Remerciements

Tout d'abord, je tiens à remercier le bon Dieu le tout Puissant de m'avoir donné la force et le courage de mener à bien ce modeste travail.

Je remercie sincèrement mon superviseur **Mme Mohammedi Amira** pour ses encouragements, sa patience infinie et ses conseils précieux.

Je tiens à exprimer mes sincères remerciements aux membres du jury :**Mme Aloui Imane** et **Mme Bouguetitiche Amina** d'avoir pris le temps de lire et d'évaluer mon mémoire.

Je n'oublie jamais de remercier tous mes professeurs du département d'informatique qui m'ont appris les principes de base de l'informatique.

Enfin, je suis profondément reconnaissant à ma mère :**Hasnaoui Saliha**, à mon père :**Ababsa abd allah** et à tous les membres de ma famille.

Table des matières

Liste des tableaux

Table des figures

Glossaire

SOA : Service Orienté Architecture.

CSP : Communicating Sequential Processes.

CADP : Construction Analysis Distributed Processes.

BPEL : Busness Process Execution Language.

TLA : Logique Temporelle Actions.

ADT : Abstract Data Types.

LTS : Labelled Transition System.

ACP : Algebra of Communicating Processes.

OSI : Open Systems Interconnection.

ITU : International Telecommunication Union.

ETSI : European Telecommunications Standards Institute.

SDL : Specification Description Langage.

WS : Web Service.

B2C : Busness two Busness.

SOAP : Simple Object Access Protocol.

WS-I : Web Services Interoperability.

SMTP : Simple Mail Transfer Protocol.

JMS : Java Messagerie Services).

FTP : File Transfer Protocol.

WSDL : Web Service Description Language.

UDDI : Universal Description Discovery Integration.

HTTP : Hypertext Transfer Protocol.

XML : Extensible Markup Language.

BPEL : Busness Process Execution Language.

RDP : Réseau de Petri.

IBM : International Business Machines.

CCS : Calculus Communicating Systems.

LOTOS : Language Of Temporal Ordering Specification.

VDM : Vienna Developement Methode.

OCIS : Open Cæsar Interactive Simulator.

MCL : Model Checking Language.

Introduction

Ces dernières années ont vu l'émergence d'un nouveau style architectural reflété par les architectures orienté service (SOA). Celles-ci sont conçues pour faciliter la création, l'exposition, l'interconnexion et la réutilisation d'application à base de services.Plusieurs réalisation de ce type d'architecture ont vu le jour dont la plus commune se base sur les services Web (5).

La composition des services Web est un sujet qui a suscité l'intérêt des chercheurs depuis son apparition. Elle offre la possibilité de traiter des problèmes complexes même avec des services simples existants tout en coopérant entre eux. Même des comportements très simples peuvent devenir largement complexes quand exécutés en parallèle (6). En même temps, ils ont souvent un caractère critique, car un dysfonctionnement de leur part peut entrainer une perte matérielle, des dégâts considérables ou même une perte de vies humaines. C'est pourquoi, les éventuelles erreurs doivent être détectées le plus tôt possible dans le processus d'élaboration de ces systèmes (7).

Les langages semi-formels comme le langage (BPEL, UML,...etc) sont des langages de composition des services Web très utilisés, et qui fournissent des services composites difficiles à vérifier à cause de leur manque de formalisme. Pour remédier à ce problème, il est nécessaire d'avoir une méthode qui permet d'analyser les services Web avant leur mises en œuvre, dans le but de fournir et de créer des systèmes corrects. Dans ce cas, les méthodes formelles apparaissent comme la solution la plus adéquate et l'utilisation de méthodes de spécification et de vérification formelle, assistées par des outils informatiques adaptés est devenue indispensable.

Les systèmes basés sur la composition des services Web ont plusieurs caractéristiques, entre autres : la distribution, le parallélisme asynchrone, communication,... Pour spécifier un tel type de systèmes en détail, il est impératif de les disséquer en leurs composants concurrents (6). Il faut aussi pouvoir exprimer la communication entre eux, et de représenter les données échangés. Parmi les outils formels existants, on trouve l'algèbre des processus.

L'algèbre des processus est un Framework mathématique dans lequel le comportement du système est exprimé sous la forme de termes algébriques, améliorant ainsi les techniques de manipulation disponibles. Il permet un raisonnement formel sur les processus et les données, en mettant l'accent sur les processus exécutés simultanément. Le traitement de base de

l'algèbre est un opérateur parallèle qui décompose les systèmes en leurs composants concurrents (7). La spécification algébrique permet de donner des définitions relativement simples et précises des types abstraits de données (8).

Plusieurs langages de l'algèbre des processus sont apparus :CCS, CSP, LOTOS,...etc. Nous avons opté pour le langage LOTOS car il possède des opérateurs et des types abstraits assez complets pour exprimer le fonctionnement du système.

L'objectif de notre projet est la spécification formelle d'un système à base de SOA, et plus particulièrement une composition de services Web, illustrée dans le cas d'une agence de voyage, en utilisant LOTOS et l'outil CADP pour la vérification de notre spécification.

La suite de ce mémoire est structuré comme suit : il se compose principalement deux parties. La Partie I s'intéresse aux concepts nécessaires pour notre travail. elle est divisée, elle-même, en deux chapitres. Le chapitre 1 décrit l'architecture orientée service (SOA) et présente les services Web qui permettent de mettre en œuvre ce type d'architecture. Nous aborderons ensuite la composition des services Web. Le chapitre 2 étudie quelques outils formels permettant la spécifications des services Web et leur comparaison. Ensuite, nous nous concentrons sur l'algèbre des processus et surtout le langage LOTOS que nous utiliserons pour la spécification formelle de notre système. Nous présenterons ensuite, la boîte à outils CADP qui permet d'analyser et de vérifier formellement des systèmes décrits à l'aide de LOTOS.

La Partie II présente notre travail qui consiste a spécifier un système de réservation de voyage, puis la vérification de quelques propriétés, en utilisant CADP, il est divisée en deux chapitres. Le chapitre 3 illustre quelques diagrammes issus de l'analyse du système, qui permettra de faire notre spécification. Cette dernière est donnée en LOTOS et ses différentes parties sont expliquées. Le chapitre 4 explique l'utilisation de l'outil CADP pour la vérification de quelques propriétés de notre application et les arbres de spécification résultants de la compilation en utilisant l'outil CADP, sont présentés.

A la fin de ce mémoire, la conclusion générale, récapitule les résultats et présente quelques perspectives de notre travail .

Première partie

Etat de l'art

Chapiter 1

Architecture Orientée Service et Services Web

Chapitre 1

Architecture Orientée Service et Services Web

Introduction

De nos jours, les entreprises vivent dans un environnement commercial de plus en plus concurrentiel et axé sur la technologie. La capacité d'adapter rapidement ses processus métier en réponse à des changements internes et externes est indispensable pour toute entreprise (9). Une solution très en tendance ces dernières années, consiste à utiliser une architecture orientés services pour les systèmes informatiques.

L'architecture Orientée service (SOA) a été largement employée dans plusieurs domaines tel que dans les systèmes e-business, e-gouvernement, systèmes automobiles, services multimédia, finances et dans beaucoup d'autres domaines. Les services Web sont devenus une technique incontournable pour construire des systèmes distribués faiblement couplés basés sur les SOA(10).

Nous allons, dans ce chapitre, donner un aperçu sur les SOAs et leurs principes et avantages. Nous allons, en suite se concentrer sur la technologie des services Web, une réalisation largement connue et utilisée, des SOAs. Après la définition des services Web, nous allons présentés brièvement leurs technologies de base standards, et leurs approches de compositions.

Définition

Selon le point de vu considéré, la définition d'une SOA est différente pour une entreprise. Nous allons présenter ci-dessous, quelques définitions (11) :

Point de vu des dirigeants : Une SOA est un ensemble de services que l'entreprise souhaite exposer à leurs clients et partenaires, ou d'autres parties de l'organisation.

Point de vu des architectes : La SOA est un style architectural basé sur un fournis-

seur, un demandeur et une description de service et supporte les propriétés de modularité, encapsulation, découplage, réutilisation et composabilité.

Point de vu des développeurs : La SOA est un modèle de programmation avec ses standard, paradigmes, outils et technologies associées.

Point de vu des intégrateurs : La SOA est un intergiciel offrant des fonctionnalités en terme d'assemblage, d'orchestration, de surveillance et de gestion des services .

Mais une définition commune d'une SOA : est une architecture qui a vu le jour au début de l'année 2000, pour répondre aux besoins des entreprises en termes de décentralisation et de répartition des applications logicielles. A l'origine, celle-ci a été essentiellement utilisée pour résoudre les problématiques d'interopérabilité entre les différentes technologies informatiques distribuées utilisées en entreprise. C'est une solution très efficace en termes de réutilisabilité, d'interopérabilité et de réduction de couplage entre les différentes entités implémentant leurs propres systèmes d'information. SOA est un modèle d'interaction applicative qui permet de décomposer une application en un ensemble de composants basiques (c-à-d services), de décrire le schéma d'interaction entre ces composants en utilisant un format d'échange bien défini, le plus souvent XML, et des couplages externes par l'intermédiaire d'une couche d'interopérabilité des interfaces, le plus souvent un Service Web.

L'architecture SOA a été popularisée avec l'apparition des standards comme les services Web dans le-commerce, B2B (Business to Business) ou B2C (Business to Client). L'élément clé de la mise en œuvre de cette architecture est de rendre accessibles, via un réseau (12).

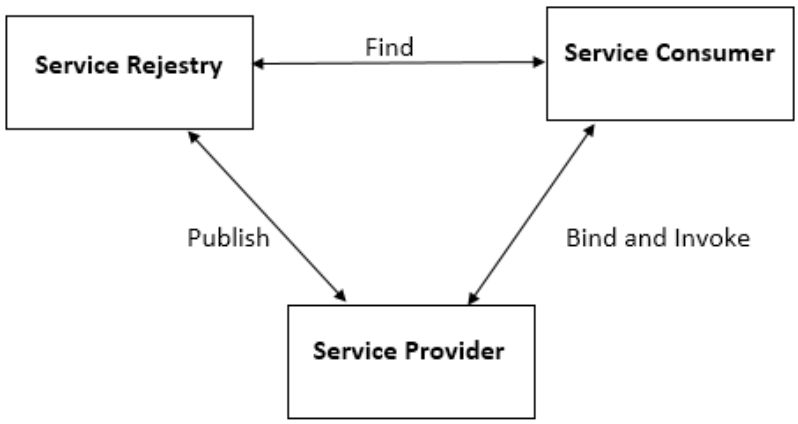

FIGURE 1.1 – Schéma du SOA

Ce schéma représente trois acteurs principales des SOAs qui sont (13) :

Consommateur de service (Service consumer) : le consommateur de service est une application, un logiciel module ou un autre service nécessitant un service. Il initie l'enquête de

le service dans le registre, se lie au service via un transport et exécute la fonction de service. Le consommateur de service exécute le service conformément à le contrat d'interface.

Fournisseur de service (Service provider) : le fournisseur de service est entité adressable par le réseau qui accepte et exécute les demandes des consommateurs. Il publie ses services et contrat d'interface avec le registre de services afin que le consommateur de services puisse découvrir et accéder au service.

Registre de service (Service registry) : Un registre de service est le moyen de détecter le service. Il contient un référentiel des services disponibles et permet la recherche de service interfaces des fournisseurs aux consommateurs de services intéressés.

1.1 Principes des SOAs

L'architecture SOA est basée sur les huit principes (14) :

- Autonomie : les services contrôlent la logique qu'ils en capsulent.
- Sans état :les services minimisent la consommation de ressources en reportant la gestion d'informations d'état lorsque nécessaire.
- Capacité de découverte : les services sont complétés par des méta données par lesquelles ils peuvent être découverts et interprétés
- Composabilité : les services sont des participants efficaces dans des compositions, quelles que soient la taille et la complexité de la composition.
- Contrat standardisé : le contrat de service adhérent à un accord de communication, collectivement défini avec un ou plusieurs documents de description.
- Faible couplage : faible couplage des services avec le maintient de relations réduisant les dépendances.
- Abstraction : l'abstraction des services doit dissimuler la logique du service à l'extérieur.
- Réutilisation :partage de la logique entre plusieurs services avec l'intention de promouvoir la réutilisation.

1.2 Avantages des SOAs

L'architecture SOA offre plusieurs avantages présenter ci-dessous (15) :

- Agilité métier.
- Réduction des couts.

- Accroissement revenues.
- Satisfaction du client.
- Plus rapidement vers le marché.
- Plus de productivité.
- Réutilisation des services.

1.3 Inconvénients des SOAs

par contre les SOAs ont des inconvénient présenter ci-dessous (14) :

- Difficile à tester.
- Risque de prolifération des messages (entre services).
- Risque liés à la sécurité des messages provenant de sources déverses.

1.4 Services Web

Définition
Le service est composant clef d'une architecture SOA. C'est une entité de traitement qui représente une fonction ou une fonctionnalité bien définie qui est divisée en opérations. Un service peut implémenter plusieurs interfaces, et aussi plusieurs services peuvent implémenter une même interface. Les services interagissent et communiquent entre eux (16).

Et aussi, un service Web est un ensemble de protocoles et de normes informatiques utilisés pour échanger des données entre les applications. Il représente une entité autonome qui est publiée, localisée et invoquée a travers le Web. Il est basé sur un ensemble de standard XML, lui permettant d'être plus portable que les technologies précédentes (17).

1.4.1 Objectifs des services Web

Le service Web offre les objectifs présentés ci-dessous (18) :

- Faciliter l'accès aux applications entre entreprise et ainsi simplifier les échanges de données .
- Permettre aux applications d'interopérer à travers un réseau, indépendamment de leur plates-forme et de leurs langages d'implémentation.

  Le fonctionnement des services Web s'articule autour de trois acteurs principaux illustrés par la figure 1.2 (19) :

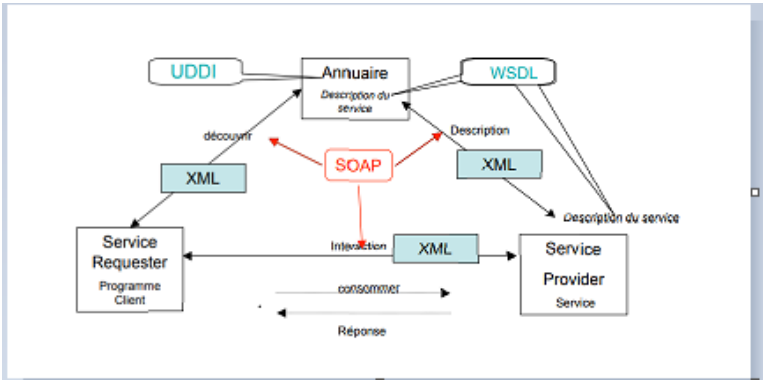

FIGURE 1.2 – Architecture de service Web (1)

Fournisseur (Service Provider) est un serveur exécutant des applications ou composants assimilables à des services Web, leurs interface sont alors décrites en WSDL. Pour se faire connaitre,le **fournisseur** de services publie la description WSDL des services qu'il fournit sur un ou plusieurs **annuaires** de services UDDI. Ainsi,lorsque le **demandeur de services** (c'est-à-dire le client) veut savoir quels sont les serveurs **fournissant** un service correspondant à une certaine description, il fait appel à un **annuaire** UDDI dont il a la connaissance. Ce dernier lui renvoie alors le ou les fichiers WSDL des services Web correspondant à la demande. **Le client** (Service Requester) fais alors son choix, s'adresse ou **fournisseur** et invoque le service Web, dont il n'avait pas nécessairement connaissance auparavant. **l'annuaire** peut être local à une application,à un réseau d'entreprises ou à l'échelle d'internet.

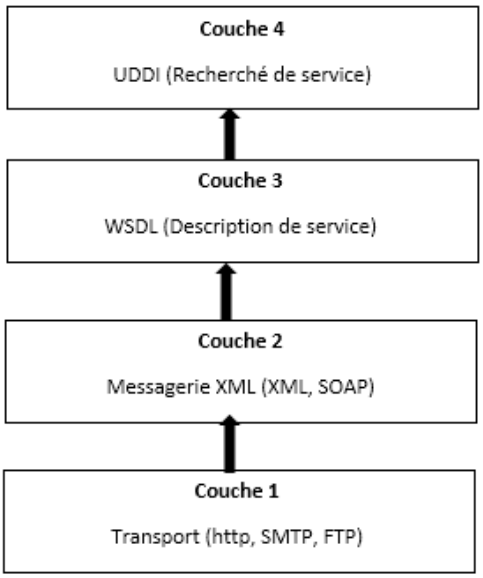

FIGURE 1.3 – Couches de la Pile de protocoles de service Web

La technologie des services Web se composent d'un ensemble de protocoles et de standards Internet permettant l'échange de données entre des applications. Les principaux composants ou couches d'une pile de protocoles de services Web incluent :

Couche de transport : Cette couche s'intéresse aux protocoles de transport de bas niveau, ces derniers vont transporter les requêtes et les réponses échangées entre services. Le protocole le plus utilisé et recommandé par le consortium WS-I (Web Service Interoperability) est l'HTTP, mais d'autres implémentations peuvent utiliser un autre ensemble de protocoles tels que : FTP, JMS (java messagerie services), SMTP, etc (20).

Couche de messagerie XML : Cette couche spécifie les protocoles d'échanges de documents XML entre le service Web et ses clients, elle caractérise aussi le mode d'échange (s'il est bloquant ou non). Le protocole SOAP est adopté comme un standard pour la messagerie entre les services Web (20).

SOAp : est un protocole de transmission de messages basé sur XML. C'est un standard recommandé par le World Wide Web Consortium qui définit un ensemble de règles pour structurer des messages qui peuvent être utilisés dans de simples transmissions unidirectionnelles, mais il est particulièrement utile pour exécuter des dialogues requête-réponse RPC (Remote Procedure Call). SOAP permet une communication en univers hétérogène, il est assez léger, simple et facile à déployer, extensible et ouvert mais ne garde pas la trace des requêtes envoyées des différents clients vers les services Webs invoqués (21).

Couche WSDL : Web Service Description Language qui repose sur la notation XML. La description d'un service avec WSDL doit inclure l'emplacement du service Web ainsi que les opérations (méthodes, paramètres et valeurs de retour) que le service propose. La description de l'interface d'un service Web est effectuée par le WSDL (22).

Couche UDDI : Universal Description Discovery and Integration est un standard (22)et est un annuaire de service fondé sur XML et plus particulièrement destiné aux services Web. Un annuaire UDDI permet de localiser sur le réseau le service Web recherché, il s'agit d'un élément clé dans les spécifications de Services, car il permet l'accès au répertoire des utilisateurs potentiels de services Web (23).

1.4.2 Caractéristiques des services Web

Les services Web sont caractérisés par (24) :

Interopérabilité : L'interopérabilité présente la caractéristique principale des services Web. En effet, quels que soient la plate-forme utilisée (Windows, Unix ou autres) et le langage de développement employé, les services Web se basent sur des standards XML pour simplifier la construction des systèmes distribués et la coopération entre ces derniers.

Simplicité d'utilisation : L'utilisation des standards tels que XML et HTTP a mis en valeur la simplicité de la manipulation des services Web et a encouragé les acteurs forts du marché tels que Microsoft et IBM pour intégrer de nouveaux produits.

Couplage souple des applications : Les services Web constituent un support d'échange des documents structurés qui traversent les contrôles d'accès dans un environnement hétérogène. La collaboration entre différentes applications afin d'échanger des documents se fait d'une manière directe entre objets.

1.4.3 Composition des services Web

Dans le domaine de la composition de services Web, il existe deux approches principales, qui sont : l'orchestration et la chorégraphie.

1.4.3.1 Orchestration

L'orchestration est un ensemble d'actions à réaliser par l'intermédiaire de services Web. Un moteur d'exécution est un service Web jouant le rôle de chef d'orchestre, gère l'enchaînement des services Web par une logique de contrôle comme illustré à la figure 1.4. Pour concevoir une orchestration de services Web, il faut décrire les interactions entre le moteur d'exécution et les services Web. Ces interactions correspondent aux appels, effectués par le moteur, d'action(s) proposée(s) par les services Web composants (25). l'orchestration basé sur le langage BPEL.

1.4.3.1.1 Business Process Execution Language

BPEL est un langage exécutable standardisé par l'OASIS et basé sur XML. Nous avons choisi d'utiliser BPEL parce qu'il s'agit du langage d'orchestration qui a gagnée la bataille des standards en étant le langage le plus utilisé actuellement par l'industrie (26). BPEL repose sur un modèle constitue d'activités qui peuvent être de deux types (26) :

- Activités de base.
- Activités structurées.

les activité de base sont présentées ci-dessous (20) :

Activité de base	**Explication**
Invoke	-Elle permet l'appel d'un partenaire (service) en se basant sur une opération de type one way ou request-response.
Receive	-C'est une attente bloquante d'un message entrant.
Reply	-Elle retourne un message suite à une réception d'un autre message (à l'aide de la primitive receive).
Assign	-Cette activité permet la mise à jour des variables.
Wait	-Elle permet de bloquer l'exécution du processus pour une période donnée.
Throw	-Elle permet d'indiquer les erreurs et les exceptions survenues lors de l'exécution du processus et de les envoyer au catch de fault handler.

TABLE 1.1 – Activité de base de BPEL

Les activités structurés sont présentées ci-dessous (20) :

Activité structuré	**Explication**
Sequence	-Elle définit une suite d'activités exécutées par ordre d'apparition dans la séquence .
Flow	-Elle définit des activités exécutées en parallèle.
Switch	-Elle définit un branchement conditionnel.
While	-Elle modélise une boucle conditionnelle.
Pick	-Elle bloque une activité jusqu'à la réception d'un message ou l'expiration d'une période de temps.

TABLE 1.2 – Activité structuré de BPEL

Les activités de base sont utilisées pour l'invocation d'une opération d'un service . Elles manipulent un certain nombre de variables et elles sont liées à des partnerLinks. La conversation entre partenaires est assurée par des correlationSets (26).

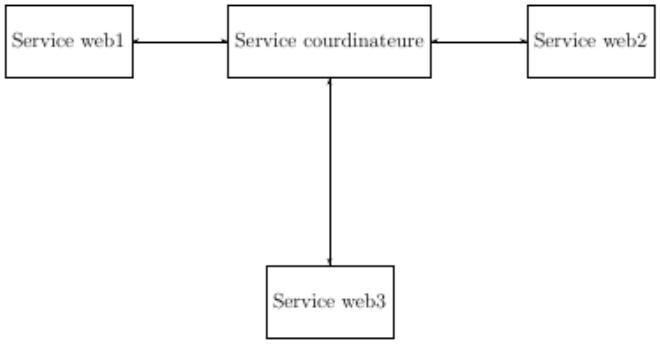

FIGURE 1.4 – Principe de l'orchestration de service Web

1.4.3.1.2 Avantages de l'orchestration

L'orchestration est la technique de composition des services Web la plus utilisée. En effet, elle offre plusieurs avantages, parmi lesquels nous citons (27) :

- La clarté : Le responsable de tout le processus métier est connu de façon exacte (le coordinateur).
- La facilité de mise en œuvre : L'orchestration permet de faciliter le passage de la conception à l'implémentation. Elle propose l'utilisation de normes et de standards ouverts, ce qui permet aussi de réduire les couts de mise en œuvre.
- La flexibilité : Les services Web peuvent être incorporés et/ou substitués sans soucis, parce qu'ils n'ont pas conscience d'appartenir à un processus métier.
- Simplicité, etc.

1.4.3.2 Chorégraphie

Contrairement à la l'orchestration, la chorégraphie est aussi appelée composition dynamique (25), ne nécessite pas de coordinateur central.

Chaque service Web impliqué dans la composition 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 processus décrit le rôle qu'il joue dans le processus. Dans une chorégraphie, les services sont vus comme des « danseurs » qui savent exactement quoi faire et de quelle manière interagir avec leurs partenaires. Il s'agit d'une approche collaborative où chacun des participants a besoin d'un document dans lequel l'interaction est décrite (28).

Le principe de la chorégraphie est illustré par la figure 1.5.

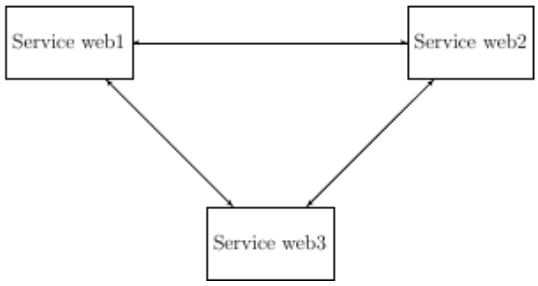

FIGURE 1.5 – Principe de chorégraphie des services Web

Remarque
Les avantages et les inconvénient des services Web sont les même des SOAs.

Conclusion

Notre travail consistant à spécifier et à vérifier un système basé SOA, et particulièrement, sur la composition des services Web, nous avons, alors présent, dans ce chapitre, de manière générale les concepts des SOAs et des services Web, qui en constituent une réalisation largement connue et utilisée. Nous avons d'abord présenté quelques définitions. Nous avons également décrit l'architecture fondamentale des SOAs et des services Web ainsi que les langages et les protocoles de base permettant leur déploiement. Enfin nous avons abordé le sujet de la composition de service Web.

Dans le prochain chapitre, nous nous intéressons aux outils formels permettant de spécifier et de vérifier ce type de système.

Chapitre 2

Approches Formelles

Chapitre 2

Approches formelles

Introduction

Ce chapitre introduit différents outils de description formelle pouvant être utilisés pour spécifier la composition des services Web.

Nous commençons par exposer les formalismes dédiés à la spécification des systèmes distribués concurrents, à savoir : les approches logiques, les systèmes état-transition et l'algèbre de processus. Cette dernière constitue le moyen que nous utiliserons pour la spécification dans notre projet. Alors, elle a été présentée en plus de détails.

Plus d'intérêt a été porté au langage LOTOS, un langage de description formelle standardisé par l'ISO, qui pourrait être utilisé pour la spécification de la composition des services Web. Les (sous-) services de cette composition pourraient être vus comme des processus concurrents capables de se synchroniser.

2.1 Définition des approches formelle

Elles sont des techniques basées sur les mathématiques pour décrire des propriétés de systèmes informatiques. Elles fournissent un cadre pour spécifier, développer et vérifier des systèmes d'une façon systématique (29). Ainsi l'emploi de méthodes de spécification formelle est d'une grande importance pour le développement de systèmes logiciels et ceci tout particulièrement pour la conception et la vérification de systèmes dits sécuritaires ou critiques (par exemple ceux impliquant la vie humaine).

Le fait que les spécifications formelles permettent une description non ambigue, précise, complète et indépendante de toute implémentation conduit à une compréhension approfondie du système à développer et beaucoup d'erreurs sont ainsi évitées. La possibilité de leur appliquer des vérifications et des validations est l'un des points qui concourent à leur

utilisation de plus en plus fréquente. Les méthodes formelles permettent aussi de dériver automatiquement des prototypes à partir des spécifications (9).

Il y'a beaucoup formalismes qui sont classés en trois classes principales, présentées ci-dessous.

2.2 Approches basées sur la logique

Ces approche peuvent se baser sur la théorie des ensembles (langage Z, Vienna Development Methode (VDM, méthode B), les logiques temporelles (TLA), les logiques d'ordre supérieur et logique linéaire.

Nous allons voir ci-dessous quelques approches basées sur la logiques.

2.2.1 Méthode B

Définition
La méthode B est une méthode de spécification formelle du logiciel. Elle permet de décrire formellement un système de sa spécification abstraite jusqu'à son implémentation. Son formalisme repose sur des notions mathématiques de la théorie des ensembles et de la logique du premier ordre. Elle se base également sur une relation qui permet de relier une machine abstraite à une machine concrète qui détaille la spécification de la machine abstraite. Cette relation, appelée raffinement (30). Méthode B utilisé pour le systèmes distribuer (31). Méthode B utilise dont plusieurs domaine (32) :

- Automobile.
- Banque.

2.2.1.1 Avantages de méthode B

La méthode B offre plusieurs avantage (33) :

- La seule méthode à couvrir le cycle de vie de manière consistante de la spécification à la production du code (technique de raffinement et preuve intrinsèquement liées)
- La seule méthode à pouvoir initialement dissocier, puis par la suite verrouiller, le quoi et le comment.
- Coût des outils attractif comparé à celui des autres technologies.

2.2.1.2 Inconvénients de la méthode B

Par contre la méthode B a des inconvénients qui sont présentés ci-dessous (33) :

- En B, on prouve difficilement à posteriori des logiciels qui n'on pas été conçus pour l'être moyennement adaptée à une spécification logicielle mouvante.
- Gain économique lié à la mise en œuvre de B (comme beaucoup d'autres techniques formelles) difficile à quantifier.

2.3 Approches par modèles à états

Ces approches peuvent se baser sur les systèmes de transitions (SDL et les Réseaux de Petri). Nous allons voir ci-dessous quelques approches basées sur état transition.

2.3.1 SDL (Specification and Description Langage)

Définition
SDL est un standard de l'ITU (International Telecommunication Union). Son objectif est de décrire de manière non ambiguë les protocoles de télécommunications afin d'assurer l'interopérabilité des équipements. Il est utilisé par l'ETSI (European Telecommunications Standards Institute) pour décrire les protocoles de télécommunications. Il a aussi une représentation graphique(34). SDL peut être utilisé pour produire (35) :

Spécification et conception d'applications diverses : automobile contrôle, électronique, systèmes médicaux.

Normes et conception des télécommunications exemple : Traitement des appels et des connexions.

2.3.1.1 Avantage de SDL

SDL offre plusieurs avantage prisent ci au-dessous (34) :

- Les modèles SDL sont indépendants de l'implémentation.
- Le SDL intègre des types de données abstraits.
- Orienté temps réel événementiel.

2.3.2 RDP (Réseau de Petri)

Définition
RDP est un modèle mathématique permettant la représentation de systèmes distribues discrets (informatique, industriel), introduit par Petri (1962) et aussi également un langage de modélisation, représente sous forme d'un graphe biparti oriente (36).

2.3.2.1 Avantage de RDP

RDP offre plusieurs avantage prisent ci au-dessous(37) :

- Ils décrivent graphiquement le système modélisé et permettent donc une visualisation facile des systèmes complexes.
- Les réseaux de petri peuvent modéliser un système hiérarchiquement (les systèmes peuvent être représentés mode descendant à différents niveaux d'abstraction et de détail).
- Les techniques d'analyse de réseau de petri bien développées fournissent une analyse systématique et complète analyse qualitative du système.
- L'existence de schémas bien formulés pour la synthèse de réseaux de petri facilite la conception et la synthèse, et les réseaux de petri chronométrés peuvent évaluer les performances du système.
- Modèle de calcul parallèle.

2.3.2.2 Inconvénient de RDP

Par contre RDP ont des inconvénient présenter ci-dessous (37) :

- La simultanéité des opérations est devenue de plus en plus courante. Cela a l'utilisation et le débit généralement améliorés, mais augmente par conséquent la complexité.
- Les sous-classes de réseaux de petri augmentent le pouvoir de décision mais ne permettent pas de modéliser un grand nombre de systèmes. Les modèles de réseaux de petri étendus augmentent le pouvoir de modélisation, mais dans tous les cas connus aux dépens du pouvoir de décision, car la plupart des questions d'analyse deviennent indécidables.
- Les modèles de réseaux de petri ont des limitations en ce qui concerne leur incapacité à rechercher exactement un marquage spécifique dans un lieu non délimité et à prendre des mesures concernant les résultats du test.

2.4 Approches algébriques

Plusieurs langages ont été proposés dans cette catégorie : (CCS, ACP, LOTOS, CSP). Nous allons voir ci-dessous quelques approches basées sur l'algèbre.

2.4.1 CCS (Calculus Communicating Systems)

Définition

CCS est un langage concurrent, ou plus spécifiquement une algèbre de processus, où les

processus sont des éléments de première classe du langage.

Le comportement d'un processus est représenté par toutes les transitions possibles du processus (38).

2.4.1.1 Avantage de CCS

CCS offre plusieurs avantage prisent ci au-dessous (2) :

- Un langage formel simple pour décrire ces systèmes en termes de leurs structures.
- Une théorie sémantique qui cherche à comprendre le comportement des systèmes décrits dans le langage, en termes de leur capacité à interagir avec l'environnement.

2.4.1.2 Inconvénient de CCS

CCS offre plusieurs inconvénient prisent ci au-dessous :

- Ne peut décrire qu'un nombre très limité de systèmes (2).
- L'exécution en parallèle est modélisée par interleaving arbitraire (39).
  En représenté la modélisation interleaving par la figure 2.1.

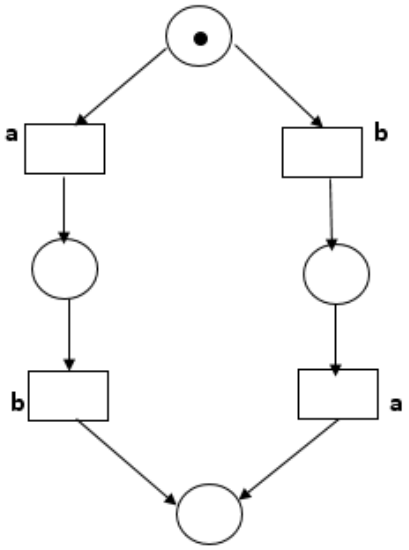

FIGURE 2.1 – Modélisation enterleaving

2.4.2 LOTOS (Language Of Temporal Ordering Specification)

Définition
LOTOS est utilisé pour la spécification de protocole Open Systems Interconnection (OSI). LOTOS est un langage algébrique composé de deux parties : une partie pour la description des données et des opérations, sur la base de types de données abstraits, et une partie pour

la description de processus simultanés, basés sur le calcul du processus (40). LOTOS basé sur CCS.

2.4.2.1 Avantage de LOTOS

LOTOS offre plusieurs avantage prisent ci au-dessous (41) :

- Il permet de donner une bonne idée du comportement du système surtout au niveau le plus externe.
- Il possède des opérateurs et des types abstraits assez complets pour exprimer les fonctions du système.

Les tableaux 2.1, 2.2 illustré les approches formelle et sont domaine d'utilisation.

Approches formelles	**Domaine d'utilisation**
Langage Z	-Langage Z utilisé pour spécifier le contenu multimédia dans les systèmes critiques de sécurité.
VDM	-
Méthode B	-Méthode B utilisé pour les systèmes distribué.
CCS	-CCS utilisé pour les systèmes concurrents (42).

TABLE 2.1 – Comparaison entre les approches formelles

Approches formelles	**Domaine d'utilisation**
LOTOS	LOTOS utilisé pour les systèmes distribuer (43).
SDL	-Spécification et conception d'applications diverses -Normes et conception des télécommunications exemple.
RDP	-RDP utiliser pour les systèmes dynamique à événement discret (44).

TABLE 2.2 – Comparaison entre les approches formelles

2.5 Travaux connexes

Il y a plusieurs travaux portant sur la spécification et la vérification formelle des systèmes distribués concurrents. Néanmoins, nous nous sommes basés dans notre travail principalement sur l'étude de deux travaux :

1)Méthode formelle pour spécifier un accident : Ce travail a spécifié en utilisant LOTOS un système qui permet de traiter les accidents. Il permet alors, d'étudier l'existence d'un accident et vérifie deux propriétés.

la premier propriété de sureté (safty) exprime que le système n'envoie pas le SAMU ni la police lorsque l'accident a déjà été rapporté. La deuxième propriété de vivacité (livenese) exprime que le système envoie toujours le SAMU sur le lieu de l'accident lorsqu'un rapport d'accident est fait et qu'il ne s'agit pas d'un doublon (45). Ce travail nous a permis de comprendre l'utilisation de LOTOS pour la spécification de ce type de système et la manière de vérifier des propriétés de sureté et de vivacité. Notre travail, par contre, porte sur une autre application qui est l'agence de voyage.

2)Méthode formelle pour spécifier emergency response workflow : Ce travail a spécifié en utilisant LOTOS un système qui permet de traiter d'urgence. Il permet alors, d'étudier la réponse d'urgence et vérifie deux propriétés.

La premier propriété sureté (safty) exprime que le système n'envoie jamais les ambulanciers, ni la police si l'urgence était déjà rapporté, la deuxième propriété vivacité (livenese) exprime que le système a chaque fois qu'un l'accident est signalé, le processus envoie toujours les ambulanciers à moins que l'urgence n'ait déjà été signalée (46). Ce travail nous a permis de comprendre l'utilisation de LOTOS pour la spécification de ce type de système et la manière de vérifier des propriétés de sureté et de vivacité. Notre travail, par contre, porte sur une autre application qui est l'agence de voyage.

Notre travail consistant à spécifier et à vérifier un système basé SOA en utilisant **l'algèbre de processus**, nous allons donc nous penché sur ce formalisme et l'étudier en plus de détails, dans la section suivante, et particulièrement le langage **LOTOS**.

2.6 Algèbre de Processus

L'algèbres de processus sont des familles de langages formels permettant de modéliser les systèmes logiciels, en particulier les systèmes distribués et concurrentiels. Elles constituent un outil pour une description du plus haut niveau d'interaction, de communication et de synchronisation entre les processus. Elles définissent des règles algébriques permettant d'une part, l'analyse et la manipulation des descriptions des processus, et d'autre part, le raison-

nement formel des équivalences entre les processus. Un tel raisonnement est très utile pour déterminer par exemple si un service peut remplacer un autre dans la composition, ou de vérifier simplement la redondance des services (47).

A cet égard, de nom-breux langages d'algèbres de processus ont vu le jour, les plus populaires entre eux sont :

CSP : Communicating Sequential processes est une algèbre de processus permettant de modéliser l'interaction de systèmes. Il permet de fournir un mécanisme unifié pour la communication et la synchronisation des systèmes, basé sur la notion de message et ne présupposant pas de l'existence d'une mémoire commune entre les processus (48).

CSP a été initialement présenté par Hoare en 1978 afin de décrire un langage de programmation concurrente, puis a subi différentes améliorations lui procurant une syntaxe plus mathématique influencée par CCS (49).

CCS : Calculus Communicating Systeme a été introduit par Robin Milner comme paradigme de langage concurrent.

Il diffère, par exemple, des réseaux de petri en ce qu'il s'utilise comme un langage de programmation dont les primitives seraient réduites aux simples envois et réceptions sur des canaux. On peut donc lire un terme CCS comme un code abstrait, correspondant à un véritable programme et l'exécuter suivant des règles bien définies (50) et utile pour évaluer le bien-fondé qualitative des propriétés d'un système comme une impasse **Deadlock** ou boucles infinies **Livelock** (48).

Deadlock :Si aucun des composants (processus) ne peut faire d'action, parce qu'il est en train d'attendre pour communiquer avec d'autres (3).

Livelock : Si un réseau communique indéfiniment de manière interne sans que ses composants communiquent avec l'extérieur (3).

LOTOS : Language Of Temporal Ordering Specification est un standardisée par ISO (Organisation internationale de normalisation) (51), utilisé pour spécifier les protocoles de télécommunication, utilisable pour les systèmes distribués en général.

Le concept sous-jacent à LOTOS est que tout système peut être spécifié en exprimant les interactions constituant le comportement observable des composantes du système (52).

Permet de décrire des ensembles de processus qui interagissent et échangent des données entre eux et avec leur environnement. (53).

La vérifiant ces processus utilisant un outil de vérification comme CADP (51). Les spécifications LOTOS se composent de deux composants (48) :

- Un composant de contrôle basé sur le calcul de systèmes de communication de Milner (CCS) et les processus séquentiels de communication de Hoare (CSP), qui traite de la description des comportements de processus et les interactions.
- Un composant de données basé sur la théorie formelle des types abstraits algébriques ACT-ONE, qui décrit les structures de données et les expressions de valeur.

2.7 Comparaison entre CCS et LOTOS

Le CCS a été étendu pour supporter les abstrait des donnés et la vrai concurrence, ce qui a fait naitre le LOTOS.

Il existe plusieurs similitudes et différences entre ces deux langages. Une comparaison est présentée dans la table 2.3.

CCS	**LOTOS**
actions(a,a1,...)	actions or gates(g,g1,...)
agents(A,A1,...)	behaviour expressions(B,B1,...)
basic calculus	basic LOTOS
value-passing calculus	full LOTOS
sort((A))	events or actions
ports(p,p1,...)	gates(g,g1,...)
labels	gates
value sets	sorts(t,t1,...)

TABLE 2.3 – Comparaison entre CCS et LOTOS

2.8 Caractéristiques principales d'algèbre de processus

Les caractéristiques principales d'algèbre de processus sont (54) :

- Spécification et étude des systèmes concurrents (communication, synchronisation).
- Abstraction sur les comportements.
- Mode de composition (parallèle, entrelacement, etc).
- Modèles sémantiques (opérationnelle, traces, bissimulation, failure-divergence).

Le langage de l'algèbre de processus que nous avons utilisé pour faire la spécification est **LOTOS**, car il permet la description des services (55), il utilise le formalisme des types abstraits algébriques (56), il fournit le nécessaire niveau d'expressivité pour décrire toutes les formes d'interaction (57) et la vrai concurrence qui est illustrée à la figure 2.2.

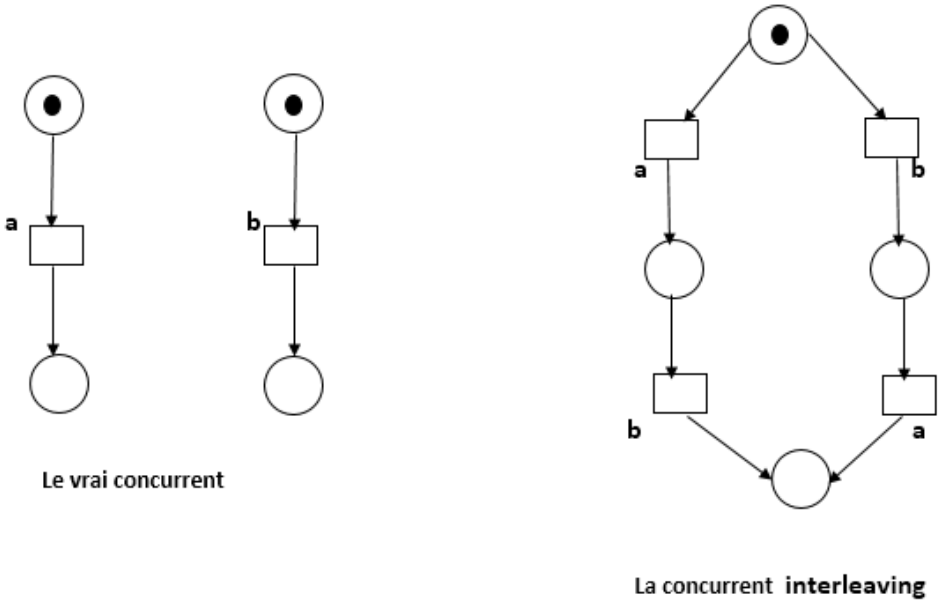

FIGURE 2.2 – Modélisation la vrais concurrent et concurrent interleaving

2.9 Langage Of Temporal Ordering Specification

LOTOS est une technique de description formelle, promue au rang de norme ISO en 1988 (2).

LOTOS est un langage permettant de spécifier l'architecture et le fonctionnement de systèmes distribues et a été défini pour permettre la description des services et des protocoles de télécommunications, en particulier pour les systèmes OSI (55).

LOTOS permet de définir et de manipuler des structures de données, mais à la différence des langages classiques, LOTOS utilise le formalisme des types abstraits algébriques (abstract data types, ADT), en s'inspirant largement du langage ActOne (56).

LOTOS s'appuie sur le langage CCS qui est étendu par un mécanisme de synchronisation multiple héritant du langage CSP pour la spécification de la partie comportementale, la partie description des structures de données est inspirée de Act-One (2).

LOTOS un système distribué est représenté par un ensemble de processus communiquant (58) qui interagissent et échange des données entre eux avec leur environnement (3).

Le processus LOTOS représente abstraitement par la figure 2.3.

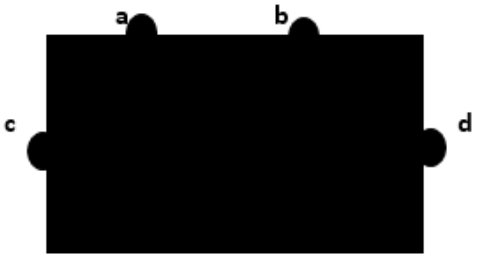

FIGURE 2.3 – Format de processus LOTOS abstraitement

Cette figure représente le processus LOTOS abstraitement, chaque composant représente (58) :

Carré : La boite noire.

les petite cercle : Les ports.

a,b,c,d : Les événements qui représente une synchronisation entre processus (ou avec son environnement).

A chaque instant, un processus peut (3) :

- Exécuter une action interne $\imath$.
- Ne plus rien faire : comportement inactif stop (deadlock).
- Ne plus rien faire : terminaison en succès exit.
- Se synchroniser sur une porte avec un autre processus ou avec l'environnement.

Remarque
Si aucune synchronisation n'est possible, il y'a interblocage (deadlock) et si le processus ne propose plus de synchronisation, il y a perte de vivacité (livelock) (3).

2.9.1 Syntaxe formelle de LOTOS

Les spécifications LOTOS se composent de deux composants :

2.9.1.1 Basic-LOTOS

En LOTOS, le contrôle des programmes est décrit par des expressions algébriques appelées comportements (3). En utilise CCS, CSP pour décrire le comportement(processus) des processus sans les données (59).

Soit P l'ensemble des identifiants de processus donc que (2) :

- X∈P.
- G l'ensemble des portes définissables (les actions observables en Basic-LOTOS).
- Soient g,g1...gn∈G.
- L un sous-ensemble (pouvant être vide) quelconque de G noté L = g1...gn.
- $\imath$ l'action interne.

Action(δ) : Cette porte qui est utilisée pour notifier la terminaison avec succès d'un processus donnée, ne peut jamais être employée explicitement dans une expression LOTOS mais elle est utilisée dans la définition sémantique du langage (2).

La structure formelle du Basic-LOTOS est donnée par la figure 2.4.

$$\textbf{Processus } X[g_1, ..., g_n] := P \textbf{ endproc}$$

$$\begin{array}{lcl} P & ::= & \textbf{stop} \mid \textbf{exit} \mid X[L] \mid i;P \mid g;P \\ & \mid & P[]P \mid P|[L]|P \mid \textbf{hide } L \textbf{ in } P \\ & \mid & P >> P \mid P[> P \end{array}$$

FIGURE 2.4 – Syntaxe de Basic-LOTOS (2).

Nous donnons les opérateurs complète des processus (P) (52) (60) (58) :

- stop $\Longrightarrow$ représente processus inactif(deadlock)et n'interagissant pas avec son environnement.
- exit $\Longrightarrow$ terminaison en succès et se transforme en stop.
- $\imath$;P $\Longrightarrow$ action préfixe (séquence) non observable, réalise l'action $\imath$ puis se transforme en P.
- a ;P $\Longrightarrow$ action préfixe (séquence) observable réalise l'action a puis se transforme en P.
- P1[]P2 $\Longrightarrow$représente le choix non déterministe qui se transforme en P1 ou en P2 suivant l'environnement.
- P1|[g1,...,gn]|P2 $\Longrightarrow$ composition parallèle avec synchronisation sur les port g1,...,gn et δ (cas générale).
- P1|||P2 $\Longrightarrow$ P1 et P2 s'exécutent en parallèle sans se synchroniser sauf sur δ(asynchrone).

- P1||P2 $\Longrightarrow$ P1 et P2 s'exécutent en parallèle et se synchronisent sur chaque porte visible.
- hide g1, ...gn in P $\Longrightarrow$ Les actions g1, ..., gn sont cachées à l'environnement de P et deviennent des actions internes.
- P1»P2 $\Longrightarrow$ P2 est activé dès que P1 se termine.
- P1[> P2 $\Longrightarrow$P2 peut interrompre P1 tant que P1 ne s'est pas terminé.

2.9.1.2 Full-LOTOS

Full -LOTOS augmentation de Basic LOTOS avec la possibilité d'échanger des valeurs lors des synchronisations entre processus(3) et en full LOTOS une action n'est pas simplement une porte de synchronisation, mais une porte plus des données échangées lors de la synchronisation (48).

Action

Action est composée d'une port, optionnelle-ment d'un prédicat et liste d'événement peut (3) :

- !$\Longrightarrow$offrir une valeur.
- ?$\Longrightarrow$accepter une valeur.

Act-One

Formalise la spécification de types abstraits de données il définit les propriétés essentielles des données et les opérations qu'une implémentation correcte du type doit assurer.

Les valeurs en Full-LOTOS peuvent être (58) :

- Echangées entre processus lors d'une synchronisation.
- Employées dans les prédicats de garde des processus.
- Utilisées comme paramètres pour la définition des processus et pour l'istanciation de valeurs.
- Associées à l'opérateur de choix généralisé.
- Exportées lors d'une terminaison avec succès d'un processus.

P1	P2	Condition de synchronisation	Type d'interaction	Résultat
g ! v1	g !v2	valeur (v1) =valeur (v2)	Concordance des valeurs	Synchronisation
g !v	g ?x : T	V∈T	Passage de valeur	Après Synchronisation x=valeur(v)
g ?x : T	g ?y : U	T=U	Choix aléatoire d'une valeur	Après Synchronisation x=y=v avec v∈T(ou v∈U)

FIGURE 2.5 – Règles de synchronisation en LOTOS avec données

La figure 2.5 illustre les nouvelles règles en prenant l'exemple de deux processus P1 et P2 composés par une synchronisation sur la porte g (la fonction valeur renvoie la valeur d'une variable) (61).

Les processus se synchronisent sur deux actions si (58) :

- Ils utilisent le même identificateur de porte.
- Ils ont des listes de données en correspondance de même type.
- Les contraintes sur les données sont satisfaites.

La règle veut qu'après une synchronisation, les actions impliquées dans cette synchronisation doivent offrir des variables de même types et de même valeurs (2). La déclaration de processus est étendue avec l'usage de paramètres. Par exemple (2) :

Processus X[g1, ...gn](x1 :T1,xm :Tm) :**Fonctionnalite** :=P

endproc
Définit, pour le processus X, m variables x1...xm : T1...Tm qui le paramètrent. Ainsi, ce processus devra être instancié avec une liste de valeurs v1...vm affectées à ces paramètres X[g1, ...gn](v1...vm) (2).

Fonctionnalite

- noexit $\Longrightarrow$ Comportement qui ne s'arrête pas (3).
- exit $\Longrightarrow$ Comportement qui s'arrête (3).

2.9.1.2.1 Abstract Data Types

Types de données sont cadre formel (mathématique) pour la définition des types, leur ensemble de données, et leurs opérations (62), permet à l'utilisateur de spécifier ses propres types de données et les opérations correspondantes sur eux (63).

Types de données prendre trois partie sont :

sorts : permettant la déclaration des ensembles de valeurs (62).

opns : permettant la déclaration des opérations manipulant les objets des sorts déclarées (62).

eqns : permettant la définition des opérations par un ensemble d'axiomes (62), permettent de prouver l'équivalence de deux termes qui appartenir à la même sorte. Les équations ne définissent pas de nouveaux termes. Ils font notre la vie plus facile en nous aidant à utiliser des termes simples qui sont équivalents à d'autres plus les complexes (63).

Type complet

Exemple sur un type complet représente dans la figure 2.6 :

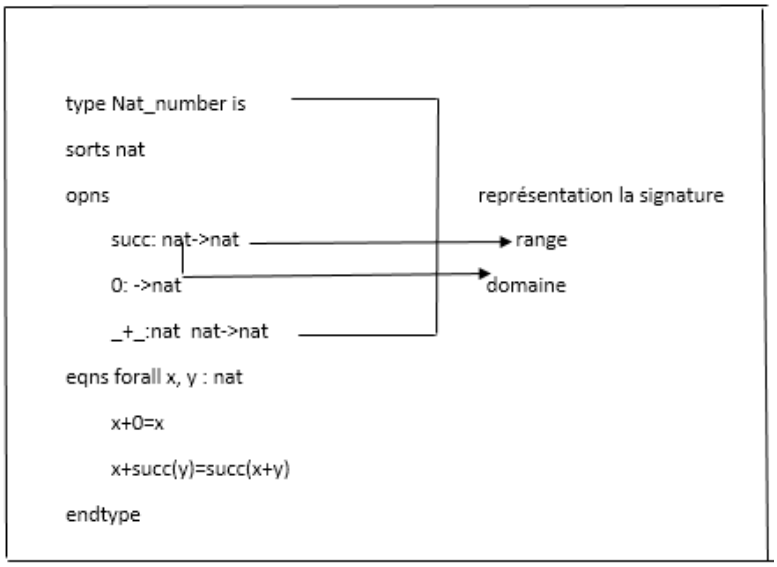

FIGURE 2.6 – Type complet

Héritage de type

Exemple sur héritage de type illustré dans la figure 2.7 :

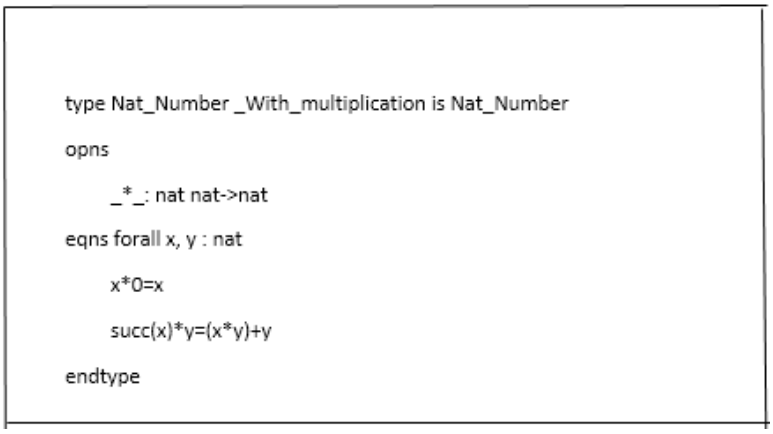

FIGURE 2.7 – Héritage de type.

2.9.2 Sémantique

La sémantique opérationnelle des processus LOTOS est donnée par un système de transitions étiquetées (LTS pour labelled transition system) utilisés par Cadp, dénoté par LTS=(S , A, T, s0) tel-que (61) :

S : est un ensemble d'états non vide qui est clairement identifié par une expression de comportement.

A : est un ensemble d'actions qu'est de la forme gv1...vn où g est un nom de porte et les vi sont des valeurs d'une sorte . Nous définissons nom(gv1...vn)=g.

s0 : s0∈S est l'état initial de Sys.

T : est un ensemble de relations de transition Ta ⊆S*S , un pour chaque a ∈A ; Ta est un ensemble de transitions de la forme : cur $\xrightarrow{a}$ next, où cur, next∈S.

Elle sera définit par des règles illustré dans la figure 2.8, 2.9, 2.10.

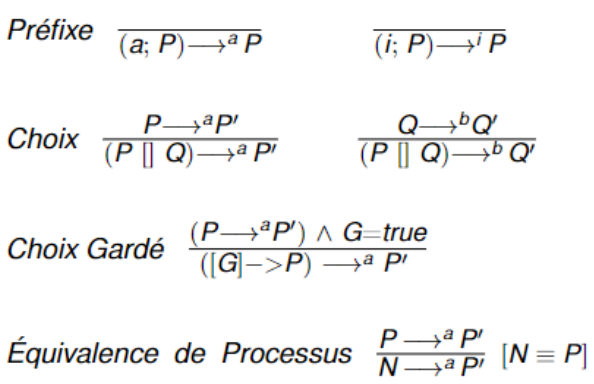

FIGURE 2.8 – Sémantique de LOTOS (3)

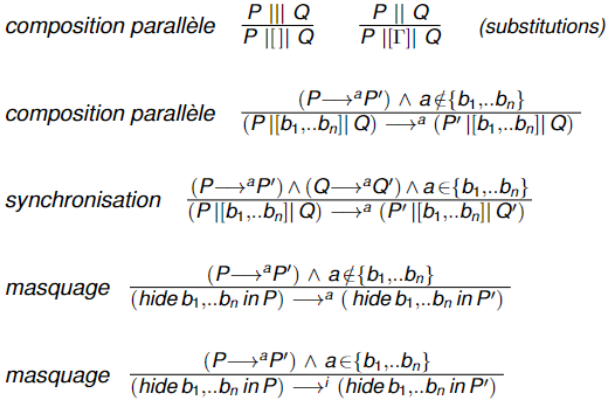

FIGURE 2.9 – Sémantique de LOTOS (3)

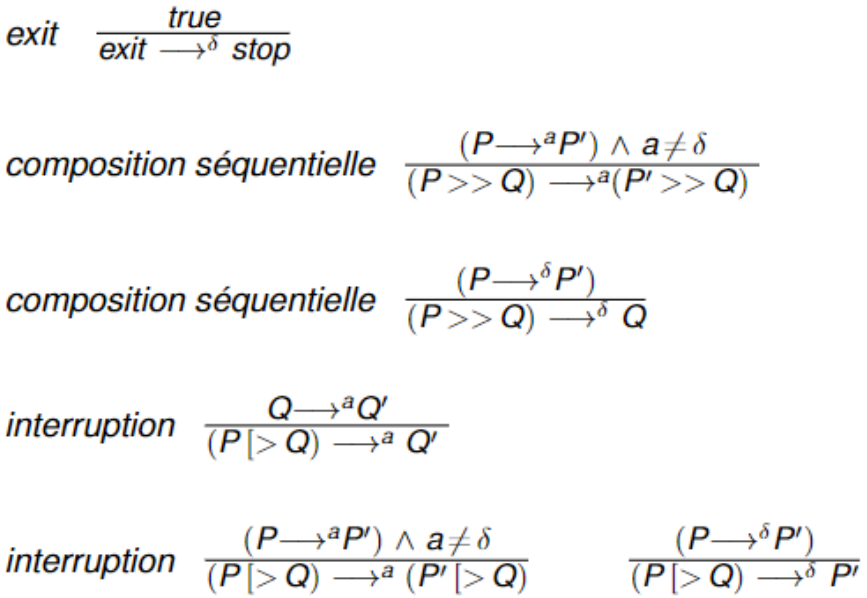

FIGURE 2.10 – Sémantique de LOTOS (3).

La sémantique de LOTOS pas de règle pour STOP (pas de transition) (3).

Nous avons présenté quelque lois algébrique pour LOTOS (3) :

- P|||EXIT≡P.
- P||Q≡Q||P.
- P[]STOP≡P.
- (P[]Q)[]R≡P[](Q[]R).
- (a ;P)[> Q≡(a ;(P[> Q))[]Q.

Exemple spécifie avec LOTOS
process DB3 [piece, choixcafe, cafe] : exit :=piece ; choixcafe ; cafe ; exit[]choixcafe ; piece ; cafe ; exit endproc.

Nous avons transforme cette exemple en utilisant LTS qui est illustré dans la figure 2.11.

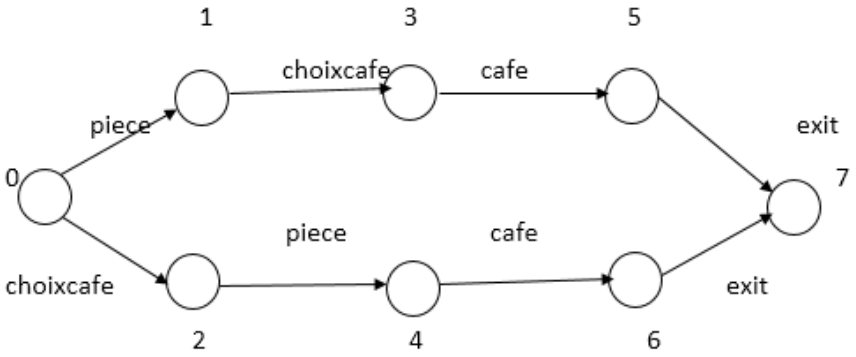

FIGURE 2.11 – Exemple du distributeur de boissons

2.10 Outil de spécification et vérification pour LOTOS

2.10.1 Construction and Analysis of Distributed Processes

CADP est une boîte à outils pour la conception de protocoles de communication et de systèmes distribués (64) asynchrones (65). Ces systèmes comportent plusieurs entités (processus ou agents) qui s'exécutent en parallèle, communiquent et se synchronisent par échange de messages (65), elle a été développée par l'équipe VASY1 de l'Inria Rhones–Alpes (66).

CADP permet la compilation, la vérification et la validation de programmes LOTOS. Elle contient de nombreux outils sous forme de modules (indépendants) de compilation ou de vérification (64).

CADP facilite la conception de systèmes fiables en utilisant des techniques de description formelles ainsi que des outils logiciels pour la simulation, le développement rapide d'applications, la vérification et la génération de tests.

2.10.2 Principe d'usage de CADP

Les module CADP sont divisés (66) :

1)Modules de base

CÆSAR.adt : traduit la partie données d'une description Lotos en langage C (les sortes et les opérations Lotos étant respectivement traduites vers des types et des fonctions C) (65).

CÆSAR : traduit la partie contrôle vers un programme C qui peut être embarqué dans un système réel ou utilisé à des fins de simulation, vérification ou génération de tests (65). Ce module admet plusieurs options (telles que -aldebaran, -bcg, -open, etc).

2)Modules spécifiques

ALDEBARAN : principal module de vérification ; il compare deux graphes, celui du modèle et celui de la propriété ou d'un programme, selon des relations indiquées (bissimulation forte, équivalence observationnelle, etc).

EXECUTOR : permet une exécution aléatoire.

GENERATOR, REDUCTOR : permettent de construire le graphe des états accessibles.

TERMINATOR : recherche les états de blocage.

BCG-MIN : minimise le graphe par les techniques de bissimulation.

3)Modules utilitaires
Tous ces modules prenent en entrée le fichier .bcg (67).

BCG-DRAW : produit une représentation graphique (2D) en postscript du graphe construit après la compilation d'une spécification.

BCG-EDIT : un éditeur interactif du graphe issu de la compilation d'une spécification.

BCG-INFO : produit des informations sur le graphe.

BCG-IO : transforme le .bcg en d'autres formats spécifiés comme options (...).

BCG-LABELS : : permet de modifier les étiquettes des transitions du graphe.

BCG-LIB : générateur de librairies dynamiques pour bcg.

BCG-OPEN : permet d'établir une passerelle entre bcg et l'environnement open/caesar.

La figure 2.12 représente les principe d'usage CADP

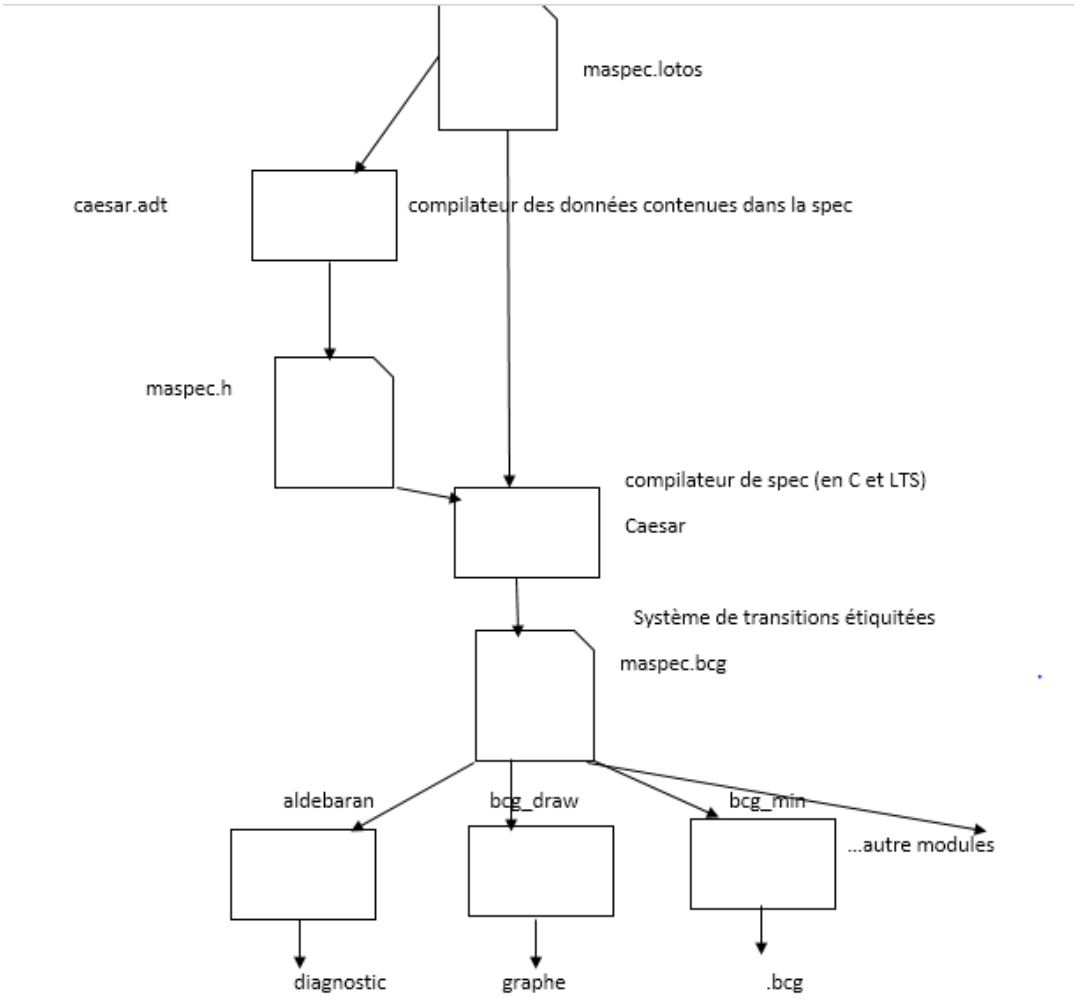

FIGURE 2.12 – Principe d'usage de CADP.

2.10.3 Propriétés de vérification

Nous voulons étudiés quelque propriété pour vérifier notre application (3) :

- **Propriété de sûreté** (safety) :quelque chose de mauvais ne se produit jamais.
- **Propriété de sûreté conditionnelle :** sous certaines conditions, certaines actions ne sont pas offertes.
- **Propriété invariante :** une condition donnée est satisfaite pour tout état atteignable.

- **Propriété de vivacité** (liveness) :quelque chose de bien finira par avoir lieu.
- **Absence de blocage** (deadlock freeness) : le système ne se trouve jamais dans une situation où il ne peut plus progresser, dans laquelle il ne peut plus faire ce qu'il doit faire.
- **Propriété de vivacité conditionnelle :** sous certaines conditions, certaines actions sont possibles.
- **Propriété d'atteignabilité** (reachability) :une situation donnée peut-être atteinte.

Conclusion

Dans ce chapitre, avons donné un aperçu général sur les classes existantes des approches formelles. Puis nous avons présenté brièvement quelques-unes de chaque classe. Nous nous sommes, ensuite, penchés sur l'algèbre des processus et présenté en plus de détail le langage LOTOS que nous allons utilisé pour notre spécification.

Par la suite, nous avons présenté l'outil de spécification et vérification «CADP», qui a été réalisé spécialement pour le langage LOTOS.

La prochaine étape de notre travail est la mise en œuvre de la spécification du système étudié, qui consiste en un système de réservation de voyage. Puis, la vérification de quelques propriétés. Cette étape sera l'objet de la Partie II de ce mémoire.

Deuxième partie

Spécification et Vérification

Chapitre 3

Spécification formelle

Chapitre 3

Spécification formelle

Introduction

La spécification formelle est importance pour le développement de systèmes logiciels et ceci tout particulièrement pour la conception et la vérification de systèmes dits sécuritaires ou critiques (par exemple ceux impliquant la vie humaine), elles fournissent un cadre pour spécifier, développer et vérifier des systèmes d'une façon systématique.

Parmi les approches formelles présentés dans le chapitre précédent nous avons opté pour l'algèbre de processus qui offre la communication entre eux, et de représenter les données échangés. Nous avons choisi, plus spécifiquement, le langage LOTOS, un langage standardisé par l'ISO.

LOTOS offre, en plus de son expressivité puissante en terme d'opérateurs, le moyen de représenter les données en utilisant les types abstraits de données (TADs).

Nous allons dans le présent chapitre utiliser le langage LOTOS pour la spécification d'un système basé SOA. Nous avons choisi pour notre cas d'étude, un exemple typique de composition de services Web qui est un système de réservation de voyage.

Nous allons commencer par l'analyse de ce système, pour identifier les services Web impliqués et leur orchestration. Puis étudier leur orchestration. Ensuite, nous présenterons la formalisation de ces différents services et de leur orchestration en utilisant le langage LOTOS.

3.1 Analyse du système

L'agence de voyage fournit un service inégalable pour le bon déroulement de votre séjour dans votre propre pays ou à l'étranger. Elle s'occupe de nombreuses fonctions. Elle sert tout d'abord de point de vente pour vendre des billets d'avion. Elle répond à la demande des

touristes et elle cible les destinations les plus prisées et les destinations les plus éloignées. Elle prend en charge de nombreux domaines notamment la vente et la réservation du billet, l'acheminement et la réservation de l'hôtel. L'agence de voyage contribue à la facilitation de l'organisation du voyage tant au niveau national qu'au niveau international. Elle garantit le déroulement du voyage des touristes pour que ces derniers puissent retrouver le confort et la totale satisfaction.

L'agence de voyage offre le service d'organisation d'un voyage (Ws-AV), qui offre une interface pour le client et a le d'invoquer et d'orchestrer les services nécessaire, cités ci-dessous : (68) :

Ws-vol : A le rôle de réserver le vol selon les informations donné (Date-Départ , Date-Retour, Ville-Départ , Ville-Arrivé , Nb-Personne), il retourne une confirmation, en cas de succès de la réservation et une infirmation (Res-not) en cas d'échec (indisponibilité de vol).

Ws-Hotel : A le rôle de réserver l'hôtel selon les informations donné (Date-Départ , Date-Retour, Ville-Arrivé , Nb-Personne), il retourne une confirmation, en cas de succès de la réservation et une infirmation (Res-not) en cas d'échec (indisponibilité de l'hôtel).

WS-Voiture : A le rôle de réserver la voiture selon les informations donné (Date-Départ , Date-Retour), il retourne une confirmation, en cas de succès de la réservation et une infirmation (Res-not) en cas d'échec (indisponibilité de la voiture).

Ws-Bank : A le rôle de la banque selon les informations donné (Liste-Prix), il retourne une confirmation, en cas de succès de la payer et une infirmation (Payement-NOT) en cas d'échec (indisponibilité de la banque).

3.1.1 Architecture du système de l'agence du voyage

A partir de l'analyse, nous pouvons schématisé l'architecture du système de l'agence de voyage comme illustré à la figure 3.1.

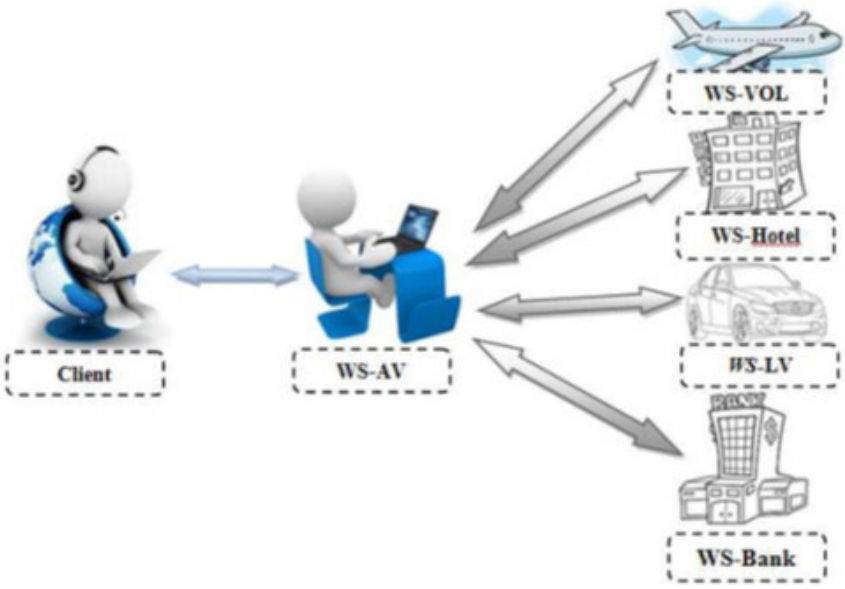

FIGURE 3.1 – Services Web offerts par l'agence de voyage (4)

3.1.2 Diagrammes de séquence

Nous avons schématisés les cas d'application avec les diagrammes de séquence dans la figure 3.2, 3.3 et 3.4 tel que : La figure 3.2 représente le cas de réservation succès par ce que le client reçois un message **Res-OK** à cause de les quatre services (vol, hôtel, voiture, et banque) ont répondu **Res-OK**.

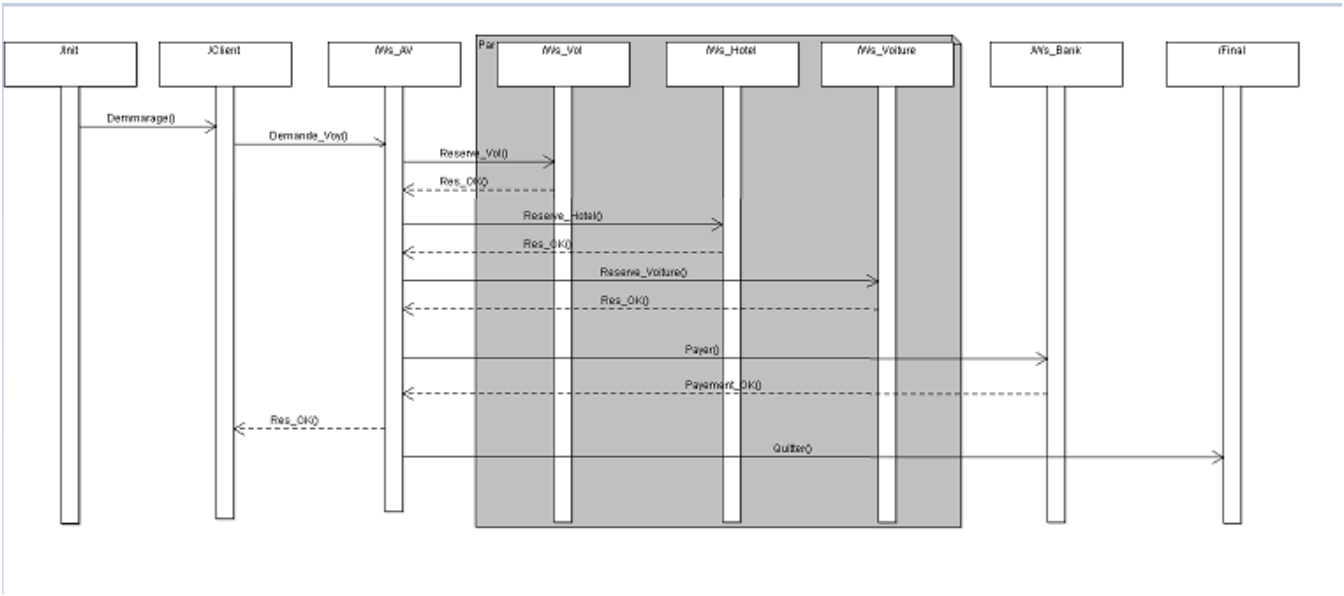

FIGURE 3.2 – Diagramme de séquence d'agence de voyage (cas de réservation-ok)

La figure 3.3 représente le cas de réservation pas succès par ce que le client reçois un message **Res-NOT** à cause de le service vol a répondu **Res-NOT** malgré les autres services ont répondu **Res-OK**.

Remarque

Ce diagramme matchs avec : minimum une seule service entre le vol, hôtel et la voiture ont répondu **Res-NOT**

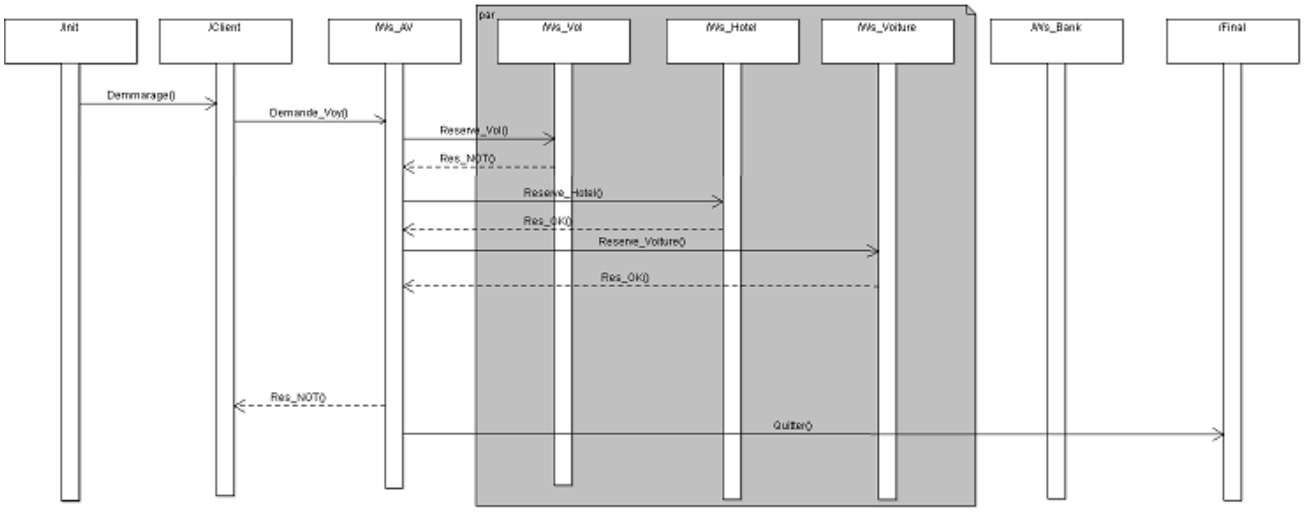

FIGURE 3.3 – Diagramme de séquence d'agence de voyage (cas de réservation-not)

La figure 3.4 représente le cas de réservation pas succès par ce que le client reçois un message **Res-NOT** à cause de le service banque a répondu **Payement-NOT** malgré les autres services ont répondu **Res-OK**.

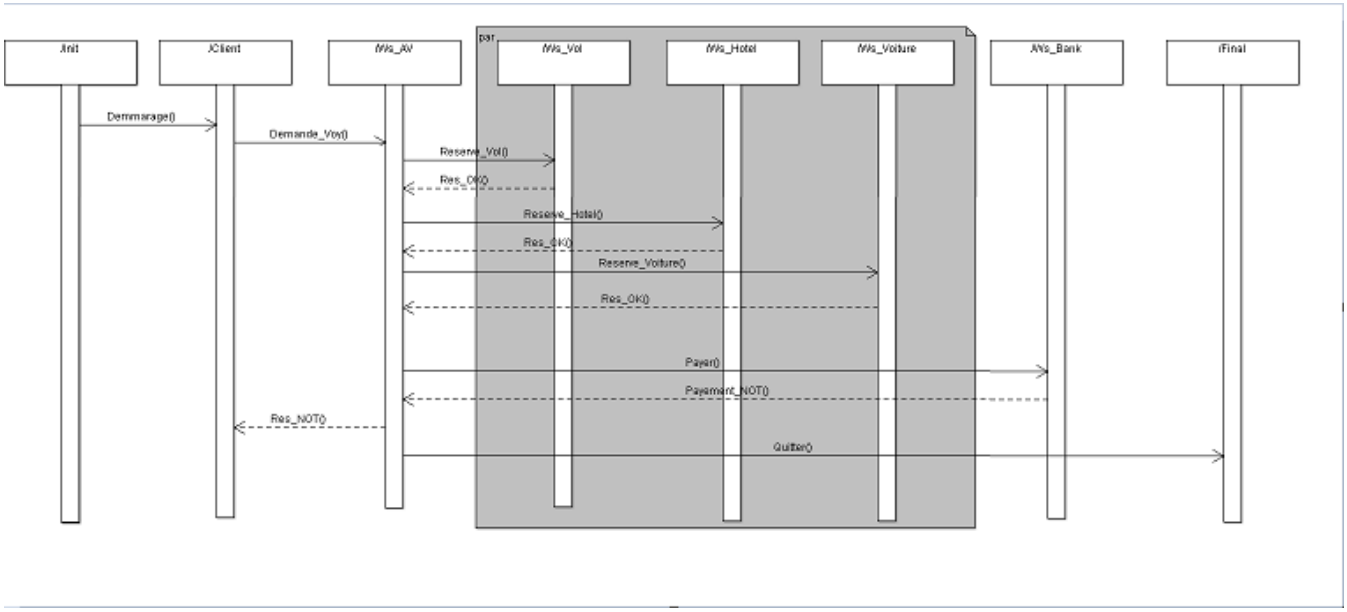

FIGURE 3.4 – Diagramme de séquence d'agence de voyage (cas de réservation-not pour la Bank)

3.1.3 Spécification de l'application avec LOTOS

En a définie un processus pour chaque étape de l'activité. Les processus sont :

- Client
- Réservation-Voyage
- Réservation-vol
- Réservation-Voiture
- Réservation-Hôtel
- Bank

Tous les processus sont exécutés de manière concurrentielle à l'aide de l'opérateur LOTOS(|||). Ceux-ci sont cependant synchronisés avec le processus BUS que représente une interface, à travers les portes SEND et RECV, à l'aide de l'opérateur |[SEND,RECV]|. Les processus peuvent ainsi communiquer entre eux à travers ces ports. Chaque processus de service se voit attribuer un identifiant qui est un entier.

L'instanciation des processus dans LOTOS abstraitement est fourni à la figure 3.5.

```
specification Agence_Voyage [SEND, RECV, reserok, resernot, payementnot, payementok]: noexit
library
type_donne
endlib
behaviour
(
Init [SEND, RECV, reserok, resernot, payementnot, payementok](0 of Int)
|||
Client [SEND, RECV, reserok, resernot, payementnot, payementok](1 of Int)
|||
Agence_Voyage [SEND, RECV, reserok, resernot, payementnot, payementok] (2 of Int)
|||
Reservation_Vol [SEND, RECV, reserok, resernot, payementnot, payementok](3 of Int)
|||
Reservation_Hotel [SEND, RECV, reserok, resernot, payementnot, payementok](4 of Int)
|||
Reservation_Voiture [SEND, RECV, reserok, resernot, payementnot, payementok](5 of Int)
|||
Bank [SEND, RECV,reserok,resernot, payementnot, payementok](6 of Int)
|||
Final [SEND, RECV, reserok, resernot, payementnot, payementok](7 of Int)
)
|[SEND,RECV]|
BUS [SEND,RECV,reserok,resernot, payementnot, payementok] (<>)
where(*Implimentation of processes*)
```

FIGURE 3.5 – Processus d'instanciation dans LOTOS

Important de l'utilisation du bus

Un bus logiciel est modélisé par LOTOS. Les services peuvent envoyer ou recevoir des messages (événements) via les portes SEND et RECV respectivement. Le processus de bus agit comme un tampon non lié initialement vide. Accepte les messages sur le porte SEND et diffuse sur la porte RECV. Chaque processus de service se voit un attribuer (identifiant) qu'est un entier. Lors de la communication via le bus les services fournir l'identifiant du service de destination (46).

Le principe de fonctionnement du bus est illustré à la figure 3.6

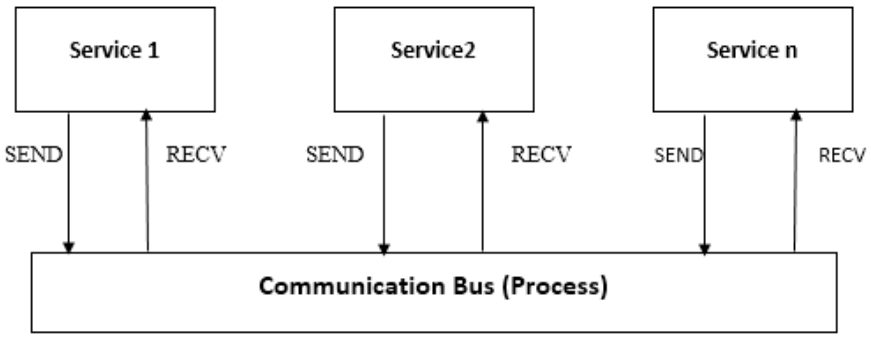

FIGURE 3.6 – Rôle de Bus

3.1.4 Implémentation des processus avec lotos

Les processus sont divisée en : les processus utiliser dans la spécification (Processus initial, Processus client, Processus agence du voyage, Processus de réservation de vol, Processus de réservation hôtel, Processus de réservation voiture, Processus de la banque, Processus final) et les processus utiliser pour les processus utiliser dans la spécification (Processus parallelsplit, processus sequence, processus sequence1, processus sequence2, processus sequence3, processus sequence4).

3.1.4.1 Processus initial

Le processus initiale (Id : 0) commence simplement le processus client (Id : 1). Comme un conséquence, il utilise le motif de séquence avant sortant, comme défini à la figure 3.7.

```
process Init [SEND, RECV,reserok,resernot, payementnot, payementok] (Id:Int) : exit :=
Sequence [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 1 of Int) >>
exit
where(*Definition of Sequence process*)
```

FIGURE 3.7 – Spécification LOTOS pour processus Init (Initiale)

3.1.4.2 Processus client

Le processus client (Id : 1) est initialement dans un état endormi, ce qui signifie qu'il attend un message de type RUN en provenance du processus initiale (Id : 0) avant se réveiller et de démarrer son exécution. Après cela, il réalise une séquence à l'agence de voyage (Id : 2) avant sortant, comme défini à la figure 3.8.

```
process Client [SEND, RECV, reserok, resernot, payementnot, payementok] (Id:Int) : exit :=
RECV !Id !0 of Int !RUN !void;
Sequence [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2 of Int) >> exit
where(*Definition of Sequence process*)
```

FIGURE 3.8 – Spécification LOTOS pour Client

3.1.4.3 Processus sequence

Le processus séquence, une activité identifiée par id-destination devrait être exécuté après l'achèvement de la activité identifiée par id-source dans le flux de travail, nous disons que les

deux activités ont exécuté de manière séquentielle (69). La spécification LOTOS est fournie à la figure 3.9.

```
process Sequence [SEND, RECV, reserok, resernot] (Id_source:Int, Id_dest:Int) : exit :=
SEND !Id_dest !Id_source !RUN !void; exit
endproc
```

FIGURE 3.9 – Spécification LOTOS pour Sequence

3.1.4.4 Processus agence du voyage

Le processus agence de voyage (Id : 2) attend un message RUN du processus client (Id : 1) avant de démarrer simultanément le réservation vol (Id : 3) et réservation hôtel (Id : 4) et réservation voiture (Id : 5). Après cela, il réalise une séquence entre la banque (Id : 6), le processus de client (Id : 1) et le processus de final (Id : 7) (cas de réservation succès) ou il réalise une séquence entre le client (Id : 1) et le processus de final (Id : 7) (cas de réservation pas succès).

La spécification correspondante est fournie à la figure 3.10.

```
process Agence_Voyage [SEND, RECV, reserok, resernot, payementnot, payementok] (Id:Int) : exit :=
RECV !Id !1 of Int !RUN !void;
ParallelSplit [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, insert(5 of Int ,
insert(4 of Int, insert(3 of Int, emptyset))
)) >>
((Sequence [SEND, RECV,reserok,resernot, payementnot, payementok] (Id, 6 of Int) >>
(Sequence1 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 1 of Int)
[]
Sequence2 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 1 of Int)) >>
Sequence [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 7 of Int))
[]
(Sequence2 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 1 of Int) >>
Sequence [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 7 of Int)))
 >>
exit
where(*Definition of Sequence process and ParallelSplit*)
```

FIGURE 3.10 – Spécification LOTOS pour Agence du voyage

3.1.4.5 Rôle du processus parallelsplit

Mécanisme pour exécuter plusieurs exécutions branches en même temps. Une seule branche diverge en deux ou plusieurs branches d'exécution parallèles. Chacun de ces branches parallèle contient des activités qui seront exécutées en même temps (46).

3.1.4.6 Processus de réservation de vol

Le processus de réservation vol (Id : 3) attend un message RUN du processus agence de voyage (Id : 2), après cela, il réalise une séquence1 (réservation succès) à l'agence de voyage (Id : 2) ou séquence2 (réservation pas succès) avant sortant.

La spécification correspondante est fournie à la figure 3.11.

```
process Reservation_Vol [SEND, RECV, reserok, resernot, payementnot, payementok] (Id:Int) : exit :
=
RECV !Id !2 of Int !RUN !void;
(Sequence1 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2 of Int)
[]
Sequence2 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2 of Int))>> exit
where(*Definition of Sequence process*)
```

FIGURE 3.11 – Spécification LOTOS pour Reservation-Vol

3.1.4.7 Processus sequence1 de reservation-ok

le conformation de reservation Le processus sequence1 est pour «la réservation-ok». La spécification LOTOS est fournie à la figure 3.12.

```
process Sequence1 [SEND, RECV,reserok,resernot] (Id_source:Int, Id_dest:Int) : exit :=
reserok !Id_dest !Id_source !RUN !void; exit
endproc
```

FIGURE 3.12 – Spécification LOTOS pour Sequence1

3.1.4.8 Processus sequence2 de reservation-not

Le processus sequence2 est pour «la réservation-not». La spécification LOTOS est fournie à la figure 3.13.

```
process Sequence2 [SEND, RECV,reserok,resernot] (Id_source:Int, Id_dest:Int) : exit :=
resernot !Id_dest !Id_source !RUN !void; exit
endproc
```

FIGURE 3.13 – Spécification LOTOS pour Sequence2

3.1.4.9 Processus sequence3 de payement-ok

Le processus sequence3 est pour «le payement-ok». La spécification LOTOS est fournie à la figure 3.14.

```
process Sequence3 [SEND, RECV,reserok,resernot,payementnot,payementok] (Id_source:Int, Id_dest:Int)
payementok !Id_dest !Id_source !RUN !void; exit
endproc
```

FIGURE 3.14 – Spécification LOTOS pour Sequence3

3.1.4.10 Processus sequence4 de payement-not

Le processus sequence4 est pour «le payement-not». La spécification LOTOS est fournie à la figure 3.15.

```
process Sequence4 [SEND, RECV,reserok,resernot,payementnot,payementok] (Id_source:Int, Id_dest:Int)
payementnot !Id_dest !Id_source !RUN !void; exit
endproc
```

FIGURE 3.15 – Spécification LOTOS pour Sequence4

3.1.4.11 Processus de réservation hôtel

Le processus de réservation hôtel (Id : 4) attend un message RUN du processus agence de voyage (Id : 2), après cela, il réalise une séquence1 (réservation succès) à l'agence de voyage (Id : 2) ou séquence2 (réservation pas succès) avant sortant.

La spécification correspondante est fournie à la figure 3.16.

```
process Reservation_Hotel [SEND, RECV, reserok, resernot, payementnot, payementok] (Id:Int) : exit
:=
RECV !Id !2 of Int !RUN !void;
(Sequence1 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2)
[]
Sequence2 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2 of Int)) >> exit
where(*Definition of Sequence process*)
```

FIGURE 3.16 – Spécification LOTOS pour Reservation-Hotel

3.1.4.12 Processus de réservation voiture

Le processus de réservation voiture (Id : 5) attend un message RUN du processus agence de voyage (Id : 2), après cela, il réalise une séquence1 (réservation succès) à l'agence de voyage (Id : 2) ou séquence2 (réservation pas succès) avant sortant.

La spécification correspondante est fournie à la figure 3.17.

```
process Reservation_Voiture [SEND, RECV, reserok, resernot, payementnot, payementok] (Id:Int) :
exit:=
RECV !Id !2 of Int !RUN !void;
(Sequence1 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2 of Int)
[]
Sequence2 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2 of Int)) >> exit
where(*Definition of Sequence process*)
```

FIGURE 3.17 – Spécification LOTOS pour Voiture

3.1.4.13 Processus de la banque

Le processus de banque (Id : 6) attend un message RUN du processus agence de voyage (Id : 2) après cela, il réalise une séquence3 (payement-OK) à l'agence de voyage (Id : 2) ou séquence4 (payement-NOT) avant sortant.

La spécification correspondante est fournie à la figure 3.18.

```
process Bank [SEND, RECV, reserok, resernot, payementnot, payementok] (Id:Int) : exit :=
RECV !Id !2 of Int !RUN !void;

(Sequence3 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2 of Int)
[]
Sequence4 [SEND, RECV, reserok, resernot, payementnot, payementok] (Id, 2 of Int))>> exit
where(*Definition of Sequence process*)
```

FIGURE 3.18 – Spécification LOTOS pour Bank

3.1.4.14 Processus final

Le processus final (Id : 7) attend un message RUN de le processus agence de voyage (Id : 2) avant de quitter.

La spécification correspondante est fournie à la figure 3.19.

```
process Final [SEND, RECV, reserok, resernot, payementnot, payementok] (Id:Int) : exit :=
RECV!Id!2 of Int !RUN !void;exit

endproc
endspec
```

FIGURE 3.19 – Spécification LOTOS pour Final

Conclusion

A travers à l'exemple dans ce chapitre, nous avons tenté d'illustrer l'utilité des méthodes de modélisation, analyser les propriétés fonctionnelles des systèmes informatiques critiques comportant du parallélisme. La langage Lotos s'avère adapté pour décrire de manière succincte et abstraite le fonctionnement des contrôleurs chargés de piloter les dispositifs physiques (7) et pour faire la validation de ce spécification, nous verrons le chapitre 4 est la mise en œuvre de la vérification avec quelque propriété a l'aide d'outil CADP.

Chapiter 4

Vérification formelle

Chapitre 4

Vérification formelle

Introduction

Nous commençons ce chapitre par une explication brève des étapes de l'installation de l'outil CADP, que nous avons utilisé pour la compilation et la vérification de notre spécification.

Nous montrerons, ensuite les résultats de la compilation et l'exécution de notre spécification dans CADP. Nous terminerons le présent chapitre par la vérification de deux propriétés, dont une de sûreté et l'autre de vivacité. Nous expliquerons l'utilisation de CADP pour la vérification de ces deux propriétés.

4.1 Installation de CADP

D'abord on commence par les étapes de l'installation de l'outil CADP. Avant utiliser l'outil CADP nous installons une machine virtuelle (Cygwin) qui permet entrer des commandes après avoir de configuration de Cygwin, après cela apparait une fenêtre pour continu l'installation de CADP, il demande le mot de passe CADP qu'est envoyé à travers email après remplir le formulaire (demande d'installation), avant se termine de l'installation il ya fenêtre pour choisir la méthode d'envoi la license de CADP illustre à la figure 4.1.

FIGURE 4.1 – Installation du CADP

Après recevoir la license de CADP, nous sauvegardons la license dans un dossier CADP, faire des commandes sur le Cygwin et faire une configuration sur le dossier CADP qui est apparait après la terminaison d'installation de CADP. Enfin, nous écrivons un commande **xeuca** pour un appel de fenêtre graphique de CADP qui permet utiliser cette outil.

4.2 Exécution de la spécification

La figure 4.2 représente le modèle cheking qui explique comment exécute la spécification qui a écrit en lange LOTOS.

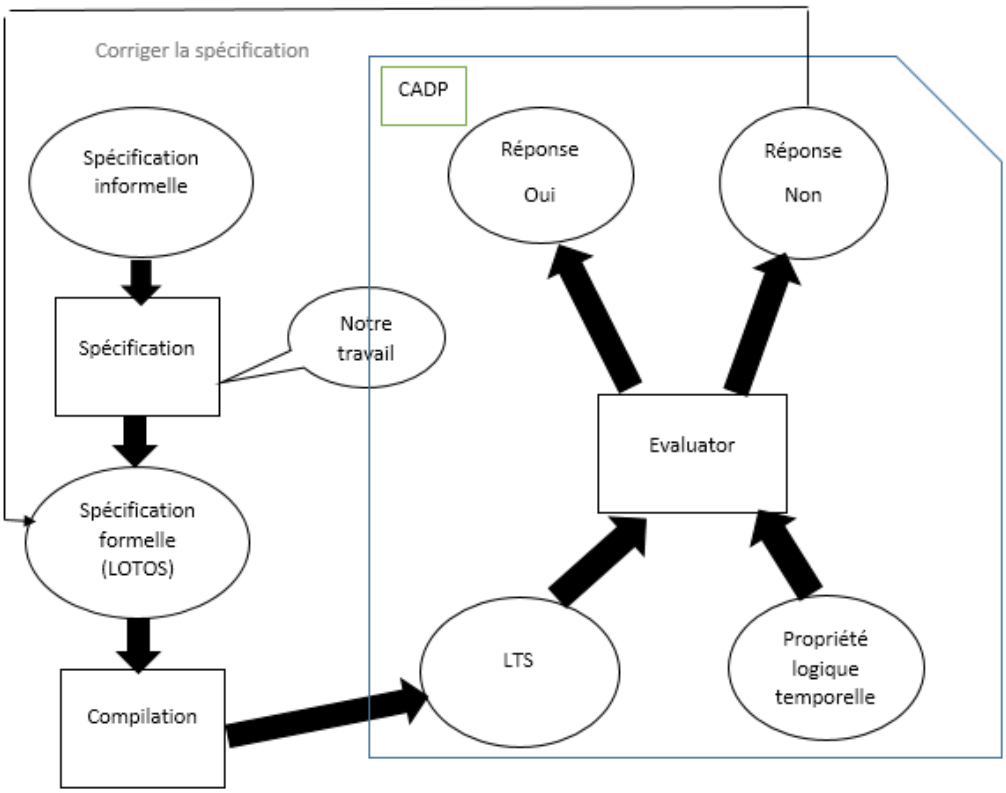

FIGURE 4.2 – Processus du Model-checking

Nous avons divisés la spécification d'agence du voyage en quatre parties qui sont illustrés à la figure 4.3, la premier partie (try.lotos) pour la spécification abstraite avec l'implémentation des processus, la deuxième partie pour définit les types des donner (type-donne.lib) la troisième partie pour définit les processus utilisent dont la spécification (PATTERN-PROC.lib), la quatrième partie pour définit le bus (BUS-PROC.lib).

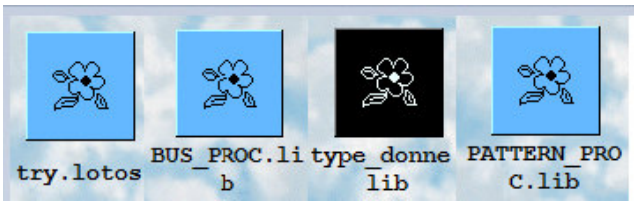

FIGURE 4.3 – Quatre fichiers de spécification

4.2.1 Arbre de spécification

La figure 4.5 et 4.6 représente l'arbre d'exécution de la spécification d'agence de voyage dans le cas si la réservation est succès, qui matchs avec le diagramme de séquence d'agence du voyage de la figure 3.2. Alors que quand exécute la spécification nous verrons la fenêtre OCIS

(simulateur) ce qui permet faire l'arbre d'exécution de cette spécification et aussi apparaît deux fichier un fichier (try.h) et un fichier (try.c) qui ont permis faire le (LTS) et faire de la vérification.

La figure 4.4 illustre les fenêtres des fichiers.

FIGURE 4.4 – Trois fichiers de spécification

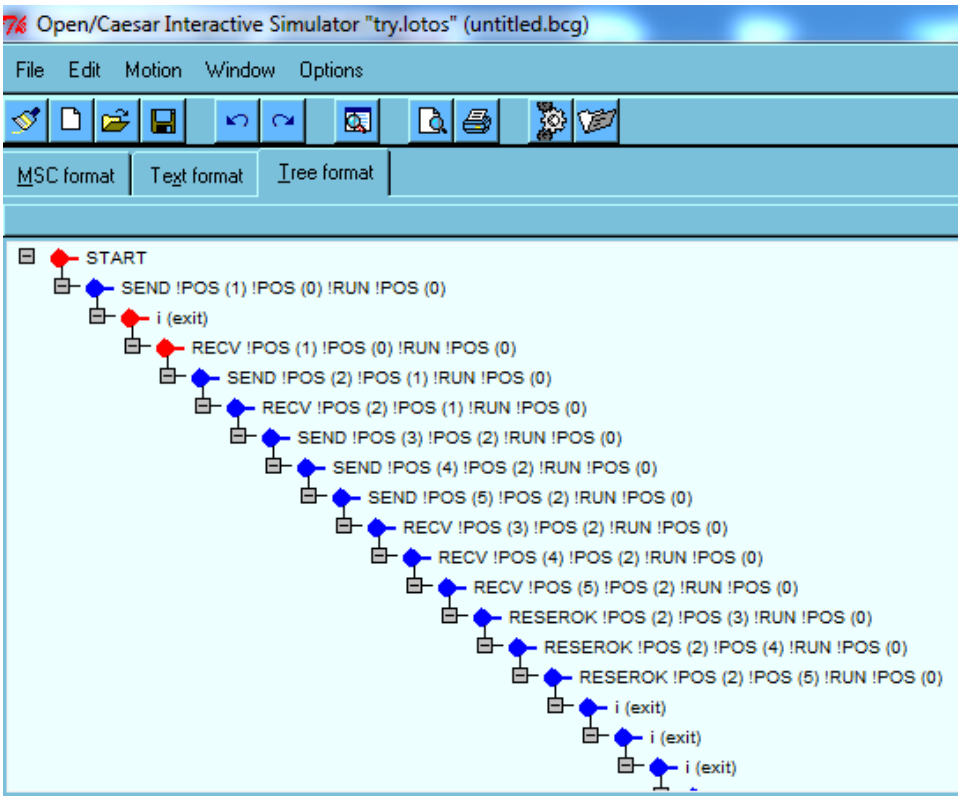

FIGURE 4.5 – Arbre d'exécution de spécification pour la réservation-ok

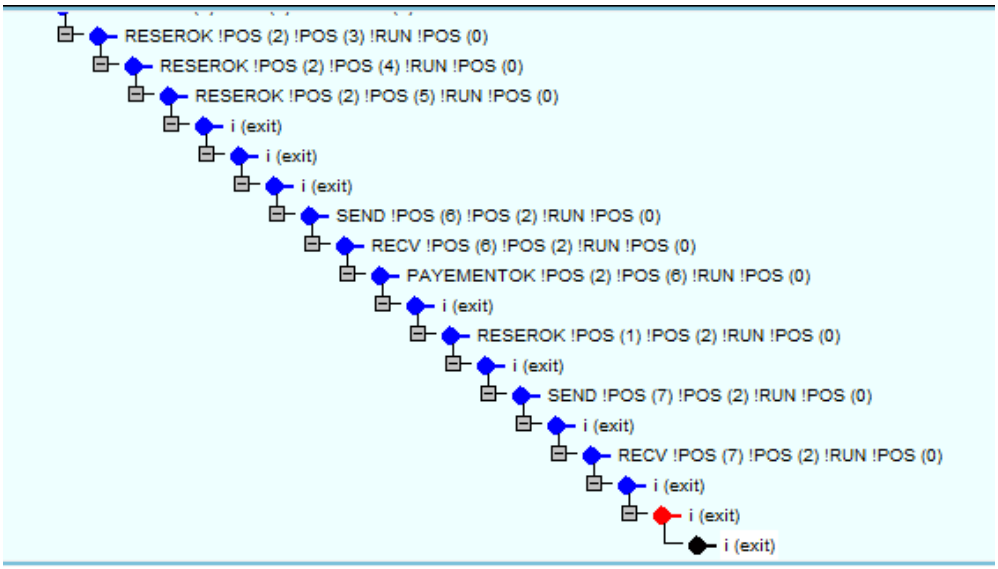

FIGURE 4.6 – Arbre d'exécution de spécification pour la réservation-ok (suite)

la figure 4.7 et 4.8 représente l'arbre d'exécution de la spécification d'agence de voyage dans le cas si la réservation pas succès, qui matchs avec le diagramme de séquence d'agence du voyage de la figure 3.3.

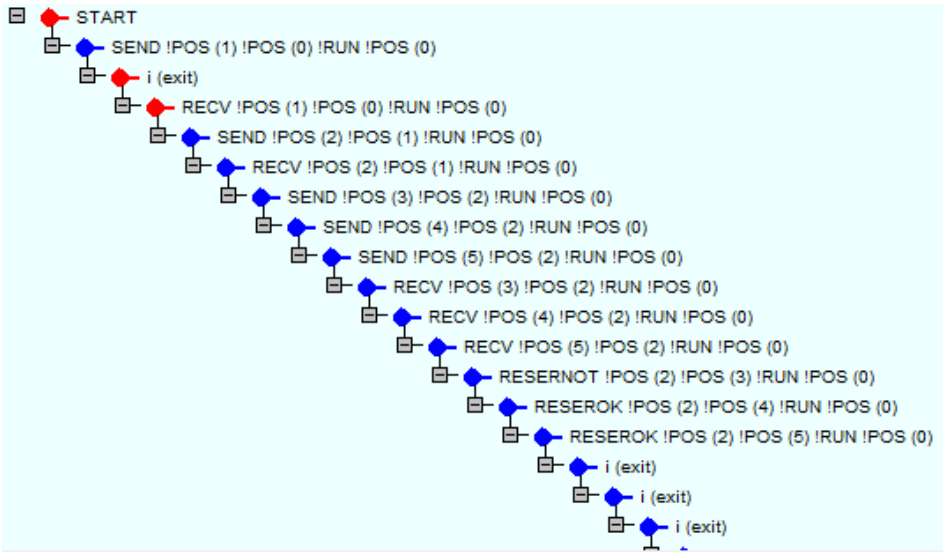

FIGURE 4.7 – Arbre d'exécution de spécification pour la réservation-not

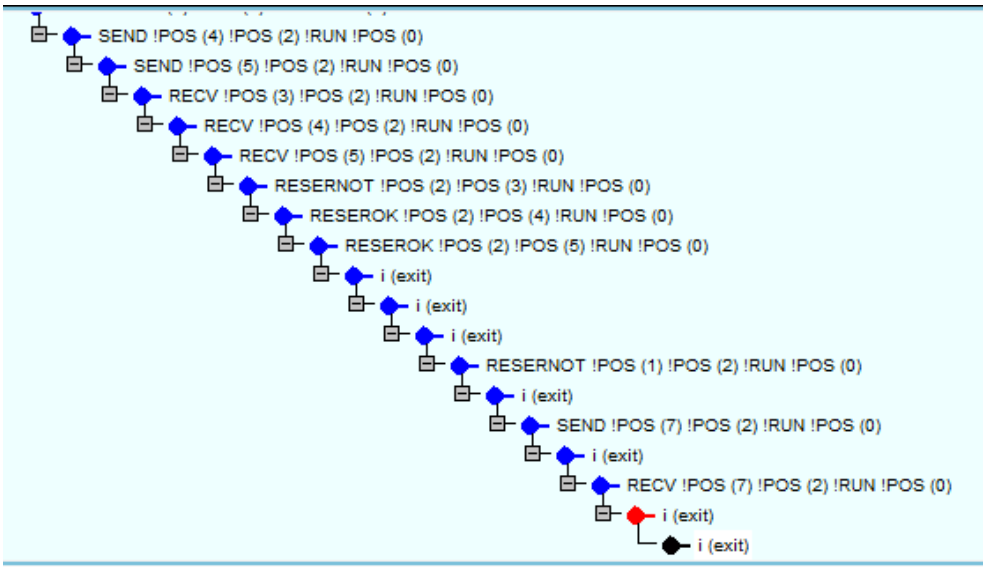

FIGURE 4.8 – Arbre d'exécution de spécification pour la réservation-not (suite)

4.2.2 Labelled Transition System

Pour générer LTS nous avons faire une clique sur icône de (try.lotos), nous avons choisi **Gnerate labelled transition system** et puis voir un fichier (try.bcg), voir la figure 4.9.

FIGURE 4.9 – Fichier bcg

Après, en faire une clique sur cette icône nous choisissons **reduce** pour minimiser LTS puis voir un autre icône voir la figure 4.10.

FIGURE 4.10 – Reduce fichier bcg

Après, en faire une clique sur cette icône nous choisissons **Visualise** qui lance le programme que nous avons installé (GSview) pour voire LTS, voir la figure 4.11.

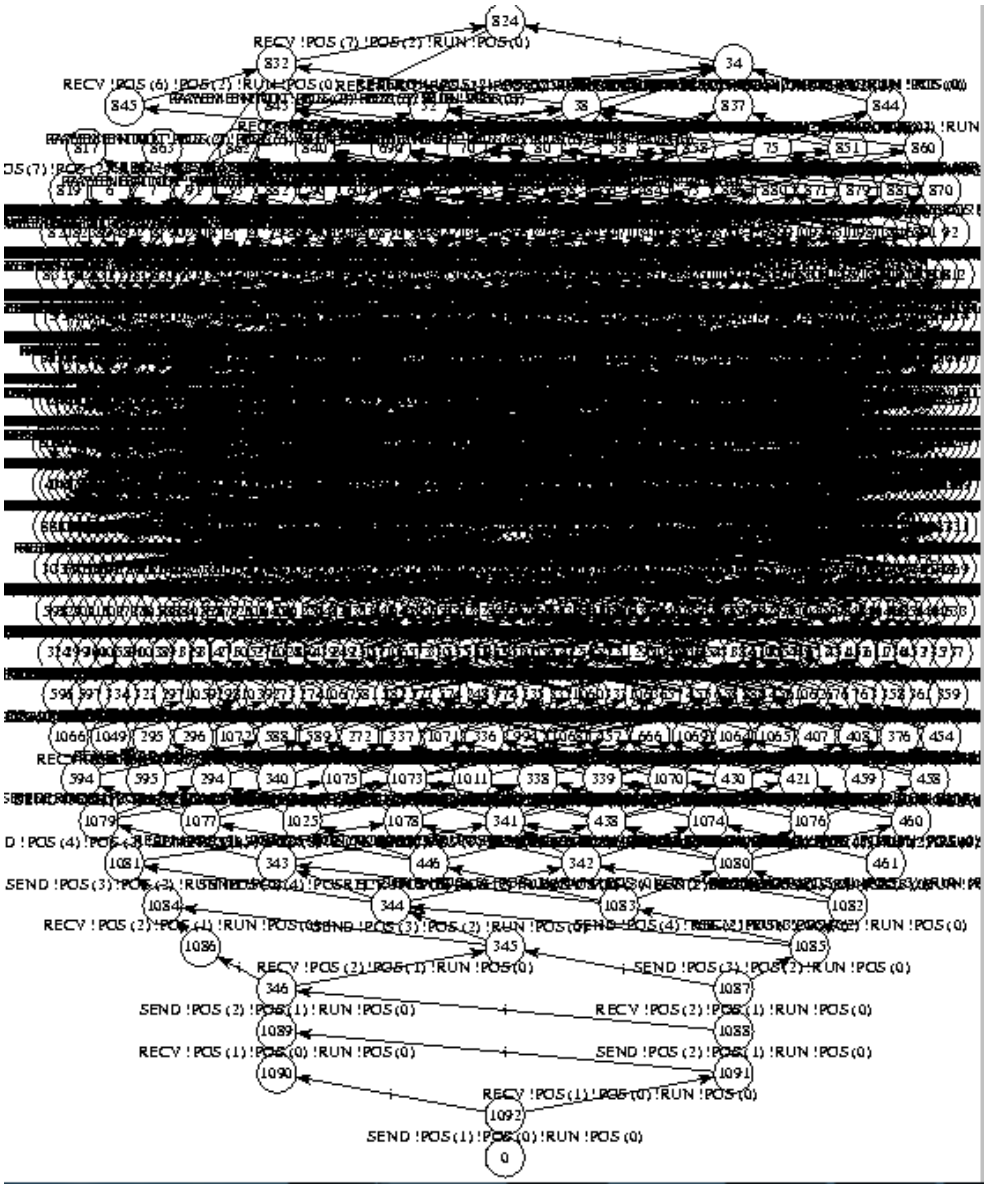

FIGURE 4.11 – LTS généré et réduit par CADP

4.3 Vérification formelle avec CADP

Nous allons maintenant utiliser la boîte CADP afin de transformer la description de la composition en LOTOS en représentation mathématique sur laquelle il sera possible de vérifier certaines propriétés (45). En effet, nous allons utiliser l'outil EVALUATOR (application de OPEN/CÆSAR) qui utilise en entrée un modèle et une propriété. La propriété à vérifier doit être décrite sous la forme d'une formule de la logique temporelle encodée en régulier alternation-free μ-calculus (7).

Pour faire la vérification nous écrivons un fichier extension.mcl qui contient les propriétés, voir la figure 4.12.

FIGURE 4.12 – Model Checking Language

Nous faisions un clique sur le fichier extension.bcg, nous trouvons une liste nous choisissons **Verify temporal formulas** et puis nous sélectionnons le fichier de propriété. Alors que faire l'ok nous verrons un fichier d'evaluator4, voir la figure 4.13.

FIGURE 4.13 – Fichier evaluator4

Nous fournissons dans la figure 4.14, 4.15 quelques propriété que nous allons utiliser dans nos spécification.

4.3.1 Propreté de safety

Un exemple de propriété qu'il serait utile de vérifier dans notre exemple est que le système n'envoie pas en parallèle (agence du voyage) à client ou final et donc a un usage abusif et inutile de ces services. Cette propriété est dite de sureté (safety) car elle assure qu'une propriété indésirable ne sera jamais vérifiée (46).

Cette propriété est exprimée en régulière alternation-free μ-calculus dans la figure 4.14.

```
[ true* . "ENTER !1" . (not "LEAVE !1")* . "ENTER !7" ] false
```

FIGURE 4.14 – Propriété de sureté

4.3.2 Propreté de atteignabilité

Ce propriété qu'il serait intéressant de vérifier sur notre exemple serait que Celles-ci sont similaires aux propriétés de vivacité, sauf qu'elles expriment l'accessibilité des actions en considérant uniquement les séquences d'exécution correctes. Une notion simple d'équité qui

peut être facilement encodée dans MCL est "l'atteinte équitable des prédicats" une séquence est juste si elle ne permet pas infiniment souvent l'atteignabilité d'un certain état sans atteindre infiniment souvent il. Par exemple, la formule suivante spécifie qu'après chaque émission de message (action SEND), toutes les séquences d'exécution correctes entraînent la réception du message (action RECV) après un nombre fini d'étapes. La propriété se traduit ici de la manière suivante (70) :

```
[ true* . SEND . (not RECV)* ] < (not RECV)* . RECV > true
```

FIGURE 4.15 – Propriété l'atteignabilité

Conclusion

Nous avons procédé à la vérification de ces propriétés temporelles sur notre description formelle de la composition avec l'outil EVALUATOR de CADP. Cet outil nous a permis de réaliser une vérification à la volée et nous a indiqué que les propriétés ont été évaluées comme étant vraies. Ceci prouve que la spécification est bien conforme vis à vis de ces comportements attendus. Comme expliqué précédemment, le service composé considéré est ici suffisamment simple pour que les propriétés vérifiées soit évidemment vraies.

Conclusion

Dans ce travail, une spécification d'un système de réservation de voyage basé sur la composition des services Web a été élaborée. Nous avons opté pour l'algèbre de processus qui est muni d'une expressivité permettant la spécification du comportement de systèmes en les disséquant en leurs composants concurrents. Ce formalisme offre le moyen de spécifier des systèmes complexes de manière précise et relativement concise. Nous avons utiliser le langage LOTOS, un langage de description formelle standardisé par l'ISO, car il offre, en plus de son expressivité puissante en terme d'opérateurs, le moyen de représenter les données en utilisant les types abstraits de données (TADs).

Nous avons également étudié l'outil de «CADP», développé spécialement par l'équipe VASY1 de l'Inria Rhones–Alpes pour le langage LOTOS. Cet outil nous a permis de compiler notre spécification et de vérifier certaines propriétés telles que : la sureté, la atteignabilité. En effet, L'étape de vérification est indispensable à toute approche moderne de développement logiciel. Il est nécessaire de vérifier et tester tout nouveau système logiciel avant sa mise en commercialisation et son utilisation. L'étape de vérification permet non seulement d'assurer la fiabilité mais contribue également à limiter les coûts puisque la découverte d'erreurs de conception après la mise en production d'un système peut engendrer des coûts importants. les outils de vérification formelle procèdent à une compilation du modèle vérifié afin d'obtenir une représentation mathématique. L'outil est alors en mesure d'explorer de manière exhaustive toutes les branches d'exécution possibles sur la représentation mathématique. Notre tâche se résume alors à définir des propriétés comportementales à l'aide de la logique temporelle et ces propriétés seront vérifiées de manière exhaustive et automatique par l'outil.

Ainsi, notre travail nous a permis d'apprendre et d'expérimenter la spécification dans le langage LOTOS, ainsi que l'utilisation de l'outil CADP pour la vérification d'un système de réservation de voyage, basé sur la composition des services Web.

Nous nous sommes intéressés dans ce travail à la composition des services Web par orchestration. Il s'agit d'une approche centralisée où les clients interagissent avec un service composé central au lieu d'interroger directement les services atomiques existants. Il serait intéressant de spécifier et vérifier dans un travail futur un système basé sur la composition de services en utilisant la chorégraphie. La chorégraphie de services est une alternative distribuée à l'orchestration qui consiste à concevoir une coordination décentralisée des applications.

Une autre perspective, serait d'étudier d'autres langage de l'algèbre des processus pour les systèmes distribués plus expressifs.

Bibliographie

[1] Y. BELAID : Service web (soap). lig-membres.imag.fr/plumejeaud/NFE107.../Service%20Web%20(SOAP).ppt.pdf.

[2] Toufik Messaoud MAAROUK : Modèles formels pour la conception des systèmes temps réel. 2012.

[3] Introduction à lotos et à l'outil cadp. https://pdfsecret.com/download/introduction-a-lotos_5a3254c7d64ab212339246a8_pdf.

[4] LIFA Siham DAHA HANANE : Une approche formelle pour la composition des services web. Mémoire de D.E.A., UNIVERSITé ECHAHID HAMMA LAKHDAR - EL-OUED, Juin 2015.

[5] Fayçal BACHTARZI : Une approche de composition des services web basée transformation de graphes. *université costantine*, 2, 2014.

[6] Wan FOKKINK : Introduction to process algebra 2nd ed, 2007.

[7] Radu MATEESCU : *Modélisation et analyse de systemes asynchrones avec CADP*. Thèse de doctorat, INRIA, 2006.

[8] Wan FOKKINK : *Modelling distributed systems*. Springer Science & Business Media, 2007.

[9] https://www.researchgate.net/publication/220442933_Une_architecture_orientee_service_web_pour_la_constitution_de_minicubes_SOLAP_pour_clients_mobiles.

[10] http://dspace.univ-biskra.dz:8080/jspui/bitstream/123456789/5238/3/chapitre%20fin2%20%281%29.pdf.

[11] Introduction à l'architecture orientée service. http://deptinfo.unice.fr/~baude/WS/cours_SOA_AO+FB.pdf, 2017.

[12] Emine ULUKÜTÜK : Soa : L'utilité organisationnelle, technique et financière de l'architecture orientée service. 2013.

[13] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo et Tony Newling : *Patterns : service-oriented architecture and web services*. IBM Corporation, International Technical Support Organization, 2004.

[14] Soh Guéhéneuc Guéhéneucet Guibault et Guibault Occello, Khomh : Modèle architectural architectures orientées services.

[15] Büschi Mathias : Mise en place d'une architecture de type soa. Mémoire de D.E.A., Janvier 2011.

[16] Aymen Baouab : *Gouvernance et supervision décentralisée des chorégraphies inter-organisationnelles*. Thèse de doctorat, Université de Lorraine, 2013.

[17] Mehdi Ben Hmida et Serge Haddad : Vers l'adaptabilité dynamique des architectures orientées services. *Actes des 3emes Journées Francophones sur le Développement de Logiciels Par Aspects (JFDLPA'07)*, page 3, 2007.

[18] Alain MILLE : Web services. Rapport technique, 2006.

[19] MOKHTARI KARIMA KHADIDJA : *ANALYSE DES PROPRIETES TEMPORELLES DE LA PRIVACITE DANS LES BUSINESSP PROTOCOLES*. Thèse de doctorat, Université d'Oran, 2012–2013.

[20] Sofiane Chemaa : *Une approche de composition de services Web à l'aide des Réseaux de Petri orientés objet*. Thèse de doctorat, ABDELHAMID MEHRI, CONSTANTINE 2, 2014.

[21] Assia Ben Shil et Mohamed Ben Ahmed : Classification, recherche et composition de services web. *In INFORSID*, pages 215–230, 2006.

[22] Emma Fki : *Sélection et composition flexible basée services abstraits pour une meilleure adaptation aux intentions des utilisateurs*. Thèse de doctorat, Univerisité Toulouse 1 Capitole, 2015.

[23] Youssef BELAID : Conservatoire nationale des arts et métiers. Rapport technique, 2008–2009.

[24] Bouchahma Karim : *Conception et développement d'un Service Web Pour l'échange d'information dans le domaine humanitaire*. Thèse de doctorat, Fribourg, Suisse.

[25] Emna FKI : *Composition des Services Web Sémantiques À base d'Algorithmes Génétiques*. Thèse de doctorat, Université Abou-bekr Belkaid Tlemcen, 2011–2012.

[26] Maha Driss : *Approche multi-perspective centrée exigences de composition de services Web*. Thèse de doctorat, Université Rennes 1, 2011.

[27] Sofiane CHEMAA. : *Une approche de composition de services Web à l'aide des Réseaux de Petri orientés objet*. Thèse de doctorat, ABDELHAMID MEHRI, CONSTANTINE 2, 2014.

[28] KHELLAF RADHIA : *Vérification de la Compatibilité des Services Web pour une Composition*. Thèse de doctorat, Université Constantine2, 2014.

[29] Guy TREMBLAY : Une introduction aux méthodes formelles. `https://www.labunix.uqam.ca/~tremblay_gu/MFs-MGL7260.pdf`, 2008.

[30] Thomas FAYOLLE : *Combinaison de méthodes formelles pour la spécification de systèmes industriels*. Thèse de doctorat, Université Paris-Est, 2017.

[31] Siti Halimah BAKRI, Hanis HARUN, Amera ALZOUBI et Rosziati IBRAHIM : The formal specification for the inventory system using z language. *In Proceedings of the 4th International Conference on Computing and Informatics, ICOCI 2013*, pages 28–30. Sintok : Universiti Utara Malaysia, 2013.

[32] Frédéric GERVAIS : La méthode b. `https://www.researchgate.net/publication/230688722`, février 2015.

[33] Retour d'expérience sur l'usage du formel dans le ferroviaire. `http://projects.laas.fr/IFSE/FMF/J1/P05_FBustany.pdf`, Nouvembre 2012.

[34] Emmanuel GAUDIN : Sdl : 20 ans de programmation basée modèle. `https://www.pragmadev.com/news/RTS-2007.pdf`, 2007.

[35] Specification description language (sdl) - part i. `http://www.ee.oulu.fi/research/tklab/courses/521265A/lectures/ch5_SDL.pdf`.

[36] Vincent AUGUSTO : Réseaux de petri. `https://www.emse.fr/~augusto/enseignement/icm/gis1/UP2-2-RdP-slides.pdf`, 2012-2013.

[37] Jean KRIVINE : *Algèbres de processus réversibles*. Thèse de doctorat, Université Pierre et Marie Curie-Paris VI, 2006.

[38] Philippe WANG : Un noyau générique pour interprète ccs. page 4, 2007.

[39] Hans HANSSON et Bengt JONSSON : A calculus for communicating systems with time and probabilities. *In [1990] Proceedings 11th Real-Time Systems Symposium*, pages 278–287. IEEE, 1990.

[40] Shubha Raj K B et Suryaprasad J : Model checkers –tools and languages for system design- a survey. `https://airccj.org/CSCP/vol6/csit66004.pdf`.

[41] T Cornilleau, V Gal et E Gressier-Soudan : Cohérences causales pour mémoire partagée répartie : Algorithmes et implantations.

[42] Jesper R Andersen, Nicklas Andersen, Søren Enevoldsen, Mathias M Hansen, Kim G Larsen, Simon R Olesen, Jirí Srba et Jacob K Wortmann : Caal : Concurrency workbench. *In International Colloquium on Theoretical Aspects of Computing*, pages 573–582. Springer, 2015.

[43] Colin Fidge : A comparative introduction to csp, ccs and lotos. *Software Verification Research Centre, University of Queensland, Tech. Rep*, pages 93–24, 1994.

[44] SAADI Djallel : *Synthèse des Superviseurs : Approche basée sur la Théorie des Réseaux de Petri*. Thèse de doctorat, Université Ahmed Ben Bella d'Oran1 Es Senia, 2011.

[45] Christophe Dumez : *Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en oeuvre de services Web composés*. Thèse de doctorat, Université de Technologie de Belfort-Montbeliard, 2010.

[46] Christophe Dumez, Mohamed Bakhouya, Jaafar Gaber et Maxime Wack : Formal specification and verification of service composition using lotos. *In Proceedings of the 7th ACM international conference on pervasive services*, 2010.

[47] Wafaa Ait-Cheik-Bihi : *Approche orientée modèles pour la vérification et l'évaluation de performances de l'interopérabilité et l'interaction des services*. Thèse de doctorat, Université de Technologie de Belfort-Montbeliard, 2012.

[48] Nicholaos C. Petalidis : Translating lotos speci
cations into promela. Mémoire de D.E.A., Heriot-Watt University, October 1994.

[49] Raphaël Chane-Yack-Fa : *VéRIFICATION FORMELLE DE SYSTéMES D'INFORMATION*. Thèse de doctorat, UNIVERSITé DE SHERBROOKE, 2018.

[50] Jean Krivine : *Algèbres de processus réversibles*. Thèse de doctorat, Université Pierre et Marie Curie-Paris VI, 2006.

[51] D. Chenouni3 N. Adadi1, M. Berrada2 : Formal specification of web services composition using lotos. Rapport technique.

[52] Ahmed Chawki Chaouche : *Une approche multi-agent pour la conception de systèmes d'intelligence ambiante : Un modèle formel intégrant planification et apprentissage*. Thèse de doctorat, Université Pierre et Marie Curie ; Université Constantine 2-Abdelhamid Mehri, 2015.

[53] Les langages de specification – generalites sur le langage lotos. `https://www.lri.fr/~wolff/teach-material/2009-10/M2Pro-GL/part-II.pdf`.

[54] Mohammed Charaf Eddine MEFTAH : *Une approche formelle pour les applications web 2.0*. Thèse de doctorat, Université Mohamed Khider-Biskra, 2016.

[55] Les algèbres de processus. https://elearn2013.univ-ouargla.dz/courses/MFP/document/Cours/Les_algebres_de_Processus_-1-.pdf?cidReq=MFP.

[56] Hubert GARAVEL : Presentation du langage lotos1. 1993.

[57] Gwen SALAÜN et Tevfik BULTAN : Realizability of choreographies using process algebra encodings. *In International Conference on Integrated Formal Methods*, pages 167–182. Springer, 2009.

[58] Chapitre 5.le langage lotos. http://webcache.googleusercontent.com/search?q=cache:jeguAYiPhhsJ:deptinfo.unice.fr/twiki/pub/Minfo/IngenierieProtocoles/IP_N72TR_5_lotos.ppt+&cd=2&hl=fr&ct=clnk&gl=dz.

[59] Christian ATTIOGBÉ : Introduction aux algèbres de processus et lotos. http://pagesperso.ls2n.fr/~attiogbe-c/mespages/MSFORMEL/ProcessusComm/slides_introAP_2pp.pdf, 1999–2000.

[60] Radu Mateescu—Pascal Poizat—Gwen SALAÜN : Behavioral adaptation of component compositions based on process algebra encodings. 2007.

[61] Chapter 4 : Theoretical basis of lotos. http://www.montefiore.ulg.ac.be/~leduc/cours/ILR/ch4.ps.

[62] Le langage lotos. https://www.powershow.com/viewfl/29f44e-ZDc1Z/Chapitre_5_Le_Langage_LOTOS_powerpoint_ppt_presentation.

[63] Nicholaos C. PETALIDIS : *Translating LOTOS Speci
cations into PROMELA*. Thèse de doctorat, Master's thesis, Heriot-Watt University, 1994.

[64] http://pagesperso.ls2n.fr/~attiogbe-c/mespages/MSFORMEL/ProcessusComm/presentation_cadp.pdf.

[65] Radu MATEESCU : *Modélisation et analyse de systemes asynchrones avec CADP*. Thèse de doctorat, INRIA, 2006.

[66] Jacob Tchoumtchoua BENOIT FRAIKIN : Présentation de la boite à outils cadp :application au protocole pots. Rapport technique, avril 2001.

[67] Frédéric LANG et Wendelin SERWE : Cadp tutorial.

[68] CHOUIREF ZAHIRA : *Un système de programmation fonctionnelle pour la composition de services Web*. Thèse de doctorat, Université de Boumerderdes.

[69] N Adadi, M Berrada et D Chenouni : Formal specification of web services composition using lotos. *International Journal of Computer Technology Applications*, 7, 2016.

[70] `http://cadp.inria.fr/man/mcl4.html`.

Résumé

De nos jours, les services web sont devenus très utilisés notamment par les entreprises pour rendre accessibles leurs métiers et leurs données via le web. La composition des services web est un sujet qui suscite l'intérêt des chercheurs. Elle offre la possibilité de traiter des problèmes complexes même avec des services simples existants tout en coopérant entre eux. Toutefois, cette tâche reste très complexe et nécessite, pour assurer sa fiabilité, des techniques formelles. L'objectif principal de notre travail est de spécifier formellement une composition de services web, illustrée dans le cas d'une agence de voyage. Il existe plusieurs outils formels de spécification, dont l'algèbre de processus qui offre un moyen à la fois puissant et relativement succinct pour spécifier des systèmes complexes. Nous avons choisi le langage LOTOS qui permet, outre la spécification des processus, la représentation des données. Nous avons utilisé l'outil CADP pour la vérification de notre spécification.

Abstract

Nowadays, web services have become very popular especially by businesses to make their trades and their data accessible via the web. The composition of services web is a topic that arouses the interest of researchers. It offers the possibility to treat complex problems even with existing simple services while cooperating between them. However, this task remains very complex and requires, to ensure its reliability, formal techniques. The main objective of our work is to formally specify a composition of web services, illustrated in the case of a travel agency. There are several formal specification tools, including process algebra that provides a powerful and relatively succinct way to specify complex systems. We have chosen the LOTOS language which allows, besides the specification of the processes, the representation of the data. We used the CADP tool to verify our specification.