
        <bpel:variable messageType="ns3:transfert_creditResponse" name="transfert_creditResponse"/>
</bpel:variables>
 <bpel:flow>
     <bpel:links>
         <bpel:link name="L1"/>
         <bpel:link name="L2"/>
         <bpel:link name="L6"/>
         <bpel:link name="L7"/>
         <bpel:link name="L9"/>
         <bpel:link name="L4"/>
         <bpel:link name="L3"/>
         <bpel:link name="L5"/>
         <bpel:link name="L8"/>
         <bpel:link name="L10"/>
 </bpel:links>
<bpel:receive createInstance="yes" operation="reserver" partnerLink="AgencePLT" variable="clientRequest">
    <bpel:sources>
        <bpel:source linkName="L1"/>
        <bpel:source linkName="L2"/>
        <bpel:source linkName="L6"/>
    </bpel:sources>
</bpel:receive>

<bpel:reply operation="reserver" partnerLink="AgencePLT" variable="clientResponse">
    <bpel:targets>
        <bpel:target linkName="L8"/>
        <bpel:target linkName="L10"/>
        </bpel:targets>
</bpel:reply>
<bpel:invoke inputVariable="reserverRequest" name="Invoque_WS_Hotel" operation="reserver"
             outputVariable="reserverResponse" partnerLink="HotelPLT">
    <bpel:targets>
        <bpel:joinCondition>$L5 and $L4</bpel:joinCondition>
        <bpel:target linkName="L4"/>
        <bpel:target linkName="L5"/>
        </bpel:targets>
        <bpel:sources>
            <bpel:source linkName="L7">
                <bpel:transitionCondition>$reserverResponse.reserverReturn
                                !='imposible de faire la reservation'</bpel:transitionCondition>
            </bpel:source>
        </bpel:sources>
</bpel:invoke>
```

```
<bpel:invoke inputVariable="transfert_creditRequest" name="Invoque_WS_Banque"
                                                     operation="transfert_credit"
              outputVariable="transfert_creditResponse" partnerLink="BanquePLT">
    <bpel:targets>
        <bpel:target linkName="L3"/>
    </bpel:targets>
    <bpel:sources>
        <bpel:source linkName="L9">
            <bpel:transitionCondition>$transfert_creditResponse.transfert_creditReturn = 'false'
            </bpel:transitionCondition>
        </bpel:source>
        <bpel:source linkName="L4">
                <bpel:transitionCondition>$transfert_creditResponse.transfert_creditReturn= 'true'
            </bpel:transitionCondition>
        </bpel:source>
    </bpel:sources>
</bpel:invoke>
<bpel:assign>
    <bpel:targets>
        <bpel:target linkName="L2"/>
    </bpel:targets>
    <bpel:sources>
        <bpel:source linkName="L3"/>
    </bpel:sources>
    <bpel:copy>
        <bpel:from>number( $clientRequest.NbreChambe ) * number($clientRequest.Nbrenuite ) * number(100
        </bpel:from>
        <bpel:to part="montant" variable="transfert_creditRequest"/>
    </bpel:copy>
    <bpel:copy>
        <bpel:from part="idCompte" variable="clientRequest"/>
        <bpel:to part="idCompteA" variable="transfert_creditRequest"/>
    </bpel:copy>
    <bpel:copy>
        <bpel:from>'idCompteB'</bpel:from>
        <bpel:to part="idCompteB" variable="transfert_creditRequest"/>
        </bpel:copy>
</bpel:assign>
 <bpel:assign>
     <bpel:targets>
         <bpel:target linkName="L1"/>
     </bpel:targets>
     <bpel:sources>
         <bpel:source linkName="L5"/>
     </bpel:sources>
     <bpel:copy>
         <bpel:from>'date'</bpel:from>
         <bpel:to part="date" variable="reserverRequest"/>
     </bpel:copy>
     <bpel:copy>
         <bpel:from part="NbreChambe" variable="clientRequest"/>
         <bpel:to part="NbreChambe" variable="reserverRequest"/>
     </bpel:copy>
     <bpel:copy>
         <bpel:from part="Nbrenuite" variable="clientRequest"/>
         <bpel:to part="Nbrenuite" variable="reserverRequest"/>
     </bpel:copy>
     <bpel:copy>
         <bpel:from>'AgenceA'</bpel:from>
         <bpel:to part="IdAgent" variable="reserverRequest"/>
     </bpel:copy>
 </bpel:assign>
<bpel:assign>
    <bpel:targets>
        <bpel:joinCondition>$L7 and $L6</bpel:joinCondition>
        <bpel:target linkName="L6"/>
        <bpel:target linkName="L7"/>
    </bpel:targets>
    <bpel:sources>
        <bpel:source linkName="L8"/>
    </bpel:sources>
    <bpel:copy>
        <bpel:from part="reserverReturn" variable="reserverResponse"/>
        <bpel:to part="codereservationReturn" variable="clientResponse"/>
    </bpel:copy>
    <bpel:copy>
        <bpel:from>number( $clientRequest.NbreChambe ) * number($clientRequest.Nbrenuite ) * number(100
            </bpel:from>
        <bpel:to part="montantreservationReturn" variable="clientResponse"/>
    </bpel:copy>
</bpel:assign>
<bpel:assign>
    <bpel:targets>
        <bpel:target linkName="L9"/>
        </bpel:targets>
    <bpel:sources>
            <bpel:source linkName="L10"/>
    </bpel:sources>
    <bpel:copy>
        <bpel:from>'imposible de faire la reservation votre compte ne
                    contient pas suffisamant dargent'</bpel:from>
        <bpel:to part="codereservationReturn" variable="clientResponse"/>
    </bpel:copy>
    <bpel:copy>
        <bpel:from>'0'</bpel:from>
        <bpel:to part="montantreservationReturn" variable="clientResponse"/>
    </bpel:copy>
</bpel:assign>

</bpel:flow>
</bpel:process>
```

VIII. Références bibliographiques

[**1**] Jérôme Daniel « Services Web Concepts, techniques et outils », Vuibert, Paris, 2003.

[**2**] Spécification SOAP: http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[**3**] Services Web avec SOAP, WSDL, UDDI, ebXML... de Jean-Marie Chauvet, © Eyrolles, 2002.

[**4**] Services Web avec J2EE et .NET Conception et implémentations de Libero Maesano, Christian Bernard et Xavier Le Galles © Eyrolles, 2003.

[**5**] Spécification SOAP avec attachement: http://www.w3.org/TR/SOAP-attachments;

[**6**] Spécification XML Schema Part 0: http://www.w3.org/TR/xmlschema-0/#CreatDt.

[**7**] Spécification WSDL 1.1 « http://www.w3.org/TR/wsdl ».

[**8**] "UDDI Version 2.03 Data Structure Reference UDDI Committee Specification, 19 July 2002" « http://uddi.org/pubs/DataStructure-V2.03-Published-0020719.htm#_Toc25130756 »

[**9**] "Web Services Business Process Execution Language Version 2.0" http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[**10**] Nicolas Salatgé «thèse sur la conception et mise en œuvre d'une plate-forme pour la sûreté de fonctionnement des Services Web », 8 décembre 2006.

[**11**] Hernán Darío ROJAS « Mémoire DEA sur ORCHESTRATION A HAUT NIVEAU ET BPEL » Soutenu le 21 juin 2006.

[**12**] GEAY Sébastien REPETTO Pierre VICARD Sébastien « Travaux d'Etudes de Licence d'Informatique » 2005.

[**13**] Nerea Arenaza « Projet de Master Composition semi-automatique de Services Web » Février 2006.

[**14**] Sanlaville, Sonia. « Thèse de doctorat : Environnement de procédé extensible pour l'orchestration Application aux services web », Décembre 2005.

[**15**] Sylvain RAMPACEK « Thèse de doctorat : Sémantique, interactions et langages de description des services web complexes », 10 novembre 2006.

[16] Anne Benoit « Thèse de doctorat : Méthodes et algorithmes pour l'évaluation des performances des systèmes informatiques à grand espace d'états », le 18 Juin 2003.

[17] Paulo Henrique Lemelle FERNANDES «Thèse de doctorat : Méthodes Numériques pour la Solution de Systèmes Markoviens à Grand Espace d'États», février 1998.

[**18**] Théorie des files d'attente « des chaines de Markov aux rseaux à forme produit ; Bruno Baynat, Hermes Science Publication, 2000.

[**19**] Peps 2007 User manual: «http://www- id.imag.fr/Logiciels/peps/san.html»

[**20**] ActiveBPEL « http://www.activevos.com/ »